Published in final edited form as:
J Proteome Res. 2009 July 6; 8(7): 3737–3745. doi:10.1021/pr801109k.

Improvements to the Percolator algorithm for peptide identification from shotgun proteomics data sets

Marina Spivak[1], **Jason Weston**[1], **Léon Bottou**[1], **Lukas Käll**[2,3], and **William Stafford Noble**[2,4,*]

[1] NEC Research, Princeton, NJ, USA

[2] Department of Genome Sciences, University of Washington, Seattle, WA, USA

[3] Center for Biomembrane Research, Department of Biochemistry and Biophysics, Stockholm University, Sweden

[4] Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA

Abstract

Shotgun proteomics coupled with database search software allows the identification of a large number of peptides in a single experiment. However, some existing search algorithms, such as SEQUEST, use score functions that are designed primarily to identify the best peptide for a given spectrum. Consequently, when comparing identifications across spectra, the SEQUEST score function *Xcorr* fails to discriminate accurately between correct and incorrect peptide identifications. Several machine learning methods have been proposed to address the resulting classification task of distinguishing between correct and incorrect peptide-spectrum matches (PSMs). A recent example is Percolator, which uses semi-supervised learning and a decoy database search strategy to learn to distinguish between correct and incorrect PSMs identified by a database search algorithm. The current work describes three improvements to Percolator. (1) Percolator's heuristic optimization is replaced with a clear objective function, with intuitive reasons behind its choice. (2) Tractable nonlinear models are used instead of linear models, leading to improved accuracy over the original Percolator. (3) A method, Q-ranker, for directly optimizing the number of identified spectra at a specified q value is proposed, which achieves further gains.

Keywords

shotgun proteomics; tandem mass spectrometry; machine learning; peptide identification

1 Introduction

A shotgun proteomics mass spectometry experiment produces, for a given biological sample, a collection of spectra, each of which may be mapped back to its generating peptide using either *de novo* or database search techniques (reviewed in [26,25]). Critical to any database search procedure is the score function that evaluates the quality of the match between an observed spectrum and a candidate peptide. This function plays two complementary roles. First, the function ranks candidate peptides relative to a single spectrum, producing a single, top-scoring peptide-spectrum match (PSM) for each spectrum. Second, the function ranks the PSMs from different spectra with respect to one another. This latter, absolute ranking task is intrinsically more difficult than the relative ranking task. A perfect absolute ranking function is by definition

*Corresponding author: E-mail: noble@gs.washington.edu.

also a perfect relative ranking function, but the converse is not true, because PSM scores may not be well calibrated from one spectrum to the next.

A variety of approaches have been developed to learn PSM scoring functions from real data. Typically, the input to these PSM post-processing methods is the relative score, as well as properties of the spectrum, the peptide and features that represent the quality of the PSM. PeptideProphet [19], for example, uses four statistics computed by the SEQUEST database search algorithm as input to a linear discriminant analysis classifier. The system is trained from labeled correct and incorrect PSMs derived from a purified sample of known proteins. Other approaches use alternative feature representations or classification algorithms, such as support vector machines (SVMs) [1] or decision trees [11].

One drawback to these machine learning approaches is that they often do not generalize well across different machine platforms, chromatography conditions, etc. Consequently, when the experimental conditions change, a new training set must be acquired, and this acquisition and training can be expensive.

To combat this problem, several methods have been described that adjust the parameters of the model with respect to each new data set. PeptideProphet, for example, uses a fixed linear discriminant function but couples it with a postprocessor that maps the resulting unitless discriminant score to an estimated probability. In the original version of PeptideProphet [19], this mapping function was learned from each data set in an unsupervised fashion (i.e., without knowing which PSMs are correct and which are incorrect) using the expectation-maximization (EM) algorithm [9].

Subsequently, several algorithms have been described that use *semi-supervised learning* to adjust model parameters with respect to each new data set. In contrast to supervised learning, in which the given training set is fully labeled, a semi-supervised learner is provided with a partially labeled training set. In the context of PSM scoring, these labels are created using a decoy database [24]. Each spectrum is searched once against the real (“target”) protein database and once against a decoy database comprised of reversed [24], shuffled [20] or Markov-chain generated proteins [6]. Matches to the target database are unlabeled—they may or may not be correct (we expect 50–90% are false positives). But matches to the decoy database can be confidently labeled “incorrect.”

The semi-supervised version of PeptideProphet [5] uses decoy PSMs to improve the mapping from discriminant scores to probabilities. During the EM step, PeptideProphet includes decoy PSMs, forcing them to be labeled “incorrect.” The resulting probabilities are significantly more accurate than probabilities estimated in an unsupervised fashion.

The Percolator algorithm [17] takes the semi-supervised approach one step further. Rather than using a fixed discriminant function and employing semi-supervised learning as a postprocessor, Percolator solves the entire problem in a semi-supervised fashion, learning a function that consistently ranks the decoy PSMs below a subset of high-confidence target PSMs. Percolotor uses an iterative, SVM-based algorithm, initially identifying a small set of high-scoring target PSMs, and then learning to separate these from the decoy PSMs. The learned classifier is applied to the entire set, and if new high-confidence PSMs are identified, then the procedure is repeated. Critical to the success of the algorithm is a statistical scoring procedure, based on estimated false discovery rates [2], that prevents explosion of the high-confidence set of PSMs.

A subsequent version of PeptideProphet [10] extends that algorithm in a similar fashion. Like Percolator, the newest version of PeptideProphet adjusts the parameters of the discriminant function to reflect specific features of the data set and allows the algorithm to use more than one PSM for the identification of the best scoring peptide. In addition, the algorithm uses a measure of spectrum quality in its model.

