OpenMS – A framework for computational mass spectrometry

Dissertation

der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

Dipl.-Inform. Marc Sturm

aus Saarbrücken

Tübingen
2010

Tag der mündlichen Qualifikation:	07.07.2010
Dekan:	Prof. Dr. Oliver Kohlbacher
1. Berichterstatter:	Prof. Dr. Oliver Kohlbacher
2. Berichterstatter:	Prof. Dr. Knut Reinert

Acknowledgments

I am tremendously thankful to Oliver Kohlbacher, who aroused my interest in the field of computational proteomics and gave me the opportunity to write this thesis. Our discussions where always fruitful—no matter if scientific or technical. Furthermore, he provided an enjoyable working environment for me and all the other staff of the working group.

OpenMS would not have been possible without the joint effort of many people. My thanks go to all core developers and students who contributed to OpenMS and suffered from the pedantic testing rules. I especially thank Eva Lange, Andreas Bertsch, Chris Bielow and Clemens Gröpl for the tight cooperation and nice evenings together.

Of course, I'm especially grateful to my parents and family for their support throughout my whole life. Finally, I thank Bettina for her patience and understanding while I wrote this thesis.

Abstract

Mass spectrometry coupled to liquid chromatography (LC-MS) is an analytical technique becoming increasingly popular in biomedical research. Especially in high-throughput proteomics and metabolomics mass spectrometry is widely used because it provides both qualitative and quantitative information about analytes. The standard protocol is that complex analyte mixtures are first separated in liquid chromatography and then analyzed using mass spectrometry. Finally, computational tools extract all relevant information from the large amounts of data produced. This thesis aims at improving computational analysis of LC-MS data—we present two novel computational methods and a software framework for the development of LC-MS data analysis tools.

In the first part of this thesis we present a quantitation algorithm for peptide signals in isotope-resolved LC-MS data. Exact quantitation of all peptide signals (so-called peptide features) is an essential step in most LC-MS data analysis pipelines. Our algorithm detects and quantifies peptide features in centroided peak maps using a multi-phase approach: First, putative feature centroid peaks, so-called *seeds*, are determined based on signal properties that are typical for peptide features. In the second phase, the *seeds* are extended to feature regions, which are compared to a theoretical feature model in the third phase. Features that show a high correlation between measured data and the theoretical model are added to a *feature candidate list*. In a last phase, contradicting *feature candidates* are detected and contradictions are resolved. In a comparative study, we show that our algorithm outperforms several state-of-the-art algorithms, especially on complex datasets with many overlapping peaks.

The second part of this thesis introduces a novel machine learning approach for modeling chromatographic retention of DNA in ion-pair reverse-phase liquid chromatography. The retention time of DNA is of interest for many biological applications, e.g., for quality control of DNA synthesis and DNA amplification. Most existing models use only the base composition to model chromatographic retention of DNA. Our model complements the base composition with secondary structure information to improve the prediction performance. A second difference to previous models is the use of a support vector regression model instead of simple linear or logarithmic models. In a thorough evaluation, we show that these changes significantly improve the prediction performance, especially at temperatures below 60°C. As a by-product, our approach allows the creation of a temperature-independent model, which can predict DNA retention times not only for a fixed temperature, but for all temperatures within the temperature range of the training data.

Finally, we present *OpenMS – a framework for computational mass spectrometry*. OpenMS provides data structures and algorithms for the rapid development of mass spectrometry data analysis software. Rapid software prototyping is especially important in this area of research because both instrumentation and experimental procedures are quickly evolving. Thus, new analysis tools have to be developed frequently. OpenMS facilitates software development for mass spectrometry by providing a rich functionality ranging from support for many file formats, over customizable data structures and data visualization, to sophisticated algorithms for all major data analysis steps. The peptide feature quantitation algorithm presented in the first part of this thesis is one of many algorithms provided by OpenMS.
We demonstrate the benefits of using OpenMS by the development of *TOPP – The OpenMS Proteomics Pipeline*. TOPP is a collection of command line tools which each perform one atomic data analysis step—typically one of the OpenMS data analysis algo-

rithms. The individual TOPP tools are used as building blocks for customized analysis pipelines. This kind of flexibility and a graphical user interface for the visual creation of analysis pipelines make TOPP a versatile instrument for LC-MS data analysis.

Kurzzusammenfassung

Die Kopplung von Massenspektrometrie (MS) und Flüssigchromatographie (LC) gewinnt immer mehr Bedeutung als analytische Technik in der biomedizinischen Forschung. Vor allem in der Hochdurchsatzproteomik und -metabolomik ist Massenspektrometrie weit verbreitet, weil sie sowohl qualitative als auch quantitative Information über Analyten liefert. Komplexe Stoffmischungen werden normalerweise mit Flüssigchromatographie aufgetrennt bevor sie mittels Massenspektrometrie analysiert werden. Danach werden alle relevanten Informationen mithilfe spezieller Computerprogramme extrahiert, da die produzierten Datenmengen sehr groß sind. Diese Arbeit hat das Ziel die computergestützte Analyse von LC-MS Daten zu verbessern. Wir stellen zwei neue Methoden zur Datenanalyse und eine Softwarebibliothek zur Entwicklung von Analyseprogrammen vor.

Der erste Teil dieser Arbeit beschäftigt sich mit einem Algorithmus zur Quantifizierung von Peptidsignalen (sogenannten Peptid-Features) in LC-MS Daten. Die exakte Quantifizierung aller Peptid-Features ist ein wichtiger Verarbeitungsschritt der meisten LC-MS Analyse-Pipelines. Unser Algorithmus detektiert und quantifiziert Peptide-Features in Peakdaten durch ein mehrstufiges Verfahren: Zuerst werden potenzielle Signalmittelpunkte gesucht anhand der für Peptidsignale typischen Eigenschaften. In einem zweiten Schritt werden die gefundenen Mittelpunkte zu Signalregionen vergrößert, die im dritten Schritt mit einem theoretischen zweidimensionalen Modell verglichen werden. Signalregionen die eine hohe Übereinstimmung zwischen Messdaten und Modell aufweisen werden dann in eine Kandidatenliste von potenziellen Peptid-Features eingetragen. Im letzten Schritt werden Widersprüche in der Kandidatenliste gesucht und diese behoben. In einer Vergleichsstudie auf komplexen Daten mit vielen überlappenden Signalen konnten wir zeigen, dass unser Algorithmus mehreren modernen Algorithmen überlegen ist.

Im zweiten Teil der Arbeit stellen wir einen neues maschinelles Lernverfahren zur Vorhersage von DNA Retentionszeiten in der Umkehrphasen-Chromatographie vor. Die Retentionszeit von DNA ist für viele biologische Anwendungen von Interesse, zum Beispiel für die Qualitätskontrolle der DNA-Synthese und der DNA-Amplifikation. Die meisten existierenden Verfahren benutzen nur die Basenzusammensetzung der DNA um die Retentionszeit zu modellieren. Unser Modell beruht auch auf der Basenzusammensetzung, bezieht aber Sekundärstrukturinformation ein, um die Vorhersageleistung zu verbessern. Ein weiterer Unterschied zu bisherigen Methoden ist die Verwendung von Support Vector Regression anstelle einfacher linearer und logarithmischer Modelle. In einer Vergleichsstudie zeigen wir, dass diese Neuerungen die Vorhersageleistung signifikant erhöhen, vor allem bei Temperaturen unter 60°C. Außerdem erlaubt unsere Methode die Erstellung temperaturunabhängiger Modelle, die Retentionszeiten nicht nur für eine feste Temperatur, sondern für den gesamten von den Trainingsdaten abgedeckt Temperaturbereich vorhersagen können.

Schließlich stellen wir OpenMS, eine Bibliothek zur Entwicklung von Software für die Massenspektrometrie, vor. OpenMS bietet alle erforderliche Datenstrukturen und viele Algorithmen zur schnellen Entwicklung von Analysesoftware. Die schnelle Entwicklung von Softwareprototypen ist gerade in der Massenspektrometrie besonders wichtig, da sowohl die Instrumente als auch die experimentellen Protokolle sehr schnell weiter-

entwickelt werden. Daher müssen regelmäßig neue Softwarelösungen zur Analyse der Daten entwickelt werden. OpenMS stellt eine umfangreiche Infrastruktur zur Verfügung und vereinfacht so die Entwicklung dieser Analysesoftware. Die Funktionalität von OpenMS reicht von der Unterstützung für weit verbreitete Dateiformate, über anpassbare Datenstrukturen and Datenvisualisierung, bis hin zu modernen Analysealgorithmen für alle Hauptanalyseschritte.
Die Vorteile die sich aus der Benutzung von OpenMS ergeben zeigen wir anhand der Entwicklung von *TOPP – The OpenMS Proteomics Pipeline*. TOPP ist eine Sammlung von Kommandozeilenprogrammen, die je einen minimalen Analyseschritt ausführen. Diese Schritte entsprechen meistens einem Algorithmus von OpenMS. Die einzelnen TOPP-Anwendungen können als ein Baukastensystem benutzt werden um daraus komplexe Analyse-Pipelines zu entwickeln. Die hieraus resultierende Flexibilität kombiniert mit einer graphischen Oberfläche zur Erstellung individueller Analyse-Pipelines, machen TOPP zu einem vielfältigen Werkzeug zur Analyse von LC-MS Daten.

Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. Proteomics . . . 1
- 1.3. Mass spectrometry . . . 2
- 1.4. Software tools for mass spectrometry . . . 2
- 1.5. Thesis overview . . . 3

2. Background **5**
- 2.1. Mass spectrometry-based proteomics . . . 5
  - 2.1.1. Mass spectrometry . . . 5
  - 2.1.2. Interpretation of mass spectrometry data and terms . . . 10
  - 2.1.3. Peptide/protein identification with mass spectrometry . . . 12
  - 2.1.4. Liquid chromatography . . . 14
  - 2.1.5. Sample preparation . . . 16
  - 2.1.6. Stable-isotope labeling . . . 16
- 2.2. Machine learning . . . 18
  - 2.2.1. Support vector classification . . . 18
  - 2.2.2. Support vector regression . . . 21
  - 2.2.3. SVM model generation and performance evaluation . . . 22
- 2.3. Software engineering . . . 25
  - 2.3.1. Software development processes . . . 25
  - 2.3.2. Requirements engineering . . . 28
  - 2.3.3. Analysis and design . . . 29
  - 2.3.4. Implementation . . . 32
  - 2.3.5. Testing . . . 34
  - 2.3.6. Deployment and maintenance . . . 35

3. A novel feature detection algorithm for centroided data **37**
- 3.1. State of the art . . . 37
  - 3.1.1. Seeding . . . 38
  - 3.1.2. Extension . . . 38
  - 3.1.3. Model fitting . . . 39
- 3.2. Our contribution . . . 39
- 3.3. Design and implementation . . . 39
  - 3.3.1. Overall design . . . 40
  - 3.3.2. Seeding phase . . . 41
  - 3.3.3. Extension phase . . . 44
  - 3.3.4. Model fitting phase . . . 45
  - 3.3.5. Feature clipping phase . . . 47
  - 3.3.6. Conflict resolution . . . 48
- 3.4. Results and discussion . . . 50
  - 3.4.1. Test datasets . . . 50

3.4.2. Parameter selection . . . 51
3.4.3. Performance on the test data . . . 51
3.4.4. Comparison to other algorithms . . . 55
3.5. Summary and conclusion . . . 58
3.6. Outlook . . . 59

4. Retention time prediction **61**
4.1. Peptide retention time prediction . . . 61
4.2. DNA retention time prediction . . . 61
4.2.1. Experimental dataset . . . 62
4.2.2. Feature selection . . . 65
4.2.3. Models . . . 67
4.2.4. Results . . . 67
4.2.5. Discussion and outlook . . . 71

5. OpenMS and TOPP **73**
5.1. State of the art . . . 73
5.2. Design goals . . . 75
5.3. Overall architecture . . . 77
5.4. Foundation classes . . . 78
5.4.1. Basic data structures . . . 78
5.4.2. Basic file system classes . . . 78
5.4.3. Progress logging . . . 78
5.4.4. Factory classes . . . 79
5.4.5. Parameter handling . . . 79
5.5. Data reduction and kernel classes . . . 82
5.5.1. Data reduction . . . 82
5.5.2. Peak data . . . 83
5.5.3. Feature data . . . 84
5.5.4. Meta data . . . 85
5.6. File and database I/O . . . 87
5.7. Visualization . . . 90
5.7.1. Peak data visualization . . . 90
5.7.2. Meta data visualization . . . 92
5.7.3. Parameter visualization . . . 93
5.8. Analysis algorithms . . . 93
5.8.1. Signal processing . . . 94
5.8.2. Feature detection and quantitation . . . 95
5.8.3. Map alignment . . . 96
5.8.4. Retention time prediction . . . 97
5.9. TOPP . . . 98
5.9.1. Packages . . . 98
5.9.2. Example pipelines . . . 101
5.9.3. TOPPView . . . 103
5.9.4. TOPPAS . . . 106
5.10. Project management . . . 107
5.10.1. Version control system . . . 107
5.10.2. Coding conventions . . . 108
5.10.3. Documentation . . . 108
5.10.4. Testing . . . 109

5.10.5. Release management 111
5.11. Discussion and outlook 112

6. Conclusion and Outlook **115**

A. List of abbreviations **117**

B. Contributions **119**

C. List of Publications **121**

D. Detailed quantitation results **123**

Bibliography **123**

1. Introduction

1.1. Motivation

Deoxyribonucleic acid (DNA) has been studied ever since it was discovered by Friedrich Miescher in 1869 [1]. Nearly one hundred years later, in 1968, Holley, Khorana and Nirenberg were awarded the Nobel Prize in medicine for their work towards the interpretation of the genetic code and its function in protein synthesis. Subsequently, many single genes could be identified and assigned to proteins. In 1990, the Human Genome Project was founded to coordinate the full sequencing of the human genome. Thirteen years later, in 2003, the first sequence of the human genome was published by the Human Genome Project [2]. This sequence of the human genome became the foundation for molecular biology as we know it today. However, it soon became obvious that the study of the genome alone could not answer many of the key questions in biology.

The low number of protein-coding genes was one of the big surprises of the human genome sequence. Only 25,000 to 30,000 protein-coding genes could be identified, while a much higher number of proteins was already known. Thus, the paradigm *'one gene codes for one protein'* needed to be revised. It became obvious that many human genes code for multiple proteins through mechanisms such as *alternative splicing* and *post-translational modifications* of proteins [3]. The huge diversity of proteins is required, because proteins carry out many functions in the cell and their activity is often subject to regulation. Proteins are essential for structural, enzymatic and signaling functions. Because of the huge number of different proteins, the large-scale study of proteins and metabolites, which interact with proteins, moved into the focus of biology.

1.2. Proteomics

The term *proteomics* has been coined by Marc Wilkins in 1994 [4]. He defined it as the study of proteins, how they are modified, when and where they are expressed, how they are involved in metabolic pathways and how they interact with one another. This definition still holds, but the focus of *proteomics* has shifted towards the large-scale study of proteins. Not single proteins but entire *proteomes* are studied. A *proteome* is defined as the complement of proteins expressed in a cell at a given time point under defined conditions. In contrast to the genome, the proteome of a cell is highly dynamic. The proteome is influenced by many factors such as the cell type, the cell cycle and changes in the environment of the cell. On the one hand, this variability complicates proteomics research. On the other hand, it allows us to directly study the effects of perturbations and diseases.

Differential proteomics is the qualitative and quantitative comparison of two or more proteomes. In differential proteomics various experimental techniques are used to study differences between samples from healthy and diseased patients, to investigate changes occurring in time series experiments, or to explore the effects of perturbations to biological systems. Thus, differential proteomics is an essential tool for understanding the

molecular foundations of diseases, for the discovery of biomarkers and for the identification of potential drug targets.

In analogy to the proteome, the complement of metabolites in a biological sample is termed *metabolome*, and the large-scale study of metabolites is termed *metabolomics*. Metabolomics, just like proteomics, is becoming increasingly important because it deepens our understanding of the biochemical processes in living cells. In both areas of research, *mass spectrometry* is a key analytical technique.

1.3. Mass spectrometry

Mass spectrometry (MS) characterizes the chemical compounds in a sample by their mass. However, most biological samples are very complex and, cannot be analyzed with mass spectrometry directly. The usual procedure is to reduce sample complexity by separation techniques, and then to conduct one or several mass spectrometric analyses. Common separation techniques are high-performance liquid chromatography (LC), capillary electrophoresis and two-dimensional gel electrophoresis (e.g., 2D PAGE). For high-throughput analysis, high-performance liquid chromatography and capillary electrophoresis are preferred over gel electrophoresis, since they can be easily automated and directly coupled to mass spectrometers.

After separation, the still complex analyte mixture is analyzed in one or several mass spectrometry runs. In mass spectrometry, analyte molecules are ionized and their mass-to-charge ratio is detected. Depending on the goal of the study, different mass spectrometer types are used. New instrument types with a higher resolution and improved scan modes become available regularly. The result of a mass spectrometry analysis is a collection of mass spectra. These spectra are used to deduce the identity and quantity of a large number of analytes that were present in the sample.

The keen interest of researchers in the field of proteomics and the rapid advancement of analytical instruments opens the door to a variety of new and complex experimental setups producing large amounts of data, which have to be analyzed jointly. In recent years it has become evident that the handling and computational analysis of this data is the major bottleneck of biomedical studies in the field of proteomics or metabolomics.

1.4. Software tools for mass spectrometry

Although algorithms for mass spectrometry-based protein analysis were available since the 1960s [5], they were not widely used because the amount of data to analyze was rather small. The breakthrough for computer-aided mass spectrometry data analysis came with the invention of soft ionization techniques in the late 1980s, which increased the data volume dramatically. The first popular algorithms were peptide identification algorithms [6], which try to infer the sequence of a peptide from the corresponding mass spectrum.

In the last decade, many analysis tools have been developed which cover all facets of mass spectrometry data analysis. Several companies offer highly specialized commercial software for specific analysis steps in LC-MS data analysis, for example MassLynx [7], DeCyder [8] and Mascot [9]. Academic software tools tend to be more flexible than the commercial tools. Customized data analysis pipelines can be created using for example the Trans-Proteomic Pipeline [10], XCMS [11] or msInspect [12]. Still, data analysis and management is a major bottleneck in the field.

Today, the biggest problem for the development of analysis software lies in the fast advancement of the field. New experimental techniques are emerging rapidly driven by continuous improvements to the instrumentation. As a consequence, new analysis tools have to be developed frequently. The lack of frameworks for rapid software development in the context of mass spectrometry makes this a very difficult task.

The main goal of this thesis is to develop a framework for rapid software development in the field of proteomics and metabolomics. The framework should provide all required infrastructure, i.e., data structures for LS-MS data, support for important file formats, algorithms for common analysis steps, etc. The second focus of this thesis is to develop new data analysis methods based on this framework, and to make the algorithms available as a flexible toolbox that can be used to create customized analysis pipelines.

1.5. Thesis overview

Chapter 2 introduces the biochemical and computational background needed for this thesis. First, mass spectrometry instrumentation and the setup of typical proteomics experiments are discussed. Then, the key concepts of the machine learning techniques *support vector classification* and *support vector regression* are introduced. Finally, an overview of modern software engineering is given.

Chapter 3 presents a novel quantitation algorithm for peptide signals in LC-MS data. The design goals of the algorithm are a high quantitation performance, a good runtime and wide applicability. To ensure these goals, the algorithm works on centroided input data in which each peak of the original mass spectra is represented as one data point. Only few other algorithms, for example SpecArray [13] and MZmine [14], can process centroided data—most algorithms operate on the much larger profile data. The use of centroided input data reduces both runtime and memory consumption of the algorithm. Additionally, it ensures applicability to basically all medium- and high-resolution data. Many other algorithms are designed for high-resolution data produced by the most recent mass spectrometer generations only, e.g., SuperHirn [15] and MaxQuant [16]. The reduced dataset size of centroided data also allows more sophisticated modeling approaches. To increase the accuracy of our algorithm, peptide signals are modeled based on an averagine [17] isotope distribution and a Gaussian elution profile. A thorough evaluation of the algorithm performance shows that it outperforms several state-of-the-art algorithms on typical proteomics datasets.

Chapter 4 deals with a machine learning approach for the prediction of DNA retention times (RT) in chromatographic separation systems. Our approach is based on support vector regression and incorporates novel features that encode the secondary structure of DNA. Other state-of-the-art prediction methods only consider the base composition and use simple linear models [18]. We could show that our method improves the prediction performance significantly, especially at low temperatures where the secondary structure is not suppressed. This chapter is only loosely related to the rest of the thesis because it focuses on oligonucleotide analytes. Still, it is not completely unrelated, because exploring the principles underlying chromatographic retention might help to improve all prediction models, also those for peptides and proteins.

Chapter 5 presents the main focus of this work—the OpenMS software framework and *TOPP – The OpenMS Proteomics Pipeline*. OpenMS provides data structures and algorithms for rapid development of mass spectrometry data analysis software. As mentioned above, software development in the field could be sped up by a feature-rich and mature software framework. The first part of the chapter presents the design goals of

OpenMS. Then, the implementation of selected functionality is described, focusing on core data structures, file I/O, visualization and selected algorithms.
The second part of the chapter introduces TOPP – The OpenMS Proteomics Pipeline. TOPP is a collection of computational tools, each providing functionality for a single analysis step. These lightweight command-line tools are used as building blocks for the construction of complex analysis pipelines. The flexibility of this modular approach makes TOPP a versatile tool for data analysis in proteomics and metabolomics. To improve the usability, TOPP provides a graphical user interface for the creation of customized analysis pipelines and a powerful visualization tool for LC-MS data.

The last chapter concludes this thesis with a summary of our work and some final remarks.

2. Background

Mass spectrometry is widely used in biochemistry and biomedical research. In the first part of this chapter, mass spectrometry and its applications in proteomics are introduced. The second part deals with machine learning techniques, which are often used to model physicochemical properties of biomolecules. In the third part, the basics of modern software engineering are introduced.

2.1. Mass spectrometry-based proteomics

The design of a proteomics LC-MS experiment depends on the goal of the study. For *drug target identification* the comparison of two cell states (e.g., healthy and diseased cells) is most important. The differentially expressed proteins are potential drug targets and can be used as markers for *diagnostics*. *Systems biology* tries to infer protein interaction networks and metabolic networks from the data. In this context, time series experiments are made that capture the dynamics of protein expression, for example after a perturbation of cells.

The general setup of a differential LC-MS proteomics experiments is shown in Fig. 2.1. In differential proteomics, protein samples from two or more states are compared. For simplicity, we assume that two states are to be compared. Thus, we start with two samples. Most protein samples are too complex to be analyzed by mass spectrometry directly. Typically, the sample complexity is reduced by fractionation, for example with liquid chromatography or two-dimensional gel electrophoresis. The protein fractions are then enzymatically digested to peptides of a limited length which can be handled more easily. After digestion, the protocols of label-free and isotope-labeled experiments differ. In stable-isotope labeling approaches, the samples are labeled, combined and analyzed on one LC-MS run. In label-free approaches, the samples are analyzed in individual LC-MS runs.

In this section, the basics of mass spectrometry in the context of proteomics are introduced. First, the principles and instrumentation of mass spectrometry are illustrated. Then, sample preparation steps and fractionation of samples by means of liquid chromatography are described.

2.1.1. Mass spectrometry

In 1898, Wilhelm Wien discovered the principles of mass spectrometry. He observed that beams of charged particles can be deflected by a magnetic field, and that the deflection depends on the mass-to-charge ratio (m/z) of the ions [19]. Ever since, these principles were used to separate small compounds in analytical chemistry.

The discovery of the soft ionization methods *electrospray ionization* (ESI) and *matrix-assisted laser desorption/ionization* (MALDI) made mass spectrometry applicable to larger biomolecules such as proteins. The importance of this discovery was underlined when Koichi Tanaka and John Bennett Fenn, the inventors of MALDI and ESI, respectively, were jointly awarded one half of the Nobel Prize in chemistry 2002.

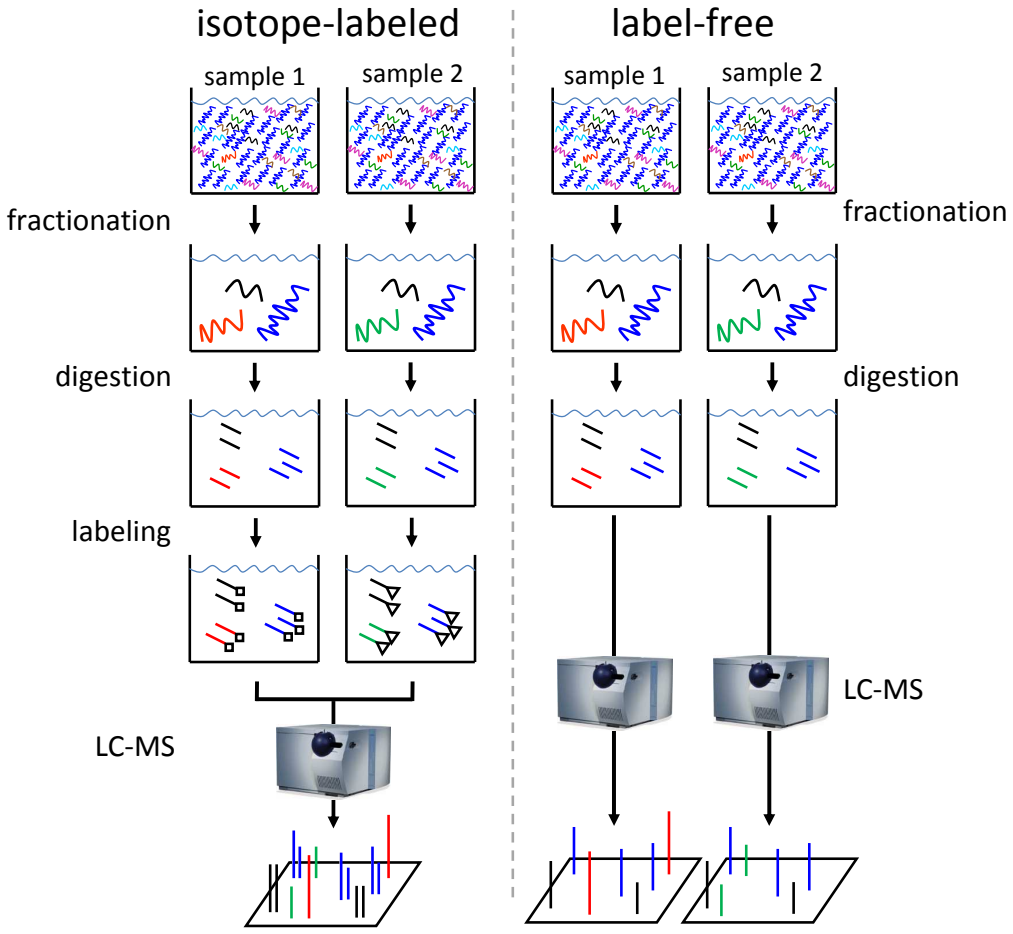


Figure 2.1.: Comparison of differential proteomics experiments with stable-isotope labeling and label-free differential proteomics experiments.

Fig. 2.2 shows the basic structure of mass spectrometers as they are used today. The analytes are first ionized in an *ion source*. Then, the ions are separated according to their mass-to-charge ratio in a *mass analyzer*. Finally, the ion count for each m/z value is detected by a *detector*. We will now describe a selection of widely used ion sources, mass analyzers and ion detectors.

Ion source

As mentioned above, ESI and MALDI are the most important ionization methods for biomolecules. Both are soft ionization methods that can ionize large biomolecules without fragmenting them. One reason for the softness of these ion sources is that ionization takes place at atmospheric pressure.

ESI [20] was established by John Bennett Fenn and co-workers in the 1980s. Fig. 2.3 shows a schematic representation of the ionization process in ESI. The solvent containing the analytes is forced through a charged capillary. At the tip of the capillary, a so-called Tylor cone forms which disperses the mixture to a fine aerosol. The solvent in the aerosol droplets is evaporated, typically by a warm stream of an inert gas such as nitrogen. There are two mechanisms which explain the final production of gas phase ions: *Coulomb Fission* assumes that the increasing charge density, due to evaporating solvent,

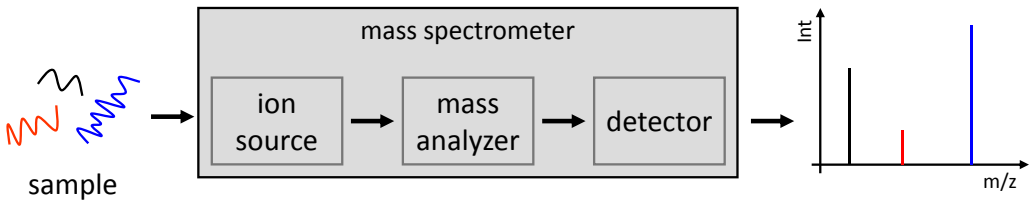


Figure 2.2.: Illustration of a mass spectrometer.

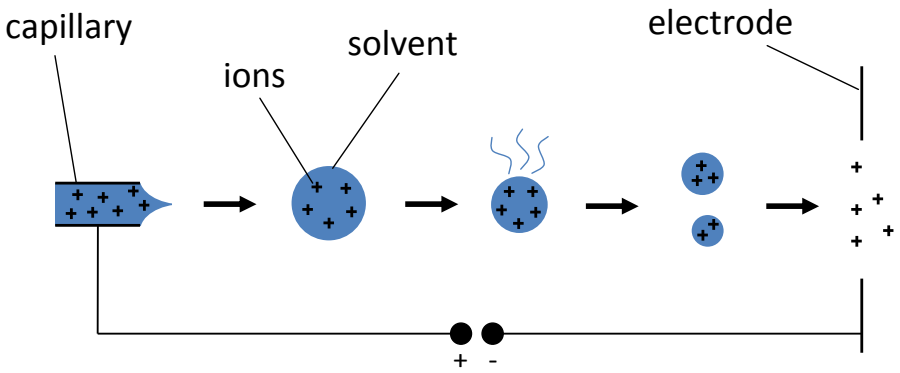


Figure 2.3.: Schematic representation of electrospray ionization.

leads to division of droplets, which finally leads to single ions. The second theory, *Ion Evaporation*, assumes that Coulombic repulsion leads to release of single ions from the droplets, because the Coulombic repulsion becomes stronger than the surface tension of the droplets. Although there is no final proof, it is believed that both mechanisms occur depending of the analyte size and other parameters. More information on ESI can be found in [21].

The second widely used ionization method for biomolecules is MALDI [22]. The breakthrough for large molecule MALDI came in 1987 when Koichi Tanaka and co-workers showed that it is possible to ionize proteins as large as carboxypeptidase-A (34472 Da).
The basic idea of MALDI is to ionize the analyte with a laser beam (see Fig. 2.4). To avoid fragmentation of the analytes by the direct laser beam, the analytes are co-crystallized with smaller molecules, the so-called matrix, on a metal plate. The matrix molecules absorb the laser beam energy and aid in vaporization and ionization of the analyte molecules. Today, mostly organic acids are used as matrix molecules because they strongly absorb specific laser wavelengths, e.g., of a 337 nm nitrogen laser. In contrast to ESI, which often produces ions of charge two and three, MALDI mainly produces singly charged ions from tryptic peptides.

Mass spectra can be recorded from positively charged ions (positive ion mode) and negatively charged ions (negative ion mode). The ion mode is also called polarity in this context. Usually, proteomics experiments are run in positive mode. Only in special cases negative ion mode is used. One application is to record spectra alternatingly in positive and negative ion mode, because the two kinds of spectra contain complementary information.

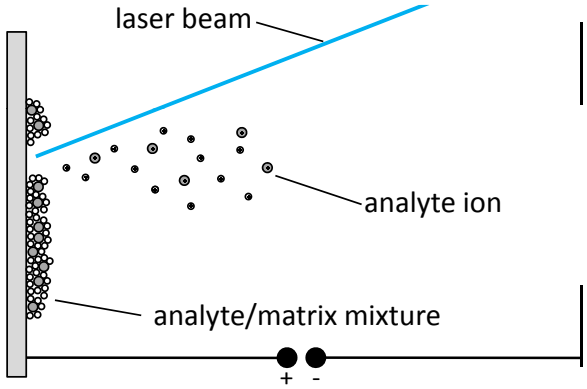


Figure 2.4.: Schematic representation of matrix-assisted laser ionization/desorption.

Mass analyzer

Mass analyzers separate ions according to their mass-to-charge ratio. A large variety of different mass analyzer types are used in mass spectrometry. All have specific strengths and weaknesses, and can be combined with certain ion sources. We will now briefly describe several common mass analyzer types.

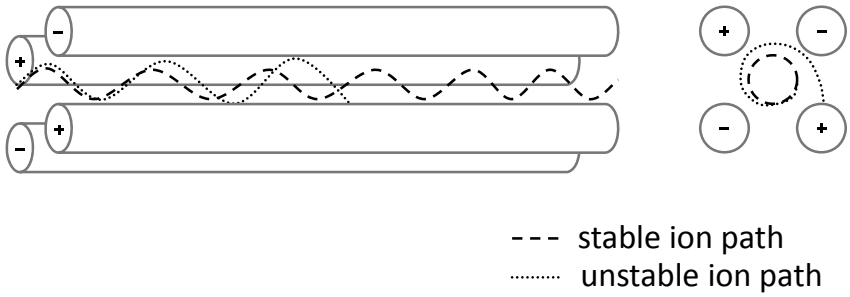


Figure 2.5.: Schematic representation of a quadrupole mass analyzer.

The schematic representation of a *quadrupole* [23] mass analyzer is shown in Fig. 2.5. A quadrupole consists of four parallel metal rods with applied electric currents, which create an oscillating magnetic field. The flight paths of ions passing through this field are stabilized or destabilized depending on their mass-to-charge ratio. Only ions of a specific m/z have a stable trajectory. All other ions leave the quadrupole at the sides or collide with the metal rods. The quadrupole effectively acts as a filter for a specific m/z value, which is why it belongs to the class of *scanning* mass analyzers. Modulation of the created electric field allows sweeping through a mass-to-charge range and recording a mass spectrum. Quadrupole instruments are popular because they are affordable and can be used for many different purposes. Most quadrupole instruments contain an ESI ion source.

The *quadrupole ion trap* [23] is based on similar principles as the simple quadrupole mass analyzer. The main difference is that the ions are trapped inside the mass analyzer

and can be sequentially ejected. This allows for a higher sensitivity, because no ions are lost. The inventor, Wolfgang Paul, was rewarded with a share of the Nobel Prize in physics in 1989. Quadrupole ion traps are also called *Paul traps* after their inventor.

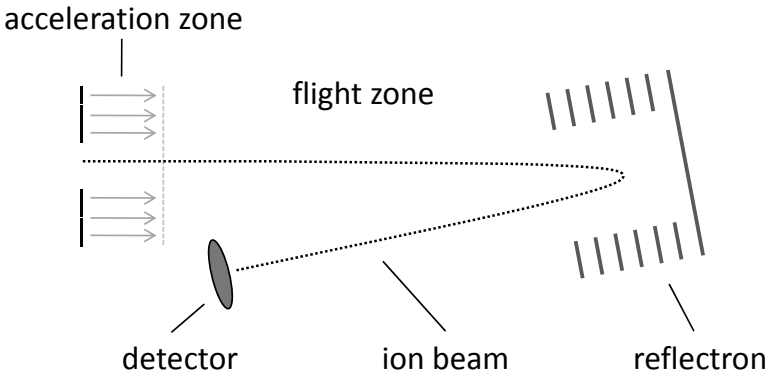


Figure 2.6.: Schematic representation of a time-of-flight mass analyzer with reflectron.

The *time-of-flight* (TOF) mass analyzer [24] is based on the acceleration of ions in an electric field of known strength. Thus, the kinetic energy transferred to the ions is known and depends only on the charge state of the ion. The velocity of an ion after acceleration depends on its mass: ions with a lower mass are faster. The m/z of an ion can be calculated from the time it needs to reach a detector at known distance from the ion source. Fig. 2.6 shows the schematics of a TOF mass analyzer. The flight path length can be extended using a reflectron, which uses a constant electric field to deflect ions towards the detector. The use of a reflectron allows for the construction of more compact instruments with an improved resolution. The TOF mass analyzer is mainly used in combination with a MALDI ion source.

The *Orbitrap* mass analyzer goes back to the work of Alexander Makarov [25] from the year 2000. It consists of a barrel-shaped outer electrode, which contains a spindle-like coaxial inner electrode. Between the two electrodes an electrostatic field is created in which ions oscillate in cycles along the inner spindle. The frequency of these harmonic oscillations can be detected, and the mass-to-charge ratio of the ions can be calculated from it. Orbitrap instruments are characterized by a high mass accuracy and a high dynamic range (the ratio of the lowest and highest detectable ion count).

Detector

Ion detectors are needed to count the number of ions with a specific mass-to-charge ratio. Because the number of ions is typically very small, a considerable amount of amplification is required. A widely used ion detector in mass spectrometry is the *electron multiplier*. We will now briefly explain the principles behind an electron multiplier with discrete dynodes. Fig. 2.7 shows the schematics of a secondary ion multiplier. The ion beam from the mass analyzer is transformed into an electron beam by a conversion dynode. The electron beam is then amplified by a cascade of secondary electron emissions. Finally, the electrons are detected by an electron collector. From the resulting signal an ion count can be calculated.

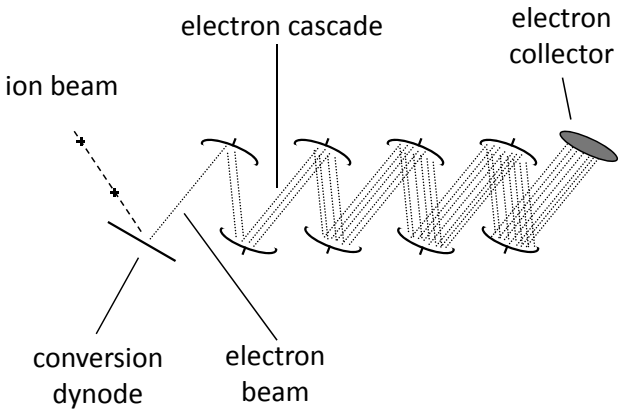


Figure 2.7.: Schematic representation of a secondary electron multiplier (ion detector).

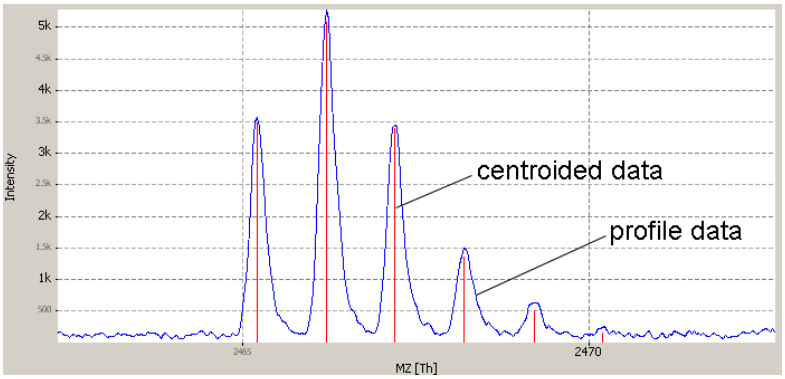


Figure 2.8.: Plot of a small m/z region of a TOF spectrum, showing a peptide isotope cluster with charge 1. In addition to the profile spectrum (blue line), peak centroids are plotted as red sticks.

2.1.2. Interpretation of mass spectrometry data and terms

The mass spectrum recorded by the MS instrument is a continuous signal sampled at regular intervals. It is typically plotted as mass-to-charge ratio (x-axis) against the intensity in arbitrary units (y-axis). Fig. 2.8 shows a small m/z region of a TOF mass spectrum. Continuous spectra are also called *profile spectra*. After centroiding a spectrum, i.e., detecting mass spectrometric peak apexes and reducing each peak to a single data point, a so-called *centroided spectrum* is obtained. Each peak in a mass spectrum corresponds to one or several chemical entities with a defined mass-to-charge ratio.

LC-MS experiments produce a collection of spectra with distinct retention times. Such a dataset is called a *peak map*. It is often visualized as a plot of m/z (x-axis) against RT (y-axis) with data points colored according to intensity. Fig. 2.9 shows a centroided peak map of a proteomics experiment. The inset of the figure shows a small data region with two *peptide features*. A peptide feature is defined as the set of signals caused by one peptide. In general, a *feature* is defined as the set of peaks that are caused by one chemical entity in a certain charge state.

Because of naturally occurring stable isotopes (e.g., ^{13}C makes up 1% of all carbon

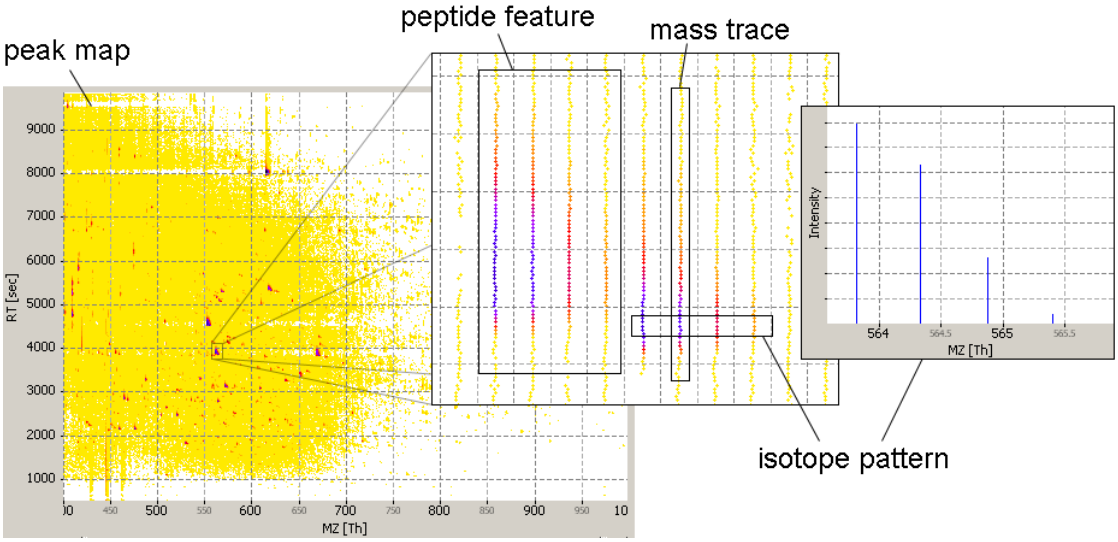
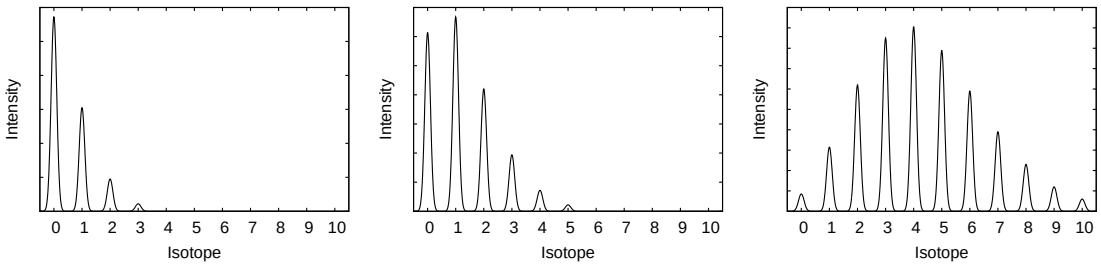


Figure 2.9.: Plot of a centroided peak map. The insets show a region with two peptide features and the corresponding isotope pattern.

atoms), each analyte species produces several peaks with one neutron mass difference. These peak clusters are called *isotope patterns*. Fig. 2.10 shows theoretical isotope patterns for peptides of different masses. The peak with the lowest mass, which is produced by analytes which contain only the most abundant isotope of each element, is called *monoisotopic peak*. An important property of isotope patterns is that the m/z distance of isotope peaks depends on the charge state of the analyte. The distance of two adjacent isotope peaks is equal to a neutron mass divided by the charge.

In LC-MS experiments, the isotope patterns of an analyte species can be observed in several adjacent spectra. Peaks recurring at similar m/z position in several adjacent spectra are called *mass traces*. Thus, each analyte species produces an isotope pattern of several mass traces. Examples of isotope patterns and mass traces of peptides can be found in Fig. 2.8 and Fig. 2.9.



(a) 1,000 Da (b) 2,000 Da (c) 7,000 Da

Figure 2.10.: Theoretical isotope patterns for peptides of different masses. On the y-axis the isotope peak number is denoted. The number 0 stands for the monoisotopic peak. All three isotope patterns are based on the averagine model [17] which reflects the average isotope distribution in proteins.

Projecting the mass spectrum intensities to the RT axis produces a *total ion current*

chromatogram (TIC). Another important type of chromatogram is the *extracted ion chromatogram* (EIC or XIC). EICs are the projections of a small m/z window to the RT axis. Fig. 2.11 shows an example of both chromatogram types.

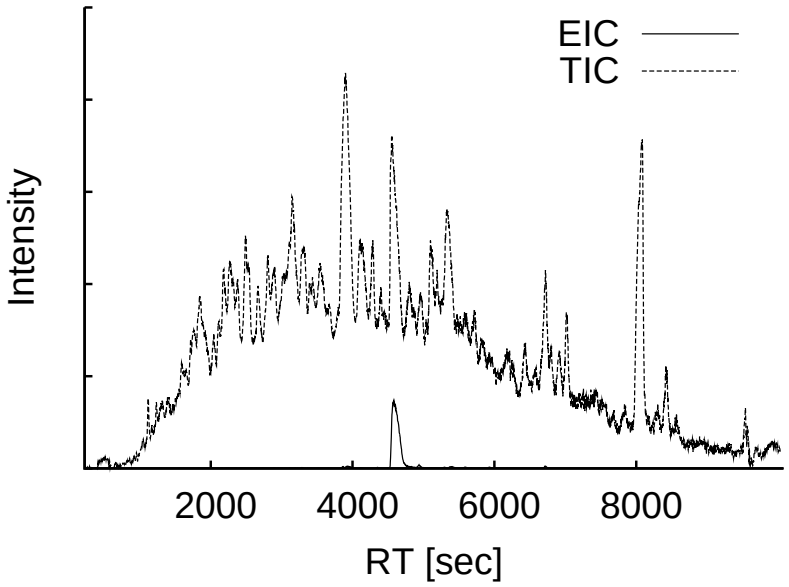


Figure 2.11.: A TIC chromatogram of the peak map shown in Fig. 2.9, and an EIC of the m/z range 552.70 to 552.71 Th from the same dataset.

2.1.3. Peptide/protein identification with mass spectrometry

A common problem in mass spectrometry is to determine the sequence of peptides or proteins. This is not easily possible, since mass spectrometry only measures the mass-to-charge ratio of analytes. The analyte identity cannot be inferred from a single mass-to-charge ratio because many sequences yield the same mass. Thus, several techniques have been established that allow the identification of proteins and peptides based on a set of fragment masses.

