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
Address
University of Tübingen

Paulitz, Emilian Nicolaus Simons:
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

Write here your abstract.

Acknowledgements

Write here your acknowledgements.

Contents

List of Figures **v**

List of Tables **vii**

List of Abbreviations **ix**

1 Introduction **1**

1.1 Background 1

1.1.1 Mass Spectrometry 1

1.1.2 Bioinformatics Tool 2

1.1.3 Percolator 3

1.1.4 ROC curves 4

2 Material and Methods **7**

2.1 Implementation of the percolator algorithm 7

2.2 Adapting Percolator to Cross-link Identification 9

2.2.1 Different Ranks 9

2.2.2 Characteristics of Cross-linked PSMs 10

2.2.3 Small datasets 11

3 Results **13**

3.1 Implementation of the percolator algorithm 13

3.2 Adapting Percolator to Cross-link Identification 14

3.2.1 Different Ranks 15

3.2.2 Characteristics of Cross-linked PSMs 15
3.2.3 Small datasets 18

4 Discussion and Outlook 19

Bibliography 21

List of Figures

1.1 Example for a mass spectrum 2
1.2 Example for an ROC curve 5
1.3 Examples for pseudo ROC curves 6

2.1 Procedure of Percolator 8

3.1 Results of implementing feature normalization 13
3.2 Examples for pseudo ROC curves as produced by Pycolator . . 14
3.3 Results of strategies regarding different ranks 16
3.4 Maintaining proportions works as intended 17
3.5 Results of maintaining the proportions of classes in the dataset . 17

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

- Motivation: Was zeichnet cross-links aus und warum probiere ich da Sachen?

1.1 Background

Proteomics is an 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 [8, 12]. In this thesis, research was done in a related field, focusing on peptides cross-linked with RNA. The chemical bond between cross-linked molecules has been artificially induced, for example using UV light [12]. Applying this to peptides and RNA could possibly give insight into their *in vivo* interactions, and may also allow conclusions about protein-DNA interaction.

1.1.1 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) combined with 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 [12]. As Han et al. [8] explains, the mass spectrometer then produces mass spectra, which have to be analyzed further. It does so by first ionizing the substrate, because it can only detect charged particles. 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 a particle 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

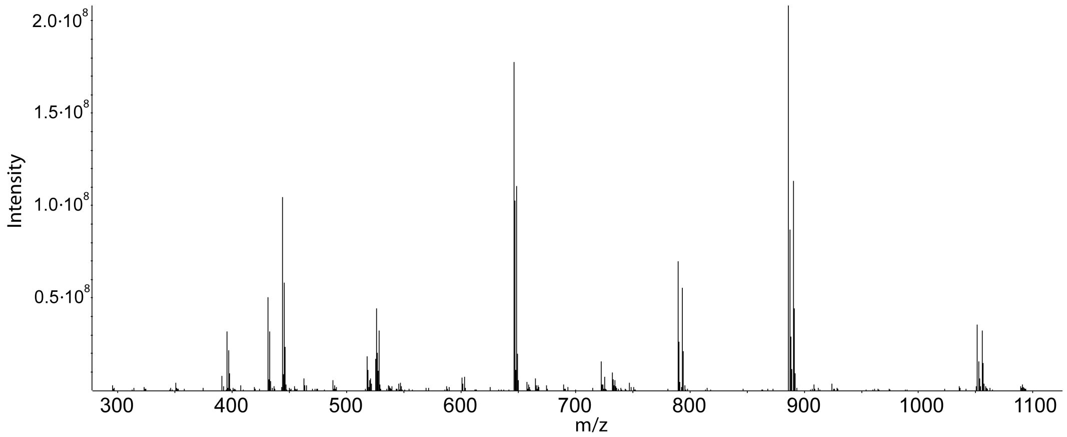


Figure 1.1: Example for a mass spectrum as recorded by a mass spectrometer. The ion intensity correlates with the amount of a molecule in the sample, $\frac{m}{z}$ is the mass-to-charge-ratio. From: Sachsenberg [12]

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 [12]. In there, the substance collides with a gas to be broken down into smaller molecules. For proteins, fragmentation happens predominantly in their backbone, producing all possible sub-sequences of the peptide. This yields a spectrum that is almost unique for its protein, which allows for peptide identification using bioinformatics tools [2].

