Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelor Thesis Bioinformatics

Semi-supervised Learning for Nucleic Acid Cross-linking Mass Spectrometry

Emil Paulitz

14.08.2020

Reviewer

Prof. Oliver Kohlbacher
Department of Computer Science
University of Tübingen

Supervisor

Timo Sachsenberg
Applied Bioinformatics Group
University of Tübingen

Paulitz, Emil:
Semi-supervised learning for nucleic acid cross-linking mass spectrometry
Bachelor Thesis Bioinformatics
Eberhard Karls Universität Tübingen
Period: 14.04.2020-14.08.2020

Abstract

Modern analysis instruments in life sciences produce data on an ever-larger scale. While this allows more comprehensive insight into biological mechanisms, it also requires computational methods able to process the data appropriately. Currently, the method of choice for analyzing a proteome, the entirety of proteins in a cell or an organism, is mass spectrometry, which produces spectra containing information corresponding to the peptide's amino acid sequence. An important step in matching every spectrum to the peptide they originate from, is the well-established Percolator algorithm. It combines every information about the spectrum, peptide and scores for the similarity of peptide-spectrum-match, to achieve best possible separation between correct and incorrect matches.
In this thesis, the algorithm was transferred from C++ into the python programming language. This implementation increases the flexibility and extends the range of potential applications by enabling the use of machine learning libraries like scikit-learn, which can be utilized to adapt the algorithm more easily to the special demands of one's experiment. As such, we present Pycolator, an adaptation of Percolator to the characteristics of peptides cross-linked onto (ribo-) nucleotides, for which the original algorithm often does not provide optimal results. Pycolator finds 1% more cross-linked peptides on the tested dataset and employs techniques to reduce variation. The analysis of such proteins promises to yield insight about the interaction between peptides and DNA or RNA, possibly providing a more comprehensive view on the interactions driving biological systems.

Acknowledgements

I would like to express my gratitude to my supervisor Timo Sachsenberg, who over all of our online meetings had input, ideas and inspiration, never tiring of explaining unknown concepts and even responded to my questions on weekends.
I also want to thank my proof-readers Jonathan Paulitz (especially for all the personal and general tips), Benedikt Rössler and Simon Schäfermann, as well as Carmen Gil Bredehöft for the best colors, fonts and the perfect refreshment in critical moments.
Lastly, I want to thank my parents for their continuous support and love, and for enabling me the studies I cherish. Also my cats for keeping me company and warming me every 10 minutes in the final phase of writing.

In accordance with the standard scientific protocol, I will use the personal pronoun *we* to indicate the reader and the writer or my scientific collaborators and myself.

Contents

List of Figures

List of Tables

List of Abbreviations

MS	Mass Spectrometry
LC	Liquid Chromatography
MS/MS	Tandem Mass Spectrometry
PSM	Peptide Spectrum Match
FDR	False Discovery Rate
ROC	Receiver Operating Characteristics
TPR	True Positive Rate
FPR	False Positive Rate
SVM	Support Vector Machine
AUC	Area Under the Curve

Chapter 1

Introduction

As experimental methods to probe biological systems advanced, many observed phenomena like illnesses could be linked to molecular-scale mechanisms. With next generation DNA sequencing, for example, genetic diseases and their exact origin could be identified. While for illnesses like Bovine spongiform encephalopathy (BSE) a linkage to misfolded proteins was found, other illnesses like Alzheimer disease, which is also possibly connected to misfolded proteins, still lack general treatment and a definite cause [4]. This shows the need for further research and understanding of proteins in biological systems.
Proteomics is the interdisciplinary research field analyzing the composition, interaction and impacts of the proteome (the entirety of proteins) of single cells or up to a whole organism [9, 14]. It has grown in importance to the scientific community over the last decade due to new and refined methods enabling large-scale data acquisition, which resulted i.a. in the publication of the first human proteome drafts in 2014 [15, 11]. Meanwhile, methods of equally high quality are often missing for the study of many interactions with other (bio-)molecules. In this thesis, research was done to fit one piece of the data acquisition pipeline, the well-established Percolator score-calibration algorithm [8], onto the characteristics of protein-RNA interaction. It will focus on the research to this field based on the study of proteins cross-linked onto RNA.
The scientific background is covered in Section 1.1. Chapters 2 and 3 describe the changes done to the Percolator algorithm and their impact, which are discussed in Chapter 4. An outlook for further possible improvements and experiments in Section 4.1 concludes this thesis.

1.1 Background

We will now briefly review the relevant scientific background for this thesis. Sections 1.1.1 and 1.1.2 cover the chemical and technological background of

data-generation. The methods for processing the data and error rate estimation are explained in Section 1.1.3, the Percolator algorithm in Section 1.1.4, and methods of measuring the performance of classifiers are described in Section 1.1.5.

1.1.1 Cross-linking

Cross-linking is the process of artificially inducing a chemical bond between different molecules, using for example UV light [14]. Applying stable chemical bonds between peptides and RNA that interact transiently *in vivo*, allows a better understanding of their interaction.

1.1.2 Mass Spectrometry

For quantitatively characterizing the proteome of a sample, large scale measuring techniques are needed. Mostly, mass spectrometry (MS) is used, or more specifically, as for the data in this thesis, tandem mass spectrometry (MS/MS), which is often combined with techniques to separate the peptides like liquid chromatography (LC). In order to analyze the protein sample with MS, its complexity has to be reduced as much as possible, for example using LC [14]. As Han et al. [9] explain, the mass spectrometer then produces mass spectra, which can be analyzed further. First, the substrate is ionized, because only charged molecules can be detected by the mass spectrometer. Then, the sample is separated in the mass analyzer by the ratio $\frac{m}{z}$, mass of the particles to their charge. The detector then quantifies the amount of ions in the sample. The result is a mass spectrum, as shown in Figure 1.1.
Because mass alone does not give enough information about a peptide to determine its sequence, tandem mass spectrometry is often used to gather more detailed evidence. In this procedure, particles of similar $\frac{m}{z}$ ratio are selected for fragmentation in a collision cell after the first round of mass measurement [14]. In the collision cell, the substance collides with a gas to be broken down into smaller fragments. For peptide ions, fragmentation happens predominantly in their backbone, producing all possible pre- and suffixes of the peptide. This yields a spectrum that is almost unique for each peptide, which allows for its identification using tools of bioinformatics [2].

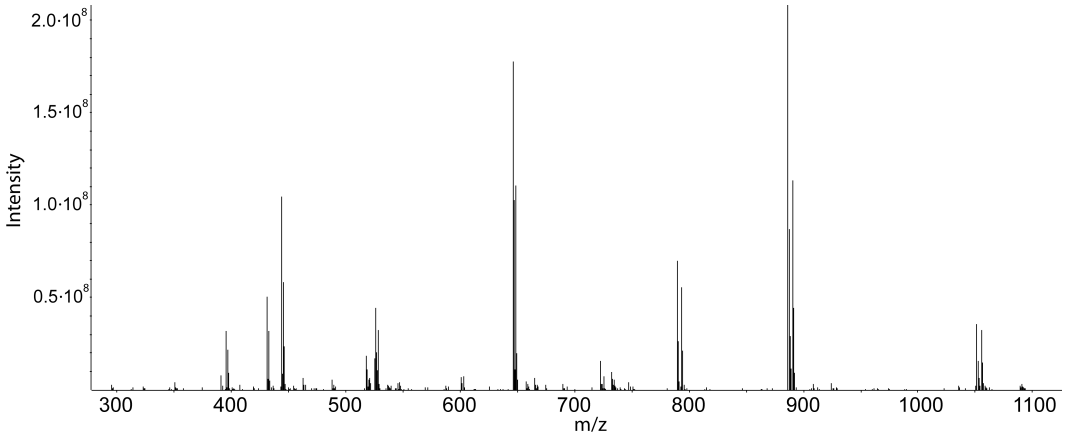


Figure 1.1: Mass spectrum as recorded by a mass spectrometer. The ion intensity correlates with the amount of a molecule in the sample, $\frac{m}{z}$ is the mass-to-charge-ratio. From: Sachsenberg [14]

1.1.3 Tools of Bioinformatics

Algorithms like Sequest [6] or X! Tandem [5] compare the resulting spectra with theoretical spectra calculated from a list of possible peptides and compute a score based on their similarity. The peptides are generated by obtaining a list of proteins expected in the sample and calculating the peptides resulting from the, for example enzyme-based, degradation of the proteins. The best scoring peptide is then considered a peptide-spectrum-match (PSM). The scores produced by those algorithms often do not distinguish well enough between correct and incorrect matches [10], but they enable false discovery rate (FDR) estimation using decoy databases and serve as a basis for score re-calibration with the Percolator algorithm [10, 8].
Decoy databases are created from the target database, and contain usually as many peptides [13, 12] in a reversed or shuffled order with respect to the amino acid sequence [1]. They are presented to the scoring algorithm either separately [8] or mixed with the target database [13]. It is assumed, that decoy and target peptides have similar features [12] (like the identical mass) and random matches are scored equally well. When the correct peptide for a given spectrum is not in the target database, and thus a wrong one will be chosen, the best scoring peptide will be a decoy approximately half of the time. This allows for an estimation of wrongly assigned targets, since the score distribution is assumed to be the same for decoys and false targets [1].
In practice, one estimates the confidence that a PSM is correct by counting the number of decoy PSMs with the same or a higher score. It is then assumed, there are as many false targets and thus a false discovery rate (FDR) can be

estimated [8]. This leads to the following formula[1]:

$$FDR = \frac{\#\text{ false target PSMs}}{\#\text{ all target PSMs}} \approx \frac{\#\text{ decoy PSMs}}{\#\text{ all target PSMs}} \tag{1.1}$$

The q-value as a property of an individual PSM rather than a set of PSMs is then derived from this as the minimum FDR of all PSMs with a lower or equal score [8, 1]. It represents the estimated rate of false targets if a PSM's score is chosen as threshold and can be seen as a confidence measurement for the PSM with respect to the whole dataset.