Peptide mass fingerprinting

In 1993, *peptide mass fingerprinting* [26] was developed for protein identification with mass spectrometry. The key idea of this technique is to fragment a protein to peptides and to identify the protein based on the fragment masses. The first step is to cleave the unknown protein to smaller peptide fragments, usually by digestion with trypsin. Then, a mass spectrum of the fragments is recorded. Finally, the measured peptide fragment masses are compared to a database of theoretical peptide fragment masses, which are determined by *in-silico* digestion of proteins with trypsin.
Because the amino-acid sequence has to be known for the in-silico digestion step, only proteins with known sequences can be identified with this method. A second drawback is that proteins have to be isolated and identified in individual MS runs. Digesting protein mixtures produces too many fragments, which complicates the statistical analysis performed during the database search. Thus, complex mixtures of more than a hand full of proteins cannot be identified with peptide mass fingerprinting.

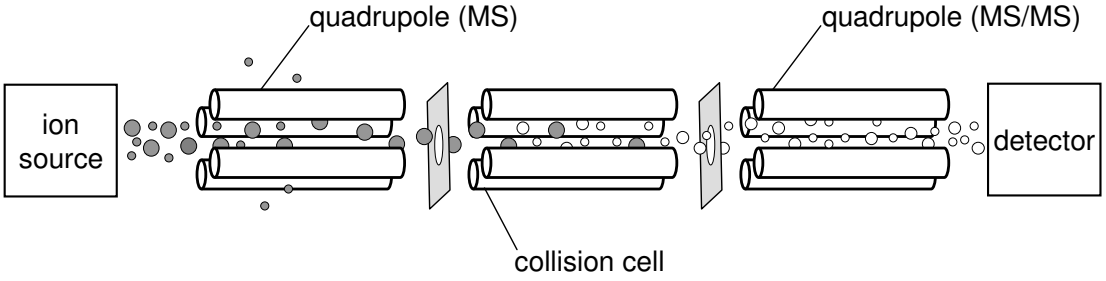


Figure 2.12.: Schematic representation of a triple quadrupole mass spectrometer.

Tandem mass spectrometry

Tandem mass spectrometry or *MS/MS* [27] aims at identifying single peptides by fragmentation of the peptides and subsequent analysis of the fragment masses. It is performed with special mass spectrometers, which provide a means to fragment selected peptides.
For example, *triple-quadrupole mass spectrometers* are frequently used for MS/MS. They contain a linear arrangement of three quadrupole mass analyzers (see Fig. 2.12). The first quadrupole acts as a mass filter to isolate a peptide ion of interest (called precursor ion). The second mass analyzer is filled with an inert gas such as Argon or Helium, and serves as a collision chamber for *collision-induced dissociation*. Finally, the third quadrupole records a fragment ion spectrum (also called product ion spectrum), from which the peptide sequence can be inferred. Other mass spectrometer types (e.g., *ion trap mass spectrometers*) and dissociation methods (e.g., *electron transfer dissociation*) are also used for MS/MS.

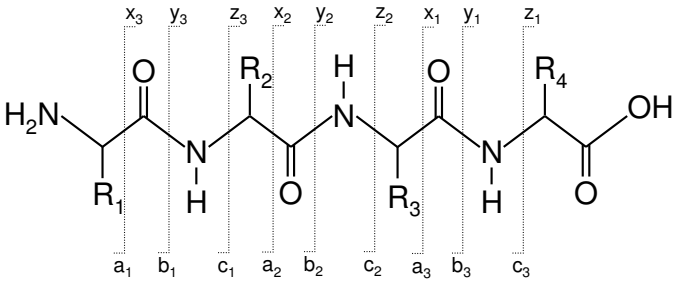


Figure 2.13.: Illustration of the most common peptide fragmentation ion series and the used nomenclature [28].

The dissociation of peptide ions does not produce random fragments. Mainly the amide-bonds of the peptide backbone break, producing y-ions if the charge is retained on the C-terminus, or b-ions if the charge is retained on the N-terminus (see Fig. 2.13). Ions of one series are numbered according to the number of residues they contain. Besides b-ion series and y-ion series, less frequent ion series are produced by other cleavages along the peptide backbone—additional ions are produced by adducts and neutral losses. Fig. 2.14 shows an example of a survey spectrum and the corresponding MS/MS spectrum.
The actual identification of peptides based on MS/MS spectra is either done by *de novo* identification or database search algorithms:

De novo identification: De novo identification algorithms [29, 30] try to identify a peptide based on the tandem MS spectrum and the precursor ion mass only. In order to achieve this, m/z distances of ions from one ion series are analyzed. The m/z distance between subsequent ions of one series corresponds to one amino acid. In theory, one ion series is sufficient to determine the sequence. However, usually not all ions of an ion series are present in an MS/MS spectrum. Another problem is that it is very difficult to correctly determine all ions of one series, because usually several ion series and many chemical noise peaks are present in an MS/MS spectrum. Due to these difficulties and long runtimes, database search algorithms are preferred over de novo identification algorithms.

Database search: The techniques employed by database search algorithms [6] are similar to those in peptide mass fingerprinting. A database with theoretical spectra is generated by in-silico fragmentation of peptides. The recorded MS/MS spectra are then compared to all theoretical spectra with matching precursor mass. If the two spectra show a high similarity, the sequence corresponding to the theoretical spectrum is reported as a putative peptide identification of the recorded MS/MS spectrum.
In general, database search algorithms show a better performance and runtime than de novo identifications. However, they are only applicable to peptides of known proteins. The most widely used database search algorithms are Mascot [9] and SEQUEST [31].

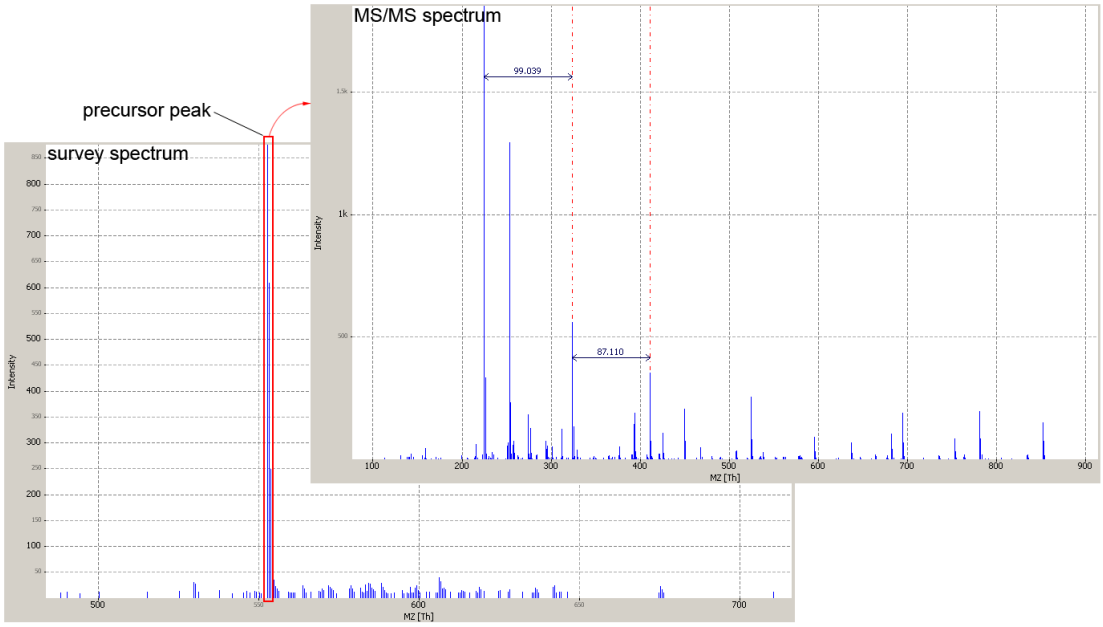


Figure 2.14.: Example spectra from a tandem mass spectrometry experiment. The precursor peak selected in the survey spectrum is fragmented to produce a tandem MS (MS/MS) spectrum. The annotated mass differences in the MS/MS spectrum could correspond to the amino acids serine (78.032 Da) and valine (99.068 Da).

2.1.4. Liquid chromatography

Liquid chromatography is an analytical chemistry technique for the separation of analytes. It is an important separation technique in the context of mass spectrometry because it can be coupled directly to certain mass spectrometer types—separating the analyte mixture before it is analyzed in the mass spectrometer. Liquid chromatography is not restricted to proteins and peptides. It is applicable to various analyte classes, for example metabolites and DNA. The basics of liquid chromatography are summarized this section.

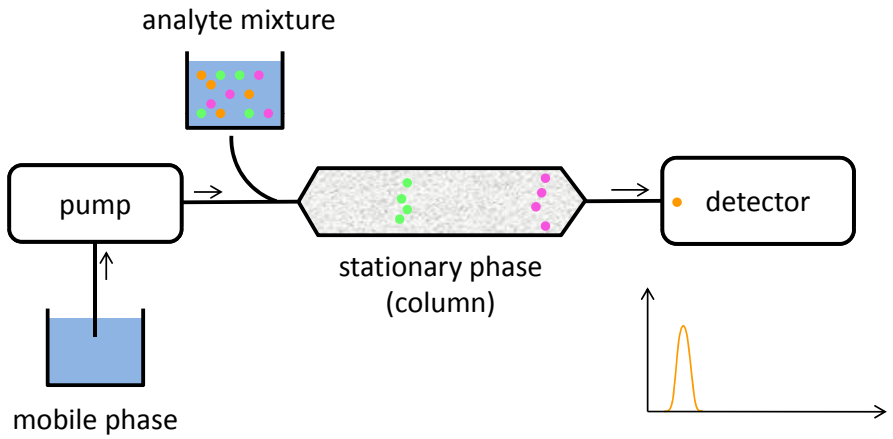


Figure 2.15.: Illustration of liquid chromatography.

In liquid chromatography an analyte mixture is dissolved in a *mobile phase*. After injection of the solution into the chromatographic system, it is forced through a column (the *stationary phase*) at high pressure. Each analyte species has a characteristic distribution ratio between mobile phase and stationary phase, which depends on the strength of interaction between the analyte and the two phases. The distribution ratio determines the time at which an analyte species elutes from the column, the *elution time* or *retention time*. Fig. 2.15 shows an illustration of a liquid chromatography run.

The choice of analytical column and solvents in the mobile phase has a large influence on the separation. Different column types and materials are used. The column is either packed with small particles or filled with a porous monolithic layer. When the stationary phase is more polar than the mobile phase, this is called normal phase liquid chromatography (NPLC). In proteomics, reverse phase liquid chromatography (RPLC) is more commonly used, in which the mobile phase is more polar. Typical materials for the stationary phase are silica and polystyrene-divinylbenzene. The mobile phase typically consists of water and acetonitrile, an organic solvent.

Liquid chromatography is called isocratic when the composition of the mobile phase remains constant. The composition can however change during the chromatography. This technique is called *gradient elution*. For example, in RPLC with water and acetonitrile as mobile phase, the amount of acetonitrile would be increased during the chromatography. Gradient elution speeds up the chromatography and thereby improves the peak shape of the analytes: Peak tailing is reduced and the peaks produced by late-eluting analytes are narrower and higher.

Liquid chromatography can be coupled on-line to certain mass spectrometer types. Alternatively, aliquots from the separation can be collected and analyzed in several

mass spectrometry runs afterwards.

2.1.5. Sample preparation

Proteomics sample preparation spans all steps between the extraction of a protein sample from an intact biological system and the analysis of the sample in an analytical instrument. Typical sample processing steps are fractionation, digestion, alkylation, reduction and labeling. In this section, we will shortly describe fractionation and digestion. Labeling is described separately in Section 2.1.6.

Most samples obtained from biological systems (e.g., whole-cell lysates) are too complex to be analyzed in a mass spectrometer directly. Thus, the complexity is reduced by fractionation of the sample. Two-dimensional gel electrophoresis is a commonly used technique for separation of protein mixtures. First, a pH gradient is applied that separates proteins by their isoelectric point. In the second step, the proteins are separated according to their size in orthogonal direction. The result is that proteins are spread out across a two-dimensional gel. Now fractions of proteins can be obtained from the gel for further analysis. Another popular fractionation technique is liquid chromatography described in Section 2.1.4.

Before the actual analysis in a mass spectrometer, most protein samples are digested using a protease. Digestion is necessary to ensure that the analytes have similar properties and are well-suited for MS analysis. The by far most widely used protease in proteomics is trypsin. Trypsin cleaves proteins after lysine or arginine, unless either of them is followed by a proline in C-terminal direction. Tryptic digestion is especially useful for mass spectrometry because each peptide fragment contains arginine or lysine. These basic amino-acids ensure ionizability of the peptides (in positive ion mode). The general size distribution of tryptic peptides is also well suited for mass spectrometry.

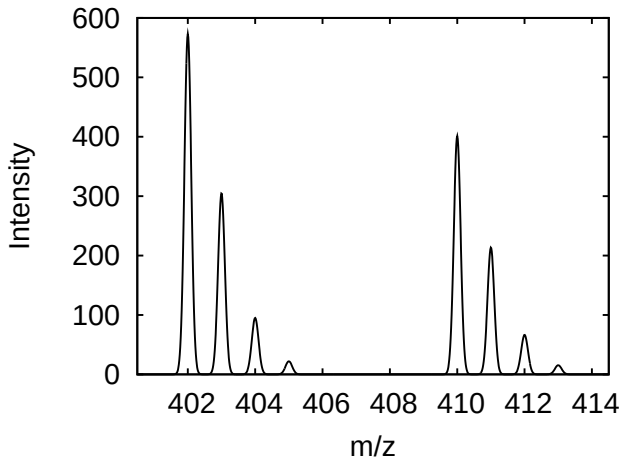


Figure 2.16.: Theoretical ICAT spectrum containing an ICAT-labeled peptide pair (light and heavy). The peptides are singly charged and, thus, have an m/z distance of 8 Th.

2.1.6. Stable-isotope labeling

In differential proteomics, stable-isotope labeling plays an important role. Several samples can be labeled and analyzed in one LC-MS run, which minimizes errors caused

by differences in sample preparation and chromatography. The relative expression differences of the same protein from different samples can be determined by comparing corresponding peaks with a defined mass difference. The mass difference depends on the labeling technique used. We will now briefly explain the most important stable-isotope labeling methods:
ICAT (Isotope-coded affinity tags) [32] is a chemical labeling for two samples with a mass difference of 8 Da. For each labeled peptide, two signals with a distance of 8 Da can be observed in the LC-MS data. Fig. 2.16 shows a theoretical example spectrum of an ICAT peptide feature pair.
SILAC [33] is a metabolic labeling technique that can be used in cell cultures only. One population of cells is grown with standard amino acids. The second population is grown with isotope-labeled amino acids. For example, arginines can be labeled with six heavy carbon atoms (^{13}C). The metabolic incorporation of the labeled amino acids causes a mass shift if the labeled peptides in the mass spectrum (see Fig. 2.17). The disadvantage of this method is that the mass difference of corresponding peptides depends on the number of arginine amino acids in the peptide.
iTRAQ [34] is a labeling technique for MS/MS experiments (see Section 2.1.3). Up to four samples are labeled using isobaric iTRAQ markers with a mass of 145 Da. The markers consist of a reporter group (114–117 Da) and a balancer group which maintains the overall mass of 145 Da. Because of their identical mass, identical peptides of different samples form one precursor peak and are fragmented together when recording an MS/MS spectrum. During the fragmentation, the reporter and balancer groups are cleaved off the peptide. Thus, the peptide can be identified from the peptide fragments, just as in label-free approaches. The relative quantitation is performed based on the reporter ion peaks with a mass-to-charge ratio between 114 and 117 Th.

Although stable-isotope labeling has many advantages, label-free quantitation is very popular as well. In label-free approaches no expensive labeling reagents are needed and the labeling step, which can also introduce errors, is not required. The downside of label-free approaches is that the evaluation of the produced data is more difficult. Several LC-MS runs have to be aligned in RT dimension, and corresponding peptide features have to be detected across several LC-MS runs.

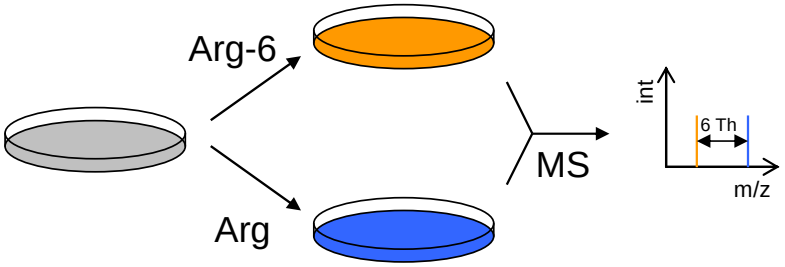


Figure 2.17.: Illustration of a SILAC experiment. One cell culture is grown with a light variant of arginine, one with a heavy variant of arginine (+6 Da). When recording a mass spectrum, the mass difference causes a 6 Th shift of the corresponding peaks (assuming the peptide contains one arginine and is singly charged).

2.2. Machine learning

Machine learning is a discipline of computer science that tries to learn characteristics from given data. In most cases the goal is to identify complex patterns in the data and to allow a classification of data points according to these patterns. Machine learning is closely related to multivariate statistics and density estimation of the underlying distribution. Depending on the used input data, three classes of algorithms are distinguished:

Supervised learning deals with labeled input data, i.e., each data point is associated with a label value. The goal is to find a general function that cannot only assign the correct labels to the given training data points, but also to new data points that were drawn from the same distribution. Examples of supervised machine learning algorithms are *decision trees*, *artificial neural networks* and *support vector machines*.

Unsupervised learning deals with unlabeled example data. Here the goal is to find patterns hidden in the data, which allow a discrimination of different classes of data points in the data. The most prominent unsupervised learning algorithms are *clustering* algorithms.

Semi-supervised learning falls between supervised and unsupervised learning. It deals with both labeled and unlabeled data. The reasoning behind this approach is that it might be impossible or undesirable to produce a large training dataset of labeled data points, but unlabeled data points are readily available. Typically, a small number of labeled data points is used in conjunction with a large number of unlabeled data points, which are mainly used for density estimation. This approach can lead to a significant improvement of the learning accuracy, compared to supervised and unsupervised learning.

In this work we will consider only *support vector machines*, a widely used supervised learning approach. As already mentioned above, support vector machines work on labeled training data. Depending on the type of label different types of supervised learning approaches are available. The simplest case is that two classes must be distinguished, i.e., only two types of labels are present. This case is called *binary classification*. If more than two labels need to be distinguished, this is called *multi-class classification*. Supervised learning with labels from a continuous domain, such as real numbers, is called *regression*. In the following, we will introduce the basics of *support vector classification* (SVC) and *support vector regression* (SVR).

2.2.1. Support vector classification

The separation of two classes of data points is a common problem in machine learning. In general, the data points are given in an n-dimensional *feature space* and the goal is to find an $(n-1)$-dimensional hyperplane, which separates the data. Fig. 2.18 shows a simple example, where two classes of two-dimensional data points and several separating hyperplanes are shown. Although each of these hyperplanes separates the data, they differ in their ability to generalize to unseen data. It is easy to see that the hyperplane with the maximum distance from the data points is the most general separator. This motivates why the separating hyperplane with the largest distance (margin) from the data points is the desired output of our classifier. This hyperplane is called *maximum-margin hyperplane* and classifiers which use this hyperplane are called *maximum-margin classifiers*. The training data points which lie on the border of the maximum-margin

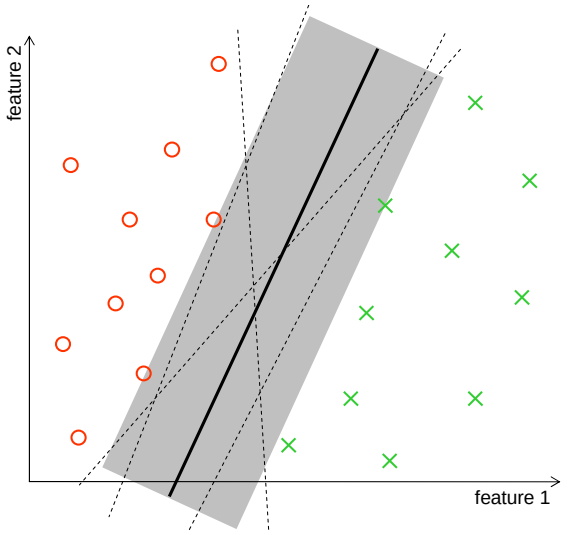


Figure 2.18.: Binary classification example with several suboptimal separating hyperplanes (dashed lines) and the maximum margin hyperplane (bold line). The maximum margin is denoted as a gray rectangle.

hyperplane define the orientation of the hyperplane in space—they are called *support vectors*.

Formal definition

After this short motivation, we will now introduce a formal definition of the problem. We are given a set of training data of the form

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^m \times \{-1, 1\}, \tag{2.1}$$

where -1 and 1 denote the labels of the binary classification. The goal is to find the maximum-margin separating hyperplane that separates the data points with $y_i = -1$ from those points with $y_i = 1$. A hyperplane can be defined as the set of points that satisfy the equation

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \tag{2.2}$$

where w is the normal vector of the hyperplane, b is the offset of the hyperplane from the origin and $\cdot$ denotes the dot product. Now we want to find a vector w and an offset b that maximize the margin of the hyperplane to the training data points. The margins can be described by the two equations

$$\mathbf{w} \cdot \mathbf{x} + b = 1 \tag{2.3}$$

and

$$\mathbf{w} \cdot \mathbf{x} + b = -1. \tag{2.4}$$

Note that the 1 and -1 are arbitrarily chosen here. They depend on the length of w only. In this case, the distance between the margins can be calculated as $\frac{2}{\|w\|}$. As the goal is to maximize the margin, we need to find parameters w and b that minimize the norm of w and separate the two classes, which can be expressed by the following optimization

problem:

$$\min_{w \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{2} \parallel w \parallel^2 \tag{2.5}$$

$$\text{subject to} \qquad y_i(\mathbf{w} \cdot \mathbf{x_i}) + b \geq 1 \qquad \forall i \in \{1, \ldots, n\} \tag{2.6}$$

The constraints in this formulation ensure that the training data points x_i have a minimum distance of 1 from the hyperplane and that the two classes are separated by the hyperplane. The convenient choice of -1 and 1 as labels y_i, allows the formulation of a single constraint for both classes by simply multiplying with the class label.
The factor $\frac{1}{2}$ and the square of the norm of w in the optimization problem are used for mathematical convenience only—they do not alter the minimum. This optimization problem can be efficiently solved by standard quadratic optimization techniques. More information about the optimization problem and support vector machines in general can be found for example in [35].

Non-linear classification

Linear support vector machines can be used only if the two classes can be separated by a hyperplane. This poses a serious restriction on the applicability of the method. In 1992, Vapnik *et al.* proposed to generalize support vector machines to non-linear problems [36] by applying the kernel trick [37].
The kernel trick employs that problems which are not separable by a linear classifier in the original feature space can become separable in a higher-dimensional feature space. Thus, the input features are transformed to a higher-dimensional feature space where a linear separation might be possible. If a linear classifier can be found in the higher-dimensional feature space, it corresponds to a non-linear separation in the original features space (see [38] for more details).
The explicit transformation of the features to a higher-dimensional feature space can be quite time-consuming. It can be circumvented, if only the inner product in the transformed feature space is required for the optimization. For support vector machines, this can be achieved by reformulating the optimization problem.
Fig. 2.19 shows an example of a simple kernel transformation. Commonly used kernel functions in support vector classification are the linear kernel, the polynomial kernel and the radial basis function kernel. Details about different kernels can be found in [39].

Soft-margin classifiers

Another important modification, which made support vector classification more robust with respect to errors in the training data, was proposed by Cortes and Vapnik in 1995. They introduced so-called slack variables, which allow the misclassification of data points in order to find a more general classifier [40]. This is necessary for data with possibly misclassified training data points or data with noisy features. The slack variables $\xi_i \geq 0$ are used to relax the constraints of the optimization problem:

$$y_i(\mathbf{w} \cdot \mathbf{x_i}) + b \geq 1 - \xi_i \qquad \forall i \in \{1, \ldots, n\} \tag{2.7}$$

To avoid the misclassification of too many training data points, the slack variables have to be integrated into the objective function as well:

$$\min_{w \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{n} \xi_i \tag{2.8}$$

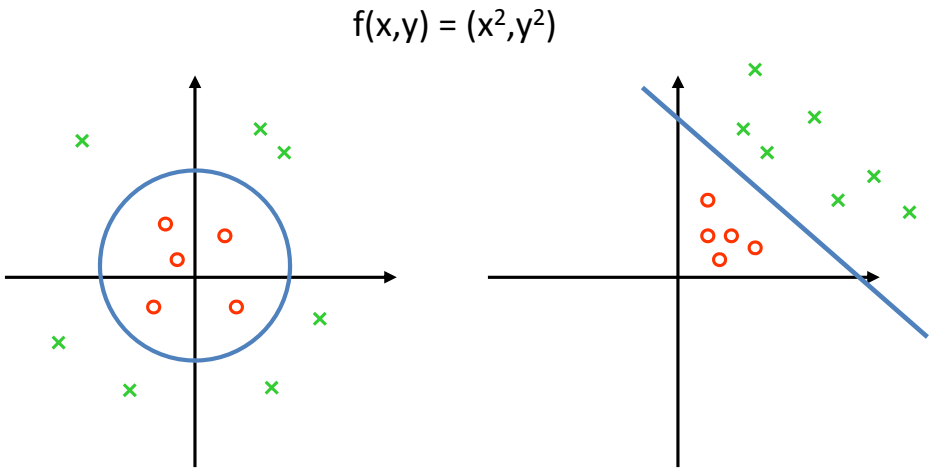


Figure 2.19.: Classification example which is not linearly separable. After the transformation of the data to another feature space of the same dimensionality, the data is linearly separable.

Now the minimization problem minimizes both the norm of w and the sum of all slack variables (also called training error). The parameter C is used to balance these contributions.

Multi-class support vector classification

One drawback of support vector classification is that it is defined for binary classification only. To solve a multi-class classification problem, the problem has to be transformed to binary classification problems. Several such transformations are known. One possibility is to train an SVM for each class that discriminates the class from all other classes (one-versus-all). In this case, the class with the highest probability is selected as result of the classification. Another possibility is that one SVM is trained for each pair of classes (one-versus-one). In this case, the class with most votes is selected as the final result.

2.2.2. Support vector regression

Support vector regression (SVR) is very similar to support vector classification. The main difference is the domain of the label. In support vector regression the label is from the continuous domain of real numbers (it is also called the *response*). The input data consists of training data points and the respective labels:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^m \times \mathbb{R} \tag{2.9}$$

In analogy to support vector classification, the optimization problem for the so-called ε-SVR [39] can be formulated as

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} |y_i - f(\mathbf{x}_i)|_\varepsilon, \tag{2.10}$$

where $f(x_i)$ is the predicted response for the data point x_i. The first addend minimizes the model complexity while the second minimizes the ε-insensitive training error, i.e.,

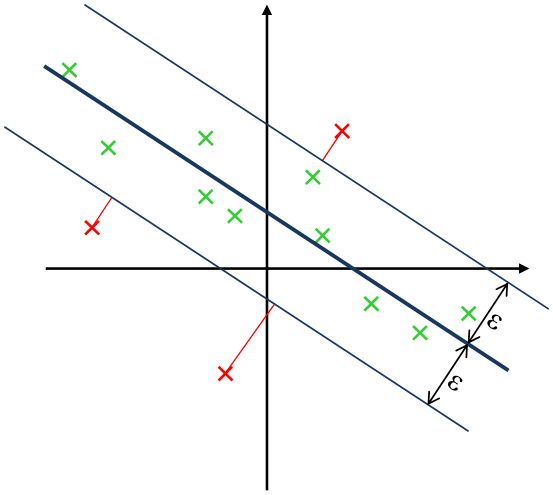


Figure 2.20.: Support vector regression example. Deviations from the model inside the ε-margin are not penalized. The sum of the larger deviations (red lines) is minimized.

training errors below a fixed ε are not penalized. C is a parameter which defines the trade-off between these two objectives. Fig. 2.20 shows an illustration of a linear SVR model.

In 2001, Schölkopf *et al.* [41] proposed ν-SVR, a modified version of ε-SVR which minimizes ε along with the model complexity and the training error. This modification simplifies the use of SVR, as ε no longer has to be chosen *a priori*. Today, ν-SVR is widely used to solve regression problems from different fields of application.

2.2.3. SVM model generation and performance evaluation

In all machine learning approaches special care has to be taken to avoid overfitting. Overfitting occurs when a model learns to classify training data based on random noise in the data, rather than on the general underlying distribution. Overfitting frequently occurs when the model has too many degrees of freedom in comparison to the number of training data points. In general, the overfitted models explain the training data very accurately, but their predictive power on other datasets is very low.
To avoid overfitting during the model generation with support vector machines, the performance of an SVM predictor is usually given as Q^2, the squared correlation coefficient between the predicted response on previously unseen data and the true response of the data. The correlation of the predicted and true response on the training data is called R^2.
Fig. 2.21 shows the general procedure of SVM training. First, the available data is randomly split into a training dataset and a test dataset. The model is created based on the training dataset. Then, the model performance is evaluated on the test dataset, which was not used for the model generation.

k-fold cross-validation

The performance for a single random split of the data may deviate significantly from the true prediction power. To counter this effect several random splits can be used, which

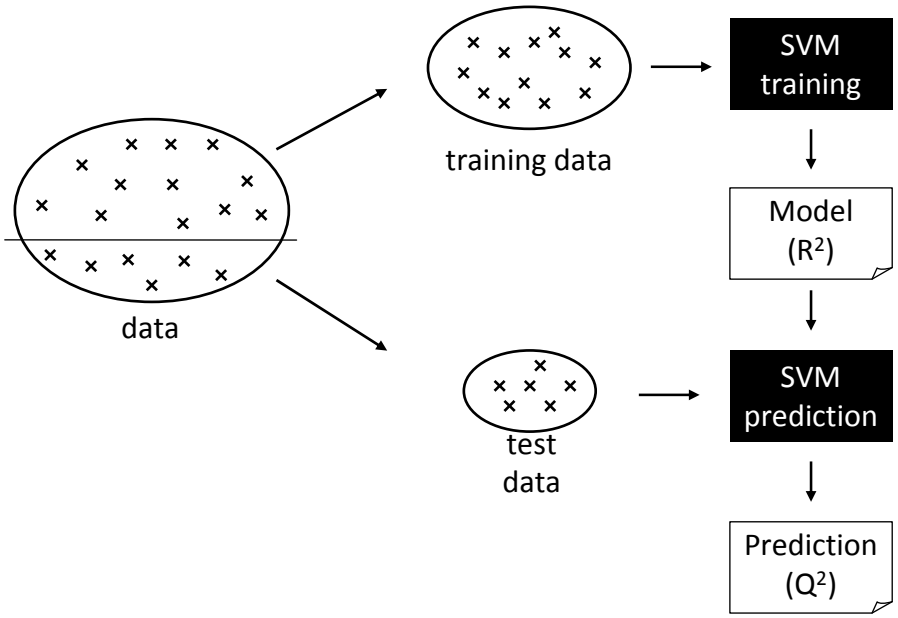


Figure 2.21.: Illustration of the typical training procedure used for support vector machine model generation.

allow calculating an average performance on the data. To make sure that each data point is part of the test dataset at least once *k-fold cross-validation* is used: The data is split into k subsets of equal size and each subset is used as test dataset once. This method allows a good estimation of the true prediction power.

Nested cross-validation

Both SVC and SVR models have parameters which must be optimized during the model creation. The parameters stem from the used kernel function and from the SVM itself (e.g., the tradeoff between model complexity and prediction error C). Typically, a second k-fold cross-validation on the training dataset is used for a grid search on the parameter space. The use of two cross-validation steps, one for parameter optimization and one for evaluation of the prediction performance, is called *nested cross validation*. Today, nested cross-validation is state-of-the-art for assessing the prediction performance of SVM models.

Performance measures

Many different scoring functions can be used to measure the performance of machine learning models. In the context of SVR the *Pearson correlation coefficient* [42] is frequently used. For the set of value pairs (x_i, y_i) it is calculated as

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{2.11}$$

where $\bar{x}$ and $\bar{y}$ are the mean of the respective samples. The Pearson correlation coefficient ranges from -1 to 1. A value of 1 means that a linear function with positive slope perfectly describes the relation between the two samples. A value of -1 means that a linear function with negative slope perfectly describes the relation between the samples

(anticorrelation). A value of 0 means that there is no correlation between the samples. As mentioned before, the squared correlation coefficient is used to measure the model performance on training data (R^2) and the model performance on unseen data (Q^2).

For SVC, and in general for two-class classification problems, the *Matthews correlation coefficient* [43] is frequently used. It is based on the counts of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). It is calculated as

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{2.12}$$

Just like the Pearson correlation coefficient it ranges from -1 to 1 and can be interpreted in a similar way.

2.3. Software engineering

The systematic approach to development and maintenance of software products is termed *software engineering*. It covers the whole software life-cycle from planning of the software, over its implementation, to maintenance. Software engineering can be divided into the following disciplines:

- Requirements engineering
- Analysis and design
- Implementation
- Testing
- Deployment
- Maintenance

The concrete practices used in these disciplines are defined by so-called *software development processes*. Depending on the process, practices for all disciplines or only a subset of the disciplines are defined. In this section we will first give an overview of software development processes. Then, we briefly summarize the most important aspects of each development discipline.

2.3.1. Software development processes

The software development process is a set of principles, practices and guidelines that define how a software product is developed. Over the last 50 years, different software development processes have been proposed, each with its advantages and disadvantages. Depending on the project setting, software engineers adopt the most appropriate process or a combination of principles from several processes.

The waterfall model

The first defined software development process was the *waterfall model* in which planning, design, implementation and testing are performed in a strict sequential order. A new phase starts only after the previous phase has been completed. However, it soon became obvious that this model is applicable to the simplest projects only. For large or complex software products an up-front complete and correct specification is not possible [44].

Iterative and evolutionary development processes

Iterative and evolutionary software development processes try to overcome uncertainties in the specification by iterative development of the software product. An iteration is a mini-project which itself contains all required steps from planning over design and implementation to testing. The goal of an iteration is to create a functional, stable and tested software that implements more features than the result of the preceding iteration. The most important features (those with the highest value for the user and those with the highest risk) are developed in early iterations. Later iterations refine the behavior based on user feedback and add lower priority features.

The *Spiral model* was developed for very large and complicated projects. It consists of a few long iterations (six months to two years). Each iteration in principle follows the waterfall model. The spiral model is more flexible than the waterfall model but still

very rigid in comparison to other iterative processes. For small and medium size projects more flexible processes like the Unified Process are preferred.

The *Unified Process* [45] is a popular iterative process framework because it is suitable for all project sizes and the degree of formality is adaptable to project needs. Nearly all artifacts (documents created in addition to the code) are optional and should only be created when needed. The key practices of the Unified Process are:

- Timeboxed iterations of two to six weeks are used. If the iteration time is insufficient, features are moved to the next iterations rather than increasing iteration time.
- High-risk and high-value elements are implemented in early iterations.
- The software quality is continuously verified through integrated testing.
- Visual modeling is used to explore the design of a software before programming.
- Requirements are collected and refined iteratively, for example in short requirements meetings in the early iterations of the project.
- Requirements changes are managed through change request protocols.

The Unified Process, like the waterfall model, is organized in phases. However, the phases do not correspond to the disciplines of software development. Each phase contains work in most disciplines but the relative effort and emphasis change. The first phase, *Inception*, is ideally very short and aims at identifying the most important requirements and the vision of the project. Prototyping of the software is encouraged. In the second phase, *elaboration*, architecturally significant elements are programmed and tested. In parallel to the programming, requirements are refined so that after this phase most requirements are stable. At the end of this phase a semi-reliable project plan is possible. In the third phase, *construction*, the remainder of the system is build while finalizing the requirements and writing all artifacts. In the *transition* phase the software product is verified and deployed.
Fig. 2.22 shows an exemplary work distribution between software development disciplines when using an iterative software development process. Each phase of the *Unified Process* would correspond to one or several iterations in this figure.

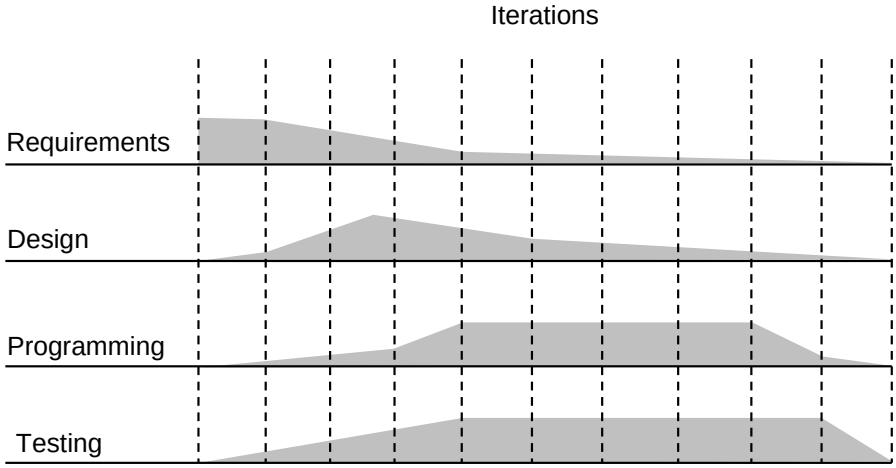


Figure 2.22.: Illustration of an example distribution of work among the software development disciplines when using an iterative software development process. The number of iterations is arbitrarily chosen here.

Agile development processes

Agile development processes are a subclass of iterative and evolutionary development processes. They place even more emphasis on *agility*, i.e., fast and flexible response to change. The common motto of all agile processes is to *embrace change* rather than avoiding it. Finding an exact definition for agile development processes is however very difficult because agile methods differ very much in concrete practices. However, the common ground of agile development processes is stated by the *Agile Manifesto*:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

Two prominent agile processes, *Extreme programming* and *Scrum*, are now summarized shortly.

Extreme Programming (XP) is a well-known agile method [46] which is used for projects with up to 20 developers and a tight timeline. It mainly provides practical programmer-relevant techniques for delivering high-quality software in spite of changing requirements. It is very communication- and team-oriented: customers, managers and developers work in a common project room. Due to the continuous availability of customers, the overhead for requirements specification is reduced to a minimum—just as all other documentation overhead. Extreme programming uses short iterations of one to three weeks. Implementation starts with the first iteration and each implemented feature is tested using automated tests. It is recommended to use *test-driven development*, i.e., write the tests before implementing the functionality. To ensure correctness after code changes, all tests are continuously executed on a dedicated test server. This is especially important because the design is not done prior to implementation, but is the result of continuous refactoring. Because refactoring can break existing functionality, continuous integration and testing is essential. The coding is always done in pairs of developers, which change frequently. This practice significantly reduces the defect rate because all code is real-time reviewed. Additionally, knowledge transfer between developers is ensured, which is important because Extreme Programming relies on high software craftsmanship of all programmers. Another important practice is team code ownership. This practice implies that any pair of programmer can implement a new feature or improve a piece of code. Enforcing a common coding style, adhering to the simplest design and automated test builds support this practice. Because Extreme Programming mainly guides implementation practices, it can be easily combined with other processes such as Scrum or Unified Process.

Scrum [47] emphasizes self-directed teams that develop with minimal overhead, using empirical processes rather than predefined processes. Typically, a team of up to seven people works in 30 calendar day iterations (called sprints). Each sprint starts with re-prioritizing remaining requirements and creating a task list for the sprint (called sprint backlog). During the sprint, the task list is not changed unless unavoidable and the team alone decides how to perform the planned tasks. Each workday starts with a 30 minutes team meeting (the Scrum meeting) which monitors the progress of the team and identifies problems. After each iteration, a software demo is held to external stakeholders and the sprint is reviewed. Scrum can also be applied to projects with more than seven developers by forming several teams and coordinating the teams using a second scrum meeting where team progress is reported. Because Scrum mainly guides

project management, it can be easily combined with other processes such as Extreme Programming.

After this short overview of software development processes, we will describe important aspects of the software development disciplines.

2.3.2. Requirements engineering

Requirements engineering is typically the first step of each software project. It ensures that all relevant requirements of a software product are known and understood on the necessary level of detail. In this context a *requirement* is defined as a statement on a characteristic of a product, of a process, or of the persons involved in the process. Requirements engineering encloses the collection of requirements, verification of requirements and management of requirements. We will now briefly describe the main tasks of a requirements engineer.

Requirements elicitation

Requirements elicitation tries to collect all requirements of a software product. This is typically done iteratively with increasing level of detail. First, the involved stakeholders (all persons that are affected by the development and the use of the system), the intended use of the product and the context of the product are defined. The second step is to collect coarse product requirements, which describe the characteristics of the product from a high-level view. User needs are often determined using questionnaires or by interviewing representative users. In a third step, the *software requirements* are defined by transforming each coarse product requirement to a set of more detailed requirements. Typically, software requirements are described by use cases, activity diagrams or state diagrams. They should be complete, unambiguous, consistent and testable.

In parallel to the requirements, a glossary is created which defines all domain-specific terms used in the requirements descriptions. A glossary is very important to make sure that all project participants have the same understanding of the terms and thereby of the requirements.

Requirements analysis and validation

Once requirements are collected, they need to be grouped and dependencies between the requirements have to be determined. Grouping is done in different ways, for example according to priority, according to user roles, or into functional and non-functional requirements.
Grouping and determining dependences is especially important for risk management. Risk management identifies risks to a software project, i.e., things that threaten the project timeline or budget. It is common that low priority requirements are postponed to later versions of the software to ensure fulfillment of the project timeline and budget.

Requirements management

Just as the source code, requirements exist in different versions. For example, one set of requirements exists for each planned release or for each level of detail. Additionally, requirements are annotated with meta information such as the cost of implementation, priority, dependencies, etc. The management of requirements is an important part of requirements engineering. To organize requirements and keep track of the changes, special software tools for requirements engineering are used.

2.3.3. Analysis and design

Today, most large software projects are implemented using object-oriented programming languages. Before implementing a complex software product, the standard approach is to model the system through objects and interactions between objects. We will now summarize the modeling process and introduce the tools used in the process. If you are not familiar with object-oriented programming, you should read Section 2.3.4 before continuing.

Object-oriented analysis and design

Object-oriented analysis and design is a modern approach towards the modeling of a software system prior to implementation. Each object represents an entity of the system through encapsulated data and methods that work on the data. The behavior of objects and interactions between objects define the behavior of the overall system. To represent object-oriented design, different notations have been developed of which the *Unified Modeling Language* (UML) is the most prominent. UML is described in the next section.

The first step, the *analysis*, models the problem domain based on requirements. In this step, the focus of the model is on the problem domain only. Constraints imposed by non-functional requirements, choice of programming language, etc. are not considered. The model is a collection of documents that describe the system on a high level of abstraction (called conceptual or semantic model). The model typically comprises use case descriptions, UML class diagrams, UML sequence diagrams and mock-up graphical user interfaces.

The second step, the *design*, re-models the system taking the constraints not considered by the semantic model into account. The outcome of the design step are refined UML class diagrams and UML sequence diagrams, which can be used as a basis for the implementation of the software. During the design step, so-called *design patterns* are used. Design patterns are reusable solutions to commonly recurring design problems.

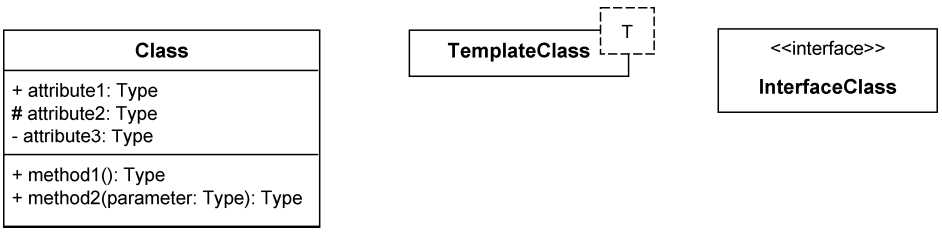


Figure 2.23.: Three UML class representations: a standard class representation with attributes and methods (left), a parametrized class representation (middle) and a stereotype class representation (right). In the middle and on the right side, the class members are omitted.

UML

The *Unified Modeling Language* (UML) is a standard tool for creating visual models of software systems. UML diagrams offer two different views on the system to model. The *static view* focuses on the structural details such as classes including members and the relationships between classes. The *dynamic view* depicts the behavior of the system by modeling interactions between objects and the resulting changes to the system. There

are several different diagram types both for static and the dynamic view. In this work, mainly the *class diagram* is used, which is explained in detail now. For more information about UML in general and the different diagram types see [48].

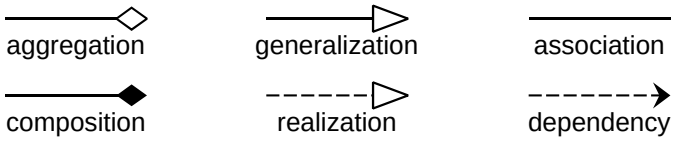


Figure 2.24.: Representations of relationships in UML diagrams.

The UML *class diagram* is a static structure diagram that provides a view on class details and the relationships between classes. Classes are represented by a rectangle subdivided into three parts. The upper part contains the class name. The optional lower parts contain the members (attributes and methods) of the class. Fig. 2.23 shows three example classes. Public members are prefixed with '+', protected members with '#' and private members with '-'.

Relationships between classes are represented by several types of lines and arrows. Fig. 2.24 gives an overview of the most important relationship types. *Aggregation* and *composition* express whole-part relationships (diamond on the container side). Aggregation is stronger and typically implies that the contained objects are destroyed when the container is destroyed, while composition is used when the container only references the contained elements. *Generalizations* are drawn as solid lines from base class to specialized class with a hollow triangle on the base class side. *Realizations* are drawn as dashed lines with a hollow triangle. In most cases they indicate that a class implements an interface (hollow triangle side). *Associations* are drawn as solid lines. They express relationships between instances of the connected classes and can be annotated with several properties, e.g., association type, role names of the participating objects and multiplicities. *Dependencies* between classes are drawn as dashed lines with an arrowhead and are usually annotated with the dependency type.

Design patterns

In software design, frequently recurring design problems exist. Reusable solutions to such problems, which have proven to solve the problem effectively, are called *design patterns*. The first comprehensive collection of design patterns has been published by Gamma *et al.* [49] in 1995. Since then, design patterns have become very popular, not least because they provide a common vocabulary that allows communicating design problems and the applied solution.

Although design patterns provide a tested solution to a common problem, they should not be applied without thorough consideration. Often, design patterns need to be adapted to fit the concrete problem, or a combination of several patterns offers the best solution.

Design patterns are traditionally categorized in *creational patterns*, *structural patterns* and *behavioral patterns*. We will give an overview of the patterns which have been frequently used in this work. More details and many patterns not mentioned here can be found for example in [49].