Despite the good performance of Percolator, the algorithm itself is somewhat heuristic; indeed, it is unclear what exactly Percolator optimizes and whether the algorithm's iterative optimization process provably converges. The current work proposes a novel, well-founded approach to this problem. Although only some of the matches to the target database are positive examples, we opt to treat this problem as a fully supervised classification problem with noisy labels; i.e., we label all the target PSMs "correct" (but some of these are mislabeled) and all the decoy PSMs "incorrect." However, we define a loss function that does not severely penalize examples that are far from the decision boundary. In this way, incorrect target PSMs do not strongly affect the learning procedure. We show how this choice of loss is superior to more classical choices of loss function, and in the linear case how this yields results similar to the original *semi-supervised* Percolator algorithm. An important benefit of using a fully supervised approach is that, in contrast to Percolator, the new approach defines a clear, intuitive objective function whose minimization is known to converge. Furthermore, the resulting classifier can be trained with tractable nonlinear models which then significantly improve the results of Percolator. Subsequently, we propose a modification of our algorithm that directly optimizes the number of PSMs relative to a user-specific statistical confidence threshold. This ability to specify the desired confidence threshold *a priori* is useful in practice and leads to further improvement in the results. The new algorithm, called Q-ranker, is implemented in Crux version 2.0, which is available with source code at http://noble.gs.washington/proj/crux.

2 Materials and Methods

2.1 Data sets

We used four previously described data sets to test our algorithms [17]. The first is a yeast data set containing 69,705 target PSMs and twice that number of decoy PSMs. These data were acquired from a tryptic digest of an unfractionated yeast lysate and analyzed using a four-hour reverse phase separation. Throughout this work, peptide were assigned to spectra by using SEQUEST with no enzyme specificity and with no amino acid modifications enabled. The next two data sets were derived from the same yeast lysate, but treated by different proteolytic enzymes: elastase and chymotrypsin. These data sets respectively contain 57,860 and 60,217 target PSMs and twice that number of decoy PSMs. The final data set was derived from a *C. elegans* lysate proteolytically digested by trypsin and processed analogously to the yeast data sets.

Each PSM was represented using the 17 features listed in Table 1. Note that, originally, Percolator used 20 features. In this work, we removed three features that exploit protein-level information, because of the difficulty of accurately validating, via decoy database search, methods that use this type of information. We also defined 20 additional features for each peptide, also defined in Table 1, corresponding to the counts of amino acids in the given peptide. Using these addition features yields a feature vector of length 37.

2.2 Statistical confidence estimates

Throughout this work, we use the q value [28] as a statistical confidence measure assigned to each PSM. If we specify a score threshold t and refer to PSMs with scores better than t as *accepted* PSMs, then the *false discovery rate* (FDR) is defined as the percentage of accepted PSMs that are incorrect (i.e., the peptide was not present in the mass spectrometer when the spectrum was produced). The q value is defined as the minimal FDR threshold at which a given PSM is accepted. Note that the q value is a general statistical confidence metric that is unrelated to the Qscore method for evaluating SEQUEST results [24].

We calculate q values by using decoy PSMs [18], derived by searching each spectrum against a database of shuffled protein sequences. Denote the scores of target PSMs $f_1, f_2, \ldots, f_{mf}$ and the scores of decoy PSMs $d_1, d_2, \ldots, d_{md}$. For a given score threshold t, the number of accepted PSMs (positives) is $P(t) = |\{f_i > t; i = 1, \ldots, m_f\}|$. The estimated number of false positives among the positives is given by $E\{FP(t)\} = \pi_0 \frac{m_f}{m_d} |\{d_i > t; i = 1, \ldots, m_d\}|$, where π_0 is the estimated proportion of target PSMs that are incorrect. In this work, as previously [17], we use a fixed $\pi_0 = 0.9$. We can then estimate the FDR at a given threshold t as

$$E\{FDR(t)\} = \frac{\pi_0 \frac{m_f}{m_d} |\{d_i > t; i = 1, \ldots, m_d\}|}{|\{f_i > t; i = 1, \ldots, m_f\}|}$$

The q value assigned to score f_i is then

$$q(f_i) = \min_{f_j \le f_i} E\{FDR(f_j)\}$$

3 Results

3.1 A supervised algorithm for target-decoy discrimination

Given a set of examples (PSMs) $(\mathbf{x_1}, \ldots \mathbf{x_n})$ (where the bold face denotes a vector) and corresponding labels $(y_1, \ldots y_n)$, the goal is to choose a discriminant function $f(\mathbf{x})$, such that

$$\begin{array}{l} f(\mathbf{x_i}) > 0 \;\; \text{if} \;\; y_i = 1 \\ f(\mathbf{x_i}) < 0 \;\; \text{if} \;\; y_i = -1. \end{array}$$

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

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

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

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

To find the function that best minimizes the loss, we choose to use gradient descent, so the loss function itself must be differentiable. This requirement prevents us from simply counting the number of mistakes (mislabeled examples), which is called the zero-one loss. Typical differentiable loss functions include the squared loss, often used in neural networks [22], the hinge loss, which is used in support vector machines [8], and the sigmoid loss. These loss functions are illustrated in Figure 1.

In general, choosing an appropriate loss function is critical to achieving good performance. Insight into choosing the loss function comes from the problem domain. In the current setting, we can safely assume that a significant proportion of the PSMs produced by a given search algorithm are incorrect, either because the score function used to identify PSMs failed to accurately identify the correct peptide, or because the spectrum corresponds to a peptide not in the given database, to a peptide with post-translational modifications, to a heterogeneous population of peptides, or to non-peptide contaminants. Therefore, in this scenario, a desirable loss function will be robust with respect to the multiple false positives in the data. In other words, a desirable loss function will not strongly penalize misclassified examples if they are too far away from the separating hyperplane. Considering the loss functions in Figure 1, the sigmoid loss is the only function with the desired property: when $y_i f(\mathbf{x}) < -5$ the gradient is close to zero. The squared loss, on the other hand, has a larger gradient for misclassified examples far from the boundary than for examples close to the boundary, whereas the hinge loss penalizes examples linearly (it has a constant gradient if an example is incorrectly classified). We therefore conjecture that the sigmoid loss function should work much better than the alternatives.