1.1.4 Percolator

As Käll et al. [10] describe, separating correct from incorrect target PSMs with already mentioned algorithms works fine, but there is still room for improvement. This is because only certain similarity aspects of the matched spectra are used, which leaves out information; also about the peptide and the measured spectrum itself. Percolator [10, 8] tries to utilize as much information as possible by using scores from different algorithms, features of the peptide (like its length), the spectrum, or the PSM itself. It joins them using a linear support vector machine (SVM) and a semi-supervised approach with cross-validation to retain as many PSMs as possible. In every iteration, the top ranking, non-decoy PSMs up to a certain threshold of q-value are chosen as positive training examples, and the decoy PSMs are used as negative training set. The PSMs are then re-ranked using the SVMs score with the intention of achieving a better separation of true and false PSMs at a given FDR threshold. If this holds true, the positive training set of the next iteration is of higher quality and the SVM can be trained even better. The algorithm usually converges within the first 10 iterations [10].
As explained by Bishop [3], an SVM is a maximum margin classifier model. In the linear case, it tries to generate a hyperplane inside the multi-dimensional space that splits the given training data into the specified classes. For generalization performance, it tries to maximize the distance or margin between the hyperplane and the nearest training points, called support vectors [3]. If the data is not linearly separable, one uses a special kind of SVM that allows but punishes outliers by their distance to the hyperplane: An outlier inside the margin but on the correct side of the hyperplane is not penalized as badly as if it was on the wrong side of the hyperplane, and the further away from

[1] In this thesis, the following approximation is used:
$FDR \approx \frac{\#\text{ decoy PSMs}}{\#\text{ all PSMs}} = \frac{\#\text{ decoy PSMs}}{\#\text{ decoy PSMs}+\#\text{ target PSMs}}$
It is faster to calculate and yields results differing by the FDR, so in the relevant range of FDRs of 0 to 5% up to 5%:
$\frac{\frac{\#\text{ decoys}}{\#\text{ targets}}}{\frac{\#\text{ decoys}}{\#\text{ decoys}+\#\text{ targets}}} = \frac{\#\text{ decoys}+\#\text{ targets}}{\#\text{ targets}} = 1 + \frac{\#\text{ decoys}}{\#\text{ targets}} \approx 1 + FDR$

the hyperplane a point lies on the wrong side, the more this gets punished [3]. The strength of the penalization is additionally controlled by the parameter C, which can also be defined differently for points of different classes[2].
To avoid having to split the data into training and testing set and consequently losing possibly correct PSMs but also avoid overfitting, a nested cross-validation approach is being used. As Granholm et al. [8] explain, overfitting is a central problem in machine learning. It happens when the model fits itself onto random variation in the data, not actually connected to a general pattern and performing poorly when presented new data. Cross-validation means splitting the data into a number of smaller batches (Percolator uses 3), presenting all but one to the model as training and using the remaining part to validate the results, or, in the case of Percolator, to score the PSMs. For every part of the data a new model is trained by every part but the one to be scored, and the scores of different parts are normalized in the end. Additionally, Percolator performs cross-validation with every training batch, evaluating different parameters for the machine learning model applied to this specific batch of data [8]. A diagram explaining this approach can be found in Figure 2.1.

1.1.5 ROC Curves

As explained in depth by Fawcett [7], performance of classification machine learning models is often evaluated using receiver operating characteristics (ROC) curves. They are charts in which the true positive rate of a model on the y-axis is plotted against its false positive rate on the x-axis. An example is shown in Figure 1.2. Generally, a classifier producing a steeper ROC curve, meaning it lies further to the northwest than the curve of another classifier, is better, because it achieves a higher true positive rate at a lower false positive cost. But, to be able to compare classifiers more directly and by using a single scalar value, often the area under an ROC curve is computed and used as a metric, because it represents the probability that "the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance" ([7], page 868).
When dealing with the problem of rating PSMs however, there is no way of knowing the ground truth and thus precisely estimating a classifiers performance. Instead, the total number of PSMs that are estimated to be correct when accepting a certain q-value are plotted against that q-value [10, 8]. The graphs shape resembles that of an ROC curve and the property that curves further to the northwest represent better classification performance is maintained, so the plot is called pseudo ROC. However, there are significant

[2]https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

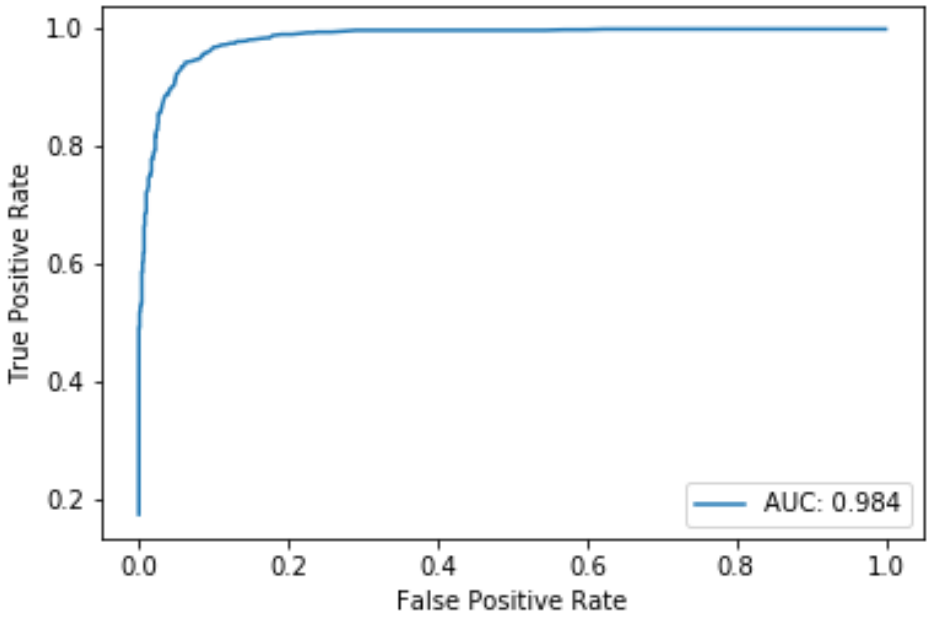
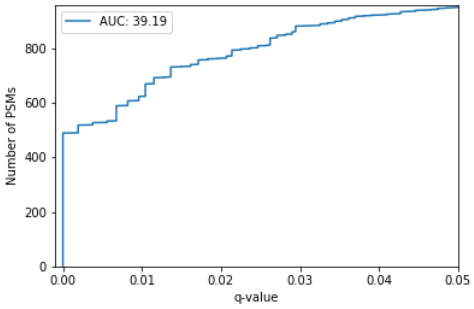
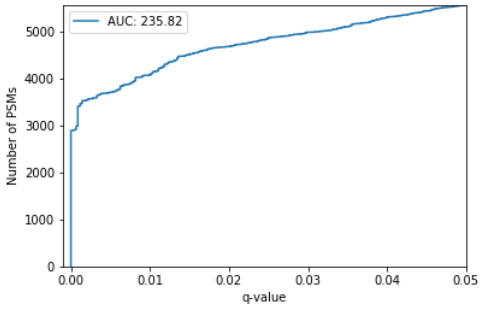


Figure 1.2: An Example for an ROC curve. It was plotted from a generated dataset with 1000 positive and negative entries respectively, being "scored" using a log normal distribution with different expected values. Calculations were done by the `pseudoROC` function as described in Section 2.1.



(a) This pseudo ROC was generated from the same generated dataset as was used for the ROC curve in Figure 1.2.



(b) This pseudo ROC was generated from the realistic dataset described in Chapter 2, using the given NuXL-score.