Creational patterns deal with object creation:

Singleton ensures that only one instance of a class exists. This is usually achieved by

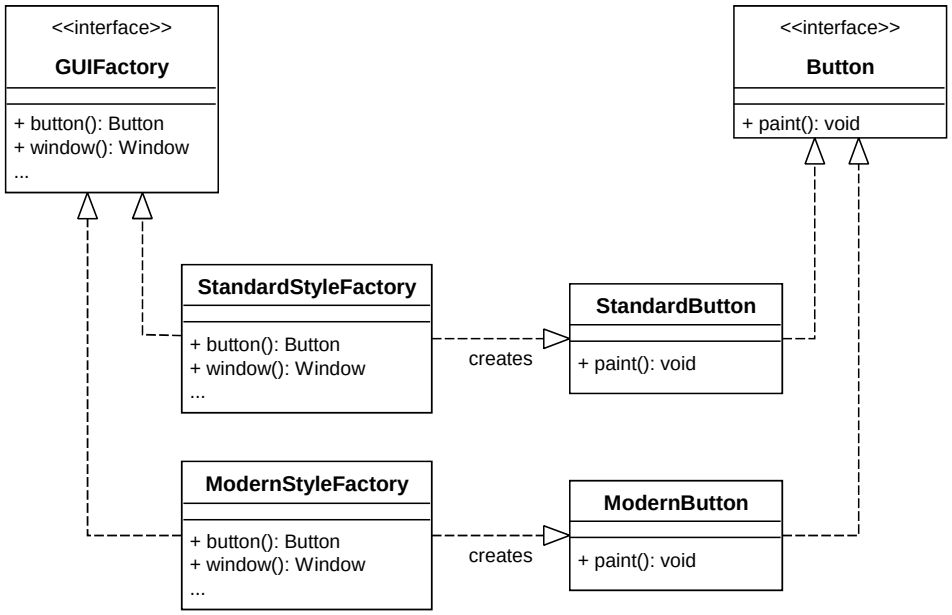


Figure 2.25.: Example of the abstract factory pattern. The classes implementing the *GUIFactory* interface are used to create concrete instances of GUI classes such as *Buttons*, *Windows*, etc.

hiding the constructor and providing a static access method to a static instance.

Factory method provides an interface for creating an object of a certain supertype. The concrete factory class, which implements the interface, defines which concrete class is instantiated.

Abstract factory provides an interface to create a family of related products. It separates the creation of an object family from its use. Fig. 2.25 shows an example of the pattern.

Behavioral patterns implement frequently occurring behavior:

Iterator allows accessing the elements of a data structure sequentially without knowing the underlying structure. It is for example used to provide easy access to tree structures.

Null object is used as a default object when a *null pointer* would have to be returned otherwise.

Observer implements a one-to-many notification system to allow multiple objects to be updated when the central object changes its state. The observer pattern is mainly used for distributed event handling, for example in GUIs. Fig. 2.26 shows an example of the pattern.

Structural patterns influence the static structure of the system:

Adapter is a wrapper that changes the interface of a single class to the interface expected by a client.

Facade provides a unified interface to a complex subsystem, in order to make the subsystem more easily usable.

Composite lets clients treat individual objects and subtrees of a tree structure alike. Fig. 2.27 shows an example of the pattern.

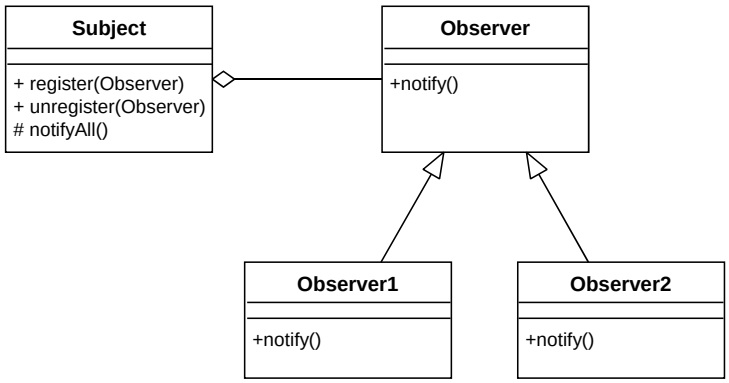


Figure 2.26.: Example of the observer pattern. The *subject* is the central object, which allows observers to register and unregister. When the state of the subject changes, it notifies all observers by calling their *notify* method.

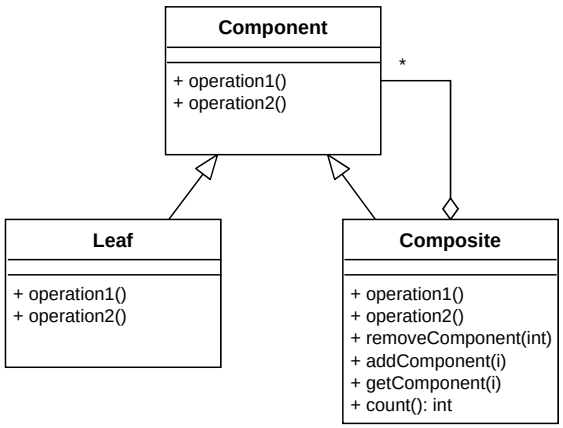


Figure 2.27.: Example of the composite pattern. The client accesses the data structure only though the component base class interface, which provides all operations needed by the client. Typically, the composite class delegates operations to leafs, which implement the operations.

2.3.4. Implementation

The implementation part of the software development process mainly consists of coding (writing new code), refactoring (improving existing code) and debugging (fixing errors in the code). Today, most software is implemented using modern object-oriented programming languages such as C++, Java and C#. In this section, we will shortly summarize the main features of object-oriented programming and introduce the most important tools which support software implementation.

Object-oriented programming

Object-oriented programming (OOP) is a programming paradigm which bundles data and operations on the data to *objects*. It is considered to be best practice to create reusable pieces of code. In this section we will introduce important terms and core concepts of object-oriented programming.

Classes can be seen as the blueprints to create objects at program runtime. They define *attributes* (data fields) and *methods* (operations) of the corresponding objects. Attributes and methods together are often referred to as *members* of the class. Additionally, classes define how objects are created. Objects of a class created at runtime are called *instances* of the class.

Not all object-oriented programming languages offer the same set of features, but a number of core concepts exist that most object-oriented languages share [50]. These concepts are inheritance, encapsulation and polymorphism:
Inheritance is perhaps the most fundamental concept of object-oriented programming. It allows a class to inherit the characteristics (members) of a parent class, and add new behavior or specialize the behavior. *Multiple inheritance*, i.e., a class can have several parent classes, is not supported by all object-oriented languages.
Encapsulation in OOP means that classes can hide details from clients. Both attributes and methods play a role in encapsulation. Methods often encapsulate complex operations to provide a simplified interface for common tasks. Methods can also encapsulate attributes, i.e., clients can access attributes through special accessor methods only. Attribute encapsulation is mostly used to ensures that objects are always in a valid state.
Polymorphism captures a language's ability to treat subclasses and parent class uniformly, as long as only the parent class interface is used. This technique is frequently used to implement generic containers of objects derived from a common parent class. Another example of polymorphism are interface classes. They only declare methods of a class, but do not provide an implementation. Thus, interface classes cannot be instantiated. However, they can be used to write generic methods, i.e., methods that can handle all classes that implement the interface.

Development tools

Software developers use a variety of tools that support all parts of the development process, e.g., coding, refactoring, code versioning, debugging and profiling. Today, the central development tool in most projects is an *integrated development environment* (IDE). Modern IDEs are very powerful tools which facilitate coding through syntax highlighting, code auto-completion, code folding and quick navigation in the code. Most IDEs also offer refactoring tools, debugging tools and integration for code version systems. Widely used IDEs for C++ development are *Microsoft Visual Studio* [51] and *Eclipse* [52]
Versioning systems, for example Subversion [38], are a very important software development tool. The principle is that the project source code (and other versioned documents) is managed in a central repository. Developers can check out a working copy from the repository, work on it and check in the changes they made. Besides being a repository of previous code versions, versioning systems support the development process in various ways. They provide a means to review and undo code changes. They allow parallel work of several developers on the same file through file locks, merging of changes to the same file and conflict resolution tools. When integrated with the requirements management system, they allow traceability of requirements to code changes. When integrated with the test system, they allow automated test builds after code commits. Another important feature they provide is tagging and branching. Tagging simply labels a certain version of the code with a unique name, for example a release version. Branching creates a copy of the main development directory tree (called trunk) to a so-called branch. This allows temporary independent development of two or more versions of the project. Changes can be merged from the trunk to the branch and vice versa. Branching is often used to develop several features independently in different branches or to allow development of

new features while preparing a release version of a software.

2.3.5. Testing

Because of the inherent complexity even of small software projects, testing is indispensable. Software testing ensures that the software behaves as specified and fulfills the user needs. How tests are implemented and in which phase of the project testing starts mainly depends on the software development process used (see Section 2.3.1).
There are two fundamental approaches to testing. *Dynamic tests* execute program code and check that the output is as expected. *Static tests* analyze the source code without executing it. The largest part of static testing is performed by compilers and interpreters, which check the syntactic correctness of source code. Other aspects of static testing are code reviews and applying code metrics.

In this section we will focus on dynamic testing. First, different ways to classify tests are explained. Then, test automation is motivated and the tools for test automation are shortly summarized. For more information on the theory and practice of testing refer to [53].

Classification of tests

There are many different types of tests and several classification schemes for tests. One common way to classify tests is according to scope: *Unit tests* validate code sections of limited size. In object-oriented programming a unit is typically a class, which is why unit tests are often referred to as *class tests* in this context. Interactions between classes or larger subsystems of a software are tested in so-called *integration tests*. Tests of the whole software system are called *system tests*.
Another common classification criterion for software tests is purpose: *Correctness tests* validate that software behaves as specified. This kind of testing is widely adopted in software development. *Performance tests* ensure that the software does not use too many hardware resources such as CPU cycles, memory and file system operations. *Reliability tests* evaluate the ability of a software to cope with stressful environment conditions, e.g., exceptional input and resource shortage. *Security tests* try to reveal security problems of the software introduced by design or implementation defects.
Independent of the scope and purpose, each test can be classified as *white-box test* or *black-box test*. This classification depends on the view taken while designing a test. In the black-box approach the actual implementation is not taken into account while designing a test. This implies that only the public API is tested. In *white-box testing* the tester has access to internal data structures and functions, and to the source code. Both approaches have certain advantages and disadvantages: One advantage of black-box testing is that implementation and testing can be performed by different persons. This can speed up development significantly and often reveals more errors than developer-testing. Another advantage is that tests can be designed before the actual implementation of the functionality (test-driven development). The big advantage of white-box testing is that critical code paths can be easily identified and tested thoroughly. This improves the code coverage of tests. A middle way between black- and white-box testing is *gray-box testing*, where the tester has access to the source code, but testing is performed on the public API only.

Test automation and tools

A recommended practice in the context of testing is to use *continuous integration*. It demands that code changes are committed to the code repository early—when a single feature is implemented and tested. To validate these atomic code commits, automated builds and automated execution of tests are performed regularly. Building and testing can be scheduled once or several times a day, or triggered by commits to the source code repository. These practices ensure that errors (files accidentally not committed, bugs, etc.) are detected early and can be assigned to a specific commit, which corresponds to a specific feature.

Many software tools have been developed to support these practices. Test frameworks such as CTest [54] facilitate the implementation of unit tests, manage unit tests and generate test result reports. Automation tools, e.g., CMake [54] and *Ant* [55], allow automated building of software and automated execution of tests. The results of the automated tests can be visualized and evaluated using test servers such as CDash [56] or Bamboo [57].

2.3.6. Deployment and maintenance

Software deployment summarizes all activities required to make the software product available to users. It includes assembly, packaging, distribution, installation and updating of the software. Because each software product uses different techniques and tools for these activities, a universal description of software deployment cannot be given. It is a highly customized process.

Software maintenance is a discipline that deals with changes to software after it has been released. The goals of the changes are to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment [58]. Of these goals, correcting defects and improving usability are most important. Maintenance of software, just as deployment, is highly dependent on the software product and the context it is used in. More information can be found in [59].

3. A novel feature detection algorithm for centroided data

Peptide identification and quantitation are the two most important steps in LC-MS data analysis pipelines [60]. For identification of MS/MS spectra, several reliable software tools are established and widely used [61]. The field of quantitation is still evolving, although many different algorithms have been published [62]. The reason for this difference in maturity is the complexity of quantitation. Identification algorithms typically work on simple centroided MS/MS spectra, while quantitation algorithms work on profile data (see Section 2.1.2). Profile data is not only highly dependent on the MS instrument type, but it also contains many signals caused by chemical noise (ions derived from compounds other than the target analyte [63]) and white noise (random noise generated by measurement instruments).

In the last few years the resolution of mass spectrometers has significantly improved, providing isotope-resolved spectra even for multiply-charged ions. The improved resolution of the data, combined with modern peak detection techniques [64] allows quantitation based on centroided data. Because centroided data is several orders of magnitude smaller than profile data, more sophisticated algorithms can be applied, increasing the specificity especially on noisy data. Another advantage of quantitation based on centroided data is that it can be used on datasets where no profile data is available.

3.1. State of the art

There are many different quantitation algorithms for mass spectrometry data available. We will now discuss the common features and differences of some recently published algorithms: msInspect [12], MaxQuant [16], MapQuant [65], SuperHirn [15], SpecArray [13], MZmine [14, 66], LCMS2D [67] and FeatureFinder [68]. FeatureFinder offers several quantitation algorithms, of which we will consider the *simple* algorithm [69] and the *isotope_wavelet* algorithm [70, 71].

Input data: All of the listed tools take profile mass spectrometry data as input. As a first processing step, smoothing, baseline-reduction and centroiding is applied by most of the tools. However, combining several data processing steps into a single tool severely limits the control over the data processing. The user is forced to use the built-in signal processing and centroiding, even if other algorithms are more suitable for the data. Only *SpecArray* and *MZmine* offer processing of centroided data in addition to profile data.

The second important property of the input data is the resolution. Recent mass spectrometers can achieve a very high mass resolution with an error of less that one ppm. Thus, recent algorithms, for example *SuperHirn* and *MaxQuant*, are designed for high-resolution data only. In contrast, *MapQuant* and *FeatureFinder (simple)* are suitable for low-resolution and medium-resolution data only. The rest of the tools presented here work both for medium-resolution and high-resolution data.

Usability: Another important aspect, which has nothing to do with the algorithms themselves, is availability and usability. All tools can be downloaded from the locations specified in the respective publications. However, not all of them provide a binary installer. The source code packages of *MapQuant*, *SuperHirn* and *LCMS2D* could not be compiled without changes to the source code or build system. This makes them unusable for most potential users.
Another common problem is the required input file format. *MaxQuant*, *MapQuant* and *LCMS2D* do not support any of the community standard XML formats. They require proprietary vendor formats or ASCII text files as input, which complicates the use of these tools.

Algorithmic approaches: Most quantitation algorithms consist of two or three basic steps. First, interesting regions in the data are determined, which probably contain a feature. This step is often referred to as *seeding*. In a second step, all peaks contributing to the feature and its charge state are determined. This step is referred to as *extension*. Some tools perform a third step, in which a theoretical peptide feature model is fitted to the data in order to remove false positive features. This step improves the specificity of the algorithm.

3.1.1. Seeding

In the *seeding* step, different approaches are used by the algorithms. The most widely used approach is *intensity descent* [72]. Here, the peaks are used as seeds in the order of descending intensity. First, the highest peak is used, then the second highest peak is used, and so on. Many algorithms (*SpecArray*, *MZmine*, *FeatureFinder (simple)* and *LCMS2D*) use this approach and combine it with different stop conditions based for example on an absolute intensity cutoff or on a signal-to-noise cutoff. *MaxQuant* and *msInspect* implement a variant of *intensity descent*, which identifies whole elution peaks (also called mass traces or chromatographic peaks) instead of single peaks.

The *MapQuant* tool uses a completely different approach based on the image processing algorithm *watershed segmentation*. Just like in *MaxQuant* and *msInspect*, the watershed segmentation is used to detect single mass traces.

The most sophisticated *seeding* strategy is used by the *FeatureFinder (isotope_wavelet)*. A hand-tailored *isotope wavelet* [73] is used to detect isotope pattern in each spectrum. The isotope wavelet is based on the *averagine* [17] model, which approximates a peptide isotope pattern by assuming an average isotope distribution. This approach is characterized by a very high specificity.

3.1.2. Extension

In the *extension* step the seed peak or seed region is extended until all signals belonging to the peptide feature are contained. Here, most algorithms use heuristics to detect mass traces—often the same m/z value has to be observed in a number of subsequent spectra. Most algorithms combine several mass traces to a feature if their m/z distances match a specific charge and if their intensities follow the *averagine* distribution.

Because *MaxQuant*, *MapQuant* and *msInspect* have already determined the mass traces during the seeding phase, they use slightly different techniques during the extension. *MapQuant* and *msInspect* use clustering approaches, where the theoretical isotope distribution is used in the similarity score. *MaxQuant* transforms the mass traces to a graph—the nodes represent mass traces, edges are drawn between nodes if their m/z

distance and intensity match the averagine model. Graph algorithms can then efficiently determine those subgraphs that correspond to one feature.

The *FeatureFinder (isotope_wavelet)* determines isotope patterns in single spectra during the seeding phase. The extension is in this case a very simple task. Isotope patterns that are present in several subsequent spectra are merged to a feature.

The *FeatureFinder (simple)* maintains a priority queue containing all peaks within a small boundary around the feature region to extend the feature. In each extension step the peak with the highest priority, which depends on the peak intensity and its distance from the feature seed, is added. To restrict the extension, configurable cutoffs are used in all three dimensions (intensity, m/z and RT).

3.1.3. Model fitting

The *MapQuant* and *FeatureFinder (simple)* algorithms implement an additional model fitting step. In this step, the plausibility of the detected feature is checked. In both tools a two-dimensional model is fitted to the data, which models the isotope pattern by an averagine model and the elution profile by an exponentially modified Gaussian (EMG) [74]. The EMG function is a Gaussian function with an additional skewness parameter. It allows for asymmetric elution profiles, which are not uncommon in chromatography.

Using model fitting as a sanity check for each features increases the specificity of the algorithms. Because most algorithms use an iterative approach (e.g., intensity descent) a higher specificity helps to avoid subsequent errors caused by false positive features.

3.2. Our contribution

In this chapter we present a novel peptide feature quantitation algorithm for centroided LC-MS data. The development goals of the algorithm were:

Usability: The algorithm should not be integrated in a fixed analysis pipeline, and should not depend on identification data as start points. It should be a general-purpose algorithm for use in different quantitation scenarios, e.g., for isotope-labeled and label-free data.
To make the algorithm widely usable, it was integrated in the C++ library OpenMS (see Chapter 5) and provided as a stand-alone tool of TOPP (see Section 5.9).

Wide applicability: The algorithm should be widely applicable to medium-resolution and high-resolution data of different MS instruments. The only requirement is that isotope-patterns are resolved to individual peaks. This implies that it is applicable to centroided data and profile data with prior centroiding.

Robustness: Because it relies on centroided data, the algorithm must be robust with respect to centroiding errors, e.g., missing peaks and misplaced peak m/z positions. It should also be robust to chemical noise and white noise present in the data. Additionally, it should be able to resolve features overlapping in RT, m/z or both dimensions.

3.3. Design and implementation

This section first discusses the overall design of the algorithm and the purpose of the different phases. Then, details of the individual phases are described.

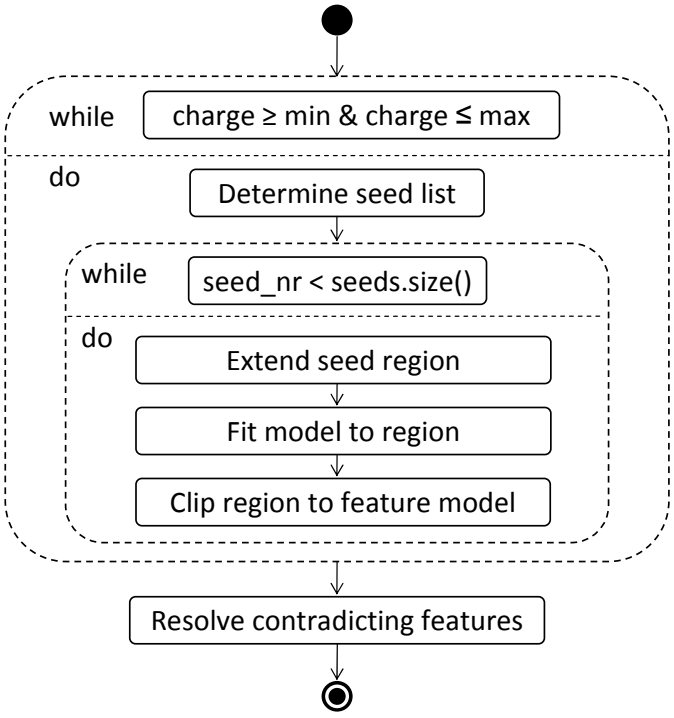


Figure 3.1.: UML activity diagram of the overall algorithm structure.

3.3.1. Overall design

Iteration strategy

Most quantitation algorithms use an iterative approach. They start with the highest data point and determine the corresponding feature. The feature peaks are then subtracted from the data or flagged as used. This procedure is repeated until an end criterion is reached, e.g., no peak above a certain signal-to-noise ratio is left.
In our experience this works well only on rather simple data. On complex datasets with many overlapping features, this approach often fails to detect all features. The problem is that subtracting feature peaks from the data corrupts the data if the feature charge, the feature borders or the overall shape are not determined correctly.

Thus, our algorithm uses a different approach. Putative feature centroid positions are maintained in a so-called *seed* list. Each seed is the start point for feature detection once. When a feature is found, it is stored in a *feature candidate* list. After all features are detected, contradicting features are removed from the candidate list using a greedy algorithm.
This approach does not manipulate the data or exclude data from subsequent processing steps. It makes sure that all features can be detected, regardless of errors in previous feature detection steps.

Feature detection strategy

The actual feature detection consists of three phases. First, a *seed* is extended to a feature region, which contains the feature and surrounding data points. In the second phase, a model based on mass, charge and an elution profile is fitted to the feature region. If the model does not fit to the data, feature detection is aborted. In the third

phase, data points not belonging to the feature are clipped and the final feature quality is determined.

A high-level overview of the algorithm phases is given in Fig. 3.1.

3.3.2. Seeding phase

The seeding phase determines putative feature centroids, which are used as start points for the extension and model fitting. Fast and simple heuristics are used in this phase, because the entire input data has to be inspected. Each seed has to have a high signal-to-noise ratio, a mass trace in RT dimension and an isotope pattern in m/z dimension. The final *seed score* of each peak combines the individual scores for these three properties.

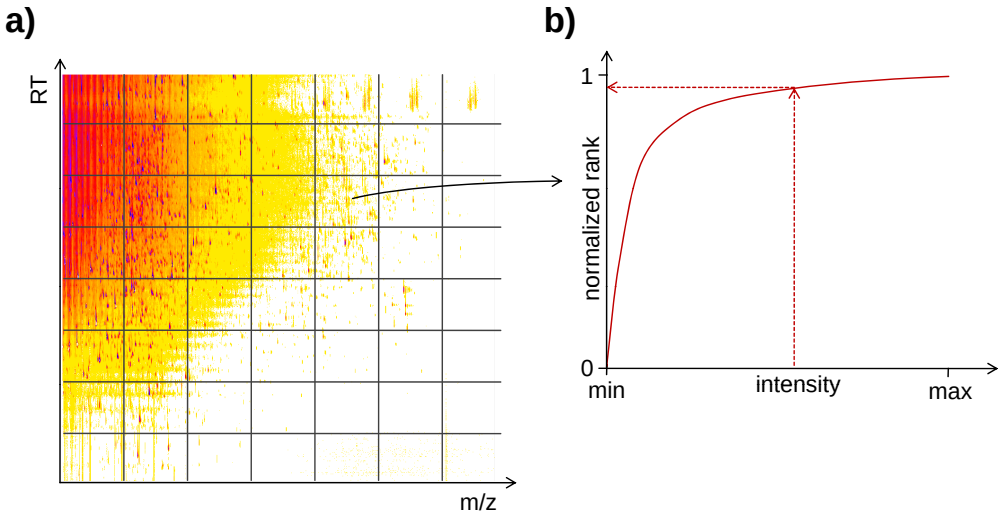


Figure 3.2.: Calculation of the intensity score during the seeding phase. a) The peak map is segmented into tiles. b) Intensities are transformed to an intensity score based on the rank of the intensity in the tile.

Intensity score

The signal-to-noise value of a peak is a widely used score to determine the significance of a peak in its local environment [75]. Unfortunately, signal-to-noise is only defined for profile data. Thus, we use a heuristic to estimate the significance of a peak. The local environment of a peak is determined by segmenting the peak map in n times n tiles. Unlike in most signal-to-noise scores, neighboring peaks in both m/z dimension and RT dimension are considered.

To determine the significance of a peak in its tile, the normalized intensity rank of the peak in the tile is calculated. The peak with the highest intensity gets a score of 1. The peak with the lowest intensity gets a score of 0.

This rank score can be calculated exactly when using the sorted intensity values of all peaks in the tile. This would however require a lot of memory, because the intensity distributions of all tiles need to be stored. To minimize the memory consumption, 20-quantile (vigintile) values of the intensity distribution are stored and used to approximate the rank of each peak. To do so, the vigintile values surrounding the peak intensity are

determined and the final peak score is interpolated according to the formula

$$intensity\_score = \frac{1}{20} * [index(vigintile_{low}) + \frac{peak\_intensity - vigintile_{low}}{vigintile_{high} - vigintile_{low}}],$$

where $index(...)$ returns the vigintile number, $vigintile_{low}$ is the lower bound vigintile and $vigintile_{high}$ is the upper bound vigintile. Fig. 3.2 shows an overview of the intensity score calculation.
The 20-quantile values have been chosen after thorough testing. In our experiments, they provided a good approximation of the overall intensity distribution. Increasing the number of quantiles did not improve the accuracy of the score.

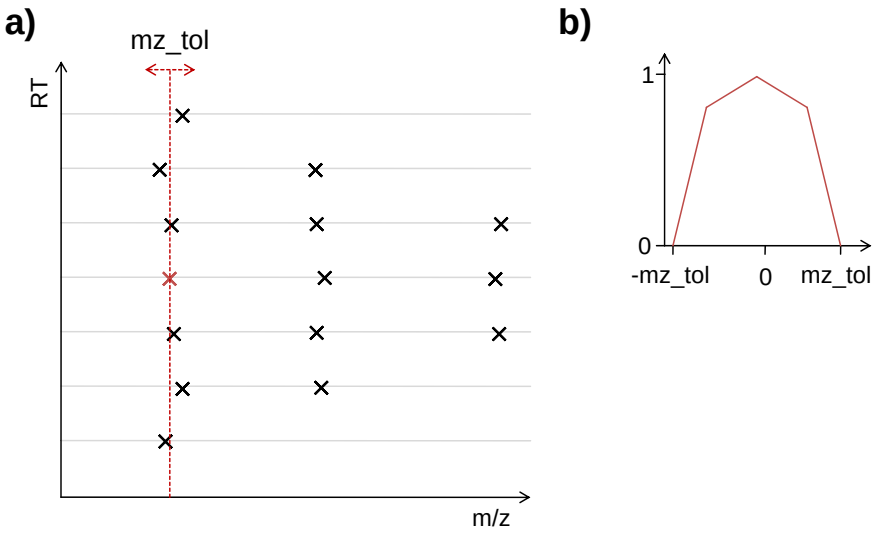


Figure 3.3.: Calculation of the mass trace score during the seeding phase. a) Peak positions in adjacent spectra are determined. b) m/z deviations are penalized using a linear approximation of a Gaussian.

Mass trace score

The mass trace score measures whether the m/z position of a peak occurs in several adjacent spectra. The number of required spectra depends on the dataset and can be set via the input parameter *min_spectra*. The mass trace score is calculated as the average of *position tolerance scores* for all spectra inside a window of *min_spectra* spectra around the central peak.
The *position tolerance score* takes two m/z positions and returns a similarity score. The allowed m/z deviation depends on the instrument, and thus can be set as a parameter. The score is calculated based on the central peak m/z and the nearest peak m/z in another spectrum. The score should tolerate small deviations between peak m/z positions and penalize large deviations. To achieve this, a linear approximation of a Gaussian function is used to calculate the score. The exact Gaussian cannot be used here because of performance issues—it is too slow when evaluated several times for each peak. Alternatively, a tabulated Gaussian could be used. Fig. 3.3 shows an overview of the mass trace score calculation.

In addition to the mass trace score, another peak property is calculated in this phase. Each peak is flagged as a local maximum peak, if no other peak in the mass trace has a higher intensity value.

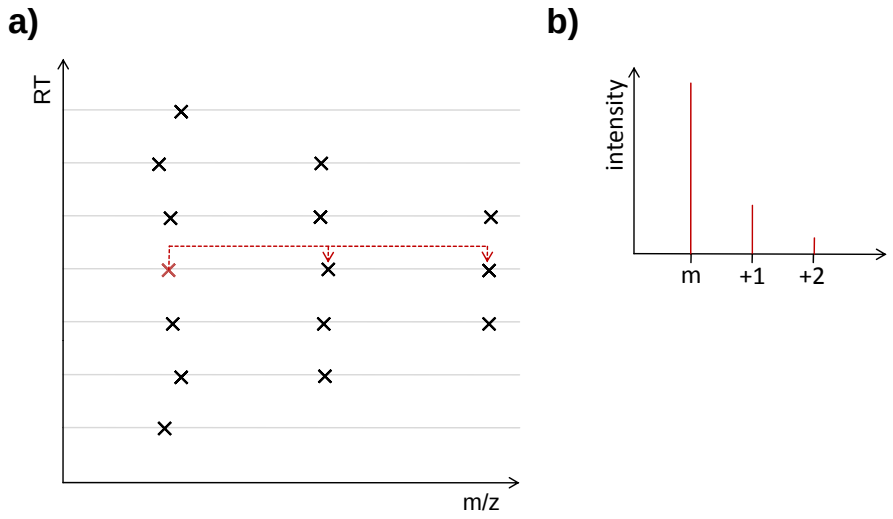


Figure 3.4.: Calculation of the isotope pattern score during the seeding phase. a) Peaks with the correct m/z distance to the central peak are determined. b) The observed peak intensities are compared to an averagine isotope pattern. Here a monoisotopic peak and two isotope peaks are shown.

Isotope pattern score

The isotope pattern score measures whether a peak is part of a peptide-like isotope pattern. Because amino-acid sequences of the features are unknown, the averagine [17] model is used to approximate the isotope pattern of peptides with a given mass.
Starting from the seed peak, peaks are searched which have an m/z distance of one over the current charge. A *position tolerance score* is calculated just as for the mass traces score.
Next, the maximum correlation of the averagine isotope distribution with the peaks found in the data has to be determined. To do so, several hypotheses are tested—the central peak is shifted to the different isotopes of the theoretical isotope pattern and the maximum achieved correlation is stored.
The final isotope pattern score is calculated as the product of the averaged *position tolerance score* and the maximum correlation to the theoretical isotope pattern. Fig. 3.4 shows an overview of the isotope pattern score calculation.

The isotope pattern score is the only score that depends on the charge state. Thus, it is recalculated for each possible charge state. The intensity score and the mass trace score are precalculated and used for all charge states.

Overall score calculation

The final *seed score* of each peak is calculated as the geometric mean of the three individual scores for intensity, mass trace and isotope pattern. The geometric mean was chosen because it strongly penalizes low values of a single score—a good score for

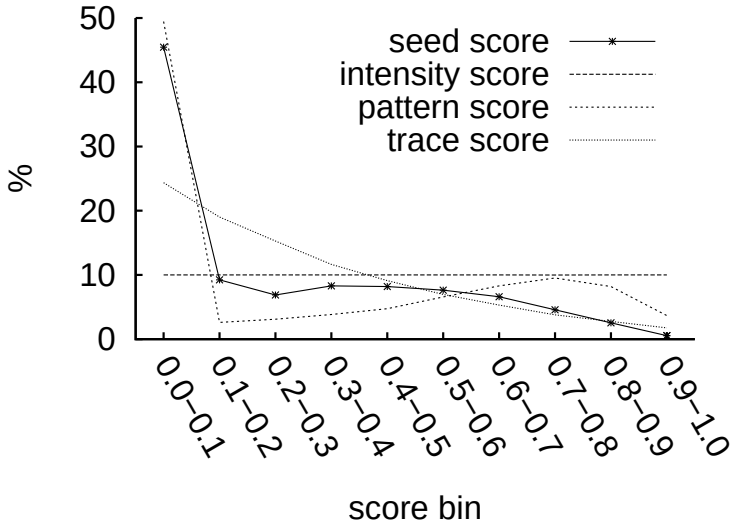


Figure 3.5.: Histogram of typical distributions (seed score and the three base scores).

each criterion is expected for true seeds. The seed score ranges from zero (worst) to one (best), just as the three contributing scores.
Fig. 3.5 shows an example of typical score distributions for reasonable parameter settings. The *intensity score* is evenly distributed because it is calculated as the normalized rank of the peak intensity. The *seed score*, *trace score* and *pattern score* produce far more low scores than high scores and, thus, allow a better discrimination of good and bad seed candidates.

Seed list creation

The seeds list for a charge state is created by looking up all peaks with a *seed score* above a given threshold (default is 0.8). Additionally, the seed has to be a local maximum of a mass trace. This makes sure that only one peak per mass trace is considered as a seed. The list of seeds is sorted with respect to seed intensity. Thus, the seeds with the highest intensity are extended first.

3.3.3. Extension phase

Each seed determined in the *seeding phase* is extended to a *feature region*, which contains the feature peaks and a number of surrounding peaks. After the extension, a feature model is fitted to the determined region.

Extension in m/z dimension

In the first step of the extension phase, the best averagine [17] isotope pattern containing the seed peak is determined. This is done similarly to the isotope pattern score calculation during the seeding phase. Each isotope peak of the theoretical isotope pattern is aligned to the seed. The quality of fit is determined by the deviation in m/z dimension and the correlation to the averagine model intensities. The best matching isotope pattern is used for the following steps.
Because peak detection does not always work reliably one or several isotope peaks might

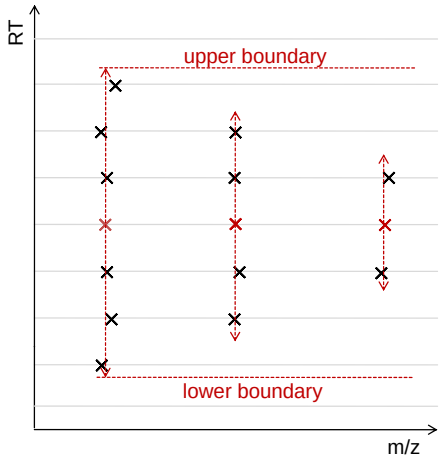


Figure 3.6.: Extension of isotope pattern mass traces. The isotope with the highest intensity (left) is extended first. The boundaries in RT dimension are used as absolute boundaries for the extension of the two other isotope traces.

be missing. Missing isotope peaks can be compensated for by looking up peaks with a similar mass in directly adjacent spectra.

Extension in RT dimension

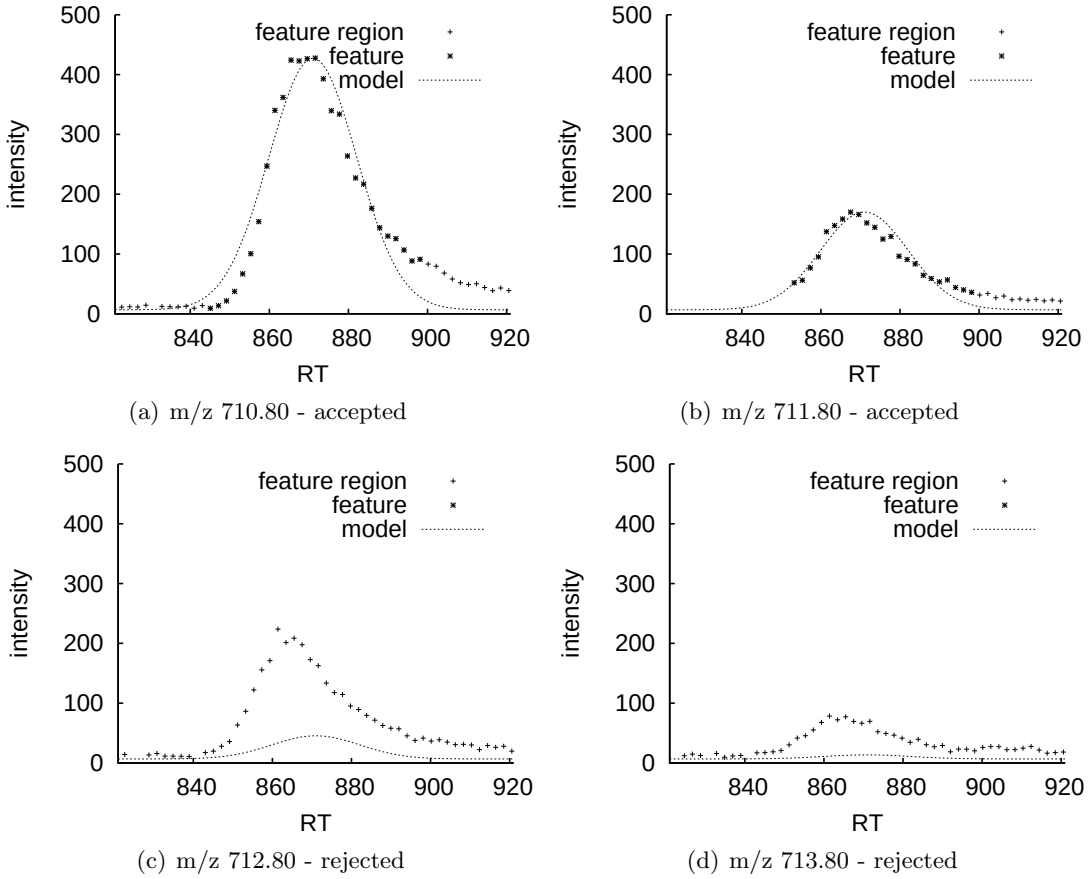
In the second step of the extension phase, the elution profiles of all isotope peaks are extended. The extension in increasing RT and decreasing RT are done independently. The extension stops if one of the following criteria is fulfilled:

- *Mass trace ends:* No peak within an m/z tolerance around the isotope position can be found in several subsequent spectra.
- *Noise peaks:* Peaks with a matching m/z position can be found, but they have a *seed score* close to zero—they are probably noise peaks.
- *Rising average intensity:* The moving average intensity over three peaks rises more than a given threshold (10% is the default). This criterion is needed to abort the extension in the case of overlapping mass traces of different features.
- *Boundary reached:* The mass trace of the most abundant isotope peak is extended first. The retention time interval of this mass trace is used as an overall bounding box for extension of all remaining mass traces (see Fig. 3.6). This criterion is needed to restrict the extension of low intensity isotope traces in regions with many noise peaks. Otherwise very large feature regions could be created, which would slow down the subsequent model fitting step.

Fig. 3.6 shows an example extension in RT dimension.

3.3.4. Model fitting phase

The feature region determined in the *extension phase* is fitted to a peptide feature model. The model is based on a Gaussian elution profile [74] in RT dimension and the averagine model in m/z dimension. All mass traces are fitted to the model simultaneously using the Levenberg-Marquardt [76] algorithm implementation of the GNU Scientific Library (GSL) [77].



(a) m/z 710.80 - accepted

(b) m/z 711.80 - accepted

(c) m/z 712.80 - rejected

(d) m/z 713.80 - rejected

Figure 3.7.: Example of a feature model fitted to four mass traces and of feature clipping (see Section 3.3.5). The first two mass traces show a good fit. The other mass traces are rejected because of large intensity deviations. They belong to an overlapping feature with a nearly identical elution profile. The rejected traces can be detected as a feature in a later extension step.

The model fitting optimizes the three parameters of the Gaussian distribution:

$$f(x) = Ae^{-\frac{(x-x_0)^2}{2\sigma^2}}$$

The parameter A determines the height of the Gaussian, σ determines the width in RT dimension and x_0 determines the center in RT dimension. Because all mass traces of an isotope pattern are fitted simultaneously, a scaling factor is needed for parameter A. It is set for each mass trace according to the averagine intensity distribution.

The Gaussian elution profile model does not always exactly fit the data, as can be seen in Fig. 3.7. Especially high-intensity features often show an elution profile with tailing. The use of a model with tailing, for example an exponentially modified Gaussian (EMG) [74] model, could improve the fit. The pros and contras of the EMG model are discussed in the outlook (3.6).

Estimation of fit start parameters

The start parameters of the Levenberg-Marquardt fit have a large influence on the success. Start parameters far from the optimum solution can lead to slow convergence or

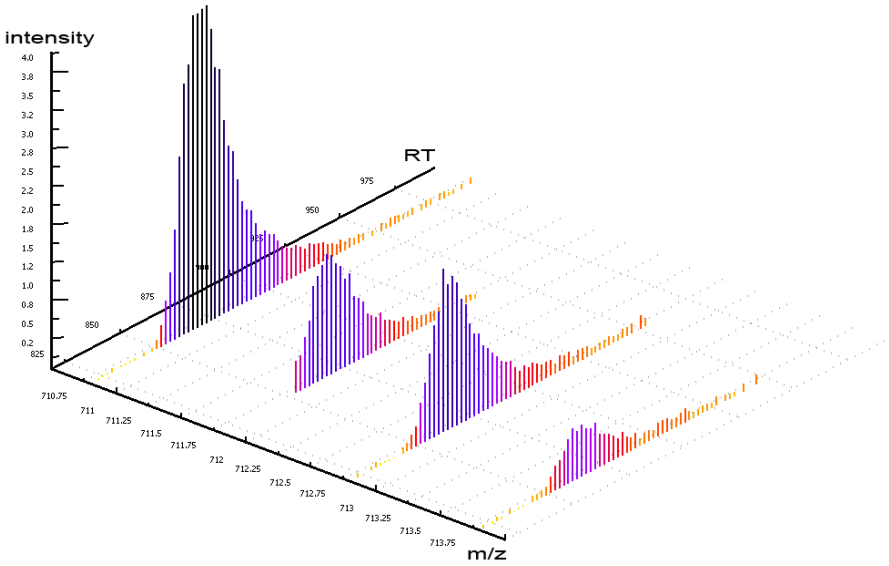


Figure 3.8.: Example of two overlapping features. The monoisotopic peak of the first features is at 710.80 Th. The monoisotopic peak of the second features is at 712.80 Th. The elution profiles of the two features show a very high correlation.

can make the algorithm find a local minimum instead of the global minimum.
The parameters A and x_0 are easy to estimate. They are set to the intensity and RT position of the most abundant peak, respectively. These estimates are not far from the optimal value in most cases.
σ is estimated as a tenth of the *feature region* width in RT dimension, which underestimates the optimal value roughly by a factor of two. Our tests showed that overestimating σ often leads to a bad model fit. Underestimating σ has rarely any effect on the fit. Thus, underestimating the parameter is preferable—it also corrects for too large feature regions determined in the extension phase.

3.3.5. Feature clipping phase

The *extension phase* often determines a too large feature region, especially on very noisy data where the ends of the mass traces cannot be easily determined. Thus, the feature region has to be clipped after the model fit.

In RT dimension, the feature is restricted to an interval of 2.5 σ around the x_0 position determined by fitting the Gaussian model. This interval should include more than 98% of the feature intensity. In m/z dimension, the consistency of each mass trace with the model is checked. First, the correlation between model intensity and data intensity is calculated. However, the correlation alone is not enough to separate features with a similar elution profile and overlapping mass traces. Thus, the average relative deviation between model intensity and data intensity is incorporated. The final score used to assess the match between model and data is calculated as the product of the intensity correlation and 1 minus the average relative deviation. Mass traces with a score below a given threshold (default is 0.5) are removed from the features. Fig. 3.7 shows an example of the clipping phase. In the 3D visualization of the same region (Fig. 3.8) it is easy to see that the monoisotopic peak of the smaller feature overlaps with the +2 peak of the

larger feature.

Feature creation

After clipping the feature according to the model fit, the overall feature quality is calculated based on the remaining data points. Just like the quality of mass traces, the overall quality score is calculated from the intensity correlation and the average relative deviation of intensities. Features below a given quality threshold (default is 0.7) are discarded. Features above the quality threshold are added to the *feature candidate* list.

For each feature candidate intensity, RT position and m/z position are reported along with some meta data. The feature intensity is calculated based on the area under the Gaussian model. This approach was chosen because it is more robust to noise than summing up the data point intensities. It also compensates for missing mass traces, e.g., in the case of overlapping features.
The user can choose from two different options for the reported m/z position—either average m/z or monoisotopic m/z. The average m/z is simply calculated as the intensity-weighted average m/z position of all peaks. The monoisotopic m/z is calculated by shifting the intensity-weighted average m/z position of the highest mass trace peaks to the monoisotopic peak, based on the averagine model of the feature.
Additionally, meta information about the data and the model fit is reported, e.g., the convex hull of each mass trace and the overall feature quality.

All *seeds* that lie inside the feature candidate area are removed from the seed list and not used for extension. This should prevent finding the same feature several times.

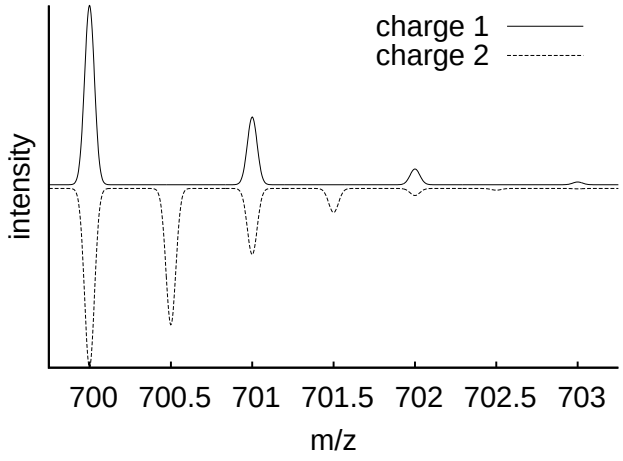


Figure 3.9.: Example of strong similarity between isotope patterns. The monoisotopic peak of both isotope patterns is at 700 Th. The charge 1 isotope pattern fits very well on the charge 2 distribution (correlation greater than 0.99). In this case the charge two feature would be preserved because it explains all peaks.

3.3.6. Conflict resolution

Our iteration strategy (see Section 3.3.1) and the independent processing of all considered charge states, can lead to the detection of multiple feature candidates for one signal in the input data. The goal of the conflict resolution step it to detect these contradicting

features and resolve the contradiction by removing all but one of the features.
Two features are considered to be conflicting, if the overlapping area of the features is higher than a given cutoff (default is 35%). The overlapping area of two features is determined by intersecting the mass trace convex hulls of the features. Small overlaps have to be tolerated—they are caused by true overlapping features, e.g., in RT dimension.

Three different cases have to be considered for features with large overlaps:
(1) Overlapping features with the *same charge* occur due to incomplete feature detection. The same feature is found twice originating from two different seeds. In most cases the features are nearly identical or differ in the number of mass traces only. The feature with the highest score (product of intensity and quality) is preserved. The feature with the lower score is removed from the feature candidate list.
(2) In certain mass-to-charge ranges, isotope patterns of different charge states are very similar. For example, the isotope pattern of a singly charged peptide of mass 699 fits well to an isotope pattern with mass 1398 and charge 2 (see Fig. 3.9). This can only occur if one charge state is a multiple of the other. Otherwise the m/z differences between mass traces would not be correct. This case is resolved by removing the lower charge feature, because the higher charge feature explains all mass traces.
(3) For all other conflicts, e.g., between charge state 2 and 3, a greedy strategy is used which preserves the feature with the highest quality.