3.2 Supervised learning yields performance comparable to Percolator

We test this conjecture by measuring the performance of the learned scoring function using a target-decoy search strategy. For this experiment, we use a collection of spectra derived via microcapillary liquid chromatography MS/MS of a yeast whole cell lysate. These spectra were searched using SEQUEST [13] against one target database and two independently shuffled decoy databases, producing a collection of PSMs. For a given ranking of target PSMs, we use the corresponding collection of decoy PSMs to estimate q values (Section 2.2). Our goal is to correctly identify as many PSMs as possible for a given q value. Therefore, in Figure 2, we plot the number of identified PSMs as a function of q value threshold.

To ensure a valid experiment, we split the target and decoy PSMs into two equal parts. We train on the data set composed of the first half of positives and negatives, and we use the second half of the data as a testing set. The q value estimates are derived from the test set, not the training set. This approach is more rigorous than the methodology employed in [17], in which the positive examples were used both for training and testing. However, the similarity between Figure 2(A) and (B) indicates that overfitting is not occurring. Nonetheless, in subsequent experiments, we retain a full separation of the train and test sets.

Figure 2 compares the performance of ranking by XCorr, Percolator and a linear model trained using three different loss functions. The figure shows that, for example, the Percolator algorithm identifies 5917 PSMs at a q value threshold of 0.01. As expected, the sigmoid loss dominates the other two loss functions that we considered, square loss and hinge loss.

In fact, the linear model with the sigmoid loss achieves almost identical results to the Percolator algorithm. This concordance can be explained in the following way. Percolator also uses a linear classifier (a linear SVM) with a hinge loss function. However, on each iteration *only a subset of the positive examples are used as labeled training data* according to the position of the hyperplane. The rest of the positive examples that have a small value of $y_i f(\mathbf{x_i})$ are ignored during training. Consequently, one can say that their gradient is zero; hence, the hinge loss function is "cut" at a certain point so that it no longer linearly penalizes mistakes at any distance, as shown in Figure 3. A cut hinge loss is effectively a piece-wise linear version of a sigmoid function. Indeed, such a cut hinge loss has been used before and is referred to as a *ramp loss* [7]. By using a sigmoid loss function, we have thus developed a method that explains the heuristic choices of the Percolator algorithm but instead implements a direct, intuitive objective function. Hereafter, we refer to this method as "direct classification."

3.3 Nonlinear families of discriminant functions yield improved performance

Having established that direct classification using a linear model performs as well as Percolator on this data set, we next consider a nonlinear family of functions by considering two-layer neural networks

$$f(\mathbf{x})=\sum_i w_i h_i(\mathbf{x})+b,$$

where $h_k(\mathbf{x})$ is defined as $\tanh((w^k)^\top \mathbf{x} + b_k)$, and w^k and b_k index the weight vector and threshold for the kth hidden unit.

We can choose the capacity of our nonlinear family of discriminant functions by increasing or decreasing the number of hidden units of our neural network. Based on preliminary experiments with the yeast training data set, we chose the first layer to have five linear hidden units. An experimental comparison in Figure 4 shows that a nonlinear classifier outperforms the linear model on the same data set as before. For every q value in the plot, the nonlinear model (the solid blue line with the label "direct classification (linear)") produces as many or more PSMs than its linear counterpart (solid black line labeled "direct classification (nonlinear)").

3.4 The Q-ranker algorithm for optimizing relative to a specified *q* value

We have established that framing our problem as a supervised classification task, utilizing nonlinear models, yields slightly improved results compared with Percolator's semi-supervised approach. We now show that reformulating the problem as a ranking task, rather than as a classification task, leads to even better performance.

Generally speaking, the goal of many shotgun proteomics experiments is to identify as many proteins as possible at a given q value threshold. For the peptide identification problem, this task corresponds to finding a ranking of PSMs that maximizes the number of accepted PSMs for a specified q value threshold. To solve this ranking problem directly, we therefore assume that the user specifies a particular desired q value threshold *a priori*. We then search for a ranking that is optimal with respect to the given q value. A standard formulation for solving the ranking problem is the ranking SVM [15,16], which can be stated as follows:

$$\min ||w||^2 \tag{1}$$

subject to

$$w^\top \mathbf{x_i} \geq w^\top \mathbf{x_j}+1, \text{ if } y_i=1 \text{ and } y_j=-1 \tag{2}$$

This algorithm re-orders the examples so that larger values of $w^\top \mathbf{x}$ correspond to positive examples. Note that, compared to the classification problem posed before, this formulation no longer has a threshold b, because a class label is no longer predicted, only an ordering. The ranking formulation is equivalent to optimizing the area under the receiver operating characteristic (ROC) curve [14], and hence would optimize all q values at once. The optimization tries to satisfy every pairwise ordering constraint. Again, as in the classification problem, because we expect 50–90% of the positive examples are false positives, the objective function will pay too much attention to these examples.

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

The proposed algorithm is thus as follows. We first find a general discriminant $f(\mathbf{x})$ using the direct classification algorithm described in the previous section. We then specify a q value to be optimized and focus sequentially on several intervals in the data set chosen in the vicinity of the specified q value. The selection of intervals is heuristic and in our case involves defining a set $\hat{Q}$ of q value thresholds 0 to 0.1 with a step size of 0.01 and iterating over these steps. The interval ε is set to equal twice the number of peptides up to the threshold point. In the course of training, we record the best result for the specified q value after each epoch. A pseudocode description of the direct ranking algorithm for specified q values (Q-ranker) is given in Algorithm 1.

Q-ranker can be extended trivially to search for optimal solutions to several q values at once by recording the best network for each of the specified q values after each epoch. In all the experimental runs presented below, the set $\hat{Q}$ of threshold q values also served as a set of specified q values.