1.1.2 Bioinformatics Tool

Algorithms like Sequest [5] or X! Tandem [4] 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 [9], but they enable FDR estimation using decoy databases and serve as a basis for score re-calibration with the Percolator algorithm [9, 7].

Decoy databases are created from the target database, contain usually as many peptides [11, 10] in a reversed or shuffled order with respect to the amino acid sequence [1]. They are presented to the scoring algorithm either separately [7] or mixed with the target database [11]. It is assumed, that decoy and target

peptides have similar features [10] and are not easily distinguishable by a scoring algorithm. When the actually fitting 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 probability of a PSM being a false target 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 [7]. 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 measure for a single PSM rather than a metric for a set of PSMs is then derived from this as the minimum FDR of all PSMs with a lower or equal score [7, 1]. It will be used for estimating the credibility for any one PSM.

1.1.3 Percolator

As Käll et al. [9] say, separating correct from incorrect target PSMs with already mentioned algorithms works fine, but there is still room for improvement. This is because often not all information is used and considered jointly. Percolator [9, 7] tries to utilize as much information as possible by using scores from different algorithms, features of the peptide (like its length), of 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 getting a better separation of true and false PSMs. If this holds true, the positive training set of the next iteration better is of higher quality and the SVM can be trained even better. The algorithm usually converges within the first 10 iterations [9].
As explained by Bishop [3], an SVM is a maximum margin classifier model. In

---

[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 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. If the data is not linearly separable, it 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 the hyperplane a point lies on the wrong side, the more this gets penalized. The strength of the penalization is also 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. [7] 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 smaller batches (Percolator uses 3), presenting all but one to the model as training and 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 [7].

1.1.4 ROC curves

As well explained by Fawcett [6], performance of classification machine learning models is often evaluated using receiver operating characteristics (ROC) curves. They are graphs in which the true positive rate of a model on the y-axis against its false positive rate on the x-axis. An example is shown in figure 1.2. Generally, a classifier producing an ROC curve 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" ([6], page 868).

---

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

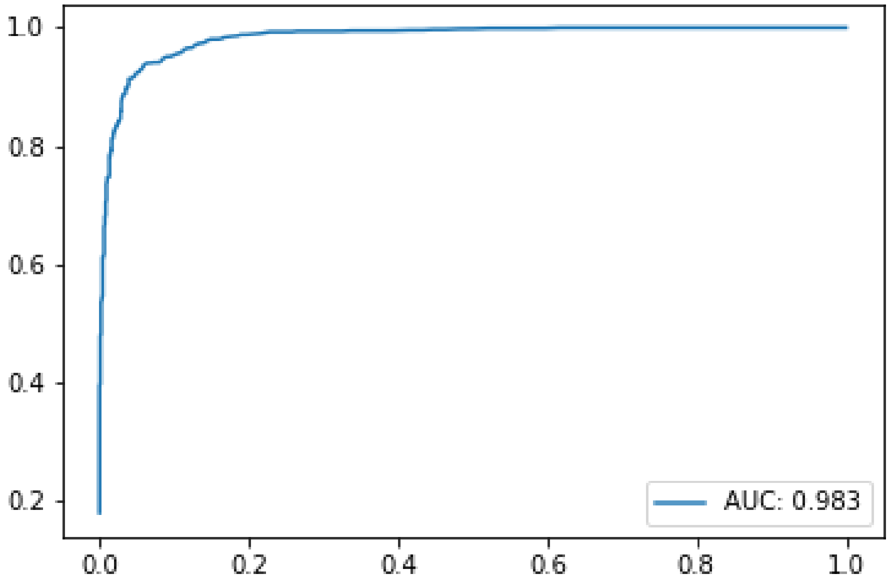
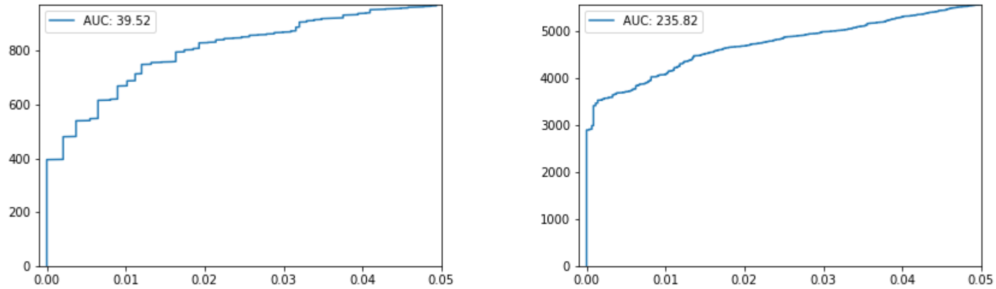