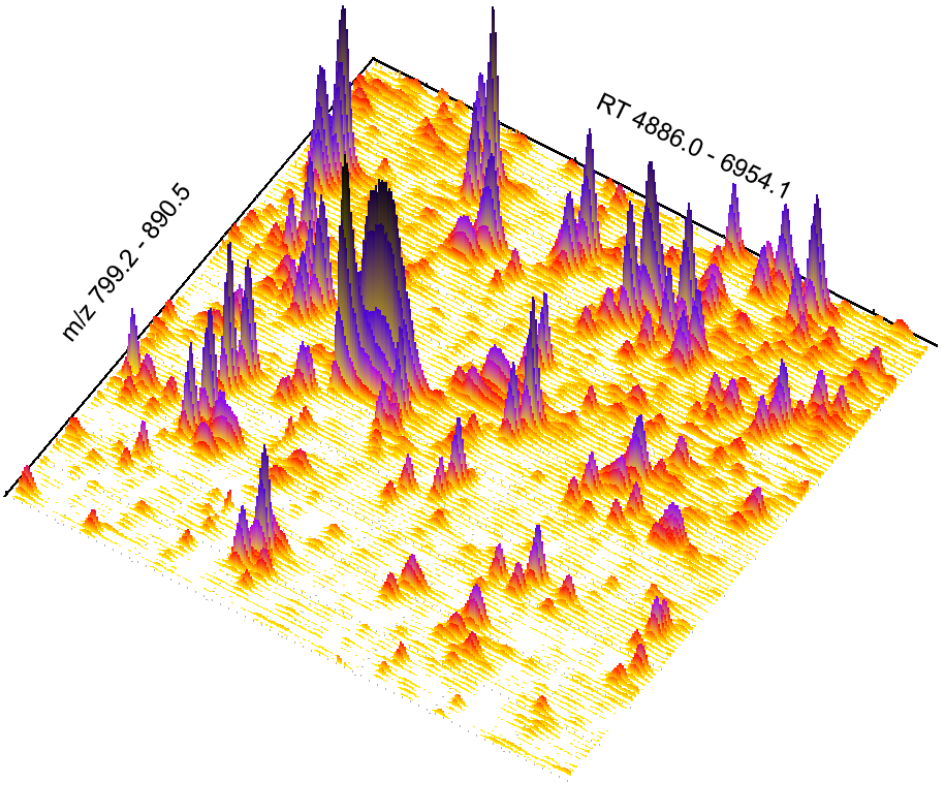


Figure 3.10.: Snapshot of the m2_r2 region (taken in TOPPView).

3.4. Results and discussion

Assessing the performance of quantitation algorithms is difficult because there are no standard datasets with annotated features available. Artificial datasets can be created, e.g., with the tool LC-MSsim [78]. It is, however, unclear if an artificial dataset shows the same characteristics as experimental data. Thus, we decided to use experimental data for this evaluation.

Many studies use experimental datasets with only a few proteins as benchmark dataset. However, the performance of quantitation algorithms on a simple dataset is in general much better than on highly complex datasets. Thus, only the maximum possible performance can be assessed when using low-complexity datasets.

Other studies use very complex datasets, but evaluate the performance based on a small number of peptide features only. Although this method gives a better impression of the algorithm performance on complex data, it is prone to errors as well. For a sound performance analysis of quantitation algorithms, a statistically significant number of peptide features from a complex sample has to be evaluated.

3.4.1. Test datasets

For the evaluation of the feature detection, we used several LC-MS maps from differential immuno-peptidomics assays. The focus of the assays were peptides bound to MHC molecules extracted from renal cell carcinoma cell lines. The goal was to compare presented MHC peptides after treatment with anticancer drugs to peptides presented without treatment. The two peptide samples were labeled with a stable-isotope nicotinic acid label (mass difference of 4 Da) and combined before they were analyzed in one LC-MS run on Waters Q-TOF instruments with ESI sources. MS survey scans were recorded in the m/z range of 400 Th to 1000 Th, where most of the MHC peptides are expected. The gradient duration of the HPLC system was between one and two hours. More details about the experimental setup can be found in [79, 80].

Because the LC-MS maps contain too many features to manually annotate all of them, two regions with different properties (m/z range, peak density, noise level, feature charge) were selected in each map, each containing several hundred features. For those regions, a gold standard feature list was manually created using the GUI tool *TOPPView*. The data was visually inspected by an expert and feature centroid positions and charge states of all features were annotated.

The datasets were selected specifically to be very challenging for the analysis algorithms. They contain a lot of noise, both chemical and white noise, and many overlapping features (see Fig. 3.10). This complexity also made the manual annotation difficult, especially for low-intensity features. Features for which the charge state or centroid position could not be determined unambiguously, were not annotated. Thus, a high number of false positive features with low intensities can be expected in the evaluation.

The two datasets described above do not contain annotated feature intensity values because the feature area cannot by easily determined by hand. To evaluate feature intensities, a third dataset from a similar study was used for which hand-annotated intensity values were available for 300 features. This dataset was not split into smaller regions because the annotated features were spread over the whole dataset.

Tables 3.1 and 3.2 give an overview of the peak datasets and the manually created gold standard datasets. The datasets are labeled $m1$, $m2$ and $m3$. The two regions selected from datasets $m1$ and $m2$ are denoted by appending $r1$ or $r2$ to the dataset label. The dataset with annotated intensity values is $m3$.

All three datasets were acquired as profile peak data. To obtain centroided data, the *PeakPicker* tool of *The OpenMS Proteomics Pipeline* [68] was used. Only the *peak_width* parameter (full width at half maximum) of the tool had to be adapted. A value of 0.08 Th was used for the dataset $m1$ and 0.1 Th was used for datasets $m2$ and $m3$.

dataset	profile data spacing	size [MiB]	spectra	min RT [sec]	max RT [sec]	min m/z [Th]	max m/z [Th]
m1_r1	0.012	19.8	799	3002.2	4628.8	1299.0	1400.9
m1_r2	0.012	13.4	619	2745.2	4005.1	699.5	758.9
m2_r1	0.031	6.9	369	4886.0	6954.1	799.2	890.5
m2_r2	0.031	5.0	332	3342.2	5200.6	611.7	669.7
m3	0.031	107.0	3639	3.1	12001.7	399.8	999.5

Table 3.1.: Overview of centroided test datasets.

dataset	features	charge				
		1	2	3	4	5
m1_r1	116	6	5	71	33	1
m1_r2	131	4	61	55	10	1
m2_r1	133	106	25	2	-	-
m2_r2	222	2	216	-	4	-
m3	300	-	288	12	-	-

Table 3.2.: Overview of the manually created gold standard datasets.

3.4.2. Parameter selection

Our algorithm has 10 base parameters. Additionally, 12 advanced parameters are available. The advanced parameters are intended for internal use only and should not be changed. Only a hand full of the base parameters has to be adapted to the dataset: (1) The charge state range of interest should be known and easy to set. (2) The m/z tolerances for mass traces and isotope patterns depend on the instrument and the peak picking accuracy. Visual inspection of several features in the peak data should give a good estimate of the expected deviations. (3) Finally, the expected number of data points per elution profile and the maximum number of missing data points has to be set. These parameters depend on the chromatography settings and on peak picking. Again visual inspection can quickly reveal the right settings.

The quality score cutoffs, e.g., seed score and feature quality, usually need no tuning. They are universal because the scores are scaled to the interval between 0 and 1. Table 3.3 shows the base parameters used in this evaluation. Default values were used for all advanced parameters, so they are not listed here.

3.4.3. Performance on the test data

Feature positions

To assess the performance of our algorithm we ran it on the five test datasets. The two regions of the same LC-MS map were processed using the same parameters. Precision

parameter	m1	m2	m3
intensity:bins	1	1	10
mass_trace:mz_tolerance	0.05	0.05	0.02
mass_trace:min_spectra	10	8	10
mass_trace:max_missing	2	1	2
isotopic_pattern:charge_low	1	1	2
isotopic_pattern:charge_high	5	4	3
isotopic_pattern:mz_tolerance	0.05	0.05	0.04
seed:min_score	0.8	0.8	0.8
feature:min_score	0.7	0.7	0.7
feature:reported_mz	maximum	maximum	maximum

Table 3.3.: Parameters used for the evaluation.

and recall were used to measure the performance:

$$precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Features are considered to be correctly identified (true positives), if they match the manually determined m/z position, RT position and charge. The m/z position was counted as a match, if the deviation from the true feature was not higher than 0.25 Da divided by the charge state. The RT position was counted as a match if the deviation was below 15 seconds, which corresponds to roughly 25% of the average elution time span of peptide features in the data. If several features were assigned to one true feature, this was counted as a mismatch.

dataset	manual features	detected features	correct matches	incorrect matches	recall	precision
m1_r1	116	143	90	10	0.78	0.63
m1_r2	131	226	83	25	0.63	0.37
m2_r1	133	214	124	0	0.93	0.58
m2_r2	222	295	179	4	0.81	0.64

Table 3.4.: Performance overview of our algorithm on the test datasets. The number of detected features and the matches to the true features are shown along with the precision and recall calculated from them. Additionally, the number of incorrect matches (true features that were not counted because of differing charge state or multiple matches) are shown.

Table 3.4 shows an overview of the performance. The algorithm could correctly detect between 63% and 93% of the true features, depending on the dataset. This is a good result, considering the complexity of the data. On the other hand, it found up to twice as many features as the annotation by hand. This leads to a bad precision value of 37% to 64%. A low precision was however expected because of the high number of borderline features which could not be annotated unambiguously by hand. Fig. 3.11 shows that

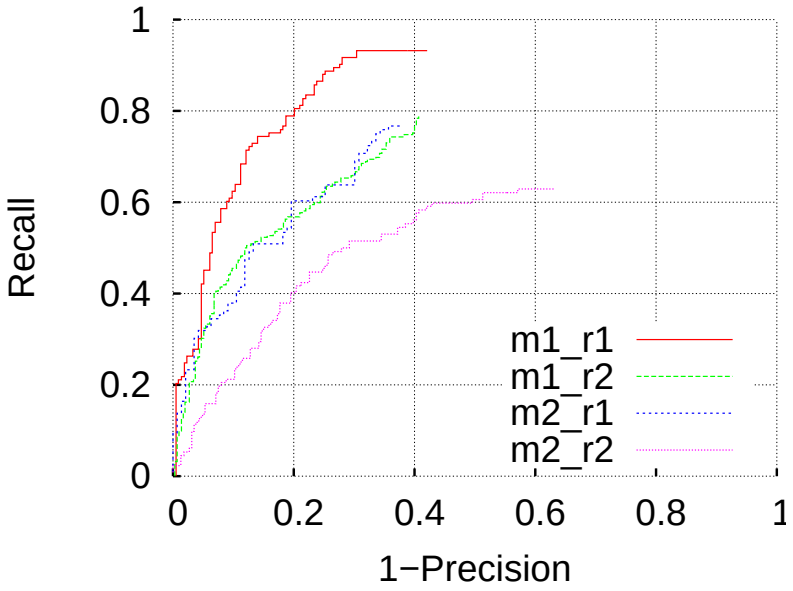


Figure 3.11.: Peptide feature recall plotted against precision. Bipartitions of the output feature set ordered according to intensity were used to calculate corresponding precision and recall values.

many false positives are low-intensity features. They can be easily removed by applying an intensity cutoff to the output feature list.
From the incorrect matches count, one can see that a large part of the undetected true features were not completely missed. They were counted as mismatches because of differing charge assignment or multiple matching features.

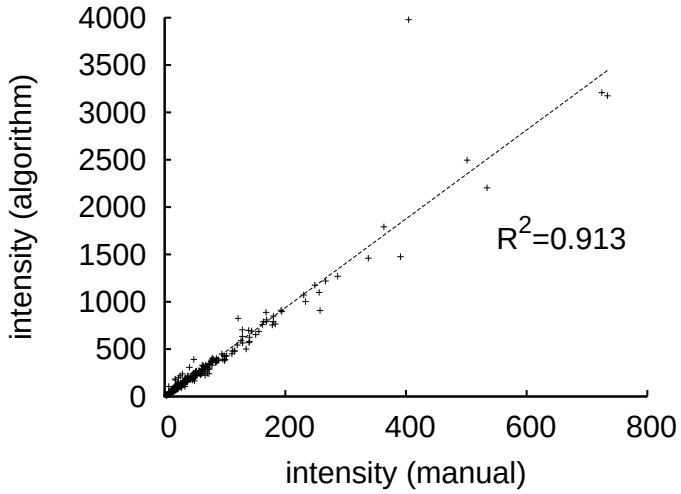


Figure 3.12.: Comparison of manual feature intensities and automatic feature intensities.

Feature intensities

Out of the 300 manually quantified features of dataset 'm3', the algorithm found 272 features. Fig. 3.12 shows a plot of manual quantification results against the automatic

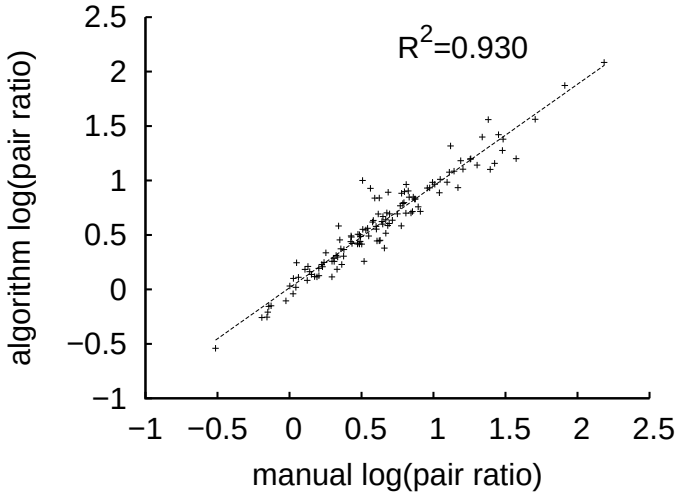


Figure 3.13.: Comparison of manual feature pair intensity ratios and automatic feature pair intensity ratios.

quantification results. This comparison exhibits a good correlation with few outliers. The R^2 between the data points is 0.913.

A closer inspection revealed a small systematic error in the test dataset. The manual intensity values were not always comparable between features because several persons contributed to the analysis. However, the two features belonging to an isotope-labeled feature pair were always quantified by the same person. Therefore, we compared feature pair intensity rations of the manual and automatic analysis. From the 150 feature pairs, 126 pairs have been found by the algorithm. Fig. 3.13 shows that feature pair intensity ratios have been reliably determined by the algorithm. The R^2 between the ratios is 0.930. The relative deviation of the algorithm ratio from the manual ratio was on average 9.1% with a standard deviation of 8.8%. The maximum relative deviation from the manual ratio was 45.2%. Fig. 3.14 shows a histogram of the deviations.

Resolution of overlapping features

Two or more features that overlap are very difficult to detect correctly. Basically, there are three different types of feature overlaps to consider (See Fig. 3.15).

Interleaved features overlap in RT and m/z range, but the mass traces of the features do not overlap in m/z. This type of overlap poses a problem for algorithms based on profile data, because the additional mass traces interfere with the modeling of the continuous m/z signal. Because our algorithm is based on centroided data, it can select peaks with the correct m/z distance from the seed and models the m/z dimension discretely. Thus, interleaved features need no special treatment and are easy to resolve to several features.

In the case of real *overlapping features*, the mass traces of two of more features interpenetrate. Fig. 3.15 shows the two basic types. In the first case (*overlap type 1*), the feature mass traces have the same m/z positions, but a slightly different retention time apex. In the second case (*overlap type 2*), the features have the same retention time apex, but only part of the isotopes patterns overlap. There are of course many hybrid forms of these two cases.

Our algorithm can resolve feature overlaps, if the overlap is not too strong. Fig. 3.7 (page 46) shows an example of two overlapping features with nearly identical retention

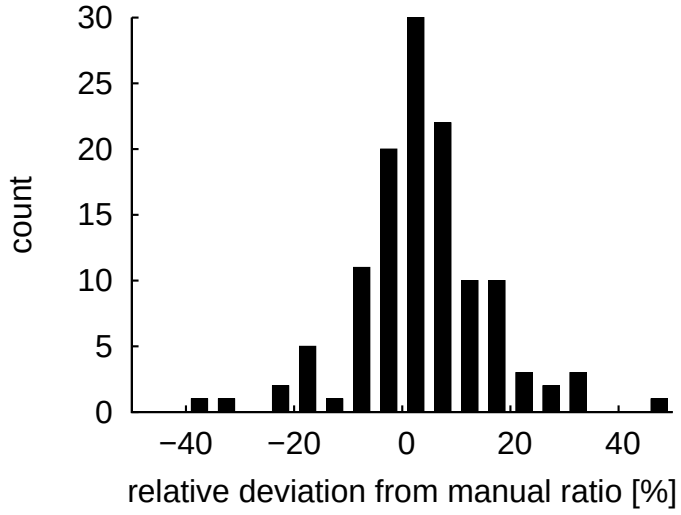


Figure 3.14.: Deviation of automatic feature pair intensity ratios from manual feature pair intensity ratios plotted as histogram.

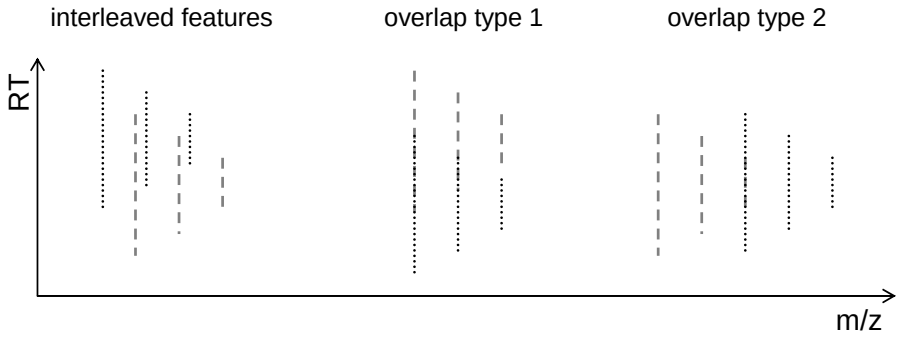


Figure 3.15.: Schematic representation of the three basic features overlap types.

time apex. The overlap is detected in the model fitting phase and the signal is correctly split to two features.
Features that have the same mass-to-charge ratio but differ slightly in elution time are split during the mass traces extension (see Section 3.3.3). Fig. 3.16 shows an example of two features with overlapping mass traces that were successfully resolved.

3.4.4. Comparison to other algorithms

Based on the evaluation of quantitation algorithms by Schulz-Trieglaff *et al.* [78], we selected four algorithms for a thorough comparison: *SpecArray* [13], *msInspect* [12] and two algorithms of the *FeatureFinder* tool [68]. The FeatureFinder algorithms are referred to as *simple* [69] and *isotope_wavelet* [71]. Our algorithm is referred to as *centroided*.
The average performance and average runtime of the selected algorithms on the four test datasets is shown in Table 3.5. Detailed results for the individual datasets can be found in Appendix D. Furthermore, the appendix contains plots of recall against precision for each algorithm.

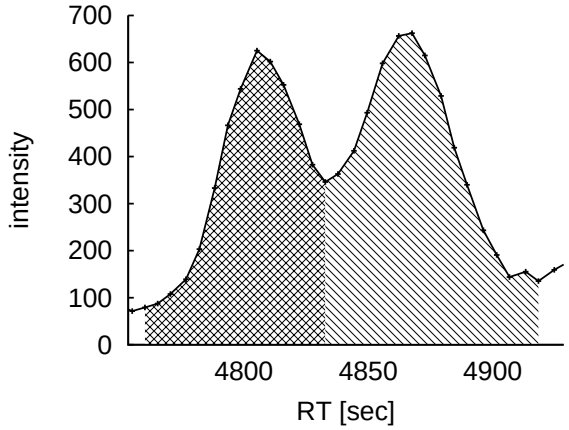


Figure 3.16.: Two features with overlapping mass traces are resolved by splitting the elution profile into two features. The XIC of m/z 627.292 is taken from dataset 'm2_r2'.

SpecArray is a software suite for the evaluation of LC-MS data written in the C programming language. It consists of several tools that form a processing pipeline. Indexed mzXML files are the only accepted input data format of SpecArray. This already poses a problem because indexing an mzXML file is not supported by SpecArray itself. Other software tools have to be used, for example the program *mzXMLIndexer* provided by the TPP [10]. Data converted to the binary SpecArray format is subject to automatic denoising, baseline-reduction and centroiding.
The actual quantitation is done using the *pepList* tool, which writes an ASCII file listing all detected peptide features. Isotope distributions are modeled by an averagine [17] distribution. The elution profile is not explicitly modeled. Isotope clusters of similar mass, charge and retention time are simply merged into one feature.

The algorithm detects only very few features (16% on average), but with a very high precision of 97%. The detected features are those with the highest signal-to-noise ratios. Because the algorithm has no parameters the very conservative internal thresholds cannot be changed. The very low number of detected features and the missing parameters make the algorithm unfit for these test datasets.

msInspect is an MS data viewer and processing tool written in Java. It cannot only be used through a GUI, but also offers a batch mode. Peptide feature detection is invoked through the *findPeptides* command line option. The input format of *msInspect* is mzXML. The output is an ASCII file containing the peptide features.
The first processing step of *msInspect* is to create a binned, image-like representation of the input map. Then, the global background level is estimated conservatively and removed. Peaks in the spectra are identified using wavelet techniques. Isotope patterns are modeled using a simple Poisson distribution. Finally, co-eluting mass traces that follow this theoretical distribution are merged into one feature.
The quality of features is assessed using the Kullback-Leibler divergence [81]. A Kullback-Leibler score of zero, indicates a prefect match. The score increases with an increasing difference between two distributions. The maximum score cutoff is the only parameter of the algorithm. We selected a very high cutoff of 40 in order to get as many features as possible.

The algorithm shows a good performance with a recall of 62% and a precision of 66%. It performs similar to our algorithm—with less recall and better precision. Another property of msInspect that has to be emphasized is its speed. It is the fastest algorithm in the comparison. The speed can be explained by the binning, which allows very efficient access to the profile data and reduces the amount of data significantly.

FeatureFinder (simple) is one of several quantitation algorithms provided by TOPP [68]. The *FeatureFinder* tool reads an mzML peak data file and writes the peptide features in the OpenMS [82] *featureXML* format.
The overall algorithm design is similar to that of our algorithm. It uses a simple signal-to-noise cutoff to determine all seeds. The seeds are extended to a feature region. The feature region is fitted using a two-dimensional model based on an averagine isotope distribution and an EMG elution profile.
The algorithm has many parameters that need to be tuned. Some of the parameters are hard to determine because they cannot be easily estimated from the peak data. Because the algorithm is the slowest in this comparison, finding good parameters is a difficult task. We could not find parameters that guarantee both good performance and acceptable runtime. Thus, we chose the best parameters in terms of performance (see Table D.8).

The algorithm cannot compete with our *centroided* algorithm or *msInspect*. It achieves only about 40% recall and 50% precision. The runtime of the algorithm is far higher than the runtime of all other algorithms. This seems to stem from the very simple seeding strategy. The same feature region can be extended and fitted very often, which leads to unacceptable runtimes. The reason for this problem is probably the high noise level of our test datasets, which results in a very high number of seeds.

FeatureFinder (isotope_wavelet) is a quantitation algorithm based on an isotope wavelet. It is invoked through the TOPP tool *FeatureFinder* as well. Thus, it also reads mzML peak data and writes the peptide features in featureXML format.
Isotopic peaks in single spectra are detected using a wavelet which models an averagine isotope distribution [73]. Signals above a given threshold that occur in several adjacent scans are combined into a peptide feature. No model is used for the elution profile.
The algorithm has only a few parameters, which are easy to set. The maximum charge state to consider has to be set. The elution profile is described by two parameters: (1) the minimum number of spectra in which a signal has to occur and (2) the maximum number of spectra in which the signal can be missing. Here we chose the same values as in the *centroided* algorithm. The most important parameter is an 'intensity_threshold' for isotope patterns in the wavelet transformed signal. It was set to 30 and 7 for the datasets 'm1' and 'm2', respectively.

With a recall of 28% and a precision of 16% the algorithm seems to have the lowest performance in this comparison. However, the output of the algorithms is not as bad at these numbers suggest. It suffers from two major problems, which make it fail in this evaluation.
The first problem is that the feature RT apex is not determined as accurately as with all other methods. When relaxing the maximum allowed RT deviation to 40 seconds (instead of 15), the recall is 35% and the number of incorrectly matched features increases to 32%.
The high number of incorrectly matched features is mainly caused by multiple matches to one true feature, which have different charge states. The missing resolution of conflicting

features is also mentioned as an unsolved problem by the authors of the algorithm.
If these two problems can be overcome, the algorithm has the potential to reach up to 70% recall, which would be a good value. Finally, one has to mention that the algorithm was designed to run either on the CPU, or on modern GPUs. The authors have shown, that an up to twohundred-fold speedup can be reached when using GPUs for calculations. This would make the algorithm faster than all other algorithms in this comparison.

3.5. Summary and conclusion

We have presented a novel peptide feature detection algorithm for centroided LC-MS data. The algorithm was designed for medium-resolution and high-resolution centroided data. Thus, it is applicable to a wide range of datasets, independent of the availability of profile data.

algorithm	recall	precision	runtime [s]	incorrect matches
centroided	0.79	0.55	31.87	7.3%
SpecArray	0.16	0.97	401.89	0.5%
msInspect	0.62	0.66	6.60	6.1%
simple	0.40	0.50	2942.94	10.9%
isotope_wavelet	0.28	0.13	413.92	19.5%

Table 3.5.: Comparison of quantitation algorithm performance. All numbers are average over the four test datasets.

In the presented evaluation the algorithm reached a good average recall (0.79) and precision (0.55) on three very complex Q-TOF datasets of different resolution. The intensity values determined by our algorithm are very accurate and show a high correlation to manually determined intensity ratios ($R^2 = 0.930$). Preliminary test runs on high-resolution Orbitrap datasets and on less complex datasets show that the algorithm works reliably on all isotope-resolved datasets (data not shown).
We compared our algorithm to four other quantitation algorithms, of which only *msInspect* showed a comparable performance (see Table 3.5). The other three algorithms only reached a precision between 16% and 50% and were significantly slower than the two best algorithms.

Although our algorithm is one of the fastest in this comparison, the runtime can be lowered even further by parallelization. Most modern computers contain two or more processor cores of which the basic implementation of the algorithm uses only one. Thus, we have implemented a simple approach for parallelization—the feature candidates for each charge state are processed by a separate thread. This allows a two-fold to five-fold speedup, depending on the number of charge states and the distribution of feature charge states in the data. All runtimes presented in this evaluation were determined without parallelization support.

Another noteworthy feature of the algorithm is the support of user-specified seeds. The user can provide a seed list in *featureXML* format, which restricts the seeding process to data points near the given seeds. The tolerated deviations in m/z and RT dimension can be set as parameters. User-specified seed lists are useful in several scenarios where only a subset of the features is of interest. For example, they can be used to find only those features for which an MS/MS spectrum was recorded.

The algorithm is available as the C++ class *FeatureFinderAlgorithmPicked* in the OpenMS [82] library. Alternatively, it can be used in analysis pipelines through the *FeatureFinder* command line tool of TOPP [68].

3.6. Outlook

Although the algorithm already shows a good performance, there is still room for improvement. Several potential problems have been identified during the evaluation:
(1) The most obvious problem of the algorithm is the very restrictive averagine isotope distribution. Peptides that contain one or several sulfur atoms deviate significantly from this distribution. This can cause a bad model fit, which in turn leads to various errors, e.g., missed features, inaccurate intensity values or wrong monoisotopic m/z. A significant improvement of the performance can be expected when additional isotope distributions are considered. For example, one could test hypotheses containing one, two and three sulfur atoms in addition to the standard averagine distribution.
(2) Overlapping features are modeled individually by the algorithm. This inaccuracy often leads to a slight overestimation of the feature intensity. In the conflict resolution phase, mixture models of two or more peptide features could be used to re-fit areas with strongly overlapping features. The accuracy of feature intensities could certainly be increased that way.
(3) The Gaussian elution profile does not fit to experimental data in all cases. Especially high-intensity features often show an asymmetric elution profile with tailing. An EMG function could be used to model the chromatographic peak shape more accurately. The effect of this change must however be evaluated carefully. Especially for low-intensity features, the additional fitting parameter for the skewness could make the model fitting instable. Thus, using an EMG function for high-intensity features and a Gaussian for low intensity features is a promising alternative to the current implementation of the elution model.
(4) Our algorithm requires several parameters that describe the input LC-MS map, e.g., charge states and peak m/z accuracy. Although all parameters are intuitive and easy to determine, it would be desirable to estimate them from the input data. Statistics on the data, perhaps in combination with a grid search on parameter space, could be used to estimate all parameters. This would improve the usability of the algorithm.
(5) Finally, the runtime could be improved even further using more sophisticated parallelization approaches. The highest speedup can be expected when processing all seeds of one charge state in parallel. This approach would allow an arbitrary number of threads and would distribute the load more evenly among threads. However, it would also require large changes to the algorithm because multiple seeds from the same region should not be extended in parallel.

4. Retention time prediction

The retention time of biomolecules in a given chromatographic separation system is an important physicochemical property. It plays a role in many biochemical applications. Thus, exact modeling of this property would be desirable. The most important biomolecules, DNA, RNA and proteins, are linear polymers consisting of a limited number of building blocks. Because of this similarity in structure, similar sequence-based models for retention time prediction have emerged. In this chapter, we will shortly summarize the advances in peptide retention time prediction and present our work on DNA retention time prediction.

4.1. Peptide retention time prediction

Because of their complex secondary and tertiary structure, the retention time of proteins is very hard to model. Thus, most models focus on smaller peptides, roughly up to a length of 60 amino acids. Rather simple models for peptide retention time based on the amino acid composition have been used for more than 20 years [83, 84, 85].
Today, one main application of peptide retention time prediction it to be found in shotgun proteomics: Retention time predictors are used to filter out false positive MS/MS peptide identifications by comparing observed and predicted retention time.
In 2004, Petritis *et al.* [86] introduced a retention time predictor based on an artificial neural network. However, this model has one major drawback. It requires several thousand peptide retention times for training, which makes the adaptation to different chromatographic conditions nearly impossible. In 2007, Klammer *et al.* [87] published a model based on SVR [41]. Their predictor still requires more than 200 peptide retention times for training.
Recently, Pfeifer *et al.* [88] proposed another retention time predictor based on an SVR which requires only roughly 50 peptides for training. Moreover, it can be applied to any type of separation system, as it is based on the novel *paired oligo-border kernel*, which considers the peptide sequence only. The good performance of the predictor ($Q^2 > 0.9$ with less then 50 training data points) was demonstrated for strong anion-exchange solid-phase extraction (SAX-PSE) and ion-pair reverse-phase high-performance liquid chromatography (IP-RP-HPLC). The authors could show, that peptide identification results can be significantly improved when using this novel predictor for filtering the peptide hits.

4.2. DNA retention time prediction

DNA retention behavior plays a role in many biochemical applications, such as polymerase chain reaction, genotyping [89], DNA sequencing [90] and gene therapy [91]. Thus, exact models for retention time prediction of DNA have many applications. They could be a tremendous help for the selection of chromatographic separation systems for a specific study and for the optimization of separation parameters such as the temperature.

The problem can be formally defined as the retention time prediction for a given DNA sequence in a specific chromatographic separation system. So far, most DNA retention time models are based on the contributions of the four base types: In 2002, Gilar *et al.* [18] proposed a model for the retention time prediction of oligonucleotides in ion-pair reverse-phase liquid chromatography by summing up the contributions of individual bases, which were determined experimentally by analyzing homo-oligonucleotides. Since the input of the model is only the nucleotide length and base composition, the model is applicable to high temperatures only (60°C–80°C), where the secondary structure is suppressed because of thermal denaturing.

In spite of the broad availability of structural data, investigations into the relationship between the 3D-structure of nucleic acids and their interaction with stationary phases in chromatographic separation systems have been quite limited [92, 93, 94], mainly because of the highly dynamic nature of the process, and the involvement of both a liquid and a solid phase in the phase transfer. Because the mechanistics of these processes are not fully understood, modeling them explicitly is not possible. Fortunately, modern machine learning approaches do not require explicit modeling in order to derive broadly applicable models from training datasets.

We propose a new model for oligonucleotide retention time prediction which differs in two ways from the previous approaches. First, we incorporate additional features of the oligonucleotides into the model, the most important being secondary structure information. Secondary structure is temperature-dependent, thus the model can be applied to lower temperatures as well. The second difference lies in the model generation. Support vector regression (SVR) is used instead of simple linear or logarithmic models. SVR can model non-linear relationships while optimizing the model performance and complexity. This leads to reliable models with significantly increased prediction performance.

The secondary structure information used in our models is derived entirely from predictions. This makes the model independent of experimental data and, thus, widely applicable. To predict RNA secondary structure of a single sequence, the most popular methods use free energy minimization based on dynamic programming algorithms [95]. The prediction accuracy is limited by the incompleteness of the models: Many sequence-dependent effects are not completely understood today. Also, some factors, e.g., non-local effects, cannot be easily integrated into dynamic programming methods. Nevertheless, many advances such as pseudoknot prediction [96], generation of suboptimal structures [97] and the use of folding kinetics in the prediction [98] have been recently made.

4.2.1. Experimental dataset

Oligonucleotide sequence selection: Our dataset consists of retention times measured for 72 oligonucleotides ranging from 15 to 48 bases. As one focus of this study is the influence of secondary structure on the retention time, the dataset consists of oligonucleotides that contain little or no secondary structure and others where nearly all bases form a hairpin. Four sequences that form stable hairpin structures even at elevated temperatures were included. Fig. 4.1 shows the predicted, average fraction of paired bases plotted against the measurement temperature.

A second point of interest is the influence of the sequence on the retention time. That is why the dataset contains two groups of oligonucleotides that have the same overall base composition, but slight differences in the base sequence. Both groups are derived from the 24-mer GTGCTCAGTGTAGCCCAGGATGCC. In the first group, one guanine is exchanged for an adenine; in the second, one guanine is exchanged for a cytosine. The

Sequence	normalized retention time [min]				
	30°C	40°C	50°C	60°C	80°C
TGTAGCTCCAAGATG	16.37	15.54	14.99	14.19	11.96
TAGCTTTCCAAGATG	16.61	16.10	15.54	14.85	12.62
GAGAGAGATCTCTCTC	13.47	13.52	14.02	13.74	11.74
TGTAGCTCCAAGATGCC	16.35	15.67	15.10	14.38	12.16
TAGCTTTCCAAGATGCA	17.30	16.79	16.23	15.49	13.37
GCTCAGTGTAGCCCACGTT	17.29	16.57	15.97	15.12	12.78
TGTAACTTTCCAGGATGCC	18.17	17.34	16.67	15.74	13.67
ACTCAGTGTAGCCCACGATGC	17.65	16.90	16.28	15.55	13.24
ATGCTTCAGTGTAGCCCAGTA	18.56	17.97	17.36	16.47	14.47
GAGAGAGAGAGATCTCTCTCTCTC	13.22	13.41	14.57	15.60	14.08
GTGTGTGTGTGTACACACACACAC	14.83	13.74	13.57	14.22	14.63
GTGCTCAGAGTAGCCCAGGATGCC	16.97	16.60	16.06	15.36	13.47
GTGCTCAGTGTAGGGCAGGATGCC	16.84	16.30	15.89	15.34	13.50
GTGCTCAGTGTAGCGGAGGATGCC	16.62	16.60	16.13	15.36	13.51
GTGCTCAAAGTAGCCCAGGATGCC	17.10	16.61	16.24	15.42	13.62
GTGCTCAGTGTAGCCCAGCATGCC	16.97	16.85	16.27	15.67	13.66
GTGCTCAGTGTAGCCCAGACAGCC	17.21	16.76	16.21	15.51	13.48
GTGCTCAGTGTAGCCCAGGATGCC	17.35	16.98	16.32	15.43	13.50
GTGCTCAGTGTAGCCCAGACTGCC	17.43	16.97	16.40	15.48	13.58
GTGCTCAGTGTAGCCCACGATGCC	17.44	17.00	16.51	15.68	13.58
GTGCTCAGTGTAGCCCAGAATGCC	17.29	16.82	16.37	15.68	13.62
GTGCTCAGTGTAGCCCAGGACAAC	17.41	16.97	16.43	15.70	13.75
CTGCTCAGTGTAGCCCAGGATGCC	17.23	17.00	16.25	15.49	13.47
GTGCTCAGTGTAGCCCAGGATACC	17.46	17.11	16.64	15.80	13.85
GTGCTCAGTGTAGCCCAGGATGCG	17.40	16.90	16.44	15.67	13.67
GTGCTCAGTGTAGCCCAGACATCC	17.65	17.25	16.59	15.83	13.92
GTGCTCAGTGTAGCCCAGGATCCC	17.52	17.06	16.44	15.79	13.72
GTGCTCACTGTAGCCCAGGATGCC	17.62	17.12	16.57	15.67	13.79
GTGCTCAGTGTAGCCCAGGATGAC	17.61	17.12	16.51	15.32	13.88
GTGCTCAGTGTAGCCCAAGATGCC	17.40	16.73	16.37	15.66	13.58
ATGCTCAGTGTAGCCCAGGATGCC	17.77	17.40	16.81	16.08	14.08
GTCCTCAGTGTAGCCCAGGATGCC	17.67	17.08	16.53	15.73	13.76
GTGCTCAATGTAGCCCAGGATGCC	17.52	17.11	16.55	15.84	13.90
GTGCTCAGTGTAACCCAGGATGCC	17.64	17.14	16.61	15.89	13.99
GTGCTCAGTATAGCCCAGGATGCC	17.50	17.08	16.62	15.87	14.00
GTGCTCAGTGTAGCCCAGGATGGG	18.07	17.37	16.61	15.75	13.68
GTGCTCAGTGTAGCCCAGGATAAC	17.82	17.17	16.78	16.11	14.05
GTGCTCAGTGTAGCCCGTGATGCC	17.77	17.15	16.59	15.81	13.57
GTGCTCAGTGTAGCCCAGGATGCA	17.82	17.36	16.83	15.98	13.99
GTGCTCAGTGTAGCCCAGGATGCT	17.81	17.16	16.71	15.94	13.96
GTGCTCAGTGTAGCCCAGGATGAA	17.82	17.31	16.86	16.19	14.29
GTGCTCAGTGTAGCCCATGATGCC	17.78	17.20	16.68	15.90	13.77
GTGCTCAGTGTACCCCAGGATGCC	17.99	17.30	16.66	15.84	13.70
GTGCTCAGTGTAGCCCAGGATGAT	17.90	17.36	16.90	16.20	14.17
GTACTCAGTGTAGCCCAGGATGCC	17.84	17.34	16.70	15.93	14.03
GTGCTCAGTCTAGCCCAGGATGCC	18.05	17.37	16.65	15.78	13.76
GTGCTCAGTGTAGCCCAGACATAC	17.98	17.39	16.85	16.21	14.04
GTGCTCAGTGTAGCCCAGGATGTT	18.16	17.50	17.09	16.31	14.23
GTGCTCAGTGTAGCCCAGGATTAA	18.40	17.87	17.40	16.69	14.68
GTGCTCAGTGTAGCCCAGGATTTT	18.75	18.13	17.61	16.72	14.81
GTGCTCAGTGTAGCTCCAGGATGCC	17.74	17.22	16.72	15.97	13.98
GTGCTTCAGTGTAGCCCAGGATGCC	17.83	17.40	16.79	16.03	13.98
GTGCTCAGTGTAGCTTCCAGGATGCC	18.10	17.72	17.18	16.47	14.48
GTGCTTTCAGTGTAGCCCAGGATGCC	18.49	17.94	17.29	16.61	14.47
GTGCTCAGTGTAGCTTTCCAGGATGCC	18.76	18.30	17.66	16.83	14.93
GTGCTTTTCAGTGTAGCCCAGGATGCC	18.99	18.37	17.72	16.93	14.90
GTGCTCAGTGTAGCTTTTCCAGGATGCC	19.13	18.57	18.04	17.21	15.30
GTGCTCAGTGTAGCCCAGTTTTGATGCC	18.99	18.39	17.85	17.09	15.25
GTGCTTTTTCAGTGTAGCCCAGGATGCC	19.51	18.85	18.20	17.45	15.40
GTGCTCAGTGTAGCCCAGTTTTTGATGCC	19.44	18.89	18.27	17.49	15.69
GTGCTCAGTGTAGCTTTTTCCAGGATGCC	19.64	19.01	18.55	17.67	15.72
GTGCTTTTTTCAGTGTAGCCCAGGATGCC	19.83	19.17	18.57	17.75	15.80
GTGCTCAGTGTAGCCCAGTTTTTTGATGCC	19.72	19.12	18.71	17.72	15.87
GTGCTCAGTGTAGCTTTTTTCCAGGATGCC	19.88	19.40	18.77	17.98	15.99
GTGCTTTTTTTCAGTGTAGCCCAGGATGCC	20.21	19.55	18.99	18.07	16.17
GAGAGAGAGAGAGAGATCTCTCTCTCTCTC	13.65	13.59	14.79	16.56	15.62
GTGCTCAGTATAGCCCAGTTTTTTGATGCCATA	20.64	20.13	19.69	19.08	17.23
CTCAGTGTAACCCAGTTTTTTGATGCCGTAGATCAT	20.57	20.04	19.80	19.27	17.46
GTGCTCAGTGTAACCCAGTTTTTTGATGCCGTAGATCAT	20.93	20.41	20.12	19.57	17.80
GTGCTCAGTGTAACCCAGTTTTTTGATGCCGTAGATCATAAA	21.20	20.75	20.47	19.93	18.41
GTGCTCAGTGTAACCCAGTTTTTTGATGCCGTAGATCATAAATTT	21.60	21.15	20.89	20.31	18.85
GTGCTCAGTGTAACCCAGTTTTTTGATGCCGTAGATCATAAATTTAGA	21.76	21.24	21.07	20.63	19.08

Table 4.1.: Oligonucleotides sequence and measured normalized retention times.

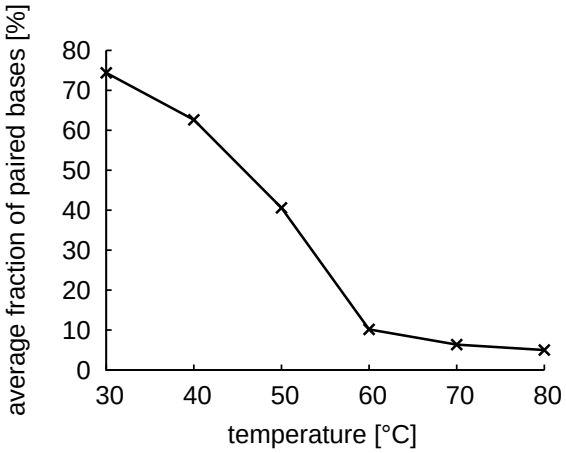


Figure 4.1.: Average fraction of bases in stems and loops for the 72 oligonucleotides. The secondary structures were predicted using *RNAFold* [99].

sequences of the 72 oligonucleotides can be found in Table 4.1.

Chemicals and oligonucleotides: Acetonitrile (far UV HPLC grade) and acetic acid (analytical reagent grade) were obtained from Riedel-de-Haën (Seelze, Germany), triethylamine (analytical reagent grade) and ethylenediamine tetraacetate (EDTA, analytical reagent grade) were purchased from Fluka (Buchs, Switzerland). Triethylammonium acetate was prepared by mixing equal amounts of triethylamine and acetic acid. Water was purified by means of a purification system (Purelab Ultra Genetic, Elga, Siershahn, Germany). Oligonucleotides were synthesized by MWG-Biotech AG (Ebersberg, Germany) or Biospring (Frankfurt, Germany).

Ion-pair reversed-phase high-performance liquid chromatography: Ion-pair reversed-phase high-performance liquid chromatography was performed with a fully automated capillary/nano HPLC system (Model Ultimate 3000, Dionex, Amsterdam, The Netherlands) equipped with a low-pressure gradient micropump (Model LPG-3600) with a vacuum degasser (Model SRD-3600), an integrated column oven, a microcolumn switching unit and flow-splitting device (Model FLM-3100), a micro-autosampler (Model WPS-3000), and a UV-detector (Model UVD 3000) with a 3-nL Z-shaped detector cell (Model Ultimate). The system was controlled by a personal computer with Chromeleon Version 6.60 SP2 (Dionex). The 60 x 0.20 mm i.d. poly-(styrene/divinylbenzene) monolithic column was prepared according to the published protocol [100] and is available from Dionex.

Generation of experimental retention time datasets: One to five nanograms of oligonucleotides, dissolved in 1 μl water, were injected and chromatographed with a 30 min linear gradient from 0–16% acetonitrile in 100 mmol/l triethylammonium acetate, 0.5 mmol/l EDTA, pH 7.0. The flow rate was adjusted to 2 μl/min. The gradient delay volume of the used LC system was 5.94 μl. Although retention times of oligonucleotides were highly reproducible (0.26% relative standard deviation in absolute retention time from three repetitive injections), $(dC)_{14}$ and $(dT)_{26}$ homooligonucleotides were co-injected as internal standards to normalize retention. A representative separation of an oligonucleotide

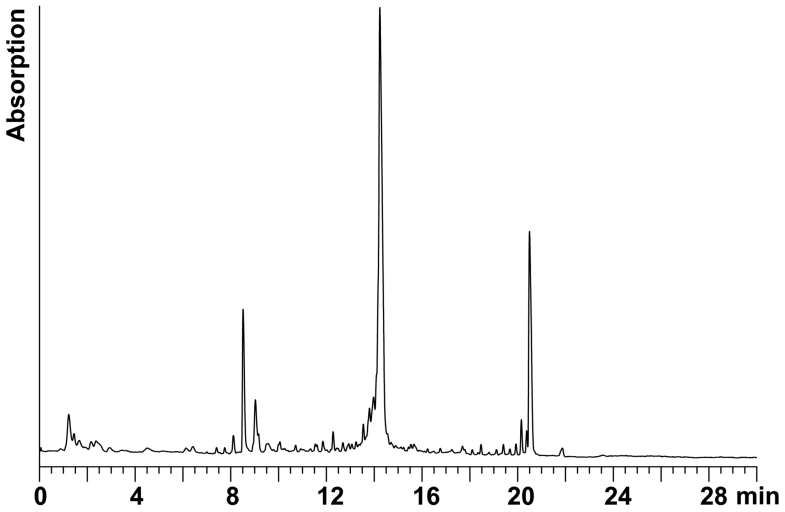


Figure 4.2.: Sample separation of an oligonucleotide (second peak) with internal standards $(dC)_{14}$ (first peak) and $(dT)_{26}$ (third peak) at 80°C.

including the two standards is illustrated in Fig. 4.2.

Normalization was performed using Eq. 4.1 (t and $\bar{t}$ representing the retention time and average retention time, respectively, of the oligonucleotides) and resulted in retention time measurement errors of less than 0.04% (0.6 sec).

$$t_{normalized} = (t_{measured} - t_{(dC)_{14}}) * \frac{\bar{t}_{(dT)_{26}} - \bar{t}_{(dC)_{14}}}{t_{(dT)_{26}} - t_{(dC)_{14}}} + \bar{t}_{(dC)_{14}} \tag{4.1}$$

The 72 different oligonucleotides were chromatographed at column temperatures of 30, 40, 50, 60, and 80°C and the measured normalized retention times are summarized in Table 4.1. All chromatographic experiments were performed by Sascha Quinten and Christian G. Huber.