In practice, because Q-ranker focuses on a subset of the training set, we found that use of regularization techniques to control for the model complexity improves our results. In this work, we use the standard weight decay procedure, which optimizes the error function:

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

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

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

Interestingly, choosing examples from a certain region of the data set is also roughly equivalent to placing the region of the sigmoid with high gradient over the region of interest about the threshold q value. Because examples further than ε are not picked, this approach is equivalent to making a loss function which has gradient zero in those regions. This means that we are able to replace the sigmoid loss function used for training the general neural net with an even more intuitive choice of loss. In particular, here we use a linear loss $L(f(\mathbf{x}), y) = |f(\mathbf{x}) - y|$ which effecively becomes a “ramp loss” (c.f. Figure 3) centered around the q value threshold with flat parts at $\pm\varepsilon$. Because we are solving a ranking problem in the nonlinear case, we now choose a network with the following architecture:

$$f(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}), \text{ where } h_k(\mathbf{x}) = \tanh((w^k)^\top \mathbf{x} + b_k)$$

i.e., we no longer have a final bias output.

3.5 Q-ranker yields even better performance

We tested our direct classification and Q-ranker algorithms on the tryptically digested yeast data set in Figure 4. It is clear from this figure that, although the linear Q-ranker algorithm does not improve over the direct classification algorithm, using a nonlinear architecture leads to a large improvement, especially for larger q values. Other choices of nonlinear architectures (number of hidden units) are given in Supplementary Figure 7, each leading to improved performance relative to Percolator.

Compared to the direct classification approach described in Section 3.1, Q-ranker also yields more consistent training behavior when observed for any given q value. To illustrate this phenomenon, we fix the interval ε for the Q-ranker algorithm to be defined by the single threshold corresponding to the specified q value. Figure 5A shows how the results for different specified q values change during the course of training the direct classification model. The number of PSMs over lower q value thresholds (for example, 0.0075, 0.01) reach their peak early during training and then become suboptimal, while the best results for higher q value thresholds take longer to achieve. This means that during the course of training, different q value thresholds are being optimized depending on the number of iterations. In contrast, as shown in Figure 5B, the Q-ranker algorithm learns the best decision boundary for a specified q value threshold and does not substantially diverge from the best result during further training. This behavior indicates that the algorithm in fact optimizes the desired quantity. In the following experiments we therefore adopt Q-ranker as our algorithm of choice, and we compare it further to Percolator and PeptideProphet.

3.6 Comparison of algorithms across multiple data sets

For our final round of experiments, we compare the performance of Q-ranker, Percolator and two versions of PeptideProphet—the original parametric version [19], which assumes that the decoys scores are distributed according to a gamma distribution and the target scores acording to a Gaussian distribution, and a newer, semiparametric approach [4], which uses a mixture model of kernel functions to model the two distributions. For both sets of PeptideProphet results, we use the semi-supervised version of the algorithm [5]. The same set of decoy PSMs is provided to Percolator, Q-ranker and PeptideProphet. For Percolator and Q-ranker, we use 50% of the PSMs for training and 50% for testing, as before. PeptideProphet does not provide the ability to learn model parameters on one set of data and apply the learned model to the second; therefore, PeptideProphet results are generated by applying the algorithm to the entire data set. This difference gives an advantage to PeptideProphet because that algorithm learns its model from twice as much data and is not penalized for overfitting.

We report results using either 17 or 37 features, as described in Table 1, for both Percolator and Q-Ranker. Figure 6 shows the results of this experiment, conducted using the four data sets described in Section 2.1. Across the four data sets, Q-ranker consistently outperforms PeptideProphet across all q value thresholds. The left half of Table 2 shows a detailed comparison of Percolator and Q-ranker on all four data sets using 17 features as input. At q values of 0.05 or 0.10, Q-ranker yields more accepted target PSMs than either Percolator or PeptideProphet, whereas Percolator performs slightly better for $q < 0.01$.

Theoretically, a nonlinear network could yield a larger benefit than a linear model when the input feature space is increased, as long as the model does not overfit. We therefore experimented with extending the PSM feature vectors, adding 20 new features corresponding to the counts of amino acids in the peptide. The results of running Q-ranker with these extended vectors are shown in Figure 6, labeled "Q-ranker 37." Increasing the number of features gives a larger boost to the performance of the nonlinear version of Q-ranker. The effect is particularly evident on data sets derived from yeast lysate digested with chymotrypsin and elastase. After this extension, Q-ranker identifies more spectra than either of the other algorithms, even at $q < 0.01$ (right half of Table 2).

Finally, we further investigated the behavior of Q-ranker by measuring the performance of networks trained for a specified q value on other q values. We focused on specified q values 0.01, 0.05 and 0.1. Table 3 shows that, when all 37 features are employed, a network trained for a specified q value is consistently better or equal to the performance on this q value, compared with networks trained for other specified q values.

4 Discussion

In this work, we have performed all of our analyses using a combination of SEQUEST and Percolator. However, the conclusions that we draw here have implications for researchers who do not employ these particular software systems. First, the conclusions likely generalize across search engines. For example, Percolator has previously been demonstrated to work well with the Inspect [17] and MASCOT search engines [3], so it seems likely that Q-ranker will also generalize to these search engines. Second, we have demonstrated the utility of shifting from a semi-supervised framework to a supervised framework with a modified loss function, both in terms of improved understanding of the objective function being maximized and improved discriminative power. A similar shift should be straightforward to apply, for example, to the semi-supervised version of PeptideProphet and may result in similar benefits.

Throughout our evaluations, we have focused on maximizing the number of spectra that are correctly assigned a peptide (i.e., the number of accepted PSMs). It is conceivable that a given algorithm might be biased in the types of peptides it can identify. In this case, the relative performance of two peptide identifications could depend on whether we count the number of accepted PSMs or the number of distinct peptides that are identified from a set of spectra. Supplementary Figure 9 demonstrates that this bias is not occurring in our results: the relative performance of the algorithms that we considered does not change significantly when we count the number of distinct peptides identified.