Figure 1.2: An Example for an ROC curve. It was plotted from a generated dataset with 1000 positive and negative entries respectively, being ”scored” using a log normal distribution with different expected values. Calculations were done by the `pseudoROC` function as described in 2.1.



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

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

Figure 1.3: Examples for pseudo ROC curves, plotted using the function `pseudoROC` as described in 2.1.

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 [9, 7]. The graphs shape resembles that of an ROC curve, and is thus called pseudo ROC. However, there are significant 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 the confusion matrix notation from Fawcett [6], 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 $\in [0, 1]$), and thus the AUC, which is still used as a metric, does not range from 0 to 1 and also does not represent a probability. Examples for such a pseudo ROC are shown in figure 1.3.

Chapter 2

Material and Methods

- Wo kommen die Daten her? Warum heißt der score NuXL score?

To be able to test the following implementations, a dataset containing $93,219$ PSMs was used. Out of these, $51,521$ are cross-linked and $41,698$ linear peptides, $50,627$ are target and $42,592$ are decoy peptides. Every PSM has 52 features, 2 columns specifying the spectrum and 2 columns describing the peptide and the protein it originated from, as well as one column indicating to the experimenter whether one entry is a target or a decoy PSM. The features contain the NuXL score, which was computed from the scores of different algorithms as discussed in 1.1.2, and was 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 sklearn 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 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