Figure 1.3: Pseudo ROC curves, plotted using the function `pseudoROC` as described in Section 2.1.

differences and neither the number of accepted PSMs nor the q-value are the same as their ROC counterparts: The q-value of a certain PSM, being related to the FDR, is the (estimated) portion of wrongly accepted PSMs out of every positively classified PSM, when using the PSMs score as threshold. The false positive rate on the other hand would be the percentage of wrongly accepted PSMs out of every wrong PSM in the whole dataset. With Y being the number of PSMs accepted by the classifier and n the number of false entries according to the ground truth, as in Fawcett [7], this can be written as:

$$FDR = \frac{FP}{Y}; \quad FPR = \frac{FP}{n} \tag{1.2}$$

With this notation, the true positive rate (TPR) is $\frac{TP}{p}$ and the total number of accepted PSMs is Y. From this, a completely different scale arises (since $Y \in \mathbb{N}$ and TPR $\in [0, 1]$). The AUC, which is still used as a metric, thus does not range from 0 to 1 and does not represent a probability, but can only be used as a relative score.

ROC curves improve when the data contains more examples with features clearly indicating the negative class membership to the classifier, because the FPR of all other examples decreases. Since pseudo ROCs are plotted only for very low fractions of false discoveries, often $[0, 0.05]$, they are insensitive to such changes in the data. This allows including lower-scoring PSMs in the dataset (like is done in the dataset used this thesis) without diluting the evaluation. Examples for pseudo ROCs are shown in Figure 1.3.

Chapter 2

Material and Methods

To be able to test the following implementations, a dataset containing 93,219 PSMs was used. Its spectra result from UV-light cross-linked extract from the nucleus of HeLa cells, and were matched against the nucleus proteins. For each spectrum, up to 4 matching peptides are recorded. The dataset was generated with OpenNuXL 1.0.
Of the PSMs, 51,521 are cross-linked and 41,698 linear peptides, 50,627 are target and 42,592 are decoy peptides. Every PSM has 52 feature columns, consisting of subscores from different algorithms as described in Section 1.1.3, properties of the peptide, the spectrum or the cross-linked nucleotide. Also 2 columns specifying the spectrum and 2 columns describing the peptide and the protein it originated from, as well as one column indicating whether an entry is a target or a decoy PSM. The features contain the NuXL-score, which was computed from different subscores and used as a reference.

2.1 Implementation of the Percolator algorithm

The Percolator algorithm as presented in Figure 2.1 was implemented in python. The pandas[1] package was used for dataset handling, and for SVM calculations and inner cross-validation the classes LinearSVC[2] and GridSearchCV[3] from the python scikit-learn package were used. The outer cross-validation was implemented utilizing the numpy `array_split`[4] method. An early stopping criterion was implemented, terminating the algorithm if the

[1] https://pandas.pydata.org/
[2] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[4] https://numpy.org/doc/stable/reference/generated/numpy.array_split.html

Figure 2.1: Rough outline of the Percolator algorithm. First, the dataset is split in three parts, of which two are used for training and one for scoring. Then, another cross-validation is performed on the training part to determine the best parameter combination. Percolator searches the grid spanned by the options for the C parameter for positive examples C^+ (0.1, 1 and 10) and negative examples C^- (1, 3 or $10 \times C^+$). The parameter combination achieving best results on the training set is then used to train an SVM that scores the validation set. For training, all target PSMs with a q-value of $\leq 5\%$ are used as positive and all decoy PSMs are used as negative examples. Because in the end, the dataset will have been scored by three different SVMs, these scores have to be normalized. This is done by a linear transformation mapping the score representing a q-value of 1% to 0 and the median decoy score to -1. If `qScore` denotes the minimum score for which a q-value of 1% is estimated and `dMedian` the median score a decoy achieves in the validation set, this sets score is transformed by the following equation: new Score $= \frac{\text{old score} - \texttt{qScore}}{\texttt{qScore} - \texttt{dMedian}}$.

area under the pseudo ROC curve does not improve over a certain number of iterations. This number is controlled by the parameter `termWorseIters` and has a default of 4.
As preprocessing, the provided file is read in as a pandas dataframe (converting strings to floating point or integer values if possible) and q-value and ranks are calculated as discussed in Subsections 1.1.3 and 2.2.1 respectively. Also, the columns containing features are normalized. This means, their value is mapped linearly onto $[0, 1]$ to avoid rounding errors in features of different orders of magnitude, caused by the representation of floating point numbers.
For the calculation of pseudo ROC curves and the area under this curve, a function `pseudoROC` was implemented. It retrieves the entries of the given datasets q-value column, enumerates them and plots the enumeration against the q-values. This results in a pseudo ROC curve. Plotting was performed using the matplotlib.pyplot[5] package, and the area under the pseudo ROC curve was calculated by the scikit-learn auc[6] method.
To distinguish the original Percolator algorithm from the version implemented in python and altered here, the latter will be called Pycolator.

2.2 Adapting Percolator to Cross-link Identification

To be able to monitor the performance of the model in different experiments, especially with respect to the cross-linked or non-cross-linked PSMs, following features were implemented:
First, in addition to the q-value, which is calculated as described in Section 1.1, the calculation of a class-specific q-value was implemented. This is done by splitting the dataset according to the class affiliation and calculating the q-value separately for both splits. This allows a more precise performance estimation when considering only one of the classes.

Second, an ROC curve using the `pseudoROC` function (see Section 2.1) is plotted after every iteration of Pycolator: one for the whole dataset, one only for cross-linked, and one only for non-cross-linked PSMs. Accordingly, the respective class-specific q-value is used. Thus, three plots covering the corresponding class(es) and every iteration, as well as the area under the curve are returned. This allows for fast visual detection of the impact a specific change to the algorithm has on certain classes, iterations or general sensitivity.

[5] https://matplotlib.org/api/pyplot_api.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html

2.2.1 Different Ranks

As experience shows, cross-linked peptides can be harder to detect than linear peptides, because of lower signal intensities attributed to a low yield of cross-linked peptides. This means, the possibly correct cross-linked peptide will frequently not get the highest score of all the peptides. It thus can be beneficial to not only consider the highest scoring peptide, but also the highest scoring cross-linked peptide or also some lower-scoring peptides and assign them ranks. Then, as following experiments show, Pycolator can correct the scores of some lower-ranking PSMs, possibly detecting more cross-linked PSMs. For sake of simplicity, it is assumed that only one of the peptides can be the correct match to a spectrum, and thus any PSM with a lower rank than 1 should be excluded at the end.
To tackle both constraints, Pycolator first trains the SVM with every PSM available and re-calculates the ranks based upon the newly assigned score after every iteration, possibly correcting the ranks of some PSMs. When the used metric, normally the area under the pseudo ROC curve, does not improve beyond a certain threshold per iteration, every PSM with rank > 1 is dropped. The threshold is controlled by the parameter `cutOffImprove` with a default of 0.01 corresponding to a 1% increase of the used metric per iteration. However, some of the PSMs that are not assigned rank 1 and are dropped in this procedure achieve a high score and are used to train the SVM. In their place, other PSMs can reach a certain q-value threshold. Thus, the algorithm runs further until the maximum iterations are reached or the score does not improve further, in order to properly integrate and score the new PSMs considered confident. In this thesis, the feature will be called optimalRanking, after the parameter it can be toggled by. Its procedure is depicted in Figure 2.2.
The performance of this feature was tested against dropping the lower ranking PSMs once at the very end of the algorithm and once at the very beginning. Pycolator was run on the given dataset (see Chapter 2) and pseudo ROCs were plotted as described in Section 2.2.

2.2.2 Characteristics of Cross-linked PSMs

Apart from being hard to detect, cross-linked peptides also have other characteristics, some of which pose problems to the computational detection of correct PSMs. As discussed in Subsection 2.2.2 c), the features of cross-linked and linear peptides are so dissimilar, splitting the dataset and training a linear SVM separately on cross-linked and non-cross-linked PSMs yields significantly better results than training one linear SVM. To reduce the impact of this heterogeneity, the following experiments were conducted:

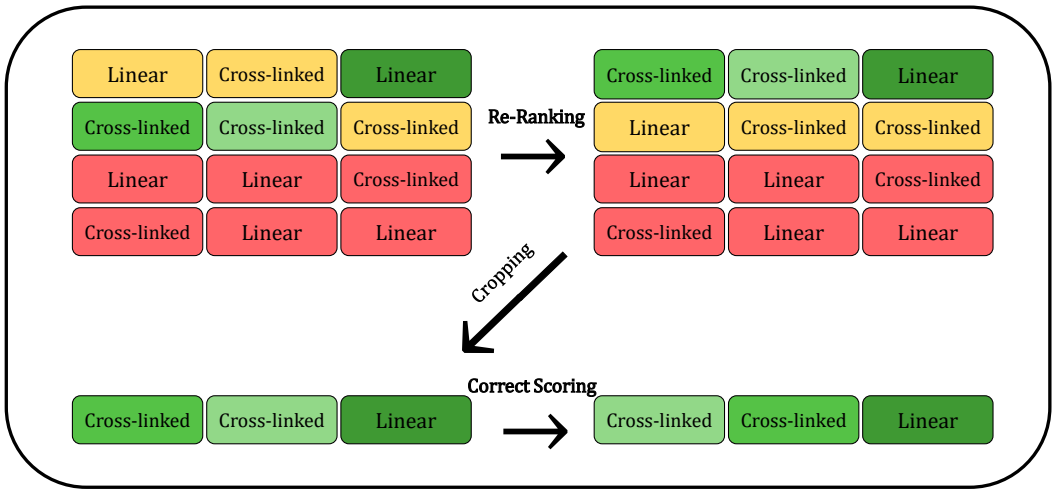


Figure 2.2: Scheme for the OptimalRanking method employed by Pycolator. Different PSMs of the same spectrum are represented as columns and the true quality via their color. This is not known in practice. In the first iterations, a re-ranking happens, assigning a rank of 1 to the best PSMs, in one case identifying a cross-linked instead of a linear peptide. Then, the worse ranks are cut and Pycolator keeps training with the remaining PSMs to properly score them.