One surprising result from our experiments is the relatively large benefit provided by amino acid composition features. We hypothesize that this information allows the classifier to learn to expect certain characteristics of a spectrum. For example, the presence of a proline implies a pair of high-intensity peaks corresponding to the cleavage N-terminal to the proline; the presence of many basic residues leads to more +2 ions, and the presence of many hydrophobic residues leads to more singly charged +1 ions [21]. However, previous experiments with Percolator using amino acid composition features did not yield significant performance improvements. The difference, in the current setting, is that we have switched from a semi-supervised to a fully supervised setting. This switch allows us to use a more complex, nonlinear model. In general, a complex model has more opportunity to improve over a simpler model if the feature space is rich. Thus, although a simple linear model such as the one in Percolator cannot fully exploit the richer, 37-dimensional feature space, the nonlinear model can. This conclusion is supported by the observation that adding compositional features also improves the performance of the direct classification method (results not shown).

An alternative, possible explanation for the added discriminative power provided by the amino acid composition feature is that they provide the algorithm with a way to “cheat.” In our experiments, we did not guarantee that the training set and the test set contain disjoint sets of peptides. Hence, an algorithm might overfit on the amino acid composition features and successfully identify the recurrence of a peptide in the train and test sets. To eliminate this alternative explanation, we performed a follow-up experiment in which we prevented the same peptide from occurring in the training and test set. The results, shown in Supplementary Figure 10, show that the improved performance of Q-ranker over Percolator still holds.

A drawback to using a non-linear discriminative classifier is the difficulty in interpreting the learned model. In this work, we have focused on optimizing error rate, not interpretability; sometimes it is hard to have both. Indeed, as shown in Figure 11, simply switching to a linear SVM in the direct classification setting yields markedly decreased performance. However, even with a nonlinear model, it is still possible to gain some insight into the relative contributions of the various features by "knocking out" each feature individually and measuring the performance of the resulting classifier. Supplementary Table 4 shows the percent reduction in the number of identified PSMs at $q < 0.01$ when we knock out each feature of Q-Ranker with 17 features. Not surprisingly, the enzymatic features are most significant, followed by the score features (XCorr and ΔC_n). The relatively small percentage decrease for many features suggests that many provide redundant information. A more detailed interpretation of the model could be derived via further knockout experiments aimed at groups of related features, as was done in [17].

It is worth noting that the relative performance of the methods that we considered does not change when we use an alternative q value estimation scheme. Elias *et al.* [12] advocate estimating the FDR using target-decoy competition (i.e., searching each spectrum against a concatenated database of targets and decoys and only retaining the single top-scoring peptide), and estimating FDRs with respect to the combined collection of target and decoy PSMs. To show that our results do not depend upon our q value estimation procedure, we report in Supplementary Figure 8 results analogous to those given in Figure 6, but using FDRs estimated by following the protocol of Elias *et al.* Even in this case, the Q-ranker algorithm outperforms Percolator and both versions of PeptideProphet.

In general, using a large feature space generally requires a concomitantly large number of training examples. For smaller collections of spectra, or for lower quality spectra in which the effective number of positive examples is small, we would expect a larger feature space to lead to overfitting. In the current version of the software, the user must check for overfitting explicitly, and select the regularization parameter explicitly. One focus of our future work will be the implementation and validation of robust methods for avoiding such overfitting, either by adjusting the regularization parameter or reducing the complexity of the model.

5 Conclusions

We have described a series of algorithms that improve in various ways upon the Percolator algorithm. Given unlabeled target PSMs and negatively labeled decoy PSMs, Percolator treats the problem as a semi-supervised classification problem. In this work, we instead use a supervised approach to the same problem. This change allows us to state an explicit objective function and also allows us to generalize to more powerful, nonlinear models. Finally, if the user is willing to specify a desired confidence threshold, then the Q-ranker algorithm finds an optimal ranking with respect to the specified threshold, yielding consistently improved performance relative to either Percolator or PeptideProphet. Both the direct classification and the Q-ranker algorithms are implemented in the Crux toolkit, which is available with source code from http://noble.gs.washington.edu/proj/crux.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

6 Funding

This work was funded by NIH award R01 EB007057.

References

1. Anderson DC, Li W, Payan DG, Noble WS. A new algorithm for the evaluation of shotgun peptide sequencing in proteomics: support vector machine classification of peptide MS/MS spectra and sequest scores. Journal of Proteome Research 2003;2(2):137–146. [PubMed: 12716127]
2. Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal Statistical Society B 1995;57:289–300.
3. Brosch M, Yu L, Hubbard T, Choudhary J. Accurate and sensitive peptide identification with Mascot Percolator. 2008Submitted
4. Choi H, Ghosh D, Nesvizhskii A. Statistical validation of peptide identifications in large-scale proteomics using target-decoy database search strategy and flexible mixture modeling. Journal of Proteome Research 2008;7(1):286–292. [PubMed: 18078310]
5. Choi H, Nesvizhskii AI. Semisupervised model-based validation of peptide identifications in mass spectrometry-based proteomics. Journal of Proteome Research 2008;7(1):254–265. [PubMed: 18159924]
6. Colinge J, Masselot A, Giron M, Dessingy T, Magnin J. OLAV: Towards high-throughput tandem mass spectrometry data identification. Proteomics 2003;3:1454–1463. [PubMed: 12923771]
7. Collobert R, Sinz F, Weston J, Bottou L. Large scale transductive svms. Journal of Machine Learning Research 2006;7:1687–1712.
8. Cortes C, Vapnik V. Support vector networks. Machine Learning 1995;20:273–297.
9. Dempster AP, Laird NM, Rubin DB. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society 1977;39:1–22.
10. Ding Y, Choi H, Nesvizhskii A. Adaptive discriminant function analysis and reranking of MS/MS database search results for improved peptide identification in shotgun proteomics. Journal of Proteome Research 2008;7(11):4878–4889. [PubMed: 18788775]
11. Elias JE, Gibbons FD, King OD, Roth FP, Gygi SP. Intensity-based protein identification by machine learning from a library of tandem mass spectra. Nature Biotechnology 2004;22:214–219.
12. Elias JE, Gygi SP. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. Nature Methods 2007;4(3):207–214. [PubMed: 17327847]
13. Eng JK, McCormack AL, Yates JR III. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. Journal of the American Society for Mass Spectrometry 1994;5:976–989.
14. Hanley JA, McNeil BJ. The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology 1982;143:29–36. [PubMed: 7063747]
15. Herbrich R, Graepel T, Obermayer K. Support vector learning for ordinal regression. Proceedings of the Ninth International Conference on Articial Neural Networks 1999:97–102.
16. Joachims T. Optimizing search engines using clickthrough data. ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2002:133–142.
17. Käll L, Canterbury J, Weston J, Noble WS, MacCoss MJ. A semi-supervised machine learning technique for peptide identification from shotgun proteomics datasets. Nature Methods 2007;4:923–25. [PubMed: 17952086]
18. Käll L, Storey JD, MacCoss MJ, Noble WS. Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. Journal of Proteome Research 2008;7(1):29–34. [PubMed: 18067246]
19. Keller A, Nesvizhskii AI, Kolker E, Aebersold R. Empirical statistical model to estimate the accuracy of peptide identification made by MS/MS and database search. Analytical Chemistry 2002;74:5383–5392. [PubMed: 12403597]

