Analysis of Mass Spectrometry Data for Protein Identification in Complex Biological Mixtures

by

Marina Spivak

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2010

Leslie Greengard

Acknowledgements

I thank my collaborators Jason Weston, William Stafford Noble and Michael MacCoss for excellent supervision and friendship. I also thank my advisor Leslie Greengard for help with writing this dissertation, as well as for valuable scientific guidance and encouragement. Finally, many thanks to my parents and friends for moral support during all the difficult years in graduate school.

Abstract

Mass spectrometry is a powerful technique in analytical chemistry that was originally designed to determine the composition of small molecules in terms of their constituent elements. In the last several decades, it has begun to be used for much more complex tasks, including the detailed analysis of the amino acid sequence that makes up an unknown protein and even the identification of multiple proteins present in a complex mixture. The latter problem is largely unsolved and the principal subject of this dissertation.

The fundamental difficulty in the analysis of mass spectrometry data is that of ill-posedness. There are multiple solutions consistent with the experimental data and the data is subject to significant amounts of noise. In this work, we have developed application-specific machine learning algorithms that (partially) overcome this ill-posedness. We make use of labeled examples of a single class of peptide fragments and of the unlabeled fragments detected by the instrument. This places the approach within the broader framework of semi-supervised learning.

Recently, there has been considerable interest in classification problems of this type, where the learning algorithm only has access to labeled examples of a single class and unlabeled data. The motivation for such problems is that in many applications, examples of one of the two classes are easy and inexpensive

to obtain, whereas the acquisition of examples of a second class is difficult and labor-intensive. For example, in document classification, positive examples are documents that address specific subject, while unlabeled documents are abundant. In movie rating, the positive data are the movies chosen by clients, while the unlabeled data are all remaining movies in a collection. In medical imaging, positive (labeled) data correspond to images of tissue affected by a disease, while the remaining available images of the same tissue comprise the unlabeled data. Protein identification using mass spectrometry is another variant of such a general problem.

In this work, we propose application-specific machine learning algorithms to address this problem. The reliable identification of proteins from mixtures using mass spectrometry would provide an important tool in both biomedical research and clinical practice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation addresses computational and algorithmic issues that arise in the following general problem: given a complex protein mixture, identify the proteins using mass spectrometry techniques.

The algorithmic procedures used for mass spectrometry data analysis are closely tied to the specifics of the design of proteomics experiments. Proteins in the mixture are first broken up into smaller pieces (peptides) to minimize the effects of complicated protein chemistry. Then, the analysis of each peptide using the mass spectrometer yields a *spectrum* (a set of peptide sub-sequences) that represents a large portion of the full peptide sequence. The data analysis task consists in interpreting the sequences contained in the spectra, by matching the spectra to candidate peptides, and subsequently inferring the original

proteins from the resulting peptides (reviewed in [P. Hernandez and Appel, 2006, Steen and Mann, 2004]).

One of the major challenges in this procedure is the presence of high noise content in the spectra. Therefore, an essential feature of a robust interpretation algorithm is the accurate evaluation of the quality of the matches between spectra and peptides; that is, an estimate of the likelihood of the correctness of a match. A lot of effort in the mass spectrometry community has been devoted to building algorithms that discriminate between correct and incorrect peptide-spectrum matches [Keller et al., 2002, Choi and Nesvizhskii, 2008, Käll et al., 2007, Ding et al., 2008, Spivak et al., 2009a].

A widely accepted approach is to generate an artificial set of incorrect matches (i.e. negative examples) by creating matches to random peptide sequences that do not exist in nature [Moore et al., 2002, Klammer and MacCoss, 2006, Colinge et al., 2003]. The result is a proxy problem of discriminating between the matches to real peptides and to the fake, randomly generated peptides. One of the many assumptions of this approach is that matches to fake peptides will have some properties in common with incorrect matches to real peptides, since both came about by chance [Elias and Gygi, 2007]. In this formulation, the matches to fake peptides have negative labels and are easy to generate, while the labels of the matches to real peptides are to be assigned,

i.e. the matches to real peptides are unlabeled.

Chapter 2 of this dissertation is concerned with an overview of the methods involved in the protein mixture analysis. It outlines the experimental details of applying mass spectrometry techniques to complex biological molecules like proteins. It then describes some of the methods involved in the interpretation of the resulting data, emphasizing the approaches used in our work either as pre-processing steps or as benchmark algorithms.

The next two chapters discuss the algorithmic framework involved in learning with examples of a single class and unlabeled examples. Most algorithms designed for these situations build a classifier that attempts to minimize the number of misclassified examples over the whole data set [Zhang and Zuo, 2008]. However, in many problems, it is not enough to find a general discriminant function that classifies examples as belonging to a certain class; ordered rankings with high precision at the top of the list are much preferred [Usunier et al., 2009]. In our context, matches between peptides and spectra can be selected by ranking them based on a discriminant function and retaining the best instances, based on a statistical threshold [Storey and Tibshirani, 2003]. The top-scoring examples are of particular interest since they are likely to be correct.

In our work, we propose an algorithm that directly optimizes the number

of peptide-spectrum matches relative to a user-specific statistical confidence threshold. For this purpose, we define a loss function that allows a direct optimization of a part of the ROC curve. This idea is useful in a variety of ranking tasks, where the focus is on the top-scoring part of the whole data set. The most common such application is document ranking [Usunier et al., 2009].

In chapter 5, we build on this approach to develop an algorithm for protein identification that exploits the structural information in the mass spectrometry data that has generally been ignored by existing approaches. The overall analysis of mass spectrometry proteomics data consists of a sequence of sub-tasks:

1. Interpreting the sequences represented by spectra and matching them up with peptides.

2. Evaluating the quality of these matches and optionally choosing a subset of the best matches.

3. Inferring proteins from the peptide set by resolving the ambiguities caused by the fact that multiple proteins share peptide sub-sequences [Steen and Mann, 2004].

In most current approaches, each sub-task is solved by a stand-alone procedure

(for example, see for the first step Eng et al. [2008]; for the second step Keller et al. [2002], Elias et al. [2004], Käll et al. [2007]; for the third step Nesvizhskii et al. [2003], Zhang et al. [2007], Bern and Goldberg [2008]). By doing so, the existing algorithms ignore higher-level information available at the protein level.

We integrate most of these separate stages into a single optimization problem at the protein level, while remaining in the semi-supervised learning framework with one labeled class mixed with unlabeled data. We define a single discriminant function at the protein level and achieve improved results over the benchmark algorithms [Nesvizhskii et al., 2003, Zhang et al., 2007]. In addition, we use results from other experimental procedures to validate the set of proteins we identify [Holstege et al., 1998, Ghaemmaghami et al., 2003]. In the process, we show that peptide and protein identification are cooperative tasks and that solving the problem at the protein level achieves superior results to relying on the peptide level alone, justifying our approach.

In the final chapter, we work toward integrating the spectra/sequence matching problem into the protein-level formulation and present some open problems in the field.

Chapter 2

Mass Spectrometry For Proteomics

2.1 Introduction

Tandem mass spectrometry has been widely used for the analysis of individual protein sequences as well as the sequences of multiple proteins in a mixture. It replaced Edman degradation, a labor-intensive technique which required purification of a substantial amount of each protein, took many hours to complete and failed if proteins were acetylated or had other post-translational modifications. The advantages of mass spectrometry are that it does not require purification of each protein, has no problem identifying modified proteins and

takes minutes rather than hours to obtain sequence results [Steen and Mann, 2004].

Despite these advantages, mass spectrometry (MS) was not adopted in the proteomics context until approximately 1980, although it had been invented over a century ago and had been widely used in other tasks. One reason for this is the complicated chemistry of whole proteins. Proteins are not all soluble under the same conditions and many of the detergents used to handle proteins interfere with MS because they ionize well and are in huge excess relative to the proteins. In addition, rather than being interested in the mass-to-charge ratio of whole protein molecules, the objective of proteomics is more complex: to identify the constituent amino acid sequences of the proteins. At the level of the whole molecule, it is difficult to determine which protein might have given rise to a single measured mass. Finally, there was a fundamental obstacle in biological mass spectrometry - namely, it was not known how to transfer highly polar, completely non-volatile molecules with a mass of tens of kilo-Daltons (kDa) into the gas phase without destroying them. This problem was resolved by the invention of so-called 'soft' ionization techniques, such as matrix-assisted desorption/ionization (MALDI) and electrospray ionization. The inventor of the latter, John Bennett Fenn, received a share of the Nobel Prize for chemistry in 2002 [Steen and Mann, 2004].

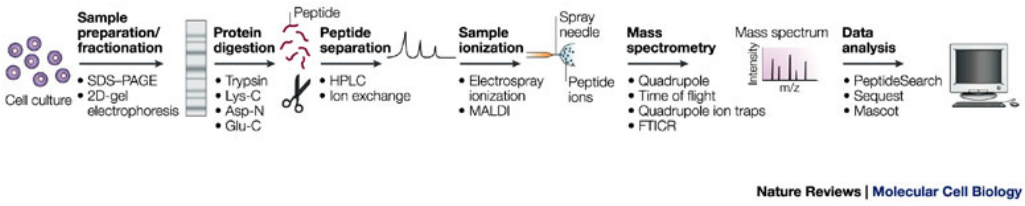


Figure 2.1: **Mass Spectrometry Pipeline**. A protein population is prepared from a biological source such as a cell culture. The proteins are first separated from a mixed sample by means of an SDS gel. They are then digested by a proteolytic enzyme such as trypsin. The resulting peptides are loaded onto an HPLC column coupled to a mass spectrometer. The peptides are ionized before entering the mass spectrometer [Steen and Mann, 2004].

2.2 Mass Spectrometry Pipeline

Due to these challenges, a series of pre-processing steps shown on Figure 2.1 were invented. This includes the digestion of the proteins by proteolytic enzymes (trypsin, chymotrypsin, elastase) to yield shorter pieces (peptides) that are subsequently sequenced. Although mass spectrometers can measure the mass of intact proteins, the sensitivity of the mass spectrometer for proteins is much lower than for peptides [Steen and Mann, 2004]. Mass spectrometry is, in fact, most efficient at obtaining sequence information from peptides that are up to 20 residues long. Digesting the whole protein into a set of peptides also

means that the physico-chemical properties of the protein, such as solubility and "stickiness", become irrelevant. It is for this reason that membrane proteins are quite amenable to MS-based proteomics, while in many other areas of protein science they are very difficult to work with because of their insolubility.

It is important to note that the resulting peptides are not introduced to the mass spectrometer all at once. Instead, they are first separated on the High Performance Liquid Chromatography (HPLC) column that is directly coupled to the mass spectrometer. The peptides are eluted (i.e. removed) from these columns using a solvent gradient of increasing organic content, so that the peptides emerge in order of their hydrophobicity. This separation step is crucial, since it is often assumed during the analysis that only a single peptide species enters the mass spectrometer during each cycle.

When a peptide species arrives at the end of the column, it flows through a needle. At the needle tip, the liquid is vaporized and the peptide is subsequently ionized by the action of a strong electric potential.

2.2.1 Tandem Mass Spectrometry (MS/MS) The pipeline described so far only allows for the determination of the mass-to-charge ratio of the whole peptide (called the precursor ion). To identify the proteins each peptide belongs to, the sequence of the peptide is necessary. For this purpose, mass

spectrometers either have two different acceleration chambers or make two runs for every peptide that arrives at the end of the HPLC columns. The first run determines the mass-to-charge ratio of the precursor ion. During the second run, gas phase peptide ions undergo collision-induced dissociation (CID) with molecules of an inert gas such as helium or argon and get broken into pieces. The result of this second run is a "spectrum" - that is, the masses of a large number of smaller peptide fragments.

The dissociation pathways (i.e. how peptides break into pieces) are strongly dependent on the collision energy, but the vast majority of instruments use low-energy CID ($< 100eV$). At low collision energies, fragmentation occurs mainly along the peptide backbone bonds, i.e the N-C bond between amino acids. At higher energies, fragments generated by breaking internal C-C bonds are also observed [Sadygov et al., 2004]. According to Roepstorff's nomenclature (Roepstorff and Fohlman,1984), ions are denoted as a, b, c when the charge is retained on the N-terminal side of the fragmented peptide, and x, y, z when the charge is retained on the C-terminal side (Figure 2.2).

Throughout the chromatographic run, the instrument will cycle through a sequence that consists of obtaining a mass-to-charge ratio of the species eluted from the HPLC column followed by obtaining tandem mass spectra of the most abundant species of peptides. The concentration of the inert gas

involved in the collision is calibrated with the expectation that each molecule of a given peptide will break in a single place. Therefore, peptides are expected to dissociate into nested sets of fragments. If A,B,C,D,E,F and G represent peptide's full chain of amino acid residues, the peptide can dissociate into any of the pairs of b, y ions shown in figure 2.3. Each break generates a b and a y-ion, for example, ABC is expected to be complemented by DEF.

Finally, the processed MS/MS spectrum is composed of the precursor mass and charge state, as well as of a list of peaks. Each peak is characterized by two values: a measured fragment mass-to-charge ratio (m/z) and an intensity value that represents the number of detected fragments [P. Hernandez and Appel, 2006]. Since successive spectrum peaks should differ by one amino acid and almost each naturally occurring amino acid has a unique mass, we can deduce a peptide's sequence by calculating the difference in mass between the neighboring peaks (see Figure 2.4).

2.3 From Spectra to Proteins

The previous section described the motivation underlying the various processing steps involved in mass spectrometry. They were invented to circumvent the difficulties of working with complex protein mixtures while trying to learn detailed sequence information about the sample, using only a device that mea-

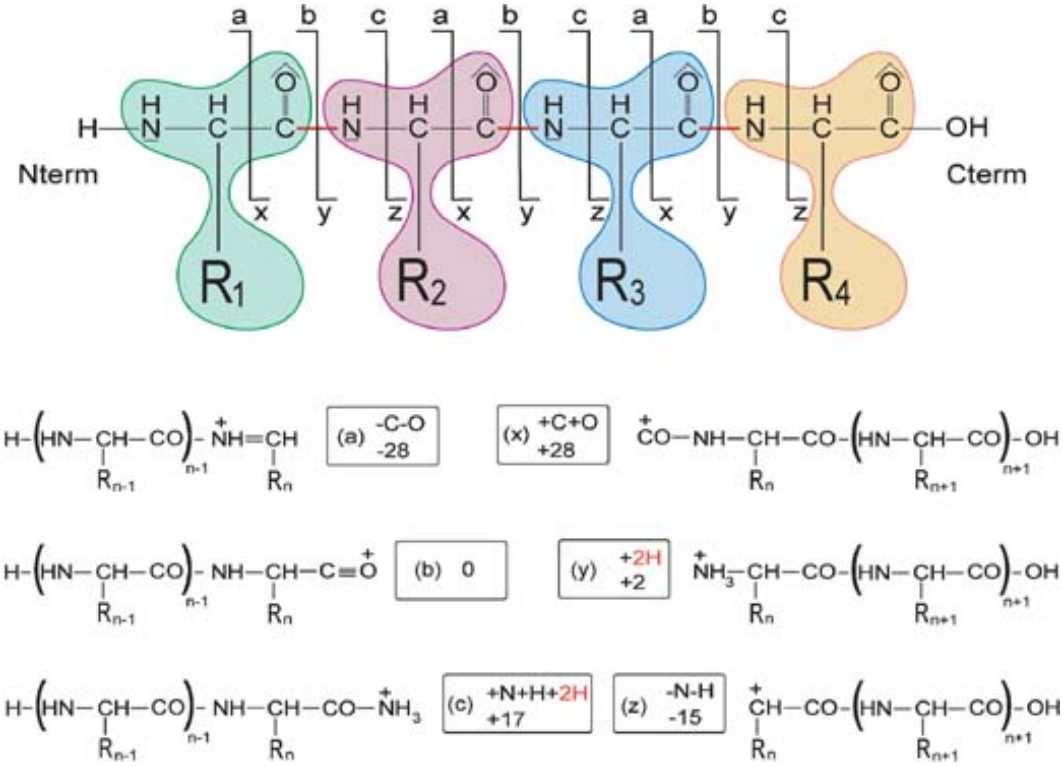


Figure 2.2: **Fragmentation of a Tetrapeptide Under the Collision Dissociation** [P. Hernandez and Appel, 2006]. At low collision energies, fragmentation mainly occurs along the peptide backbone bonds, i.e N-C bond between amino acids.

Figure 2.3: **Idealized Set of b- and y- Ions due to Collision Dissociation**. The expectation is that each molecule of a given peptide will break in a single place, so that many molecules of the same peptide will generate nested sets of fragments. The figure shows the ions for peptide ABCDEFG.

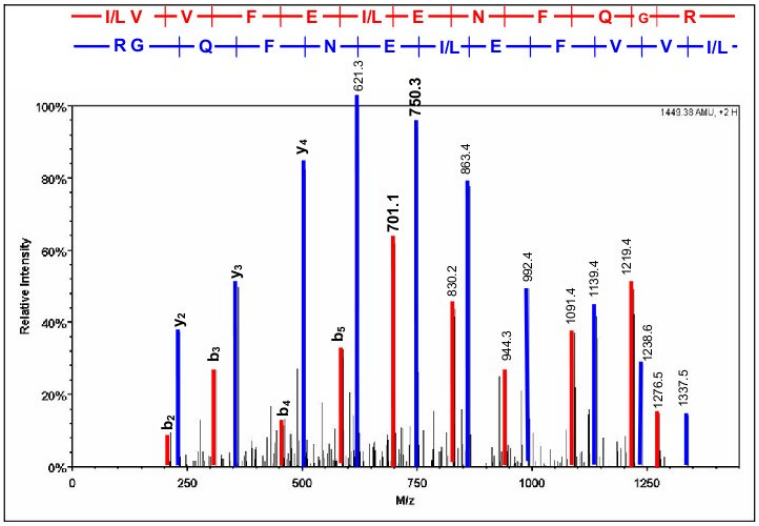


Figure 2.4: **Example of an Annotated MS Spectrum.** The information about the peptide sequence can be inferred from the mass differences between the peaks [Jonscher, 2005].

sures mass-to-charge ratio of a molecule. A large number of challenges remain in making optimal use of the raw data.

In principle, it is possible to determine "manually" the amino-acid sequence represented by a spectrum, by considering the mass difference between neighboring peaks in a series. However, the difficulty lies in the fact that the information in tandem-MS spectra is often not complete and that intervening peaks, which might or might not belong to the series, can confuse the analysis. The task is also complicated by substantial levels of noise in the spectra, arising from the fact that the peptides can break at several bond locations, and that some of the fragments might never be seen because they don't ionize. In addition, when considering large-scale protein screenings, it is unreasonable to imagine manually interpreting tens of thousands of spectra arising from a single experiment.

Therefore, computational methods are essential in inferring the peptides and proteins that give rise to observed spectra. These methods first use database search programs such as SEQUEST and CRUX to assign peptide sequences to spectra. Because they produce a significant number of incorrect peptide assignments, those results are validated in a second step using statistical approaches. The final task is to infer protein identities based on the (presumably reliable) peptide assignments. This final task is complicated by

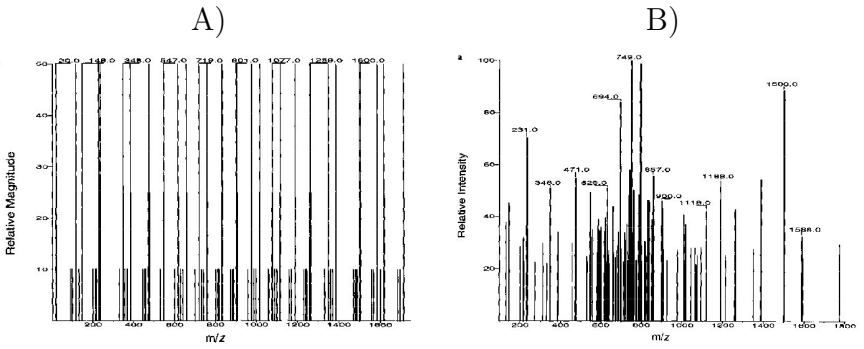


Figure 2.5: **Example of Theoretical and Observed Spectrum.** Theoretical (A) and experimental (B) spectra for the peptide sequence DLRSWTAADTAAQISQ [Eng et al., 1994]

the fact that many proteins share the same peptides.

2.3.1 Database Search SEQUEST [Eng et al., 1994] is a widely used software package that matches spectra to peptide sequences. Since it relies on full genomic sequence databases, it can only be used for organisms whose full genomic sequence is known, allowing prediction of open reading frames and protein sequences that can potentially exist in a cell of the organism.

The search algorithm relies on the fact that, given a peptide amino acid sequence, most of the mass-to-charge ratios of the peaks generated by this peptide can be predicted. (In contrast, the intensities of the peaks cannot be predicted as easily). SEQUEST virtually digests each protein in the database

with the enzyme used during the mass spectrometry experiment and represents each peptide by its theoretical spectrum (the algorithm assumes that peptides preferentially fragment into b- and y-ions [Sadygov et al., 2004].) The b- and y- ions in the series are assigned an intensity of 50, a window of 1 mass atomic unit is assigned intensity 25, and water and ammonium losses are assigned intensity 10 (see figure 2.5). These intensities have been chosen arbitrarily by the authors of the database search algorithm [Eng et al., 1994].

The algorithm compares each observed spectrum to the theoretical spectra obtained from the database using cross-correlation as a similarity measure [Eng et al., 2008]:

$$XCorr(x, y) = R_0 - (\sum_{\tau=-75}^{\tau=+75} R_\tau)/151$$

where x, y are theoretical and experimental spectra and

$$R_\tau = \sum x_i y_{i+\tau}$$

If the spectra are identical and can be aligned perfectly, only the first term, R_0, will contribute to the similarity measure. In all other cases, the contribution from exact alignment of the spectra is measures against other possible alignments resulting from sliding the experimental spectrum against the theoretical one.

The program ranks the candidate peptides of each individual spectrum

based on the cross-correlation (XCorr) measure and outputs several top matches.

One of the features of every existing database search algorithm is that it will produce at least one peptide match for each spectrum, regardless of its quality. However, there are several measurements that provide indications of how good the match is. Some of the most important are:

1. Xcorr, the cross-correlation function computed by the SEQUEST algorithm

2. δC_n, the relative difference between first and second highest XCorr for all peptides queried from the database for a given spectrum.

3. SpRank, a measure of how well the assigned peptide scored relative to those of similar mass in the database, using a preliminary correlation metric employed by the SEQUEST database.

4. The absolute value of the difference in mass between the precursor ion of the spectrum and the assigned peptide.

2.3.2 Target-Decoy Strategy As most database search engines return results even for "unmatchable" spectra, proteome researchers have devised ways to distinguish between correct and incorrect peptide identifications. The target-decoy strategy represents one such approach. The core idea behind this

strategy is that one can generate false peptide sequences (known not to be in the sample), so that any matches to them must be incorrect [Moore et al., 2002].

More precisely, one can shuffle [Klammer and MacCoss, 2006], reverse [Moore et al., 2002] or apply Markov-chain processes [Colinge et al., 2003] to the existing protein sequences in the database, creating a fake set of additional (decoy) sequences. Appending these decoy sequences to the set of real (target) sequences results in a database that is, say, twice the size of the original. MS/MS spectra are then searched against this single composite database and several top matches are retained for each spectrum for subsequent analysis. Assuming the decoy sequences are clearly labeled, any matches to the shuffled sequences are considered to be examples of incorrect matches and their properties should give a sense of the statistics of incorrect matches to the real database.

Major Assumptions of Target-Decoy Approach Despite the apparent simplicity of this method, some controversy surrounds its successful application. Therefore, it is important to note what assumptions are made when using this strategy [Elias and Gygi, 2007].

- The target and decoy databases do not overlap, if we are to trust that

the matches to decoy sequences are incorrect. Therefore, the strategy relies on the assumption that only a small fraction of all possible amino acid permutations occur in nature.

- Decoy peptide-spectrum matches are generated in a manner that is not trivially distinguishable from target peptide-spectrum matches.

2.3.3 Q-value Estimation The target-decoy strategy is an example of a situation where the labels of only a single (in this case negative) class are confidently assigned and the goal of the analysis is to assign a confidence measure to the labels of the other (positive) class.

These types of problems are often encountered in large-scale biological screening, where multiple independent events are tested against a null hypothesis to discriminate real biological effects from noisy data. In microarray experiments, the expression level of genes is tested against the background signal [Storey and Tibshirani, 2003]. In mass spectrometry protein screening, potentially positive matches between spectra and peptide sequences are tested for correctness against peptide-spectrum matches that are known to be negative because they were generated randomly [Elias and Gygi, 2007].

A widely used approach to this task, both in microarray and proteomic applications, is to assign each event a statistical confidence measure, a *q*-

value, designed to indicate how likely it is to be true when compared to the null hypothesis. Matches below a certain user-specified q-value threshold are considered statistically significant and are selected for further biological identification. In practical terms, a q-value threshold can be interpreted as the proportion of significant events that will turn out to be false leads [Storey and Tibshirani, 2003].

If we consider the problem of peptide-spectrum match validation in a classification framework, one possible approach to assigning q-values is as follows. Given a set of peptide-spectrum matches, we provisionally assign labels to them such that all the matches to the real (target) peptides receive labels of the positive class and the matches to the shuffled (decoy) peptides receive labels of the negative class. (Note that many of the positive label assignments may be incorrect).