---

[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

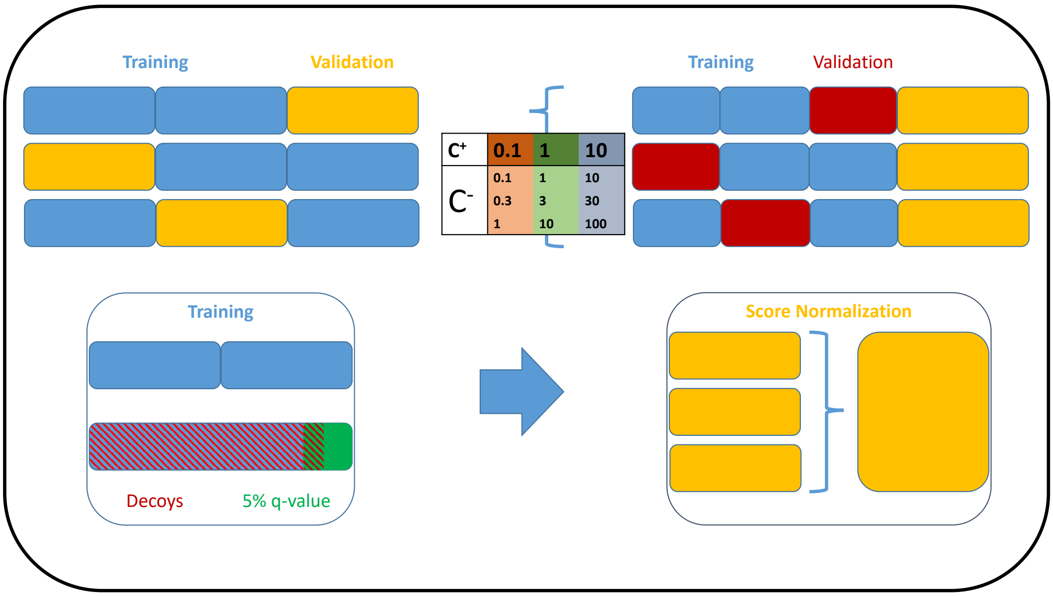


Figure 2.1: Rough outline of the Percolator algorithm. First, the dataset is split in three parts, of which two are used for training and one for scoring. Then, another cross-validation is performed on the training part to determine the best parameter combination. Percolator searches the grid spanned 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 $= \frac{\text{old score} - \texttt{qScore}}{\texttt{qScore} - \texttt{dMedian}}$.

are calculated as discussed in 1.1.2 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 sklearns 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 difference any experiment makes, 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 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 looking at only one of the classes.

Second, a ROC curve using the `pseudoROC` function (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 output. This allows for fast visual detection of the impact a specific change to the algorithm has on certain classes, iterations or general sensitivity.

2.2.1 Different Ranks

As experience shows, cross-linked peptides can be harder to detect than linear 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, Percolator can correct the scores of some lower-ranking PSMs, possibly detecting more cross-linked PSMs. Meanwhile it is known to the experimenter, 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.

---

[5] https://matplotlib.org/api/pyplot_api.html

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

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 2+ 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. Since also some of the best scoring PSMs will be dropped (even though they were not assigned rank 1), the algorithm then runs further until the maximum iterations are reached or the score does not further improve, in order to properly integrate and score the new PSMs considered confident.

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 (2) and pseudo ROCs were plotted as described in 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 2.2.2 "Splitting the dataset", the features of cross-linked and linear peptides are so dissimilar, splitting the dataset and training a linear SVM separately on cross-links and non-cross-links yields significantly better results than training one linear SVM. To reduce the impact of this heterogeneity, the following experiments were conducted:

Proportions of Different Classes

As discussed in 1.1.3, Percolator employs a nested cross-validation approach, splitting the dataset, training on all parts than one and testing/scoring on the remaining part. If the splitting was uneven by chance, the SVM would be trained badly and the scoring inaccurate. 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 has been measured by running the algorithm 10 times, plotting and recording the results of the best, worst and median run w.r.t. to the q-value any number of PSMs was identified at. Because this took approximately 90 minutes, the test was performed in a Google Colaboratory notebook[7].

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 [4] hyper score for the cross-linked peaks. In the given dataset (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 (2.1) will tell the SVM to give the PSM a higher score. However, the scikit-learn package provides solutions for this problem[8], and one of these, the `IterativeImputer`, was tested.

Splitting the Dataset

Because the SVM is linear, one feature can not 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. It was also tested splitting the dataset by the nucleotide the peptide was cross-linked onto, and non-cross-linked PSMs as a fifth class. As will be discussed later, neither is applicable to most datasets, but it gives an upper limit for how well the algorithm could perform on a joint dataset.