20. Klammer AA, MacCoss MJ. Effects of modified digestion schemes on the identification of proteins from complex mixtures. Journal of Proteome Research 2006;5(3):695–700. [PubMed: 16512685]
21. Klammer AA, Reynolds SR, Hoopmann M, MacCoss MJ, Bilmes J, Noble WS. Modelling peptide fragmentation with dynamic Bayesian networks yields improved tandem mass spectrum identification. Bioinformatics 2008;24(13):i348–i356. [PubMed: 18586734]
22. LeCun, Y.; Bottou, L.; Orr, GB.; Müller, KR. Efficient backprop. In: Orr, G.; Müller, KR., editors. Neural Networks: Tricks of the Trade. Springer; 1998. p. 9-50.
23. Mason L, Bartlett PL, Baxter J. Improved generalization through explicit optimization of margins. Machine Learning 2000;38(3):243–255.
24. Moore RE, Young MK, Lee TD. Qscore: An algorithm for evaluating sequest database search results. Journal of the American Society for Mass Spectrometry 2002;13(4):378–386. [PubMed: 11951976]
25. Nesvizhskii AI, Vitek O, Aebersold AR. Analysis and validation of proteomic data generated by tandem mass spectrometry. Nature Methods 2007;4(10):787–797. [PubMed: 17901868]
26. Hernandez MMP, Appel RD. Automated protein identification by tandem mass spectrometry: Issues and strategies. Mass Spectrometry Reviews 2006;25:235–254. [PubMed: 16284939]
27. Shen X, Tseng GC, Zhang X, Wong WH. On (psi)-learning. Journal of the American Statistical Association 2003;98(463):724–734.
28. Storey JD. A direct approach to false discovery rates. Journal of the Royal Statistical Society 2002;64:479–498.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

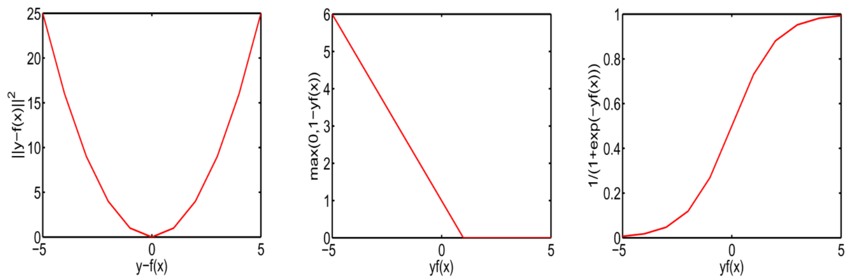
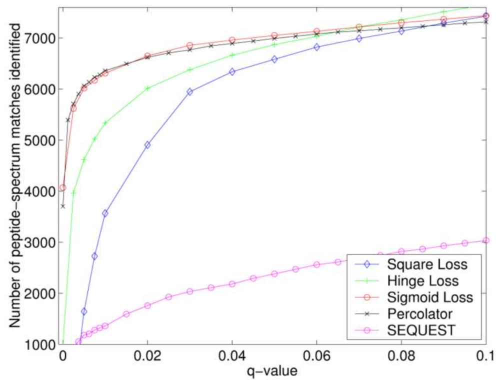


Figure 1. Three types of loss function

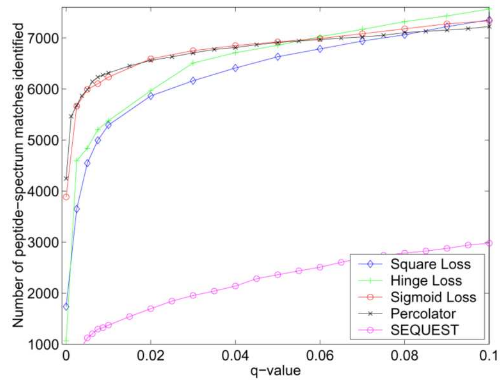
Each panel plots the loss as a function of the difference in the true and predicted label. The squared loss $L(f(x), y) = (f(x) - y)^2$ is often used in regression problems, but also in classification [22]. The hinge loss $L(f(x), y) = \max(0, 1 - yf(x))$ is used as a convex approximation to the zero-one loss in support vector machines [8]. The sigmoid loss $L(f(x), y) = 1/\exp(1 + f(x))$ is perhaps less commonly used, but is discussed in, e.g., [23,27].