We then build a classifier and use the ranking induced by the discriminant function to assign q-values (see figure 2.6). For each example in the ranked list, set a threshold such that all the examples ranking above it are considered "accepted" by the classifier, and all those below are considered "rejected". Then the q-value associated with this particular example is determined by the fractions of positives and negatives found above this particular threshold:

$$q = \frac{P_{FP}}{P_{TP}}$$

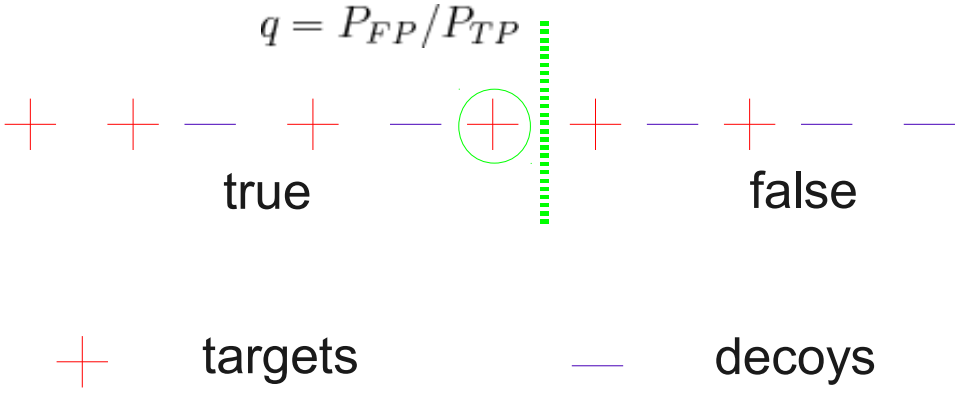


Figure 2.6: **Q-value assignment.** The ranking is induced by the discriminant function of a classifier. To assign a q-value to an example, set the threshold such that all the examples ranking above it are considered "accepted" by the classifier, and all those below are considered "rejected".

where P_{FP} is the probability of false positives (i.e. the fraction of negatives that were identified as positives), P_{TP} is the probability of true positives (i.e. the fraction of positives that were correctly identified as positives). Given a training set with N_p positives and N_n negatives and supposing that n_p is the number of positives identified correctly as positives and n_n is the number of negatives identified as positives, the q-value can be re-written as:

$$q = \frac{n_n}{n_p} * \frac{N_p}{N_n}$$

In this report, we extensively use the notion of a q-value threshold. Provided that each example has a q-value assigned to it, the position of such a

threshold in the ranked list identifies all the examples with q-values below a certain specified value q as "accepted" by the classifier. The q-value determining the position of the threshold provides a measure of confidence in the positive examples found above it. Often, the goal of an algorithm is to maximize the number of positive examples found above a particular q-value threshold, provided that no over-fitting occurs.

2.3.4 Peptide-Spectrum Match Verification: PeptideProphet and Percolator In any database search procedure, the score function that evaluates the quality of the match between an observed spectrum and a candidate peptide plays two complementary roles. First, the function ranks candidate peptides relative to a single spectrum, producing several top-scoring peptide-spectrum matches (PSMs) for each spectrum. Second, the function ranks the peptide-spectrum matches from different spectra with respect to one another. This latter, absolute ranking task is intrinsically more difficult than the relative ranking task. A perfect absolute ranking function is by definition also a perfect relative ranking function, but the converse is not true, because peptide-spectrum match scores may not be well calibrated from one spectrum to the next.

Several approaches have been developed to learn a peptide-spectrum match

scoring function from a single (real) data set. The expectation of these approaches is that, once determined, the discriminant function will generalize to all the subsequent mass spectrometry data sets. Typically, the input to these methods is the relative score of the peptide-spectrum match, as well as properties of the spectrum, the peptide and features representing the quality of the match. PeptideProphet [Keller et al., 2002], for example, uses four statistics computed by the SEQUEST database search algorithm mentioned in section 2.3.1 as inputs to a linear discriminant analysis classifier. The system is trained from labeled correct and incorrect peptide-spectrum matches derived from a purified sample of known proteins. Other approaches use alternative feature representations or classification algorithms, such as support vector machines (SVMs) [Anderson et al., 2003] or decision trees [Elias et al., 2004].

One drawback to these approaches based on building the scoring function from a single data set is that they may not generalize well across different machine platforms, chromatography conditions, etc. To combat this problem, several algorithms adjust the model parameters with respect to each new data set.

For example, the semi-supervised version of PeptideProphet [Choi and Nesvizhskii, 2008] couples a fixed linear discriminant function (learned once on a known data set) with a post-processor that maps the resulting unitless

discriminant score to an estimated probability. During this stage, the PeptideProphet uses the EM algorithm and includes decoy peptide-spectrum matches into the dataset, forcing them to belong to one of the probability distributions in the mixture model. The resulting probabilities are significantly more accurate than probabilities estimated in an unsupervised fashion.

The Percolator algorithm [Käll et al., 2007] takes the semi-supervised approach one step further. Rather than using a fixed discriminant function and employing semi-supervised learning as a post-processor, Percolator learns a new discriminant function on each mass spectrometry data set. Percolator uses an iterative, SVM-based algorithm, initially identifying a small set of high-scoring target peptide-spectrum matches, and then learning to separate these from the decoy matches. The learned classifier is applied to the entire set, and if new high-confidence matches are identified, then the procedure is repeated.

2.3.5 Composing the Protein Set After deriving a set of peptides up to a certain q-value threshold, all the proteins that could account for these peptides are collected. However, since a single peptide sequence can be mapped to multiple proteins in a database (because of the existence of homologous proteins, etc.), such naive protein assembly can substantially overstate the

number of proteins found in samples. Therefore, this protein list represents the maximum possible number of proteins for this database search that could explain the observed spectra.

The final stage of the analysis is to resolve the ambiguities of shared peptides and to produce a minimal protein list accounting for the observed spectra. This approach to the problem (i.e. parsimony) can be applied with varying levels of stringency. ProteinProphet [Nesvizhskii et al., 2003] introduces weights for the shared peptides; these weights determine how much each degenerate peptide will contribute to the probabilities of the proteins that contain it. Thus, ProteinProphet does not output a minimal set of proteins because each degenerate peptide is allowed to contribute to more than one protein it matches.

In contrast, IDPicker [Zhang et al., 2007] and ComByne [Bern and Goldberg, 2008] both output a parsimonious protein set by assigning each degenerate peptide to only one protein. IDPicker starts with filtering the peptide-spectrum matches based on user-specified q-value threshold using target-decoy strategy. It then creates a bipartite graph of the remaining peptides and the proteins containing them, identifies independent groups of proteins and applies a greedy set-cover algorithm to each group to create a minimal protein set. In this last step, each degenerate peptide is assigned in an arbitrary fashion to

exactly one of the proteins that contain it.

ComByne computes initial protein scores using all the peptide matches and assigns each degenerate peptide to the highest-scoring protein. It then computes the final protein score by incorporating retention time information and taking into account the fact that some peptide identification corroborate others of similar sequence.

2.4 Conclusion

This chapter gives a brief summary of the main algorithmic issues involved in mass spectrometry data interpretation in the domain of proteomics. Despite major progress in the field, the identification of the proteins in complex biological mixtures using mass spectrometry techniques remains an open problem. Some of the challenges are addressed in this dissertation.

One of the main observations in this work is that the dominant approach to the problem of breaking up the data analysis into 3 separate stand-alone steps - database search, peptide-spectrum match verification, protein set reconstruction - is not efficient since these three tasks are cooperative and the solution to each could benefit from information about the solution to the others. Chapters 5 and 6 are concerned with integrating these three tasks into a single problem solved directly at the protein level.

When protein identification is considered in the target-decoy setting, it falls into the category of *semi-supervised* problems with examples of a single class and unlabeled examples. This general type of problems remain an open area in machine learning research. In addition, while it is not clear that the mass spectrometry algorithms were developed with full awareness of the general developments in this domain, the analysis in Chapters 3 shows that mass spectrometry algorithms fit well into the current machine learning framework on this topic and possess many of the features of the general approaches to the problem.

In addition, the overwhelming majority of the algorithms developed for the problems with the examples of a single class and unlabeled examples are concerned with building a classifier on the whole data set. However, in many important applications, ranking with high accuracy at the top of the list is preferable. Chapter 4 formulates a ranking loss function that allows to choose the most relevant part of the list to optimize.

Finally, while both evaluation of the quality the peptide-spectrum matches and protein reconstruction are crucial steps in mass spectrometry data analysis, they can only use the information provided by the procedure that created the matches, i.e. the database search. The state-of-the-art database search algorithms contain many heuristic choices that were introduced by the original

developers of the software and were never examined further. This remains a domain with many open questions, only one of them is addressed in Chapter 6.

Chapter 3

PSM Verification

As described in the previous chapter, each peptide analyzed during a mass spectrometry run has its mass-to-charge ratio and a spectrum representing its amino acid sequence associated with it. Then, several sub-problems remain to be resolved in order to reconstruct the protein composition of the original mixture. The first sub-problem involves associating spectra to the peptide sequences they represent. The second involves estimating the correctness of these peptide-spectrum matches. The third consists in composing the final protein set. This chapter and the next address the algorithmic issues of the second sub-problem in the analysis pipeline: peptide-spectrum match verification in the target-decoy setting.

To be more precise, we are given a set of observed spectra $\mathcal{S} = \{s_1, \ldots, s_{N_\mathcal{S}}\}$

and a database $\mathcal{D}$ of real (target) proteins, as well as a database $\mathcal{D}'$ of shuffled (decoy) proteins. We perform a database search against each database separately or against a concatenated version of both databases. The search produces a set of peptide-spectrum matches (PSMs). Denoting the set of peptides as $\mathcal{E} = e_1, \ldots, e_{N_\mathcal{E}}$, the peptide-spectrum matches $\mathcal{M}$ are written as pairs $(e_j, s_k) \in \mathcal{M}$, each representing a match of peptide j to spectrum k. Note that, in general, we may opt to retain a single best-scoring peptide for each spectrum from target and decoy searches, or a small constant number of top-ranked matches per spectrum. We define a feature representation $\mathbf{x} \in \mathbb{R}^d$ for any given peptide-spectrum match pair (e, s). The task is to estimate which of the target peptide-spectrum matches are correct based on a statistical confidence measure.

Since the matches against the decoy database are known to be incorrect by design, they can confidently receive labels of the second class, $y = -1$. The matches against the real database, on the other hand, cannot receive confident labels of the first class, $y = 1$, and can be considered unlabeled data. Therefore, we identify the problem setup as a special type of *semi-supervised* setting [Chapelle et al., 2006] where only the examples of a single class have confidently assigned labels, and the rest of the examples are unlabeled.

Several machine learning methods have been proposed to address the re-

sulting classification task of distinguishing between correct and incorrect peptide-spectrum matches (PSMs). The two recent examples, PeptideProphet and Percolator, fit well into the general algorithmic framework developed for this type of *semi-supervised* problem reviewed in section 3.1. The analysis of these algorithms shows that PeptdieProphet tends to make excessively strong assumptions on the models it uses while Percolator fails to define a clear objective function it optimizes.

The next two chapters describe the following improvements to the state-of-the-art mass spectrometry algorithms. (1) We consider the problem in fully-supervised setting and propose a clear objective function, with intuitive reasons behind its choice. (2) We use tractable nonlinear models instead of linear models. (3) A method for directly optimizing the number of identified spectra at a specified q value threshold is proposed.

3.1 Learning with Positive and Unlabeled Examples

This section focuses on the machine learning algorithms that are primarily relevant to the mass spectrometry methods described in the subsequent sections. One of the common approaches (reviewed in Zhang and Zuo [2008]) to learning in settings where only labeled examples of one class are available takes a two-step strategy: 1) it identifies a set of *reliable* negative documents from

the unlabeled set; 2) it then builds a set of classifiers by iteratively applying a classification algorithm and then selecting a good classifier from the set [Liu et al., 2003, 2002, Li and Liu, 2003, Yu et al., 2002]. An alternative approach is to revert to the *fully-supervised* setting and to reformulate the problem as that of learning with high one-sided noise, treating the unlabeled set as noisy negative examples. For example, biased SVM method [Liu et al., 2003] directly uses the asymmetric cost formulation of the SVM algorithms for this purpose.

3.1.1 Two-Step Strategy: First Step

Bayes Classifier (BC) To identify *reliable* negative examples among the unlabeled data, Liu et al. [2003] build a two-class Bayes classifier and use it to assign labels of the negative class to some of the unlabeled examples, which are then used in the second step. (The positives are the examples of the first, labeled class).

The Bayes classifier is a generative model, because it makes an assumption that the data was generated from several classes by the following process: 1) a class c_j was chosen with probability $P(c_j)$; 2) an example $\mathbf{x}_i$ was drawn from the probability distribution of the examples belonging to this class, $p(\mathbf{x}_i|c_j)$. Then by Bayes' rule, the probability that some example $\mathbf{x}_i \in \mathcal{X}$ belongs to

class c_j is given by

$$P(c_j|\mathbf{x}_i) = \frac{P(c_j)p(\mathbf{x}_i|c_j)}{\sum_{c_r \in C} P(c_r)p(\mathbf{x}_i|c_r)} \tag{3.1}$$

where $C = \{c_j\}$ is a set of possible classes. For classification purposes, a newly presented example $\mathbf{x}_i$ is assigned to the class with the highest value of $P(c_j|\mathbf{x}_i)$.

Building a Bayes classifier involves estimating the model parameters using empirical data. It requires choosing a family of probability distributions to represent each class, $p(\mathbf{x}_i|c_j)$, and determining the parameters of these distributions. In addition, class probabilities $P(c_j)$ must be estimated.

A common approach to probabilistic model parameter estimation is the maximum likelihood principle. Based on a generative model with parameters θ, the likelihood of an example $\mathbf{x}_i \in \mathcal{X}$ is

$$P_\theta(\mathbf{x}_i) = \sum_{c_r \in C} P_\theta(c_r)p_\theta(\mathbf{x}_i|c_r) \tag{3.2}$$

And the likelihood of all the examples in the data set is

$$P_\theta(\mathcal{X}) = \prod_i P_\theta(\mathbf{x}_i) \tag{3.3}$$

The maximum likelihood principle suggests setting the parameters θ of the generative model such that the likelihood $P_\theta(\mathcal{X})$ is maximum.

For example, assuming that a labeled data set is available and that $p_\theta(\mathbf{x}_i|c_j)$ are modeled as Gaussian distributions, the maximum likelihood estimates will

amount to determining the mean and variance of the training data belonging to each class. The class probabilities $P_\theta(c_j)$ are given by the ratio of the number of examples of this class to the total number of examples [Bishop, 1995].

If the labels are not available or are not confidently assigned, it is possible to maximize the likelihood of a data set without recourse to labels using the EM algorithm [Dempster et al., 1977]. The EM assumes that the data was generated by a mixture model

$$P_\theta(\mathbf{x}_i) = \sum_{j=1}^{|C|} P_\theta(\alpha_j) p_\theta(\mathbf{x}_i|\alpha_j) \tag{3.4}$$

where α_j are the component probability distributions in the mixture. Provided that the training data complies with this assumption, the algorithm will recover the parameters of the mixture model in an *unsupervised* setting.

The parameters of the mixture model determined by the EM procedure can be used to build a Bayes classifier, by setting the probabilities of classes to equal the probabilities of the mixture components and the probability distributions in the mixture to represent the distributions of examples in the classes. However, this step cannot occur in an *unsupervised* setting, since some labeled examples are required to determine the correspondence between the components of the mixture model and classes [Nigam et al., 2000].

When the data set is composed of the examples of a single class and unlabeled examples, the two-class Bayes classifier can be build in either *fully-*

supervised setting, by assigning all the unlabeled data provisional labels of the second class, or in *unsupervised* setting, by ignoring all the labels. Once the parameters of the Bayes classifier are estimated, the *reliable* negative set is composed of only those unlabeled examples that are classified as belonging to the second class (i.e. $P(c_2|\mathbf{x}_i) \geq P(c_1|\mathbf{x}_i)$).

Spy Technique Due to potentially high levels of noise in the unlabeled data, both *fully-supervised* and *unsupervised* settings for Bayes classifier parameter estimation may give suboptimal results. There are attempts to ensure that the final set of negatives is in fact *reliable*. For example, the Spy technique [Liu et al., 2002] first selects $10\% - 15\%$ of the positive examples randomly and adds them to the unlabeled set. It then proceeds to use EM algorithm for the mixture of two Gaussian distributions. It initializes the first Gaussian with the remaining positive examples and the second Gaussian with mixed - 10% of the chosen positives and unlabeled - examples.

To select the reliable negative set after the EM, it contrasts the *a posteriori* probabilities $P(c_2|\mathbf{x}_i)$ of the positive "spies" with the *a posteriori* probabilities of the unlabeled examples and chooses those most likely to be true negatives. In effect, the "spies" play a role of the testing set and indicate that the EM gave reasonable results.

Rocchio Technique An alternative approach to identifying a *reliable* negative set is Rocchio method used in Roc-SVM algorithm [Li and Liu, 2003]. Building a Rocchio classifier is achieved by constructing a prototype vector $\vec{C}_j$ for each class.

$$\vec{C}_j = \alpha \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \frac{\mathbf{x}}{||\mathbf{x}||} - \beta \frac{1}{|\mathcal{X}| - |C_j|} \sum_{\mathbf{x} \in (\mathcal{X} - C_j)} \frac{\mathbf{x}}{||\mathbf{x}||} \tag{3.5}$$

where $\mathcal{X} = \{\mathbf{x}_1 ... \mathbf{x}_n\}$ is the set of examples, $|\mathcal{X}|$ is the total number of examples in the data set, $|C_j|$ is the number of examples in class j, α and β are parameters. During classification, the method uses the cosine similarity measure to compute how close each test example $\mathbf{x}_i$ is to each prototype vector.

Then the approach to identifying the *reliable* negative examples is the same as above: use the Rocchio classifier to classify the unlabeled data. The examples that are classified as negatives are considered reliable.

3.1.2 Two-Step Strategy: Second Step

The second step in the family of the two-stage algorithms occurs in the *semi-supervised* setting. The positive examples receive the labels of the first class, $y_i = 1$, the negatives, identified *reliably* during the first step, receive labels of the second class, $y_i = -1$, the rest of the unlabeled examples remain without labels.

Building Bayes Classifier in Semi-Supervised Setting. The S-EM algorithm [Liu et al., 2002] builds a Bayes classifier again during the second

stage, this time in a *semi-supervised* setting.

The parameter estimation again occurs based on the maximum likelihood principle. When the data set is composed of labeled L and unlabeled U sets of examples, the likelihood of the whole data set 3.3 can be rewritten as

$$P_\theta(\mathcal{X}) = \prod_{i \in L} P_\theta(\mathbf{x}_i, \mathbf{c}_i) \prod_{j \in U} P_\theta(\mathbf{x}_j) = \prod_{i \in L} \left[P_\theta(\mathbf{c}_i) P_\theta(\mathbf{x}_i|\mathbf{c}_i)\right] \prod_{j \in U} \left[\sum_{\mathbf{c}} P_\theta(\mathbf{c}) P_\theta(\mathbf{x}_j|\mathbf{c})\right]. \tag{3.6}$$

While the standard EM procedure occurs in *unsupervised* setting, the information about the labels associated with some of the examples can be introduced into the algorithm[Nigam et al., 2000]. First, the parameters of the mixture model are initialized based only on the labeled data. Since the initial parameter values strongly influence the course of the EM procedure, the additional information contained in the labeled data is significant. Second, during the E step of the EM algorithm, the labeled examples are fully assigned to the mixture components corresponding to the class labels set during initialization, (i.e. $P_\theta(c_j|\mathbf{x}_i) = 1$, where $i \in L$ and c_j corresponds to the class label of $\mathbf{x}_i$). The unlabeled data is treated in a standard fashion during the E step.

However, the accuracy of the resulting Bayes classifier is still dependent on the fact that the data should comply with the **assumptions** made by the parameter estimation algorithm:

- the data was indeed generated by a mixture model

- there is a one-to-one correspondence between mixture components in the model and classes given by the labels (i.e. α_j in equation 3.4 correspond to some c_j in equation 3.1).

The EM can suffer strongly from the misfit between the model assumptions and the data. For example, if the mixture components do not correspond to classes, the presence of unlabeled data can harm the classifier performance. One of the approaches to control for this problem is to model each class with multiple mixture components instead of a single mixture component [Nigam et al., 2000].

Iterative Application of SVM The Roc-SVM technique [Li and Liu, 2003] constructs a *reliable* negative set from unlabeled data using Rocchio technique. It then applies SVM classifier iteratively and re-assigns the negative labels in the process. During the construction of the set of SVM classifiers, the current classifier is applied to the unlabeled data and those examples classifying as negative contribute to the new set of negatives. These new labels are then used in the next iteration that builds the next SVM classifier.

This approach is not restricted by many of the assumptions necessary to build Bayes classifiers, since SVM classifiers are based on more flexible discriminative (rather than generative) models. However, it may suffer from the

fact that both the first and the second step of the two-step procedure just described are ill-posed problems. If the set of *reliable* negatives identified during the first step is inaccurate, it may strongly bias the subsequent solutions. Moreover, the same problem may arise when finding each new classifier in the series, since the algorithm re-labels the unlabeled set based on the current discriminant function in order to find the next one.

3.1.3 Fully Supervised Setting with Noisy Negatives This family of methods transforms the problem of learning with examples of one class and unlabeled data into learning with high one-sided noise by treating the unlabeled set as noisy negative examples.

Biased SVM For example, Liu et al. [2003] propose a Biased SVM (B-SVM) method which directly uses the asymmetric cost formulation of the SVM algorithm. This approach allows noise (or error) in both positive and negative label assignments and uses the biased SVM formulation with two parameters, C_+ and C_-, to weight positive and negative errors differently.

Suppose the training set contains L positive examples and K negative examples and y_i is the label of the ith example. Then the problem formulation is

$$Minimize : \frac{1}{2}\mathbf{w}^T\mathbf{w} + C_+ \sum_{i=1}^{L} \xi_i + C_- \sum_{i=1}^{K} \xi_i$$

$$Subject : y_i(\mathbf{w}^T\mathbf{x}_i + b) > 1 - \xi_i, i = 1, ..., n$$

$$\xi_i \geq 0, i = 1, .., n$$

The parameter values C_+ and C_- can be determined by cross-validation. Intuitively, it is preferable to give a big value to C_+ and a small value to C_- because the unlabeled set, which is assumed to be negative, may also contain positive data.

3.2 Algorithms for Mass Spectrometry

We now turn to the analysis of some of the state-of-the-art algorithms designed to validate the correctness of the matches between peptides and spectra: **PeptideProphet and Percolator**. As established earlier, the peptide-spectrum match validation is an instance of the problem of learning with examples of a single class and unlabeled examples, since the matches to shuffled database peptides are confidently labeled, but the labels of the matches to the real peptides remain to be assigned. The methods developed for the task fit well into the algorithmic framework described above.

3.2.1 PeptideProphet PeptideProphet [Keller et al., 2002] builds a Bayes classifier in the semi-supervised setting. It starts by determining the score of each peptide-spectrum match based on 4 indicators of match quality produced

by the database search (SEQUEST): (1) Cross-correlation, XCorr; (2) δC_n, the relative difference between first and second highest XCorr for all peptides queried from the database; (3) SpRank, a measure of how well the assigned peptide scored relative to those of similar mass in the database, using a preliminary correlation metric; (4) The absolute value of the difference between the mass of the peptide that generated the spectrum (measured during the mass spectrometry run) and the predicted mass of the assigned peptide in the database. The score of the peptide-spectrum match is given by a linear function

$$F(x_1, x_2, .., x_S) = w_0 + \sum_{k=1}^{S} w_k x_k \tag{3.7}$$

The parameters $\mathbf{w}$ of the function F were learned once on a model data set of known composition during the design of the algorithm. They are subsequently applied to all the new data sets to be analyzed.

However, the model as such proved to be not robust. Due to varying experimental conditions and the use of different mass spectrometry methods, the properties of peptide-spectrum matches vary substantially among data sets. Therefore, PeptideProphet also fits the parameters of Bayes classifier to each new data set, using the scores for peptide-spectrum matches based on the pre-defined function F:

$$P(c_1|F(\mathbf{x}_i)) = \frac{p(F(\mathbf{x}_i)|c_1)P(c_1)}{p(F(\mathbf{x}_i)|c_1)P(c_1) + p(F(\mathbf{x}_i)|c_2)P(c_2)} \tag{3.8}$$

where c_1, c_2 correspond to positive class containing correct matches and negative class containing incorrect matches. The probability distributions $p(F|c_1)$ and $p(F|c_2)$ are modeled as Gaussians. Their parameters as well as prior probabilities $P(c_1)$ and $P(c_2)$ are estimated using EM algorithm based on maximum likelihood principle (see sections 3.1.1 and 3.1.2 for details).

While PeptideProphet is a very popular tool in the mass spectrometry community, its main disadvantage is that it makes very strong **assumptions** on the model:

1. Using a pre-defined score function $F(x)$ that was learned on a small data set containing only 19 known proteins assumes strong similarities between mass spectrometry data sets. The designers of PeptideProphet recognize this as a weakness, since the latest improved version of the software includes an option that allows to heuristically adjust the original parameters of F based on each newly presented data set [Ding et al., 2008].

2. The probability distributions of correct and incorrect matches are modeled as Gaussian distributions, whereas the real distributions vary vastly among data sets.

3. The use of the EM algorithm for the estimation of the parameters of the

naive Bayes classifier implies the two underlying assumptions already mentioned in Nigam et al. [2000]: (1) that the data comes from a mixture model and (2) that the mixture model components correspond to classes.

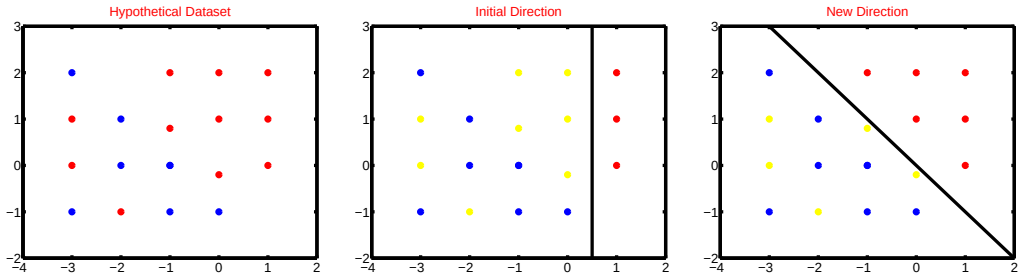
3.2.2 Percolator In contrast to PeptideProphet, Percolator [Käll et al., 2007] learns new parameters of the discriminant function for each new experimental data set. The algorithm significantly expands the feature representation of peptide-spectrum matches, attempting to maximize the use of the information provided by the database search (table 3.1). It then employs the SVM-based two-step strategy [Li and Liu, 2003] described in the section 3.1.2.