a) Proportions of Different Classes

As discussed in Subsection 1.1.4, Percolator employs a nested cross-validation approach: It first splits the dataset in three, then trains on two parts and scores the remaining part. If the splitting was uneven by chance, the SVM would be trained on class proportions that do not reflect the underlying data. Having different classes with significant differences in the dataset, like in our case cross-linked and linear PSMs or targets and decoys, increases this problem. For example, if there was a testing split with many cross-linked PSMs, the SVM had to be trained on the remaining data with few cross-linked PSMs, resulting in probably poor scoring of the many cross-linked PSMs in the test set. In the average case, this should not be a problem, but it can occasionally produce worse or better results, which would follow from overfitting.
To solve this problem, a mechanism of maintaining the proportion of the classes in the whole dataset for every inner and outer split was implemented. It can be toggled for targets and decoys or cross-linked and non-cross-linked PSMs as well as for inner and outer split independently. The impact was measured by running the algorithm 10 times, plotting and recording the results of the best, worst and median run w.r.t. to the area under the pseudo ROC curve for all PSMs.

b) Imputation

Cross-linked PSMs naturally have features linear PSMs do not have, which can however be used for training the SVM. An example would be the nucleotide it was linked to, or the partial loss score, which is the X! Tandem [5] hyper score for the cross-linked peaks. In the given dataset (see Chapter 2), 16 out of 61 features were only given for cross-linked PSMs. Optimally, these should not influence the score a non-cross-linked PSM gets. However, 0 was filled in for the missing values and because that is a valid value for the linear SVM, it biases the decision made. For example, if a high value in a feature leads the SVM to a decision against the PSM, 0 as the lowest value possible after feature normalization (see Section 2.1) will tell the SVM to give the PSM a higher score. The procedure of replacing missing values with numbers minimizing the bias is called imputation. The scikit-learn package provides methods for the implementation in python[7], and one of these, the `IterativeImputer` function, was tested.

c) Splitting the Dataset

Because the SVM is linear, one feature cannot alter the influence another feature has on the decision. Thus, splitting the dataset into cross-linked and non-cross-linked PSMs should allow the SVM to fit more precisely onto the characteristics of each. This was tested by running Pycolator on each split and then concatenate the datasets, recalculating the q-values from the Pycolator scores. It was also tested to split the dataset by the nucleotide the peptide was cross-linked onto and non-cross-linked PSMs as a fifth class.

2.2.3 Small datasets

The portion of cross-linked PSMs in a dataset is often very small. To evaluate when splitting as proposed in Subsection 2.2.2 c) is possible and obtain general insight into the scalability of Pycolator, two experiments were conducted to conclude how small a dataset may be, so that the Percolator algorithm still works. In each case, Pycolator was applied to a dataset sampled from the given one (see Chapter 2). The area under the pseudo ROC curve and the number of PSMs identified at 1% q-value were recorded and used as metrics. The results achieved by Pycolator were compared to the effect the splitting had on the q-value estimation utilizing the given, precomputed NuXL-score.
First, the smaller dataset was sampled using every 2^i-th PSM from the whole dataset with $i \in \mathbb{N} \wedge i \in [0, 13]$. Second, the smaller dataset was sampled randomly using a uniform distribution from all PSMs with a q-value of $< 10\%$.

[7] https://scikit-learn.org/stable/modules/impute.html

The portions sampled were $\frac{1}{2^i} : i \in \mathbb{N} \wedge i \in [0, 12]$. In addition to the metrics mentioned above, also the expected number of identifications at 1% FDR for each split were calculated and recorded. This was done by counting the number of such PSMs before re-estimating the q-value or letting Pycolator run, thus being the number of identifications by the NuXL-score if the q-value is estimated based on the whole dataset. To compensate for the randomness in this experiment, Monte-Carlo-Sampling with 10 iterations was performed and the results are presented as boxplots. Because Pycolator re-ranks PSMs and then drops the ones with a lower rank than 1, the dataset sizes are not equal in every of the 10 iterations and differ from the sizes when Pycolator is not run. Only one resulting size of each split is drawn as x-labels, and to still be able to compare the results after Pycolator is run and before, the metrics are used as proportion of dataset size.
Because numerical problems occur when calculating the area under a pseudo ROC curve of a very small dataset (since the curve can consist of only one point), another metric was introduced: the number of identified PSMs at an FDR threshold of 1%. Per default, Pycolator decides automatically after its first iteration if this metric is required based on the dataset, and adjusts its log, plots per iteration and functions like the early stopping criterion accordingly.

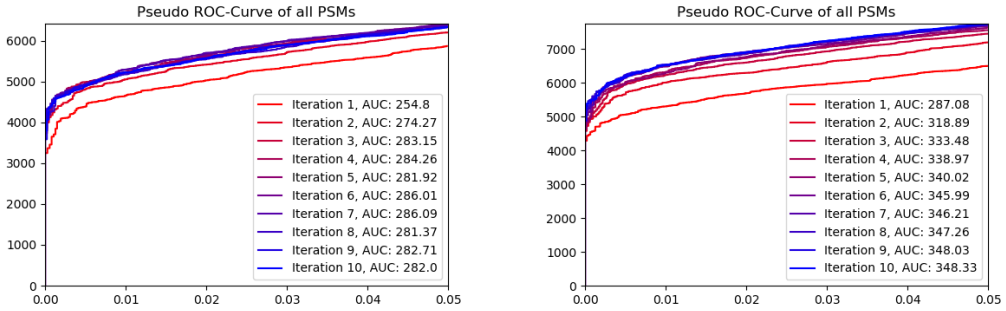
Chapter 3

Results

3.1 Implementation of the Percolator algorithm

The original Percolator algorithm (version 3.2) finds 5412 cross-linked PSMs with a q-value of $\leq$ 1% on the used dataset (see Chapter 2). The re-implementation in python yields 5450 such PSMs on rank 1.

The implementation of feature normalization yielded an increase in the number of confident PSMs found, as the curves and legend in Figure 3.1 show. This type of plot is explained in Section 3.2. The performance of Pycolator with feature normalization is better from the first iteration on, and increases until iterations 7 or 8, when the results converge. Without feature normalization, the area under the pseudo ROC increases until iteration 3, when it begins to oscillate.



(a) **(b)**

Figure 3.1: Result of Pycolator without (3.1a) and with (3.1b) the implementation of feature normalization.

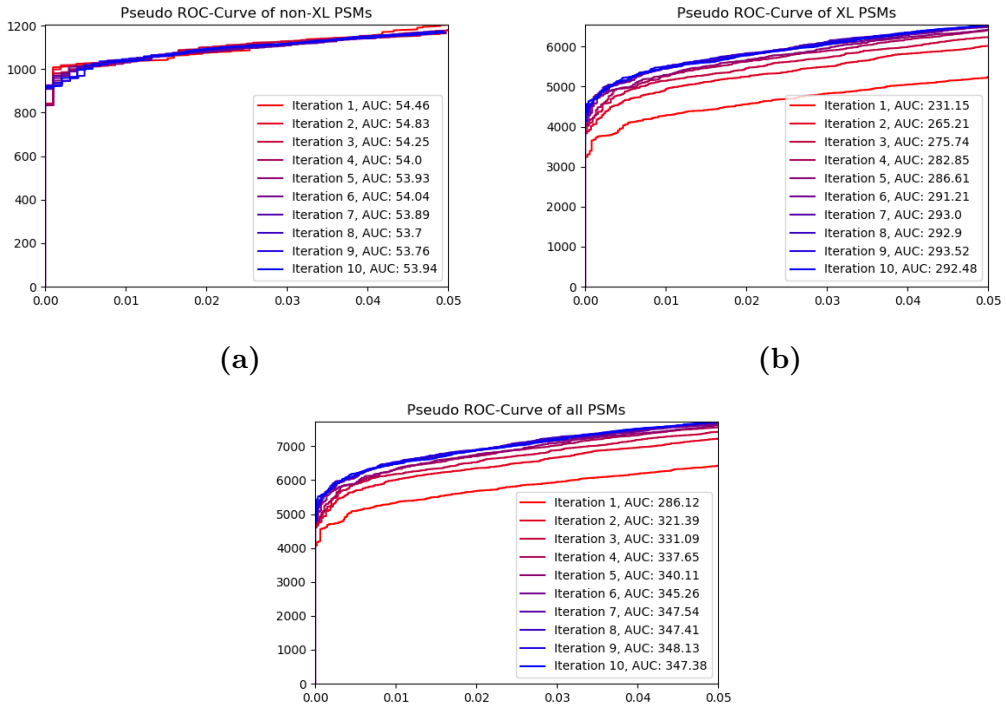


Figure 3.2: Example of the plots output by Pycolator, generated by running with default parameters on the given dataset (see Chapter 2). 3.2a shows the result of non-cross-linked PSMs, 3.2b that of cross-linked and 3.2c that of all PSMs in the dataset.