NIH-PA Author Manuscript

NIH-PA Author Manuscript



(A) Training set



(B) Test set

Figure 2. Comparison of loss functions

Each panel plots the number of accepted PSMs for the yeast (A) training set and (B) test set as a function of the *q* value threshold. Each series corresponds to one of the three loss functions shown in Figure 1, with series for Percolator and SEQUEST included for comparison.

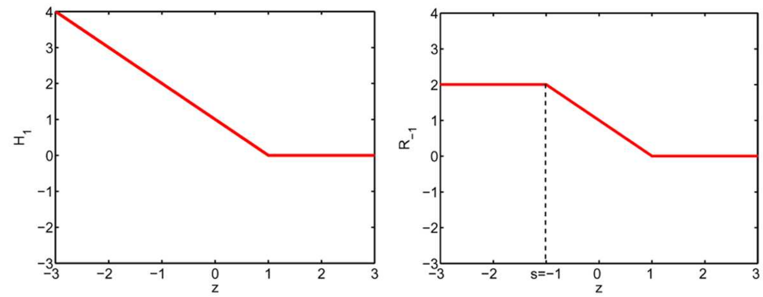


Figure 3. "Cutting" the hinge loss makes a sigmoid-like loss called the *ramp loss*

Making the hinge loss have zero gradient when $z = y_i f(x) < s$ for some chosen value s effectively makes a piece-wise linear version of a sigmoid function.

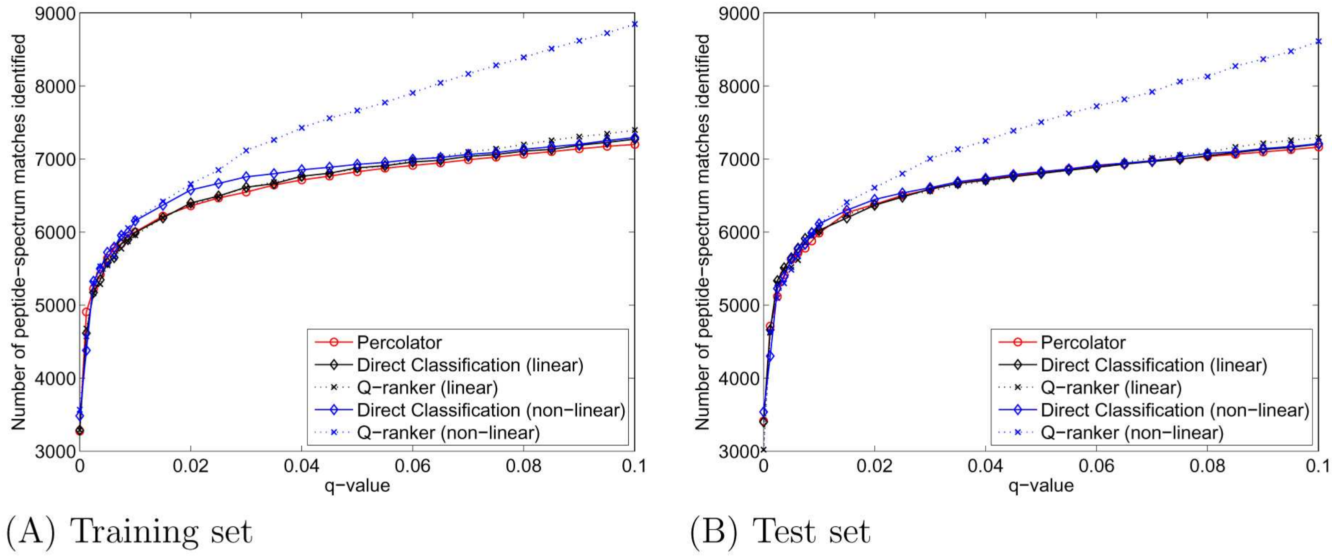
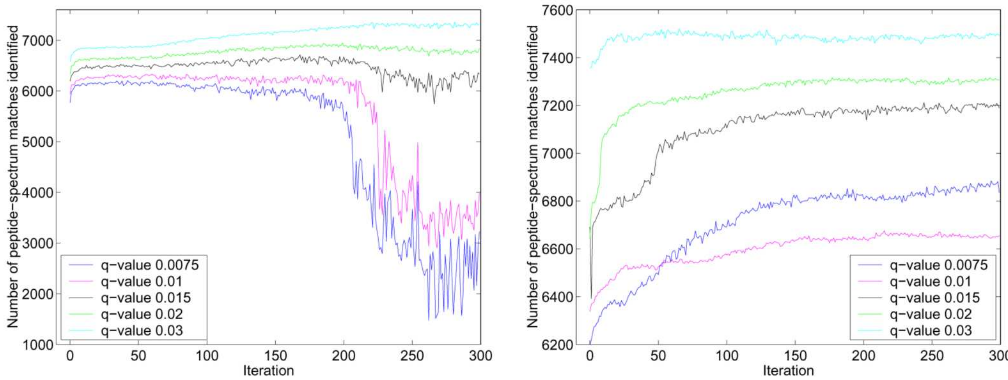


Figure 4. Comparison of Percolator, direct classification and Q-ranker

The figure plots the number of accepted PSMs as a function of *q* value threshold for the yeast data set. Each series corresponds to a different ranking algorithm, including Percolator as well as linear and nonlinear versions of the direct classification algorithm and Q-ranker. The nonlinear methods use 5 hidden units.



(A) Direct classification (B) Q-ranker

Figure 5. Comparison of training optimization methods (iteration vs. error rate)

The Q-ranker optimization starts from the best result of direct optimization achieved during the course of training and continues for a further 300 iterations. These results are on the training set. Note that for each *q* value choice, Q-ranker improves the training error over the best result from the classification algorithm.