2.2.3 Small datasets

The portion of cross-linked peptides in a dataset is often very small. To evaluate when splitting as proposed in 2.2.2 "Splitting the Dataset" is

---

[7] The Google Colaboratory notebook used:
https://colab.research.google.com/drive/1VqZAmdta57YhgobA0WkQMqe_U9YIUnDl?usp=sharing

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

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 (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 dataset utilizing the given, precomputed NuXL score.
First, the smaller dataset was sampled using every 2^i-th PSM from the whole dataset with $i \in \mathbb{N} \land i \in [0, 13]$. Second, the smaller dataset was sampled randomly using a uniform distribution from all PSMs with a q-value of $< 10\%$. The portions sampled were $\frac{1}{2^i} : i \in \mathbb{N} \land i \in [0, 12]$. Additionally to the metrics mentioned above, also the expected number of identifications at 1% q-value for the split were recorded. For the NuXL score as reference and the Pycolator score respectively one plot showing the portion of PSMs with a q-value of 1% and one plot showing the ratio of found to number of expected PSMs with 1% q-value as a function of the dataset size were drawn. In the end, the ratio of AUCs and identifications at 1% q-value generated by the NuXL vs Pycolator score were plotted as a function of the dataset size. To compensate for the randomness in this experiment, Monte-Carlo-Sampling with 10 iterations was performed and the results were presented with boxplots.
Because numerical problems occur when calculating the area under a pseudo ROC curve of a very small dataset (since the curve is only one point), another metric was introduced: the number of identified PSMs at a q-value threshold of 1%. Pycolator decides automatically if this metric is required based upon the size of the dataset, and adjusts its log, plots per iteration and functions like the early stopping criterion accordingly.

(- Performance auf anderem Datensatz
- Vergleich mit Entrapment FDR)

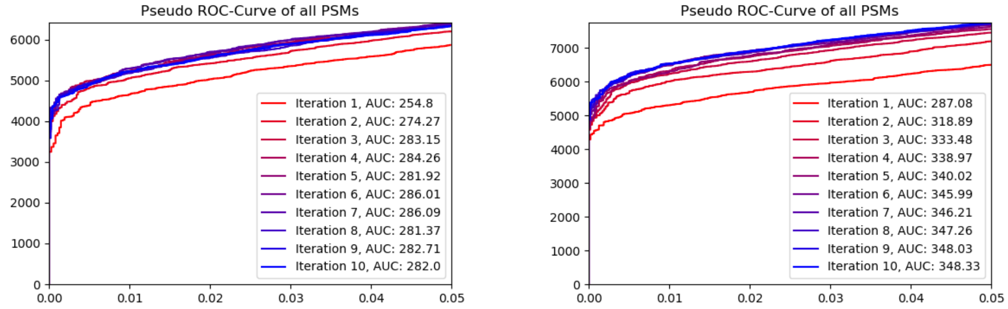
Chapter 3

Results

3.1 Implementation of the percolator algorithm

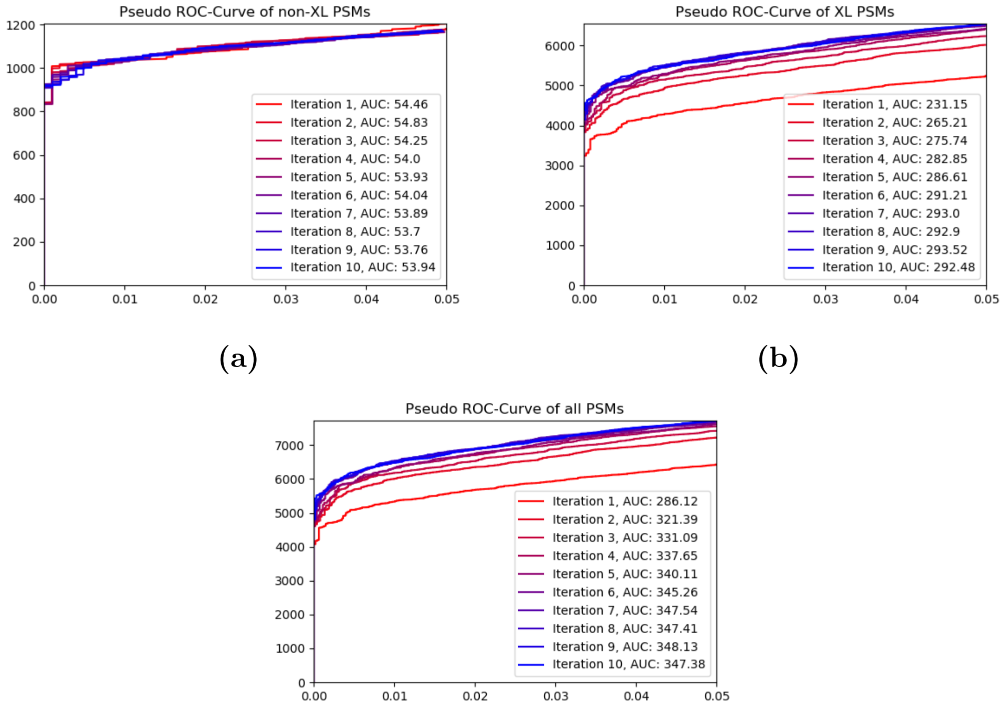
- Reimplementierung funktioniert wie Original → Dafür müsste man irgendwie C Percolator zum testen nutzen können.