It first selects an initial set of positive peptide-spectrum matches based on the SEQUEST cross-correlation (XCorr) score (the negative set is confidently labeled, since it is composed of matches to the shuffled database). This step replaces the Rocchio technique in the Roc-SVM algorithm [Li and Liu, 2003]. This initial set of labeled examples is used to learn the first SVM classifier. Then the algorithm proceeds in an iterative manner: (1) it ranks the entire set of peptide-spectrum matches using the current SVM classifier and selects a new set of positives; (2) it then trains the next SVM classifier.

The **advantage** of Percolator is that it relaxes many of the assumptions made by PeptideProphet:

1	XCorr	Cross correlation between calculated and observed spectra
2	ΔC_n	Fractional difference between current and second best XCorr
3	ΔC_n^L	Fractional difference between current and fifth best XCorr
4	Sp	Preliminary score for peptide versus predicted fragment ion values
5	ln(rSp)	The natural logarithm of the rank of the match based on the Sp score
8	Mass	The observed mass $[M+H]^+$
6	ΔM	The difference in calculated and observed mass
7	abs(ΔM)	The absolute value of the difference in calculated and observed mass
9	ionFrac	The fraction of matched b and y ions
10	ln(NumSp)	The natural logarithm of the number of database peptides within the specified m/z range
11	enzN	Boolean: Is the peptide preceded by an enzymatic (tryptic) site?
12	enzC	Boolean: Does the peptide have an enzymatic (tryptic) C-terminus?
13	enzInt	Number of missed internal enzymatic (tryptic) sites
14	pepLen	The length of the matched peptide, in residues
15–17	charge1–3	Three Boolean features indicating the charge state
18–37	A, ..., Y	Counts of each of the 20 amino acids

Table 3.1: **Features For Representing Peptide-Spectrum Matches Used by the Percolator Algorithm** [Käll et al., 2007].



a) b) c)

Figure 3.1: **Schematic Description of the Percolator Algorithm** [Käll et al., 2007]. Percolator first selects an initial set of positive peptide-spectrum matches based on the SEQUEST cross-correlation score. It then proceeds in an iterative manner: 1) learns an SVM classifier; 2) re-assigns positive labels based on the discriminant function scores.

1. It uses a richer feature representation for peptide-spectrum matches (see 3.1).

2. It estimates the parameters of the discriminant function for each new data set.

3. It uses more flexible linear models with larger number of adjustable parameters, in comparison to the Gaussian distributions, with 2 parameters per class.

4. It takes a discriminative rather than generative approach to model estimation. In other words, it does not attempt to model how each example $\mathbf{x}$ occurred by modeling the probability distributions $P(\mathbf{x}|c_j)$ and then using them to estimate the *a posteriori* probabilities $P(c_j|\mathbf{x})$. Instead, it solves an easier problem of estimating directly the discriminant function which only predicts labels for testing set examples [Vapnik, 1995].

These advantages are corroborated by superior results of Percolator in comparison with PeptideProphet on the mass spectrometry data sets.

However, the **disadvantages** of the Percolator algorithm, as of all the two-step approaches, is that they are largely heuristic. It is unclear what exact loss function Percolator optimizes and whether the algorithm's iterative optimization process provably converges. As a result, the following problems

can arise:

1. The first step - identifying *reliable* examples of the second class - has multiple solutions, since the unlabeled data may contain the examples of many classes different from the first class, rather than a single "second" class. Therefore, it is an ill-posed problem which will give highly varying solutions depending on the algorithm used.

2. The second step - building a series of classifiers in the semi-supervised setting- may fail for the same reason that the unlabeled data may turn out to be very varied and noisy. The use of unlabeled data in the semi-supervised setting is only beneficial if this data complies with the assumptions made by the particular algorithm employed. If these assumptions do not hold, the use of unlabeled data may prove detrimental to the classifier performance [Chapelle et al., 2006].

Far from being hypothetical issues, these undesirable effects are well observed and the designers of the two-stage algorithms attempt to offer methods to deal with these problems. For example, for the first step, S-EM algorithm [Liu et al., 2002] proposes the Spy technique designed specifically to identify the unlabeled examples that "differ most" from the positive set (see section 3.1.1 for details). The Roc-SVM algorithm [Li and Liu, 2003] makes a check that

after the second-step - fitting a series of classifiers - the final classifier performs better than the first one, based on the original selection of the examples of the second class from the unlabeled data.

3.3 Fully-Supervised Approach

Motivated by these arguments, we seek to define an objective function suitable for dealing with noisy data sets such as mass spectrometry data sets, for example. One possible approach is to solve the problem in *semi-supervised* setting using TSVM algorithm [Vapnik, 1998]. Another possibility is to switch to the noisy *fully-supervised* setting. Here we pursue the second alternative with the motivation that non-linear TSVM algorithm takes longer to train than the algorithms we use in the *fully-supervised* setting in our work. In general, many *semi-supervised* algorithms scale as $(L+U)^3$, where L and U are numbers of labeled and unlabeled examples respectively [Weston, 2008], while standard stochastic gradient descent in *fully-supervised* setting scales as L. However, TSVM approach remains valid and we leave it as future work.

3.3.1 Motivation for Percolator Algorithm An obvious objective function formulation in the *fully-supervised* setting with noisy data already reported in the literature is the biased SVM formulation. Indeed, Liu et al.

[2003] try all the possible combinations of the two-step techniques presented in the literature and report that biased SVM gives superior results. It is easy to hypothesize that the improved results of biased SVM come from the existence of a clear objective function during the optimization procedure.

We test this conjecture by measuring the performance of the biased SVM in a *fully-supervised* setting against the two-step semi-supervised algorithm, Percolator, using a target-decoy search strategy (see section 2.3.2). For this experiment, we use a collection of spectra derived via microcapillary liquid chromatography MS/MS of a yeast whole cell lysate. These spectra were searched using SEQUEST [Eng et al., 1994] against one target database and two independently shuffled decoy databases, producing a collection of peptide-spectrum matches. For our experiments, we use 17- and 37-feature representations for peptide-spectrum matches given in Table 3.1. For each ranking of target peptide-spectrum matches, we use the corresponding collection of decoy peptide-spectrum matches to estimate q values (Section 2.3.3).

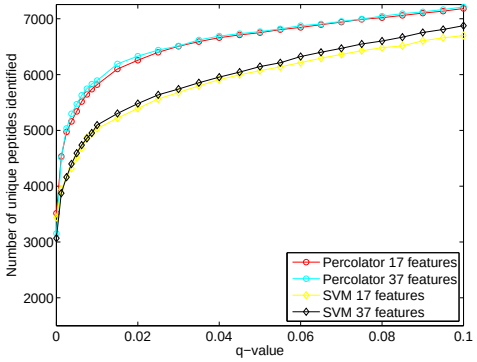
Our goal is to correctly identify as many target peptide-spectrum matches as possible for a given q value threshold. Therefore, in Figure 3.2, we plot the number of identified target peptide-spectrum matches as a function of the q value threshold. To ensure a valid experiment, we split the target and decoy peptide-spectrum matches into two equal parts. We train on the data set

composed of the first half of positives and negatives, and we use the second half of the data as a testing set. The q value estimates are derived from the test set, not the training set. This approach is more rigorous than the methodology employed in Käll et al. [2007], in which the positive examples were used both for training and testing. However, the similarity between Figure 3.2(A) and (B) indicates that over-fitting is not occurring. Nonetheless, in subsequent experiments, we retain a full separation of the training and testing sets.

Figure 3.2 shows that the conclusions by [Liu et al., 2003] that biased SVM performs better than all the two-step algorithms do not hold for mass spectrometry data sets, since the two-step algorithm, Percolator, performs better than the SVM classifier. We hypothesize that the data set is too noisy for the SVM to handle: we expect 50–90% incorrect (negative) examples among the matches to the target database. Therefore, to switch to the *fully-supervised* setting, we would like to define a robust loss function capable of handling high levels of noise in the data.

3.3.2 Choosing Loss Functions in Fully Supervised Setting Given a set of examples (PSMs) $(\mathbf{x}_1, ...\mathbf{x}_n)$ and corresponding labels $(y_1, ...y_n)$, the

(A) Yeast trypsin trainset



(B) Yeast trypsin testset

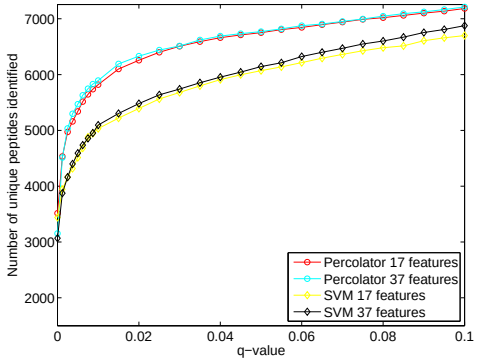


Figure 3.2: **Comparison Between Percolator and a Linear SVM**. Each panel plots the number of distinct peptides as a function of q value. The series correspond to two different algorithms, including variants of each that use 17 features and 37 features (see table 3.1).

goal is to find a discriminant function $f(\mathbf{x})$, such that

$$f(\mathbf{x}_i) > 0 \quad \text{if} \;\; y_i = 1$$

$$f(\mathbf{x}_i) < 0 \quad \text{if} \;\; y_i = -1.$$

To find $f(\mathbf{x})$ we first choose a parametrized family of functions and then search for the function in the family that best fits the empirical data. The quality of the fit is measured using a loss function $L(f(\mathbf{x}), y)$, which quantifies the discrepancy between the values of $f(\mathbf{x})$ and the labels y.

Initially, we consider the family of discriminant functions that are implemented by a linear model:

$$f(\mathbf{x}) = \sum_i w_i x_i + b \tag{3.9}$$

The possible choices of weights define the members of the family of discriminant functions.

To find the function $f(\mathbf{x})$ that best minimizes the loss L, we choose to use gradient descent, so the loss function itself must be differentiable. This requirement prevents us from simply counting the number of mistakes (mislabeled examples), which is called the zero-one loss. Typical differentiable loss functions include the squared loss, often used in neural networks [LeCun et al., 1998], the hinge loss, which is used in support vector machines [Cortes and Vapnik, 1995], and the sigmoid loss. These loss functions are illustrated in

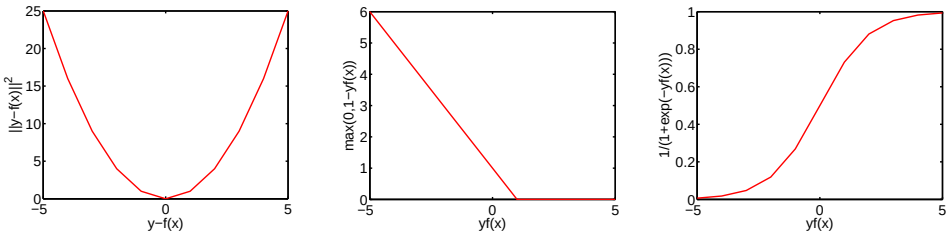


Figure 3.3: **Three Types of Loss Function.** Each panel plots the loss as a function of the difference in the true and predicted label. The squared loss $L(f(x), y) = (f(x) - y)^2$ is often used in regression problems, but also in classification [LeCun et al., 1998]. The hinge loss $L(f(x), y) = \max(0, 1 - yf(x))$ is used as a convex approximation to the zero-one loss in support vector machines [Cortes and Vapnik, 1995]. The sigmoid loss $L(f(x), y) = 1/(1 + e^{yf(x)})$ is perhaps less commonly used, but is discussed in Mason et al. [2000], Shen et al. [2003].

Figure 3.3.

In general, choosing an appropriate loss function is critical to achieving good performance. Insight into choosing the loss function comes from the problem domain. In the current setting, we can safely assume that a significant proportion of the peptide-spectrum matches produced by a given search algorithm are incorrect, either because the score function used to identify peptide-spectrum matches failed to accurately identify the correct peptide, or because the spectrum corresponds to a peptide not in the given database, to a peptide with post-translational modifications, to a heterogeneous population of peptides, or to non-peptide contaminants. Therefore, in this scenario, a desirable loss function should be robust with respect to the multiple false positives in the data. In other words, a desirable loss function will not strongly penalize misclassified examples if they are too far away from the separating hyperplane. Considering the loss functions in Figure 3.3, the sigmoid loss is the only function with the desired property: when $y_i f(\mathbf{x}) < -5$ the gradient is close to zero. The squared loss, on the other hand, has a larger gradient for misclassified examples far from the boundary than for examples close to the boundary, whereas the hinge loss penalizes examples linearly (it has a constant gradient if an example is incorrectly classified). We therefore conjecture that

the sigmoid loss function given by

$$L(f(\mathbf{x}), y) = 1/(1 + e^{yf(\mathbf{x})}) \tag{3.10}$$

should work much better than the alternatives.

We use stochastic gradient descent to optimize this loss function. We update the parameters $\mathbf{w}_k$ on iteration k as

$$\mathbf{w}_k = \mathbf{w}_k - \gamma \nabla_{\mathbf{w}} L(f(\mathbf{x}_k), y_k) \tag{3.11}$$

where γ is the learning rate [LeCun et al., 1998].

3.3.3 Supervised Learning Yields Performance Comparable to Percolator

Figure 3.4 compares the performance of ranking by XCorr, Percolator and a linear model trained using three different loss functions. As expected, the sigmoid loss dominates the other two loss functions that we considered, square loss and hinge loss.

In fact, the linear model with the sigmoid loss achieves almost identical results to the Percolator algorithm. This concordance can be explained in the following way. Percolator also uses a linear classifier (a linear SVM) with a hinge loss function. However, on each iteration *only a subset of the positive examples are used as labeled training data* according to the position of the hyperplane. The rest of the positive examples that have a small value of $y_i f(\mathbf{x}_i)$

(A) Trainset (B) Testset

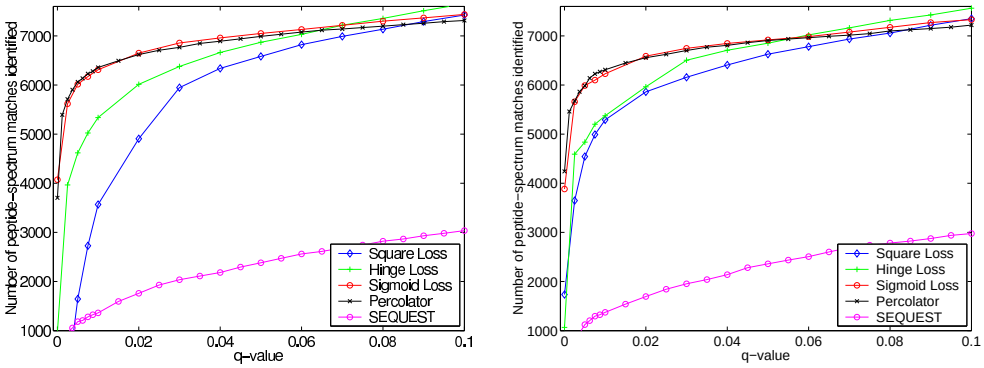


Figure 3.4: **Comparison of Loss Functions.** Each panel plots the number of accepted peptide-spectrum matches for the yeast (A) training set and (B) test set as a function of the q value threshold. Each series corresponds to one of the three loss functions shown in Figure 3.3, with series for Percolator and SEQUEST included for comparison.

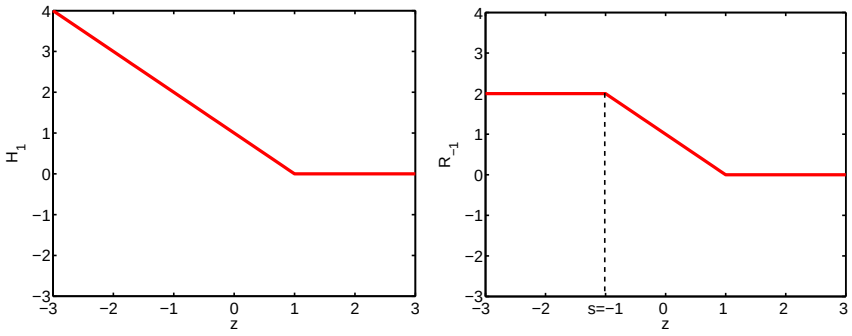


Figure 3.5: **"Cutting" the Hinge Loss Makes a Sigmoid-like Loss Called the *Ramp Loss*.** Making the hinge loss have zero gradient when $z = y_i f(x) < s$ for some chosen value s effectively makes a piece-wise linear version of a sigmoid function.

(i.e. are misclassified and are found far away from the decision boundary) are ignored during training. Consequently, one can say that their gradient is zero; hence, the hinge loss function is "cut" at a certain point so that it no longer linearly penalizes mistakes beyond this point, as shown in Figure 3.5. A cut hinge loss is effectively a piece-wise linear version of a sigmoid function. Indeed, such a cut hinge loss has been used before and is referred to as a *ramp loss* [Collobert et al., 2006]. By using a sigmoid loss function, we have thus developed a method that explains the heuristic choices of the Percolator algorithm but instead implements a direct, intuitive objective function.

3.4 Conclusion

This chapter considers the verification of the quality of the peptide-spectrum matches in the context of learning with labeled examples of a single class and unlabeled examples. It argues that some of the state-of-the-art algorithms developed in the mass spectrometry setting follow a widely-used two-step approach: 1) identifying a set of *reliable* examples of the second class; 2) building a series of classifiers in the standard *semi-supervised* setting, now using the examples of the first class and the reliable examples of the second class as a small labeled set and leaving the rest of the examples unlabeled.

We argue that both stages in the two-step formulation are ill-posed problems. Since the unlabeled set may contain examples of many different classes other than the first class, the resulting *reliable* examples of the second class will heavily depend on the specific algorithm used for the purpose. Similarly, the heterogeneous composition of the unlabeled set may interfere with the second stage of building a series of classifiers, since it introduces high amounts of noise that may interfere with learning.

In this chapter, we advocate considering this problem in a *fully-supervised* setting, where all the unlabeled examples receive the labels of the second class. An important benefit of using a *fully-supervised* approach is that it allows to define a clear, intuitive objective function whose minimization is

known to converge. However, we discover that the objective function must be chosen such that it is robust to high amounts of noise in the data and does not penalize very strongly the misclassified examples found far from decision boundary during training. Furthermore, the resulting classifier can be trained with tractable nonlinear models. This property is used in the next chapter, where we develop an algorithm to optimize a set of q-value thresholds of choice.

Chapter 4

Ranking

The approaches mentioned in the previous chapter search for a discriminant function that makes the minimum number of classification errors on the whole data set, while the ultimate goal of the analysis is to rank only the examples in the vicinity of a single (or several) q-value thresholds specified by a user. This latter task is equivalent to optimizing a part of the ROC curve corresponding to the points close to the q-value threshold of interest.

Figure 4.1 shows two possible ROC curves corresponding to different classifiers. Both are feasible since they satisfy the monotonicity constraint: the number of true positives does not decrease as the number of false positives increases. If a target q-value threshold is chosen, it defines a line given by $q = P_{FP}/P_{TP}$ on the same graph. While ROC curve B may correspond to a

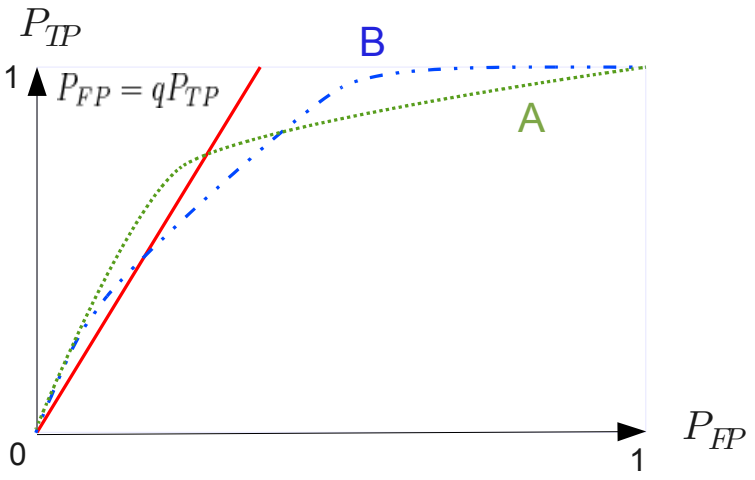


Figure 4.1: **Optimal ROC Curve for Q-value Threshold Optimization.** While ROC curve B may reflect better ranking performance of a classifier on an entire data set, ROC curve A is more desirable as an output of a q-value optimization procedure. ROC curve A crosses the line defined by the specified q-value threshold at a point that will give more true positives with the same statistical confidence level.

classifier with superior performance on the whole data set, ROC curve A is more optimal for the final goal, since it will give more positives at the specified q-value threshold. In the current work, we propose a family of loss functions that are based on ordered weighted average (OWA) operators [Yager, 1988, Usunier et al., 2009] and that are designed to allow directed optimization of a part of an ROC curve. We then describe a method, Q-ranker, for directly optimizing the number of identified peptide-spectrum matches at a specified

q value threshold. The ability to specify a desired statistical confidence threshold *a priori* is useful in practice and leads to significant improvement in the results over the algorithms that optimize the whole area under the ROC curve.

4.1 Previous Work: Review

4.1.1 Optimizing Area Under the ROC Curve A standard formulation for solving the ranking problem is the ranking SVM [Herbrich et al., 1999, Joachims, 2002], which can be stated as follows:

$$\min||\mathbf{w}||^2 \tag{4.1}$$

subject to

$$\mathbf{w}^\top \mathbf{x}_i \geq \mathbf{w}^\top \mathbf{x}_j + 1, \quad \text{if } y_i = 1 \text{ and } y_j = -1 \tag{4.2}$$

This algorithm re-orders the examples so that larger values of $\mathbf{w}^\top \mathbf{x}$ correspond to positive examples. Note that, compared to the classification problem, this formulation no longer has a threshold b, because a class label is no longer predicted, only an ordering.

Equivalently, we can rewrite this objective function as

$$\forall i, j : y_i = 1, y_j = -1$$

$$\min[||\mathbf{w}||^2 + C * \sum_{i=1}^{N_p} \sum_{j=1}^{N_n} \max(0, 1 - (f(\mathbf{x}_i) - f(\mathbf{x}_j)))] \tag{4.3}$$

where C is a parameter that can be chosen by cross-validation. This ranking formulation is equivalent to optimizing the area under the receiver operating characteristic (ROC) curve [Hanley and McNeil, 1982], and hence would optimize all q values at once. The optimization tries to satisfy every pairwise ordering constraint. Again, because we expect 50–90% of the positive examples to be false positives, this objective function will pay too much attention to these examples.

If optimization of only a certain q value is desired, then reordering of examples far beyond the q value threshold point on either side of the boundary will not have an effect on the q value of interest. It is more efficient to focus on a subset of examples in the vicinity of the q value cut-off and seek to re-order the examples specifically in this region.

Therefore, in the remainder of this section we focus on review of three different approaches to optimizing parts of the ranked lists generated by sorting examples based on the discriminant function of a classifier or, equivalently, parts of ROC curves.

4.1.2 OWA-based Optimization Usunier et al. [2009] aim to address tasks that require ranked lists with high precision at the top of the list. For this purpose, they propose to optimize a class of loss functions based on an

ordered weighted average (OWA) operators [Yager, 1988] as an alternative to the widely accepted pairwise loss which represents the mean over all the incorrectly ranked pairs [Herbrich et al., 1999, Joachims, 2002].

Let $\alpha = (\alpha_1, ..., \alpha_n)$ be a sequence of n non-negative numbers such that $\sum_{j=1}^{n} \alpha_j = 1$. The ordered weighted average (OWA) operator owa^{α} associated to $\alpha : R^n \rightarrow R$ is defined as:

$$\forall \mathbf{t} = (t_1, ..., t_n) \in R^n, owa^{\alpha}(\mathbf{t}) = \sum_{j=1}^{n} \alpha_j t_{\sigma(j)} \tag{4.4}$$

where $\sigma \in \Sigma_n$ is a permutation of $1..n$ such that $\forall j, t_{\sigma(j)} \geq t_{\sigma(j+1)}$. An OWA operator makes a weighted sum of t_js, such that the weight given to t_j depends on the position of t_j in the list $t_{\sigma(1)} \geq t_{\sigma(2)} \geq .. \geq t_{\sigma(n)}$. The max and mean are special cases of OWA operators: for the max, set $\alpha_1 = 1$ and $\alpha_j = 0$ for $j > 1$; for mean, set $\alpha_j = \frac{1}{n}$ for all j.

The family of proposed ranking functions is defined as follows. Let $\alpha^n \in [0, 1]$ be OWA weights complying with the condition above that $\sum_{j=1}^{n} \alpha_j = 1$ and, in addition, be non-increasing, $\alpha_1 \geq \alpha_2 \geq .. \geq \alpha_n \geq 0$. Given a set of examples $\mathcal{X} = \{\mathbf{x}_1..\mathbf{x}_N\}$ and a vector of labels $\mathbf{y}$, suppose that among these examples there are N_p positives $\mathbf{x}_1^+...\mathbf{x}_{N_p}^+$ and N_n negatives $\mathbf{x}_1^-...\mathbf{x}_{N_n}^-$.

Define the rank of an observation $\mathbf{x}^+$ of the positive class in terms of all

the negative examples $\mathbf{x}_1^- ... \mathbf{x}_{N_n}^-$ as

$$rank(\mathbf{x}^+) = \sum_{j=1...N_n} I(f(\mathbf{x}^+ \leq f(\mathbf{x}_j^-)) \tag{4.5}$$

where I is an indicator function. The rank of a positive example is the number of negatives that scored above it in the ranked list.

The ranking error of a real-valued function f on all the positive examples $\mathbf{x}_1^+ ... \mathbf{x}_{N_p}^+$ is then defined as

$$err(f, \mathcal{X}, \mathbf{y}) = \frac{1}{N_p} \sum_{i=1...N_p} \Phi_{N_n}(rank(\mathbf{x}_i^+)) \tag{4.6}$$

where the function $\Phi_n(k)$, $\forall k \in 0...n$, is given by

$$\Phi_n(k) = \sum_{j=1}^{k} \alpha_j. \tag{4.7}$$

The function $\Phi_n(k)$ is an OWA operator applied to a vector $\mathbf{z} \in \mathbf{R}^{N_n}$ with entries $z_j = I(f(\mathbf{x}^+ \leq f(\mathbf{x}_j^-))$, where $\mathbf{x}^+$ is some positive example, and $\mathbf{x}_j^-$, $\forall j \in N_n$ are negative examples.