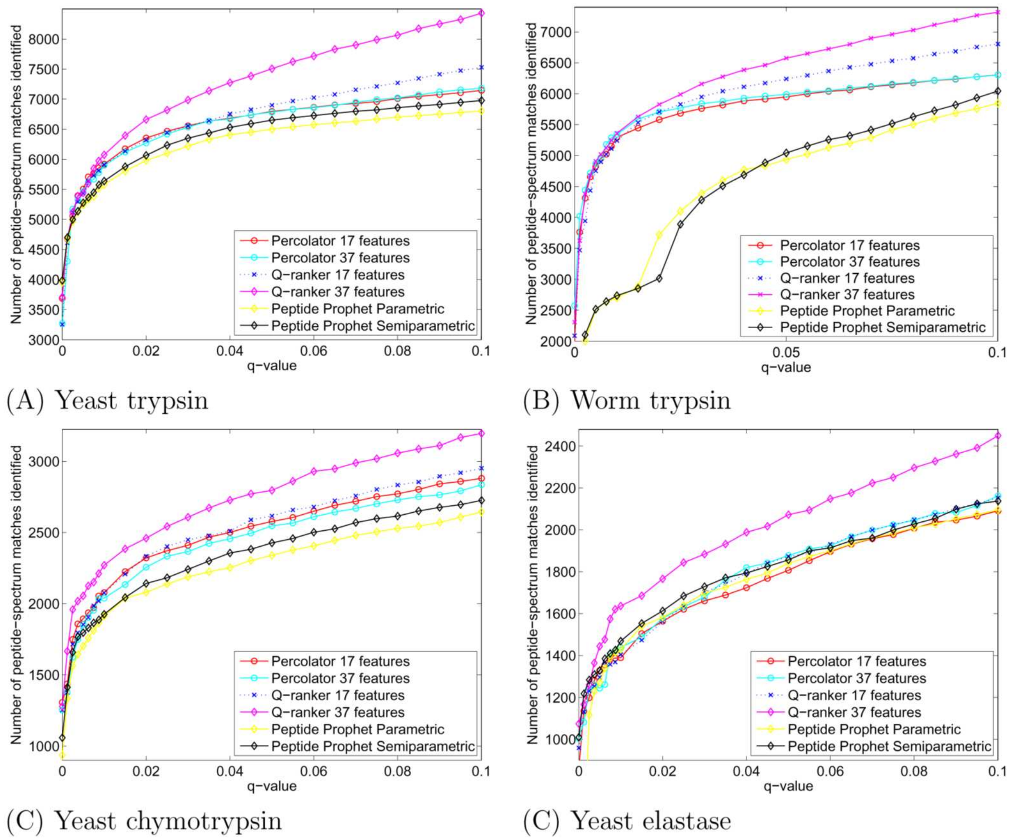


Figure 6. Comparison of PeptideProphet, Percolator and Q-ranker on four data sets

Each panel plots the number of accepted target PSMs as a function of *q* value. The series correspond to the three different algorithms, including two variants of Q-ranker that use 17 features and 37 features.

NIH-PA Author Manuscript

NIH-PA Author Manuscript

Table 1

Features used to represent PSMs

The first ten features are computed by SEQUEST. Features 18–37 are used in Section 3.6.

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

NIH-PA Author Manuscript
NIH-PA Author Manuscript

Table 2

Comparison of Percolator and Q-ranker on 17 and 37 feature data sets

Each entry in the table indicates the number of accepted PSMs for the given algorithm (column) on the given data set at the given specified *q* value (row). Entries in boldface indicate that this algorithm performed better than the other algorithm for this data set and *q* value threshold.

		17 features		37 features	
Data set	***q* value**	**Percolator**	**Q-ranker**	**Percolator**	**Q-ranker**
Yeast trypsin	0.01	**5917**	5885	5983	**6072**
	0.05	6793	**6940**	6813	**7501**
	0.1	7168	**7610**	7200	**8430**
Yeast elastase	0.01	**1389**	1380	1491	**1615**
	0.05	1806	**1851**	1958	**2140**
	0.1	2103	**2196**	2301	**2561**
Yeast chymotrypsin	0.01	2077	**2086**	2158	**2312**
	0.05	2576	**2620**	2680	**2844**
	0.1	2914	**2961**	3057	**3214**
Worm trypsin	0.01	**5116**	5031	5192	**5238**
	0.05	5864	**6119**	5830	**6419**
	0.1	6169	**6730**	6146	**7128**

Table 3

Q-ranker successfully optimizes the specified *q* value

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

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

Algorithm 1

The Q-ranker algorithm

The input variables are the training set X of PSM feature vectors, the corresponding binary labels Y, indicating which PSMs are targets and which are decoys, the set Q of specified q values, the set $\hat{Q}$ of threshold q values and the number n of training iterations. The `chooseRandom` subroutine selects a random positive or negative (depending on the first, Boolean parameter) example x that satisfies $|f(\mathbf{x})| < \varepsilon$. The `gradientStep` subroutine makes a gradient step to satisfy the constraint $f(\mathbf{x}^+) > f(\mathbf{x}^-)+1$. The algorithm returns the learned weight vector w.

---

procedure Q-ranker(X, Y, Q, $\hat{Q}$, n)
  $w \leftarrow$ initialize using direct classification ▷ Solve the direct classification problem.
  for $q_t \in Q$ **do**
    for $q \in \hat{Q}$ **do**
      $t \leftarrow$ compute Threshold(X, Y, w, q) ▷ Calculate the threshold corresponding to q
      $\varepsilon \leftarrow 2 * |\{x \in X | f(\mathbf{x}) > t\}|$
      for i larr; 1 … n **do**
        $x^+ \leftarrow$ chooseRandom(TRUE, X, Y, w, ε) ▷ Randomly select a pair of examples
        $x^- \leftarrow$ chooseRandom(FALSE, X, Y, w, ε)
        $w \leftarrow$ gradientStep(w, $f(\mathbf{x}^+)$, $f(\mathbf{x}^-)$) ▷ Update the weights.
      end for
    end for
    Record best result on q_t
  end for
  return (w)
end procedure