The implementation of feature normalization yielded an increase in the number of confident PSMs found, as the y-axis and legend of 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 before (3.1a) and after (3.1b) the implementation of feature normalization.



(a) **(b)**

(c)

Figure 3.2: Example of the plots output by Pycolator, generated by running with default parameters on the given dataset (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

An example of the plots created by Pycolator are shown in figure 3.2. One plot shows the pseudo ROC of the non-cross-linked PSMs in every iteration, one the cross-linked and one all of the PSMs. In each, every iterations pseudo ROC curve has its own color, ranging 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 1.1.4, a higher and more left (pseudo) ROC curve is better. Thus, as one can see in figure 3.2c, the result of Pycolator improves over almost every iteration.

3.2.1 Different Ranks

Figure 3.3 shows the pseudo ROCs of Pycolator before the implementation of the new mechanism. 3.3a was plotted when Pycolator used all PSMs available, meaning all ranks, and only at the end lower ranking PSMs than 1 were excluded. 3.3b is the result of running Pycolator only with rank 1 PSMs of the given dataset 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. Multiple runs yielded very similar results.

3.2.2 Characteristics of Cross-linked PSMs

Proportions of Different Classes

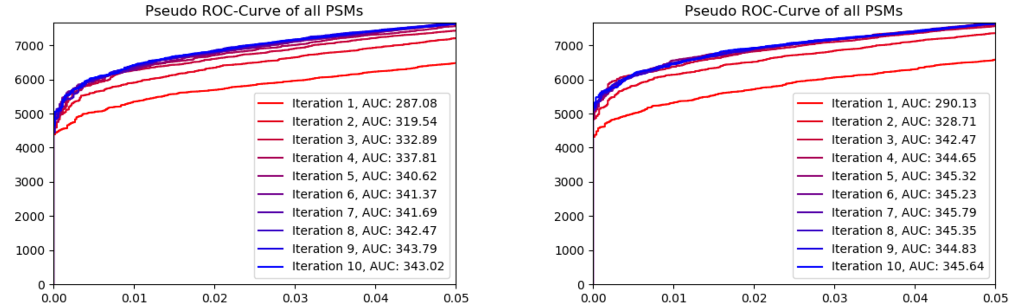
- Verhältnis Targets:Decoys und XL:non-XL verringt die Streuung: MinMax-Median Auswertungen
Maintaining the proportion of classes works as intended, as figure 3.4 shows. It depicts the proportion of targets (t) to decoys (d) in each of the three outer splits and the whole dataset.
The evaluation produced the results shown in figure 3.5. The corresponding areas under the curve are listed in table ??.

Imputation

- Bei Imputation kam nichts heraus

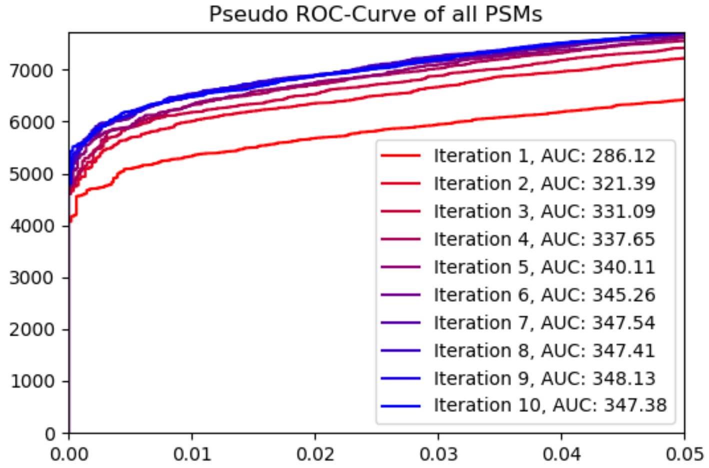
Splitting the Dataset

- Großer Unterschied wenn man den (großen) Datensatz nach XL/nXL oder sogar cross-linking target aufteilt



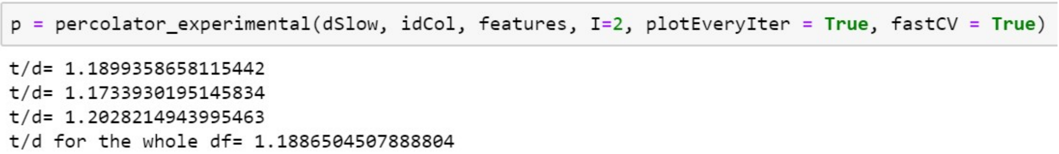
(a) The resulting pseudo ROCs (explained in 3.2) if Pycolator is given every PSM regardless of its rank. Lower ranking PSMs are only dropped in the end, which results in a final AUC of 343.21.

(b) The resulting pseudo ROCs (explained in 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 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.

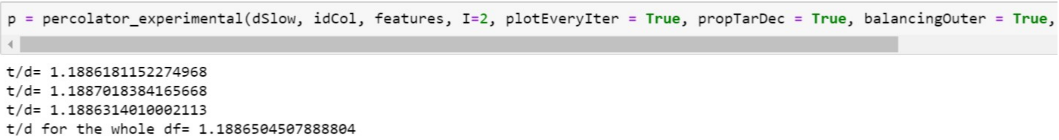
Figure 3.3: Results of running Pycolator with different strategies as to how lower ranking PSMs are dealt with.



```
p = percolator_experimental(dSlow, idCol, features, I=2, plotEveryIter = True, fastCV = True)

t/d= 1.1899358658115442
t/d= 1.1733930195145834
t/d= 1.2028214943995463
t/d for the whole df= 1.1886504507888804
```

(a) Ratios of target to decoy PSMs in each of the three splits, ranging from 1.173 to 1.203, while the ratio is 1.8865 in the whole dataset.

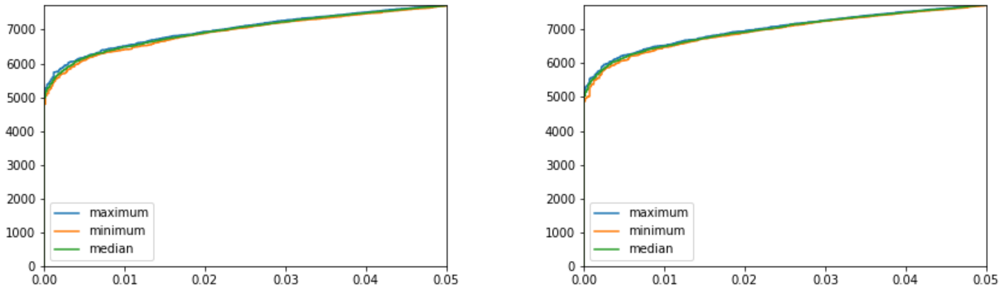


```
p = percolator_experimental(dSlow, idCol, features, I=2, plotEveryIter = True, propTarDec = True, balancingOuter = True,

t/d= 1.1886181152274968
t/d= 1.1887018384165668
t/d= 1.1886314010002113
t/d for the whole df= 1.1886504507888804
```