3.2 Adapting Percolator to Cross-link Identification

To evaluate the performance of the model, a method returning pseudo ROCs and their AUC after every iteration and for cross-links, non-cross-links and the whole dataset was implemented. An example of these plots are shown in Figure 3.2. 3.2a shows the pseudo ROC of the non-cross-linked PSMs in every iteration, 3.2b the cross-linked and 3.2c all of the PSMs. In each, every iterations pseudo ROC curve has its own color, fading from red in iteration 1 to blue in iteration 10. The colors can be assigned from the legend, which also shows the area under the depicted pseudo ROC curve. Every curve ranges from q-values 0 to 5% and the y-axis is scaled appropriate for the data. As explained in Subsection 1.1.5, a steeper and further to the northwest ROC curve represents better classification. Thus, Figure 3.2c shows that the result of Pycolator improves over most iterations.

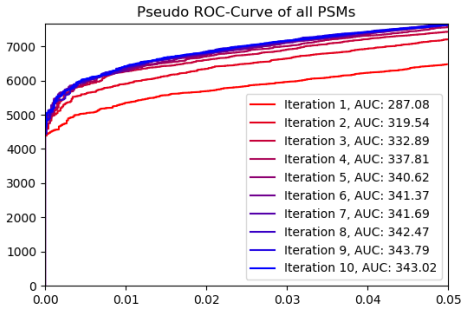
3.2.1 Different Ranks

Figure 3.3 shows the pseudo ROCs of Pycolator with and without the optimalRanking feature disabled. 3.3a was plotted when Pycolator used all PSMs available, meaning all ranks, and only at the end PSMs ranking lower than 1 were excluded. 3.3b is the result of running Pycolator only with rank 1 PSMs of the given dataset (see Chapter 2) and 3.3c shows the final result with the new feature.
As one can see, without the new mechanism Pycolator gradually improves the area under the curve and takes 5 and 4 iterations to roughly converge when given every PSM or only rank 1 PSMs to train respectively. This can be seen by the AUC results after every iteration or the ROC curves themselves. The end result, however, is better when removing the lower ranking PSMs right at the beginning (AUC of 345.79) instead of in the end (AUC of 343.21 after dropping the lower ranking PSMs). Since in the latter case Pycolator ended with an AUC of 343.79 as one can see in 3.3a, dropping the PSMs slightly worsened the end result. With the new feature, the algorithm takes about 5 iterations to converge, then performs a jump and again improves over two iterations. The end result, an AUC of 348.13, is higher than both alternatives. A t-test with the AUC results of 10 repetitions between the new procedure and providing Pycolator with all PSMs yielded a p-value of $2.77 \cdot 10^{-12}$ and between the new procedure and providing Pycolator with only rank one PSMs yielded $4.17 \cdot 10^{-9}$ as p-value.

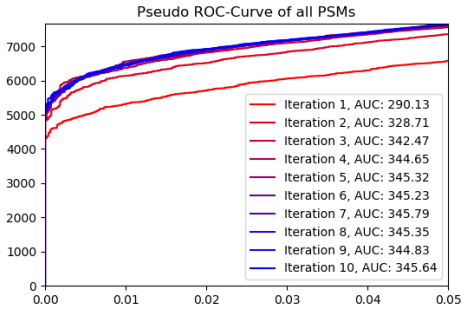
3.2.2 Characteristics of Cross-linked PSMs

a) Proportions of Different Classes

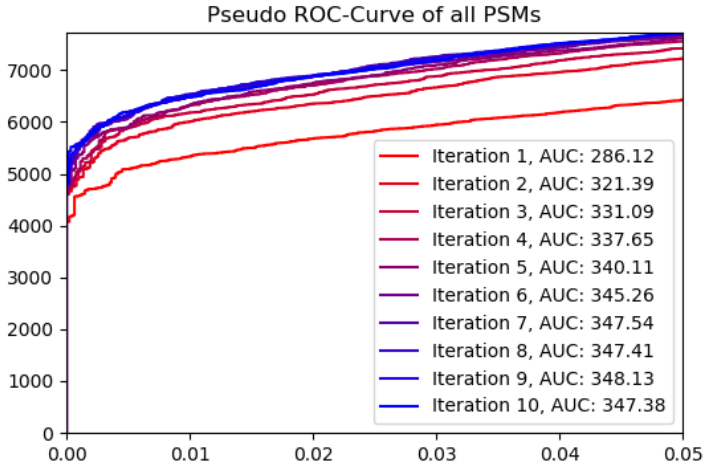
Maintaining the proportion of classes works as intended: The log of two test runs of Pycolator when it calculated and output the ratio of targets to decoys in every of the outer split and the whole dataset showed following results: Without maintaining the ratios of target to decoy PSMs in each of the three splits, these ranged from 1.173 to 1.203, while the ratio is 1.8865 in the whole dataset. With maintaining the ratio according to the new method however, these ranged from 1.18862 to 1.18870, while the ratio is 1.18865 in the whole dataset.
The evaluation produced the results shown in Table 3.1. The AUC of the best run is lower and the AUC of the worst run is higher for cross-linked and all PSMs when balancing the splits, meaning the variation could be reduced. The AUC of the median run is slightly higher for all categories, for all PSMs by 0.14%.



(a) The resulting pseudo ROCs (explained in Section 3.2) if Pycolator is given every PSM regardless of its rank. Lower ranking PSMs are only dropped in the end, which results in a final AUC of 343.21.



(b) The resulting pseudo ROCs (explained in Section 3.2) if Pycolator is given only the top scoring peptide for every spectrum. Lower ranking PSMs are dropped before running the algorithm.

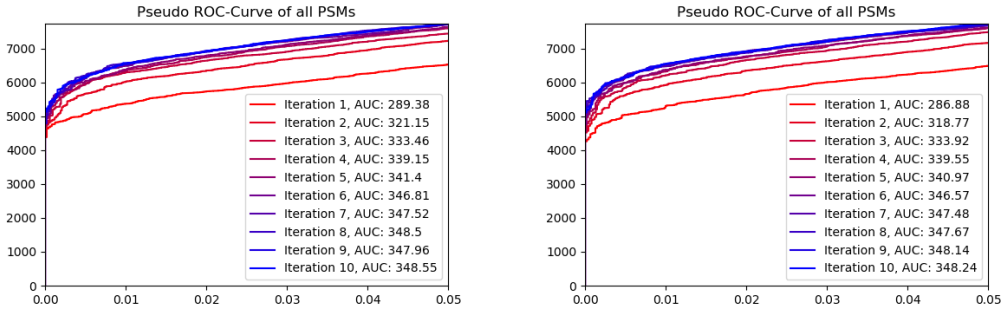


(c) The resulting pseudo ROCs (explained in Section 3.2) if Pycolator is run with the newly implemented feature. It first trains with every PSM available, and after the results converge, lower ranking PSMs are dropped. Then, the algorithm runs for some more iterations.

Figure 3.3: Results of running Pycolator with different strategies as to how lower ranking PSMs are dealt with.

Table 3.1: Shows the area under the pseudo curves of cross-linked, non-cross-linked and all PSMs in the best, worst and median run w.r.t. the AUC of the pseudo ROC of all PSMs. Pycolator was run with and without balancing of target/decoy and cross-linked/non-cross-linked classes in outer and inner splits 10 times each.