4.2.2. Feature selection

The selection of input features is a critical step, as the features determine the performance of the prediction. Leaving out essential features or adding unnecessary features both lead to a drop in prediction accuracy. We thus tested several different models, i.e., input feature sets. Each model consists of several model components which group closely related features. Besides these model components, each model implicitly contains the temperature and the length of the oligonucleotide. The model components can be grouped into composition components, structural components and energy components. Table 4.2 gives an overview of all components.

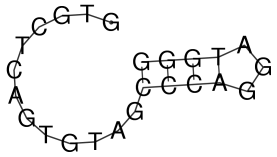
All secondary structure components are calculated from predicted secondary structures. For this purpose, the *RNAFold* tool of the *Vienna RNA Package* [99] version 1.4 was used.

Sequence components are calculated from the oligonucleotide sequence and describe base composition or sequence details of the oligonucleotide. COUNT reflects the base composition, while CONTACT and SCONTACT count dinucleotide frequencies, which contain information on the adjacency of bases. The key idea here is that stacking bases will influence the secondary structure and the interaction with the stationary phase.

Name	Features	Description
Composition components:		
COUNT	4	Base frequencies (#A, #C, #G and #T in the sequence).
CONTACT	16	Dinucleotide frequencies (#CG, #CA, #CT, #CC, ...).
SCONTACT	10	Dinucleotide frequencies, independent of direction (#CG+#CG, #CA+#AC, #CC, ...).
Secondary structure components (for measurement temperature only):		
PAIRED	4	Fraction of A, C, G and T inside stems.
UNPAIRED	4	Fraction of A, C, G and T outside stems.
STRUCTURE	12	Fraction of A, C, G and T in stems, in loops, or unpaired.
Secondary structure components (for temperatures 30, 40, 50, 60, 70, 80° C):		
MULTISTRUCT	6	Fraction of bases in stems.
MULTITWO	12	Fraction of bases that are unpaired, and in stems or loops.
MULTIDETAIL	36	Fraction of bases in stems, and loops. Fraction of unpaired A, C, G and T.
Energy components:		
SESUM	1	Sum of the stacking energies of adjacent bases.
STACKING	2	Sum of the enthalpic (ΔH) and entrophic ($T\Delta S$) contributions to the free energy.

Table 4.2.: Overview of the model components used in this work.

Since retention time behavior strongly depends on the secondary structure, sequence based features alone are suitable only for high temperatures. At lower temperatures, the secondary structure requires the inclusion of structure-based components in the prediction. These components can be subdivided into two groups. The first group considers only the secondary structure of the temperature at which the measurement was performed, whereas the second group contains information on secondary structures for temperatures of 30, 40, 50, 60, 70 and 80°C.



Feature	C	G	T	A
Composition	5	10	5	4
Bases in stems	3	3	1	1
Fraction of bases in stems	0.60	0.30	0.20	0.25
Bases in loops	0	2	0	1
Fraction of bases in loops	0.00	0.20	0.00	0.25

Figure 4.3.: Example of the feature calculation from the predicted secondary structure of GTGCTCAGTGTACCC at 40°C.

Fig. 4.3 shows the predicted secondary structure of the oligonucleotide GTGCTCAGTGTAGCCCAGGATGGG at 40°C and the features calculated from the predicted structure. For the simple structural components, only the measurement temperature is considered. For the multi-temperature structural components, the secondary structure and the resulting features are calculated for different temperatures.

The single temperature components PAIRED and UNPAIRED have four features each reflecting the fraction of A, T, C and G inside stems and outside stems, respectively. The STRUCTURE component contains more detailed information, as it considers the fraction of A, T, C, and G inside stems and inside loops and unpaired nucleotides.

The multi-temperature structural components reflect secondary structure information

as well. In contrast to the single temperature components, they do not only consider the measurement temperature but a range of temperatures (30, 40, 50, 60, 70 and 80°C). Adding this gradient of secondary structure information for different temperatures provides information similar to a melting curve. The simplest component in this group is MULTISTRUCT, which, for each of the six temperatures, contains the fraction of all bases that are inside stems. MULTITWO contains the fraction of bases in stems and loops, as well as the fraction of unpaired bases. The most detailed representation of the secondary structure is MULTIDETAIL. It contains the fraction of bases in stems, the fraction of bases in loops, and the fractions of A, C, G and T that are unpaired.

Energy components describe the ring stacking effects between adjacent bases. These energies might influence retention time as they have to be overcome in order for the aromatic ring system of the base to interact with the column packing. The EMBOSS suite [101] was the source of the stacking energies of base pairs used in the SESUM component. The thermodynamic parameters used for the STACKING component were taken from [102].

4.2.3. Models

Support vector regression In this study, the libSVM [103] implementation of ν-SVR [41] was used with the *Radial Basis Function* kernel. The kernel parameter γ and the trade-off C have been optimized using grid search and three-fold cross validation on the training data.

Single temperature and combined temperatures models: The standard approach in predicting retention time is to create fixed-temperature models, i.e., each model is based on data for a single temperature. Thus, if predictions for several temperatures are needed, one model has to be created for each temperature.

The model components that consider multiple temperatures allow for a more general approach. The data points of one dataset may differ in temperature, which leads to a model that can predict retention times over the whole temperature range of the training data. The advantage of this approach is that all available data can be integrated into a single, more general model. This reduces the amount of data needed for the individual temperatures.

Combining datasets of all temperatures yields a dataset with 432 data points (72 data points from six temperatures). To avoid overfitting, we included data from only one randomly chosen temperature for each oligonucleotide. The resulting model was then used to predict the retention times of the remaining measurements.

Model creation: From the 11 model components presented in Table 4.2, we created models containing one or more of these components. As the number of models that can be built out of 11 components is huge, we focused on models built out of two or three components, which mostly contain one composition component and one structural component. These models will subsequently be referred to by the names of the components they contain (e.g., 'count_sesum' for a model that consists of the COUNT component and the SESUM component).
For each temperature, the models were trained and the results evaluated. Additionally, the models were trained on the combined temperature dataset.

4.2.4. Results

For each model, the average prediction correlation Q^2 in three-fold cross-validation was determined for all temperatures. Table 4.3 shows the prediction performance for a selection of models. Among the composition components SCONTACT and CONTACT did not show much difference. In the group of single-temperature secondary structure components, the combination of PAIRED and UNPAIRED was the only one that performed at a similar level to the multi-temperature structural components, where MULTISTRUCT and MULTITWO were the best components. Out of the energy components, STACKING always performed better than SESUM.

Model	30°C	40°C	50°C	60°C	80°C	ALL	average
contact	0.934	0.891	0.891	0.947	0.968	0.934	0.927
count	0.583	0.621	0.762	0.918	0.988	0.873	0.791
count_contact	0.925	0.888	0.855	0.953	0.976	0.929	0.921
count_contact_multistruct	0.968	0.962	0.926	0.964	0.974	0.948	0.957
count_multidetail	0.943	0.969	0.964	0.953	0.954	0.919	0.950
count_multistruct	0.951	0.976	0.962	0.959	0.974	0.954	0.963
count_multistruct_stacking	0.956	0.977	0.960	0.957	0.983	0.953	0.964
count_multithree	0.929	0.970	0.954	0.955	0.972	0.929	0.952
count_multitwo	0.938	0.974	0.964	0.963	0.981	0.934	0.959
count_multitwo_stacking	0.950	0.975	0.964	0.959	0.985	0.936	0.961
count_paired_stacking	0.956	0.940	0.956	0.955	0.986	0.936	0.955
count_paired	0.948	0.946	0.956	0.958	0.982	0.935	0.954
count_scontact	0.902	0.866	0.843	0.961	0.988	0.924	0.914
count_scontact_multistruct	0.978	0.978	0.950	0.965	0.975	0.954	0.967
count_structure_stacking	0.921	0.912	0.959	0.958	0.983	0.928	0.943
count_structure	0.921	0.916	0.957	0.964	0.976	0.928	0.944
paired_unpaired_stacking	0.947	0.953	0.956	0.955	0.983	0.934	0.955
paired_unpaired	0.942	0.949	0.954	0.962	0.963	0.929	0.950
scontact	0.912	0.897	0.897	0.952	0.975	0.931	0.927
scontact_multistruct	0.972	0.968	0.949	0.957	0.955	0.955	0.959
scontact_multistruct_stacking	0.971	0.972	0.953	0.954	0.963	0.954	0.961
scontact_multitwo_stacking	0.958	0.965	0.946	0.950	0.964	0.941	0.954
scontact_paired_stacking	0.933	0.946	0.935	0.960	0.972	0.933	0.946
scontact_paired	0.938	0.944	0.931	0.960	0.967	0.935	0.946
scontact_stacking	0.907	0.899	0.890	0.950	0.977	0.930	0.925
scontact_structure_stacking	0.931	0.912	0.941	0.968	0.981	0.929	0.944
scontact_structure	0.929	0.904	0.934	0.969	0.978	0.928	0.940
structure_stacking	0.937	0.919	0.955	0.959	0.983	0.924	0.946
average	0.928	0.928	0.931	0.957	0.975	0.933	

Table 4.3.: Performance overview of all tested models (Q^2).

Single temperatures datasets: All models show a good prediction correlation on the 80°C dataset, but the prediction performance averaged over all models drops with the temperature. The models that are exclusively based on sequence information ('count', 'scontact' and 'count_scontact') cannot compete with those models that use secondary structure information at lower temperatures.

Combined temperatures dataset: The models with multi-temperature structural components outperform all other models on the combined temperature dataset. Out of these

components, 'multistruct' performs best.

For all further evaluations, we chose the model 'count_multistruct_stacking'. It was preferred over the model 'count_scontact_multistruct', which shows a slightly better average performance, because it has far fewer features and thus has less tendency to overfit.

Homology reduction: To rule out the possibility that the good prediction performance stems from overfitting caused by sequence homology present in the dataset, three homology-reduced datasets were created. We simply removed oligonucleotides from the original dataset until the remaining sequences differed in at least two, three or five bases, respectively.

dataset	oligonucleotides	30°C	80°C
unreduced	72	0.957	0.983
two bases differing	52	0.954	0.986
three bases differing	38	0.957	0.965
five bases differing	29	0.911	0.963

Table 4.4.: Prediction performance on homology-reduced datasets at 30°C and 80°C.

Table 4.4 shows that there is almost no drop in the prediction performance for homology-reduced datasets. Only the dataset with oligonucleotides that differ at least in five bases actually performs a little worse than the rest, which is probably caused by the small number of training data points.

Comparison to the Gilar model: In 2002, Gilar *et al.* [18] proposed a simple mathematical model for predicting the retention time of oligonucleotides. The model takes only the overall length and the base composition of oligonucleotides into account. As Eq. (4.2) shows, the model is a sum where each addend corresponds to the contribution of one base type to the retention time:

$$rt = \sum_{i \in \{A,T,C,G\}} \frac{a_i N_i + b_i N_i ln(N)}{N} \tag{4.2}$$

where N_i is the number of occurrences of base i in the oligonucleotide and N is the overall length of the oligonucleotide (i.e., $N = N_A + N_T + N_C + N_G$). There are two coefficients a_i and b_i for each base type i which have to be determined experimentally. Therefore, the retention times of homo-oligonucleotides of different lengths were measured for each base type. The individual contributions of each base can then be fitted to the measured times in order to determine a_i and b_i.

We compared the Gilar model to our model 'count_multistruct_stacking'. The Gilar model fits very well at 80°C but drops to R^2=0.58 at 30°C (see Fig. 4.4). In contrast, the 'count_multistruct_stacking' model maintains a performance of $R^2 \geq 0.95$ for all temperatures. Fig. 4.5 shows the retention times predicted by the Gilar model at 30°C and 80°C. At 80°C, the prediction performance is equal to the performance of our models. However, at 30°C, the hairpin structures (marked with circles) cannot be predicted correctly anymore and generally, prediction errors increase significantly (see also Fig. 4.5).

Influence of secondary structure: To demonstrate the effect of adding secondary structure information to a sequence-based model, we compared the model error of the models

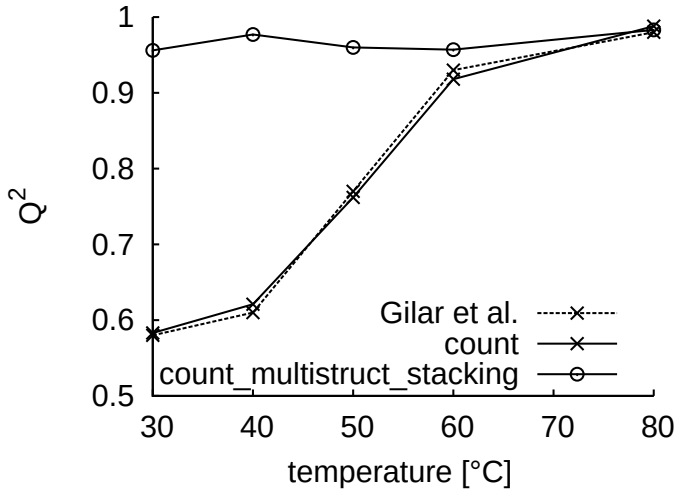


Figure 4.4.: Comparison of the Gilar model to our models 'count' and 'count_multistruct_stacking'.

'count' and 'count_multistruct_stacking'. Table 4.5 shows the average relative model error of oligonucleotides with a similar fraction of bases involved in their secondary structure.

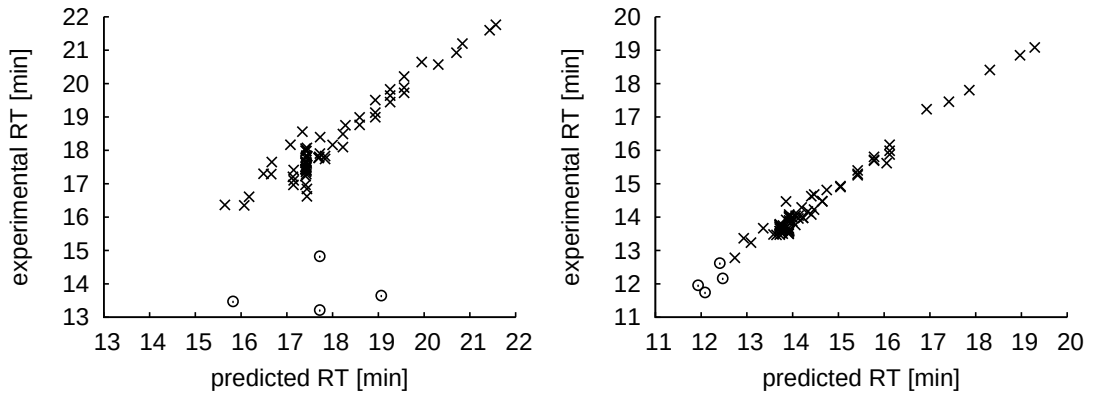
Temp.	**Model**	**Fraction of bases in secondary structure**				
		0.0 - 0.2	0.2 - 0.4	0.4 - 0.6	0.6 - 0.8	0.8 - 1.0
30°C	count	0.98%	1.12%	1.41%	15.47%	37.62%
30°C	count_multistruct_stacking	0.65%	1.01%	1.21%	2.35%	1.19%
50°C	count	0.78%	0.80%	1.11%	17.95%	22.70%
50°C	count_multistruct_stacking	0.38%	0.39%	0.43%	0.27%	0.26%
80°C	count	0.57%	2.12%	na	1.94%	0.40%
80°C	count_multistruct_stacking	0.44%	0.34%	na	0.28%	0.09%

Table 4.5.: Average relative model error of oligonucleotides with a similar fraction of bases involved in secondary structure. 'na' indicates that there is no oligonucleotide with the corresponding fraction of bases in the secondary structure for that temperature.

The 'count_multistruct_stacking' model can compensate for the influence of the secondary structure. However, the relative model error of the 'count' model rises up to nearly 38% and 23% for several oligonucleotides at 30°C and 50°C, respectively. It is simply impossible for the 'count' model to derive a reasonable model from the training data at lower temperatures as it does not incorporate secondary structure information. At 80°C, the 'count_multistruct_stacking' model still has lower relative model errors than the 'count' model.

The amount of data required for training: For practical use, the number of training data points that are required for a reasonable model is very important. So we determined an average prediction performance for the range of 10 to 60 training data points. We used the following procedure:

1. Repeat the following steps 200 times:



(a) 30°C (b) 80°C

Figure 4.5.: Results of the Gilar model at 30°C and 80°C. The hairpin structures are marked with circles.

a) Randomly choose 10 data points as a test set.

b) Randomly choose 10, 20, 30, 40, 50 and 60 of the data points not contained in the test set as training set.

c) For each of the training sets, train a model, predict retention times for the test set and store the squared correlation coefficient.

2. Calculate the average squared correlation for 10, 20, 30, 40, 50 and 60 training data points.

Fig. 4.6 shows the results for the 'count_multistruct_stacking' model. From the figure one can see that even for a temperature of 30°C, 40 training sequences are sufficient to construct a model describing oligonucleotide retention with acceptable accuracy. No significant improvement is observed for more than 50 data points.

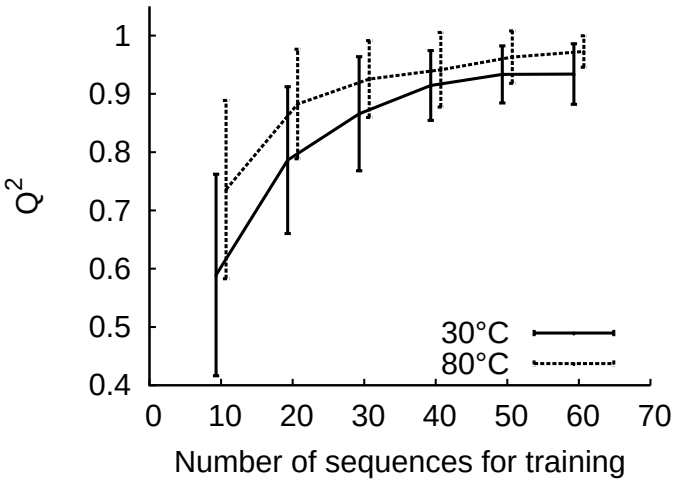


Figure 4.6.: Number of training data points plotted against prediction performance. The error bars show the standard deviation, derived from 200 repetitions of the experiment.

4.2.5. Discussion and outlook

The methods, experiments and results presented in this chapter clearly demonstrate that predicting oligonucleotide retention time can be significantly improved using secondary structure information and support vector regression.

The use of secondary structure information improves the prediction performance, especially at low temperatures and for oligonucleotides that form highly stable secondary structures. The second point of interest in this study was the effect that the base sequence had on the retention time. The fact that our best model contains no explicit sequence information shows that it plays only a subordinate role in this process. Rather, it is the influence of the base sequence on the secondary structure that seems to be the base sequences' main contribution. Explicitly modeling the base sequence is therefore not necessary. However, a closer examination of the influence of the base sequence remains to be done.

The second performance boost came from the use of SVR, probably because of its ability to model non-linear relations and because of the improved handing of outliers in the training data. The essential step, and limiting factor, when working with an SVR model is the selection of the training data. A good prediction performance for oligonucleotides that lie far outside the training feature space is very unlikely. Thus, it is crucial to ensure a broad coverage of feature space, both in terms of secondary structure and base composition.

We tested 11 model components that cover base composition, base sequence, secondary structure and base stacking. Although our model performed very well, there is still room for improvements. New features that model additional chemical properties can be easily added to our model by appending them to the feature vector. We expect that this kind of model extensibility can further help to improve the oligonucleotide retention time prediction model in the future.

5. OpenMS and TOPP

Mass spectrometry has become a key technique for biomedical research, especially in proteomics and metabolomics. Although MS is used in these fields for many years now, there are still rapid advances in instrumentation and experiment design. As a result, data analysis software is evolving quickly as well—existing software has to be adapted to new instruments, and new software has to be written for novel experiments.

Thus, it would be advantageous to view data analysis as a series of smaller analysis problems. For example, one specific quantitation protocol could consist of a process involving *peak picking, quantitation, normalization, map alignment* and *marker finding*. Other quantitation protocols, e.g., for labeled data, differ from the above, although they have many steps in common. Therefore, it would be desirable to have smaller algorithmic components that can be readily combined into more complex workflows or tools.

5.1. State of the art

Commercial software: Commercial software packages, like MassLynx [7] and DeCyder [8], often perform specific tasks very well, however, their use as has many drawbacks. Almost all commercial software relies on proprietary data formats. The import and export of raw data and analysis result in non-proprietary data formats is not always possible. This makes the construction of large analysis pipelines with different tools very difficult or even impossible. The use of proprietary data formats also makes the joint analysis of data from different MS instruments or even different instrument vendors impossible.

Moreover, the algorithms implemented in commercial software are often not published and the parameters that govern their behavior are rarely accessible. Comparison of analysis algorithm performance is also very difficult, because intermediate results of single analysis steps are not accessible.

In summary, one can say that the use of commercial tools severely limits the control the user has over the data analysis.

Academic software: There are many academic projects for mass spectrometry data analysis. Table 5.1 gives an overview of the available tools and summarizes their features. In this section we will highlight the features and shortcomings of the individual tools.

As we have discussed before, support for non-proprietary file formats is a requirement for wide usability of data analysis tools. All but one of the listed tools support at least one of the non-proprietary mass spectrometry data formats ANDI/MS [104], mzData [105], mzXML [106] or mzML [107]. *MaxQuant* support the Thermo Finnigan RAW file format only and, thus, is not applicable to data from different instrument vendors. The only tool that can read all four non-proprietary file formats is *MZmine*.

Another important feature for analysis tools is flexibility. The analysis should be adaptable to new instruments and experiments. However, most of the present tools combine several analysis steps to a fixed pipeline. Some tools can be used through a GUI only, which does not allow high-throughput analysis or scripting.

There are only a few tools that allow a flexible use of several analysis steps in a high-throughput manner: *msInspect* offers all methods from the GUI in a CLI batch mode. *XCMS* allows flexible scripting through an *R* interface, but the project is focused on metabolomics and the functionality for proteomics data analysis is very limited. *SpecArray* and the *Trans-Proteomic Pipeline (TPP)* support flexible analysis pipelines—they both come as a set of CLI tools which can be used to build custom analysis pipelines.

Besides flexibility, reusability is also quite important. Analysis methods and algorithms should be implemented in such a way, that they can be reused by other projects. Otherwise, algorithms with similar goals are developed over and over again. However, only *msInspect*, *XCMS* and *ProteoWizard* make their functionality available as a well-documented class library.

Finally, availability of analysis tools is very important. Not every tool is available for all major operating systems. Here, the choice of programming language and build system plays a large role. *MaxQuant* and *Viper* are available for Windows only, because they are developed in C# and Visual Basic, respectively. *MZmine* and *msInspect* are written in Java, a platform-independent programming language that runs all major operating systems. Most other projects are implemented in C++, which can be compiled on various platforms as well. However, *SpecArray* and *SuperHirn* use an outdated build system without Windows support.

Our contribution: We present *OpenMS – An open-source framework for mass spectrometry* [82]. OpenMS aims at providing building blocks for analysis tools and is therefore more flexible than all current commercial tools and openly available libraries. It serves as a framework for developing mass spectrometry data analysis tools, providing everything from basic data structures over file input/output (I/O) and visualization to sophisticated algorithms. Thus, OpenMS allows developers to focus on new algorithmic approaches instead of implementing infrastructure.

Three academic projects with very similar goals and functionality have been developed in parallel to OpenMS: XCMS [11, 108], ProteoWizard [109] and msInspect [12]. XCMS focuses on metabolomics, providing only very limited proteomics data analysis capabilities. ProteoWizard is a platform-independent C++ library. However, the analysis functionality of ProteoWizard is rudimentary. The current focus of the project is file I/O, especially the support of all major MS vendor formats. Most of the high-level functionality needed for the actual analysis of MS data is still missing. msInspect is the most mature toolset in terms of functionality and usability. It also provides a Java class library. However, the design of the library and the quality of its documentation are not satisfying.

To make the rich functionality of OpenMS available to users not skilled in software development, we developed a suite of computational tools called *TOPP – The OpenMS Proteomics Pipeline* [68]. TOPP is a collection of command line tools—each performing one atomic analysis step. These lightweight tools serve as building blocks for more complex proteomics pipelines. This modular setup allows flexible construction of new analysis pipelines and rapid adaptation of existing pipelines to new experiments.
An academic project with goals similar to those of TOPP is the Trans-Proteomic Pipeline (TPP) [10]. TPP integrates several tools developed at the ISB into a coherent framework. Among those tools are *ProteinProphet* [110], *PeptideProhet* [111], *ASAPRatio* [112] and *SuperHirn* [15]. Despite the similar philosophy, TPP and TOPP differ in scope. TPP focuses on quantitation and identification of peptides/proteins only. TOPP is not focused on one particular area of MS data analysis. It offers tools for all analysis steps including

signal processing, file handling and visualization.

In this chapter, we will highlight the core architectural features of OpenMS and how it can be used to construct powerful applications for proteomics data analysis. We will also elaborate the different algorithms already contained in OpenMS and demonstrate its versatility in several small code examples. The last part of this chapter describes the construction of powerful analysis pipelines using the TOPP tools.

5.2. Design goals

OpenMS is intended to offer a rich functionality while keeping in mind the design goals of ease-of-use, robustness, extensibility and portability. We will now briefly describe the techniques used to achieve these goals.

Ease-of-use: A very important aspect for the acceptance of a software library is ease-of-use. Software developers expect clear interfaces and a good documentation.
The object-oriented programming paradigm aims at mapping real-world entities to comprehensible data structures and interfaces. Combining it with a coding style that enforces consistent names of classes, methods and member variables, leads to intuitive usability of a software library. For these reasons we adapted this paradigm for OpenMS.
The documentation of OpenMS is created using Doxygen [114]. This tool allows integration of the class documentation into the source code, which ensures consistency of code and documentation. The documentation is generated in HTML format making it easy to read with a web browser. OpenMS also provides several tutorials in HTML and PDF format which introduce the most important concepts and analysis tools using example applications.

Robustness: Robustness is the ability of a software system to handle abnormal situations. While robustness is not of the essence when developing new algorithms, it is essential if a new method will be applied routinely to large scale datasets. Typically, there is a tradeoff between performance and robustness. OpenMS tries to address both issues equally.
In general, we try to tolerate recoverable errors, e.g., files that do not entirely fulfill the format specifications. On the other hand, exceptions are used to handle fatal errors. To check for correctness, unit tests are implemented for each method of a class. These tests check the behavior for both valid and invalid use. Additionally, preprocessor macros are used to enable additional consistency checks in debug mode, which are disabled in productive mode for performance reasons.

Extensibility: Since OpenMS itself is based on several external libraries, it is designed for the integration of external code. All classes are encapsulated in the *OpenMS* namespace to avoid symbol clashes with other libraries.
Through the use of template code, the core data structures are adaptable to specific problems. For example, it is possible to replace the representation of the mass-spectrometric peak. Also, OpenMS supports standard formats and is itself open-source software. The use of standard formats ensures that applications developed with OpenMS can be easily integrated into existing analysis pipelines. OpenMS source code is located on SourceForge [115], a repository for open-source software. This allows users to participate in the project and to contribute to the code base.

	supported community file formats					supported platforms				
	ANDI	mzData	mzXML	mzML	pipeline	library	Win	Linux	Mac	language	visualization	algorithms
MaxQuant [16]	-	-	-	-	-	-	+	-	-	C#	1D, 2D, feature	feature detection, label-free quantitation, identification mapping, calibration
msInspect [12]	-	-	+	-	+	+	+	+	+	Java, R	1D, 2D, feature	signal processing, feature detection, label-free quantitation, MRM quantitation, alignment, identification mapping
MZmine [14, 66]	+	+	+	+	-	-	+	+	+	Java	1D, 2D, 3D	signal processing, alignment, normalization, statistics
ProteoWizard [109]	-	-	+	+	-	+	+	+	+	C++	-	signal processing
SpecArray [13]	-	-	+	-	+	-	-	+	-	C	2D	signal processing, feature detection, label-free quantitation, alignment, statistics
SuperHirn [15]	-	-	+	-	-	-	-	+	+	C++	-	signal processing, feature detection, label-free quantitation, alignment, identification mapping, normalization, statistics
TPP [10]	-	-	+	-	+	-	+	+	+	C, C++	2D	peptide validation, protein assignment, protein validation, ICAT quantitation, iTRAQ quantitation
Viper [113]	-	+	+	-	-	-	+	-	-	VB, C++	2D, feature	feature detection, calibration, alignment, identification mapping
XCMS [11, 108]	+	+	+	-	+	+	+	+	+	R	1D, 2D	feature detection, feature linking, alignment, spectral library search, statistics
OpenMS [82, 68]	+	+	+	+	+	+	+	+	+	C++	1D, 2D, 3D, feature, consensus	signal processing, feature detection, MRM quantitation, iTRAQ quantitation, ICAT quantitation, label-free quantitation, calibration, identification mapping, alignment

Table 5.1.: Overview of mass spectrometry analysis software tools and libraries.

Portability: To ensure wide acceptance of a software framework, it should be usable on all major operating systems. OpenMS can be used on most Linux distributions, MacOS and Windows. This portability is achieved by using ANSI C++ [116] combined with the build tool CMake [54]. CMake creates project files for all major compilers and platforms. In the OpenMS project it is used to create Makefiles for Linux systems, Makefiles or Xcode [117] project files for MacOS and MSVC [118] project files for Windows.
OpenMS is distributed as source and binary packages. The source package can be used to compile OpenMS on nearly all modern platforms and compilers. It is mainly used by developers. For users which are not interested in software development, binary packages are provided. Using the binary packages, the applications that come with OpenMS can be easily installed and used on Windows, MacOS and Ubuntu Linux.

5.3. Overall architecture

The overall architecture of OpenMS is shown in Fig. 5.1. The general structure is very simple—applications can be implemented using OpenMS, which in turn relies on several external libraries: Qt [119] provides visualization, database support and a platform abstraction layer. Xerces [120] allows XML file parsing. The library libSVM [103] is used for machine learning tasks, CoinMP [121] for linear and integer programming optimization. The Computational Geometry Algorithms Library (CGAL) [122] provides data structures and algorithms for geometric computation. The GNU Scientific Library (GSL) [77] is used for different mathematical and statistical tasks. Finally ANDI/MS [104] and NetCDF [123] are needed for support of the file format ANDI/MS. These libraries are from now on referred to as *contrib* libraries.

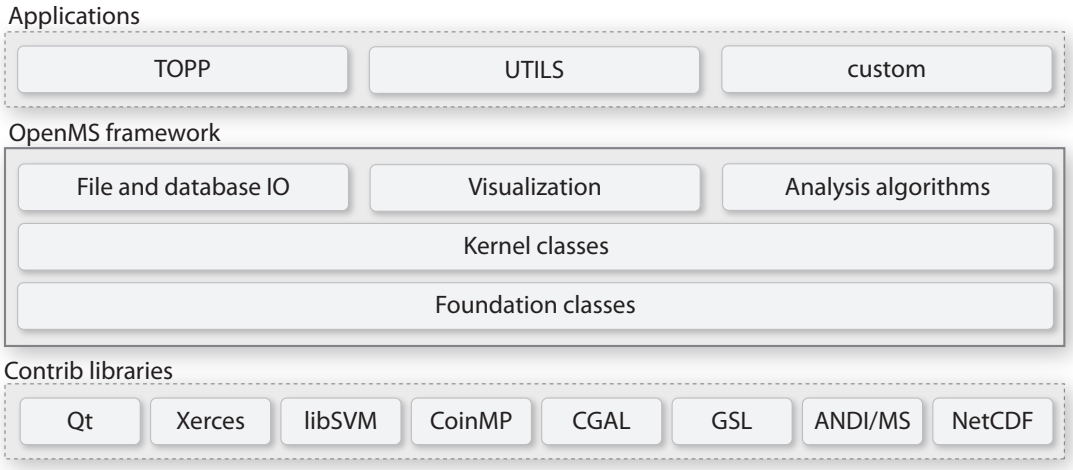


Figure 5.1.: Overall architecture of OpenMS.

OpenMS itself can be subdivided into several layers. At the very bottom are the foundation classes which implement low-level concepts and data structures. They include basic concepts (e.g., factory pattern, exception handling), basic data structures (e.g., string, points, ranges) and system-specific classes (e.g., file system, date, time). The kernel classes which capture the actual MS data and meta data, are built upon the foundation classes. Finally, there is a layer of higher-level functionality that relies on the kernel classes. This layer contains database I/O, file I/O, visualization and all analysis

algorithms.

5.4. Foundation classes

This section will briefly describe the most important foundation classes, which are the basis for the kernel classes and all higher-level functionality of OpenMS. First, very basic data structures are described which are needed by many classes and applications (strings, file system, etc.). The second part of this section focuses on classes that are often needed by algorithms, i.e., progress logging, parameter handling and the abstract factory pattern.

5.4.1. Basic data structures

The lack of convenient implementations for some very basic data structures is a major drawback of the programming language C++. For example, the string implementation provided by the STL [124] is quite rudimentary. Thus, the following data structures were implemented in order to facilitate common programming tasks:

String: The OpenMS *String* class is derived from the STL *string* to make interfacing with part of the contrib libraries more convenient. The *String* class extends the interface of its base class by conversion methods from/to integers, floating-point numbers and Qt strings; string manipulations methods for prefix, suffix and substrings; whitespace handling and trimming; and concatenation methods for many types.

Date and time: C++ does not implement date and time handling classes. Instead, methods of the C programming language have to be used. However, the Qt library offers platform-independent classes for this purpose. *Date* and *DateTime*, the OpenMS classes for date and time handling, are derived from the corresponding Qt classes and slightly extended by some convenience functions.

Lists: Arrays of strings, integers or floating-point numbers are frequently used data structures. The STL template class *vector* can be used and provides an efficient implementation. OpenMS contain such array implementations based on the STL and extends them with more convenient constructors (e.g., from comma-separated strings) and other methods.

5.4.2. Basic file system classes

Platform-independent file and directory handling is a prerequisite for portable applications. OpenMS implements a *File* class for searching files, checking file properties and handling file paths. *File* is based on Qt file and directory classes.
Monitoring changes of existing files is also frequently needed, for example to trigger updates in viewer applications. The class *FileWatcher* uses Qt classes and the *signals and slots* mechanism to provide this functionality.

5.4.3. Progress logging

Even very efficient algorithms working on MS data often have quite long runtime, because of the huge amount of data that are being processed. Thus, progress logging is an important feature for the usability of many applications. OpenMS offers a *ProgressLogger*

class which allows progress reporting for tasks with a known or unknown number of processing steps. Progress is reported either as text to the command line or as a progress dialog for GUI applications. The following code snippet demonstrates the use of the class in a command line application:

```
ProgressLogger logger;
logger.setLogType(ProgressLogger::CMD);
logger.startProgress(0,10,"Starting progress");

for (UInt i=0; i<=10; ++i)
{
  //... time-consuming task
  logger.setProgress(i);
}

logger.endProgress();
```

5.4.4. Factory classes

OpenMS makes use of various design patterns for object-oriented programming [49], for example Singleton, Facade, Composite and Iterator. A variation of the *factory method* pattern is used very frequently and thus it is explained now in more detail.
In general, factory pattern is used to provide a means of constructing objects of a certain family without specifying their concrete type. In OpenMS this technique is frequently used to construct different variants of the same algorithm. This allows changing the algorithm used for a certain task at run-time. Fig. 5.2 shows the OpenMS factory implementation.

The factory base class is implemented by the template class *Factory*. The template parameter T of the factory is the base class of the factory products. The factory provides all methods for registering different factory products and creating instances of the products. In our example the concrete factory instance *Factory<MapAlignmentAlgorithm>* creates products derived from the class *MapAlignmentAlgorithm*. *MapAlignmentAlgorithm* defines the interface of all subclasses and registers all derived classes with the factory. The actual factory products need to implement the methods *create()* and *getProductName()* for the registration of the products in the factory.

A common problem of factory patterns is that different factory products need different sets of parameters. Individual interfaces of the different classes are not possible, because only the interface of the base class is accessible after creating an object using the factory. Thus, the factory product base class has to offer a generic mechanism for parameter handling. The OpenMS classes for parameter handling, which solve this problem, are explained in the following section.

5.4.5. Parameter handling

The behavior of most algorithms is controlled by parameters. OpenMS offers many algorithms for different tasks, which is why parameter handling is an important feature of the library. Parameters are represented as tuples consisting of name, type and value. Parameters can be arranged in a tree structure, because some algorithms can have many parameters, which would make a simple parameter list confusing. The string keys used to access parameters can contain colons to denote hierarchy levels—similar to the

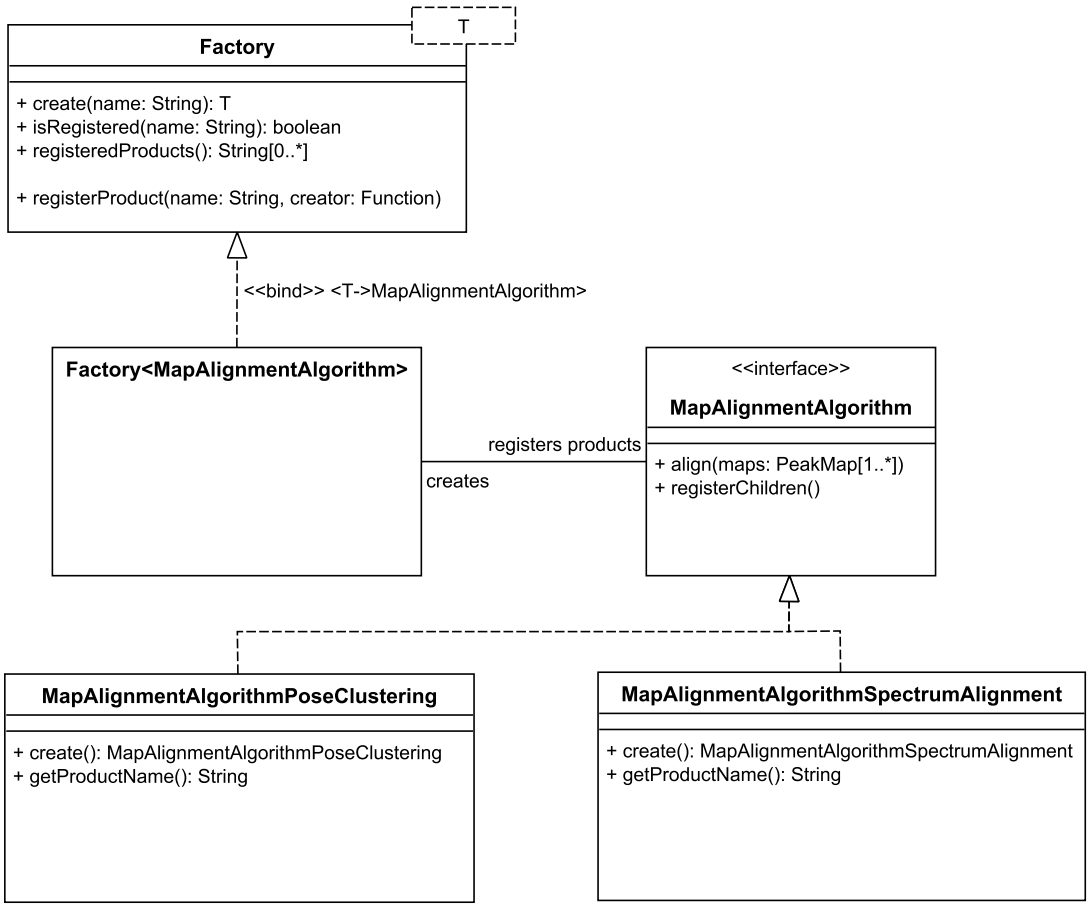


Figure 5.2.: UML class diagram of the *factory* implementation in OpenMS.

slashes used in file paths. For further categorization, each parameter can be tagged with arbitrary strings. These string tags are for example used to distinguish the basic parameters of an algorithm from advanced parameters.

Fig. 5.3 shows the main parameter handling classes. The *Param* class is the central class. It implements a *facade pattern*, i.e., it provides a unified high-level interface to the parameter data structure, which is made up of several classes. The *Param* class provides methods for

- setting and getting single parameters,
- manipulation of whole subtrees,
- iteration over all parameters,
- tagging parameters,
- setting descriptions of parameters,
- setting restrictions of parameters and
- loading/storing the parameters to/from XML files.