The claim is that using this function, it is possible to define errors that focus more or less on the top of the list, since the penalty incurred by losing in rank is $\Phi_n(k+1) - \Phi_n(k) = \alpha_{k+1}$. However, we observe that the general tendency of this class of ranking functions is to put all the positive examples before all the negative examples in the ranking.

We hypothesize that the family of ranking functions defined above may not have the desired property for noisy data sets: it does not allow to focus on a

particular part of the training set and to ignore or penalize less severely the misclassified examples found far from the decision boundary. It will penalize the misclassified examples either progressively or equally as they become more distant from the decision boundary, depending on the choice of α.

4.1.3 ROC Optimization at a Single Point Mozer et al. [2002] propose several algorithmic alternatives to the problem that asks the classifier to achieve a user-specified correct acceptance rate (CA rate), correct rejection rate (CR rate) or both. This is equivalent to seeking to optimize a single point on the ROC curve, since P_{CA} is the same as the probability of true positives, P_{TP}, and the probability of false positives is given by $P_{FP} = 1 - P_{CR}$. That is, the task dictates finding a classifier with an ROC curve that contains a specified point (or a point optimally close to the specified one), possibly sacrificing the ranking performance on the whole set.

The first algorithm involves training a series of classifiers, attempting to achieve better CA and CR rates on each subsequent classifier (see figure 4.2A). The strategy is to focus on the examples that lie between the thresholds ρ_{CA} and ρ_{CR} corresponding to the specified values of CA and CR rates on the ROC curve produced by the nth classifier; this is termed the *emphasis* approach. The algorithm implementation defines a weighting for each example i based on

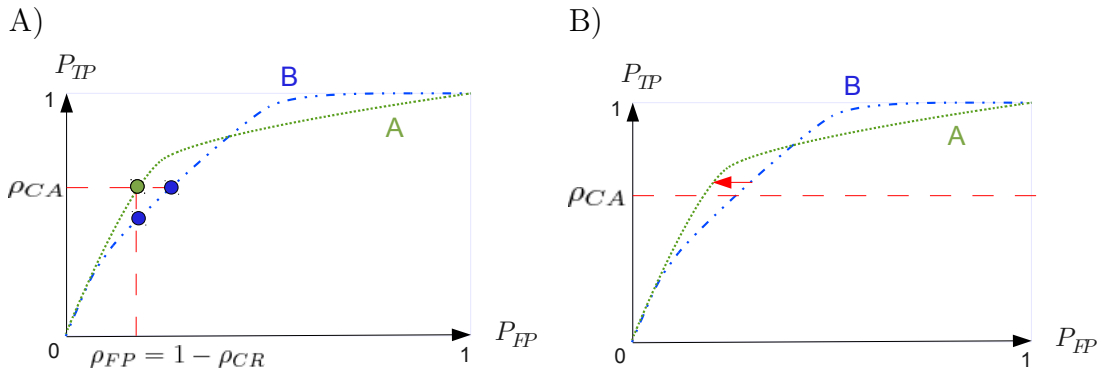


Figure 4.2: **Optimization of a Single Point on the ROC Curve** A) On ROC curve B, the target values of ρ_{CA} and ρ_{CR} correspond to two distinct points on the curve. The goal of the algorithms in [Mozer et al., 2002] is to find a classifier producing ROC curve A such that the target values ρ_{CA} and ρ_{CR} correspond to a single point on the curve. The first family of algorithms emphasizes the examples lying between the two points on the ROC curve B to arrive to a classifier with ROC curve A. (Alternatively it de-emphasizes irrelevant examples.) B) The second family of algorithms defines a constrained optimization problem which maximizes the correct rejection (CR) rate while maintaining the correct acceptance rate (CA) fixed to ρ_{CA}, for example.

the nth training classifier, λ_i^n. For the first classifier, $\lambda_i^1 = 1$. For subsequent classifiers, $\lambda_i^{n+1} = \lambda_i^n$ if example i is not in the region of emphasis, or $\lambda_i^{n+1} = \kappa_e \lambda_i^n$ otherwise, where $\kappa_e > 1$. In the *deemphasis* version of this algorithm, again a series of classifiers are trained, but when training classifier $n+1$, less weight is placed on the examples whose correct classification is unnecessary to achieve the target CA and CR rates.

A different approach proposes to maximize the correct rejection (CR) rate while maintaining the correct acceptance rate (CA) fixed to ρ_{CA} (see figure 4.2B). The CA and CR rates can be approximated with smooth differentiable functions:

$$CA(\mathbf{w}, t) = \frac{1}{|P|} \sum_{i \in P} \sigma_\beta(f(\mathbf{x}_i, \mathbf{w} - t)), \tag{4.8}$$

$$CR(\mathbf{w}, t) = \frac{1}{|N|} \sum_{i \in N} \sigma_\beta(t - f(\mathbf{x}_i, \mathbf{w})) \tag{4.9}$$

where P is the set of positives, N is the set of negatives and σ_β is a sigmoid function with scaling parameter β:

$$\sigma_\beta(y) = (1 + e^{-\beta y})^{-1}$$

which approximates the indicator variable $I(f(\mathbf{x}_i, \mathbf{w} - t) > 0)$.

Then the optimization problem is formulated such that CA is a constraint and CR is a figure of merit.

$$\max P_{CR}$$

subject to

$$P_{CA} \leq \rho_{CA}.$$

This constrained optimization problem is converted into an unconstrained problem by Lagrangian multipliers method.

Potentially, this approach can be easily extended to the maximization of true positives below a q-value threshold. The problem can be re-formulated such that CR is a constraint and CA is a figure of merit. Since by definition of q-value, $qP_{TP} = P_{FP}$, and $P_{CA} = P_{TP}$, $P_{CR} = 1 - P_{FP}$, the new optimization problem with constraints is

$$\max P_{TP}$$

subject to

$$P_{FP} \leq qP_{TP}.$$

4.1.4 Lambda Rank Burges [2005] and Burges et al. [2006] propose to define ranking cost function implicitly in terms of its partial derivatives with respect to each example. The motivation is that most of the ranking metrics present special challenges during optimization. First, most of them are not differentiable functions of their arguments and are replaced in practice with more tractable smooth objective functions which do not adequately approx-

imate the final goal. Second, they all depend not just on the output of the classifier for a single feature vector, but on the outputs of all feature vectors in the training set.

In this situation, given the ranking produced on the current iteration by the classifier being trained, it is much easier to specify how to change the ranking for this particular instance to improve the desired objective. The much more difficult task is to specify the objective function that would dictate how to change the order for every possible instance of ranking occurring during training to improve the desired objective on the next iteration.

Therefore, the approach is to define a loss function implicitly by way of specifying its partial derivatives with respect to each example based on the current ordering. In this approach, the gradients of the implicit cost function C with respect to the score of the example at rank position j are given by

$$\frac{\delta C}{\delta s_j} = -\lambda(s_1, l_1, ..., s_n, l_n)$$

where $s_1 ... s_n$ are the scores of the examples and $l_1 ... l_n$ are their labels. Defining an implicit cost function amounts to choosing suitable values of λ_j, which themselves are specified by rules that depend on the ranked order (and scores) of all the examples in the set.

The following restrictions hold on the choices of λ in order to ensure that first, the corresponding cost C exists and second, C is convex. The first

condition if fulfilled if

$$\frac{\delta\lambda_j}{\delta s_k} = \frac{\delta\lambda_k}{\delta s_j},$$

in other words, the matrix $J_{jk} = \delta\lambda_j/\delta s_k$ must be symmetric. The second condition is fulfilled if the Hessian of the cost function C (given by the same matrix J) is positive semidefinite.

This approach allows to focus only on a certain part of the ranked training set by steeply discounting the examples that occur low in the ranking and thereby to optimize only the area under the part of the ROC curve.

4.2 Algorithm for Direct Q-value Optimization: Q-ranker

The LambdaRank algorithm [Burges et al., 2006] defines an implicit loss function that allows to focus on ranking the pairs of positive-negative examples at the top of the sorted list during training. In this section, we define a family of explicit loss functions serving the same objective.

4.2.1 Ordered Weighted Average(OWA) Operator To define the family of loss functions, we make use of the Ordered Weighted Averaging (OWA) operator [Yager, 1988]. Given a set of non-negative weights α such that $\sum_{j=1}^{n} \alpha_j = 1$, OWA operator associated to $\alpha : R^n \rightarrow R$ is defined as:

$$\forall \mathbf{t} = (t_1, ..., t_n) \in R^n, owa^{\alpha}(\mathbf{t}) = \sum_{j=1}^{n} \alpha_j t_{\sigma(j)} \tag{4.10}$$

where σ is a permutation of $1..n$ such that $\forall j, t_{\sigma(j)} \geq t_{\sigma(j+1)}$. An OWA operator makes a weighted sum of t_js, such that the weight given to t_j depends on the position of t_j in the list $t_{\sigma(1)} \geq t_{\sigma(2)} \geq .. \geq t_{\sigma(n)}$.

In particular, we rely on the fact mentioned by Usunier et al. [2009] that OWA operator with non-increasing weights, $\alpha_1 \geq \alpha_2 \geq .. \geq \alpha_n \geq 0$, is a maximum over all permutations of t_js.

$$owa^{\alpha}(\mathbf{t}) = \max_{\sigma \in \Sigma_n} \sum_{j=1}^{n} \alpha_j t_{\sigma(j)} \tag{4.11}$$

This follow from the rearrangement inequality which states that for every choice of real numbers

$$x_1 \geq x_2 \geq ... \geq x_n$$

and

$$y_1 \geq y_2 \geq ... \geq y_n$$

for every permutation $\sigma \in \Sigma_n$, we have

$$x_{\sigma(1)}y_1 + x_{\sigma(2)}y_2 + ...x_{\sigma(n)}y_n \leq x_1y_1 + x_2y_2 + ...x_ny_n \tag{4.12}$$

4.2.2 Loss Function Definition Given a set of examples $\mathcal{X} = \{\mathbf{x}_1..\mathbf{x}_N\}$ and a vector of labels $\mathbf{y}$, let N_p and N_n be the number of positive and negative examples in the set, $\mathbf{x}_i^+$ and $\mathbf{x}_j^-$ represent ith positive and jth negative example, $\Pi(n)$ be the set of all possible permutations of integers $1, ..., n$, and let $\mathbf{c}$ and $\mathbf{d}$ be sets of non-increasing OWA weights for positive and negative examples.

Consider the quantity

$$\max_{\nu \in \Pi(N_n)} \sum_{j=1}^{N_n} d_j \max(0, 1 - (f(\mathbf{x}_i^+) - f(\mathbf{x}_{\nu(j)}^-))) \tag{4.13}$$

for any arbitrary positive example i. This formulation is equivalent to OWA operator applied to a vector $\mathbf{z} \in \mathbf{R}^{N_n}$ with entries $z_j = \max(0, 1 - (f(\mathbf{x}_i^+) - f(\mathbf{x}_{\nu(j)}^-)))$, where $\mathbf{x}^+$ is some positive example, and $\mathbf{x}_{\nu(j)}^-$, $\forall j \in N_n$ are negative examples.

There can be several permutations ν that maximize the above formula (for example if some d_js are zero), but the permutation that arranges $f(\mathbf{x}_{\nu(j)}^-)$ in non-increasing order is always one of them. Moreover, this solution is independent of the positive example i.

Now for all the positive examples,

$$E(f, \mathcal{X}, \mathbf{y}) = \min_{\sigma \in \Pi(N_p)} \sum_{i=1}^{N_p} c_i \max_{\nu \in \Pi(N_n)} \sum_{j=1}^{N_n} d_j \max(0, 1 - (f(\mathbf{x}_{\sigma(i)}^+) - f(\mathbf{x}_{\nu(j)}^-))) \tag{4.14}$$

The minimum over the permutations σ is reached for the permutation that orders all the $f(\mathbf{x}_{\sigma(i)}^+)$ in non-increasing order. This solution does not depend on either j or ν.

Therefore, suppose that permutations σ and ν are chosen such that, given the current parameters $\mathbf{w}$ of the function f, they arrange all $f(\mathbf{x}_{\sigma(i)}^+)$ and all

$f(\mathbf{x}^-_{\nu(j)})$ in non-increasing order. Then the loss function is given by

$$E(f, \mathcal{X}, \mathbf{y}) = \min[\sum_{i=1}^{N_p} \sum_{j=1}^{N_n} c_i d_j \max(0, 1 - (f(\mathbf{x}^+_{\sigma(i)}) - f(\mathbf{x}^-_{\nu(j)})))] \tag{4.15}$$

Now by choosing the non-increasing OWA weights $\mathbf{c}$ and $\mathbf{d}$, we can select arbitrary subsets of the best-ranking positives and negatives for training and minimize E.

The function $\max(a, b)$ is differentiable everywhere, except when $a = b$. However, stochastic gradient descent optimization will converge. Let $L(\mathbf{x}^+, \mathbf{x}^-, \mathbf{w})$ be

$$L(\mathbf{x}^+, \mathbf{x}^-, \mathbf{w}) = \max(0, 1 - (f(\mathbf{x}^+) - f(\mathbf{x}^-))) \tag{4.16}$$

where $\mathbf{w}$ are the parameters of f. Then, define the update of the parameters $\mathbf{w}_k$ on iteration k as follows

$$\mathbf{w}_k = \mathbf{w}_k - \gamma H(\mathbf{x}^+_i, \mathbf{x}^-_j, \mathbf{w}) \tag{4.17}$$

where γ is the learning rate and

$$H(\mathbf{x}^+_i, \mathbf{x}^-_j, \mathbf{w}) = c_i d_j \nabla_{\mathbf{w}} L(\mathbf{x}^+_i, \mathbf{x}^-_j, \mathbf{w}) \tag{4.18}$$

if L is differentiable at $(\mathbf{x}^+_i, \mathbf{x}^-_j)$ and

$$H(\mathbf{x}^+_i, \mathbf{x}^-_j, \mathbf{w}) = 0 \tag{4.19}$$

otherwise [Bottou, 1998].

4.2.3 Training Heuristics We introduce the following heuristics in order to optimize this loss function.

1. Since the loss function associates weights with the position of positives and negatives in a ranked list, it requires two sorting operations at each step of the stochastic gradient descent. Instead, we sort the data set after every epoch consisting of $N_p + N_n$ iterations.

2. For a given threshold t with t_p and t_n positive and negative examples above it in the ranked list, we set $c_i = 1/t_p$ for $i \leq t_p$ and $c_i = 0$ for $i > t_p$; d_j are similarly defined in terms of t_n. We then try several different thresholds t and choose the threshold giving the best result for the q-value of interest.

3. Once the threshold t and $\mathbf{c}$, $\mathbf{d}$ are chosen, the loss function dictates training only with examples that fall within the threshold. Instead, we start by running the optimization algorithm with sigmoid loss function described in section 3.3.2. We use the resulting discriminant function parameters $\mathbf{w}$ as initialization each time a new threshold is chosen, as further described in section 4.2.6.

4.2.4 Weight Decay In this work, we use the standard weight decay procedure, which optimizes the error function:

$$E' = E + \mu \frac{1}{2} \sum_i w_i^2$$

where $\mathbf{w}$ are all the weights of the discriminant function $f(\mathbf{x})$ that we are attempting to learn, μ is a weight decay parameter, and E is the original error function given by the equation 4.15. Before final training of the network, we perform a 3-fold cross-validation procedure to choose the learning rate γ and the parameter μ.

4.2.5 Use of Non-Linear Models Having established in the previous chapter that direct classification using a linear model performs as well as Percolator, we next consider a nonlinear family of functions implemented by two-layer neural networks

$$f(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}), \tag{4.20}$$

where $h_k(\mathbf{x})$ is defined as $\tanh((\mathbf{w}^k)^\top \mathbf{x} + b_k)$, and $\mathbf{w}^k$ and b_k index the weight vector and threshold for the kth hidden unit. Because we are solving a ranking problem in the nonlinear case, we do not have a final bias output.

We can choose the capacity of our nonlinear family of discriminant functions by increasing or decreasing the number of hidden units of the neural network. Based on preliminary experiments with the yeast training data set,

we chose the first layer to have five linear hidden units.

4.2.6 Algorithm The proposed algorithm is as follows (Algorithm 1). We first find a general discriminant function $f(\mathbf{x})$ using the classification algorithm with sigmoid loss described in section 3.3.2. We then specify a q value to be optimized and focus sequentially on several intervals in the ranked data set chosen in the vicinity of the specified q value. In the course of training, we record the best result for the specified q value after each epoch.

The selection of intervals is heuristic and in our case involves defining them based on the results of the initial sigmoid loss training. We choose a set $\mathcal{T}$ of q value thresholds 0 to 0.1 with a step size of 0.01; they are different from the target q-values to be optimized and serve to define intervals in the ranked data set for the algorithm to explore. The interval t to focus on at each cycle is set to equal twice the number of peptide-spectrum matches up to the threshold point.

The intervals t_p and t_n for positive and negative examples respectively are determined based on the current interval t and are set to include all the positives and negatives such that $f(\mathbf{x}^+)$ ranks above t, i.e. $rank(f(\mathbf{x}^+)) < t$, and $f(\mathbf{x}^-)$ ranks above t, i.e $rank(f(\mathbf{x}^-)) < t$).

Q-ranker can be extended trivially to search for optimal solutions to several

Algorithm 1 The input variables are the training set $\mathcal{X} = \{\mathbf{x}_1 \dots \mathbf{x}_N\}$ of peptide-spectrum match feature vectors, the corresponding binary labels $\mathbf{y}$, indicating which examples are targets and which are decoys, the set $\mathcal{Q}$ of specified target q values, the set $\mathcal{T}$ of thresholds (or intervals on which the algorithms focuses during each training cycle) and the number n of training iterations for each cycle.

procedure Q-RANKER($\mathcal{X}, \mathbf{y}, \mathcal{Q}, \mathcal{T}, n$)
    $\mathbf{w} \leftarrow$ initialize using sigmoid loss classification (section 3.3.2).
    for $q_{targ} \in \mathcal{Q}$ **do**
        for $t \in \mathcal{T}$ **do**
            $t_p \leftarrow |\{\mathbf{x} \in \mathcal{X}^+ | rank(f(\mathbf{x})) < t\}|$
            $t_n \leftarrow |\{\mathbf{x} \in \mathcal{X}^- | rank(f(\mathbf{x})) < t\}|$
            for $i \leftarrow 1 \dots n$ **do**
                $\mathbf{x}^+ \leftarrow$ chooseRandom($\mathcal{X}^+, t_p$) ▷ Random pair of examples.
                $\mathbf{x}^- \leftarrow$ chooseRandom($\mathcal{X}^-, t_n$)
                $\mathbf{w} \leftarrow$ gradientStep($\mathbf{w}, f(\mathbf{x}^+), f(\mathbf{x}^-)$) ▷ Update the weights.
            end for
        end for
        Record best result on q_{targ}
    end for
    return (best $\mathbf{w}$)
end procedure

q values at once by recording the best network for each of the specified q values after each epoch. In all the experimental runs presented below, the set of target q-values $\mathcal{Q}$ also serves as the set the attempted thresholds $\mathcal{T}$, however, the two sets can be different.

Q-ranker generalizes the ranking SVM formulation in two ways: (i) this formulation is nonlinear (but does not use kernels); and (ii) if the interval t is very large, then the algorithms are equivalent, but as the interval t is reduced, our algorithm begins to focus on given q values.

4.2.7 Comparison of Algorithms Across Multiple Data Sets

Data sets We used four data sets previously described in Käll et al. [2007] to test our algorithms. The first is a yeast data set containing 69,705 target peptide-spectrum matches and twice that number of decoy peptide-spectrum matches. These data were acquired from a tryptic digest of an unfractionated yeast lysate and analyzed using a four-hour reverse phase separation. Throughout this work, peptides were assigned to spectra using SEQUEST database search with no enzyme specificity and with no amino acid modifications enabled. The next two data sets were derived from the same yeast lysate, but treated by different proteolytic enzymes: elastase and chymotrypsin. These data sets contain 57,860 and 60,217 target peptide-spectrum respectively and

twice that number of decoy peptide-spectrum matches. The final data set was derived from a *C. elegans* lysate proteolytically digested with trypsin and processed analogously to the yeast data sets.

Results To compare the performance of Q-ranker, Percolator and PeptideProphet [Keller et al., 2002], we provided the same set of target and decoy peptide-spectrum matches to each algorithm. For Percolator and Q-ranker, we use 50% of the peptide-spectrum matches for training and 50% for testing. PeptideProphet software does not provide the ability to learn model parameters on one set of data and apply the learned model to the second; therefore, PeptideProphet results are generated by applying the algorithm to the entire data set. This difference gives an advantage to PeptideProphet because that algorithm learns its model from twice as much data and is not penalized for over-fitting.

We report results using either 17 or 37 features, as described in Table 3.1, for both Percolator and Q-Ranker. Figures 4.3, 4.4 show the results of this experiment, conducted using the four data sets described in Section 4.2.7. Across the four data sets, Q-ranker consistently outperforms PeptideProphet across all q value thresholds. At q values of 0.05 or 0.10 and using 17 features, Q-ranker yields more accepted target peptide-spectrum matches than either

Percolator or PeptideProphet, whereas Percolator performs slightly better for $q < 0.01$.

Theoretically, a nonlinear network could yield a larger benefit than a linear model when the input feature space is increased, as long as the model does not overfit. We therefore experimented with extending the peptide-spectrum match feature vectors, adding 20 new features corresponding to the counts of amino acids in the peptide. The results of running Q-ranker with these extended vectors are shown in Figures 4.3 and 4.4, labeled "Q-ranker 37". Increasing the number of features gives a larger boost to the performance of the nonlinear version of Q-ranker. After this extension, Q-ranker identifies more spectra than either of the other algorithms, even at $q < 0.01$.

Therefore, we observe relatively large benefit provided by amino acid composition features. We hypothesize that this information allows the classifier to learn to expect certain characteristics of a spectrum. For example, the presence of a proline implies a pair of high-intensity peaks corresponding to the cleavage N-terminal to the proline; the presence of many basic residues leads to more +2 ions, and the presence of many hydrophobic residues leads to more singly charged +1 ions [Klammer et al., 2008]. However, previous experiments with Percolator using amino acid composition features did not yield significant performance improvements. The difference, in the current setting, is that we

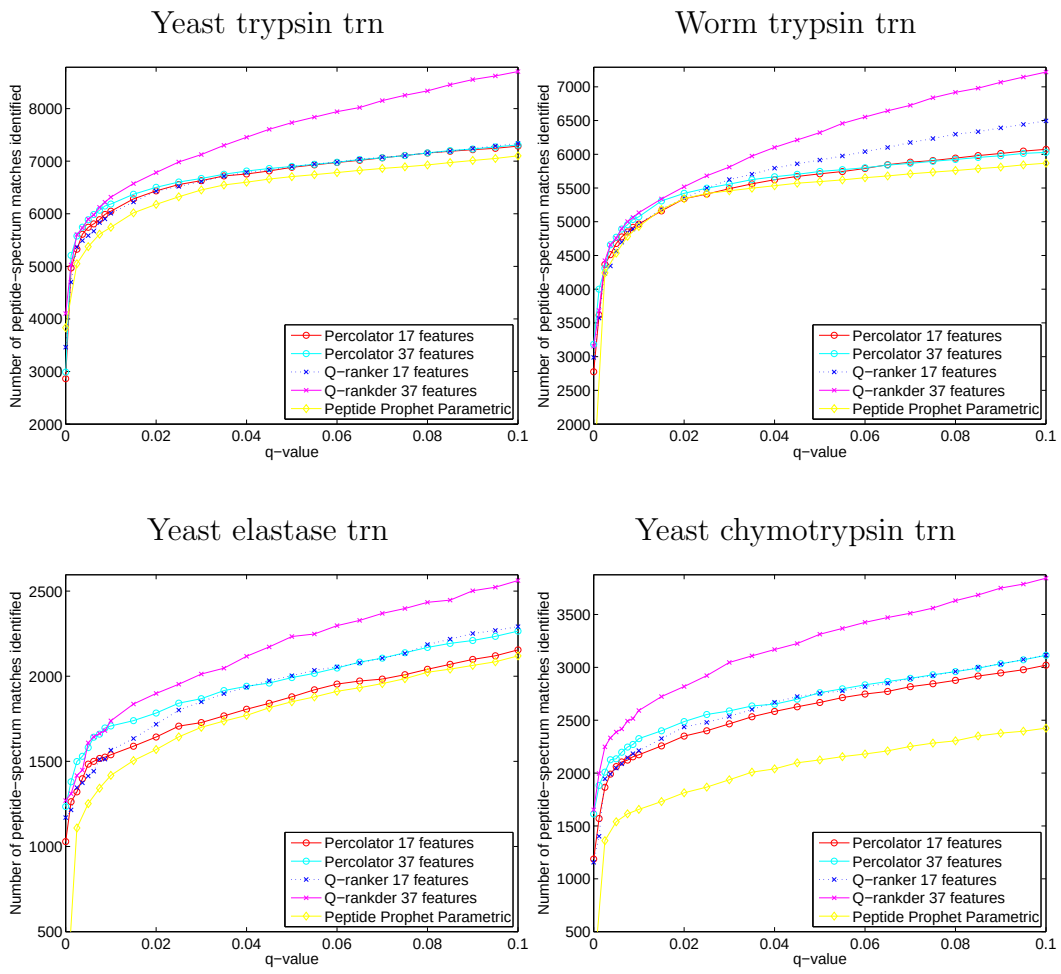


Figure 4.3: **Comparison of PeptideProphet, Percolator and Q-ranker on Training Set.** Each panel plots the number of accepted target peptide-spectrum matches as a function of q value on the training (trn) set. The series correspond to the three different algorithms, including two variants of Q-ranker that use 17 features and 37 features.

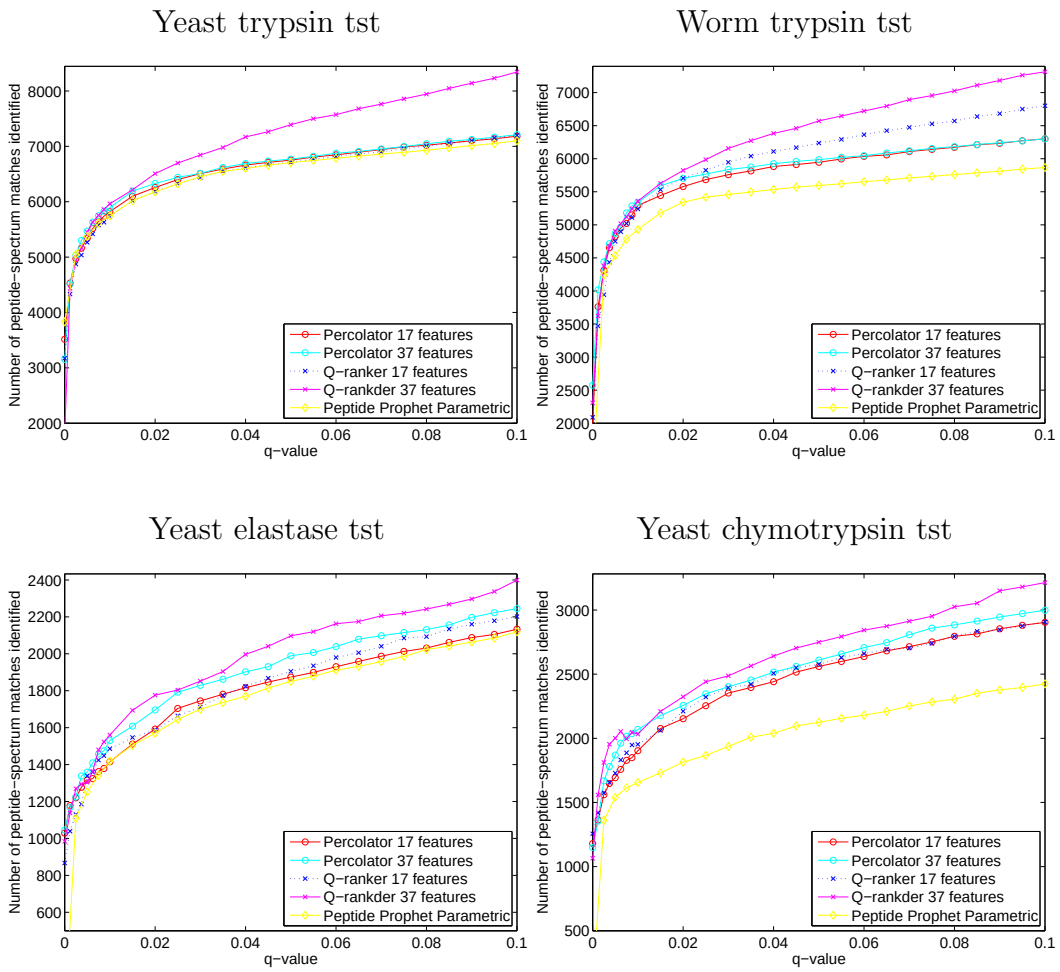


Figure 4.4: **Comparison of PeptideProphet, Percolator and Q-ranker on Testing Set.** Each panel plots the number of accepted target peptide-spectrum matches as a function of q value on the testing (tst) set. The series correspond to the three different algorithms, including two variants of Q-ranker that use 17 features and 37 features.

(A) Sigmoid loss classification (B) Q-ranker

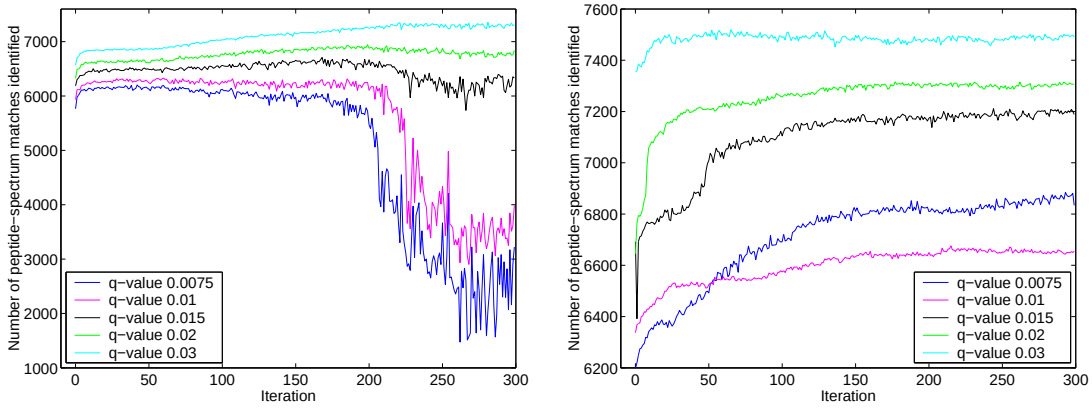


Figure 4.5: **Comparison of Training Optimization Methods (Iteration vs. Error Rate).** The Q-ranker optimization starts from the best result of sigmoid loss optimization achieved during the course of training and continues for a further 300 iterations. These results are on the training set. Note that for each q value choice, Q-ranker improves the training error over the best result from the classification algorithm.

have used a more complex, nonlinear model. In general, a complex model has more opportunity to improve over a simpler model if the feature space is rich. Thus, although a simple linear model such as the one in Percolator cannot fully exploit the richer, 37-dimensional feature space, the nonlinear model can.

Algorithm Indeed Optimizes for the q value of Interest Compared to the classification approach with sigmoid loss described in the previous chapter,

Q-ranker yields more consistent training behavior when observed for any given q value. To illustrate this phenomenon, we fix the interval t for the Q-ranker algorithm and measure how it performs on various q-value thresholds in the course of training. Figure 4.5A shows how the results for different specified q values change during the course of training with the sigmoid loss classification model. The number of peptide-spectrum matches over lower q value thresholds (for example, 0.0075, 0.01) reach their peak early during training and then become suboptimal, while the best results for higher q value thresholds take longer to achieve. This means that during the course of training, different q value thresholds are being optimized depending on the number of iterations. In contrast, as shown in Figure 4.5B, the Q-ranker algorithm learns the best decision boundary for a specified q value threshold and does not substantially diverge from the best result during further training. This behavior indicates that the algorithm in fact optimizes the desired quantity.

We further investigated the behavior of Q-ranker by measuring the performance of networks trained for a specified q value threshold on other q values. We focused on specified q values 0.01, 0.05 and 0.1. Table 4.1 shows that, when all 37 features are employed, a network trained for a specified q value is consistently better or equal to the performance on this q value, compared with networks trained for other specified q values.

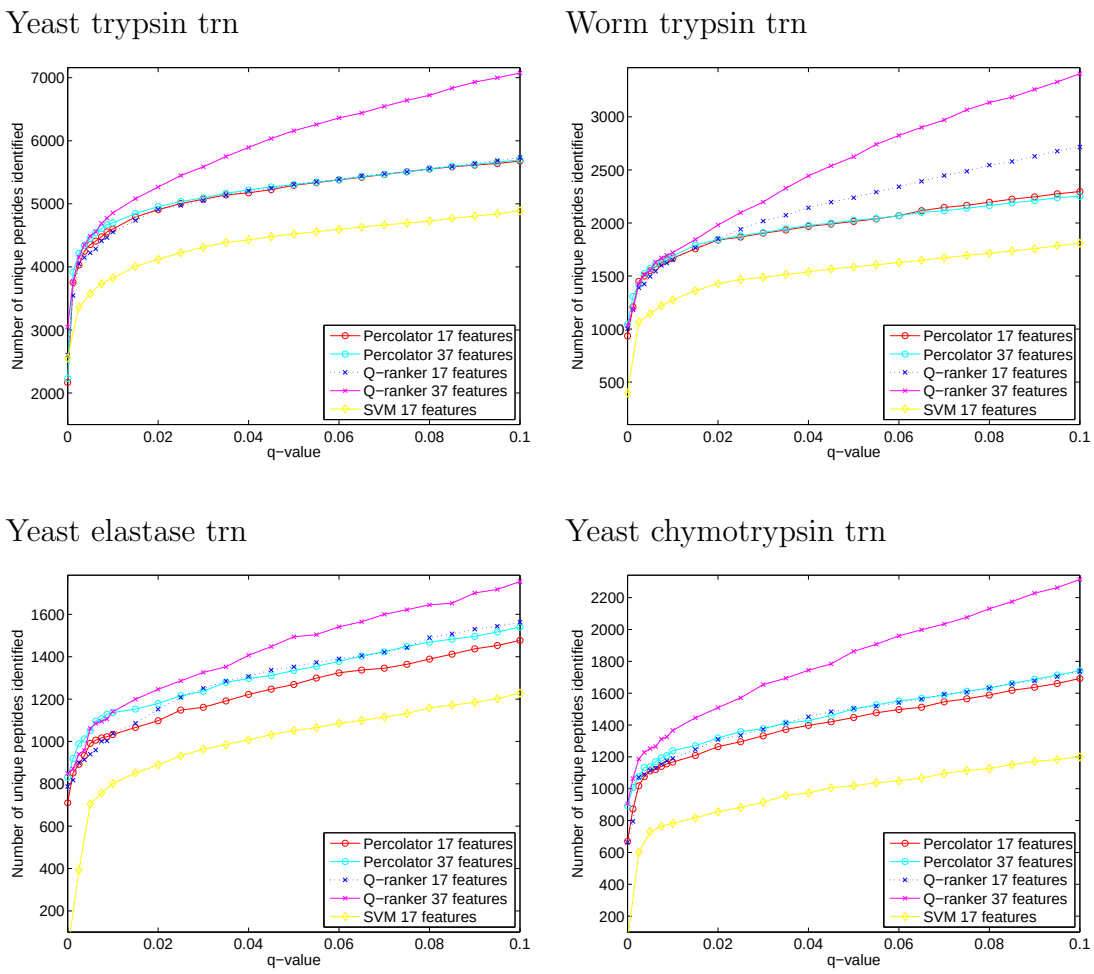


Figure 4.6: **Comparison of PeptideProphet, Percolator and Q-ranker on Training Set in Terms of the Number Unique Peptides Identified Over the Range of Q-values.** Each panel plots the number of unique real database peptides as a function of q value on the training (trn) set. The series correspond to the three different algorithms, including two variants of Q-ranker that use 17 features and 37 features.

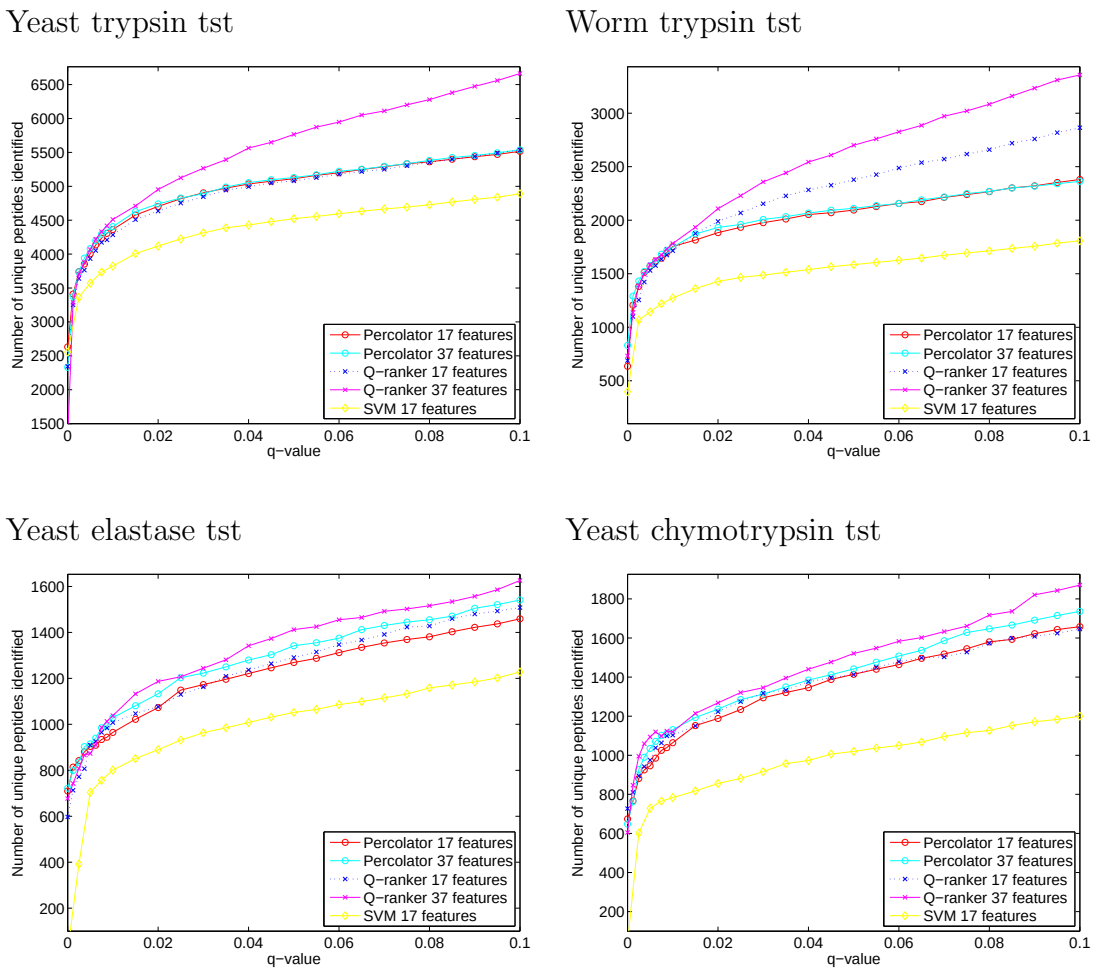


Figure 4.7: **Comparison of PeptideProphet, Percolator and Q-ranker on Testing Set in Terms of the Number Unique Peptides Identified Over the Range of Q-values.** Each panel plots the number of unique real database peptides as a function of q value on the testing (tst) sets. The series correspond to the three different algorithms, including two variants of Q-ranker that use 17 features and 37 features.

Specified	Yeast trypsin			Worm trypsin			Yeast elastase			Yeast chymotrypsin		
	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10
0.01	**6072**	7453	8360	**5238**	6412	7098	**1615**	2054	2395	**2312**	2843	3199
0.05	6032	**7501**	8426	**5238**	**6419**	7047	**1615**	**2140**	**2561**	2302	**2844**	3198
0.10	6030	7500	**8430**	5213	6418	**7128**	**1615**	**2140**	**2561**	2300	2830	**3214**

Table 4.1: **Q-ranker Successfully Optimizes the Specified Q value.** Each entry in the table lists the number of accepted peptide-spectrum matches at a given q value threshold (column) obtained by Q-ranker with 37 features when optimizing a specified q value (row). Entries in boldface indicate the maximum value within each column. Note that, for each data set, all diagonal entries are in boldface.

So far we have focused on maximizing the number of spectra that had a correct peptide assigned to them (i.e., the number of accepted peptide-spectrum matches). It is conceivable that a given algorithm might be biased in the types of peptides it can identify. In this case, the relative performance of two peptide identifications could depend on whether we count the number of accepted peptide-spectrum matches or the number of distinct peptides that are identified from a set of spectra. Figures 4.6 and 4.7 demonstrate that this bias is not occurring in our results: the relative performance of the algorithms that we considered does not change significantly when we count the number of distinct peptides identified.

4.3 Conclusion

Generally speaking, the goal of many tasks associated with the *semi-supervised* problems with labeled examples of a single class and unlabeled data is to determine an accurate ranking on some specific part of the data set. For example, in information retrieval tasks we prefer ranked lists with high precision at the top of the list. Similarly, the desired outcome of shotgun proteomics experiments is to identify as many peptide-spectrum matches as possible at a given q-value threshold. However, all the algorithms reviewed in section 3.1, designed mostly for document retrieval and web search applications, as well as existing algorithms designed for proteomics settings (reviewed in section 3.2) attempt to solve a classification rather than a ranking problem.

In this chapter, we directly address the problem of ranking in the vicinity of a q-value threshold of interest. For this purpose, we define a family of mostly differentiable ranking loss functions based on Ordered Weighted Average (OWA) operator. We use the observation made in the previous chapter that a loss function applied to data sets with high amounts of noise should not severely penalize examples that are far from the decision boundary. We subsequently define a loss function that permits selective ranking of only some of the positive-negative pairs at the top of the sorted list, rather than satisfying all the pairwise constraints in a given data set. The experiments presented

in this chapter show that reformulating the problem as a ranking task, rather than as a classification task, and optimizing for a q value threshold of interest leads to better performance.

Chapter 5

Protein Identification

Typically, the problem of identifying proteins from a collection of tandem mass spectra involves assigning spectra to peptides and then inferring the protein set from the resulting collection of peptide-spectrum matches (PSMs).

Many algorithms designed for solving the second part, inferring the set of proteins from the peptide-spectrum matches, break up the process into two stages [Nesvizhskii et al., 2003, Alves et al., 2007, Zhang et al., 2007, Li et al., 2008]. First, these methods attempt to assess the quality of the peptide-spectrum matches, assigning to each match a score that measures its chance of being correct. This step sometimes includes filtering the peptide-spectrum matches to retain only those believed to be correct up to a given error rate. Next, these methods infer the protein set from the peptides obtained from the

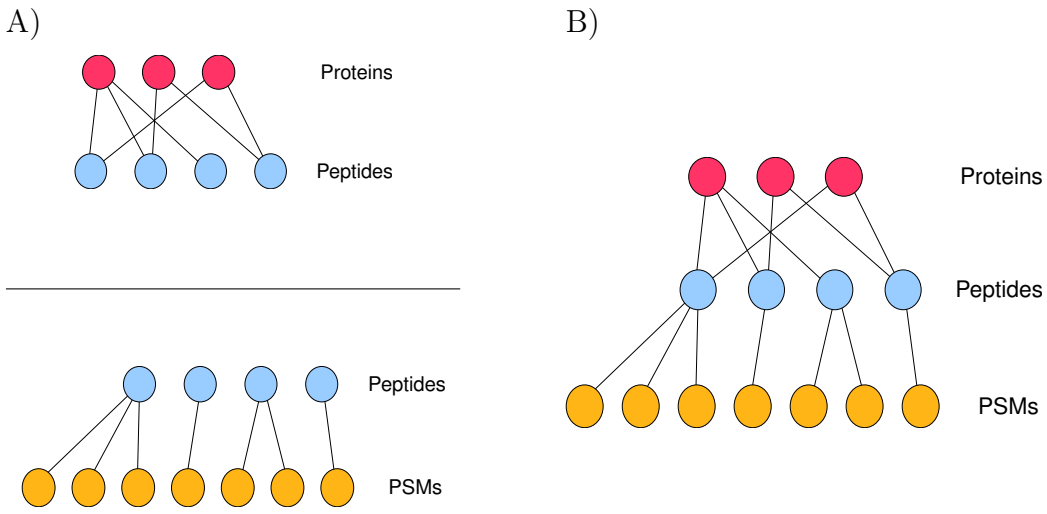


Figure 5.1: **Difference Between State-of-the-art Algorithms (A), and the Algorithm Proposed in this chapter (B).** Most algorithms infer proteins from peptide-spectrum matches in two separate stages: PSM verification and then protein reconstruction. This chapter proposes to accomplish both tasks simultaneously by optimization at the protein level.

previous step (see figure 5.1A).

When the goal of a shotgun proteomics experiment is to identify proteins, rather than peptides, we propose to skip the intermediate peptide identification step entirely, moving directly from a collection of peptide-spectrum matches to a predicted set of proteins (see figure 5.1B). We use a target-decoy search strategy, searching each spectrum against a database of real (target) proteins

and shuffled (decoy) proteins. We then train a supervised learning algorithm to induce a ranking on the combined set of target and decoy proteins, learning the parameters of the model so that the top of the ranked list is enriched with target proteins.

The advantage of this approach is that we can directly optimize the desired quantity—for example, the number of proteins identified at a specified error rate. The alternative approach first solves a related problem, fitting a statistical model to the peptide-spectrum match and then to protein data sets. This latter approach may be more challenging and typically requires making assumptions about peptide-spectrum match and protein distributions that are not known and that may vary substantially between data sets. In addition, our algorithm does not filter any peptide-spectrum matches at any stage of the analysis, with the motivation that even low scoring PSMs can carry information about a protein's presence when considered in the context of other PSMs belonging to this protein.

5.1 Previous Work

Here we describe the algorithms that are used as benchmarks for the results presented in this chapter. These algorithms take as input peptide-spectrum matches and the scores, derived from algorithms such as PeptideProphet (see

description in section 3.2), indicating the correctness of these matches. The goal of these algorithms is to resolve the ambiguities of shared peptides and to produce a minimal protein list accounting for the observed spectra. The approach to this problem (i.e. parsimony rules) vary in stringency among algorithms. ProteinProphet [Nesvizhskii et al., 2003] assigns partial weight of shared peptides to proteins containing them. IDPicker [Zhang et al., 2007] uses more stringent parsimony rules stating that only a single protein will get the full weight of each shared peptide.

5.1.1 ProteinProphet ProteinProphet [Nesvizhskii et al., 2003] computes probabilities that each protein was present in the sample based on the peptides assigned to the spectra using each peptide as independent evidence. The input to the algorithm is a list of peptides along with probabilities that they were correctly identified; these probabilities are produced by the companion program PeptideProphet.

The probability P_r that a protein r was present is given in terms of the set of identified peptides $e_r = e_1...e_{N_r}$ contained in this protein's sequence and the probabilities of these peptides $p(e_1)...p(e_{N_e})$. The algorithm computes the probability that at least one peptide assignment is correct:

$$P_r = 1 - \prod_i (1 - p(e_i))$$

As written, this formula ignores the ambiguities arising from proteins sharing peptides, since it assigns a full weight of each peptide to every protein containing it. Therefore, for every peptide e_i that belongs to a set of K proteins ($R = r_1...r_K$), the algorithm introduces a set of weights $\mathbf{w^i}, 0 \leq w_k^i \leq 1$ that indicate what portion of the peptide will contribute to the score of the kth protein in this set:

$$w_k^i = \frac{P_{r_k}}{\sum_j^K P_{r_j}}$$

Based on this formulation, the protein with the highest probability P_{r_k} among the K other members of the set will get the biggest contribution of the peptide e_i to its score.

Then the probability of a protein P_r, taking into account that each of its peptides could belong to other proteins is given by

$$P_r = 1 - \prod_i (1 - w_r^i p(e_i))$$

The model learns the peptide weights iteratively using an EM-like algorithm: 1) peptides are apportioned to proteins using current weights and protein probabilities are determined; 2) new sets of weights are re-calculated.

5.1.2 IDPicker While ProteinProphet assigns peptides partially to the proteins containing them, IDPicker uses more conservative strategy of assign-

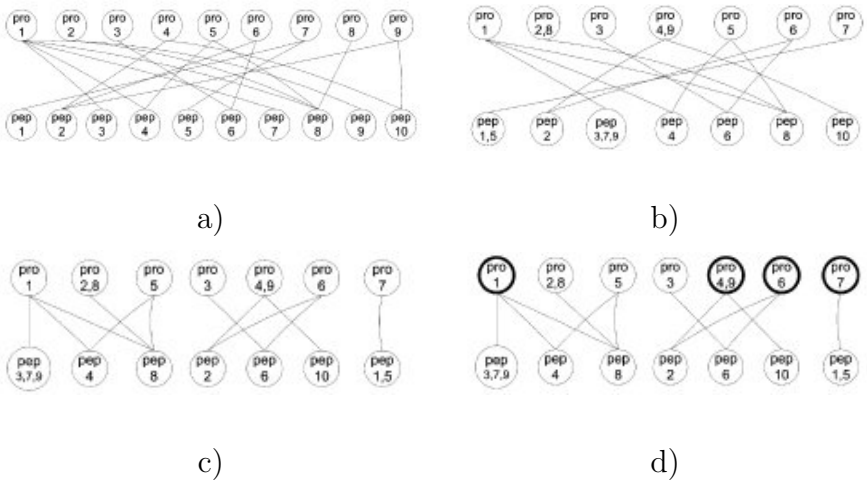


Figure 5.2: **Four stages of the IDPicker Algorithm**. Zhang et al. [2007]: initialize, collapse, separate, reduce.

ing the full peptide to a single protein containing it. Therefore, it produces the minimal list of proteins which can be accounted for by the peptides identified by selection of the higher quality peptide-spectrum matches.

IDPicker models the protein-peptide relationship as a bipartite graph and uses graph algorithms to identify protein clusters with shared peptides and to derive the minimal list of proteins. The algorithm proceeds in four stages illustrated on figure 5.2: initialize, collapse, separate and reduce.

1. The peptide identification data is represented as a bipartite graph that includes 2 sets of vertices: protein vertices and peptide vertices. There is an edge between protein and peptide vertices if the peptide sequence could be matched to the protein sequence.

2. Those proteins that are connected to exactly the same peptide vertices are indiscernible from each other and are collapsed to form meta-proteins.

3. The next step finds independent proteins which do not have any peptides in common. This is accomplished by depth-first search to find connected components. The result is a set of independent subgraphs.

4. To create a list of meta-proteins that are necessary to explain the observed spectra, the algorithms generates a minimal set of meta-proteins for each cluster using greedy set-cover algorithm. At each stage, it iteratively chooses a meta-protein vertex that connects to the largest number of peptide vertices.

5.2 The Algorithm: Protein Q-ranker

In this section, we address directly the problem of identifying a set of proteins that can explain the observed spectra by defining a model of proteins based on their peptide-spectrum matches. The main features of the model are the following. The model consists of three score functions, defined with respect to peptide-spectrum matches (PSMs), peptides and proteins (see Figure 5.3). The peptide-spectrum match score function is a nonlinear function of the 17 input features; the function is defined by a two-layer neural network with three

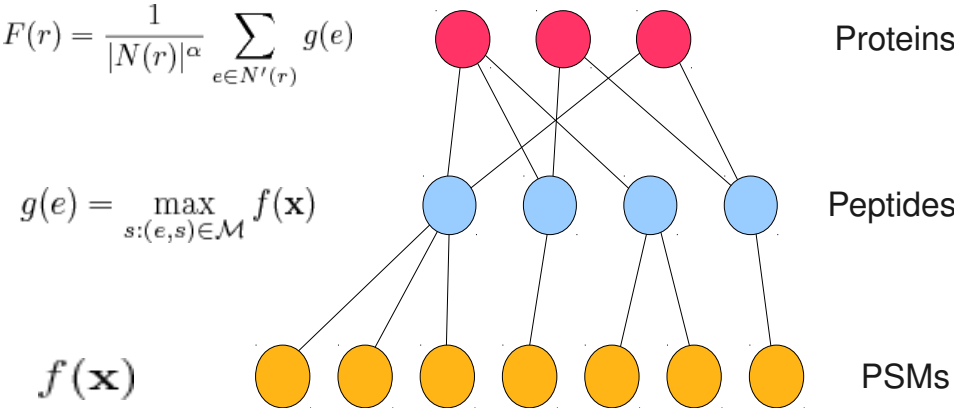


Figure 5.3: **Protein Identification and Scoring Functions.** The figure illustrates the association between different levels of protein identification task and the scoring functions assigned to them by the algorithm proposed in this chapter.

hidden units. The score function of a peptide is the maximum score of the matches associated with it, and the score function of a protein is a normalized sum of its peptide scores. The normalization factor $|N(r)|$ is defined as the number of peptides that occur in protein r, assuming enzymatic cleavages. The method uses the theoretical number of peptides, rather than the number of observed peptides, because the theoretical peptide number implicitly supplies an additional piece of information: how many peptides appear in the protein but have not been matched by any spectrum. This information allows to penalize longer proteins, which are more likely to receive random matches during the database search procedure.

We then learn a protein score function that performs well on the target-decoy training task using a simple iterative update procedure (Algorithm 2). During training, the weights of the neural network that define the peptide-spectrum match score function are optimized, because the PSM score is part of the protein score calculation. These weights are the only adjustable parameters of the learning task.

We will now discuss the model in more detail.

5.2.1 Input We are given a set of observed spectra $\mathcal{S} = \{s_1, \ldots, s_{N_\mathcal{S}}\}$ and a database $\mathcal{D}$ of target and decoy proteins (mixture) against which we perform a database search. The search produces a set of peptide-spectrum matches (PSMs). Denoting the set of peptides as $\mathcal{E} = e_1, \ldots, e_{N_\mathcal{E}}$, the peptide-spectrum matches $\mathcal{M}$ are written as pairs $(e_j, s_k) \in \mathcal{M}$, each representing a match of peptide j to spectrum k. Note that, in general, we may opt to retain the single best-scoring peptide for each spectrum, or a small constant number of top-ranked matches per spectrum. Each of the identified peptides e_k belongs to one or more proteins, leading to a set of proteins $\mathcal{R} = r_1, \ldots, r_{N_\mathcal{R}}$ that cover the set of peptides. $\mathcal{R}$ includes every protein in $\mathcal{D}$ that has at least one identified peptide (i.e. the maximal set of proteins that can explain the observed spectra).

For our algorithm, we define a feature representation $\mathbf{x} \in \mathbb{R}^d$ for every peptide-spectrum match pair (e, s). Our particular choice for this feature representation, which is described in table 5.1, contains a variety of scores of the quality of the peptide-spectrum match, as well as features that capture properties of the spectrum and properties of the peptide.

5.2.2 PSM Scoring Function We now define the score of a peptide-spectrum match to be a parametrized function of its feature vector $\mathbf{x}$. We choose a family of nonlinear functions given by two-layer neural networks:

$$f(\mathbf{x}) = \sum_{i=1}^{\mathcal{HU}} w_i^O h_i(\mathbf{x}) + b, \tag{5.1}$$

where $w^O \in \mathbb{R}^{\mathcal{HU}}$ are the output layer weights for the $\mathcal{HU}$ hidden units, and $h_k(\mathbf{x})$ is the k^{th} hidden unit, defined as:

$$h_k(\mathbf{x}) = \tanh((\mathbf{w}_k^H)^\top \mathbf{x} + b_k), \tag{5.2}$$

where $\mathbf{w}_k^H \in \mathbb{R}^d$ and $b_k \in \mathbb{R}$ are the weight vector and threshold for the k^{th} hidden unit. The number of hidden units $\mathcal{HU}$ is a hyperparameter that can be chosen by cross-validation. Throughout this work, we use a fixed value of 3 hidden units. In preliminary experiments, we observed that 3 or 4 hidden units provided approximately the same performance, whereas using 5 hidden units led to evidence of over-fitting.

5.2.3 Peptide Scoring Function A single peptide can have several spectra matching to it, since during the database search, several spectra can choose the same peptide as their best match. For each distinct peptide we would like to rank the likelihood that it has been matched correctly. We define the score of a peptide as the maximum score assigned to any of its peptide-spectrum matches:

$$g(e_j) = \max_{s_k:(e_j,s_k)\in\mathcal{M}} f(\mathbf{x}_{jk}) \tag{5.3}$$

where $(e_j, s_k) \in \mathcal{M}$ is the set of peptide-spectrum matches assigned to peptide e_j and $\mathbf{x}_{jk}$ is the feature representation of the match between peptide e_j and spectrum s_k. We take the max over the matches for each peptide because of the argument presented in [Nesvizhskii et al., 2003] that many spectra matching the same peptide are not an indication of the correctness of the identification.

5.2.4 Protein Scoring Function Finally, the score of a protein is defined in terms of the scores of the peptides in that protein as follows:

$$F(r) = \frac{1}{|N(r)|^\alpha} \sum_{e\in N'(r)} g(e) \tag{5.4}$$

where $N(r)$ is the set of predicted peptides in protein r, $N'(r)$ is the set of peptides in the protein r that were observed during the MS/MS experiment, and α is a hyperparameter of the model. The set $N(r)$ is created by virtually digesting the protein database $\mathcal{D}$ with the protease used to digest the protein

mixture for the mass spectrometry experiment.

The motivation for this protein score function is the following. The sum of the scores of all the peptides identified during the database search is used to estimate the accuracy of the protein identification. Dividing by a function of the predicted number of peptides is designed to correct for the number of the peptides not identified during the database search. Setting $\alpha = 1$ penalizes the failure to observe the predicted peptides linearly, whereas setting $\alpha < 1$ punishes larger sets of peptides to a lesser degree - for example, this can be used if not all peptides in a protein are observable. In our results we use the fixed value $\alpha = 0.3$, after selecting it in validation experiments.

5.3 Training the Model

5.3.1 Target-Decoy Protein Identification Problem We propose to use a target-decoy training strategy for proteins. Previously, target-decoy learning has been employed successfully to discriminate between correct and incorrect peptide-spectrum matches produced by a database search algorithm [Käll et al., 2007, Spivak et al., 2009b], but has never been employed at the protein level.

A database is composed of real (target) and shuffled (decoy) proteins which are virtually digested into peptides for the purpose of matching spectra to

them. We search each spectrum against a database of target peptides and decoy peptides. The target proteins that contain the matched target peptides are labeled as positive examples for training, decoy proteins containing the matched decoy peptides are labeled as negative examples.

The target-decoy strategy allows to re-formulate the original problem in a more tractable way. Originally, the goal is to determine the correct proteins among all the proteins identified by the matches to real sequences in the database. Instead, we solve a proxy problem of discriminating between target and decoy protein matches. Given a set of proteins $r_1...r_n$ and corresponding labels $y_1...y_n$, the goal is to choose the parameters $\mathbf{w}$ of the discriminant function $F(r)$, such that

$$F(r_i) > F(r_j) + 1 \text{ if } y_i = 1 \text{ and } y_j = -1.$$

Checking that the Assumptions Behind Target-Decoy Strategy Apply. In general, using the target/decoy distinction as a proxy for the correct/incorrect distinction depends upon the assumption that *incorrect* matches to the real sequences in the database are distributed similarly to the matches to the shuffled database sequences (which are, by definition, incorrect). Therefore, we do not want to introduce any bias in the decoy matches that would make them trivially distinguishable from all the real matches.

We control for this effect by using target-decoy competition [Elias and Gygi, 2007]. Elias *et al.* paper argues against matching each spectrum separately to a target and then a decoy database. Instead, it advocates using a concatenated target-decoy database for search. It argues that without competition, decoy sequences that partially match to high quality MS/MS spectra may often receive elevated scores relative to other top-ranked hits, suggesting that high filtering thresholds need to be applied.

In addition, they believe that in the situation when target and decoy databases are composed such that they generate different distributions for a certain feature (for example, XCorr), regardless of the quality of the spectrum being matched, separate matching will necessarily introduce bias. Concatenating these two databases will eliminate this problem.

Therefore, we search each spectrum against a concatenated target and decoy database and then select N top-scoring peptide-spectrum matches for each spectrum (N is specified by the user, we chose 3), irrespective of whether the peptides are targets or decoys.

It is also possible to introduce a bias making targets and decoys trivially distinguishable at the protein level, as was noted in [Käll et al., 2007]. We explicitly check that this is not happening by plotting the histogram of peptide counts in target and decoy proteins. Figure 5.4 verifies that the peptide

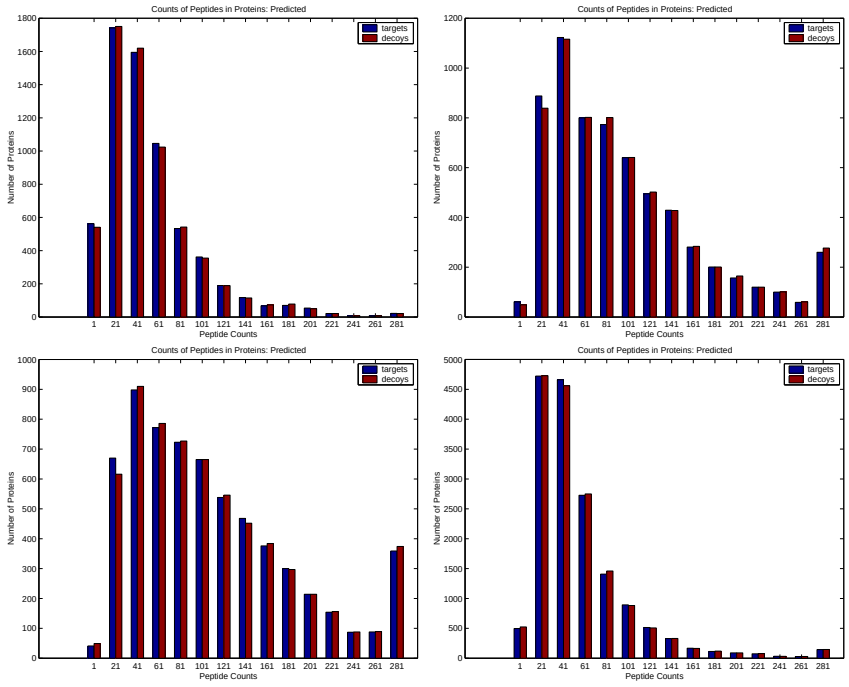


Figure 5.4: **Peptide Counts in Four Data Sets.** Peptide counts in data sets used for experiments to test the performance of the Protein Q-ranker algorithm (see description in section 5.4.1).

counts are similarly distributed. This check is crucial for any protein-level score function. If there is a significant difference in distributions of peptide counts in target and decoy proteins, they may introduce a bias allowing trivial separation between target and decoy proteins that does not reflect accurately the quality of peptide-spectrum matches belonging to them.

5.3.2 Loss Function To find the parameters $\mathbf{w}$ of the function F (equation 5.4), we search for the function in the family that best fits the empirical data. The quality of the fit is measured using a loss function defined in terms of F, the set of training proteins $\mathcal{R}$ and associated labels $\mathbf{y}$. The role of the loss function is to quantify the discrepancy between the values of $F(r_i)$ and the true labels y_i.

Similarly to the problem of peptide-spectrum match verification, the main goal is to achieve optimal pairwise ranking in the vicinity of a given q-value threshold. Therefore, we use the pairwise loss function that preferentially optimizes the performance at the top of the ranked list (see section 4.2.2).

Here we briefly repeat the definition of the loss function from section 4.2.2. Given a set of protein examples $\mathcal{R} = \{r_1..r_{\mathcal{R}}\}$ and a vector of labels $\mathbf{y}$, let N_p and N_n be the number of negative and positive examples in the data set, r_i^+ and r_j^- represent ith positive and jth negative protein example, and $\Pi(n)$ be a set of all possible permutations of integers $1, ..., n$. Suppose that the permutations $\sigma \in \Pi(n)$ and $\nu \in \Pi(n)$ are such that, given the current weights $\mathbf{w}$ of the function F, they arrange all $F(r^+_{\sigma(i)})$ and all $F(r^-_{\nu(j)})$ in non-increasing order. Then we seek to minimize

$$E(F, \mathcal{R}, \mathbf{y}) = \min[\sum_{i=1}^{N_p} \sum_{j=1}^{N_n} c_i d_j \max(0, 1 - (F(r^+_{\sigma(i)}) - F(r^-_{\nu(j)})))] \qquad (5.5)$$

where the OWA weights $\mathbf{c}$ and $\mathbf{d}$ are such that $\forall i, c_i, d_i \geq 0$, $\sum_i c_i = 1$ and

$\sum_i d_i = 1$ and $c_1 \geq c_2 \geq .. \geq c_n \geq 0$, $d_1 \geq d_2 \geq .. \geq d_n \geq 0$, i.e. $\mathbf{c}$ and $\mathbf{d}$ are non-inscreasing.

The choice of the weights $\mathbf{c}$ and $\mathbf{d}$ allows to focus on top parts of the ranked data set during training. We set a threshold t to indicate the most interesting part of the ranked data set, and we let the thresholds t_p and t_n be the counts of positive and negative proteins such that $F(r^+)$ and $F(r^-)$ rank above the threshold t, i.e. $rank(F(r^+)) < t$ and $rank(F(r^-)) < t$. The OWA weights are then set to $c_i = 1/t_p$ for $i \leq t_p$ and to $c_i = 0$ for $i > t_p$; d_j are similarly defined in terms of t_n.

Clearly, if the threshold $t = N_{\mathcal{R}}$, the total number of examples in the data set, the loss function is equivalent to the mean pairwise loss

$$E(F, \mathcal{R}, \mathbf{y}) = \frac{1}{N_p} \sum_{i=1}^{N_p} \frac{1}{N_n} \sum_{j=1}^{N_n} max(0, 1 - (F(r_i^+) - F(r_j^-))) \qquad (5.6)$$

which optimizes the ranking performance on the whole set. In the subsequent discussion, we will refer to the loss function given by equation 5.6 as "meanPL" (mean pairwise loss) optimization and to the loss function given by equation 5.5 as "Protein Q-ranker", by analogy to the previous chapter.

We train the parameters $\mathbf{w}$ of F using stochastic gradient descent. As discussed in section 4.2.3, we make several heuristic choices to optimize the loss function given by the equation 5.5. First, we perform the sorting operation at each epoch consisting of $|\mathcal{R}|$ iterations, instead of sorting at every gradient

step. Second, we choose heuristically several thresholds t to optimize and select the best resulting parameters $\mathbf{w}$. Finally, we train the meanPL loss (mean pairwise loss) function that optimizes all the pairwise constraints first and use the resulting parameters $\mathbf{w}$ to initialize the model for each new threshold t.

5.3.3 Training the Model The training proceeds as follows (Algorithm 2). We first find a discriminant function $F(r)$ using the meanPL (mean pairwise loss) given by equation 5.6, which optimizes the whole area under the ROC curve. We then focus sequentially on several thresholds t in the ranked data set. The selection of thresholds is heuristic and in our case involves 1/2 and 1/4 of the whole training set.

During training, we draw a protein pair of positive and negative examples at random from the interval given by the specified threshold and determine their scores based on the scores of the peptides contained in them. Because the parameters $\mathbf{w}$ of the peptide-spectrum match scoring function $f(\mathbf{x})$ (equation 5.2) change during training, the scores of all peptide-spectrum matches belonging to the peptides are recalculated, and a max operation is performed each time a protein is drawn. Then we make a gradient step if the pair is misranked and the loss function is differentiable at this point (see details in section 4.2.2). In the course of training, we record the best result for the speci-

fied q value after each epoch. The output is the protein ranking function $F(r)$ (equation 5.4) with best performance on the target q-value threshold and a ranked list of proteins, sorted by score.

5.3.4 Reporting Final Results: Parsimony Rules As described in section 5.1, the approaches to dealing with peptides that appear in several proteins differ in strigency. ProteinProphet [Nesvizhskii et al., 2003] partially assigns the weight of each shared peptide to all the proteins that contain this peptide. IDPicker [Zhang et al., 2007, Ma et al., 2009], on the other hand, assigns each shared peptide to a single protein among those contianing this peptide.

In our work, during *training* stage, we assign each peptide fully to all the proteins. However, when *reporting results*, we assign each shared peptide only to a single protein among those to which this peptide belongs.

More specifically, our approach to resolving the ambiguities is as described in Zhang et al. [2007]. 1) We merge all proteins that contain a common set of identified peptides into a single meta-protein, and count it as a single protein in all the reported results. 2) We identify proteins whose peptides are completely contained in another protein, and report only the larger protein. 3) For proteins sharing only a portion of their peptides, each such peptide

Algorithm 2 The input variables are the training set of target and decoy proteins $\mathcal{R}$, the corresponding binary labels $\mathbf{y}$, the target q value, the set $\mathcal{T}$ of tentative thresholds and the number n of training iterations.

procedure PROTEIN Q-RANKER($\mathcal{R}, \mathbf{y}, q_{targ}, \mathcal{T}, n$)
    $\mathbf{w} \leftarrow$ initialize using meanPL (mean pairwise loss, equation 5.6).
    for $t \in \mathcal{T}$ **do**
        $t_p \leftarrow |\{r \in \mathcal{R}^+ | rank(F(r)) < t\}|$
        $t_n \leftarrow |\{r \in \mathcal{R}^- | rank(F(r)) < t\}|$
        for $i \leftarrow 1 \ldots n$ **do**
            $r^+ \leftarrow$ chooseRandom($\mathcal{R}^+, t_p$) ▷ Randomly select a pair of examples.
            $r^- \leftarrow$ chooseRandom($\mathcal{R}^-, t_n$)
            Compute $F(r^+)$, $F(r^-)$ given by equation (5.4).
            $\mathbf{w} \leftarrow$ gradientStep($\mathbf{w}, F(r^+), F(r^-)$) ▷ Update the weights.
        end for
    end for
    Record best result on q_{targ}
    return ($\mathbf{w}$)
end procedure

is assigned to a single one of these proteins in a greedy fashion. The other proteins receive the scores based only on the remaining peptides.

5.4 Results

We compared ProteinProphet [Nesvizhskii et al., 2003], IDPicker 2.0 [Zhang et al., 2007, Ma et al., 2009] and our algorithm using four previously described data sets [Käll et al., 2007].

5.4.1 Data Set Description The first data set consists of spectra acquired from a tryptic digest of an unfractionated yeast lysate and analyzed using a four-hour reverse phase separation. Peptides were assigned to spectra using the Crux implementation of the SEQUEST algorithm [Park et al., 2008], with tryptic enzyme specificity and with no amino acid modifications enabled. The search was performed against a concatenated target-decoy database composed of open reading frames of yeast and their randomly shuffled versions. The top three peptide-spectrum matches were retained for each spectrum. For the purposes of training the model described in section 5.2, the peptide-spectrum matches are represented by 17-feature vectors given in table 5.1.

The next two data sets were derived in a similar fashion from the same yeast lysate, but treated using different proteolytic enzymes, elastase and chy-

1	XCorr	Cross correlation between calculated and observed spectra
2	ΔC_n	Fractional difference between current and second best XCorr
3	ΔC_n^L	Fractional difference between current and fifth best XCorr
4	Sp	Preliminary score for peptide versus predicted fragment ion values
5	ln(rSp)	The natural logarithm of the rank of the match based on the Sp score
6	ΔM	The difference in calculated and observed mass
7	abs(ΔM)	The absolute value of the difference in calculated and observed mass
8	Mass	The observed mass $[M+H]^+$
9	ionFrac	The fraction of matched b and y ions
10	ln(NumSp)	The natural logarithm of the number of database peptides within the specified m/z range
11	enzN	Boolean: Is the peptide preceded by an enzymatic (tryptic) site?
12	enzC	Boolean: Does the peptide have an enzymatic (tryptic) C-terminus?
13	enzInt	Number of missed internal enzymatic (tryptic) sites
14	pepLen	The length of the matched peptide, in residues
15–17	charge1–3	Three Boolean features indicating the charge state

Table 5.1: **Features Used to Represent Peptide-Spectrum Matches.** Each peptide-spectrum match obtained from the search is represented using 17 features for the Protein Q-ranker algorithm.

motrypsin. The database search was performed using no enzyme specificity and with no amino acid modifications enabled. The fourth data set is derived from a *C. elegans* lysate digested by trypsin and processed analogously to the tryptic yeast data set. The peptide-spectrum matches obtained from the search were subsequently analyzed by Protein Q-ranker, ProteinProphet and IDPicker.

5.4.2 Main Result We first present the results from training the discriminant function $F(r)$ using the meanPL (mean pairwise loss) given by the equation 5.6. Figure 5.5 shows the results of this experiment, conducted using the four data sets described in Section 5.4.1. Mean pairwise loss training outperforms ProteinProphet and IDPicker on three out of four of the experimental data sets.

These results indicate that optimizing directly at the protein level allows to combine the information about the correctness of individual peptide-spectrum matches into higher-level knowledge about the quality of all the matches belonging to the same protein. While the mean pairwise loss on the whole training set gives poor results when learned at the *peptide-spectrum match* level, it performs much better when used at the *protein* level.

We explain the poor performance on the yeast data set digested with elastase by the fact that elastase is a frequent cutter in comparison to trypsin and chymotrypsin and that it is known to be less specific. Therefore, this data set contains higher amounts of noise, which interfere with training of the mean pairwise loss on the whole training set.

We then proceed to optimize the pairwise ranking loss at the top of the ranked list given by equation 5.5 and achieve further improvement in results. Figure 5.5 demonstrates that Protein Q-ranker successfully identifies more

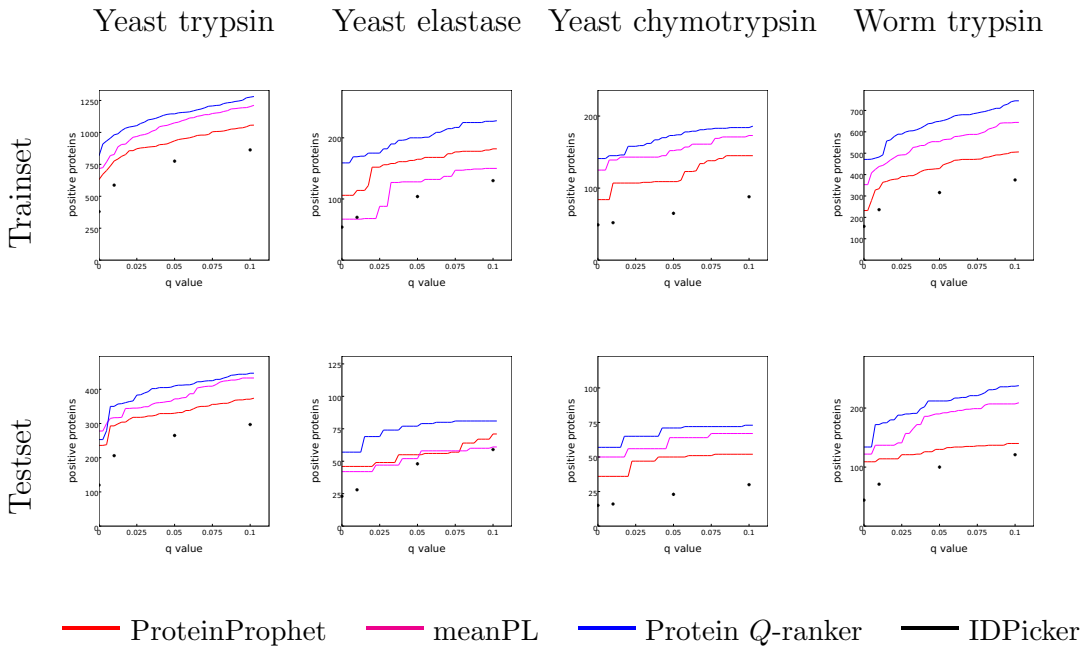


Figure 5.5: **Comparison of ProteinProphet, IDPicker and Protein Q-ranker and meanPL (mean pairwise loss).** This figure plots the number of true positive protein identifications as a function of q-value threshold.

target proteins than ProteinProphet and IDPicker across a wide range of false discovery rates and across all four data sets. At a q-value threshold of 1%, our algorithm consistently identifies more proteins on all four data sets (see table 5.2).

ProteinProphet and IDPicker do not support training a model on one data set and then applying the trained model to a separate data set. Therefore, these results are at a disadvantage. Even when we split the data into four

Method	PP	Prot Q-ranker	%>PP	IDP	%>IDP
Yeast trypsin trn	784	981	25%	585	40%
Yeast trypsin tst	283	306	8%	206	48%
Worm trypsin trn	334	482	44%	236	97%
Worm trypsin tst	126	172	36%	63	104%
Chymotrypsin trn	110	145	24%	52	170%
Chymotrypsin tst	36	51	40%	16	200%
Elastase trn	114	169	48%	70	140%
Elastase tst	48	57	18%	28	100%

Table 5.2: **Comparison of Protein Identification Methods at a Q value Threshold of 0.01.** The table lists, for each of the four data sets, the number of proteins identified at $q < 0.01$ by ProteinProphet (PP), Protein Q-ranker and IDPicker (IDP), as well as the improvement provided by Protein Q-ranker relative to the other two methods.

equal parts and train on only 3/4 of the data, Protein Q-ranker still performs better on the held-out test set than ProteinProphet and IDPicker. Furthermore, Figure 5.5 provides evidence that Protein Q-ranker is not overfitting on the training set, because the performance on the test set is similar to the performance on the training set. In the following experiments we therefore adopt Protein Q-ranker as our algorithm of choice (as opposed to mean pairwise loss), and we compare it further to ProteinProphet and IDPicker.