AUC of pseudo ROC		maximum	minimum	median
balancing	cross-linked	293.91	293.12	293.58
	non-cross-linked	53.85	53.76	53.84
	all PSMs	348.65	347.48	348.19
no balancing	cross-linked	294.42	291.51	292.86
	non-cross-linked	53.63	54.34	53.78
	all PSMs	349.14	347.04	347.69



(a) **(b)**

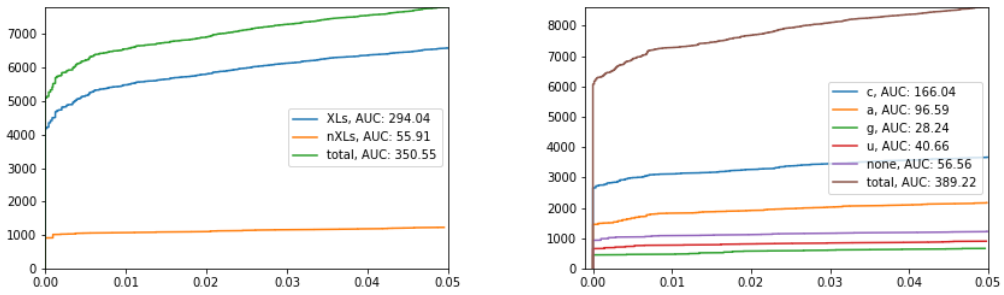
Figure 3.4: Results of running Pycolator with (3.4b) and without (3.4a) the use of imputation. Both are very similar.

b) Imputation

The resulting pseudo ROCs and their AUC for all PSMs are shown in Figure 3.4. The pseudo ROCs for cross-linked and not cross-linked PSMs show similarly few differences.

c) Splitting the Dataset

The results of splitting the dataset by cross-linked and non-cross-linked PSMs before running Pycolator are shown in Figure 3.5a. This yielded an AUC of 350.55, which is slightly higher than when running Pycolator on all PSMs (typically 348.5). Splitting by the cross-linked nucleotide (results depicted in Figure 3.5b) yielded an AUC of 389.22.



(a) **(b)**

Figure 3.5: Resulting pseudo ROCs and their AUC for each split and the combined dataset when splitting the dataset by 3.5a) cross-linked and non-cross-linked PSMs, 3.5b) the cross-linked nucleotide. XLs stands for cross-linked PSMs and nXLs for non-cross-linked PSMs.

3.2.3 Small datasets

When measuring the AUC in the first experiment, a roughly linear correlation to the datasets size was found. However, the AUC could not be measured when the dataset became smaller than approximately 200 PSMs, as shown in Figure 3.6. The number of PSMs at 1% FDR was larger relative to dataset size when using a smaller portion of the dataset, whether using a fixed (3.7a) or Pycolator's score (3.7b). In the case of Pycolator, with smaller dataset size also more fluctuations occurred.
The second experiment yielded following results: Figure 3.8a shows the portion of the dataset that was originally estimated to have a q-value of $< 1\%$. The ratio of how many PSMs were assigned such a q-value after the sampling to before, is shown in Figure 3.8b without re-calculating the score with Pycolator. Figure 3.9 shows how well the Pycolator score performed compared with the precomputed NuXL-score when measured by the identifications with 1% q-value or area under the pseudo ROC curve. The boxplots for the smallest datasets do not always contain 10 results, because Pycolator did, depending on the split, not work on datasets with 12 to 93 PSMs.

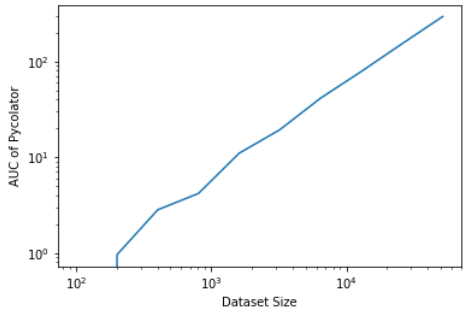
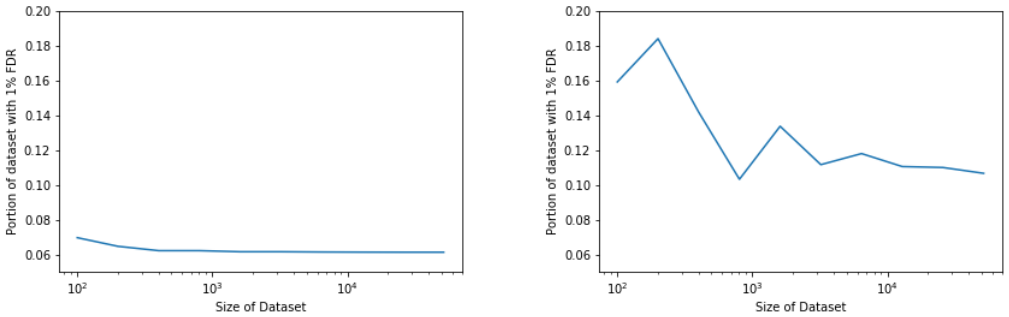
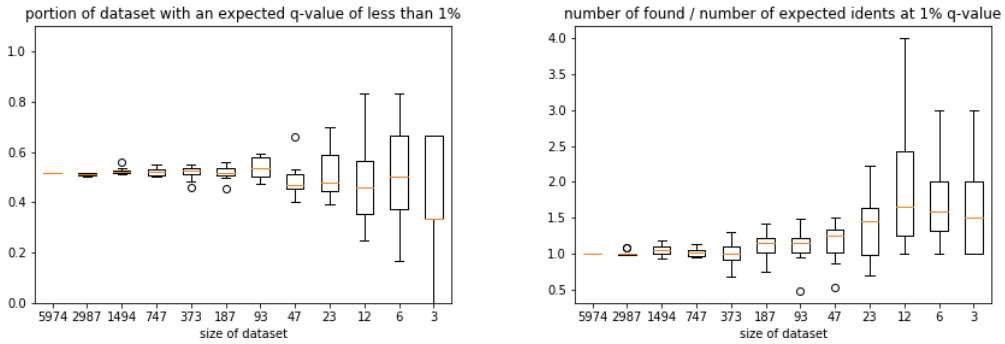


Figure 3.6: The AUC of a Pycolator run on the y-axis is plotted against the size of the dataset sampled as described in Subsection 2.2.3 on the x-axis. Below about 200 PSMs in the dataset, the pseudo ROC consisted of data at one q-value and thus enclosed an area of 0 or could not be determined.



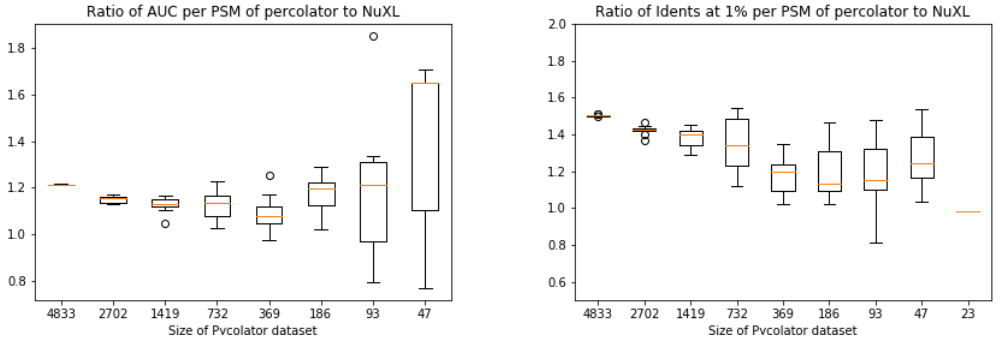
(a) **(b)**

Figure 3.7: The portion of PSMs in the sampled dataset with an estimated FDR of 1% on the y-axis is plotted against the datasets size on the x-axis. Left (3.7a) when estimating the FDR based on the unchanged NuXL-score, and right (3.7b) after running Pycolator on the sampled dataset.



(a) . **(b)**

Figure 3.8: a) Proportion of high confidence PSMs (q-value $\leq$ 1%) in the subsampled dataset from all PSMs with 10% FDR is plotted on the y-axis. b) The ratio of high confidence PSMs found after to before re-calculating the FDR in the sampled dataset is plotted. In both cases, the x-axis shows the datasets size, and the data from every of the 10 repetitions is shown combined in a boxplot.



(a) **(b)**

Figure 3.9: The ratio of Pycolators performance to the unchanged NuXL-score on the y-axis is plotted against the datasets size on the x-axis. Left (a), to measure the performance, the area enclosed by the pseudo ROC is used, right (b) the number of PSMs with a q-value of $\leq$ 1%. The data from every of the 10 repetitions is shown combined in a boxplot. While the median performance ratio measured by the AUC increases for datasets smaller than about 200 PSMs, Pycolator identifies less PSMs with a confidence of 1% the smaller the dataset is.

Chapter 4

Discussion

We reimplemented the Percolator algorithm in python and showed that it produced similar results as the original implementation. We observed a small increase in the number of found cross-linked PSMs (38 or 0.7%), which is likely due to stochastic differences during splitting the datasets and subtle differences in FDR calculations, as pointed out in Subsection 1.1.3.