The tree structure is implemented by a class for internal nodes and a class for leaves—*ParamNode* and *ParamEntry*, respectively. Both classes store a name and description. The leaves additionally contain assigned tags, restrictions and the values assigned to the parameter. The value is stored in the class *DataValue*, which can efficiently store data of

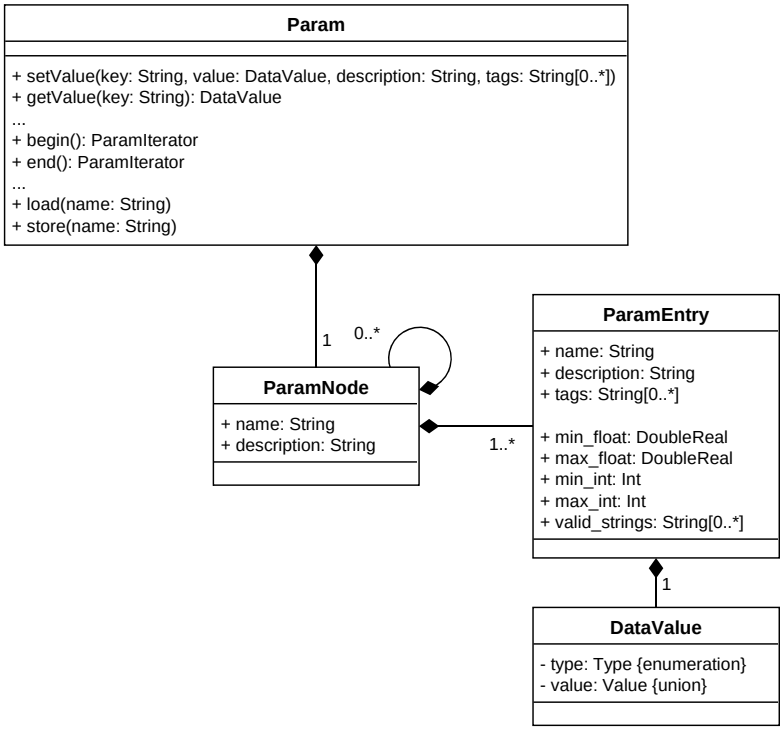


Figure 5.3.: UML class diagram of the generic parameter handling in OpenMS.

different types using a *union* of different data structures. It supports integers, floating-point numbers and strings, as well as arrays of these three basic types. The numerical parameters can be restricted to intervals by settings a minimum and maximum value. The string parameters can be restricted to a set of valid strings. The following code example shows how parameters can be set and stored:

```
1 Param param;
2 param.setValue("debug","true");
3 param.setValue("mass_trace:mz_tolerance",0.1);
4 param.setValue("isotope_pattern:mz_tolerance",0.14);
5
6 param.store("parameters.xml");
```

In the example three parameters are set. The string parameter *debug* is located in the root level of the parameter tree. The *mz_tolerance* parameters are located in two different subsections. In line 6 the parameters are stored in ParamXML format. The XML representation of the parameters in this format looks like that:

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <PARAMETERS version="1.3">
3   <ITEM name="debug" value="true" type="string" description="" />
4   <NODE name="mass_trace" description="">
5     <ITEM name="mz_tolerance" value="0.1" type="float" description="" />
6   </NODE>
7   <NODE name="isotope_pattern" description="">
8     <ITEM name="mz_tolerance" value="0.14" type="float" description="" />
```

```
  </NODE>
</PARAMETERS>
```

DefaultParamHandler: All OpenMS algorithms provide default parameters, which should work on most datasets. These default parameters are typically managed by the class *DefaultParamHandler*. It holds two *Param* instances, one for the defaults and one for the currently used parameters. Knowing the defaults allows validating parameters that are assigned to an algorithm. For each parameter the following checks are performed:

- if the parameter name exists in the defaults,
- if the parameter type is equal to the default type and
- if the value restrictions are fulfilled.

Besides handling the defaults, *DefaultParamHandler* also helps keeping member variables synchronized with certain parameters. This is needed in time-critical parts of algorithms, where the overhead of searching the parameter in the tree structure is not acceptable. In this case, synchronized member variables with constant access time are used. The member variables are simply updated every time the parameters change. This is achieved by a member function of *DefaultParamHandler* which the derived class has to re-implement.

TOPPBase: The last important parameter handling class described in more detail is *TOPPBase*. It is a base class for applications based on OpenMS and handles command line parameters. The command line parameters are registered during the initialization of the application. The registration includes parameter names, value type, default value, restrictions, requirement level and description. Given this information, *TOPPBase* can do the parsing of command line parameters automatically, before any other code is executed. In case of an error, a detailed description of problem is automatically generated and the execution is aborted. The documentation of command line parameters, which is invoked with the special parameter *--help*, is automatically generated as well. This kind of automation is very convenient for programmers and users. Programmers do not need to implement parameter handling for each application, which can be time-consuming. The users benefit from this approach in two ways. All tools share a similar user interface and the documentation cannot easily deviate from the functionality.

5.5. Data reduction and kernel classes

The kernel classes are the foundation of all high-level functionality of OpenMS. They hold the actual MS data and meta data about the data. In this section, the design of the kernel classes is described and the concept of data reduction used in OpenMS is introduced.

5.5.1. Data reduction

Data reduction is a central concept of OpenMS and will now be considered in more detail. Data reduction comprises two transformation steps as displayed in Fig. 5.4: The conversion of profile spectra to centroided spectra and the conversion of a profile or centroided map (a collection of MS spectra produced by an LC-MS experiment) to a feature map. By *feature* we denote the two-dimensional signal created by some chemical entity, e.g., a peptide or metabolite. A feature is characterized by its isotopic pattern in

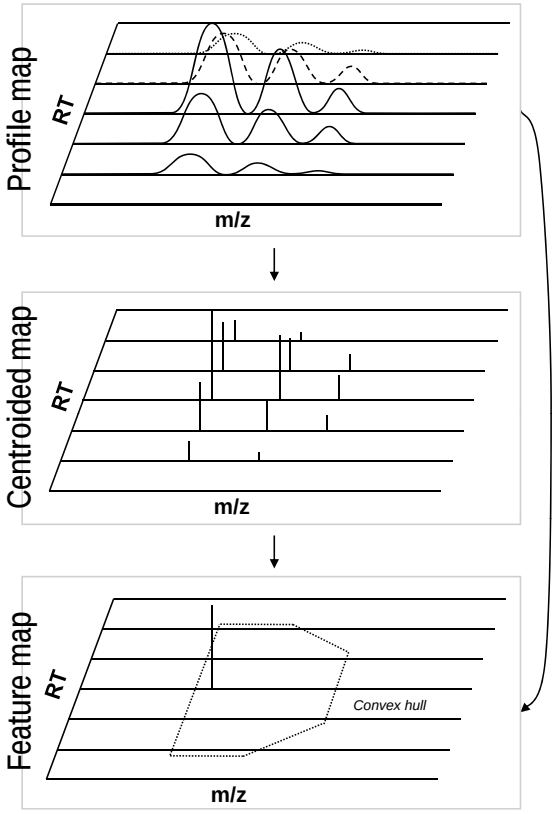


Figure 5.4.: The data reduction concept of OpenMS. A profile map (top) which is first reduced to a centroided map (middle) and finally to a feature map (bottom). The direct reduction of a profile map to a feature map is also possible.

mass-to-charge dimension and by the elution profile in retention time dimension.
Each of these reduction steps reduces the amount of data by roughly two to four orders of magnitude. It is this kind of reduction which makes the application of sophisticated algorithms with high runtimes in up-stream analysis steps possible. On the other hand, information is lost in each reduction step. Special care has to be taken to retain as much information as possible, while reducing the overall amount of data. This tradeoff is reflected by the flexibility of the OpenMS kernel data structures.

5.5.2. Peak data

The kernel data structures store raw MS data as well as the results of the data reduction steps, thus they are the very basis of each OpenMS application. First, we will have a closer look at the kernel classes for peak data.

Peak maps are stored as a set of spectra, which in turn hold the individual peaks (see Fig. 5.5). The peak type of maps and spectra is customizable through the use of template code. Different peak types are available depending on the amount of information required for a specific use case. Profile spectra are stored using *Peak1D*, which holds only an intensity value and an m/z position. For centroided spectra *RichPeak1D* can be used, which stores additional meta information, for example a signal-to-noise value calculated from the profile data. If these predefined peak types are not sufficient, custom peak types can be used as long as they provide the proper interface. This kind of customization

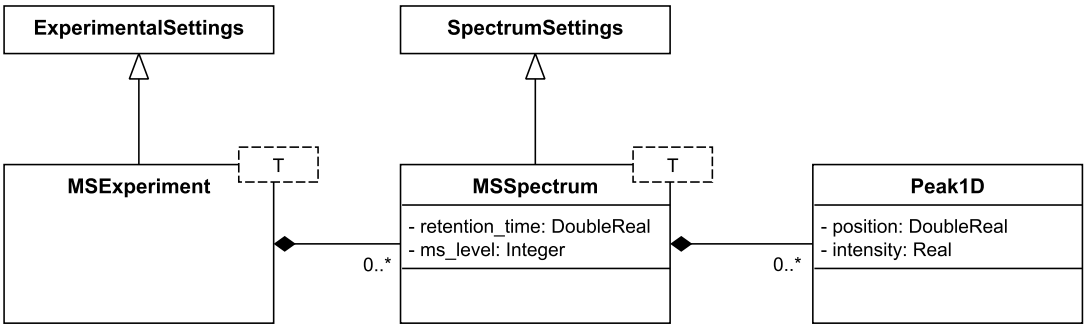


Figure 5.5.: UML class diagram of map, spectrum and peak data structures.

allows a very flexible and efficient use of the kernel classes. A 115 MiB mzData file containing approximately 2.1 million data points can be stored in 27 MiB of RAM on a 32 bit computer.
In addition to peak data, maps and spectra also store meta information about the experiment and spectrum acquisition, respectively. More information about the meta data can be found in Section 5.5.4.

To make the use of theses template classes more convenient, several type definitions are provided for maps and spectra: *PeakMap* and *PeakSpectrum* are the standard types using *Peak1D* as peak type. *RichPeakMap* and *RichPeakSpectrum* allow storing of meta data through the use to *RichPeak1D* as peak type.

Maps and spectra are not mere container classes. They provide a lot of convenience functionality for sorting and iterating. The following listing shows how one can iterate over an area of the map using a special iterator:

```
1 PeakMap map;
2 //... fill the map with data
3 map.sortSpectra(true);
4
5 PeakMap::AreaIterator it = map.areaBegin(3000.0,6000.0,500.0,800.0);
6 for (; it!=map.areaEnd(); ++it)
7 {
8   cout << it.getRT() << " " << it->getMZ() << endl;
9 }
```

In lines 1 to 3 the map is created and the spectra are sorted with respect to RT. The boolean parameter of the *sortSpectra* method enables sorting of the peaks according to m/z. Sorted data is a prerequisite for the area iterator. In line 5 the *AreaIterator* is initialized with the intervals 3000–6000 s in RT dimension and 500–800 Th in m/z dimension. Finally, in lines 6 to 9 the peak position of each peak in the selected area is printed to the command line.

5.5.3. Feature data

Feature maps are produced from profile or centroided peak maps as the result of *feature detection*. The feature detection step groups all peaks belonging to one chemical entity to one feature. Just as a peak, a feature is characterized by a position in m/z dimension

and an intensity value. In addition, a position in RT dimension is needed, because feature maps contain no information about spectra. However, this position information alone would not be sufficient to reconstruct the feature creation. Thus, the feature also contains a convex hull of all peaks that contributed to the feature (see Fig. 5.4). If available, convex hulls of the individual mass traces can also be added.

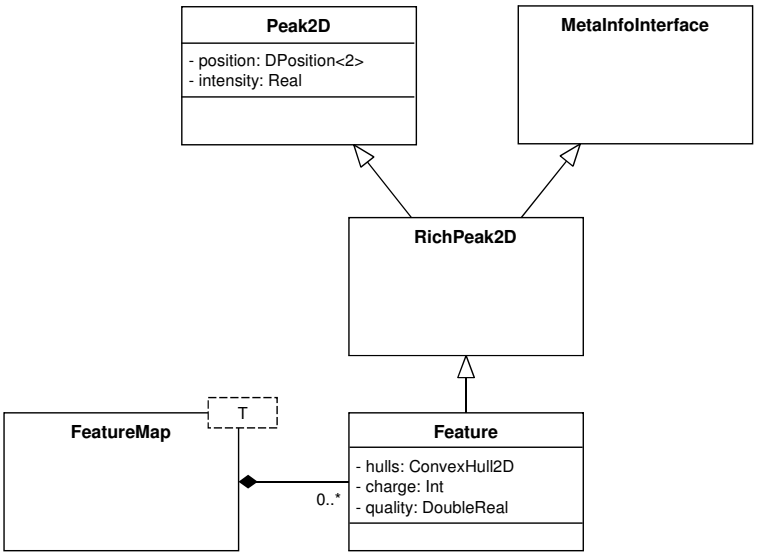


Figure 5.6.: UML class diagram of the feature data structures.

Fig. 5.6 shows the *FeatureMap* class and the *Feature* class along with its base classes. Just as for the spectrometric peaks, base classes with different levels of detail are available for peaks with two-dimensional peaks. *Peak2D* is the most basic one. *RichPeak2D* extends it with meta data. The *Feature* class adds convex hulls, quality, charge, etc. The container for the features, *FeatureMap*, is a template class just as the map and spectrum classes.
Besides peak and feature maps, there is a third map type (*ConsensusMap*) used for relative and absolute quantitation. Consensus maps aggregate several features from one or several feature maps to feature groups, so-called *ConsensusFeatures*. Using *ConsensusFeatures*, intensity ratios can be calculated for label-free or isotope-labeled experiments.

5.5.4. Meta data

The meta data spectra and maps are annotated with is a crucial part of MS data. Without this information data analysis is hampered, because the experiment design and instrumentation have a strong influence on the data processing pipeline. The importance of meta data is emphasized by the HUPO-PSI [125] effort to define the minimum information about a proteomics experiment (MIAPE [126]) which must be provided for publication of the data. Similar guidelines are enforced by all major journals in the field of proteomics.
As described in Section 5.5.2, the meta data is stored along with the peak data in the *MSExperiment* and *MSSpectrum* classes. In the following, the modeling of map and spectrum meta data is shortly described.

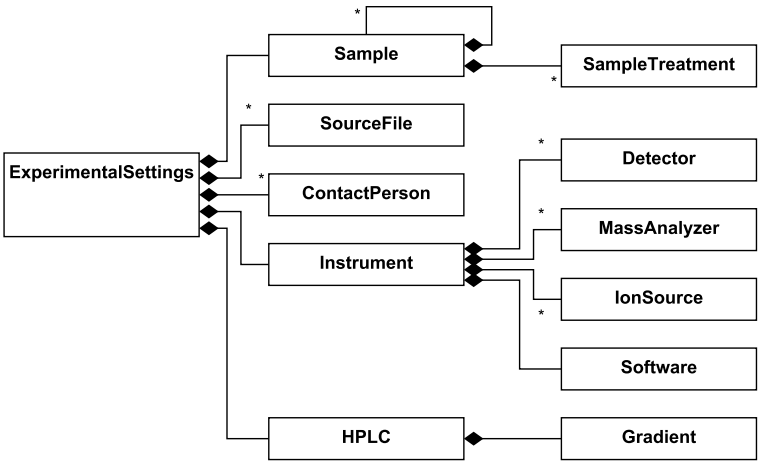


Figure 5.7.: UML class diagram of the peak map meta data.

Peak map meta data: Peak maps mainly contain meta data about the experimental setup—sample material, sample processing, instrumentation and operator are described. The class *ExperimentalSettings* is the container for these different pieces of information (see Fig. 5.7). The sample is described by a hierarchy of *Sample* instances with annotated sample treatment information. A single sample would not be sufficient in the case of stable isotope labeling experiments, where two or more biological samples are mixed.
As for the instrumentation, the mass spectrometer is described in the *Instrument* class. It may consist of several ion sources, mass analyzers and ion detectors. The acquisition software, which operates the instrument, can also be described. In the case of an LC-MS experiment the LC column and the used gradient can be given as well.
Another important piece of information are contact persons. Here, the operator of the instrument or the person who did the computational analysis of the data can be listed. Finally, one or several source files can be given from which the data was obtained.

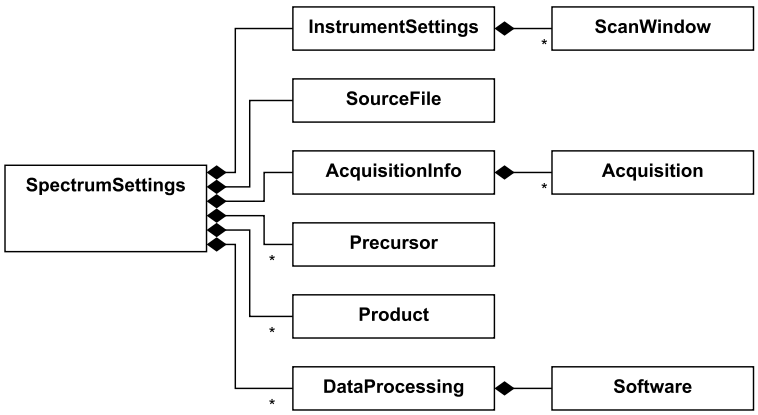


Figure 5.8.: UML class diagram of the spectrum meta data.

Spectrum meta data: Spectrum meta data mainly covers the acquisition of the data and the processing applied. Instrument settings which can vary between scans are stored in *InstrumentSettings*. This covers scan polarity, scanning method, scan windows, etc. Several special scanning methods, e.g., MS/MS scans, require additional information about the fragmentation of ions. This information is stored in the *Precursor* and *Product* classes. *AcquisitionInformation* describes the raw micro-scans performed by the instrument. These micro-scans are processed by the instrument software in order to create a single spectrum.
The second focus of the spectrum settings is data processing. Each processing step applied to the spectrum is documented as a single *DataProcessing* instance, which contains information about the software that performed the processing, the applied parameters and the completion time. This information about data processing is of utmost importance for traceability and repeatability of data processing protocols.

5.6. File and database I/O

Standardized data exchange formats are especially important because they allow seamless integration of different software tools into a single analysis pipeline. Therefore, OpenMS supports the most important non-proprietary file formats. Proprietary file formats of instrument vendors are not supported because most of the required access libraries are available for Windows only and are not freely distributable.

Format	version	reading support	writing support
ANDI/MS	1.0	yes	no
mzData	1.0.5	yes	yes
mzXML	2.1 / 3.0	yes	yes
mzML	1.1.0	yes	yes
DTA	-	yes	yes

Table 5.2.: Peak data file formats supported by OpenMS.

Peak data files: OpenMS supports all standard formats for peak data. ANDI/MS [104], the fist official data exchange format for MS data is out of date, but still used by legacy software tools. It was replaced by two modern XML formats, which were developed in parallel. The HUPO-PSI mass spectrometry work group released the mzData [105] format in the year 2004. In the same year the Institute of Systems Biology (ISB) presented the mzXML format with a very similar scope as the mzData format. Because both formats were—and are still—widely used, software developers have to support mzData and mzXML.
The existence of two standard formats was considered a barrier for adoption, especially by instrument vendors. Therefore, in 2006 the merge of the two formats was announced. Version 1.0 of the new mzML [107] format was released in 2008, but was not widely adopted due to several flaws in its design and missing validation tools. After the official release of mzML the OpenMS project started implementing it and actively contributed to the format development. The revised version 1.1.0 of mzML was finally released in June 2009, replacing all other formats.
Despite the fact that there is only one community-supported format left, there are still many useful applications which do not support mzML. Thus, OpenMS has to support

as many formats as possible, in order to make the integration of external tools possible. Table 5.2 lists the most important file formats supported by OpenMS.
The following listing demonstrates how easily one can convert between mzData and mzML format using the respective OpenMS classes:

```
1 PeakMap map;
2 MzDataFile infile;
3 infile.load("example.mzData",map);
4 MzMLFile outfile;
5 outfile.store("example.mzML",map);
```

In line 1 the data structure that holds the peak data is created. In lines 2 and 3 the input file adapter is instantiated and used to fill the *map* variable with the data from an mzData file. In lines 4 and 5 the data is written to an mzXML file using the corresponding file adapter.

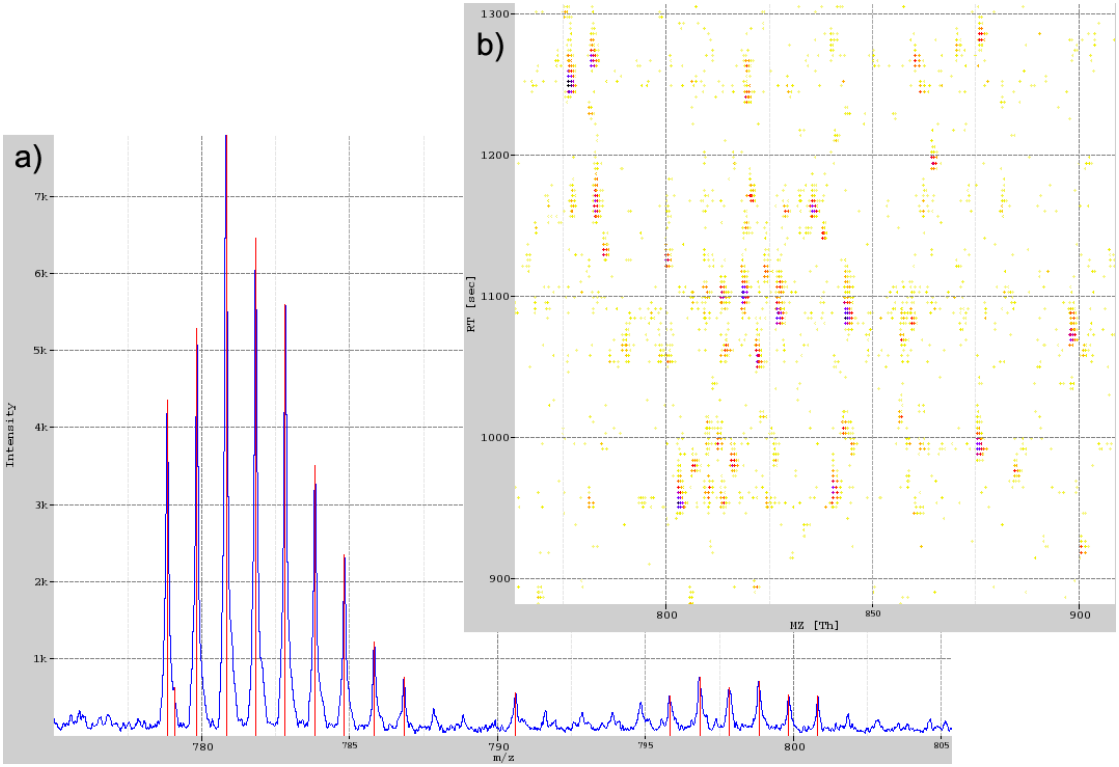


Figure 5.9.: Visualization widget examples. (a) Visualization of a profile spectrum and the corresponding peaks spectrum as two superimposed layers. (b) Part of an LC-MS map displayed in a 2D view.

The mzML format: Version 1.1.0 of the mzML format addresses many shortcomings of the previous standard formats mzData and mzXML. The main improvement is the possibility to store chromatogram data, which is indispensable for new MS techniques such as *multiple reaction monitoring* (MRM).
The second main innovation of the mzML format is a very rich *controlled vocabulary* (CV) which allows the annotation of MS data with essential meta data. The CV covers

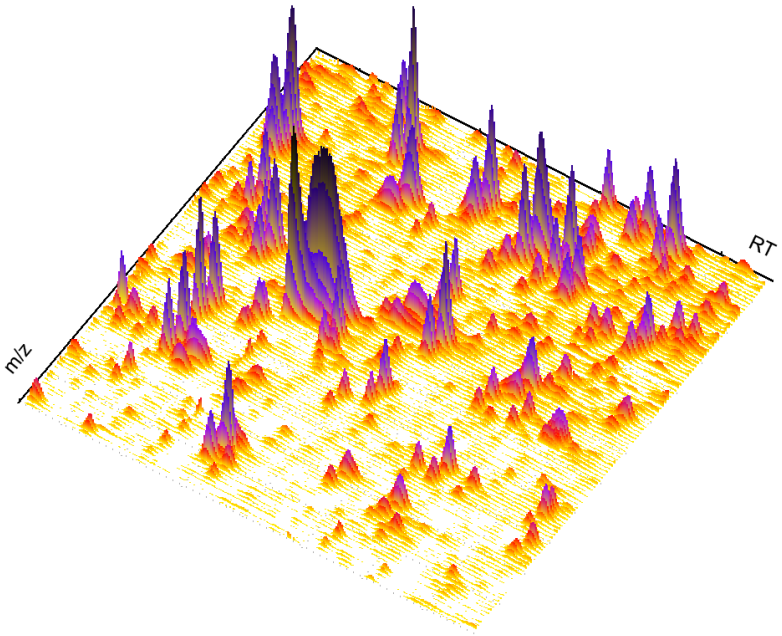


Figure 5.10.: Example of the 3D map visualization provided by OpenMS. A small region of a proteomics experiment is shown.

experimental setup, MS instrumentation and information about spectra. We will now give two examples of its use: It is possible to document all applied data processing steps, which could not be done with previous formats. The support for hybrid instruments, such as LTQ-Orbitrap, was improved by supporting different spectrum types in one experiment.
Another important point to mention is the involvement of several MS instrument vendors. This will ensure that mzML export will be supported by most instrument software in the future.

However, the use of a CV for annotation of meta data also has disadvantages. Files in mzML format cannot be entirely validated by standard XML tools. The validation against an XSD schema can only check the general structure. The correct use of CV terms in an XML element has to be checked in a second step, the *semantic validation*. Thus, OpenMS offers data structures and algorithms for the validation of mzML and other PSI formats. For the semantic validation a CV definition file and a *mapping file* are needed. The *mapping file* defines which CV terms are allowed for a certain XML element and which are of those terms are mandatory.

Identification and quantitation formats: As there are no official standard file formats for quantitation and peptide identification data, we created our own formats for these tasks (featureXML and idXML). Support for the identification formats pepXML and protXML [10] developed by the ISB is planned for the future.
Eventually, our formats will be replaced by standard formats released by the HUPO-PSI. Currently, we are actively contributing to the development of the upcoming HUPO-PSI standard mzIdentML, which captures the results of peptide and protein search engines.

Database I/O: Currently, most mass spectrometry tools operate on files. Because of the constantly growing data volume created by LC-MS experiments, database systems will become increasingly important for data management. Therefore, we developed a database adapter that can persistently store the kernel data structures in an SQL database. Through the use of Qt as an additional layer of abstraction, the implementation works with most SQL-compliant relational database management systems including MySQL, PostgreSQL, ORACLE and DB2. The interface of the database adapter is very similar to the file adapters, which is why we omit an example.

5.7. Visualization

Graphical user interfaces are important in many aspects of data analysis. In general, they improve the usability of software, especially for users not experienced in command line use. In scientific visualization GUIs are used to increase the accessibility of information. Different visualizations can give a quick overview and deep insights into huge amounts of data.
OpenMS mainly provides widgets for comprehensive visualization of peak data, meta data and algorithm parameters. Additionally, many other widgets and dialogs are provided for convenience, e.g., a widget for creating gradients of multiple colors.

5.7.1. Peak data visualization

Visual inspection is an important data analysis tool for MS data. It can instantly reveal properties of the data that would go unnoticed using command line tools. Errors during separation or polymeric contamination of the sample can, for example, be easily noticed during visual inspection of an LC-MS map. OpenMS provides efficient widgets for visualization of single spectra or whole peak maps. A single spectrum is displayed by a standard plot of profile or centroided data. Peak maps are displayed either in a 2D view from a bird's eye perspective with color-coded intensities or in a 3D view. Fig. 5.9 shows an example of the spectrum and the 2D map view. An example of the 3D view can be found in Fig. 5.10.

All visualization widgets are intended for interactive exploration of the data. They allow zooming into the data, measuring distances between data points and filtering the displayed data (intensity cutoff and meta data filters). Another important feature is customizability. Through the settings dialog provided by each view, the user can customize many properties such as colors, icons and line widths.

In order to display two given spectra as shown in Fig. 5.9 (a) the following code could be used:

```
1  PeakMap map1,map2;
2  //... fill the maps with data
3  Spectrum1DWidget* gui;
4  gui = new Spectrum1DWidget(Param());
5  gui->canvas()->addLayer(map1);
6  gui->canvas()->addLayer(map2);
```

In line 1 two maps are created which should be filled with data in a real-world application but remain empty in this toy example. In lines 3 and 4 the widget used for visualization is created with an empty set of parameters. Therefore, default parameters are used for peak color, background color, etc. Finally, both maps are added to

the canvas subwidget. Handing over a map to the widget, although it displays only one spectrum, is necessary, because all spectrum and map visualization widgets share a common base class and thus a common interface.

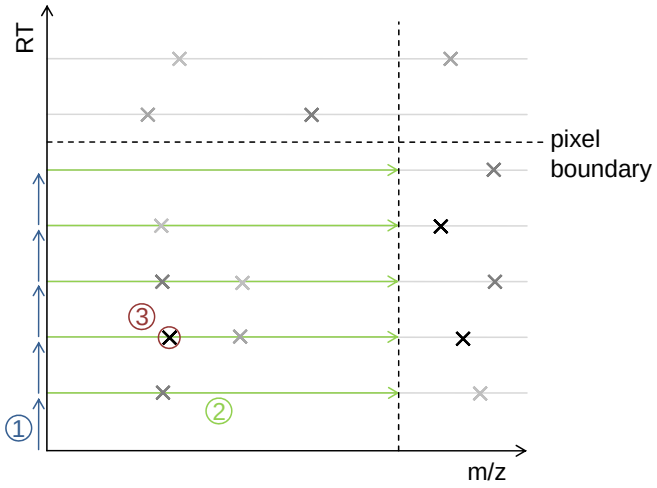
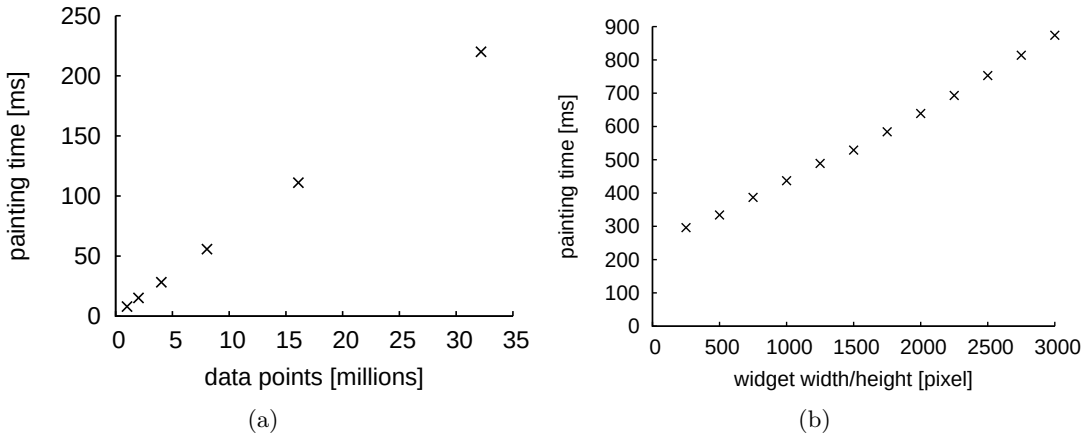


Figure 5.11.: Illustration of the pixel-oriented painting: (1) First, the spectra which lie inside a row of pixels are determined. (2) Intensities of all peaks that lie inside the pixel are checked. (3) Finally, the highest intensity is assigned to the pixel. The procedure is repeated for all pixels of the row, then the next row is processed.

Visualization of large datasets

As the resolution of modern MS instruments is quickly improving, the amount of produced data per run is growing rapidly. Today, a single dataset can already contain several hundred million raw data points, which corresponds to several GB file size. Visualizing datasets of this size poses a big challenge. Especially for the 2D view, which is typically used to display whole datasets, fast visualization techniques had to be developed. The naive approach, painting each data point to the screen, is too slow for large datasets. In our approach, the data is partitioned into rectangles that correspond to one pixel on the screen. For each rectangle the maximum intensity is determined and painted onto the screen. Fig. 5.11 illustrates the mapping of data to pixels. This approach restricts the number of transformations from data coordinates to screen coordinates and the number of color calculations to the number of pixels. Iterating over all displayed data points is done in $O(n)$, where n is the number of displayed data points (see Fig. 5.12(a)).

The downside of the pixel-oriented approach is that the runtime depends on the number of pixels (see Fig. 5.12(b)). It is slow, compared to the naive approach, when very few data points are displayed. Therefore, a heuristic was implemented which switches between pixel-oriented painting and the naive approach. As determining the exact number of data points on the screen requires $O(n)$ steps, an estimation of the number of displayed data points is used. First, the number of spectra contained in the visible RT range is determined. Then, the number of displayed peaks of the spectrum closest to the center of the RT range is determined. The overall number of displayed peaks is estimated by multiplication of these two values. Naive painting is used, when the number of displayed data points is less than a fourth of the number of pixels. Switching between



(a) (b)

Figure 5.12.: (a) 2D view painting time plotted against the number of displayed data points. (b) 2D view painting time plotted against the widget size in pixels (square widget). The test dataset contained 65 million data points and had a file size of 576 MiB in mzData format.

painting modes ensures good painting performance for large amounts of data as well as very fast painting for few data points.

The actual painting of the data is done on a buffer, which is copied to the screen. The buffer is needed for user interaction with the data, i.e., for highlighting selected data points and measuring distances between data points. Highlighting the selected peaks is possible without repainting, by simply copying the buffer to the screen and painting the highlighting on top. Repainting the buffer is only necessary when the displayed data range or the data itself changes.

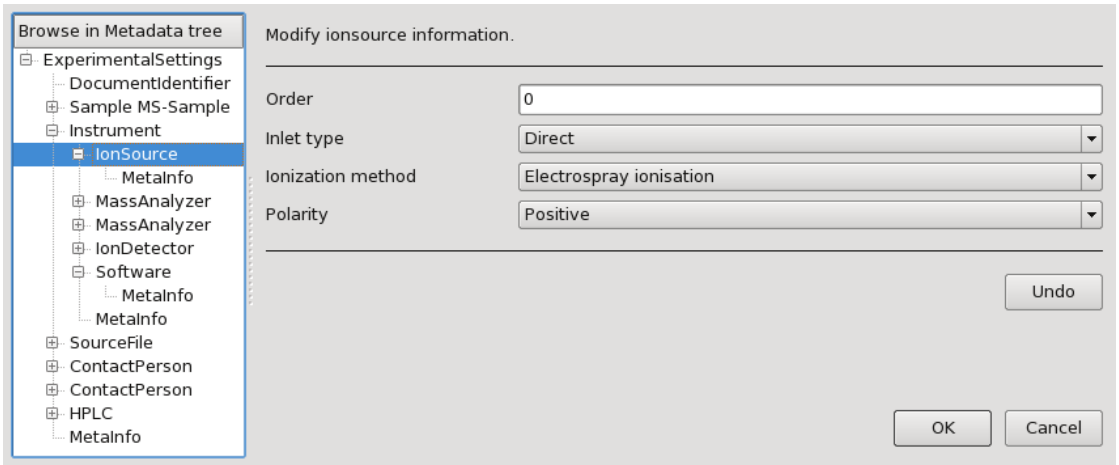


Figure 5.13.: Meta data visualization of a peak map.

5.7.2. Meta data visualization

As discussed in Section 5.5.4, meta data is an important part of MS data. Editing meta data is frequently required for publication of the data, data management, etc. In

OpenMS the *MetaDataBrowser* class is used to visualize and edit meta data. Fig. 5.13 shows the visualization of an *ExperimentalSettings* object. On the left, the tree of meta data objects can be browsed. When an object is selected, its properties can be edited on the right side. In the shown example the ion source of the MS instrument is selected. The corresponding properties—ionization method, polarity, etc.—are shown on the right.

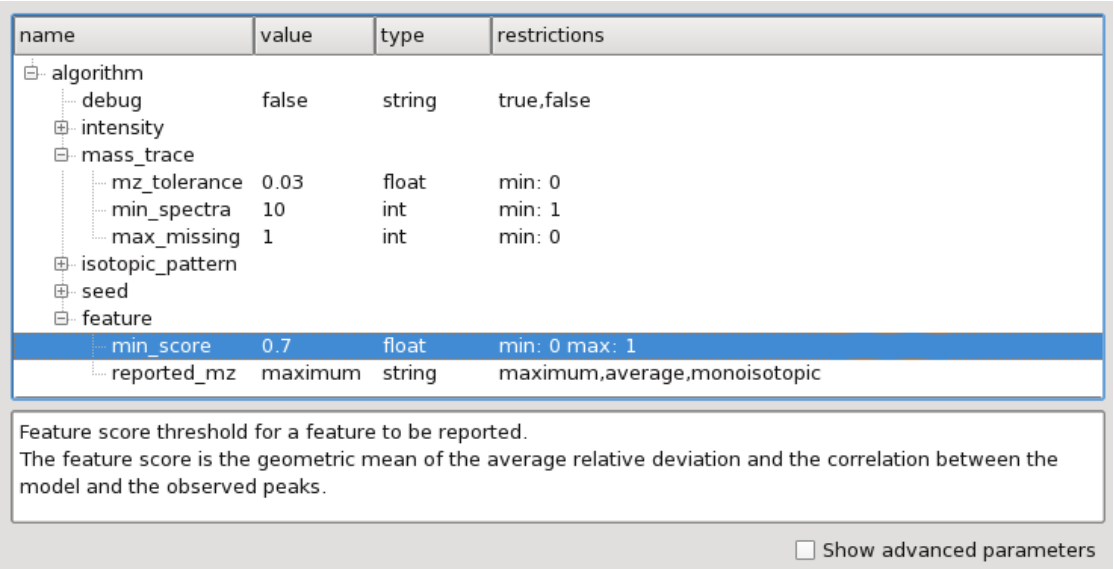


Figure 5.14.: Algorithm parameters of the feature detection algorithm for centroided data.

5.7.3. Parameter visualization

Another frequently used widget is *ParamEditor*, which displays *Param* objects (see Fig. 5.14). It displays the parameter names as a tree structure in the first column. Parameter value, type and possible restrictions are shown on the right of the parameter name. In the text box on the bottom, the documentation of the currently selected parameter is shown. The checkbox below the documentation can be used to display additional advanced parameters.
This very general design makes *ParamEditor* a versatile tool. It is used in several applications. It is the central widget of the TOPP tool *INIFileEditor*, providing a stand-alone editor for parameter files. Moreover, it is integrated in *TOPPView* and *TOPPAS*, a pipeline editor for TOPP pipelines.

5.8. Analysis algorithms

The parts of OpenMS we described so far—foundation classes, kernel classes, file I/O and visualization—form the scaffold used to build higher-level functionality like analysis algorithms. OpenMS offers a wide range of algorithms covering data reduction (see Section 5.5.1), map alignment and protein/peptide identification. In the following sections, selected algorithms of OpenMS are presented using small code examples.

5.8.1. Signal processing

OpenMS offers several filters to reduce chemical and random noise as well as baseline trends in MS measurements. Profile spectra may either be denoised by a Savitzky-Golay filter or a peak-area-preserving Gaussian low-pass filter. Both smoothing filters are commonly used and recommended for spectrometric data [127, 128]. For the baseline in MS experiments, no universally accepted analytical expression exists. Hence, we decided to implement a nonlinear filter, known in morphology as the top-hat operator [129]. This filter does not depend on the underlying baseline shape and its applicability to MS measurements has already been shown in [130]. For extraction of the accurate information about the mass spectral peaks in a profile spectrum we developed an efficient peak picking algorithm [64] that uses the multi-scale nature of spectrometric data. First, the peak positions are determined in the wavelet-transformed signal. Afterward, important peak parameters (centroid, area, height, full-width-at-half-maximum, signal-to-noise ratio, asymmetric peak shape) are extracted by fitting an asymmetric peak function to the profile data. In an optional third step, the resulting fit can be improved further by using techniques from nonlinear optimization. In contrast to currently established techniques, our algorithm yields accurate peak positions even for noisy data with low-resolution and is able to separate overlapping peaks of multiply charged peptides.

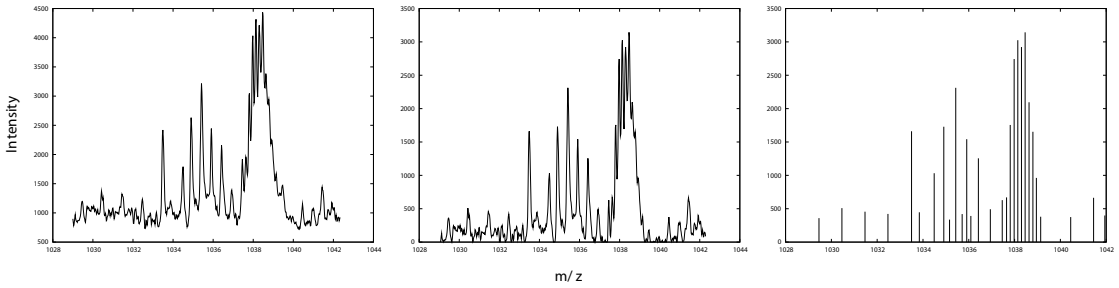


Figure 5.15.: Signal processing example on low-quality data: From the profile spectrum (left) the baseline is subtracted (middle) and then centroiding is performed (right).

The next code example demonstrates a small analysis pipeline consisting of a baseline reduction and a peak picking step:

```
1  PeakMap exp_profile;
2  //... fill the map with data
3  MorphologicalFilter mf;
4  mf.filterExperiment(exp_profile);
5
6  PeakMap exp_centroided;
7  PeakPickerCWT pp;
8  pp.pickExperiment(exp_profile, exp_centroided);
```

After filling the *PeakMap* the baseline is removed using the class *MorphologicalFilter*. Then, the peak centroids are detected and stored in *exp_centroided*. In this example no algorithm parameters are explicitly set, so the default parameters are used. Fig. 5.15 visualizes the effect of the above program on a single profile spectrum.

5.8.2. Feature detection and quantitation

Feature detection is a central concept in OpenMS. As noted before, a feature is characterized by its mass-to-charge ratio, the centroid of its elution curve and the signal area.

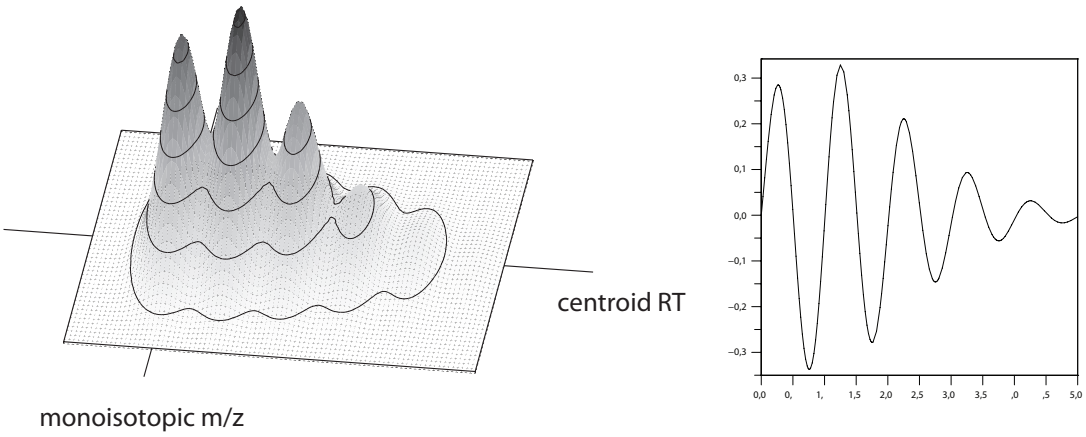


Figure 5.16.: The OpenMS feature model. The averagine model (left) and the corresponding isotope wavelet (right) used by our algorithm for mass 2000 Da and charge 1.

OpenMS includes several algorithms for the detection of peptide features in LC-MS data. The quantitation algorithm presented in Chapter 3 is available as the *centroided* algorithm. It is applicable to centroided data, and to medium- and high-resolution profile data after centroiding. For profile data with low resolution the *simple* and *isotope_wavelet* algorithms can be used.
All our approaches are based on a two-dimensional model. We use the averagine isotope model to approximate the amino acid composition for a peptide of a given mass. From this we can estimate its atomic composition and derive its isotope distribution in a mass spectrum [131]. Similarly, we approximate the elution curve by a Gaussian or exponentially modified Gaussian distribution [74]. The combined two-dimensional model can be seen in Fig. 5.16. In addition, our isotope pattern model takes different mass resolutions into account by incorporating a parameter for the width of the isotopic peaks in a feature.

Fitting the two-dimensional model is an expensive computational task, especially on profile data. Therefore, it is important to select the candidate regions carefully. Thus, we designed the *isotope_wavelet* algorithm [132, 70] that uses a hand-tailored isotope wavelet [73] to filter the mass spectra for isotopic patterns of a given charge state. The isotope wavelet explicitly models the isotope distribution of a peptide (see Fig. 5.16 right). This prefiltering results in a lower number of potential peptide candidates that need to be refined using the model fit. The next code example demonstrates how the feature detection can be used:

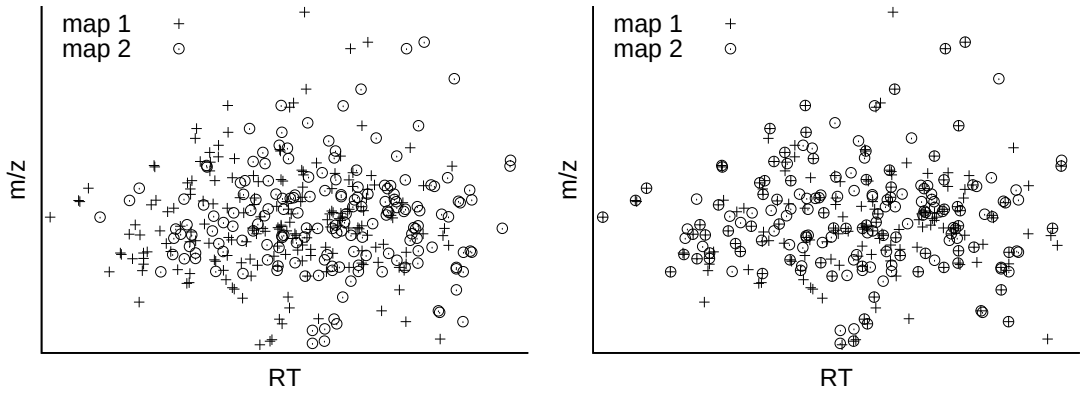
```
1 PeakMap exp_profile;
2 //... fill the map with data
3 FeatureMap<> features;
4 FeatureFinder ff;
```

```
ff.run("centroided", exp_profile, features, Param());
```

The experimental data are stored in a *PeakMap* and the extracted features in a *FeatureMap*. To quantify, we invoke the *run* method of the *FeatureFinder*. It is a template function, therefore, other data structures for input and output are supported as well. Various types of algorithms are available—in this case we choose *centroided*. As in the examples before, algorithm parameters are handed over in a *Param* object. Each algorithm has a different set of parameters, which can be read from a ParamXML file or set directly in the code.

5.8.3. Map alignment

An important step in a typical LC-MS analysis workflow is the combination of results from multiple experiments, e.g., to improve confidence in the obtained measurements or to compare results from different samples. In order to do so, a suitable mapping or *alignment* between the datasets needs to be established. The alignment has to correct for random and systematic variations in the observed elution times that are inevitable in experimental datasets.



(a) before alignment (b) after alignment

Figure 5.17.: Map alignment example. Left: Two feature maps with varying retention time and mass-to-charge dimensions. Right: The features of the second feature maps were transformed onto the coordinate system of the first feature map.

OpenMS offers algorithms to align multiple experiments and to match the corresponding ion species across many samples. A novel and generic algorithm was developed to correct for the variation of retention time and mass-to-charge dimensions between two maps. It uses an adapted pose clustering approach [133, 134] to efficiently superimpose peak maps as well as feature maps. In Fig. 5.17 two feature maps are shown. In the left plot the retention times and the mass-to-charge ratio of corresponding features vary extremely and corresponding ion species are hard to determine. However, after the mapping of the two feature maps onto a consistent coordinate system the correspondence between the two maps can easily be seen in the right plot.

To detect and combine corresponding features in multiple feature maps to a so-called *consensus map*, we developed an algorithm based on techniques from computational geometry. The superimposition algorithm and the algorithm for the determination of

a consensus map are combined to a star-wise approach for the alignment of multiple peak or feature maps. The overall methods are fast, reliable and robust, even in the presence of many noise signals and large random fluctuations of retention time. Details of the map alignment algorithms can be found in [135]. In the following code example three feature maps are aligned, corresponding features are grouped and the resulting consensus map is stored in a file:

```
1  vector< FeatureMap<> > maps(3);
2  //... fill the feature maps with data
3  MapAlignmentAlgorithmPoseClustering mla;
4  vector< TransformationDescription > trafos;
5  mla.alignFeatureMaps(maps, trafos);
6
7  ConsensusMap output;
8  FeatureGroupingAlgorithmUnlabeled fga;
9  fga.group(maps,output);
10
11 ConsensusXMLFile file;
12 file.store("output.xml", output);
```

After instantiating a vector of three feature maps in line 1 they are filled with data in line 2. In lines 3 to 5 a *MapAlignmentAlgorithmPoseClustering* object is used to correct the retention time distortions between the maps. Grouping of corresponding features is done in the lines 7 to 9. To store the resulting consensus map in consensusXML format we use the appropriate file handler in lines 11 and 12.

5.8.4. Retention time prediction

Peptide identification is an essential step in MS/MS data analysis. A major problem with existing identification routines lies in the significant number of false positive and false negative annotations. So far, standard algorithms for protein identification have not used the information gained during the separation processes usually involved in peptide analysis, such as retention time information. Identification can thus be improved by comparing measured retention times to predicted retention times. Current prediction models are derived from a set of measured test analytes but they usually require large amounts of training data.
OpenMS offers a new kernel function, the *paired oligo-border kernel (POBK)*, which can be applied in combination with support vector machines to a wide range of computational proteomics problems. This enables the user to predict peptide adsorption/elution behavior in strong anion-exchange solid-phase extraction (SAX-SPE) and ion-pair reversed-phase high-performance liquid chromatography. Using the retention time predictions for filtering significantly improves the fraction of correctly identified peptide mass spectra. OpenMS offers a wrapper class to libSVM [103] for support vector learning. Our *POBK* is well-suited for the prediction of chromatographic separation in computational proteomics and requires only a limited amount of training data. Usually 40 peptides are sufficient. A more detailed description of the methods for retention time prediction, as well as the application of the retention time prediction to improve tandem MS identification results, can be found in [88]. The following code example shows how retention times can be predicted:

```
SVMWrapper svm;
svm_problem* training_data = NULL;
svm_problem* test_data = NULL;
//... load and encode data
//... set parameters of svm
svm.train(training_data);

vector<DoubleReal> predicted_rts;
svm.predict(test_data, predicted_rts);
```

After loading and encoding the data using the *LibSVMEncoder* class, one has to set the parameters of the support vector machine. It is also possible to determine the best parameters by performing a cross validation over parameter ranges. Afterwards the SVM can be trained. The trained SVM can then be used to predict retention times. At the end of the code example the predicted retention times are stored in the *predicted_rts* vector.

5.9. TOPP

The rapid development of both instrumentation and experimental techniques in mass spectrometry, make data analysis a difficult task. Analysis software has to be adaptable to different MS instruments with differing properties of the recorded signal. Moreover, the analysis strategy has to be adaptable to the experimental setup of different studies. Thus, it would be desirable to have a flexible toolbox, which facilitates data analysis and development of new analysis methods.

To address these issues we have developed TOPP – The OpenMS Proteomics Pipeline [68]. TOPP is a collection of command line tools for rapid development of data analysis pipelines in proteomics and metabolomics. Each TOPP tool performs one atomic analysis step, e.g., peak picking, normalization or map alignment. Complex analysis pipelines are constructed by combining several TOPP tools. A TOPP analysis pipeline can be easily adapted to new experiments or new analysis strategies: Parameters of the tools can be adapted or parts of the pipeline can be easily exchanged. The added value arising from such a modular concept was demonstrated by other bioinformatics toolboxes like EMBOSS [101].

Most other MS data analysis tools [16, 136, 109, 137, 14, 11, 113, 13, 138, 12] do not provide the flexibility of a customizable pipeline. They focus on a single analysis step, offer only a fixed sequence of analysis steps or are not executable in batch mode. However, some of these applications can be interfaced with TOPP using the standard file formats mzML [107], mzData [105] or mzXML [106].

Next to flexibility, ease of use is the second focus of TOPP. Binary installers of TOPP are available for Windows, MacOS and Ubuntu Linux. Along with the TOPP tools, documentation and tutorials are installed in PDF and HTML format. The documentation covers a general introduction, first analysis steps, more complex use cases.

The following sections will give a short overview and demonstrate the flexibility of the TOPP tools using three example pipelines. Then, TOPPView, the data viewer application of TOPP, is described. Finally, we demonstrate the use of TOPPAS to visually construct and edit TOPP pipelines.

5.9.1. Packages

The individual TOPP tools can be grouped into several distinct packages: file handling, signal processing, quantitation, identification, peptide property prediction and misc (see Fig. 5.18). We will now briefly discuss the most prominent tools of each package.

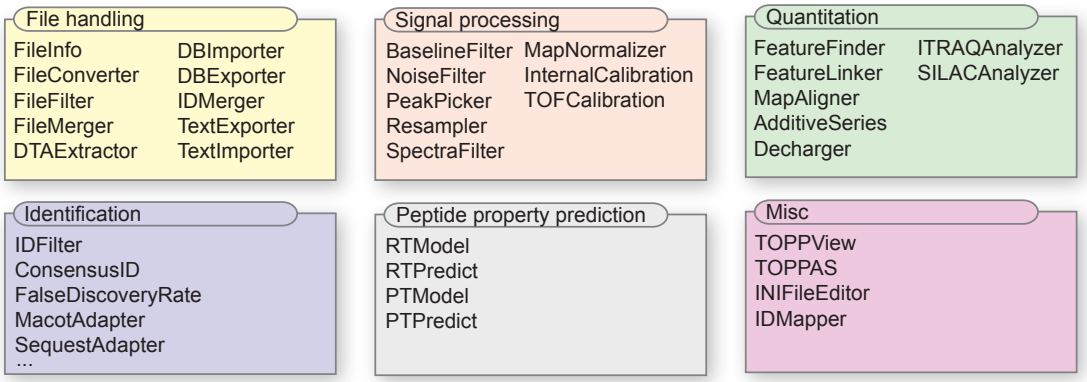


Figure 5.18.: The TOPP tools are grouped into several packages, each addressing one major area of functionality.

File handling

File handling is important for all data analysis pipelines, as there are several open standard formats and many proprietary vendor formats. *FileConverter* converts between several commonly-used file formats. Supported formats are mzXML [106] (ISB), mzData [105] (HUPO-PSI), mzML [107] (HUPO-PSI), ANDI/MS [104] and several other text-based formats.
FileFilter allows extraction of data from an MS file based on a combination of restricting rules. Peaks can be filtered according to RT interval, m/z interval, intensity interval and signal-to-noise ratio. Filters for whole spectra are for example spectrum type, MS-level and precursor peak m/z. The *FileMerger* tool is used to merge data from several files into one file.
FileInfo shows basic information about MS files, i.e., m/z range, RT range, intensity range and spectrum type. Additionally, meta data about sample, instrumentation, operator and applied processing can be shown. It can also be used for validation of XML file formats and for checking if peak data files contain corrupt data.
All TOPP tools are file-based. Database connectivity is provided through two auxiliary tools. *DBExporter* exports MS data from an OpenMS database to one or several files. Data can be imported into the database using *DBImporter*. Database connectivity is especially useful to distribute data for grid and workflow applications.

Signal processing

MS profile data always contains chemical noise and white noise. For some instrument types strong baseline fluctuations are also common. In order to increase the reliability of the data in downstream analysis steps, this noise and baseline should be removed. For noise reduction two smoothing filters are provided through the tool *NoiseFilter*: a Gaussian low-pass filter and a Savitzky-Golay filter [127].

Baseline reduction is performed with the *BaselineFilter*. It implements several morphological filters, of which the top-hat filter is best suited for removal of MS spectrum baseline.
The most important signal processing step is peak picking, the reduction of profile data to centroided data. The algorithm of the *PeakPicker* tool [139] uses wavelet-transformation of the signal to detect approximate peak positions. All other properties (exact centroid position, intensity, width, symmetry) are determined by fitting an asymmetric peak function to the data. In an optional, third step the detected peak positions can be further improved by methods from nonlinear optimization.

Quantitation

For absolute or relative quantitation of compounds in an LC-MS experiment all peaks belonging to a chemical species have to be aggregated to one feature. The *FeatureFinder* tool offers several algorithms to perform this task. The algorithms differ in the input data type and the used techniques to detect features:

- *centroided* - The feature detection algorithm for centroided data described in Chapter 3.
- *simple* - A feature detection algorithm for profile data based on a statistical model [69].
- *isotope_wavelet* - A feature detection algorithm for profile data based on a handcrafted wavelet for isotopic distributions [73, 70].

After detecting features, they often have to be grouped to perform the actual quantitation. Depending on the experimental setup, features are grouped within a single map (e.g., for ICAT [32] experiments) or across multiple maps (e.g., label-free approaches). The *FeatureLinker* tool offers several algorithms for grouping features. In label-free experiments with several measurements the RT has to be corrected before grouping can be performed. The *MapAligner* corrects retention time distortions between several maps through a star-wise alignment of the maps.
For the isotopic labeling techniques SILAC [33] and iTRAQ [34] the specialized tools *SILACAnalyzer* [140] and *ITRAQAnalyzer*, respectively, were developed.

Identification

There are plenty tools for identification of MS/MS spectra by database search or de novo sequencing. TOPP offers several adapters for those identification tools: *MascotAdapter* [9], *SequestAdapter* [31], *InspectAdapter* [141], *OMSSAAdapter* [142] and *XTandemAdapter* [143]. The adapters transform the input spectra and settings to the format required by the search engine. The results of the search are parsed and stored as an idXML file. The results can be filtered according to peptide/protein hit scores by *IDFilter*. Decoy database searches and calculation of false discovery rates is supported through the tool *FalseDiscoveryRate*.

Peptide property prediction

Identification engines report many false-positive peptide hits. The number of false positive hits can be reduced by taking secondary information like peptide RT and proteotypicity into account. The tools *RTModel* and *RTPredict* are used to create a model and predict peptide RT [88]. After prediction of peptide retention times, peptide hits

with a large deviation between measured and predicted RT can be filtered out. Prediction of peptide proteotypicity [144] can be utilized in a similar way. The tools *PTModel* and *PTPredict* are used for this task.

Misc

This package contains the graphical TOPP tools and tools that do not fit into other packages. Visual inspection of raw MS data and processing results is a fast and convenient method to assess the quality of the data. Therefore, *TOPPView* was developed, a powerful viewer for MS data and the graphical user interface for TOPP. It offers view modes for single spectra, whole LC-MS maps, feature maps and consensus maps. A detailed description of TOPPView can be found in Section 5.9.3. *TOPPAS* facilitates the use of TOPP by visual creation and execution of TOPP pipelines.

5.9.2. Example pipelines

The simplest way to set up an analysis pipeline with TOPP is using shell scripts. The TOPP tools are called in a defined order and results are passed on from tool to tool. In this section, the use of TOPP is illustrated by means of three important building-blocks of complex analysis pipelines. The first example pipeline shows the transformation of profile to centroided data. The second demonstrates peptide identification. The third shows how absolute label-free quantitation is performed with TOPP.

Signal processing

The first step of a typical analysis pipeline is signal processing. The TOPP tools for signal processing are *NoiseFilter*, *BaselineFilter* and *PeakPicker*. First, the *NoiseFilter* is used to smooth the data. Smoothing positively affects all following signal processing steps. Then, the *BaselineFilter* can be applied if the signal contains a baseline which must be removed. Finally, the profile data can be transformed to centroided data with the *PeakPicker*. This step reduces the amount of data by several orders of magnitude. Most high-level analysis steps, e.g., peptide identification, work only on centroided data, which is why peak detection is necessary in all analysis pipelines starting with profile data. The following script applies the described processing:

```
1 #smoothing of profile data
2 NoiseFiler -in profile.mzML -out smoothed.mzML -ini sp.ini
3 #baseline filtering
4 BaselineFilter -in smoothed.mzML -out baseline.mzML -ini sp.ini
5 #transformation to centroided data
6 PeakPicker -in baseline.mzML -out centroided.mzML -ini sp.ini
```

The algorithm parameters used are handed over to the tools in the common configuration file *sp.ini*. It contains individual sections for each tool. For signal processing the main parameters deal with describing the peak shape, e.g., the peak width in m/z dimension.

Peptide identification

Peptide identification based on MS/MS spectra is probably the most important analysis step of proteomics pipelines. The standard procedure is to compare the recorded MS/MS

spectra to theoretical spectra generated from a peptide database [6]. The peptide hit scores obtained by these searches are dependent on the search engine and the database used. Thus, results of different searches are often not comparable. To make them comparable, a background distribution of false positive hit scores can be used to calculate the false discovery rate corresponding to a score [145]. The background distribution is often determined by searching against a database with decoy sequences, e.g., tryptic peptides of reversed protein sequences. The following script demonstrates a typical peptide identification pipeline:

```
#extract MS/MS spectra
FileFilter -in run.mzML -out ms2.mzML -level 2
#peptide identification (forward search)
MascotAdapter -in ms2.mzML -out id.idXML -ini id.ini
#peptide identification (reverse search)
MascotAdapter -in ms2.mzML -out id_r.idXML -ini id_rev.ini
#Assign false discovery rates
FalseDiscoveryRate -in_fwd id.idXML -in_rev id_r.idXML -out id_fdr.idXML
#Filter hits
IDFilter -in id_fdr.idXML -out id_filtered.idXML -ini id.ini
```

First, the MS/MS spectra are extracted from the MS run using *FileFilter*. *MascotAdapter*, a wrapper for the Mascot [9] search engine, is used to generate lists of peptide identification candidates for each MS/MS spectrum. The search against the reversed database is performed with the same tool. Then, the two search results are fed into *FalseDiscoveryRate* to calculate false discovery rates for all peptide hits. In the last step, *IDFilter* is used to filter the peptide and protein hits according to false discovery rates.

Absolute label-free quantitation

The second main application of MS is peptide quantitation. As an example, the pipeline for absolute label-free quantitation of human myoglobin, a marker for myocardial infarction, in blood serum is used. The experimental setup has already been described in detail [146], so it is only shortly summarized here. The quantitation was performed using an additive series, i.e., adding known amounts of human myoglobin to aliquots of the sample with unknown concentration. In total, 32 measurements were performed: three technical replicates of eight addition experiments with different spiked amounts. To further lower the quantitation error, a known amount of horse myoglobin was added as an internal standard. The following script implements an analysis pipeline for the quantitation:

```
#Find features in all measurements
for i in `seq 1 32`; do
  FeatureFinder -in $i.mzData -out $i.featureXML -ini quant.ini
done
#Align feature maps
MapAligner -ini quant.ini
#Calculate a linear regression
AdditiveSeries -ini quant.ini
```

First, the peptide features of all peak maps have to be determined using the *FeatureFinder* tool. In a second step, the *MapAligner* aligns all feature maps to correct for retention time distortions. Finally, the feature intensities of myoglobin (human and horse) are used to calculate the final concentration by linear regression. Using this automated analysis a concentration of 0.417 ng/μl was determined (true value is 0.463 ng/μl). Whereas a manual expert analysis yielded a result of 0.382 ng/μl.

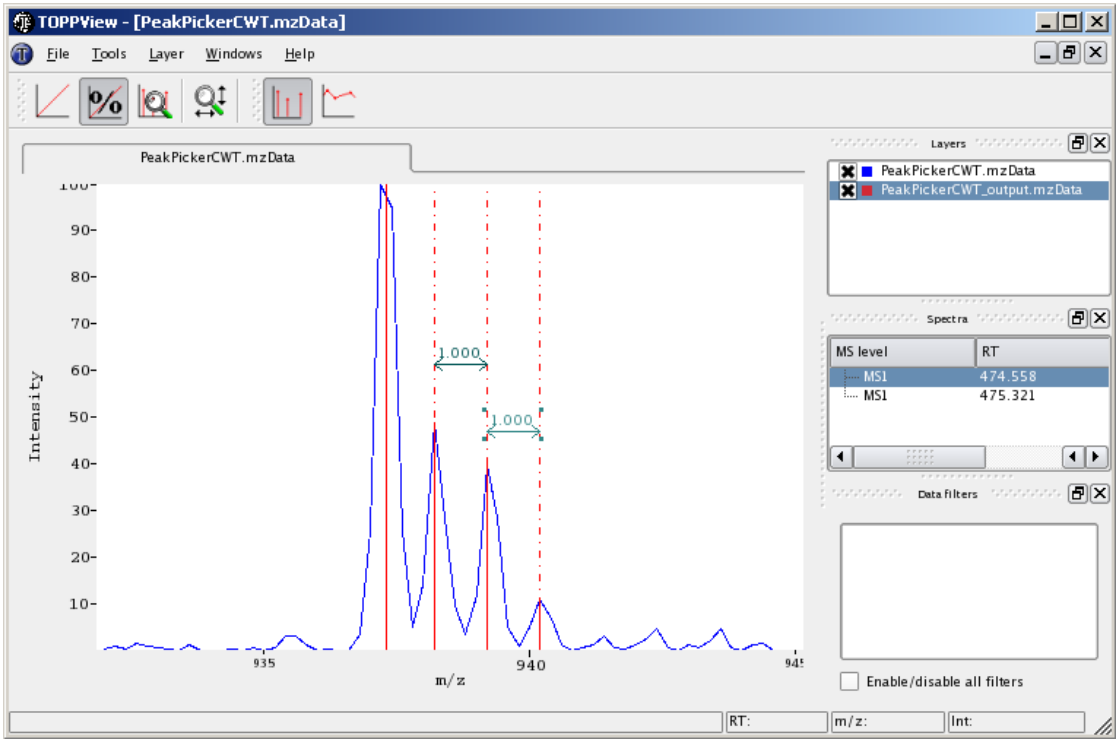


Figure 5.19.: The figure shows the TOPPView main window with an open 1D view. The view contains two layers, one with profile data (blue) and one with the corresponding centroided data (red).

5.9.3. TOPPView

TOPP was designed as a collection of command line tools, which each perform a single analysis task. This design makes TOPP very flexible and easy to integrate into complex analysis pipelines, which is especially important from a computer scientist's point of view. On the other hand, it makes TOPP hard to use for natural scientists which are not experienced in command line use. Therefore, a central GUI for the TOPP tools was developed: TOPPView [147] offers a Multi Document Interface (MDI) with advanced visualizations of the main data types of typical analysis pipelines. TOPPView is not only the visual front-end for TOPP. The second main use of TOPPView is visual exploration of the data, which is a very effective way to detect problems during the measurement or errors during data processing.

Several other MS data viewers are available, but most are not as flexible and powerful as TOPPView. Each MS instrument comes with vendor software for visualization and processing of the acquired data. However, vendor software is not freely available and cannot import data from other instruments. The latter is a major drawback when

working with several MS instruments of different manufacturers. A viable alternative to instrument software is freely-available software: Pep3D [148] displays LC-MS maps as a density plot with highlighted precursor peak positions. Another viewer, msInspect [12] can both visualize MS data and offers several data processing algorithms. The visualization consists of plots for whole LC-MS maps and plots for single spectra. Insilicos Viewer [149] displays the total ion current chromatogram of an LC-MS map and the spectrum corresponding to a selected time point of the chromatogram.

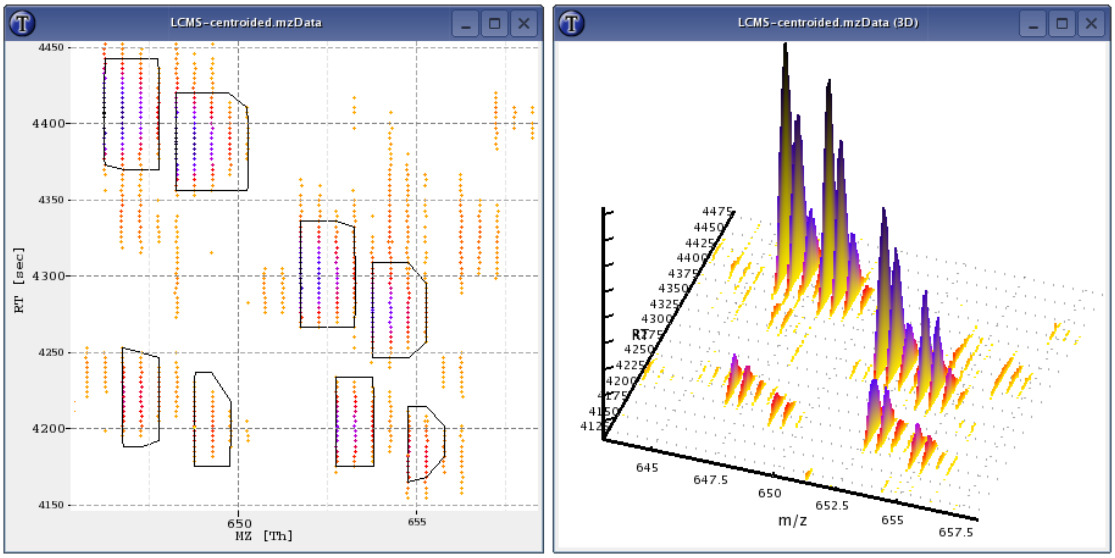


Figure 5.20.: Left: A 2D view displaying part of an LC-MS peak map is shown (colored dots) along with the corresponding feature outlines (black polygons). Right: The peak data from the left in a 3D view.

Main features

TOPPView shares many features with the viewers mentioned above, but also offers several unique features which set it apart. The main features of TOPPView are described in this section.

The views: TOPPView can be used to visualize MS peak maps (centroided and profile), feature maps and consensus feature maps. For peak data, several views are available: single spectra are shown in a 1D view, a plot of peak m/z on the x-axis and peak intensity on the y-axis (see Fig. 5.19). Whole LC-MS maps are displayed in a 2D view from bird's eye perspective with color-coded intensities. Smaller parts of LC-MS maps can also be displayed in a 3D view for closer inspection (see Fig. 5.20). Feature maps and consensus maps can be displayed in the 2D view.
Each of the views supports zooming into the data, measuring distances between data points and customization through a preferences dialog. All views are part of the OpenMS library and, thus, can be reused to create custom viewer applications.

Layer interface: The most powerful new feature of TOPPView is the layer interface. Each view can display several layers, i.e., several datasets, at a time. This is often superior to displaying only one layer. In the 1D view, profile data and centroided data

can be displayed together in order to assess the performance of peak detection algorithms. Another use case would be to display recorded MS/MS spectra and theoretical MS/MS spectra side by side. In the 2D view the layer concept becomes even more powerful. Peptide feature centroids and convex hulls can be displayed along with the peak data in order to supervise feature detection. The performance of map alignment can be easily checked by displaying the aligned data in one window. Examples can be found in Fig. 5.19 and Fig. 5.20.

Data filters: Another key concept of TOPPView is data filtering. Displaying only part of the data is often desirable because of the vast amount of data. For each displayed dataset filters can be added, e.g., all peaks above an intensity threshold. The filters can also be based on annotated meta data of the features, e.g., the estimated charge of a peptide features.

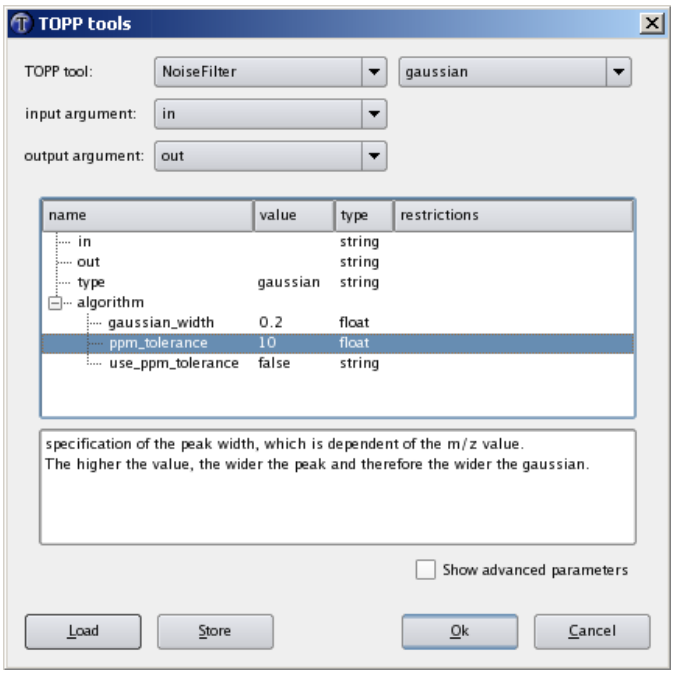


Figure 5.21.: This screenshot shows the TOPP tools dialog in TOPPView. The Gaussian noise filter is selected and its main parameters are shown. Advanced parameters could be enabled using the check box at the bottom. Below the parameters the description of the currently selected parameter is displayed.

TOPP integration: TOPPView also offers rich data analysis functionality through the integration of the TOPP tools (Fig. 5.21). When a user opens the TOPP dialog, a selection of TOPP tools that can process the data in the active layer is given. After selecting a tool, its parameters are shown and can be adapted to the data to process. The processing is performed in the background in a second thread, so that the user can continue to interact with TOPPView. When the processing is finished, the results are loaded and can be opened in a new window or in an additional layer of an existing window.
The GUI to the TOPP tools is especially useful to find suitable algorithm parameters

for a dataset. As an example, optimizing the peak picker parameters is described in the following: Starting with an LC-MS peak map, a single spectrum can be selected and opened in a 1D view. The selected spectrum is run through the *PeakPicker* algorithm. The resulting centroided data can be displayed along with the profile data, which makes a quick visual assessment of the algorithm performance possible. The user can now adapt the algorithm parameters and rerun the tool, until suitable parameters are found.

Meta data visualization: When submitting MS data to public databases, e.g., PRIDE [150], certain meta data is required. Often, at least part of the meta data required by MIAPE [126] or journal guidelines is missing. TOPPView offers a GUI for MS meta data, which is a very convenient way to complement the missing information.

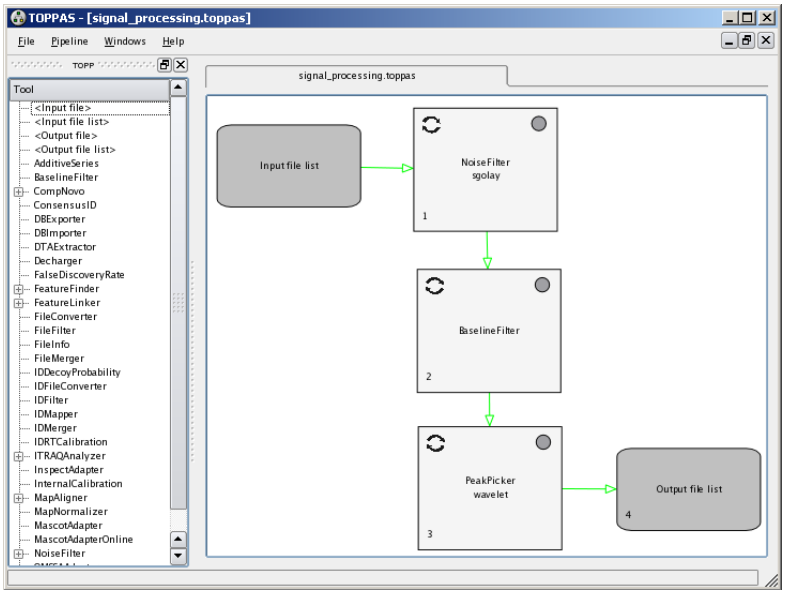


Figure 5.22.: Screenshot of TOPPAS showing a typical signal processing pipeline.

5.9.4. TOPPAS

The modular design of TOPP allows a very flexible use, but has one major drawback. The user has to work through a command line interface and has to write scripts to conveniently apply TOPP pipelines to several data files. Especially Windows users are not familiar with command line interfaces, which makes using TOPP difficult. TOPPView already allows execution of single TOPP tools from a GUI. To facilitate the use of larger pipelines, a graphical user interface for whole pipelines was created: *TOPPAS* allows the creation and execution of TOPP pipelines in a convenient way.

Fig. 5.22 shows a screenshot of TOPPAS visualizing a simple signal processing pipeline. Pipelines are created by dragging tools from the toolbar on the left to the workspace and connecting the tools via edges. Edges represent the data flow between tools. During the creation process, TOPPAS performs sanity checks to avoid the construction of inoperative pipelines. The parameters of each tool can be visually edited by double-clicking a tool. Pipelines can be stored and loaded as *.toppas* files in a simple XML-based file format.

When the pipeline is fully configured, it can be executed—either in the GUI or in the command line. In the GUI execution mode, colors are used to indicate the status of the pipeline. Additionally, a *log* window shows the output of the tools and error messages. In case of an error, the pipeline can be modified and restarted where it aborted. After execution of a pipeline, the output of the pipeline can be opened in TOPPView through the context menu.
The CLI execution is intended for batch execution of a pipeline, or for scripting. To use it, TOPPAS is called with the command line parameter *–execute*. In this mode a given pipeline is executed only—the GUI is not shown.

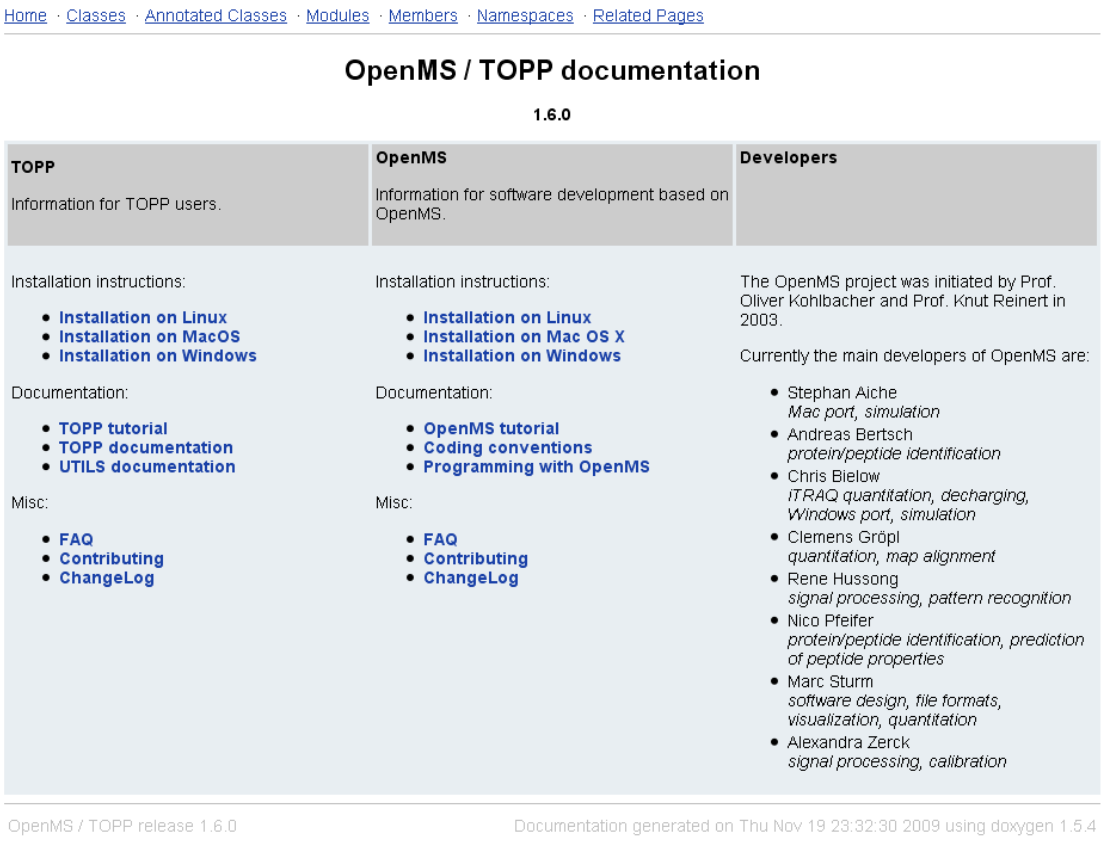


Figure 5.23.: Screenshot of the OpenMS documentation start page.

5.10. Project management

The OpenMS library is a large project with many contributors from different geographical locations. Currently it offers more than 500 classes that add up to about 200,000 lines of code. A project of this size requires a significant amount of project management to ensure a high quality standard. The following sections will describe the most important aspects of project management in the OpenMS project.

5.10.1. Version control system

Source code version control systems are essential for all software projects. The most obvious benefit is that simultaneous editing of one source file by two or more persons is possible. Version control systems assist the developers with merging of code versions and help to resolve occurring conflicts. Another very important use case is the management of release branches and merging bug fixes between development and release branches. These features make a version control system indispensable.
For the OpenMS source code the version control system Subversion [38] is used. The Subversion server is provided by SourceForge [115], a development platform for open-source software. OpenMS also uses SourceForge infrastructure for the project website, distribution of release packages, mailing lists and bug tracking.

5.10.2. Coding conventions

Individual programming style highly affects the readability of source code. Each programmer has its own style in terms of brackets, variable names, etc. In order to ensure good readability a fixed style has to be enforced for larger projects. However, it should not restrict the individual programmer too much. The OpenMS project mainly enforces class, method and variable names through the *Coding convention* [151]—a part of the documentation.
Unfortunately, such rules are often ignored, unless enforced. Thus, the PHP [152] script *checker.php* is provided that checks for coding convention violations. In order to check variable and method names an XML representation of the OpenMS classes created by Doxygen [114] is used. Additionally, several other checks are performed, e.g., if a test has been written for each method of a class.

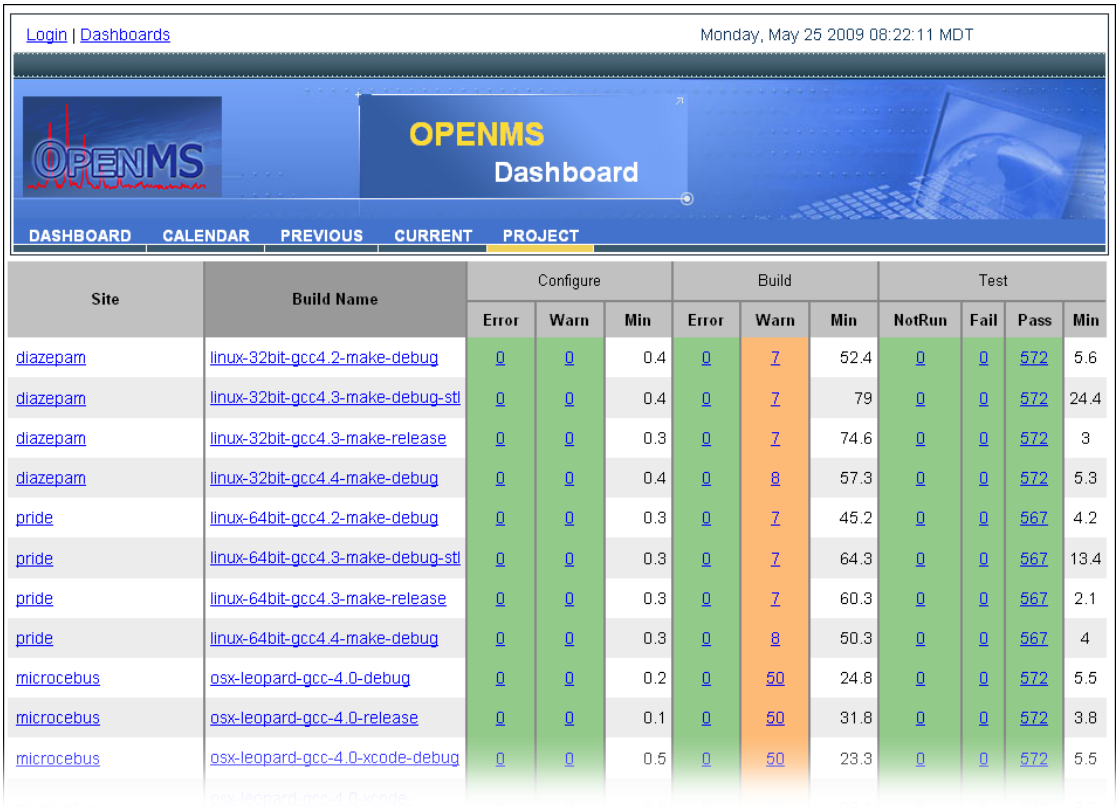
5.10.3. Documentation

As stated in Section 5.2, documentation of the source code is a very important part of each software project. OpenMS uses Doxygen [114] to generate the class documentation in HTML format. Doxygen extracts the documentation from the source code, which ensures that the documentation and the implementation cannot diverge. The following listing show an example class with Doxygen documentation:

```
1  /**
2    @brief Short class documentation.
3
4    More elaborate class documentation,
5    consisting of several lines.
6  */
7  class Example
8  {
9    public:
10     ///Default constructor
11     Example();
12     /**
13       @brief Member that sets the name.
14
15       @param name The name to be set.
16     */
```

```
17      void setName(const String& name);
18      /**
19        @brief Member that returns the name.
20
21        @return The name of this class.
22      */
23      const String& getName() const;
24
25    protected:
26      ///string member
27      String name_;
28  }
```

Doxygen defines its own commands, which are used to document the source code. Some HTML tags can also be used to format the documentation and even LaTeX formulas can be integrated.
Besides generating the class documentation Doxygen allows adding custom pages. This mechanism is used to integrate other relevant information into the class documentation. On the documentation main page (see Fig. 5.23) installation instructions, tutorials, coding convention, FAQ and many other pages are referenced.



Site	Build Name	Configure			Build			Test			
		Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min
diazepam	linux-32bit-gcc4.2-make-debug	0	0	0.4	0	7	52.4	0	0	572	5.6
diazepam	linux-32bit-gcc4.3-make-debug-stl	0	0	0.4	0	7	79	0	0	572	24.4
diazepam	linux-32bit-gcc4.3-make-release	0	0	0.3	0	7	74.6	0	0	572	3
diazepam	linux-32bit-gcc4.4-make-debug	0	0	0.4	0	8	57.3	0	0	572	5.3
pride	linux-64bit-gcc4.2-make-debug	0	0	0.3	0	7	45.2	0	0	567	4.2
pride	linux-64bit-gcc4.3-make-debug-stl	0	0	0.3	0	7	64.3	0	0	567	13.4
pride	linux-64bit-gcc4.3-make-release	0	0	0.3	0	7	60.3	0	0	567	2.1
pride	linux-64bit-gcc4.4-make-debug	0	0	0.3	0	8	50.3	0	0	567	4
microcebus	osx-leopard-gcc-4.0-debug	0	0	0.2	0	50	24.8	0	0	572	5.5
microcebus	osx-leopard-gcc-4.0-release	0	0	0.1	0	50	31.8	0	0	572	3.8
microcebus	osx-leopard-gcc-4.0-xcode-debug	0	0	0.5	0	50	23.3	0	0	572	5.5

Figure 5.24.: Screenshot of the OpenMS nightly builds page.

5.10.4. Testing

A fundamental step in software development is testing. In OpenMS different testing techniques are used: All classes throw exceptions when they receive invalid input or enter an invalid internal state. Additionally preprocessor macros enable assertion checks if OpenMS is build in *Debug* mode. These very basic tests can however not make sure that a class behaves according to its specification. In order to achieve this, *unit tests* are used, which check the behavior of each class.

Unit testing: Several different approaches for unit testing are available. We decided to use simple specification-based *black-box testing* for OpenMS.
In order to facilitate testing, a framework of preprocessor macros is used to set up the testing environment and perform subtests for each method of the tested class. The following listing shows part of the *String* test:

```
1  #include <OpenMS/CONCEPT/ClassTest.h>
2  #include <OpenMS/DATASTRUCTURES/String.h>
3
4  START_TEST(String, "$Id: String_test.C 5019 2009-04-05 20:49:49Z ms $")
5
6  String* s_ptr = 0;
7  START_SECTION(String())
8    s_ptr = new String;
9    TEST_NOT_EQUAL(s_ptr, 0)
10 END_SECTION
11
12 START_SECTION(~String())
13   delete s_ptr;
14 END_SECTION
15
16 START_SECTION(Int toInt() const)
17   String s = "12.3";
18   TEST_EQUAL(s.toInt(),12);
19 END_SECTION
20
21 START_SECTION(Real toFloat() const)
22   String s = "12.3";
23   TEST_REAL_SIMILAR(s.toFloat(),12.3);
24 END_SECTION
25
26 END_TEST
```

First, the header containing the test macros (*ClassTest.h*) and the header of the tested class are included. The *START_TEST* macro sets up the testing environment and prints some information to the command line, e.g., the *Id* string which contains the test name and the SVN revision. This information is automatically updated by Subversion whenever the file changes.
Next, each method is tested in a separate subtest, surrounded by *START_SECTION* and *END_SECTION* macros. In this scope, one or several test macros are used to test the behavior of the method. In the example, constructor and destructor are tested first, then

two conversion methods to numerical values are tested. As can be seen in the example, different test macros are provided for different types. This is necessary because not all types can be easily compared, e.g., a small deviation must be allowed for floating-point numbers. Each test macro prints the line number in the source code, the test code and the result to the command line.
The *END_TEST* macro prints a summary of the whole class test. This includes the line number of all failed tests, which is helpful especially in lengthy tests. The output of the above example looks like this:

```
checking String() ...
 +  line 9:  TEST_NOT_EQUAL(s_ptr,0): got 0x6bfdf0, forbidden is 0
: passed

checking ~String() ...
: passed

checking Int toInt() const ...
 +  line 18:  TEST_EQUAL(s.toInt(),12): got 12, expected 12
: passed

checking Real toFloat() const ...
 +  line 23:  TEST_REAL_SIMILAR(s.toFloat(),12.3): got 12.3, expected 12.3
: passed

PASSED
```

The management and execution of the unit tests is done with the *CTest* tool, which is distributed as part of the CMake [54] package. *CTest* allows execution of all tests or a subset of the tests, for example defined by a regular expression. When all tests have been executed, it prints a summary of the test results.

Automated test builds: Because OpenMS is under continuous development, continuous testing is required. In theory, each developer should execute all unit tests before committing new code. However, this is quite time-consuming and thus often neglected. Another, even bigger problem is portability. Different C++ compilers tend to implement different interpretations of the ANSI C++ standard. Thus, even correct code might fail on some compiler or platform.
In order to ensure high code quality, nightly test builds and unit testing on most supported platforms and architectures have been set up. The test builds are executed on several different servers and the results are submitted to a *CDash* [56] web server, which creates an HTML summary of all tests of each day (see Fig. 5.24). These browsable test results allow easy tracking and removal of code problems.

5.10.5. Release management

A fast release cycle is important for scientific software libraries which are under ongoing development. Releasing often ensures that new functionality is distributed to the users of the library. On the other hand, each release creates a lot of overhead due to documentation, testing and packaging tasks.
The OpenMS release plan strives for a release approximately every six months. One

month before the release date the active development in the *trunk* is stopped. The month before the release is dedicated to consolidation of the functionality and the code. Only documentation, testing and refactoring tasks are performed on the *trunk* in this time. New features have to be developed in a *branch* and can be merged into the *trunk* after the release.

OpenMS offers two release types of release packages. Source code packages are provided for software developers. Binary packages are provided for users that only want to use the suite of applications that comes with the software library. Binary packages are available for Microsoft Windows, MacOS and Ubuntu Linux. Besides the actual software, each release package contains the full documentation, tutorials and the change log.

Table 5.3 shows the some statistics on OpenMS releases. It illustrates that both the code base and the user community of OpenMS are rapidly growing.

Version	release date	lines of code	downloads per month
1.6	12/2009	212.320	303
1.5	08/2009	200.172	191
1.4	04/2009	168.654	283
1.3	02/2009	164.366	442
1.2	08/2008	142.766	205
1.1	04/2008	130.513	172
1.0	07/2007	126.039	74

Table 5.3.: OpenMS release statistics.

5.11. Discussion and outlook

We have presented OpenMS—a large, versatile and functional software framework for mass spectrometry data analysis. OpenMS is a joint project of several academic research groups. It is under active development for more than six years in which we continuously improved and augmented the library. OpenMS 1.6, which improved tremendously upon the previous versions [153], was released in November 2009. In its current state OpenMS can dramatically cut down on development time for devising analysis pipelines and testing new algorithmic strategies in the field of MS-based proteomics and metabolomics. Thus, we anticipate that OpenMS will contribute to speeding up biomedical research. OpenMS has proven its usefulness in several projects and studies. Selected projects are summarized now:

TOPP – The OpenMS Proteomics Pipeline: OpenMS has been successfully used for the implementation of TOPP – The OpenMS Proteomics Pipeline [68]. TOPP is a set of computational tools that can be chained together to tailor problem-specific analysis pipelines for LC-MS data. It transforms most of the OpenMS functionality into small command line tools that are the building blocks for more complex analysis pipelines. The functionality of the tools ranges from data preprocessing and signal processing over quantitation to peptide and protein identification.
The source code of the TOPP tools is on average no longer than 150 lines and the largest part of it deals with the evaluation of command line parameters. The core functionality of most tools can be implemented in less than 20 lines of code. This is

possible not only because OpenMS facilitates file handling and offers clear interfaces for its algorithms. OpenMS also offers powerful parameter handling classes both for parameters of algorithms and command line arguments of applications. Given such an infrastructure, the development time of new tools is significantly reduced.

Clinical proteomics: In March 2010, the Swedish company *MedicWave* introduced their commercial Software for label-free quantitative proteomics that is based on OpenMS and TOPP as part of their *MedicWave Bioinformatics Suite*^{TM}(MBS). The quantification module of MBS, *LEQuant*, re-implements functionality of several TOPP tools and integrates it into the graphical interface of MBS.

Protein quantitation: OpenMS contains several algorithms for peptide quantitation based on model fitting [69, 70, 71]. Using the data structures provided by OpenMS and these algorithms, we were able to implement data analysis code for various complex quantitation tasks (labeled/unlabeled strategies, relative/absolute quantitation). In a case study [146] we could thus show that the use of these algorithms improved quantitation accuracy in a complex absolute quantitation scenario (myoglobin in human blood plasma) while drastically reducing analysis times.

Protein and peptide identification: Kapp *et al.* [61] showed that most of the correct peptide identifications are found by various search engines. Nevertheless, they also demonstrated that there is a certain amount of correctly identified spectra which could only be identified by one or two of the search engines which they compared in their study.
For these reasons we integrated the identifications of different search engines like Mascot [9], InsPecT [141] and OMSSA [142]. The results of the different search engines were combined by calculating an average rank for the identification candidates. Tightening the used score thresholds of the different search engines results in more accurate identification, as only high quality identifications are kept. Using a looser score threshold the number of correctly identified peptides increases.
To find more true positive identifications one can lower score thresholds of the search engines further and filter out most of the additional false positive identifications using our retention time filtering approach. In this approach the retention time prediction model (Section 5.8.4) is trained by a small set of high confidence identifications. This model is then used to predict retention times for all further identifications. If there is a large difference between observed and predicted chromatographic behavior for some identifications, these identifications are excluded. This evaluation is described in more detail in [88].

Final remarks and future work: Developers using OpenMS are strongly encouraged to take part in the project by contributing their algorithms. Providing an algorithm in a framework allows a much more flexible reuse than providing an application only.
Driven by collaborative projects with experimental partners we will add more functionality as the project proceeds. Planned improvements for the near future comprise automatic algorithm parameter estimation, more powerful visualization techniques and support for the upcoming HUPO-PSI formats mzIdentML and TraML.

6. Conclusion and Outlook

In this thesis we have presented two new algorithmic approaches for LC-MS data analysis and a C++ framework for rapid prototyping of analysis software in the field of proteomics and metabolomics.

In Chapter 3 we have presented a novel quantitation algorithm for centroided LC-MS data. It is based on an averagine isotope pattern and a Gaussian elution profile model. Feature candidate regions are fitted to a two-dimensional model to increase the sensitivity of the algorithm. Finally, a greedy approach is used to resolve contradicting features.

In a comprehensive comparison, we could show that our algorithm combines an excellent feature detection performance with a good runtime. Only one out of four state-of-the-art algorithms could reach a similar performance in our comparison. Besides a good performance, further design goals of the algorithm were wide applicability and robustness. The algorithm works reliably both on medium-resolution and high-resolution centroided LC-MS data, even at high levels of chemical and white noise. It is provided as a command line tool which allows an easy integration into existing analysis pipelines. Additionally, it is available as a C++ class which can be used for the development of new analysis application.

Although our algorithm shows good performance and was successfully used in several studies, there is still much room for improvement. Future versions of the algorithm could benefit from improved peptide feature models: An EMG elution profile model could improve the recall and quantitation accuracy for asymmetric chromatographic peaks. Additionally, the averagine isotope model could be extended by a fixed number of additional sulfur atoms. This would allow a more accurate modeling of cysteine-rich peptides. Metabolite feature detection would also be possible with the algorithms when replacing the averagine isotope distribution by a more general isotope model.

In Chapter 4 we have presented a machine learning approach for the prediction of oligonucleotide retention times in liquid chromatography. Its main improvements over existing approaches are the incorporation of DNA secondary structure information and use of support vector regression.

Secondary structure information allows an accurate modeling of the retention time at lower temperatures and for oligonucleotides which form a highly stable secondary structure. It improves the prediction accuracy for fixed-temperature models, especially at low temperatures where secondary structure formation is not completely suppressed. Additionally, it can be used to create multi-temperature models. These models predict oligonucleotide retention times within the training data temperature range—in our case from 30°C to 80°C.

The second major improvement was the use of support vector regression. Support vector regression maximizes the model accuracy while minimizing model complexity. It models non-linear relationships with high robustness towards outliers, which ensures a good generalization of the model. Our results demonstrate that the use of secondary structure information and support vector regression significantly improves the prediction performance over existing models.

In Chapter 5 we have presented OpenMS, a platform-independent software framework for mass spectrometry, and TOPP – The OpenMS Proteomics Pipeline. OpenMS provides data structures and algorithms for rapid software development in the field of computational proteomics. One goal of OpenMS is to facilitate the development of novel algorithms for mass spectrometry data analysis. OpenMS offers a rich infrastructure for software engineers—including data structures for mass spectrometry data, file I/O for most non-proprietary formats, parameter handling classes, progress logging classes, etc. Thus, developers can focus on the actual algorithm design while relying on the OpenMS infrastructure for all technical aspects. OpenMS also provides algorithms for all major analysis steps in proteomics. For example, the peptide feature quantitation algorithm presented in Chapter 3 of this thesis.

Besides the development of new algorithms, the second focus of OpenMS is to allow the rapid development of powerful analysis tools. We have demonstrated the versatility of OpenMS with the development of TOPP – The OpenMS Proteomics Pipeline. TOPP is a set of computational tools for mass spectrometry, each performing one atomic analysis step. Individual tools can be combined to construct problem-specific analysis pipelines via an intuitive graphical user interface. The functionality of TOPP is rounded off by TOPPView, a powerful viewer for mass spectrometry data. TOPPView is typically used to visualize raw data along with analysis results in order to assess the quality of the data analysis.

In summary, OpenMS and TOPP facilitate the development of analysis pipelines and testing new algorithmic strategies in the field of MS-based proteomics and metabolomics. After several years of continuous development and testing, they have reached a high degree of maturity. We anticipate that OpenMS and TOPP will be useful for the whole community and that they can contribute to speeding up biomedical research.

A. List of abbreviations

ASCII	American Standard Code for Information Interchange
API	Application programming interface
CE	Capillary electrophoresis
CLI	Command line interface
CPU	Central Processing Unit
CV	Controlled vocabulary
DNA	Deoxyribonucleic acid
EMG	Exponentially modified Gaussian
EIC	Extracted ion chromatogram
ESI	Electro spray ionization
GPU	Graphics Processing Unit
GUI	Graphical user interface
HPLC	High performance liquid chromatography
HUPO	Human Proteome Organization
I/O	Input/Output
IDE	Integrated development environment
ISB	Institute for Systems Biology
LC	Liquid chromatography
MALDI	Matrix-assisted laser ionization/desorption
MDI	Multi-document interface
MIAPE	Minimum Information About a Proteomics Experiment
MRM	Multiple reaction monitoring
MS	Mass spectrometry
MSVC	Microsoft Visual C++
MS/MS	Tandem MS
m/z	Mass-to-charge ratio
NPLC	Normal phase liquid chromatography
PSI	Proteomics Standards Initiative
OOP	Object-oriented programming
Q-TOF	Quadrupole Time-of-Flight mass spectrometer
RNA	Ribonucleic acid
RT	Retention time
RPLC	Reverse phase liquid chromatography
STL	Standard Template Library
SVC	Support vector classification
SVM	Support vector machine
SVR	Support vector regression
TIC	Total ion current chromatogram
TOF	Time-of-flight mass analyzer
UML	Unified Modeling Language
XIC	Extracted ion chromatogram
XML	Extensible Markup Language

B. Contributions

A novel feature detection algorithm for centroided data

MS designed and implemented the algorithm, and performed the evaluation.

DNA retention time prediction

MS and OK designed and implemented the prediction method, and preformed the evaluation. SQ, BM and CH provided the training dataset and performed all experiments. The method was published in [154, 155] .

OpenMS

OK and KR initiated and coordinate the project. **MS** designed and implemented the kernel data structures, meta data handling, file and database I/O classes, and visualization classes. **MS**, AB, SA and CB maintain the build system and the unit test system. CG, KR, OST and **MS** designed and implemented the feature detection algorithms. AH and RH contributed the isotope wavelet. EL, AH and AZ designed and implemented the signal processing and peak picking algorithms. EL and CG designed and implemented the map alignment algorithms. AB, NP, OK and **MS** designed and implemented the algorithms related to peptide identification and retention time prediction. The OpenMS software framework was published in [153, 82].

TOPP

OK and KR initiated and coordinate the project. **MS** implemented file handling tools, visualization tools and the base functionality shared by all TOPP tools. CG and EL implemented the map alignment tool. EL, AH and AZ implemented the signal processing and peak picking tools. CG, OST and **MS** implemented the feature detection tools. CG, AB and NP implemented the tools for peptide identification. NP implemented the tools for peptide property prediction. The TOPP software tools were published in [68, 147].

AZ: Alexandra Zerck, AB: Andreas Bertsch, AH: Andreas Hildebrandt, BM: Bettina Mayr, CB: Chris Bielow, CG: Clemens Gröpl, CH: Christian Huber, EL: Eva Lange, KR: Knut Reinert, OK: Oliver Kohlbacher, MS: Marc Sturm, NP: Nico Pfeifer, OST: Ole Schulz-Trieglaff, SA: Stephan Aiche, SQ: Sascha Quinten

C. List of Publications

Journal publications

M. Sturm, O. Kohlbacher. *TOPPView: An Open-Source Viewer for Mass Spectrometry Data.* Journal of Proteome Research. 2009;8(7):3760–3.

M. Sturm, A. Bertsch, C. Gröpl, A. Hildebrandt, R. Hussong, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, A. Zerck, K. Reinert, O. Kohlbacher. *OpenMS – An open-source software framework for mass spectrometry.* BMC Bioinformatics. 2008 Mar 26;9:163.

M. Sturm, S. Quinten, C.G. Huber, O. Kohlbacher. *A statistical learning approach to the modeling of chromatographic retention of oligonucleotides incorporating sequence and secondary structure data.* Nucleic Acids Res. 2007;35(12):4195–202.

B.M. Mayr, O. Kohlbacher, K. Reinert, **M. Sturm**, C. Gröpl, E. Lange, C. Klein, C.G. Huber. *Absolute myoglobin quantitation in serum by combining two-dimensional liquid chromatography-electrospray ionization mass spectrometry and novel data analysis algorithms.* Journal of Proteome Research. 2006 Feb;5(2):414–21.

O. Kohlbacher, S. Quinten, **M. Sturm**, M.B. Mayr, C.G. Huber. *Structure-activity relationships in chromatography: retention prediction of oligonucleotides with support vector regression.* Angewandte Chemie International Edition. 2006 Oct 27;45(42):7009–12.

P. Dönnes, A. Höglund, **M. Sturm**, N. Comtesse, C. Backes, E. Meese, O. Kohlbacher, H.P. Lenhof. *Integrative analysis of cancer-related data using CAP.* FASEB Journal. 2004 Sep;18(12):1465–7.

Peer reviewed conference contributions

O. Kohlbacher, K. Reinert, C. Gröpl, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, **M. Sturm**. *TOPP – The OpenMS Proteomics Pipeline.* ECCB 2006 conference proceedings. Bioinformatics. 2007 Jan 15;23(2):191–7.

C. Gröpl, E. Lange, K. Reinert, O. Kohlbacher, **M. Sturm**, C.G. Huber, B.M. Mayr, C.L. Klein. *Algorithms for the automated absolute quantification of diagnostic markers in complex proteomics samples.* In Proceedings of the 1st International Symposium on Computational Life Science (CompLife05), pages 151–163, 2005.

Other publications

K. Reinert, O. Kohlbacher, C. Gröpl, E. Lange, O. Schulz-Trieglaff, **M. Sturm**, N. Pfeifer. *OpenMS – A Framework for Quantitative HPLC/MS-Based Proteomics.* In Proceedings of the Dagstuhl Seminar on Computational Proteomics 2006, edited by Christian G. Huber and Oliver Kohlbacher and Knut Reinert. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl,

Germany, 2006.

M. Sturm, S. Quinten, C.G. Huber, O. Kohlbacher. *A machine learning approach for the prediction of DNA and peptide retention times.* In Proceedings of the Dagstuhl Seminar on Computational Proteomics 2005, edited by Christian G. Huber and Oliver Kohlbacher and Knut Reinert. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

Accepted manuscripts

L. Nilse, **M. Sturm**, D. Trudgian, M. Salek, P. Sims, K. Carroll, S.J. Hubbard. *SILACAnalyzer – a tool for differential quantitation of SILAC data.* Lecture Notes in Bioinformatics. 2010, accepted for publication.

Submitted manuscripts

L. Martens, M. Chambers, **M. Sturm**, D. Kessner, F. Levander, J. Shofstahl, W. Tang, A. Roempp, A.D. Pizarro, L. Montecchi-Palazzi, N. Tasman, M. Coleman, F. Reisinger, P. Souda, H. Hermjakob, P.A. Binz, E. Deutsch. *mzML – a Community Standard for Mass Spectrometry Data.* Molecular and Cellular Proteomics. 2010, submitted.

D. Detailed quantitation results

	m1_r1	m1_r2	m2_r1	m2_r2
centroided	143	226	214	295
SpecArray	0	35	23	53
msInspect	83	232	85	219
simple	20	93	148	304
isotope_wavelet	273	419	370	283

Table D.1.: Overall number of detected features.

	m1_r1	m1_r2	m2_r1	m2_r2
centroided	90	83	124	179
SpecArray	0	33	23	50
msInspect	65	104	61	149
simple	12	40	76	140
isotope_wavelet	44	41	23	59

Table D.2.: Number of correctly matched features.

	m1_r1	m1_r2	m2_r1	m2_r2
centroided	10	25	0	4
SpecArray	0	0	1	3
msInspect	7	12	4	14
simple	2	21	19	26
isotope_wavelet	21	39	29	19

Table D.3.: Number of incorrectly matched features, i.e., wrong charge state or multiple matches.

	m1_r1	m1_r2	m2_r1	m2_r2	Average
centroided	77.59	63.36	93.23	80.63	78.70
SpecArray	0.00	25.19	17.29	22.52	16.25
msInspect	56.03	79.39	45.86	67.12	62.10
simple	10.34	30.53	57.14	63.06	40.27
isotope_wavelet	37.93	31.30	17.29	26.58	28.27

Table D.4.: Recall (only correctly matched features).

	m1_r1	m1_r2	m2_r1	m2_r2	Average
centroided	62.94	36.73	57.94	60.68	54.57
SpecArray	100.00	94.29	100.00	94.34	97.16
msInspect	78.31	44.83	71.76	68.04	65.74
simple	60.00	43.01	51.35	46.05	50.10
isotope_wavelet	16.12	9.79	6.22	20.85	13.24

Table D.5.: Precision (only correctly matched features).

	m1_r1	m1_r2	m2_r1	m2_r2	Average
centroided	86.21	82.44	93.23	82.43	86.08
SpecArray	0.00	25.19	18.05	23.87	16.78
msInspect	62.07	88.55	48.87	73.42	68.23
simple	12.07	46.56	71.43	74.77	51.21
isotope_wavelet	56.03	61.07	39.10	35.14	47.83

Table D.6.: Recall (including incorrectly matched features).

	m1_r1	m1_r2	m2_r1	m2_r2	Average
centroided	6.01	112.24	2.06	7.16	31.87
SpecArray	71.32	402.10	251.84	882.31	401.89
msInspect	12.14	7.77	3.52	2.96	6.60
simple	16.85	10733.38	159.97	861.56	2942.94
isotope_wavelet	981.75	597.38	45.71	30.82	413.92

Table D.7.: Runtimes in seconds. The runtimes were determined on a Dual Core AMD Opteron(tm) 275 machine with 2.0 GHz CPU frequency and 8 GB RAM.

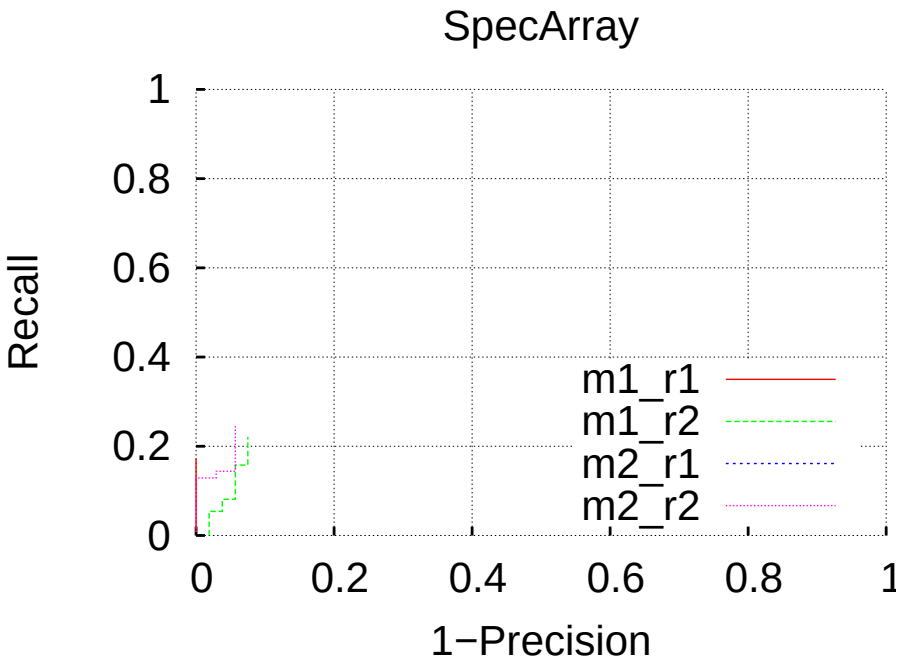


Figure D.1.: Plot of precision against recall for SpecArray quantitation results. Bipartitions of the output feature set ordered according to intensity were used to calculate corresponding precision and recall values.

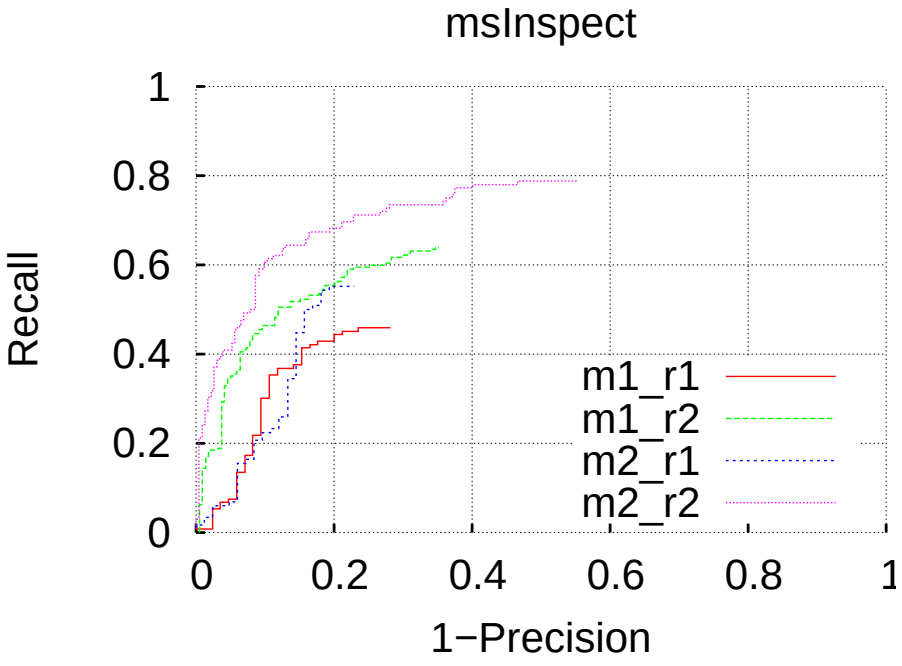


Figure D.2.: Plot of precision against recall for msInspect quantitation results. Bipartitions of the output feature set ordered according to intensity were used to calculate corresponding precision and recall values.

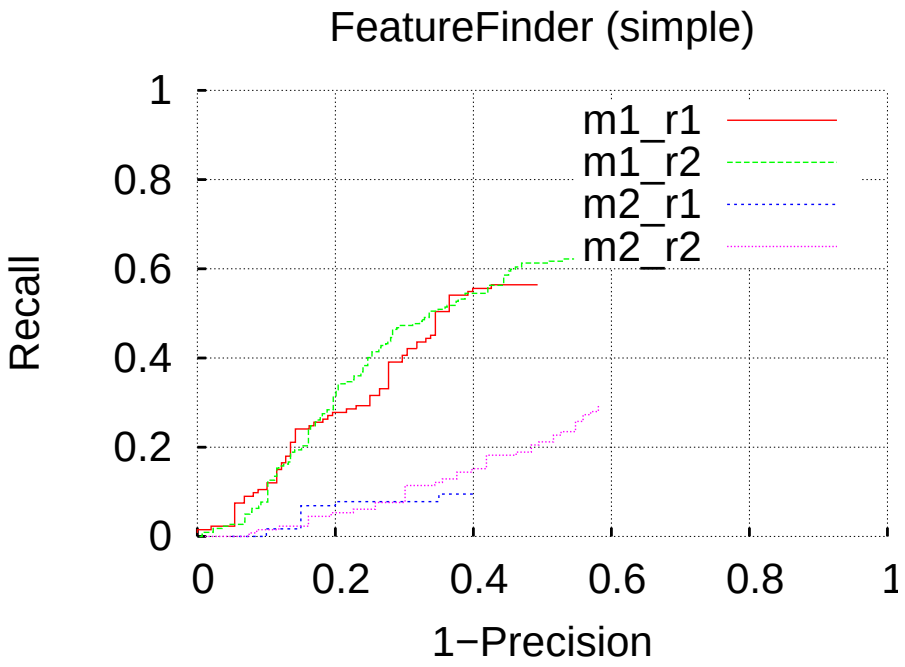


Figure D.3.: Plot of precision against recall for FeatureFinder (simple) results. Bipartitions of the output feature set ordered according to intensity were used to calculate corresponding precision and recall values.

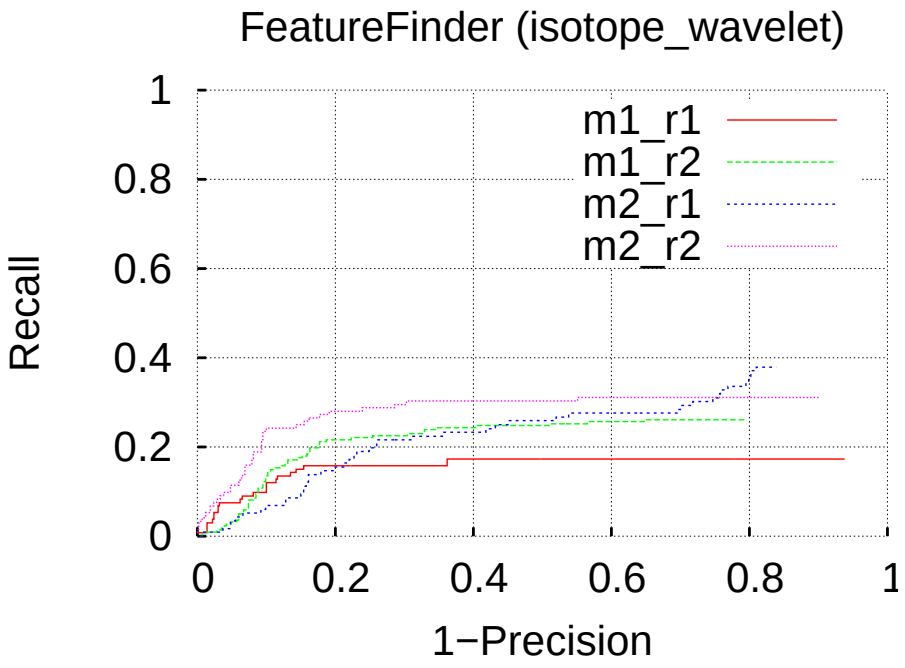


Figure D.4.: Plot of precision against recall for FeatureFinder (isotope_wavelet). Bipartitions of the output feature set ordered according to intensity were used to calculate corresponding precision and recall values.

parameter	m1	m2
seeder:min_intensity	0	0
seeder:signal_to_noise	5	5
seeder:SignalToNoise:max_intensity	-1	-1
seeder:SignalToNoise:auto_max_stdev_factor	3	3
seeder:SignalToNoise:auto_max_percentile	95	95
seeder:SignalToNoise:auto_mode	0	0
seeder:SignalToNoise:win_len	200	200
seeder:SignalToNoise:bin_count	30	30
seeder:SignalToNoise:min_required_elements	10	10
seeder:SignalToNoise:noise_for_empty_window	1E+20	1E+20
extender:dist_mz_up	5	5
extender:dist_mz_down	2	2
extender:dist_rt_up	30	30
extender:dist_rt_down	30	30
extender:priority_thr	-0.1	-0.1
extender:intensity_factor	0.2	0.2
fitter:fit_algorithm	simple	simple
fitter:max_iteration	500	500
fitter:deltaAbsError	0.0001	0.0001
fitter:deltaRelError	0.0001	0.0001
fitter:tolerance_stdev_bounding_box	3	3
fitter:intensity_cutoff_factor	0.05	0.05
fitter:feature_intensity_sum	1	1
fitter:min_num_peaks:final	20	20
fitter:min_num_peaks:extended	40	40
fitter:rt:interpolation_step	2	2
fitter:mz:interpolation_step	0.03	0.03
fitter:mz:model_type:first	1	1
fitter:mz:model_type:last	4	4
fitter:quality:type	Correlation	Correlation
fitter:quality:minimum	0.5	0.5
fitter:isotope_model:stdev:first	0.06	0.06
fitter:isotope_model:stdev:last	0.14	0.14
fitter:isotope_model:stdev:step	0.02	0.02
fitter:isotope_model:averagines:C	0.0443	0.0443
fitter:isotope_model:averagines:H	0.007	0.007
fitter:isotope_model:averagines:N	0.0012	0.0012
fitter:isotope_model:averagines:O	0.013	0.013
fitter:isotope_model:averagines:S	0.00037	0.00037
fitter:isotope_model:isotope:trim_right_cutoff	0.001	0.001
fitter:isotope_model:isotope:maximum	100	100
fitter:isotope_model:isotope:distance	1.00049	1.00049

Table D.8.: FeatureFinder (simple) algorithm parameters used in this evaluation.

Bibliography

[1] R. Dahm and F. Miescher. Discovering DNA: Friedrich Miescher and the early years of nucleic acid research. *Hum. Genet.*, 122:565–581, 2008.

[2] International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931–945, 2004.

[3] V. Le Texier, J. J. Riethoven, V. Kumanduri, C. Gopalakrishnan, F. Lopez, D. Gautheret, and T. A. Thanaraj. AltTrans: transcript pattern variants annotated for both alternative splicing and alternative polyadenylation. *BMC Bioinformatics*, 7:169, 2006.

[4] M. R. Wilkins, C. Pasquali, R. D. Appel, K. Ou, O. Golaz, J. C. Sanchez, J. X. Yan, A. A. Gooley, G. Hughes, I. Humphery-Smith, K. L. Williams, and D. F. Hochstrasser. From proteins to proteomes: large scale protein identification by two-dimensional electrophoresis and amino acid analysis. *Biotechnology (N.Y.)*, 14:61–65, 1996.

[5] M. O. Dayhoff and R. V. Eck. MASSPEC: a computer program for complete sequence analysis of large proteins from mass spectrometry data of a single sample. *Comput. Biol. Med.*, 1:5–28, 1970.

[6] Kapp, E. and Schütz, F. Overview of tandem mass spectrometry (MS/MS) database search algorithms. *Curr. Protoc. Protein Sci.*, Chapter 25:Unit25.2, 2007.

[7] Waters Corporation – MassLynx. http://www.waters.com/.

[8] GE Healthcare – DeCyder. http://www.gelifesciences.com/.

[9] D. N. Perkins, D. J. Pappin, D. M. Creasy, and J. S. Cottrell. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis*, 20:3551–3567, 1999.

[10] A. Keller, J. Eng, N. Zhang, X. J. Li, and R. Aebersold. A uniform proteomics MS/MS analysis platform utilizing open XML file formats. *Mol. Syst. Biol.*, 1:2005.0017, 2005.

[11] C. A. Smith, E. J. Want, G. O'Maille, R. Abagyan, and G. Siuzdak. XCMS: processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching, and identification. *Anal. Chem.*, 78:779–787, 2006.

[12] M. Bellew, M. Coram, M. Fitzgibbon, M. Igra, T. Randolph, P. Wang, D. May, J. Eng, R. Fang, C. Lin, J. Chen, D. Goodlett, J. Whiteaker, A. Paulovich, and M. McIntosh. A suite of algorithms for the comprehensive analysis of complex protein mixtures using high-resolution LC-MS. *Bioinformatics*, 22:1902–1909, 2006.

[13] X. J. Li, E. C. Yi, C. J. Kemp, H. Zhang, and R. Aebersold. A software suite for the generation and comparison of peptide arrays from sets of data collected by liquid chromatography-mass spectrometry. *Mol. Cell. Proteomics*, 4:1328–1340, 2005.

[14] M. Katajamaa and M. Oresic. Processing methods for differential analysis of LC/MS profile data. *BMC Bioinformatics*, 6:179, 2005.

[15] Mueller, L. N. and Rinner, O. and Schmidt, A. and Letarte, S. and Bodenmiller, B. and Brusniak, M. Y. and Vitek, O. and Aebersold, R. and Müller, M. SuperHirn – a novel tool for high resolution LC-MS-based peptide/protein profiling. *Proteomics*, 7:3470–3480, 2007.

[16] J. Cox and M. Mann. MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification. *Nat. Biotechnol.*, 26:1367–1372, 2008.

[17] M. W. Senko, S. C. Beu, and F. W. McLafferty. Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions. *J. Am. Soc. Mass. Spectrom.*, 6(4):229–233, 1995.

[18] M. Gilar, K. J. Fountain, Y. Budman, U. D. Neue, K. R. Yardley, P. D. Rainville, R. J. Russell, and J. C. Gebler. Ion-pair reversed-phase high-performance liquid chromatography analysis of oligonucleotides: retention prediction. *J. Chromatogr. A*, 958:167–182, 2002.

[19] E. de Hoffmann and V. Stroobant. *Mass Spectrometry: Principles and Applications*. Wiley & Sons, 2004.

[20] J. B. Fenn, M. Mann, C. K. Meng, S. F. Wong, and C. M. Whitehouse. Electrospray ionization for mass spectrometry of large biomolecules. *Science*, 246:64–71, 1989.

[21] N. B. Cech and C. G. Enke. Practical implications of some recent studies in electrospray ionization fundamentals. *Mass Spectrom. Rev.*, 20:362–387, 2001.

[22] M Karas and U. Bahr. Laser Desorption Ionization Mass Spectrometry of Large Biomolecules. *Trends Anal. Chem.*, 9:321–5, 1990.

[23] W. Paul and H. Steinwedel. Apparatus For Separating Charged Particles Of Different Specific Charges. US patent 2939952.

[24] W. E. Stephens. A Pulsed Mass Spectrometer with Time Dispersion. *Phys. Rev.*, 69:691, 1946.

[25] A. Makarov. Electrostatic axially harmonic orbital trapping: a high-performance technique of mass analysis. *Anal. Chem.*, 72:1156–1162, 2000.

[26] D. J. Pappin, P. Hojrup, and A. J. Bleasby. Rapid identification of proteins by peptide-mass fingerprinting. *Curr. Biol.*, 3:327–332, 1993.

[27] R. Aebersold and M. Mann. Mass spectrometry-based proteomics. *Nature*, 422:198–207, 2003.

[28] P. Roepstorff and J. Fohlman. Proposal for a common nomenclature for sequence ions in mass spectra of peptides. *Biomed. Mass Spectrom.*, 11:601, 1984.

[29] J. A. Taylor and R. S. Johnson. Implementation and uses of automated de novo peptide sequencing by tandem mass spectrometry. *Anal. Chem.*, 73:2594–2604, 2001.

[30] A. Frank and P. Pevzner. PepNovo: de novo peptide sequencing via probabilistic network modeling. *Anal. Chem.*, 77:964–973, 2005.

[31] R. G. Sadygov and J. R. Yates. A hypergeometric probability model for protein identification and validation using tandem mass spectral data and protein sequence databases. *Anal. Chem.*, 75:3792–3798, 2003.

[32] S. P. Gygi, B. Rist, S. A. Gerber, F. Turecek, M. H. Gelb, and R. Aebersold. Quantitative analysis of complex protein mixtures using isotope-coded affinity tags. *Nat. Biotechnol.*, 17:994–999, 1999.

[33] S. E. Ong, B. Blagoev, I. Kratchmarova, D. B. Kristensen, H. Steen, A. Pandey, and M. Mann. Stable isotope labeling by amino acids in cell culture, SILAC, as a simple and accurate approach to expression proteomics. *Mol. Cell. Proteomics*, 1:376–386, 2002.

[34] L. R. Zieske. A perspective on the use of iTRAQ reagent technology for protein complex and profiling studies. *J. Exp. Bot.*, 57:1501–1508, 2006.

[35] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.

[36] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[37] A. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Autom. Remote Control*, 25:821–837, 1964.

[38] Subversion – an open source version control system. http://subversion.tigris.org/.

[39] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York Inc., 2001.

[40] C. Cortes and V. N. Vapnik. Support vector networks. *Mach. Learn.*, pages 273–297, 1995.

[41] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Comput.*, 12(5):1207–1245, 2000.

[42] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *Bioinformatics*, 42:59–66, 1988.

[43] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16:412–424, 2000.

[44] Ziv H., D. J. Richardson, and R. Klsch. The Uncertainty Principle in Software Engineering, 1996.

[45] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process.* Addison-Wesley Longman, 1999.

[46] K. Beck and C. Andres. *Extreme programming explained – embrace change.* Addison-Wesley Longman, 2004.

[47] K. Schwaber and M. Beedle. *Agile software development with scrum.* PEARSON STUDIUM, 2008.

[48] J. Rumbaugh, I. Jacobson, and G. Booch. *The unified modeling language reference manual.* Addison Wesley, 1999.

[49] E. Gamma, R. Helm, and R. E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* Cambridge University Press, 1995.

[50] D. J. Armstrong. The quarks of object-oriented development. *Commun. ACM*, 49(2):123–128, 2006.

[51] Microsoft Visual Studio. http://www.microsoft.com/express/.

[52] Eclipse. http://www.eclipse.org/.

[53] G. J. Myers. *The Art of Software Testing.* Wiley, second edition edition, 2004.

[54] CMake – Cross Platform Make. http://www.cmake.org/.

[55] Apache Ant. http://ant.apache.org/.

[56] CDash – an open source, web-based testing server. http://www.cdash.org/.

[57] Bamboo. /http://www.atlassian.com/software/bamboo/.

[58] Software Engineering – Software Life Cycle Processes – Maintenance. International Organization for Standardization, ISO/IEC 14764:2006, 2006.

[59] P. Grubb and A. T. Armstrong. *Software Maintenance: Concepts and Practice.* World Scientific Pub Co, 2003.

[60] J. Listgarten and A. Emili. Statistical and computational methods for comparative proteomic profiling using liquid chromatography-tandem mass spectrometry. *Mol. Cell. Proteomics*, 4:419–434, 2005.

[61] Kapp, E. A. and Schütz, F. and Connolly, L. M. and Chakel, J. A. and Meza, J. E. and Miller, C. A. and Fenyo, D. and Eng, J. K. and Adkins, J. N. and Omenn, G. S. and Simpson, R. J. An evaluation, comparison, and accurate benchmarking of several publicly available MS/MS search algorithms: sensitivity and specificity analysis. *Proteomics*, 5:3475–3490, 2005.

[62] L. N. Mueller, M. Y. Brusniak, D. R. Mani, and R. Aebersold. An assessment of software solutions for the analysis of mass spectrometry based quantitative proteomics data. *J. Proteome Res.*, 7:51–61, 2008.

[63] K. L. Busch. Chemical Noise in Mass Spectrometry. *Spectroscopy*, 17:32–36, 2002.

[64] Lange, E. and Gröpl, C. and Reinert, K. and Kohlbacher, O. and Hildebrandt, A. High-accuracy peak picking of proteomics data using wavelet techniques. *Pac. Symp. Biocomput.*, pages 243–254, 2006.

[65] K. C. Leptos, D. A. Sarracino, J. D. Jaffe, B. Krastins, and G. M. Church. MapQuant: open-source software for large-scale protein quantification. *Proteomics*, 6:1770–1782, 2006.

[66] M. Katajamaa, J. Miettinen, and M. Oresic. MZmine: toolbox for processing and visualization of mass spectrometry based molecular profile data. *Bioinformatics*, 22:634–636, 2006.

[67] P. Du, R. Sudha, M. B. Prystowsky, and R. H. Angeletti. Data reduction of isotope-resolved LC-MS spectra. *Bioinformatics*, 23:1394–1400, 2007.

[68] O. Kohlbacher, K. Reinert, C. Gröpl, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, and M. Sturm. TOPP – the OpenMS proteomics pipeline. *Bioinformatics*, 23:191–197, 2007.

[69] C. Gröpl, E. Lange, K. Reinert, O. Kohlbacher, M. Sturm, C. G. Huber, B. Mayr, and C. Klein. Algorithms for the automated absolute quantification of diagnostic markers in complex proteomics samples. In *Procceedings of CompLife 2005*, Lecture Notes in Bioinformatics, pages 151–163. Springer, Heidelberg, 2005.

[70] Schulz-Trieglaff, O. and Hussong, R. and Gröpl, C. and Leinenbach, A. and Hildebrandt, A. and Huber, C. and Reinert, K. Computational quantification of peptides from LC-MS data. *J. Comput. Biol.*, 15:685–704, 2008.

[71] R. Hussong, B. Gregorius, A. Tholey, and A. Hildebrandt. Highly accelerated feature detection in proteomics data sets using modern graphics processing units. *Bioinformatics*, 2009.

[72] L. Chen, S. K. Sze, and H. Yang. Automated intensity descent algorithm for interpretation of complex high-resolution mass spectra. *Anal. Chem.*, 78:5006–5018, 2006.

[73] R. Hussong, A. Tholey, and A. Hildebrandt. Efficient Analysis of Mass Spectrometry Data Using the Isotope Wavelet. In *Proceedings of the Third International Symposium on Computational Life Science*, AIP Conference Proceedings Volume 940, pages 139–49. American Institute of Physics, 2007.

[74] V. B. Di Marco and G. G. Bombi. Mathematical functions for the representation of chromatographic peaks. *J. Chromatogr. A*, 931:1–30, 2001.

[75] S. C. Wang, C. M. Huang, and S. M. Chiang. Improving signal-to-noise ratios of liquid chromatography-tandem mass spectrometry peaks using noise frequency spectrum modification between two consecutive matched-filtering procedures. *J. Chromatogr. A*, 1161:192–197, 2007.

[76] J. J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In *Numerical Analysis*, pages 105–116. Springer, Berlin, 1977.

[77] GSL – GNU Scientific Library. http://www.gnu.org/software/gsl/.

[78] Schulz-Trieglaff, O. and Pfeifer, N. and Gröpl, C. and Kohlbacher, O. and Reinert, K. LC-MSsim–a simulation software for liquid chromatography mass spectrometry data. *BMC Bioinformatics*, 9:423, 2008.

[79] C. Lemmel, S. Weik, U. Eberle, J. Dengjel, T. Kratt, H. D. Becker, H. G. Rammensee, and S. Stevanovic. Differential quantitative analysis of MHC ligands by mass spectrometry using stable isotope labeling. *Nat. Biotechnol.*, 22:450–454, 2004.

[80] Weinzierl, A. O. and Lemmel, C. and Schoor, O. and Müller, M. and Krüger, T. and Wernet, D. and Hennenlotter, J. and Stenzl, A. and Klingel, K. and Rammensee, H. G. and Stevanovic, S. Distorted relation between mRNA copy number and corresponding major histocompatibility complex ligand density on the cell surface. *Mol. Cell. Proteomics*, 6:102–113, 2007.

[81] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Stat.*, 22(1):79–86, 1951.

[82] Sturm, M. and Bertsch, A. and Gröpl, C. and Hildebrandt, A. and Hussong, R. and Lange, E. and Pfeifer, N. and Schulz-Trieglaff, O. and Zerck, A. and Reinert, K. and Kohlbacher, O. OpenMS – an open-source software framework for mass spectrometry. *BMC Bioinformatics*, 9:163, 2008.

[83] R. Hoffmann, G. Bril, and L. Otvos. Prediction of retention times of peptide nucleic acids during reversed-phase high-performance liquid chromatography. *J. Chromatogr. A*, 814:111–119, 1998.

[84] D. Guo, C. T. Mant, A. K. Taneja, J. M. R. Parker, and R. S. Hodges. Prediction of peptide retention times in reversed phase HPLC: determination of retention coefficients of amino acid residues of model synthetic peptides. *J. Chromatogr.*, 359(143):499–518, 1986.

[85] J. L. Meek. Prediction of peptide retention times in high-pressure liquid chromatography on the basis of amino acid composition. *Proc. Natl. Acad. Sci. U.S.A.*, 77:1632–1636, 1980.

[86] K. Petritis, L. J. Kangas, B. Yan, M. E. Monroe, E. F. Strittmatter, W. J. Qian, J. N. Adkins, R. J. Moore, Y. Xu, M. S. Lipton, D. G. Camp, and R. D. Smith. Improved peptide elution time prediction for reversed-phase liquid chromatography-MS by incorporating peptide sequence information. *Anal. Chem.*, 78:5026–5039, 2006.

[87] A. A. Klammer, X. Yi, M. J. MacCoss, and W. S. Noble. Improving tandem mass spectrum identification using peptide retention time prediction across diverse chromatography conditions. *Anal. Chem.*, 79:6111–6118, 2007.

[88] N. Pfeifer, A. Leinenbach, C. G. Huber, and O. Kohlbacher. Statistical learning of peptide retention behavior in chromatographic separations: a new kernel-based approach for computational proteomics. *BMC Bioinformatics*, 8:468, 2007.

[89] Braun, A. and Little, D. P. and Köster, H. Detecting CFTR gene mutations by using primer oligo base extension and mass spectrometry. *Clin. Chem.*, 43:1151–1158, 1997.

[90] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. U.S.A.*, 74:5463–5467, 1977.

[91] D. Bumcrot, M. Manoharan, V. Koteliansky, and D. W. Sah. RNAi therapeutics: a potential new class of pharmaceutical drugs. *Nat. Chem. Biol.*, 2:711–719, 2006.

[92] S. L. Gelhaus, W. R. LaCourse, N. A. Hagan, G. K. Amarasinghe, and D. Fabris. Rapid purification of RNA secondary structures. *Nucleic Acids Res.*, 31:e135, 2003.

[93] C. G. Huber and G. N. Berti. Detection of partial denaturation in AT-rich DNA fragments by Ion-Pair Reversed-Phase Chromatography. *Anal. Chem.*, 68:2959–2965, 1996.

[94] L. C. Tan, P. W. Carr, and M. H. Abraham. Study of Retention in Reversed-Phase Liquid Chromatography Using Linear Solvation Energy Relationships I. The Stationary Phase. *J. Chromatogr. A*, 752(1-2):1–18, 1996.

[95] D. H. Mathews. Revolutions in RNA secondary structure prediction. *J. Mol. Biol.*, 359:526–532, 2006.

[96] E. Rivas and S. R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.*, 285:2053–2068, 1999.

[97] S. Wuchty, W. Fontana, I. L. Hofacker, and P. Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49:145–165, 1999.

[98] H. Isambert and E. D. Siggia. Modeling RNA folding paths with pseudoknots: application to hepatitis delta virus ribozyme. *Proc. Natl. Acad. Sci. U.S.A.*, 97:6515–6520, 2000.

[99] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatsh. Chem.*, 125:167–188, 1994.

[100] C. G. Huber, A. Premstaller, and H. Oberacher. Method and Apparatus for Separating Polynucleotides Using Monolithic Capillary Columns. patent application U. S.

[101] P. Rice, I. Longden, and A. Bleasby. EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet.*, 16:276–277, 2000.

[102] N. Sugimoto, S. Nakano, M. Yoneyama, and K. Honda. Improved thermodynamic parameters and helix initiation factor to predict stability of DNA duplexes. *Nucleic Acids Res.*, 24:4501–4505, 1996.

[103] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm/, 2001.

[104] ANDI/MS. http://sourceforge.net/projects/andi/.

[105] S. Orchard, H. Hermjakob, C. Taylor, P. A. Binz, C. Hoogland, R. Julian, J. S. Garavelli, R. Aebersold, and R. Apweiler. Autumn 2005 Workshop of the Human Proteome Organisation Proteomics Standards Initiative (HUPO-PSI) Geneva, September, 4–6, 2005. *Proteomics*, 6:738–741, 2006.

[106] P. G. Pedrioli, J. K. Eng, R. Hubley, M. Vogelzang, E. W. Deutsch, B. Raught, B. Pratt, E. Nilsson, R. H. Angeletti, R. Apweiler, K. Cheung, C. E. Costello, H. Hermjakob, S. Huang, R. K. Julian, E. Kapp, M. E. McComb, S. G. Oliver, G. Omenn, N. W. Paton, R. Simpson, R. Smith, C. F. Taylor, W. Zhu, and R. Aebersold. A common open representation of mass spectrometry data and its application to proteomics research. *Nat. Biotechnol.*, 22:1459–1466, 2004.

[107] E. Deutsch. mzML: a single, unifying data format for mass spectrometer output. *Proteomics*, 8:2776–2777, 2008.

[108] H. P. Benton, D. M. Wong, S. A. Trauger, and G. Siuzdak. XCMS2: processing tandem mass spectrometry data for metabolite identification and structural characterization. *Anal. Chem.*, 80:6382–6389, 2008.

[109] D. Kessner, M. Chambers, R. Burke, D. Agus, and P. Mallick. ProteoWizard: open source software for rapid proteomics tools development. *Bioinformatics*, 24:2534–2536, 2008.

[110] A. I. Nesvizhskii, A. Keller, E. Kolker, and R. Aebersold. A statistical model for identifying proteins by tandem mass spectrometry. *Anal. Chem.*, 75:4646–4658, 2003.

[111] A. Keller, A. I. Nesvizhskii, E. Kolker, and R. Aebersold. Empirical statistical model to estimate the accuracy of peptide identifications made by MS/MS and database search. *Anal. Chem.*, 74:5383–5392, 2002.

[112] X. J. Li, H. Zhang, J. A. Ranish, and R. Aebersold. Automated statistical analysis of protein abundance ratios from data generated by stable-isotope dilution and tandem mass spectrometry. *Anal. Chem.*, 75:6648–6657, 2003.

[113] Monroe, M. E. and Tolić, N. and Jaitly, N. and Shaw, J. L. and Adkins, J. N. and Smith, R. D. VIPER: an advanced software package to support high-throughput LC-MS peptide identification. *Bioinformatics*, 23:2021–2023, 2007.

[114] D. van Heesch. Doxygen – Source code documentation generator tool. http://www.stack.nl/ dimitri/doxygen/.

[115] SourceForge. http://sourceforge.net/.

[116] Programming languages – C++. American National Standards Institute, New York, INCITS/ISO/IEC 14882:2003, 2003.

[117] Xcode. http://developer.apple.com/TOOLS/xcode/.

[118] Microsoft Visual C++. http://msdn.microsoft.com/en-us/visualc/default.aspx.

[119] Qt: Cross-Platform Rich Client Development Framework. http://qt.nokia.com/products/.

[120] Xerces-C++. http://xml.apache.org/xerces-c/.

[121] COIN-OR: CoinMP. http://www.coin-or.org/projects/CoinMP.xml.

[122] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org/.

[123] NetCDF (network Common Data Form). http://www.unidata.ucar.edu/software/netcdf/.

[124] M. H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library.* Addison Wesley Pub Co Inc, 1998.

[125] J. Kaiser. Proteomics. Public-private group maps out initiatives. *Science*, 296:827, 2002.

[126] C. F. Taylor, N. W. Paton, K. S. Lilley, P. A. Binz, R. K. Julian, A. R. Jones, W. Zhu, R. Apweiler, R. Aebersold, E. W. Deutsch, M. J. Dunn, A. J. Heck, A. Leitner, M. Macht, M. Mann, L. Martens, T. A. Neubert, S. D. Patterson, P. Ping, S. L. Seymour, P. Souda, A. Tsugita, J. Vandekerckhove, T. M. Vondriska, J. P. Whitelegge, M. R. Wilkins, I. Xenarios, J. R. Yates, and H. Hermjakob. The minimum information about a proteomics experiment (MIAPE). *Nat. Biotechnol.*, 25:887–893, 2007.

[127] A. Savitzky and M. J. E. Golay. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Anal. Chem.*, 36:1627–1639, 1964.

[128] W. H. Press, S. A. Teykolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++: The art of scientific computing.* Cambridge University Press, 2002.

[129] P. Soille. *Morphological Image Analysis.* Springer, 1999.

[130] E. J. Breen, F. G. Hopwood, K. L. Williams, and M. R. Wilkins. Automatic poisson peak harvesting for high throughput protein identification. *Electrophoresis*, 21:2243–2251, 2000.

[131] D. M. Horn, R. A. Zubarev, and F. W. McLafferty. Automated reduction and interpretation of high resolution electrospray mass spectra of large molecules. *J. Am. Soc. Mass Spectrom.*, 11:320–332, 2000.

[132] O. Schulz-Trieglaff, R. Hussong, C. Gröpl, A. Hildebrandt, and K. Reinert. A Fast and Accurate Algorithm for the Quantification of Peptides from Mass Spectrometry Data. In *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology*, pages 473–487, 2007.

[133] D. H. Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognit.*, 13(2):111–122, 1981.

[134] G. C. Stockman, S. Kopstein, and S. Benett. Matching Images to Models for Registration and Object Detection via Clustering. *PAMI*, 4(3):229–241, 1982.

[135] Lange, E. and Gröpl, C. and Schulz-Trieglaff, O. and Leinenbach, A. and Huber, C. and Reinert, K. A geometric approach for the alignment of liquid chromatography-mass spectrometry data. *Bioinformatics*, 23:i273–281, 2007.

[136] Gärdén, P. and Alm, R. and Häkkinen, J. PROTEIOS: an open source proteomics initiative. *Bioinformatics*, 21:2085–2087, 2005.

[137] Hartler, J. and Thallinger, G. G. and Stocker, G. and Sturn, A. and Burkard, T. R. and Körner, E. and Rader, R. and Schmidt, A. and Mechtler, K. and Trajanoski, Z. MASPECTRAS: a platform for management and analysis of proteomics LC-MS/MS data. *BMC Bioinformatics*, 8:197, 2007.

[138] D. Radulovic, S. Jelveh, S. Ryu, T. G. Hamilton, E. Foss, Y. Mao, and A. Emili. Informatics platform for global proteomic profiling and biomarker discovery using liquid chromatography-tandem mass spectrometry. *Mol. Cell. Proteomics*, 3:984–997, 2004.

[139] Lange, E. and Gröpl, C. and Reinert, K. and Kohlbacher, O. and Hildebrandt, A. High-accuracy peak picking of proteomics data using wavelet techniques. *Pac. Symp. Biocomput.*, pages 243–254, 2006.

[140] L. Nilse, M. Sturm, D. Trudgian, M. Salek, P. Sims, K. Carroll, and S. J. Hubbard. SILACAnalyzer – a tool for differential quantitation of SILAC data. In *Sixth International Meeting on Computational Intelligence methods for Bioinformatics and Biostatistics*, volume accepted, 2009.

[141] S. Tanner, H. Shu, A. Frank, L. C. Wang, E. Zandi, M. Mumby, P. A. Pevzner, and V. Bafna. InsPecT: identification of posttranslationally modified peptides from tandem mass spectra. *Anal. Chem.*, 77:4626–4639, 2005.

[142] L. Y. Geer, S. P. Markey, J. A. Kowalak, L. Wagner, M. Xu, D. M. Maynard, X. Yang, W. Shi, and S. H. Bryant. Open mass spectrometry search algorithm. *J. Proteome Res.*, 3:958–964, 2004.

[143] R. Craig and R. C. Beavis. TANDEM: matching proteins with tandem mass spectra. *Bioinformatics*, 20:1466–1467, 2004.

[144] B. Kuster, M. Schirle, P. Mallick, and R. Aebersold. Scoring proteomes with proteotypic peptide probes. *Nat. Rev. Mol. Cell Biol.*, 6:577–583, 2005.

[145] Käll, L. and Storey, J. D. and MacCoss, M. J. and Noble, W. S. Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. *J. Proteome Res.*, 7:29–34, 2008.

[146] Mayr, B. M. and Kohlbacher, O. and Reinert, K. and Sturm, M. and Gröpl, C. and Lange, E. and Klein, C. and Huber, C. G. Absolute myoglobin quantitation in serum by combining two-dimensional liquid chromatography-electrospray ionization mass spectrometry and novel data analysis algorithms. *J. Proteome Res.*, 5:414–421, 2006.

[147] M. Sturm and O. Kohlbacher. TOPPView: an open-source viewer for mass spectrometry data. *J. Proteome Res.*, 8:3760–3763, 2009.

[148] X. J. Li, P. G. Pedrioli, J. Eng, D. Martin, E. C. Yi, H. Lee, and R. Aebersold. A tool to visualize and evaluate data obtained by liquid chromatography-electrospray ionization-mass spectrometry. *Anal. Chem.*, 76:3856–3860, 2004.

[149] Insilicos Viewer. http://www.insilicos.com/Insilicos_Viewer.html.

[150] P. Jones, R. G. Côté, L. Martens, A. F. Quinn, C. F. Taylor, W. Derache, H. Hermjakob, and R. Apweiler. PRIDE: a public repository of protein and peptide identifications for the proteomics community. *Nucleic Acids Res.*, 34:D659–663, 2006.

[151] OpenMS coding style. http://www-bs2.informatik.uni-tuebingen.de/services/OpenMS-release/html/coding_conventions.html.

[152] PHP: Hypertext Preprocessor. http://www.php.net/.

[153] K. Reinert, O. Kohlbacher, C. Gröpl, E. Lange, O. Schulz-Trieglaff, M. Sturm, and N. Pfeifer. OpenMS – a framework for quantitative hplc/ms-based proteomics. In *Computational Proteomics*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[154] O. Kohlbacher, S. Quinten, M. Sturm, B. M. Mayr, and C. G. Huber. Structure-activity relationships in chromatography: retention prediction of oligonucleotides with support vector regression. *Angew. Chem. Int. Ed. Engl.*, 45:7009–7012, 2006.

[155] M. Sturm, S. Quinten, C. G. Huber, and O. Kohlbacher. A statistical learning approach to the modeling of chromatographic retention of oligonucleotides incorporating sequence and secondary structure data. *Nucleic Acids Res.*, 35:4195–4202, 2007.