(b) Ratios of target to decoy PSMs in each of the three splits, ranging from 1.18862 to 1.18870, while the ratio is 1.18865 in the whole dataset.

Figure 3.4: The log of two test runs of Pycolator. It calculated and output the ratio of targets to decoys in every of the outer split and the whole dataset. In 3.4b the proportions were maintained, and in 3.4a they were not.



(a) (b)

Figure 3.5

3.2.3 Small datasets

First, the smaller dataset was sampled using every 2^i-th PSM, with i being a natural number up to 13. This non-random approach ensures the appearance of PSMs originally classified as correct, but it also keeps the ratio of correct to incorrect PSMs. This leads to cases where only one or two PSMs appear, which originally were assigned a q-value of $< 1\%$.
1. Experiment: Hints dass zu viele identifizierungen kommen wenn dataset zu klein und neue Metrik eingebaut
2. Experiment dann richtig
differing lengths of datasets dxl 7 p is because of filtration of ranks (?) - Sinnvolle Plots zu Ratio Testing
- Neue Metrik erlaubt es der Implementierung, auch auf kleineren Datensätzen zu funktionieren

Chapter 4

Discussion and Outlook

As already explained in 2.2.1, cross-linked peptides often are harder to score than linear ones. Therefore, they can get a lower score than appropriate and it can be useful to also include the best scoring cross-linked peptide when running the Percolator algorithm. It may be able to revise the PSMs scores and the actually correct cross-linked peptide may become the best scoring PSM. This thesis is also supported by the findings in 3.2.1. If only including the best scoring PSM, of which 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 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). Then, as the log shows, 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 improves 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.

Comparing the AUC of Pycolator when run with the new mechanism (3.3c) after iteration 6 (345.26) with the end result of running Pycolator with every PSM available (3.3a, 343.21), yields the following insight: Dropping all PSMs with rank 2+ yields a worse result when the Percolator algorithm has been running for 10 than in the case of 6 iterations. This implies an overfitting onto the PSMs with lower quality and thus a worse scoring.

- Methoden hinterfragen oder begründen, Ergebnisse interpretieren, Anwendbarkeit diskutieren, z.B.:
- Falsche Formel für q-value
- C Parameter für jeden split neu optimieren führt zu overfitting? → Original-Algorithmus macht es auch so
- Wie sinnvoll ist die neue Metrik (idents bei 1%)?
- ScanNr Versuche: Gleiche Spektren (identifiziert anhand der ScanNr.) auf verschiedene splits verteilen verändert nichts, d.h. vermutlich sind die niedrigeren Ränge dann so schlecht, dass es nichts bringt die schonmal gesehen zu haben.
- Peptide Versuche: Schlechtere Ergebnisse, aber vllt ehrlicher?
- Mögliche weiterführende Experimente: mächtigere Klassifikatoren + monotonic constraints (wie von Timo ausprobiert), Ada-Boosting, feature selection

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] 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`.

[5] 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.

[6] 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`.

[7] 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`.

[8] Xuemei Han, Aaron Aslanian, and John R Yates. Mass spectrometry for proteomics. *Current Opinion in Chemical Biology*, 12(5):483–490, Octo-

ber 2008. doi: 10.1016/j.cbpa.2008.07.024. URL `https://doi.org/10.1016/j.cbpa.2008.07.024`.

[9] 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`.

[10] 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`.

[11] 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`.

[12] 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`.

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