Matching cross-linked peptides is often harder than linear ones because of lower signal intensities attributed to a low yield of cross-linked peptides. This means, the possibly correct cross-linked peptide will frequently not get the highest score of all the peptides. It thus can be beneficial to not only consider the highest scoring peptide, but also the highest scoring cross-linked peptide or also some lower-scoring peptides and assign them ranks. The Percolator algorithm may be able to revise the PSMs scores and the actually correct cross-linked peptide may become the best scoring PSM. This hypothesis is also supported by the findings in Subsection 3.2.1. If only including the best scoring peptide, of which Figure 3.3b shows the results, the end result is worse than when giving Pycolator some iterations to re-rank the found PSMs. However, it was better than when giving Pycolator all of the PSMs available, which is unexpected, since giving a machine learning model more information should generally improve its learning. Apparently, the lower ranking PSMs, even when having such a high score that a q-value of $\leq 5\%$ is estimated, contain misleading information and thus the SVM learns patterns not valid for correct PSMs. This also explains why the algorithm converges faster when only given rank 1 PSMs. The higher quality of data lets the SVM learn the correct patterns after fewer iterations.
The pseudo ROC generated when using the newly implemented mechanism (3.3c) shows a convergence after 5 iterations, just like when using every PSM (3.3a). Lower ranking PSMs are dropped and the next iteration has a much higher AUC, probably as a result of the better quality of the PSMs. Letting the algorithm run on the new dataset again im-

proves the AUC even beyond that of Pycolator when only using rank 1 PSMs. This suggests that indeed a re-ranking takes place in the first half of iterations.

Although implementing the balancing of different classes in the nested cross-validation splits had no big impact, it reduced the variation between the individual models. On other datasets, containing much fewer cross-linked than non-cross-linked PSMs, this procedure could have greater impact. This needs to be tested in further experiments. The feature has no disadvantages except negligible worse running time and can be generalized to other classes that could be present in a dataset.

Using imputation did not improve the performance of Pycolator. Neither for a single class nor for both. It is thinkable that imputation has a different effect on other datasets, which contain far fewer cross-links than non-cross-links. It could then prevent that the SVM fits onto the characteristics of non-cross-links because they are more frequent and find very few cross-links, but rather fit onto both classes in the same way. This needs to be tested in further experiments. Because the imputation takes a significant amount of time, this option is switched off per default.

Splitting the dataset by cross-linked and non-cross-linked PSMs slightly improved the performance of Pycolator, and splitting by the cross-linked nucleotide yielded a great improvement. In practice, neither is applicable to most datasets, because they often only contain a very small number of cross-linked PSMs. And, as the experiments in Section 3.2.3 show, applying Pycolator on a small dataset brings a lot of variation and worsens the performance.
The non-random approach in the first experiment regarding small datasets keeps the ratio of correct to incorrect PSMs and ensures the appearance of PSMs originally classified as correct. However, maintaining the ratio when sampling a very small subset means it will contain one or two PSMs, which originally were assigned a q-value of $< 1\%$, and many worse PSMs. This general bad quality of the dataset makes it difficult to obtain quantitative results for the very small datasets, which is why the second experiment, where we sampled randomly from the PSMs with 10% FDR, was conducted. The findings however do suggest that down to a dataset size of about 500, Pycolator does not suffer from any drawbacks. With smaller datasets, the AUC metric does not work anymore and a lot of variation happens (3.7b). Only the first problem could be solved: by implementing the new metric of identifications at 1% q-value. Figure 3.7a was generated to test if the FDR estimation would suffer from small datasets. This cannot be observed, as the smallest dataset for which a point is plotted has approximately 100 PSMs, and thus the small increase from 0.061 to 0.07 is due to rounding.

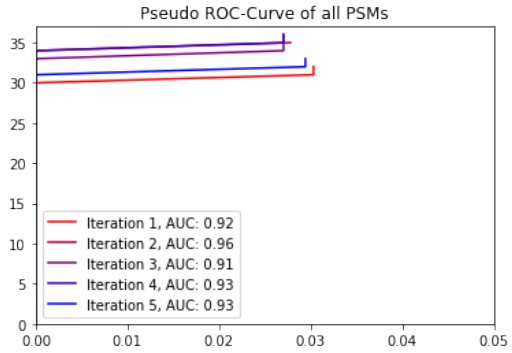


Figure 4.1: The pseudo ROC curve for a Pycolator run with a small dataset. Only one decoy is included in the PSMs that count into this figure: Every PSMs with a higher score than this decoy is plotted at 0% q-value, and every PSMs with a score equal or worse to the second decoy has a q-value of $\geq 5\%$ and is thus not plotted.

In the next experiment however, this hypothesis seems to be supported by Figure 3.8b, showing that when re-calculating the q-value after sampling, without changing the score, more PSMs receive a high confidence. But this is likely since few high-scoring decoys determine the q-value of many high-scoring targets. When one samples randomly, the chance that many of those decoys will not be sampled increases with a smaller sampling size, enhancing the FDR estimation of the targets, which usually lie above 1% q-value.
As Figure 3.9 shows, with the whole dataset the Pycolator score achieves a better separation of true and false PSMs than the NuXL-score. However, the smaller the given dataset gets, the smaller this advantage becomes. Rarely, Pycolator even performs worse, as depicted in Figure 3.9b. In Figure 3.9a the ratio rises for datasets smaller than approximately 200 PSMs. This is likely because in small datasets, one or two decoys determine which PSMs receive a q-value of $\leq 5\%$, which leads to pseudo ROCs like in Figure 4.1. For these, the AUC is very small, since the area is only calculated from 0 to the largest occurring q-value smaller than 5%, and not until 5%. Because Pycolator may find only a few more targets with a high score, these may be enough to allow another decoy in the top PSMs, in the case of Figure 4.1 almost doubling the range of which the AUC is calculated.

A certain threshold of dataset size, for which the Percolator algorithm gets much worse, could not be found. It gradually finds less high-confidence PSMs when the dataset size gets smaller. It is however crucial for SVM training, that there is always at least one decoy and one target PSM in every of the inner splits. This is why the algorithm returned with an error after 12 to 93 PSMs.

The new metric for small datasets, the number of PSMs at 1% q-value,

will likely not often be used, because the AUC is only not available for extremely small datasets, when Pycolator does not perform well anyways. Additionally, if the problem with the AUC on small datasets is fixed and it will always be calculated from 0 to 5%, this might be a more reliable evaluation method for datasets it works on. While the number of PSMs at 1% q-value might exhibit fluctuation, since the score of one decoy might make the difference if many targets obtain a q-value below or above 1%, this is not the case in datasets this metric is intended for. In these, a target either has a q-value of 0 or above 5% and is ranked higher than any decoy. While this ignores the slight inaccuracy this method of FDR estimation has (since estimating a q-value of 0% for the target PSM directly ranked above the first decoy, but $\geq$ 5% for the one below seems unrealistic), it is not possible to achieve a more accurate estimation for small datasets with this method.

To prevent an overfitting to one spectrum or one peptide, experiments prohibiting the distribution of PSMs stemming from one spectrum or one peptide on training and validation splits were conducted. Because they yielded no usable results, they are not shown in detail in this thesis. For both, the results did not change, suggesting that neither PSMs with the same spectrum nor with the same peptide are significantly more similar than unrelated PSMs.

This thesis produced a working version of Percolator in python. As Fondrie and Noble, who only recently also transferred Percolator to python and called it mokapot[1], note, this can be used to add flexibility to ones analyses. It is quite simple, for example, using another classifier than linear SVMs, since the machine learning methods from scikit-learn usually support the same arguments and attributes. Also, experiments like performing imputation are implemented conveniently with the scikit-learn library.
Although the experiments with smaller datasets brought no concrete findings, it confirmed the presumption that the Percolator algorithm works worse on smaller datasets.
Furthermore, two improvements were found to adapt Percolator to the characteristics of cross-linked PSMs: The ranking procedure and maintaining the proportions of different classes inside the splits. With these, Pycolator now finds 5518 cross-linked PSMs on the used dataset, as opposed to the re-implemented Percolator, which found 5450. A diagram of the algorithm with the added functionalities is shown in Figure 4.2.
The improved sensitivity of this algorithm regarding cross-linked peptides might enhance the results of future experiments researching the interaction between proteins and RNA. It could for example identify cross-linked peptides

[1] `https://github.com/wfondrie/mokapot`

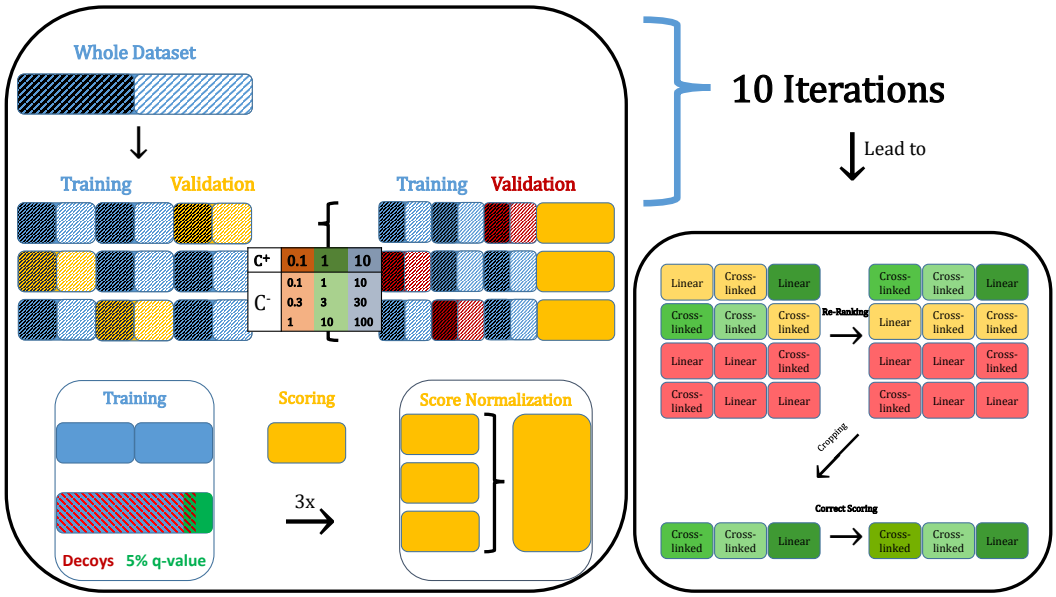


Figure 4.2: A diagram of Pycolator. Additionally to the Percolator algorithm shown in Figure 2.1, the feature maintaining the proportions of classes inside the dataset, depicted as black and white halves, and the procedure of re-ranking, cropping and correctly scoring the PSMs with rank 1 are shown.

that are not found by the original Percolator but play an important role in a biological process like post-transcriptional regulation.