5.4.3 Validation Against Alternative Experimental Techniques In addition to target-decoy validation, we compared the ability of ProteinProphet, IDPicker and Protein Q-ranker to recover proteins that had been identified in yeast cells using alternative experimental methods. For this purpose, we gathered a set of 1295 proteins whose presence in yeast cells during log-phase growth is supported by three independent assays: (1) mRNA counts established by microarray analysis [Holstege et al., 1998], (2) incorporating antigen specific tags into the yeast ORFs and detecting the expression of the resulting protein with an antigen, and (3) incorporating the sequence of green fluorescent protein into the yeast ORFs and detecting the resulting fluorescence [Ghaemmaghami et al., 2003].

However, these techniques are up to 100 times less sensitive than mass

spectrometry. Therefore, if a protein was identified by mass spectrometry as well as mRNA or antibody analysis, it is highly likely to have been present in the original lysate. On the other hand, if a protein was not detected by either mRNA or antibody analysis, the mass spectrometry identification may still be valid. The results obtained from ProteinProphet confirm this assertion. For example, on the yeast data set digested with trypsin, 30 out of 300 training set proteins that received probability 1 of being present in the ProteinProphet analysis were not confirmed by either mRNA or antibody assays.

Therefore, for the purposes of validation, we assigned positive labels only to the proteins from the real database that were also identified by the mRNA or the antibody analysis, i.e. we used the union of proteins identified by the two alternative methods. However, we still used shuffled proteins as examples with negative labels (instead of assigning negative labels to the real proteins that were not identified by either of the alternative methods).

Figure 5.6 shows that, across the three yeast data sets, Protein Q-ranker's sorted list of proteins is more highly enriched with these externally validated proteins than ProteinProphet's and IDPicker's sorted lists. We also used the abundance levels assigned to the proteins identified by antibody and GFP tagging experiments [Ghaemmaghami et al., 2003] to investigate the extent to which Protein Q-ranker scores correlate with protein abundance. Figure 5.7

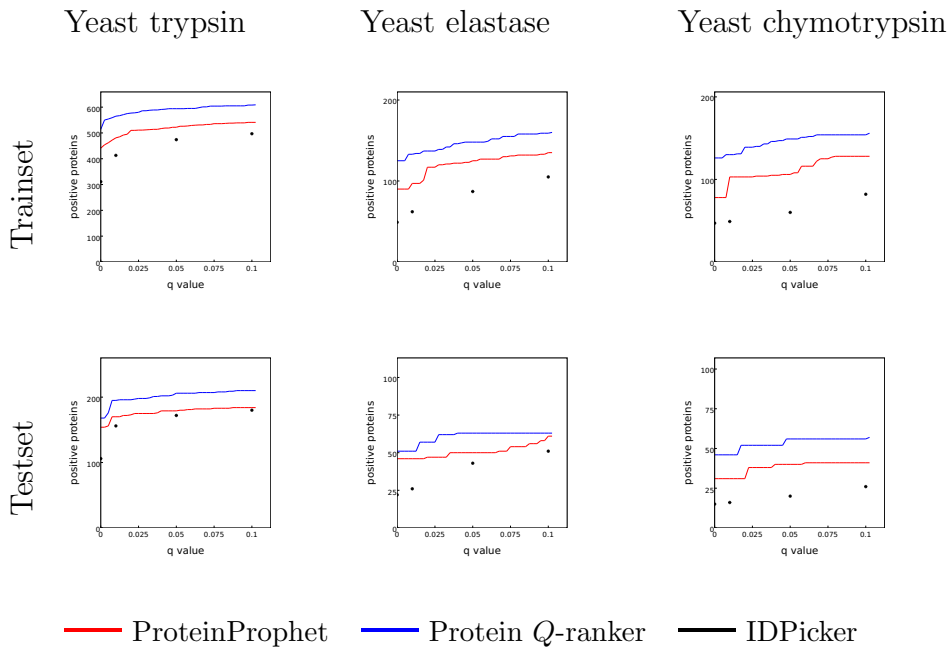


Figure 5.6: **Comparison of ProteinProphet, IDPicker and Protein Q-ranker.** This figure plots the number of externally validated yeast proteins identified by Protein Q-ranker and ProteinProphet as a function of q-value threshold.

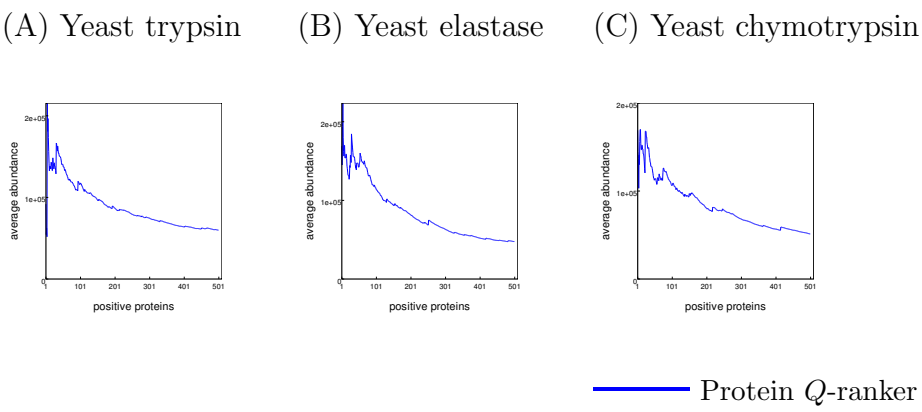


Figure 5.7: **Abundances of Proteins Identified by Protein Q-ranker.** The figure plots average protein abundance of the top n proteins, as a function of n. Protein abundances are taken from [Ghaemmaghami et al., 2003].

shows that when target proteins are ranked by Protein Q-ranker score, the top of the list is enriched with high-abundance proteins. This property, combined with Protein Q-ranker's ability to identify more target proteins at a fixed number of false positives, implies that Protein Q-ranker will successfully identify more low-abundance proteins than ProteinProphet.

5.4.4 Overlap between ProteinProphet and Protein Q-ranker To better understand the relationship between the proteins identified by ProteinProphet and Protein Q-ranker, we computed the overlap between the sets of proteins identified as true positives by the two methods at a fixed number of false positives (Figure 5.8). For all four data sets, ProteinProphet and Protein

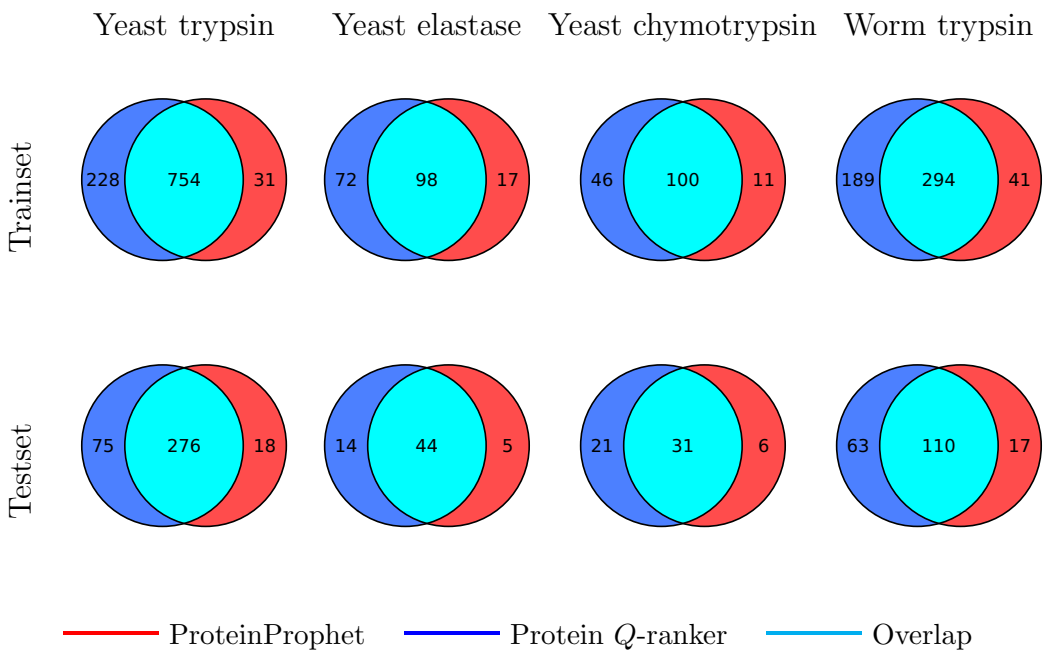


Figure 5.8: **Comparison of ProteinProphet and Protein Q-ranker.** This figure shows the overlap between proteins identified by Protein Q-ranker and ProteinProphet at q-value threshold 0.01.

Data set	q-value	Only Protein Q-ranker	Only ProteinProphet
Yeast Trypsin Trn	0.01	38%	13%
Yeast Trypsin Tst	0.01	36%	11%
Yeast Chymotrypsin Trn	0.01	49%	37%
Yeast Chymotrypsin Tst	0.01	66%	40%
Yeast Elastase Trn	0.01	42%	22%
Yeast Elastase Tst	0.01	50%	45%

Table 5.3: **External Validation of Non-overlapping Proteins** The table shows the percentage of non-overlapping proteins identified by Protein Q-ranker and ProteinProphet that were confirmed by alternative experimental methods.

Q-ranker identify many of the same proteins.

We further investigated the composition of the non-overlapping sets in the yeast data sets identified by ProteinProphet and Protein Q-ranker by checking them against the proteins established by the alternative experimental methods described above. For all four data sets, the external validation more strongly supports the Protein Q-ranker identifications than the ProteinProphet identifications (see 5.4.4).

Data set	q-value	Protein Q-ranker	ProteinProphet
Yeast Trypsin trn	0.01	524	562
Yeast Trypsin tst	0.01	578	627
Worm Trypsin trn	0.01	540	559
Worm Trypsin tst	0.01	458	530
Yeast Chymotrypsin trn	0.01	355	379
Yeast Chymotrypsin tst	0.01	359	371
Yeast Elastase trn	0.01	390	432
Yeast Elastase tst	0.01	359	367

Table 5.4: **Averages Lengths of Proteins.** The table records the average lengths of proteins below q-value threshold of 1% identified by Protein Q-ranker and ProteinProphet.

5.4.5 Length of Identified Proteins A general feature of protein identification algorithms is that they are more likely to successfully identify longer proteins, simply because such proteins contain more peptides. Protein Q-ranker is less biased against short proteins compared to ProteinProphet. The average length of the proteins identified by both methods is 577 amino acids, which is substantially longer than the average length of 451 amino acids across all proteins in the database.

5.4.6 Multitask Learning The experiments presented thus far in this chapter focused on optimizing a single value—the number of proteins identified from a shotgun proteomics experiment. This approach contrasts with previous applications of machine learning to this task [Anderson et al., 2003, Keller et al., 2002, Elias et al., 2004, Käll et al., 2007, Spivak et al., 2009a], which optimize at the level of peptide-spectrum matches or peptides. In general, selecting one optimization target or the other will depend on the goal of the proteomics experiment. However, in some applications, it may be desirable to simultaneously achieve high levels of peptide and protein identification. In such applications, we can perform joint optimization on both the protein and peptide levels. We use multi-task learning [Caruana, 1997], training the protein and peptide ranking tasks in parallel using a shared neural network representation.

For the multi-task learning, we use the mean pairwise (meanPL) loss on the whole training set given by the equation 5.6 for both proteins and peptides. For a single pair of positive protein example r^+ and negative protein example r^-, the loss function is

$$L_{prot}(r^+, r^-) = \max(0, 1 - (F(r^+) - F(r^-))) \tag{5.7}$$

where $F(r)$ is given by the equation 5.4. For a single pair of positive peptide

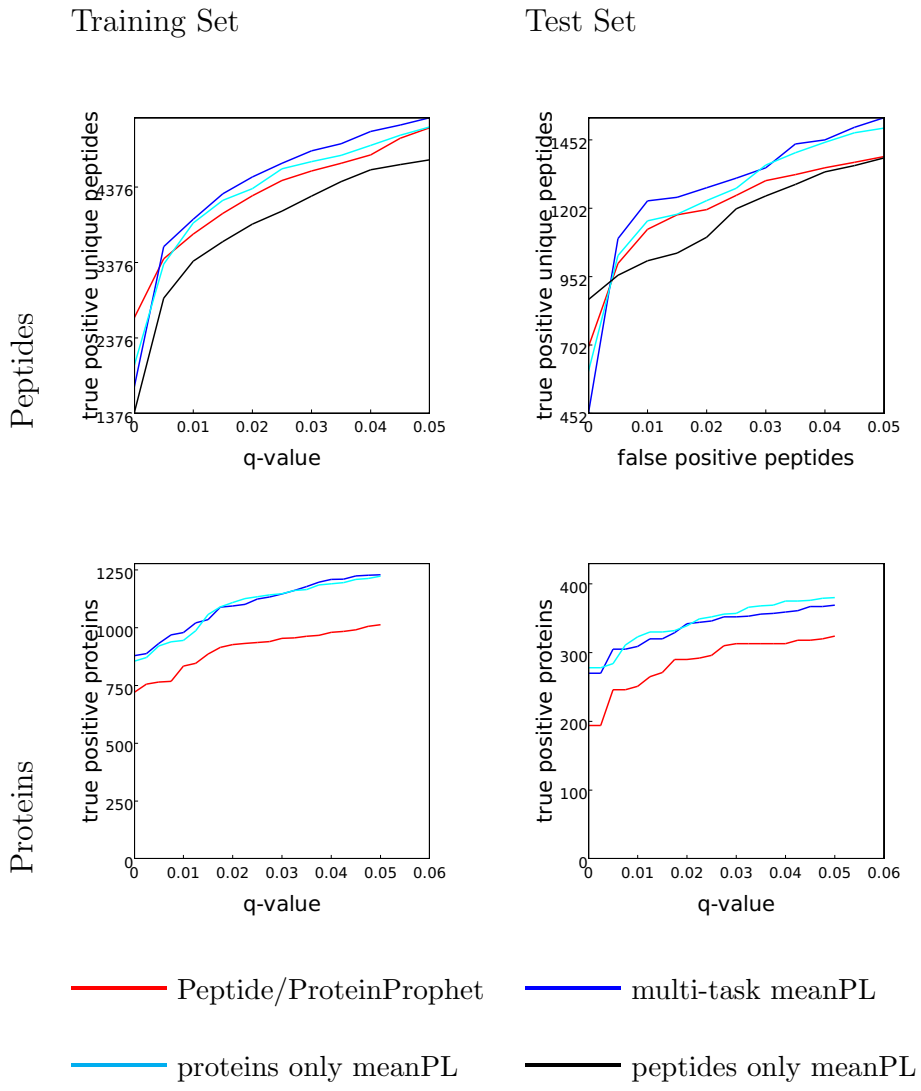


Figure 5.9: **Performance of ProteinProphet, PeptideProphet and Three Variants of Mean Pairwise Loss Training** Each panel plots the number of distinct peptides (top) or proteins (bottom) as a function of the number of false positives.

example e^+ and negative peptide example e^-, the loss function is

$$L_{pep}(e^+, e^-) = \max(0, 1 - (g(e^+) - g(e^-))) \tag{5.8}$$

where $g(e)$ is given by equation 5.3. Finally, the loss function for multi-tasking experiments is defined as

$$L_{multi} = L_{prot}(r^+, r^-) + L_{pep}(e^+, e^-) \tag{5.9}$$

The training follows the procedure described in Collobert and Weston [2008]: 1) select next task; 2) select a random pair of training examples for this task; 3) update the neural net for this task by taking a gradient step; 4) Go to 1.

Figure 5.9 compares the performance of three variants of mean pairwise loss training: optimizing protein identifications, optimizing peptide identifications and jointly optimizing both. The methods are evaluated both at the protein and peptide level. Comparing the multi-task and peptide-only training, we see that multi-tasking gives improved performance on peptide level. The improvement in peptide-level ranking occurs because the protein ranking task introduces higher-level information about the scores of all peptides belonging to the same protein.

On the other hand, comparing the multi-task and protein-only training, we see that improvements at the protein level are not significant. The reasons are the frequency with which the protein-level and peptide-level tasks were

selected during training and the learning rates for the two tasks in these particular experiments. Different choices of task frequencies and learning rates will produce more favorable results at the protein level (experiments not shown). In general, choosing whether to optimize for peptide identification, protein identification or both will depend upon the goal of the proteomics experiment.

5.5 Conclusion

In this chapter, we address directly the problem of identifying a set of proteins that can explain the observed spectra. We use machine learning techniques to solve this problem at the protein-level, yielding improved results in comparison to previous methods. We are careful to rule out the possibility of introducing a bias in our data set that can potentially arise when considering protein-level score functions. We also make several checks to ensure that our protein identifications not only meet statistical criteria, but are corroborated by other analytical and experimental methods. Finally, we report on the overlap of the identified proteins with the results of ProteinProphet [Nesvizhskii et al., 2003].

We explain the superior results of our method by the fact that protein, peptide and PSM-level optimizations are cooperative tasks and the solution of each task can benefit from the information about the solutions of the others. Our multi-task experiments corroborate this assertion. We show that solving

the problem at the *protein* level achieves superior results at the *peptide* level as well.

Due to integrating the information at the protein, peptide and PSM levels into the learning procedure, we are able to achieve superior results even with general pairwise loss functions that optimize the full area under the ROC curve, whereas the learning at the peptide-spectrum match level was not tractable using such simple objective functions. However, focused optimization of the pairwise constrains only in the vicinity of a q-value threshold of interest proves beneficial in both peptide-spectrum match and protein-level training.

Chapter 6

Learning Parameters of Theoretical Spectrum Generation

6.1 Introduction

The core problem in the analysis of shotgun proteomics data is to map each observed spectrum to the peptide sequence that generated it. Methods for solving this problem (reviewed in [Nesvizhskii et al., 2007]) can be subdivided according to whether they take as input only the observed spectrum—*de novo* methods—or take as input the observed spectrum and a database of peptides,

although the distinction between these two types of algorithms is sometimes fuzzy. In this section, we focus on the latter, database search formulation of the peptide identification problem.

Numerous previous efforts have applied machine learning methods of various types—linear discriminant analysis [Keller et al., 2002], support vector machines [Anderson et al., 2003], neural networks [Spivak et al., 2009a], decision trees [Elias et al., 2004], etc.—as post-processors on the initial database search. These methods typically integrate information across a given data set and also integrate additional pieces of information that were not available to the search engine.

In this work, we aim to integrate machine learning directly into the search procedure. Rather than ranking peptides once with respect to a static score function and then post-processing the results, we perform a learning procedure that optimizes the score function used in the search itself.

The key idea behind the current work is to parameterize the model that generates a theoretical spectrum from peptides in the database and then learn the parameters of this model. The method requires that the database contain a mixture of real (target) and shuffled (decoy) peptides and we use a learning procedure to learn model parameters that rank targets above decoys. We demonstrate that this optimization yields improved performance relative to

the baseline Q-ranker model at the protein level described in the previous chapter.

6.2 Parameterizing the Theoretical Spectrum

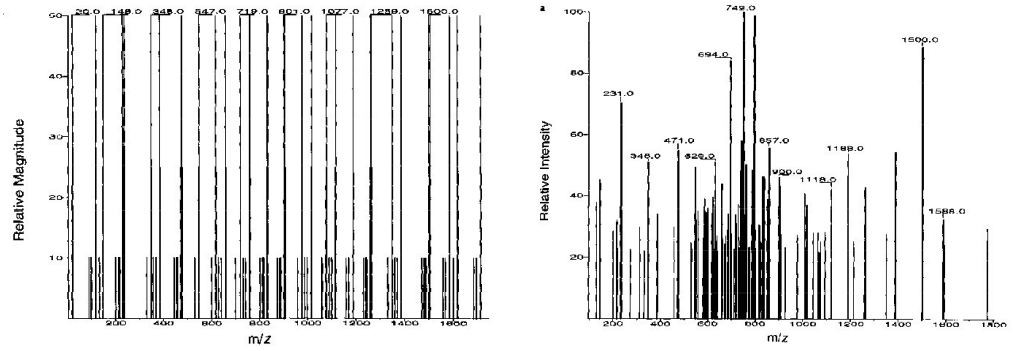
6.2.1 SEQUEST-style Search SEQUEST's theoretical spectrum generator converts a charged peptide string into a spectrum by identifying all prefix and suffix ions (b-ions and y-ions), and generating six peaks for each ion (see review on how the peptides break in section 2.2.1). The six peaks correspond to

1. the primary peaks, with an m/z value based on the sum of the masses of the amino acids in the corresponding b- or y-ion,

2. flanking peaks, occupying the 1-Da bins on either side of the primary peak, and

3. three neutral loss peaks corresponding to loss of water, ammonia or carbon monoxide.

SEQUEST assigns the primary peak a height of 50, flanking peaks heights of 25 and neutral loss peaks heights of 10 (figure 6.1). Because only the relative magnitudes of these peaks affect the calculation of XCorr between theoretical



a) b)

Figure 6.1: **Theoretical and Experimental Spectra.** This figure shows an example of theoretical (a) and experimental (b) spectra for the peptide sequence DLRSWTAADTAAQISQ [Eng et al., 1994]. Theoretical spectrum contains 3 types of peaks - main(b- or y-ion) , flanking, neutral loss - and their intensities are arbitrarily set to 50, 25 and 10 respectively.

and observed spectrum, this spectrum generator has two free parameters: the relative height of the flanking peaks and of the neutral loss peaks.

The SEQUEST search routine can be summarized as follows. For a given observed spectrum, the algorithm first identifies all database peptides whose mass lies within a specified range of the inferred mass of the original peptide that generated the spectrum. Next, for each candidate peptide in the database, the search computes the $XCorr$ function $\Xi(\mathbf{t}, \mathbf{s})$, where $\mathbf{s}$ is the observed spectrum, $\mathbf{t}$ is the theoretical spectrum, and Ξ is given by

$$\Xi(\mathbf{t}, \mathbf{s}) = \sum_{i=1}^{N} t_i s_i - \frac{1}{151} \sum_{\tau=-75, \tau \neq 0}^{75} \sum_{i=1}^{N} t_i s_{i+\tau} \tag{6.1}$$

The output is a ranking of candidate peptides according to Ξ for each observed spectrum $\mathbf{s}$.

Eng et al. [Eng et al., 2008] showed that SEQUEST-style cross-correlation can be calculated efficiently:

$$\Xi(\mathbf{t}, \mathbf{s}) = \mathbf{t}^T * \mathbf{s}' \tag{6.2}$$

where

$$s'_i = s_i - (\sum_{\tau=-75}^{\tau=75} s_\tau)/150 \tag{6.3}$$

and $\mathbf{s}'$ can be computed only once for each observed spectrum $\mathbf{s}$.

6.2.2 Learning the Theoretical Spectrum Peak Heights We make a simple observation that the SEQUEST scoring function (6.2) is a linear oper-

ator on the theoretical spectrum and post-processed version of the observed spectrum given by (6.3). According to the SEQUEST model, the theoretical spectrum $\mathbf{t}$ contains three types of peaks with different heights. Therefore, the theoretical spectrum $\mathbf{t}$ can be represented as a weighted sum of three vectors containing single type of ions with unit intensities: $\mathbf{t}_{by}$ containing only b and y ion peaks, $\mathbf{t}_f$ containing only the corresponding flanking peaks and $\mathbf{t}_l$ containing only the neural loss peaks.

Then,

$$\mathbf{t} = w_1 * \mathbf{t}_{by} + w_2 * \mathbf{t}_f + w_3 * \mathbf{t}_l \tag{6.4}$$

where $\mathbf{t}, \mathbf{t}_{by}, \mathbf{t}_f, \mathbf{t}_l \in R^N$ and N is the number of bins used to discretize the observed and theoretical spectra. The weights $w_1, w_2, w_3 \in R$ are the model parameters to be determined by the learning procedure. (To re-iterate, in the SEQUEST algorithm, these weights are arbitrarily set to $w_1 = 50$, $w_2 = 25$, $w_3 = 10$.)

Let three products P_{by}, P_f and P_l be defined as

$$P_{by} = \mathbf{t}_{by}^T * \mathbf{s}' \tag{6.5}$$

$$P_f = \mathbf{t}_f^T * \mathbf{s}' \tag{6.6}$$

$$P_l = \mathbf{t}_l^T * \mathbf{s}' \tag{6.7}$$

Since the cross-correlation is a linear operator, it can be represented in terms

of these three products and the parameters $\mathbf{w} = (w_1, w_2, w_3)$ as

$$\Xi(\mathbf{t}, \mathbf{s}, \mathbf{w}) = w_1 * P_{by} + w_2 * P_f + w_3 * P_l \tag{6.8}$$

The products can be precomputed in advance.

The result is a parametrized expression for SEQUEST cross-correction which can be optimized using simple gradient descent procedures, provided a set of negative and positive examples and a loss function are available. Moreover, this approach can potentially be generalized to linear operators other than that currently used in SEQUEST.

6.2.3 Learning Model Parameters Given the scoring scheme described above, the goal of the learning procedure is to select the parameters of the theoretical spectrum generation model, based on a given set of observed spectra $\mathcal{S} = \{s_1, \ldots, s_{N_s}\}$ and a peptide database $\mathcal{E} = \{e_1, \ldots, e_{N_e}\}$. For simplicity, in this report, we assume that the charge state of each spectrum is known *a priori*. Hence, for each spectrum, we can retrieve from the database a set of all candidate peptides whose total mass is within a specified range of the precursor mass of the spectrum. We denote the set of theoretical spectra derived from candidate peptides for spectra $\mathcal{S}$ as $\mathcal{T} = \{t_1, \ldots, t_{N_t}\}$.

Assigning labels to peptide-spectrum matches To create a data set for training, we identify all candidate peptides for each spectrum and then assign

positive and negative labels to peptide-spectrum pairs based on the target-decoy approach. For the target-decoy learning, we require that the search is performed against a database comprised of a mixture of real (target) and shuffled (decoy) peptides. We assign positive labels to matches to real peptides ($y = 1$) and negative labels to matches to decoy peptides ($y = -1$).

Note that, with this labeling scheme, the vast majority of the positively labeled peptide-spectrum matches are actually incorrect matches. This is because, in principle, only one candidate peptide among a set of target candidate peptides is actually responsible for generating the observed spectrum. Thus, similar to the problems solved by Percolator [Käll et al., 2007] and Q-ranker [Spivak et al., 2009a], this problem is semi-supervised: the negative examples are all truly negative, but the "positive" examples are more accurately described as unlabeled, because the "positive" set actually consists of a mixture of (mostly) negative examples and a few positive examples.

Loss Function We would like to choose the parameters $\mathbf{w} = (w_1, w_2, w_3)$ so that when the peptide-spectrum matches are ordered by the parametrized correlation function described above, the top-ranked examples would have positive labels.

Let each peptide-spectrum match consist of observed spectrum $\mathbf{s}$ and the-

oretical spectrum $\mathbf{t}$. Since the final goal is ranking, we choose a ranking loss function: for each pair of positive and negative peptide-spectrum matches, such that $y_j = 1, y_k = -1$, we prefer that $\Xi(\mathbf{t}_j, \mathbf{s}_j, \mathbf{w}) > \Xi(\mathbf{t}_k, \mathbf{s}_k, \mathbf{w})$, i.e. the positive example is scored higher.

$$E(\mathcal{S}, \mathcal{T}, \mathbf{w}) = \sum_{s \in \mathcal{S}, y_i = 1, y_j = -1} \max(0, 1 - (\Xi(\mathbf{t}_i, \mathbf{s}_i, \mathbf{w}) - \Xi(\mathbf{t}_j, \mathbf{s}_j, \mathbf{w}))) \qquad (6.9)$$

We use stochastic gradient descent to find the appropriate parameters $\mathbf{w}$.

6.3 Results

For our experiments, we used the data sets described in section 5.4.1. We made 10 runs starting from random parameter values on each of the four described data sets. During the training, we monitored the decrease in the objective function (6.9) after each epoch, and stopped when the value of the objective function from the previous epoch was less than that of the current epoch, i.e. $E_{k-1} < E_k$. In other words, the training stopped as soon as the objective function did not decrease monotonically.

We recorded the average values of the learned parameters $\mathbf{w}$, normalized such that the height of the b and y-ion peaks remained 50 (since the heights of the flanking and neutral loss peaks are defined relative to the height of the main peaks). We also recorded the standard deviation over the 10 runs (table

	Yeast trypsin		Worm trypsin		Yeast elastase		Yeast chymotrypsin	
	mean	std	mean	std	mean	std	mean	std
BY-ions	50.00	0.08	50.00	0.04	50.00	0.01	50.00	0.20
Flanking	0.00	0.00	0.05	0.00	0.01	0.00	0.00	0.00
Neut. loss	18.00	0.07	16.00	0.03	18.00	0.05	18.00	0.09

Table 6.1: **Average Values of Ion Peak Heights.** The table shows the average values of the peak heights and their standard deviations learned over 10 runs.

6.1).

The training resulted in the following peak intensities for the model: b,y-ions are 50, neutral loss peaks are 18, the flanking peaks are 0 (i.e. they are random, do not carry useful information). This result makes sense because flanking peaks are not based on the model of how peptides break, but are used to account for the fact that mass spectrometry peaks can be wide and spread into several bins.

6.3.1 Learning in the Context of Protein Identification We now use the parametrization approach developed for the SEQUEST cross-correlation function to modify the feature set for peptide-spectrum matches serving as

input to the protein-level learning algorithm described in the previous chapter. In all the previous work, we represented peptide-spectrum matches as 17-feature vectors given in table 5.1, one of these features is the value of the cross-correlation function, XCorr. This set of features can be modified such that the single XCorr feature is replaced with three products P_{by}, P_f and P_l.

The new feature set is given in table 6.2. Each peptide-spectrum match obtained from the search is now represented using 19 features, because the feature XCorr in the original feature set (see 5.1) is replaced by 3 features: P_{by}, P_f and P_l (equations 6.5, 6.6 and 6.7).

For the experiments with the augmented feature set, we use the mean pairwise loss that attempts to satisfy all the pairwise constraints in the training set. In the previous chapter, we referred to this loss function as "meanPL". Figure 6.2 shows that, on all four datasets, meanPL with extended 19-feature set achieves substantially better results than the same loss function on the 17-feature set, when trained at the protein level. Moreover, meanPL with extended feature set achieves better or as good results as the algorithm that optimizes the pairwise constraints preferentially at the top of the ranked list, but uses the old 17-feature representation (i.e. the Protein Q-ranker algorithm described in the previous chapter).

We also repeated the verification procedures reported in the previous chap-

1	P_{by}	see equation 6.5
2	P_f	see equation 6.6
3	P_l	see equation 6.7
4	ΔC_n	Fractional difference between current and second best XCorr
5	ΔC_n^L	Fractional difference between current and fifth best XCorr
6	Sp	Preliminary score for peptide versus predicted fragment ion values
7	ln(rSp)	The natural logarithm of the rank of the match based on the Sp score
8	ΔM	The difference in calculated and observed mass
9	abs(ΔM)	The absolute value of the difference in calculated and observed mass
10	Mass	The observed mass $[M+H]^+$ calculated and observed mass
11	ionFrac	The fraction of matched b and y ions
12	ln(NumSp)	The natural logarithm of the number of database peptides within the specified m/z range
13	enzN	Boolean: Is the peptide preceded by an enzymatic (tryptic) site?
14	enzC	Boolean: Does the peptide have an enzymatic (tryptic) C-terminus?
15	enzInt	Number of missed internal enzymatic (tryptic) sites
16	pepLen	The length of the matched peptide, in residues
17–19	charge1–3	Three Boolean features indicating the charge state

Table 6.2: **Extended Feature Set Used to Represent Peptide-Spectrum Matches.** Each peptide-spectrum match obtained from the search is now represented using 19 features, because the feature XCorr in the original feature set (see 5.1) is replaced by 3 features: P_{by}, P_f and P_l (equations 6.5, 6.6 and 6.7).

ter for the meanPL training with the new extended 19-feature set. Figure 6.3 shows that the identities of the proteins at the top of the ranked list produced by meanPL on the 19-feature data sets were confirmed by alternative experimental techniques. Finally, figure 6.4 shows that majority of the proteins identified by meanPL with 19-feature set were also identified by ProteinProphet.

6.4 Conclusion

There have been developed many quite sophisticated algorithms to enable protein inference from the peptide-spectrum matches, previously obtained from the database search. But the database search itself has not benefited from the same variety of machine learning techniques. However, it is conceivable that a lot of information is lost during the database search due to heuristic choices made at this step of the analysis.

The results presented in this chapter confirm that the limitations of the theoretical spectrum generation during the database search are contributing to the difficulties in protein identification. These results raise further possibilities of making the model more flexible by allowing more types of peaks in the theoretical spectrum and learning their intensities in the context of protein identification. Another feasible direction is to explore various functions used to match observed and theoretical spectra and to choose a more general family of

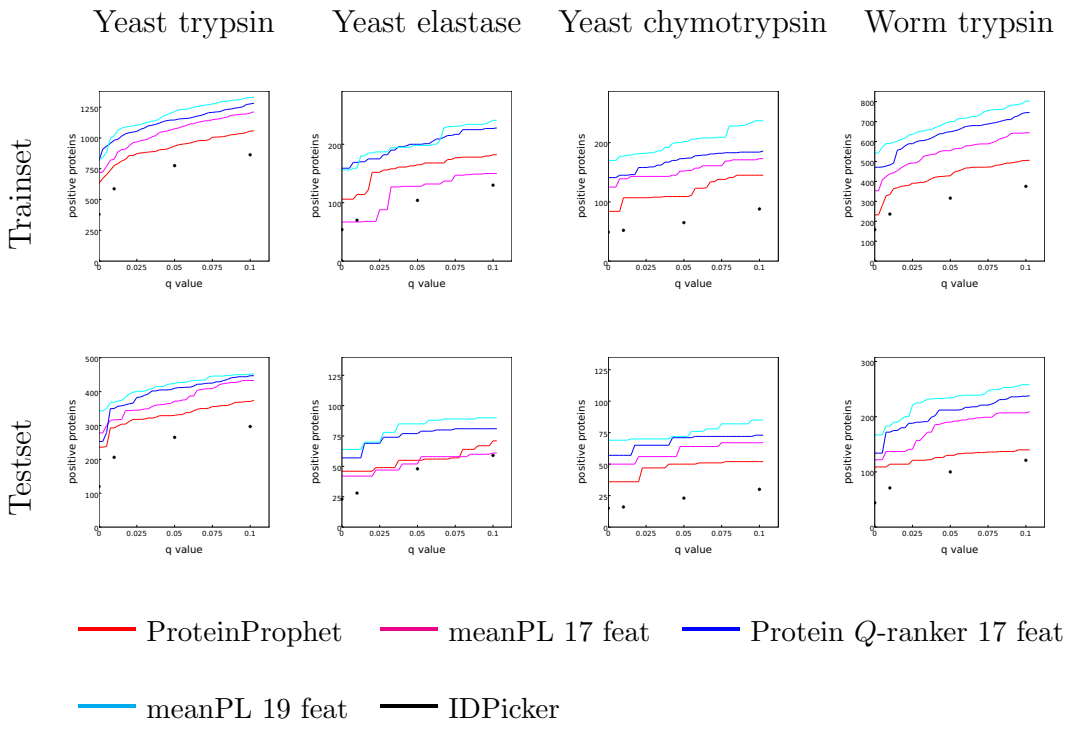


Figure 6.2: **Comparison of ProteinProphet, Protein Q-ranker and meanPL (mean pairwise loss).** This figure compares the performance of meanPL (mean pairwise loss) and Q-ranker on 17-feature data set (table 5.1), with meanPL on 19-feature data set (table 6.2). It plots the number of true positive protein identifications as a function of q-value threshold.

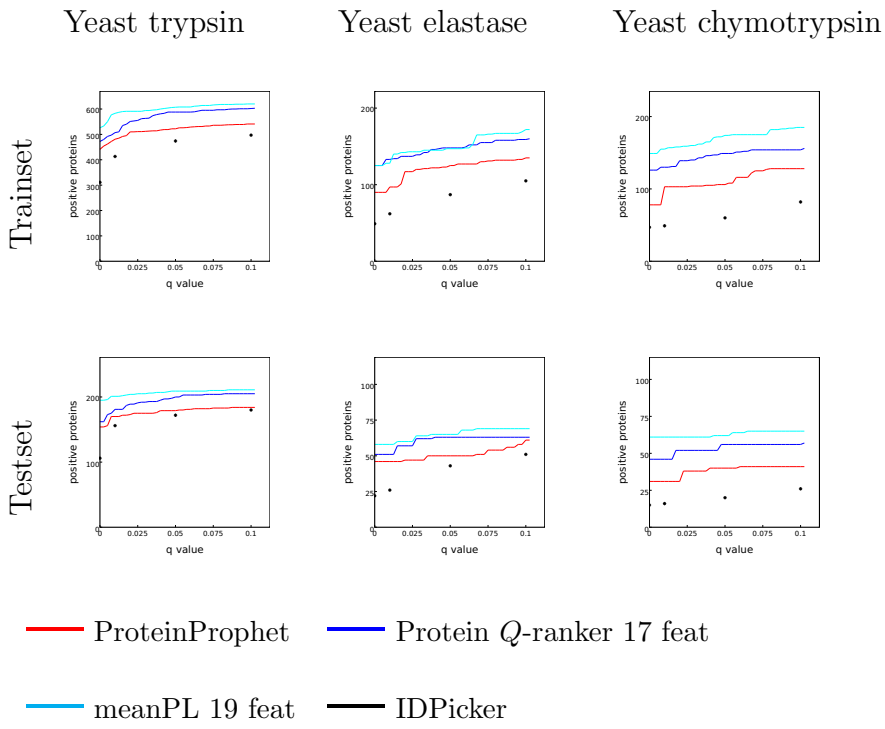


Figure 6.3: **Validation of ProteinProphet, Protein Q-ranker and meanPL (mean pairwise loss).** This figure compares the performance of ProteinProphet, Protein Q-ranker on 17-feature data set and meanPL on 19-feature data set, when validated against alternative experimental methods. It plots the number of externally validated protein identifications as a function of q-value threshold.

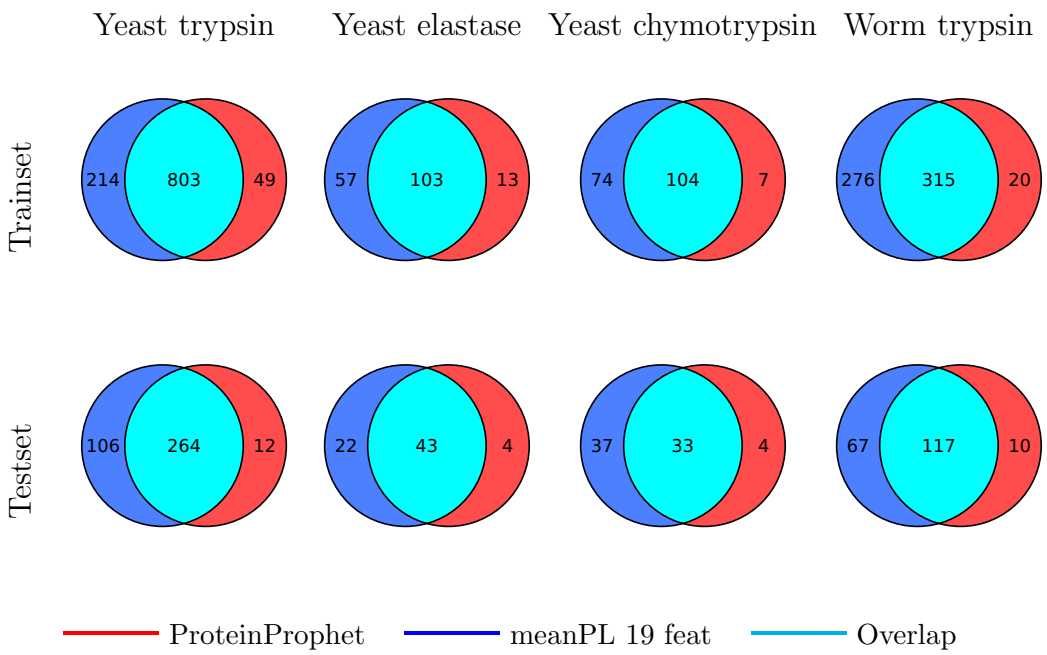


Figure 6.4: **Comparison of Proteins Identified by ProteinProphet and meanPL on 19-feature Data Set.** The figure shows the overlap between proteins identified by meanPL and ProteinProphet at q-value threshold 0.01.

functions with adjustable parameters and learn these parameters on empirical data sets.

Chapter 7

Conclusions and Future Work

In this dissertation, we address several issues relevant to protein identification in complex biological mixtures using mass spectrometry data. First, we consider the algorithmic framework for learning with examples of a single class and unlabeled examples. We propose a ranking "loss" function designed for this specific setting that produces orderings with high precision at the top of the list. Second, we build on this approach to develop an algorithm for protein identification that exploits the structural information in the mass spectrometry data that has generally been ignored in existing approaches to the problem. Finally, we use machine learning techniques to replace some of the heuristic choices of scoring functions made in mass spectrometry algorithms with parametrized functions that can be adjusted based on empirical data.

The current work opens several directions for future research. First, the algorithms discussed in chapters 3-5 take as input the results of database search which performs the crucial task of matching spectra to peptide sequences. However, the state-of-the art database search algorithms make multiple heuristic choices in their definition of the cross-correlation function used to measure the similarity between observed and theoretical spectra. In chapter 6 of this dissertation we use machine learning techniques to relax one of these choices by adjusting the peak heights of the theoretical spectra that serve as one of the inputs to the cross-correlation function. However, a more general approach would be to choose a broader family of parametrized function and learn their parameters based on empirical data.

Second, the discussion of the protein identification in chapter 5 highlights the necessity of going beyond the standard training-testing set validation of the results of the algorithms developed for this task. In this dissertation, we used protein sets recognized by alternative experimental methods as expressed in yeast cells to verify the proteins identified by our algorithm. However, proteome screening using several experimental procedures is too labor-intensive and expensive to be performed for every organism of interest. One possible approach to validation problem is to simulate the results of mass spectrometry runs and validate the results of the protein identification algorithms against

known protein sets used for simulation.

Finally, the current work addresses the algorithmic problems of learning with labeled examples of a single class and unlabeled examples (reviewed in Zhang and Zuo [2008]). In essence, all of the known machine learning approaches turn this problem into either supervised or semi-supervised learning tasks and then apply standard classification algorithms. In every case the solution will depend on the particular assumptions made during the formulation of the two-class problem. It is reasonable to ask: how stable are these solutions? The question represents a more general study of stability of algorithms depending on the assumptions used in setting up the two-class learning problem. This topic remains largely unexplored.

The incorporation of these ideas into biological data analysis is likely to lead to important breakthroughs in mass spectrometry as well as other large scale screening/classification techniques.

Bibliography

P. Alves, R. J. Arnold, M. V. Novotny, P. Radivojac, J. P. Reilly, and H. Tang. Advancement in protein inference from shotgun proteomics using peptide detectability. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 409–420, Singapore, 2007. World Scientific.

D. C. Anderson, W. Li, D. G. Payan, and W. S. Noble. A new algorithm for the evaluation of shotgun peptide sequencing in proteomics: support vector machine classification of peptide MS/MS spectra and SEQUEST scores. *Journal of Proteome Research*, 2(2):137–146, 2003.

M. Bern and D. Goldberg. Improved ranking functions for protein and modification-site identifications. *Journal of Computational Biology*, 15:705–719, 2008.

C. Bishop. *Neural Networks for Pattern Recognition*. Oxford UP, Oxford, UK, 1995.

L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Algorithms and Neural Networks*. Cambridge University Press, 1998.

C. J. C. Burges. Ranking as learning structured outputs. In *Proceedings of the NIPS 2005 Workshop on Learning To Rank*, 2005.

C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Proc. of Adv. in Neural Inf. Processing Syst.*, pages 193–200, 2006.

R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-supervised Learning*. MIT Press, Cambridge, MA, 2006.

H. Choi and A. I. Nesvizhskii. Semisupervised model-based validation of peptide identifications in mass spectrometry-based proteomics. *Journal of Proteome Research*, 7(1):254–265, 2008.

J. Colinge, A. Masselot, M. Giron, T. Dessingy, and J. Magnin. OLAV: Towards high-throughput tandem mass spectrometry data identification. *Proteomics*, 3:1454–1463, 2003.

R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the International Conference on Machine Learning*, 2008.

Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Large scale transductive svms. *Journal of Machine Learning Research*, 7:1687–1712, 2006.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20: 273–297, 1995.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–22, 1977.

Y. Ding, H. Choi, and A. Nesvizhskii. Adaptive discriminant function analysis and reranking of MS/MS database search results for improved peptide identification in shotgun proteomics. *Journal of Proteome Research*, 7(11): 4878–4889, 2008.

J. E. Elias and S. P. Gygi. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nature Methods*, 4(3):207–214, 2007.

J. E. Elias, F. D. Gibbons, O. D. King, F. P. Roth, and S. P. Gygi. Intensity-based protein identification by machine learning from a library of tandem mass spectra. *Nature Biotechnology*, 22:214–219, 2004.

J. K. Eng, A. L. McCormack, and J. R. Yates, III. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry*, 5: 976–989, 1994.

J. K. Eng, B. Fischer, J. Grossman, and M. J. MacCoss. A fast SEQUEST cross correlation algorithm. *Journal of Proteome Research*, 7(10):4598–4602, 2008.

S. Ghaemmaghami, W. K. Huh, K. Bower, R. W. Howson, A. Belle, N. Dephoure, E. K. O'Shea, and J. S. Weissman. Global analysis of protein exression in yeast. *Nature*, 425:737–741, 2003.

J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *Proceedings of the Ninth International Conference on Articial Neural Networks*, pages 97–102, 1999.

F. C. P. Holstege, E. G. Gennings, J. J. Wyrick, T. I. Lee, C. J. Hengartner, M. R. Green, T. R. Golub, E. S. Lander, and R. A. Young. Dissecting the regulatory circuitry of eukaryotic genome. *Cell*, 95:717–728, 1998.

T. Joachims. Optimizing search engines using clickthrough data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 133–142, 2002.

K. R. Jonscher. Validating sequence assignments for peptide fragmentation patterns: A primer in ms/ms sequence identification. *Proteome Software*, June 2005.

L. Käll, J. Canterbury, J. Weston, W. S. Noble, and M. J. MacCoss. A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets. *Nature Methods*, 4:923–25, 2007.

A. Keller, A. I. Nesvizhskii, E. Kolker, and R. Aebersold. Empirical statistical model to estimate the accuracy of peptide identification made by MS/MS and database search. *Analytical Chemistry*, 74:5383–5392, 2002.

A. A. Klammer and M. J. MacCoss. Effects of modified digestion schemes on the identification of proteins from complex mixtures. *Journal of Proteome Research*, 5(3):695–700, 2006.

A. A. Klammer, S. R. Reynolds, M. Hoopmann, M. J. MacCoss, J. Bilmes, and W. S. Noble. Modeling peptide fragmentation with dynamic Bayesian networks yields improved tandem mass spectrum identification. *Bioinformatics*, 24(13):i348–i356, 2008.

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G.B. Orr and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

Q. L. Li and B. Liu. Learning to classify text using positive and unlabeled data. *Proceedings of Eighteenth International Joint Conference on Aritificial Inteligence, (IJCIA-03)*, 2003.

Y. F. Li, R. J. Arnold, Y. Li, P. Radivojac, Q. Sheng, and H. Tang. A Bayesian approach to protein inference problem in shotgun proteomics. In M. Vingron and L. Wong, editors, *Proceedings of the Twelfth Annual International Conference on Computational Molecular Biology*, volume 12 of *Lecture Notes in Bioinformatics*, pages 167–180, Berlin, Germany, 2008. Springer.

B. Liu, W.S. Lee, P.S. Yu, and X.L. Li. Partially supervised classification of text documents. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002)*, pages 387–394, Sydney, July 2002.

B. Liu, W.S. Lee, and Philip Y. Building text classifiers using positive and unlabled examples. *ICDM-03*, pages 19–22, Nov. 2003.

Z.-Q. Ma, S. Dasari, M. C. Chambers, M. Litton, S. M. Sobecki, L. Zimmerman, P. J. Halvey, B. Schilling, P. M. Drake, B. W. Gibson, and D. L. Tabb. IDPicker 2.0: Improved protein assembly with high discrimination peptide identification filtering. *Journal of Proteome Research*, 2009.

L. Mason, P. L. Bartlett, and J. Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 38(3):243–255, 2000.

R. E. Moore, M. K. Young, and T. D. Lee. Qscore: An algorithm for evaluating SEQUEST database search results. *Journal of the American Society for Mass Spectrometry*, 13(4):378–386, 2002.

M. C. Mozer, R. Dodier, M. D. Colagrosso, C. Guerra-Salcedo, and R. Wolniewicz. Prodding the roc curve: Constrained optimization of classifier performance. In *NIPS-2002*, 2002.

A. I. Nesvizhskii, A. Keller, E. Kolker, and R. Aebersold. A statistical model for identifying proteins by tandem mass spectrometry. *Analytical Chemistry*, 75:4646–4658, 2003.

A. I. Nesvizhskii, O. Vitek, and A. R. Aebersold. Analysis and validation of

proteomic data generated by tandem mass spectrometry. *Nature Methods*, 4(10):787–797, 2007.

Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

M. Muller P. Hernandez and R. D. Appel. Automated protein identification by tandem mass spectrometry: Issues and strategies. *Mass Spectrometry Reviews*, 25:235–254, 2006.

C. Y. Park, A. A. Klammer, L. Käll, M. P. MacCoss, and W. S. Noble. Rapid and accurate peptide identification from tandem mass spectra. *Journal of Proteome Research*, 7(7):3022–3027, 2008.

R. G. Sadygov, D. Cociorva, and J. R. Yates, III. Large-scale database searching using tandem mass spectra: Looking up the answer in the back of the book. *Nature Methods*, 1(3):195–202, 2004.

X. Shen, G. C. Tseng, X. Zhang, and W. H. Wong. On (psi)-learning. *Journal of the American Statistical Association*, 98(463):724–734, 2003.

M. Spivak, J. Weston, L. Bottou, L. Käll, and W. S. Noble. Improvements to

the percolator algorithm for peptide identification from shotgun proteomics data sets. *Journal of Proteome Research*, 8(7):3737–3745, 2009a.

M. Spivak, J. Weston, M. J. MacCoss, and W. S. Noble. Direct maximization of protein identifications from tandem mass spectra. Manuscript under review, 2009b.

H. Steen and M. Mann. The ABC's (and XYZ's) of peptide sequencing. *Nature Reviews Molecular Cell Biology*, 5:699–711, 2004.

J. D. Storey and R. Tibshirani. Statistical significance for genome-wide studies. *Proceedings of the National Academy of Sciences of the United States of America*, 100:9440–9445, 2003.

N. Usunier, D. Buffoni, and P. Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1057–1064, 2009.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

V. N. Vapnik. *Statistical Learning Theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley, New York, 1998.

J. Weston. Large-scale semi-supervised learning. *Proceedings of the NATO Advanced Study Institute on Mining Massive Data Sets for Security*, 2008.

R. R. Yager. On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18:183–190, 1988.

H. Yu, J. Han, and K. C.-C. Chang. Pebl: Positive example based learning for web page classification using svm. *Proc. Eighth Int'l Conf. Knowledge Discovery and Data Mining*, pages 236–248, 2002.

B. Zhang and W. Zuo. Learning from positive and unlabeled examples: A survey. In *2008 International Symposiums on Information Processing*, May 2008.

B. Zhang, M. C. Chambers, and D. L. Tabb. Proteomic parsimony through bipartite graph analysis improves accuracy and transparency. *Journal of Proteome Research*, 6(9):3549–3557, 2007.