4.1 Outlook

The AUC calculation is currently exact, which might lead to problems in datasets where there is no or one decoy below the q-value cutoff. In these cases, it might be beneficial to estimate the AUC for the whole range from 0 to the q-value threshold. This could for example be done by linearly interpolating between the two targets above and below the threshold.
Trying out more flexible machine learning models like histogram-based gradient boosting classification trees could yield an even better separation of true and false PSMs. More flexible models should achieve results at least as high as when splitting the dataset for Pycolator (see Figure 3.5b), since they can learn the different characteristics of each of the splits and classify depending on for example the nucleotide a peptide was cross-linked onto. However, care must be taken with more flexible machine learning models. First experiments indicate that these models could detect characteristics of targets as opposed to decoys rather than true PSMs, introducing a bias that renders the FDR estimation unreliable. Confirming this hypothesis however requires further research.
Since redundant features often tend to worsen linear models and proteomics

datasets often contain correlated scores, feature selection to remove redundancy might improve the models quality. Scikit-learn provides various methods for feature selection[2], which could be implemented into Pycolator in further experiments.
A method to increase model complexity in a controlled way is AdaBoost, where multiple weak models are combined into one strong one. This could improve the performance for linearly not separable data. Simultaneously, this only selects features that have a good impact on the predictive performance and thus reduces feature dimensionality. Scikit-learn also provides functions for this[3], which could be implemented conveniently in further experiments.

[2] https://scikit-learn.org/stable/modules/classes.html?highlight=feature%20selection#module-sklearn.feature_selection

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

Bibliography

[1] Suruchi Aggarwal and Amit Kumar Yadav. False discovery rate estimation in proteomics. In *Methods in Molecular Biology*, pages 119–128. Springer New York, 2016. doi: 10.1007/978-1-4939-3106-4_7. URL https://doi.org/10.1007/978-1-4939-3106-4_7.

[2] Thomas E. Angel, Uma K. Aryal, Shawna M. Hengel, Erin S. Baker, Ryan T. Kelly, Errol W. Robinson, and Richard D. Smith. Mass spectrometry-based proteomics: existing capabilities and future directions. *Chemical Society Reviews*, 41(10):3912, 2012. doi: 10.1039/c2cs15331a. URL https://doi.org/10.1039/c2cs15331a.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science + Business Media, LLC, 2006. ISBN 978-0387-31073-2.

[4] A. Burns and S. Iliffe. Alzheimer's disease. *BMJ*, 338(feb05 1):b158–b158, February 2009. doi: 10.1136/bmj.b158. URL https://doi.org/10.1136/bmj.b158.

[5] R. Craig and R. C. Beavis. TANDEM: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, February 2004. doi: 10.1093/bioinformatics/bth092. URL https://doi.org/10.1093/bioinformatics/bth092.

[6] Jimmy K. Eng, Ashley L. McCormack, and John R. Yates. 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*, 1994.

[7] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006. doi: 10.1016/j.patrec.2005.10.010. URL https://doi.org/10.1016/j.patrec.2005.10.010.

[8] Viktor Granholm, William Noble, and Lukas Käll. A cross-validation scheme for machine learning algorithms in shotgun proteomics. *BMC*

Bioinformatics, 13(Suppl 16):S3, 2012. doi: 10.1186/1471-2105-13-s16-s3. URL https://doi.org/10.1186/1471-2105-13-s16-s3.

[9] Xuemei Han, Aaron Aslanian, and John R Yates. Mass spectrometry for proteomics. *Current Opinion in Chemical Biology*, 12(5):483–490, October 2008. doi: 10.1016/j.cbpa.2008.07.024. URL https://doi.org/10.1016/j.cbpa.2008.07.024.

[10] Lukas Käll, Jesse D Canterbury, Jason Weston, William Stafford Noble, and Michael J MacCoss. Semi-supervised learning for peptide identification from shotgun proteomics datasets. *Nature Methods*, 4(11):923–925, October 2007. doi: 10.1038/nmeth1113. URL https://doi.org/10.1038/nmeth1113.

[11] Min-Sik Kim, Sneha M. Pinto, Derese Getnet, Raja Sekhar Nirujogi, Srikanth S. Manda, Raghothama Chaerkady, Anil K. Madugundu, Dhanashree S. Kelkar, Ruth Isserlin, Shobhit Jain, Joji K. Thomas, Babylakshmi Muthusamy, Pamela Leal-Rojas, Praveen Kumar, Nandini A. Sahasrabuddhe, Lavanya Balakrishnan, Jayshree Advani, Bijesh George, Santosh Renuse, Lakshmi Dhevi N. Selvan, Arun H. Patil, Vishalakshi Nanjappa, Aneesha Radhakrishnan, Samarjeet Prasad, Tejaswini Subbannayya, Rajesh Raju, Manish Kumar, Sreelakshmi K. Sreenivasamurthy, Arivusudar Marimuthu, Gajanan J. Sathe, Sandip Chavan, Keshava K. Datta, Yashwanth Subbannayya, Apeksha Sahu, Soujanya D. Yelamanchi, Savita Jayaram, Pavithra Rajagopalan, Jyoti Sharma, Krishna R. Murthy, Nazia Syed, Renu Goel, Aafaque A. Khan, Sartaj Ahmad, Gourav Dey, Keshav Mudgal, Aditi Chatterjee, Tai-Chung Huang, Jun Zhong, Xinyan Wu, Patrick G. Shaw, Donald Freed, Muhammad S. Zahari, Kanchan K. Mukherjee, Subramanian Shankar, Anita Mahadevan, Henry Lam, Christopher J. Mitchell, Susarla Krishna Shankar, Parthasarathy Satishchandra, John T. Schroeder, Ravi Sirdeshmukh, Anirban Maitra, Steven D. Leach, Charles G. Drake, Marc K. Halushka, T. S. Keshava Prasad, Ralph H. Hruban, Candace L. Kerr, Gary D. Bader, Christine A. Iacobuzio-Donahue, Harsha Gowda, and Akhilesh Pandey. A draft map of the human proteome. *Nature*, 509(7502):575–581, May 2014. doi: 10.1038/nature13302. URL https://doi.org/10.1038/nature13302.

[12] Roger E. Moore, Mary K. Young, and Terry D. Lee. Qscore: An algorithm for evaluating SEQUEST database search results. *Journal of the American Society for Mass Spectrometry*, 13(4):378–386, April 2002. doi: 10.1016/s1044-0305(02)00352-5. URL https://doi.org/10.1016/s1044-0305(02)00352-5.

[13] Junmin Peng, Joshua E. Elias, Carson C. Thoreen, Larry J. Licklider, and Steven P. Gygi. Evaluation of multidimensional chromatography coupled

with tandem mass spectrometry (LC/LC-MS/MS) for large-scale protein analysis: the yeast proteome. *Journal of Proteome Research*, 2(1):43–50, February 2003. doi: 10.1021/pr025556v. URL `https://doi.org/10.1021/pr025556v`.

[14] Timo Sachsenberg. Computational methods for mass spectrometry-based study of protein-RNA or protein-DNA complexes and quantitative metaproteomics. 2017. URL `http://hdl.handle.net/10900/83311`.

[15] Mathias Wilhelm, Judith Schlegl, Hannes Hahne, Amin Moghaddas Gholami, Marcus Lieberenz, Mikhail M. Savitski, Emanuel Ziegler, Lars Butzmann, Siegfried Gessulat, Harald Marx, Toby Mathieson, Simone Lemeer, Karsten Schnatbaum, Ulf Reimer, Holger Wenschuh, Martin Mollenhauer, Julia Slotta-Huspenina, Joos-Hendrik Boese, Marcus Bantscheff, Anja Gerstmair, Franz Faerber, and Bernhard Kuster. Mass-spectrometry-based draft of the human proteome. *Nature*, 509(7502):582–587, May 2014. doi: 10.1038/nature13319. URL `https://doi.org/10.1038/nature13319`.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift