Positive unlabeled learning for deriving protein interaction networks

Cumhur Kılıç · Mehmet Tan

Received: 15 January 2012/Revised: 14 May 2012/Accepted: 23 May 2012/Published online: 21 June 2012
© Springer-Verlag 2012

Abstract Binary classification is the process of labeling the members of a given data set on the basis of whether they have some property or not. To train a binary classifier, normally one needs two sets of examples from each group, usually named as positive and negative examples. However, in some domains, negative examples are either hard to obtain or even not available at all. In these problems, data consist of positive and unlabeled examples. This paper first presents a survey of algorithms which can handle such problems, and then it provides a comparison of some of these algorithms on the protein–protein interaction derivation problem by using the available (positive) interaction information.

1 Introduction

Binary classification problems include two groups of examples. The first group includes the examples which possess some certain characteristics and referred to as positive class. The negative class, on the other hand, may include all the other examples. An example may be unlabeled, in which case we do not know whether it is positive or negative. The goal of a binary classifier is to correctly label each of these unlabeled examples, based on the insight gained from the given positive and negative examples.

Supervised learning algorithms, in general, use positive and negative examples for training. But, in most of the domains, compared to positives, acquiring negative examples is more costly and sometimes even not possible. In such cases, an algorithm that only exploits positive and unlabeled examples is needed. A set of methods called Positive Unlabeled (PU) learning algorithms [4] try to achieve this task of learning in the absence of negative examples.

Binary classification is used in different applications like gene regulatory networks derivation, symptom–illness relations, and text or webpage classification. In this paper, we use PU learning to reveal protein–protein interaction (PPI) networks. A PPI network can be represented as a graph where a node in the graph represents a protein, and an edge represents the presence of an interaction between proteins. By means of laboratory experiments it is relatively easy to learn an interaction that takes place between two proteins. But it is much harder to know that they do not. Because even if we have not observed their interaction during an experiment, this does not necessarily prove that they will not interact in another environment or time.

We can describe the nature of the problem we consider as follows:

1. In our data set, we only have positive and unlabeled examples. Negative examples are not initially available. Unlabeled data set may include both positive and negative examples.
2. In the unlabeled set, number of negative examples is expected to be larger than positive examples. Within all possible pairs, only a small portion of protein pairs interact.

A related domain of problems is the semi supervised learning (SSL). SSL includes the set of problems where it is hard to obtain labeled examples (both positive and

C. Kılıç · M. Tan (✉)
Department of Computer Engineering, TOBB University
of Economics and Technology, Ankara, Turkey
e-mail: mtan@etu.edu.tr

C. Kılıç
e-mail: ckilic@etu.edu.tr

negative). Therefore, in this case, we have a large number of unlabeled examples in addition to the small number of labeled examples from both classes. PU learning can be considered as a subcategory of SSL, but the algorithms for SSL and PU learning differ as there are no negative examples in PU learning, which makes the problem harder and brings the necessity to design the algorithm to compensate for the absence of negative examples. SSL algorithms is out of scope of this paper, interested reader can find more details and algorithms for SSL in [27, 29–31].

In this paper, we summarize and classify the existing PU learning algorithms. While some of the algorithms we cover in this paper are already suitable to be applied for PPI network derivation, others are not directly applicable because of their domain-specific steps as most of them are designed for text classification. But we included these algorithms in the list of methods given in Sect. 2 as well, for the sake of completeness. A paper by Zhang et al. [28] also surveys some PU learning algorithms but our paper is more comprehensive and their paper lacks a comparative evaluation.

We perform a high-level overview of each algorithm to describe how it works, how they are different from others and in which settings they're designed to work. Then, we compare eight PU learning algorithms depending on their results for PPI Network derivation. To the best of our knowledge, this is the first time PU learning algorithms are used to derive protein interactions based on the available biological data.

The organization of the paper is as follows. In Sect. 2, we group and describe each algorithm in detail. In Sect. 3, we report the experimental results. In Sect. 4, we give a brief summary of our contributions and list some future research directions.

2 PU learning

PU learning algorithms are designed for settings without negative examples. But when classifying unlabeled examples, an algorithm should know characteristics of positive and/or negative examples. What makes a PU learning algorithm different from others is the way it follows when learning characteristics of classes. In this paper, we classify the algorithms depending on the method they use to deal with the absence of negative examples for extracting the characteristics. Throughout the process, almost all the methods use classical supervised classification methods such as a support vector machine (SVM) or a logistic regression classifier.

In this section, algorithms of two main approaches will be discussed: (1) Two-step strategies which first extract some reliable negative examples, and then apply a learning

method, (2) one-step strategies that use the positive and unlabeled examples directly to classify new examples. Throughout the paper, unless otherwise stated, P is used for the set of positive examples, N for negative examples and U for unlabeled examples. TP, FN, FP, TN are the number of true positives, false negatives, false positives and true negatives, respectively.

2.1 Two-step algorithms

The algorithms that first extract some negative examples from the unlabeled set are called two-step algorithms. These two steps are:

1. Finding a set of reliable (strong) negative examples (examples that have a high probability to be a negative) from the unlabeled set.
2. Training one or a series of classifiers and making classifications with this/these classifiers.

There exist a large number of two-step algorithms. The most primitive method is to consider all unlabeled examples as negative. A classifier trained with these negative examples and positive data set may do some correct classifications since most of the unlabeled examples were already negative. On the other hand, since there are also positives in the unlabeled set, this classification may result in many false positive or false negatives. We implemented this method with the name SVM_{only} and showed its results in Sect. 3. Algorithms we deal with in this subsection extract negatives from the unlabeled set in more elegant ways.

2.1.1 Algorithm of Carter et al.

Algorithm of [1] first labels all the examples in U as negative. It then trains a classifier with this negative set and P . But in our setting, U is much larger than P . Therefore using U in training stage with P may lead to a weak classifier, because a classifier is usually more successful with a balanced set of training data.

To overcome this problem, Carter et al. separated U into n different unlabeled subsets which have the same size as the positive set ($n = 5$ in authors test with *E. coli* data set). They used unlabeled subsets with P to create n different data sets. All of these data sets are independently used for training and prediction. This process is similar to the strategy of Bagging SVM [16].

On the other hand, this process is not performed by the algorithm itself. Authors ran the program for each data set. What the algorithm actually does is to use U as N without using a measure to select negatives from unlabeled set.

While working on a data set, first algorithm performs prediction by using unlabeled set as negative set. Since

algorithm uses this negative set for training and then tests it with result classifier, leave-one-out cross-validation (LOOCV) is performed.

After the first prediction, the algorithm purifies its negative set by removing the examples which are predicted positive. Finally, LOOCV is repeated with original positive set and purified negative set.

Selection of negative examples randomly from U can be regarded as an effective way since most of the examples in the unlabeled data set are negative examples. There is a high chance of creating an unlabeled data set that consists of negative examples. But it is also possible to have positive examples in the generated negative set. This may result in a classifier which is unable to select class boundaries and labels of examples correctly. The following algorithms try to perform selections based on some data dependant measures to avoid this.

2.1.2 Positive sample only learning (PSoL)

PSoL [2] selects some negative examples from the unlabeled examples, using Euclidean distance, a technique called Maximum Distance Minimum Redundancy (MDMR) [24] and iterative usage of SVM classifiers. These selected examples are considered as strong negative examples. Then, the algorithm uses this new negative set with positive set to classify the rest of the unlabeled examples. Since PSoL selects its negatives using these techniques, it has a greater chance of selecting correct negative examples than selecting them randomly.

PSoL has 3 steps in it: Selection of initial negatives, expansion of negative example set, classification with positive and negative examples.

First, algorithm calculates the distance between every unlabeled example and every example in P . Then, it finds the unlabeled example with the greatest total distance to the examples of P . This unlabeled example is extracted from U and selected as the first negative example. Starting with this negative example, PSoL iteratively selects more negatives from the unlabeled data. At each iteration, it adds one unlabeled example that satisfies Eq. 1 to the negative set.

$$\max_{x_i \in U} [\min_{x_j \in P} d(x_i, x_j) \times \sum_{x_k \in N} d(x_i, x_k)] \quad (1)$$

Algorithm extracts unlabeled examples to the negative set via 1. If there are negative examples in the unlabeled set, we can expect them to be located far from the positives. Also, by maximizing the distance to previous negative examples, the algorithm creates a negative set that can represent all negative examples in the dataset.

After the selection of initial negatives, PSoL iteratively trains an SVM classifier with positive and negative sets.

Classifier calculates decision function results for unlabeled examples and adds K of them to the negative set. To make sure it is expanding the set with strong negatives, it extracts examples with results less than a threshold (T). Also, if there is a serious size difference between two classes, classifiers tend to label most unlabeled examples as one large class. So, to prevent sudden expansion of the negative set, amount of examples to extract at each iteration is also limited by $K = |P| \times r$. T is set to -0.2 and r value is set to 3 in the algorithm.

Iteration ends at the point when no unlabeled example get necessary decision values to relocate them to the negative set. At this point, PSoL has new negative, positive and unlabeled data sets. By using these data sets, it trains a classifier and uses it to classify remaining unlabeled examples.

Like PSoL, The Rocchio technique [5] also uses a distance measure to compare unlabeled examples when selecting strong negatives. The difference is, while PSoL compares each pair of examples in the sets, Rocchio method creates a positive and a negative prototype to compare them with unlabeled examples.

2.1.3 The Rocchio technique and SVM (Roc-SVM)

The algorithm in [5] uses the Rocchio method for selecting strong negatives. In the second step, it uses SVM to create a classifier with selected negative examples and already known positive examples.

In the first step Rocchio method selects strong negatives with its own classifying technique (Rocchio Classifier). Briefly, construction of this classifier is performed by defining prototype vectors for both P and U . Prototype vectors are called $\vec{c}^+$ and $\vec{c}^-$, respectively. After prototypes are created, similarity of every sample to both prototypes are calculated. Examples which are more similar to $\vec{c}^-$ than $\vec{c}^+$ are labeled as negatives and extracted to the set of selected negatives (RN). For similarity measure, cosine measure is used.

Roc-SVM uses the following equations to define prototype vectors, where $\alpha = 16$ and $\beta = 4$ in [5];

$$\vec{c}^+ = \alpha \frac{1}{|P|} \sum_{\vec{d} \in P} \frac{\vec{d}}{\|\vec{d}\|} - \beta \frac{1}{|U|} \sum_{\vec{d} \in U} \frac{\vec{d}}{\|\vec{d}\|} \quad (2)$$

$$\vec{c}^- = \alpha \frac{1}{|U|} \sum_{\vec{d} \in U} \frac{\vec{d}}{\|\vec{d}\|} - \beta \frac{1}{|P|} \sum_{\vec{d} \in P} \frac{\vec{d}}{\|\vec{d}\|} \quad (3)$$

In the second step, we have P and a set of RN . Starting with these two sets, the algorithm extracts negative examples from unlabeled set in an iterative manner. At each iteration algorithm trains an SVM classifier and tests unlabeled set with it. The examples which are predicted as

negative are extracted to the negative set. Iteration ends when no more example can be extracted.

After iteration ends, the algorithm uses the classifier constructed at the last iteration (C_{last}) to label P . If more than 5% of P are labeled as negative, the algorithm chooses the classifier constructed at the first iteration (C_{first}) as the final classifier. Otherwise, it chooses C_{last} . The final classifier can be used to label any unlabeled example.

In this algorithm, Rocchio classifier is used for labeling when there is only unlabeled and positive data. On the other hand, the algorithm uses SVM classifier when there is positive, negative and unlabeled data (after initial negative set is generated). Authors of [5] attribute this to the fact that both classifiers get better results in their own kind of data.

There is an optional process in this algorithm. During the classification stage, some of the positive examples may be incorrectly marked as negatives (false negative). To find these examples, clustering is applied on RN . Every cluster is used with P for training a classifier using Rocchio method again. Number of these clusters effects the success of the process.

While algorithms like PSoL [2] and Rocchio [5] use feature vectors of examples with distance measures, some other algorithms like PN-SVM [6] and M-C [3] use features frequencies in examples.

2.1.4 Positive-negative document enlarged classifier (PN-SVM)

PN-SVM [6] is an algorithm designed to be able to work with data sets where percentage of positive examples is very low. It consists of three stages. At the first stage it extracts strong negatives from unlabeled set by using its knowledge about positive examples. At the second stage it starts extracting both positive and negative examples iteratively. Finally, it trains an SVM classifier with positive and negative examples it collected so far and labels the rest of the unlabeled set.

PN-SVM, first of all, performs feature normalization on the data set. After normalization, it computes the frequency of features' appearances in the examples of P . This process creates the core vocabulary of P , where core vocabulary contains the features which have strengths greater than a threshold.

PN-SVM finds the unlabeled examples which contain the fewest of all features from the positive core. These examples are called strong negatives. Set of strong negatives do not contain all of the negative examples in the unlabeled set. Some other negatives may have some of the features which are in the core vocabulary of positive class.

After strong negatives are extracted from the unlabeled set, it extracts positive and negative examples from the

unlabeled set iteratively. An important difference of this algorithm is while most of the PU algorithms only extract negative examples from U , PN-SVM also extracts positives. This makes PN-SVM a suitable algorithm for data sets with very few positive examples.

Since PN-SVM is for settings where the negative class consists of multiple sub classes, the negative set can be separated into clusters by their feature vectors. The algorithm applies k -means clustering to N , depending on the feature similarities of its examples. To select the positive and negative examples that will be extracted, the algorithm compares every example with the centroid of clusters. Like Rocchio method [5] constructs positive and negative prototypes to represent P and N , PN-SVM uses centroids of the clusters it constructs to compare with unlabeled examples. Results of these comparisons are used to decide which examples will be extracted and to which set they will be extracted to.

Clustering of N is important for the algorithm. Using the whole N without clustering may result in incorrect predictions. For example, consider an example e which is very similar to the centroid of a single subset of N (C_{Ni}) and not similar to the other subsets of N . If the algorithm compares e with the centroid of the whole N (C_N) and the centroid of P (C_P), e may be more similar to C_P than C_N . Therefore e would be incorrectly classified as positive. PN-SVM clusters N to handle each negative subset separately. In this way, examples which are significantly close to the centroid of a single negative subset (like e) can be selected as negative.

To label an example d as positive, two following conditions should hold:

$$S(d, C_P) > \frac{1}{|P|} \sum_{d_i \in P} S(d_i, C_P) \quad (4)$$

$$S(d, C_P) - \max_{j=1, \dots, k} S(d, C_{Nj}) > \frac{1}{|P|} \sum_{d_i \in P} \left(S(d_i, C_P) - \max_{j=1, \dots, k} S(d_i, C_{Nj}) \right) \quad (5)$$

where k is the number of clusters and S is the similarity function. Similarly, to label an example d as negative, two following conditions should hold:

$$S(d, C_N) > \frac{1}{k} \sum_{i=0}^k \left(\frac{1}{|N_i|} \sum_{d_j \in N_i} S(d_j, C_{N_i}) \right) \quad (6)$$

$$\max_{i=1, \dots, k} S(d, C_{N_i}) - S(d, C_P) < \frac{1}{k} \sum_{i=0}^k \left(\frac{1}{|N_i|} \sum_{d_j \in N_i} (S(d_j, C_{N_i}) - S(d_j, C_P)) \right) \quad (7)$$

Examples which does not satisfy these conditions are not labeled at this step and left unlabeled.

Positive and negative examples which are extracted in the first and second stages of the algorithm are used with the original P to train a classifier. Using this classifier algorithm labels rest of the U .

Later, PN-SVM has been revised by its authors and Positive examples and Negative examples Labeling Heuristic (PNLH) [7] algorithm is introduced.

2.1.5 Positive examples and negative examples labeling heuristic (PNLH)

PNLH Algorithm [7] is an extension of PN-SVM [6]. It extracts positive and negative examples from unlabeled set using the core vocabulary and clustering strategies like PN-SVM. But there are important differences between two algorithms.

In PN-SVM algorithm, when a feature is selected to be put in the core vocabulary, feature's strength should be greater than a previously defined threshold. But it does not make sense to select a threshold without processing the data set. In PNLH, this threshold is calculated by the algorithm during run time. Average strength of the features is used as the threshold.

Another difference is that, clustering amount is also predefined in PN-SVM, but in PNLH it is automatically calculated. When the data set is going to be clustered into k clusters, k value should be optimum for this data. A large or small k value may result in incorrect labeling. In PNLH, k depends on the cardinality of P and N :

$$k = \sqrt{\frac{|N|}{|P|}} \quad (8)$$

Third difference is, when extracting positive and negatives iteratively from the unlabeled data set, number of features in domain is taken into account in PNLH. In this paper, it is claimed that reducing the number of features decreases false positives and false negatives. Therefore, before centroids are chosen, the algorithm first performs feature selection (selects features that will be used in this process). When selecting features for creating the core vocabulary, PNLH chooses top n features (where n is predefined).

In PNLH algorithm, when strong negatives are being selected, selection is done using the strength of features. For example, even if examples a and b have the same number of features from core vocabulary, their probability of being positive can be different. So, PNLH finds out the one with the strongest features. If a has stronger features than b , a has a higher chance to be a positive example. As a result, a will not be selected as strong negative.

Finally, while PN-SVM trains an SVM classifier in its third stage, PNLH is considered as a two stage algorithm and the type of classifier used afterwards is not important.

Another algorithm using feature frequencies is Mapping-Convergence (M-C) algorithm [3]. Like PN-SVM and PNLH, it finds the features which occur in the examples of P more frequently than they occur in the examples of U .

2.1.6 Mapping-convergence (M-C) algorithm

In [3] Positive Example Based Learning (PEBL) framework is introduced. This framework uses Mapping-Convergence (M-C) Algorithm which includes two stages: Mapping stage (for selecting strong negatives) and convergence stage (for iterative classification).

In the mapping stage, algorithm splits unlabeled data set into two sets: set of strong negatives N_1 and set of other examples P_1 . An example is called strong negative if it has a high probability of being a negative example. To find strong negatives, algorithm calculates each feature's frequency in positive set (f_p) and frequency in unlabeled set (f_u). All features with a f_p/f_u value lesser than a threshold are used for constructing a classifier. The examples which are predicted negative by this classifier are selected as strong negatives. In [3], it is stated that the threshold should be adjusted in order to ensure the classifier does not generate false negatives.

In the convergence stage, the algorithm iteratively extracts negatives from P_i . At the beginning of every iteration, it takes examples in N_i to the set of strong negatives (NEG). At this point, the algorithm has a set of known positives (POS), NEG, and set of unlabeled examples (P_i). It uses SVM with POS and NEG to define a boundary between positive and negative examples in the feature space. This boundary is chosen to keep a maximal margin between positives and negatives. The algorithm applies the boundary on P_i and this divides P_i into two parts: Part that includes the examples that should be labeled as negative and part that includes examples which still can not be labeled. First part is moved to set N_{i+1} and second part to P_{i+1} .

Iteration continues until no example can be moved to N . At the end, P_{last} contains positive examples, and NEG contains negative examples.

2.1.7 Augmented expectation maximization (A-EM)

The idea behind the A-EM [8] is very different from the other algorithms we cover in this subsection. It is designed to create an algorithm which can work in an environment where known positive (labeled) examples may be non-identical to the positives in the unlabeled data set. Most of the two-step algorithms assume they are identical, but in real life problems, the opposite may be observed. A-EM algorithm can be used in both settings.

Another difference is that, in addition to positive and unlabeled examples, this algorithm uses another set, set of irrelevant examples (O). This set should consist of examples which are not related to the positive class. In this case we expect this set to contain almost no positive examples. This set will be used to decrease noise in unlabeled data set by increasing the negative example density. For example, data used in [8] consists of P (web pages of a single product type from a single commercial Web site), U (web pages of all products from other commercial web sites) and O (20 Newsgroup and Reuters).

This algorithm joins U and O and creates a joint negative data set (N). Then it trains a Naive Bayesian Classifier (NB-C). A-EM works with EM (which is also used in S-EM algorithm). In new iterations of EM, we expect new classifiers to be better at extracting positive examples from N . But of course this depends on the performance of the classifier at the first iteration. This is why A-EM tries to increase the performance at the first stage by using O . Irrelevant set decreases noise in U , so the algorithm expects even the first classifier to be successful.

EM generates a series of classifiers and then the algorithm selects a final classifier from the series. The following value (F) is calculated for each classifier to compare their results:

$$F = \frac{2 \times TP}{(TP + FP) + (TP + FN)} = \frac{2/TP}{|CP| + PD} \quad (9)$$

where $TP + FP$ is the number of examples labeled positive (CP) and $TP + FN$ is the size of U (PD).

Algorithm uses the change in F to select the final classifier. It uses the following equation to calculate the change in F at the iteration i .

$$\Delta_i = \frac{F_i}{F_{i-1}} = \frac{TP_i}{TP_{i-1}} * \frac{|CP_{i-1}| + PD}{|CP_i| + PD} \quad (10)$$

The last classifier which increases F is selected as the final classifier. For example, if the classifier at iteration n is the last classifier with Δ greater than 1, this classifier is selected as final.

While both CP and PD numbers are known, since algorithm does not know which unlabeled examples are originally positive, TP_i/TP_{i-1} should be estimated. To do this, the algorithm first chooses features which can represent the positives (keywords) and creates a set K with these features. Then it estimates TP_i/TP_{i-1} as;

$$\sum_t^{[K]} N(f_t, d_i), d_i \in CP_i / \sum_t^{[K]} N(f_t, d_i), d_i \in CP_{i-1} \quad (11)$$

where $\sum_t^{[K]} N(f_t, d_i), d_i \in CP_i$ is the total number of keywords in CP_i . So when two CPs are compared, if one has more features from K in its examples, it is considered to contain more true positives.

It is reasonable to ask why A-EM does not use P and O to train a classifier, since O does not contain almost any positives. The problem of this idea is that irrelevant set can be very different from real negatives, so a classifier trained by P and O may be unable to extract negatives (and therefore positives) from U successfully.

Like A-EM, another algorithm that uses a Naive Bayesian Classifier is PU Learning by Generating Negative Examples (LGN) [9].

2.1.8 PU learning by generating negative examples (LGN)

LGN [9] algorithm creates a single artificial negative example (A_n) by using entropy and then uses this example with P to create a Naive Bayesian Classifier (NB-C).

NB-C calculates an example's probabilities to be a member of the positive and negative class. These probabilities are shown as $Pr(d|c)$, where d is an example and c is the class (+ or -). If, $Pr(d|+) > Pr(d|-)$, d is labeled as positive, or vice versa.

When NB-C calculates $Pr(d|c)$ value of an example, it needs two different values. First one is the conditional probabilities of $Pr(f|+)$ and $Pr(f|-)$, where f is a feature. Every feature has a probability to be examined in the examples of P and N ($Pr(f|c)$). For example, if features of an example has high probabilities to be examined by examples of P , this example may have a high probability to be a positive example. The second type of needed value is the prior probabilities of $Pr(+)$ and $Pr(-)$. To calculate these two type of values, one needs examples of both classes. Since we do not have any negative examples, both values should be estimated.

First, the algorithm creates an artificial negative example (A_n) which it constructs according to every feature's estimated $Pr(f|-)$ value. To estimate $Pr(f|-)$ values, LGN uses entropy on feature frequencies in positive and unlabeled examples.

$$\text{entropy}(f_i) = - \sum_{x \in \{+, -\}} Pr(f_i|x) \times \log(Pr(f_i|x)) \quad (12)$$

We name the features which mostly included by positive examples as positive features (f_+) and features that mostly included by negative examples as negative features (f_-). A positive feature can be seen in both P and U since there are positive examples in both P and U . On the other hand, a negative example can only be seen in U . LGN calculates entropy of feature frequencies and finds out which features are common in both classes and which are not. For example, if a feature has a high entropy, it means that this feature occurs in both P and U , therefore has a low chance to be a negative feature. Finally it calculates weights of features, which represent their $Pr(f|-)$ values, with the following equation:

$$\text{Weight}(f_i) = \frac{\text{entropy}(f_i)}{\max_{j=1,2,\dots,|V|} \text{entropy}(f_j)} \quad (13)$$

LGN constructs A_n according to weights of features. This means that if a feature has a high weight, it will be placed in A_n more times than other features.

After A_n is created, it uses P and A_n to calculate $\Pr(f|+)$ and $\Pr(f|-)$. The problem is, algorithm can not calculate prior probabilities of $\Pr(+)$ and $\Pr(-)$ with a single negative example. In the empirical evaluation section of the original paper, different prior probabilities are tested. Since results are virtually the same, $\Pr(+)=\Pr(-)=0.5$ is used.

Once we have a positive set and a negative example, the algorithm uses them for training an NB-C, with prior probabilities of $\Pr(+)$ and $\Pr(-)$ as 0.5. Finally, this classifier is used for classifying unlabeled examples in the data set.

In this subsection we covered 2 two-step algorithms that use entropy. The first one is LGN and the second one is Entropy-Based Semi-Supervised Learning (SLE) [10].

2.1.9 Entropy-based semi-supervised learning (SLE)

SLE [10] is an algorithm which is designed for environments where positive class consists of different sub classes. It first extracts strong positive and negative samples from U using entropy. Then, it trains a logistic regression classifier with these two data sets and previously known positives and label the rest of the examples.

SLE uses three basic processes. First one is feature extraction. By measuring frequency of features in the examples, feature weights are calculated. Second one, oversampling, is a type of resampling. When there are two or more classes in a data set and one has fewer examples than other one(s), oversampling is used on the class with the smaller set to balance the data set. In our case, positive class has much less examples than the unlabeled set. Third is based on entropy, which was also used in LGN algorithm [9]. For our problem, SLE calculates the entropy of unlabeled examples' posterior probabilities. Results of entropy calculation show each example's probability to be a member of each class. Therefore, it will be used for classification in SLE. Equation for entropy of unlabeled examples' posterior probability is defined as;

$$H(d_i) = - \sum_{j=1}^{|C|} p(c_j|d_i) \lg p(c_j|d_i) \quad (14)$$

where $p(c|d)$ is the class posterior probability of example d belonging to the class c ; $|C|$ is the number of known class labels in the training set.

At the first step, the algorithm extracts positive examples from unlabeled set. It creates an empty set for each subclass of P . Then it extracts the examples in U to the set of subclass that they have the highest posterior probability with. Since these sets may have different sizes at the end, algorithm reduces their sizes relatively to the size of the smallest set. Reducing is performed by removing the examples with the highest entropy. Finally union of these sets (S_p) is created.

In second step, a set of negatives (S_n) is extracted from the rest of unlabeled set. Algorithm extracts examples with the highest entropy to the S_n .

Before the final classification, the algorithm performs oversampling. After that, SLE trains a logistic classifier with positive ($P \cup S_p$) and negative (S_n) examples. This classifier is then used for labeling the rest of the unlabeled examples.

2.1.10 Annotating genes with positive samples (AGPS)

AGPS algorithm [11] is designed for gene function prediction by PU learning. In the first step, AGPS creates its data set by merging three different kinds of data: protein-protein interactions, protein complex data and gene expression data. But in this paper, we overview its method to handle PU problem regardless of the data it uses.

AGPS includes 3 steps; selecting strong negatives, expanding the negative set and final classification. But before all these steps, first it splits P into two sets; P_1 and P_2 . Then it adds P_2 to U and creates U_{new} . P_2 in U_{new} is used for evaluating the classifiers which will be constructed in second step.

To use all examples of P for evaluation, the algorithm performs 10-fold cross validation. It divides P into 10 subsets and use one of them for P_2 and rest for P_1 at each iteration. Therefore all three steps of the algorithm are repeated 10 times with the same U but different P_1 and P_2 sets.

At the first step, AGPS trains a 1-Class SVM with P_1 . Using this classifier it labels U_{new} . Examples which are predicted as negative are extracted from U_{new} to N (which is empty at the beginning).

In the second step, the algorithm iteratively expands the negative set by extracting new negatives from the unlabeled set. This time, at each iteration, it trains a 2-class SVM with current negative set and P_1 . It uses this classifier to predict rest of the unlabeled set. Examples which are predicted negative are extracted to N . Iterations ends when size of the unlabeled set reaches $|P|$.

At each iteration of second step, a new classifier is built and its training data and results are saved. So after the end of iterations, at the third step, the algorithm selects the best

classifier by comparing their success on predicting P_2 . The algorithm uses this classifier to label initial U_{new} .

At each iteration of the cross validation, algorithm labels all U . Therefore every iteration has a list of examples which are predicted as negative. At the end, the algorithm sorts these examples according to the number of times they are predicted as negative in 10 runs of cross validation. Then it selects the first $|P|$ examples as the best negative examples and creates a final negative set FN by extracting them from U . Finally it trains a 2-class SVM with P with FN and labels rest of the U .

2.2 One-step estimators

In this family of algorithms, negative examples will not be selected from the unlabeled list for binary classification. In this set of algorithms, knowledge gained from positive and unlabeled data is used directly. These algorithms mostly depend on similarity of unlabeled and negative sets (result of negative example's high ratio in the unlabeled set) and ability to gain characteristics of positive examples from available positive data.

2.2.1 PNB—positive naive bayesian

PNB algorithm [12] uses a naive Bayesian classifier which it modifies to make it work with positive and unlabeled data. This paper is based on document classification. Instead of analysing it this way, we can view documents as examples and words as features.

In addition to the positive and unlabeled data sets, the method gets a positive class probability $\hat{P}(1)$ as input. $\hat{P}(1)$ is an estimate of percentage of positive examples where $\hat{P}(0) = 1 - \hat{P}(1)$.

Second part of the algorithm calculates the probabilities of features to be a member of each class. Depending on the amount of times a feature is seen in positive examples, the algorithm calculates the importance of this feature for the positive class. When we have importance of a feature for positive class, it can be calculated for negative class as well.

PNB uses the following equation to compute the class label of an example d ;

$$\text{PNB}(d) = \arg \max_{c \in \{0,1\}} \hat{P}(c) \prod_{i=1}^{i=n} \hat{\Pr}(w_i|c) \quad (15)$$

Later, PNB algorithm is revised. The name of the revised version of the algorithm is PNNB [13].

2.2.2 PNNB Algorithm

PNNB [13] is an extension of PNB [12]. This extension can be used with or without the presence of negative examples.

The algorithm calculates positive feature probabilities $\hat{\Pr}(w_i|1)$ (the probability of feature w_i to be in a positive example's feature vector) in the same way as in PNB algorithm, but it has a different approach for negative feature possibilities ($\hat{\Pr}(w_i|0)$).

There are two ways to calculate a feature's negative feature probabilities; a direct approach or an indirect approach. The indirect way is the way that PNB algorithm uses (see previous subsection).

In PNNB algorithm, the direct approach is implemented. This approach needs negative examples, so PNNB first finds negative examples inside the unlabeled set. The advantages of this algorithm is that, while indirect approach can only use knowledge it gets from unlabeled and positive examples, this approach can also use a negative set (N) in the learning stage. It calculates feature probabilities by combining both sources. In our case, which we do not have any negative examples initially, this algorithm is still usable since it can start with an empty negative data set which it will fill as iterations continue.

$$\hat{\Pr}(w_i|0) = (1 - \alpha)\hat{\Pr}(w_i|0, P, U) + \alpha\hat{\Pr}(w_i|0, N) \quad (16)$$

When calculating feature probabilities (16), weight of the negative data set depends on the α value (17).

$$\alpha = \frac{1}{2} \times \frac{|N|}{|P|} \times \frac{\hat{\Pr}(1)}{1 - \hat{\Pr}(1)} \quad (17)$$

α value is related to, most importantly, cardinality of negative data set. If there are too many negative examples, effectiveness of negative data set will be higher than positive and unlabeled data set. But for example, if there are no negative examples in data set, α value will be equal to 0, therefore effectiveness of the negative data set will be 0, too.

Besides PNNB, a second algorithm called PNCT is introduced in [13]. PNCT simply splits feature vector into two and constructs two PNNB classifiers.

2.2.3 PNCT algorithm

PNCT algorithm [13] is designed for conditions where there are very few positive examples. In this case, instead of depending on variety of positive examples, PNCT algorithm depends on the variety of features. It uses technique of using two classifiers at the same time which was introduced by Blum and Mitchell introduced in [19].

This algorithm can be used in settings where feature vector can be divided into two parts that are independent of each other. In addition, of course each of these parts should be adequate to be used for training a valid classifier.

At the beginning, the algorithm has unlabeled, positive and negative sets. For our problem negative set is initially

empty. Also, for every example, two feature vectors called f_{v_a} and f_{v_b} are available.

Then PNCT iteratively creates two PNNB classifiers: PNNB_a using f_{v_a} and PNNB_b with f_{v_b} (How PNNB classifier is created the sets used are explained in the previous subsection). Every time it trains classifiers, it predicts classes of unlabeled examples with these classifiers. And at every iteration, the algorithm chooses $|P|/\hat{P}(1)$ number of predicted examples and takes them to the set they belong to. These examples to be moved are selected by their probability degree. Classification process continues with construction of a new classifier with new P , U and N sets in the next iteration. When the algorithm labels every example, iteration ends.

PNB [12], PNNB and PNCT [13] are in the same algorithm family and have similar characteristics. Another one-step algorithm that uses feature frequencies is Biased-PrTFIDF algorithm.

2.2.4 Biased-PrTFIDF algorithm

Biased-PrTFIDF is the algorithm that was introduced in [14]. If we represent the probability of a positive example to be unlabeled p , positive class as C_+ , negative class as C_- , the following equations can be written:

$$\Pr[P|x] = \Pr[C_+|x](1 - p) \quad (18)$$

$$\Pr[U|x] = p \times \Pr[C_+|x] + \Pr[C_-|x] \quad (19)$$

where, $\Pr[P|x]$ is the probability that an example x is in P and $\Pr[U|x]$ is the probability it is in U .

Using Eqs. (18) and (19), one can show that,

$$\Pr[C_+|x] - \Pr[C_-|x] = ((1 + p)/(1 - p))\Pr[P|x] - \Pr[U|x] \quad (20)$$

where, $(1 + p)/(1 - p)$ is represented as b in this paper. The classification function, then, is given as;

$$f(x) = \text{sgn}(\Pr[C_+|x] - \Pr[C_-|x]) \quad (21)$$

We can combine (20) and (21) to find the classifier function as

$$f(x) = \text{sgn}(b \times \Pr[P|x] - \Pr[U|x]) \quad (22)$$

To be able to create this classifier, the algorithm needs class probabilities ($\Pr[P|x]$ and $\Pr[U|x]$) and b value.

To find the class probabilities of a given example, PrTFIDF technique [20] is used. PrTFIDF gets a set and a subset of it as parameter. It calculates the probabilities of examples in the set to be a member of the given subset by using feature frequencies in the documents. In our case, the algorithm runs PrTFIDF two times: $P \cup U$ as set and P as subset to learn probabilities of positive class, $P \cup U$ as set and U as subset to learn probabilities of unlabeled class.

Second part of the algorithm finds the b value. It can not be calculated easily since we do not know p . To find the optimum b , the algorithm selects the value that classifier gets the best result with. This process requires the ability to evaluate classification results. As the real class labels are assumed unknown, a special performance measure which is discussed in the next subsection is used (Eq. 26).

After p value and class probabilities are calculated, Biased-PrTFIDF algorithm creates the classifier function. This function is then used to classify each example in the data set.

2.2.5 Spy technique and the expectation-maximization (S-EM) procedure

The algorithm in [4] includes Spy Technique and EM algorithm (Dempster et al. 1977).

EM Algorithm uses a Naive Bayesian Classifier (NB-C) trained with positive and unlabeled data. First step of the algorithm, the expectation step, is for filling the missing data with expected values. Therefore, it is claimed to be a suitable algorithm for settings without negative examples. Expected values are calculated using existing data of examples. In maximization step, parameters are estimated.

The algorithm uses NB-C to make probabilistic predictions of unlabeled examples. After the prediction of each example, S-EM creates a new NB-C. Therefore, the label of next example will be predicted by a new and improved NB-C.

Since this process makes probabilistic predictions, the algorithm needs a threshold to compare example's classification probabilities with. Depending on the difference with the threshold, classes of examples should be decided. This is where Spy Technique is used; the selection of threshold value.

The algorithm selects a variable percentage of random positive examples and adds them to the unlabeled data set. These examples are called Spy Positives. After calculating every example's probabilities to be member of each class, it checks the results of spy positives. These results show how a positive example's result should be. The algorithm defines a threshold using these results. It compares the results of other examples with the threshold and decides which class they belong.

A similar approach is implemented by the algorithm PosOnly [17]. Both S-EM and PosOnly perform prediction on a subset of P and define thresholds depending on their probabilistic results.

2.2.6 PosOnly

PosOnly is a PU learning algorithm introduced in [17] and used for derivation of gene regulatory networks in [18].

Although this method has 2 steps, since it does not select negative examples to use them in classification, it is not considered as a two-step method. First, a probabilistic classifier calculates examples' probability of being labeled. Then, by using this data, it acquires their probability of being positive.

Let $y \in \{0, 1\}$ and $s \in \{0, 1\}$ be two random variables for being positive and labeled, respectively. For a random example, $s = 1$ if it is labeled, and $s = 0$ if it is unlabeled. Examples in positive set are labeled, so if $s = 1$ then it is certain that $y = 1$. On the other hand, unlabeled examples' ($s = 0$) real labels (y) may be either positive ($y = 1$) or negative ($y = 0$). In this method, first we should be able to calculate $\Pr(s = 1|x)$ value (probability of an example to be labeled or unlabeled). This will be done by a probabilistic classifier.

$\Pr(y = 1|x)$ is the probability of an example to be positive. With the assumption that positive examples are labeled randomly, the following equation can be written;

$$\Pr(y = 1|x) = \Pr(s = 1|x) / \Pr(s = 1|y = 1) \quad (23)$$

Detailed explanation of this equation can be seen in [17] and [18]. $\Pr(s = 1|y = 1)$ is a constant which should be calculated using the data set. There is more than one way to calculate this constant. But the exact value of this constant is the average value of $\Pr(s = 1|x)$ for x in V_p , where V is a set of random examples from the data set and V_p is the set of labeled examples in V .

$$\Pr(s = 1|y = 1) = \frac{1}{n} \times \sum_{x \in V_p} \Pr(s = 1|x) \quad (24)$$

Another estimation of $\Pr(s = 1|y = 1)$ which is used in [18] is:

$$\Pr(s = 1|y = 1) = \frac{\sum_{x \in V_p} \Pr(s = 1|x)}{\sum_{x \in V} \Pr(s = 1|x)} \quad (25)$$

After the algorithm calculates examples' observed class probabilities ($\Pr(s = 1|x)$) and constant $\Pr(s = 1|y = 1)$ value, it uses these values in Eq. 23 and finds real class probabilities ($p(y = 1|x)$). Finally, it compares these probabilities with a threshold, which is set to 0.5 in [18], and decides if this example should be labeled as positive or negative. If an example e has $\Pr(y = 1|e) > 0.5$, the algorithm labels it as positive, and negative otherwise.

2.2.7 Bagging SVM

PU learning algorithms use two sources. They have the positive set to learn the characteristics of positive examples and the unlabeled set for negative examples. While it is possible to directly analyze positives, they should learn about negatives indirectly from the unlabeled set. Since algorithms use unlabeled set as a reference of negative

examples, amount of positive examples in the unlabeled set is important for success of the classifier. If there are not too many positive examples in the unlabeled set, it will be less noisy and can be used to create a better classifier. But, one can not know a priori, the number of positive examples in the unlabeled set.

To solve this problem, Bagging SVM [16] algorithm is proposed. In this algorithm, instead of using the whole unlabeled set to create a single classifier, random subsets from the unlabeled set is created and a classifier is trained from each of the subsets. Finally, these multiple classifiers are merged. With this process, the algorithm creates variability in the classifiers which will be used to create the final classifier. Since subsets are created randomly, it is possible to create a subset which has no positive examples, or a subset which is filled with positive examples. As a result of the variability some successful classifiers may increase the quality of the final classifier.

Effect of the subset size (K) is examined in [16]. Results of the tests performed with different K shows that K value has a very little effect on the performance of the final classifier and $K = NP$ is considered as a safe choice in general.

This algorithm has two versions for inductive and transductive PU learning. In inductive bagging, after multiple classifiers are created, these classifiers are aggregated to create a final classifier which can be used to label any unlabeled example. In [16] it is proposed to aggregate classifiers by simple average. Transductive bagging algorithm creates multiple classifiers as well, but in this algorithm, when a classifier is trained with a subset, created classifier is not used in classification of this subset. So, the algorithm adds this classifier to the list of classifiers that will be used to predict the rest of unlabeled examples (unlabeled set / unlabeled subset which is used in training of this classifier). Therefore transductive bagging algorithm should be used in situations where the goal is only to label the elements of U .

2.2.8 Weighted logistic regression (W-LR)

We already discussed that using positive and unlabeled sets directly in training stage of PU learning may result in weak classifiers. So, one can see the unlabeled set as a negative set which is contaminated with positive examples.

W-LR algorithm [15] considers the unlabeled data set as a noisy negative set where positive examples are the cause of the noise. It uses a linear function to perform classification on this noisy data set.

First of all, the algorithm weights examples to handle high noise. Then, it performs logistic regression to acquire the linear function it should use for this data set. Finally, with this linear function, it labels examples in the data set.

An optimum regularization parameter is needed for logistic regression to prevent overfitting. In this paper, a performance measure which can be used with only positive and unlabeled data is introduced. Using this measure in Eq. 26, the algorithm selects its regularization parameter.

$$\text{Performance measure} = \frac{r^2}{\Pr[f(X) = 1]} \quad (26)$$

where, f is the classification function, r is recall value and can be calculated using only the accuracy of positive examples' results (27).

$$r = \Pr[f(X) = 1 | Y = 1] \quad (27)$$

3 Experimental results

In this section we compare eight PU learning algorithms based on their results on deriving PPI networks. We implemented PSol, Bagging SVM, AGPS, PosOnly, Roc-SVM, Algorithm of Carter et al. (we used the name Carter for this algorithm) and the algorithm we call SVM_{only}. SVM_{only} is included as the most naive algorithm that uses the whole U as negative set and labels U with 10-fold cross validation. For S-EM we used the implementation of the authors.¹

We tested 8 of the 19 algorithms covered in this paper. These are the algorithms which can be used with gene expression data without any changes. For others, we explained how they work so one can modify them or get inspired by them to create new algorithms.

PN-SVM, PNLH, PNB, PNNB, PNCT, Biased-PrTFIDF, M-C and LGN algorithms use frequencies of words (features) to label the documents that include these words. A-EM algorithm creates a series of classifiers and compares them to choose the best one as a final classifier. For this comparison process, A-EM needs frequencies of the words in the examples. Since in our data set feature vector consists of numeric data, these algorithms are not suitable for our data. SLE algorithm is designed for environments where positive set is constructed by different positive subsets but our positive set in PPI data does not contain subclasses. Finally, we excluded WLR from comparison, as WLR algorithm includes some points that are either not clear or not mentioned in the paper, and the implementation by authors is not available.

3.1 Datasets

We used the *Escherichia coli* gene expression data set provided by Faith et. al [21]. This data set includes 445 microarray expression profiles for 4,345 genes. For known protein interactions we used the IntAct database² [25].

¹ <http://www.cs.uic.edu/liub/LPU/LPU-download.html>.

² <http://www.ebi.ac.uk/intact/main.xhtml>.

Table 1 Comparison between PU Learning algorithms by precision

r	SVM _{only}	Carter	PSol _o	PSol _m	Roc - SVM	AGPS	S-EM	PosOnly	Bagging
10%	0.000 ± 0.000	0.155 ± 0.002	0.107 ± 0.105	0.267 ± 0.208	0.581 ± 0.569	0.251 ± 0.039	0.258 ± 0.047	0.144 ± 0.009	0.583 ± 0.589
20%	0.333 ± 0.653	0.150 ± 0.008	0.409 ± 0.421	0.484 ± 0.395	0.689 ± 0.330	0.203 ± 0.050	0.217 ± 0.008	0.141 ± 0.005	0.512 ± 0.512
30%	0.883 ± 0.142	0.141 ± 0.005	0.932 ± 0.076	0.691 ± 0.161	0.854 ± 0.050	0.248 ± 0.023	0.204 ± 0.006	0.126 ± 0.008	0.916 ± 0.027
40%	0.889 ± 0.218	0.127 ± 0.011	0.805 ± 0.041	0.376 ± 0.049	0.704 ± 0.037	0.197 ± 0.030	0.156 ± 0.010	0.125 ± 0.019	0.704 ± 0.090
50%	0.939 ± 0.119	0.110 ± 0.006	0.830 ± 0.137	0.482 ± 0.129	0.708 ± 0.026	0.136 ± 0.018	0.145 ± 0.003	0.112 ± 0.004	0.767 ± 0.049
60%	0.605 ± 0.107	0.101 ± 0.011	0.575 ± 0.173	0.300 ± 0.106	0.456 ± 0.098	0.130 ± 0.037	0.105 ± 0.008	0.169 ± 0.080	0.494 ± 0.104
70%	0.548 ± 0.065	0.070 ± 0.006	0.668 ± 0.176	0.202 ± 0.045	0.449 ± 0.105	0.115 ± 0.012	0.095 ± 0.018	0.116 ± 0.044	0.513 ± 0.160
80%	0.510 ± 0.261	0.051 ± 0.002	0.504 ± 0.120	0.181 ± 0.071	0.315 ± 0.050	0.079 ± 0.010	0.060 ± 0.001	0.089 ± 0.014	0.484 ± 0.269
90%	0.196 ± 0.060	0.024 ± 0.002	0.267 ± 0.088	0.089 ± 0.033	0.147 ± 0.058	0.033 ± 0.008	0.029 ± 0.013	0.038 ± 0.009	0.248 ± 0.104
avg.	0.545	0.103	0.566	0.341	0.545	0.154	0.141	0.118	0.580

Rows and columns stand for the ratio ($r = |P|/(|P| + |Q|)$) and the algorithm A. Each entry presents the average precision result of an algorithm at a certain ratio

Table 2 Comparison between PU Learning algorithms by recall

r	SVM _{only}	Carter	PSoL _o	PSoL _m	Roc-SVM	AGPS	S-EM	PosOnly	Bagging
10%	0.000 ± 0.000	0.993 ± 0.006	0.498 ± 0.549	0.514 ± 0.521	0.030 ± 0.033	0.576 ± 0.239	0.604 ± 0.087	0.948 ± 0.080	0.014 ± 0.015
20%	0.005 ± 0.010	0.948 ± 0.028	0.172 ± 0.307	0.210 ± 0.272	0.083 ± 0.057	0.497 ± 0.149	0.592 ± 0.062	0.963 ± 0.040	0.059 ± 0.058
30%	0.027 ± 0.024	0.931 ± 0.053	0.059 ± 0.030	0.170 ± 0.047	0.147 ± 0.049	0.714 ± 0.123	0.606 ± 0.028	0.941 ± 0.083	0.118 ± 0.030
40%	0.022 ± 0.012	0.931 ± 0.016	0.071 ± 0.016	0.180 ± 0.026	0.147 ± 0.033	0.549 ± 0.016	0.544 ± 0.049	0.820 ± 0.170	0.105 ± 0.028
50%	0.067 ± 0.019	0.939 ± 0.014	0.117 ± 0.032	0.205 ± 0.014	0.208 ± 0.031	0.688 ± 0.196	0.621 ± 0.014	0.850 ± 0.130	0.128 ± 0.027
60%	0.087 ± 0.026	0.953 ± 0.043	0.083 ± 0.046	0.167 ± 0.023	0.200 ± 0.074	0.557 ± 0.101	0.520 ± 0.030	0.553 ± 0.173	0.113 ± 0.046
70%	0.125 ± 0.023	0.916 ± 0.046	0.142 ± 0.009	0.204 ± 0.031	0.267 ± 0.026	0.671 ± 0.071	0.604 ± 0.077	0.648 ± 0.075	0.173 ± 0.054
80%	0.187 ± 0.065	0.940 ± 0.023	0.180 ± 0.060	0.240 ± 0.078	0.247 ± 0.069	0.707 ± 0.026	0.580 ± 0.000	0.660 ± 0.185	0.193 ± 0.052
90%	0.160 ± 0.045	0.920 ± 0.091	0.147 ± 0.069	0.187 ± 0.069	0.240 ± 0.091	0.627 ± 0.114	0.547 ± 0.232	0.683 ± 0.118	0.160 ± 0.045
avg.	0.075	0.941	0.163	0.231	0.174	0.621	0.580	0.785	0.118

Rows and columns stand for the ratio ($r = |P|/(|P| + |Q|)$) and the algorithm A. Each entry presents the average recall result of an algorithm at a certain ratio

Table 3 Comparison between PU Learning algorithms by F-measure

r	SVM _{only}	Carter	PSoL _o	PSoL _m	Roc-SVM	AGPS	S-EM	PosOnly	Bagging
10%	0.000 ± 0.000	0.268 ± 0.003	0.173 ± 0.170	0.203 ± 0.112	0.056 ± 0.061	0.338 ± 0.022	0.361 ± 0.060	0.250 ± 0.016	0.026 ± 0.029
20%	0.010 ± 0.019	0.259 ± 0.011	0.093 ± 0.127	0.160 ± 0.087	0.139 ± 0.090	0.287 ± 0.074	0.317 ± 0.018	0.246 ± 0.006	0.105 ± 0.104
30%	0.051 ± 0.046	0.245 ± 0.005	0.110 ± 0.053	0.266 ± 0.041	0.248 ± 0.070	0.366 ± 0.014	0.305 ± 0.010	0.221 ± 0.009	0.209 ± 0.046
40%	0.043 ± 0.022	0.223 ± 0.018	0.130 ± 0.026	0.241 ± 0.012	0.242 ± 0.044	0.289 ± 0.032	0.243 ± 0.015	0.216 ± 0.023	0.181 ± 0.043
50%	0.124 ± 0.033	0.197 ± 0.009	0.203 ± 0.045	0.285 ± 0.029	0.321 ± 0.040	0.225 ± 0.020	0.235 ± 0.004	0.198 ± 0.004	0.218 ± 0.039
60%	0.151 ± 0.042	0.182 ± 0.017	0.144 ± 0.075	0.212 ± 0.043	0.276 ± 0.088	0.210 ± 0.056	0.175 ± 0.012	0.244 ± 0.065	0.183 ± 0.068
70%	0.203 ± 0.034	0.130 ± 0.011	0.233 ± 0.006	0.201 ± 0.028	0.333 ± 0.045	0.197 ± 0.020	0.164 ± 0.030	0.194 ± 0.057	0.259 ± 0.081
80%	0.272 ± 0.104	0.096 ± 0.003	0.263 ± 0.074	0.206 ± 0.076	0.275 ± 0.057	0.142 ± 0.016	0.109 ± 0.001	0.155 ± 0.014	0.273 ± 0.092
90%	0.175 ± 0.048	0.047 ± 0.004	0.189 ± 0.080	0.120 ± 0.045	0.183 ± 0.071	0.062 ± 0.015	0.055 ± 0.024	0.072 ± 0.017	0.193 ± 0.063
avg.	0.114	0.183	0.171	0.210	0.230	0.235	0.218	0.200	0.183

Rows and columns stand for the ratio ($r = |P|/(|P| + |Q|)$) and the algorithm A. Each entry presents the average F measure result of an algorithm at a certain ratio

Table 4 p values of one-sided Wilcoxon signed rank test for each pair of F measure values of algorithms

	SVM _{only}	Carter	PSoL _o	PSoL _m	Roc-SVM	AGPS	S-EM	PosOnly	Bagging
SVM _{only}		0.91	0.99	0.97	0.99	0.93	0.91	0.93	1
Carter	0.11		0.5	0.89	0.87	0.99	0.99	0.84	0.5
PSoL _o	0.01	0.54		0.91	0.97	0.91	0.85	0.75	0.95
PSoL _m	0.04	0.12	0.10		0.84	0.71	0.5	0.24	0.37
Roc-SVM	0.004	0.15	0.04	0.18		0.5	0.41	0.15	0.005
AGPS	0.08	0.004	0.10	0.32	0.54		0.06	0.08	0.17
S-EM	0.10	0.004	0.17	0.54	0.63	0.95		0.21	0.32
PosOnly	0.08	0.18	0.28	0.78	0.87	0.93	0.82		0.41
Bagging	0.002	0.54	0.06	0.67	0.99	0.85	0.71	0.63	

If two proteins are known to be interacting, the example consisting of their expression profiles is a positive example. A feature vector consists of the expression data for the proteins in that sample. As the expression data has 445 samples for each protein, the length of a single feature vector is 890.

In the data set, positive examples consist of examples known to be interacting, downloaded from IntAct. We assumed all other protein pairs as (unlabeled) negative examples to evaluate the algorithms. Then, we randomly chose a subset from P and assumed that the examples in this set are unlabeled as in [18]. This new set is called Q . Therefore, P contains labeled positive examples, Q contains unlabeled positive examples and N contains unlabeled negatives. So our set of unlabeled examples consist of Q and N ; $U = Q \cup N$. The goal of a classifier is to label the examples in U correctly.

We used the above mentioned data to generate 27 data sets with different balance ratios. Since, in real life, negative examples are more abundant than positives, in our data sets the negatives are 5 times more than positives: $|N| = (|P| + |Q|) \times 5$. This ratio is the same for all 27 sets. On the other hand, the ratio, r , varies, where $r = |P| / (|P| + |Q|) = \{\%10, \%20, \dots, \%90\}$. For each r , we tested the algorithms on 3 randomly created data sets. For example, precision result of an algorithm for $r=\%40$ was calculated as the average of three precision values of 3 different data sets where $r=\%40$. Also, in all of our data sets, $|P| + |Q| = 250$ and $|N| = 1250$. We used these set sizes since they are large enough for reliable tests and small enough for reasonable algorithm running times.

3.2 Experimental setting

This section includes some implementation details and parameters chosen for the evaluated algorithms.

Bagging SVM has 2 versions: inductive and transductive (see Sect. 2.2.7). Since we want to perform prediction on our training data (U), we implemented the transductive version of Bagging SVM. Number of known positive

examples ($|P|$) is used for K as it is suggested in [16]. Subset (iteration) amount (T) is set to 20. In [16], $T = 10$ is used when bag size is greater than 30. Our bag size is always greater than 30, but since more iterations leads to less biased results, we chose $T = 20$.

We implemented PSoL and an alternative version of PSoL. During the process of negative set expansion, PSoL iteratively selects unlabeled examples and extracts them to the negative set. In the original version, number of examples to extract is limited by a threshold (T_1) where $T_1 = |P| \times 3$. But, as more examples are extracted at each iteration, negative set gets too large which results in an unbalanced data set. In our version of the algorithm, instead of limiting the extractions by each iteration, we limited the total number of extracted examples by a threshold (T_2): $T_2 = |P| \times 4$. This modified version is named as $PSoL_m$, where the original is as $PSoL_o$.

Rocchio algorithm [5] constructs a final classifier which can be used for labeling new unlabeled examples. Since we should label unlabeled examples in the training set, this algorithm is implemented with 10-fold cross validation. The algorithm splits U into 10 subsets and at each iteration it use one of these subsets for testing and others for training.

In Carter, both neural networks and SVM are tested in the classification step. In our implementation we used SVM to make a fair comparison since most algorithms covered in this paper use SVM.

For S-EM algorithm we used LPU software with parameters: “-s1 spy -s2 em -f demo”. LIBSVM³ [22] is used as the SVM implementation. For 2-class SVM we used the same kernel type and training parameters as those were used in PosOnly algorithm [18]. So, radial basis function is selected for kernel type, with $\gamma = 1.0/128$ and the cost parameter, $C = 1,000$ as in [18]. For 1-class SVM of AGPS algorithm we used the radial basis kernel since it was used in [11] with default values for other parameters.

³ <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>.

Also in [26] a biased formulation of SVM is introduced. Although Bagging SVM uses this formulation, to make fair comparisons we implemented all algorithms with C-SVC in LIBSVM.

We used Precision (PR), Recall (RC), F-Measure (FM) and Matthews correlation coefficient (MCC) for selected algorithms on each data set as the evaluation measures. Since there are 3 data sets for every P - U ratio, the average result of 3 data sets is shown in the tables. The following equations give the definitions for these measures. We also performed Wilcoxon signed rank test to compare FM and MCC performances of the algorithms.

$$PR = \frac{TP}{TP + FP} \quad (28)$$

$$RC = \frac{TP}{TP + FN} \quad (29)$$

$$FM = \frac{2 \times PR \times RC}{PR + RC} \quad (30)$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (31)$$

3.3 Results

Precision, recall, F-measure and matthews correlation coefficient (MCC) values can be seen in Tables 1, 2, 3 and 5. p values of one sided Wilcoxon signed rank test of F measure and MCC values are shown in Tables 4 and 6, respectively.

As expected, SVM_{only} generally performed worse than others in our experiments. It classified almost all of the examples in the unlabeled set as negative. This resulted in few false positives (high precision) but too many false negatives (low recall). Even if it has higher MCC results than 3 other algorithms, it has the lowest FM and can be considered as an unsuitable algorithm for our problem.

Recall values for Carter is good, but besides that, it has a very low precision due to high number of false positives. It labeled most of the U as positive. Therefore, even if it does not result in too many false negatives, since there are much more negative examples than positives in the unlabeled set, its false positive number is higher than average of other algorithms.

$PSoLo$ got worse F measure results than expected; It has the second lowest F measure. Like SVM_{only} , $PSoLo$ labels most of the U as negative. With the alternative version ($PSoL_m$) we succeeded to increase recall and F-Measure results by increasing the number of correctly classified positives and decreasing incorrectly labeled negatives. Considering this, one of the reasons for the low results of $PSoLo$ may be the overgrowth of negative set during iterations.

Table 5 Comparison between PU Learning algorithms by Matthews correlation coefficient

r	SVM_{only}	Carter	$PSoLo$	$PSoL_m$	Roc-SVM	AGPS	S-EM	PosOnly	Bagging
10%	0.000 ± 0.000	0.041 ± 0.028	0.019 ± 0.023	0.056 ± 0.059	0.117 ± 0.118	0.196 ± 0.042	0.217 ± 0.088	-0.101 ± 0.110	0.078 ± 0.076
20%	0.034 ± 0.078	0.090 ± 0.038	0.053 ± 0.075	0.129 ± 0.147	0.184 ± 0.100	0.135 ± 0.103	0.178 ± 0.028	0.030 ± 0.032	0.154 ± 0.154
30%	0.134 ± 0.064	0.115 ± 0.009	0.212 ± 0.053	0.298 ± 0.015	0.324 ± 0.047	0.281 ± 0.020	0.187 ± 0.017	0.042 ± 0.043	0.305 ± 0.034
40%	0.130 ± 0.050	0.118 ± 0.050	0.220 ± 0.018	0.200 ± 0.010	0.290 ± 0.032	0.187 ± 0.044	0.123 ± 0.025	0.077 ± 0.052	0.244 ± 0.043
50%	0.236 ± 0.035	0.124 ± 0.018	0.289 ± 0.022	0.270 ± 0.056	0.355 ± 0.035	0.144 ± 0.033	0.150 ± 0.006	0.125 ± 0.017	0.290 ± 0.026
60%	0.209 ± 0.049	0.154 ± 0.027	0.197 ± 0.090	0.176 ± 0.062	0.266 ± 0.086	0.142 ± 0.085	0.090 ± 0.021	0.184 ± 0.071	0.210 ± 0.071
70%	0.243 ± 0.038	0.097 ± 0.037	0.290 ± 0.037	0.153 ± 0.032	0.316 ± 0.058	0.178 ± 0.034	0.124 ± 0.053	0.174 ± 0.060	0.276 ± 0.097
80%	0.291 ± 0.134	0.101 ± 0.009	0.284 ± 0.074	0.171 ± 0.081	0.253 ± 0.055	0.151 ± 0.021	0.086 ± 0.002	0.164 ± 0.010	0.286 ± 0.127
90%	0.162 ± 0.050	0.057 ± 0.026	0.186 ± 0.080	0.103 ± 0.049	0.167 ± 0.075	0.071 ± 0.042	0.051 ± 0.069	0.111 ± 0.020	0.185 ± 0.070
avg.	0.160	0.100	0.194	0.173	0.252	0.165	0.134	0.089	0.225

Rows and columns stand for the ratio ($r = |P|/(|P| + |Q|)$) and the algorithm A. Each entry presents the average Matthews correlation coefficient result of an algorithm at a certain ratio

Table 6 p values of one-sided Wilcoxon signed rank test for each pair of MCC values of algorithms

	SVM _{only}	Carter	PSoL _o	PSoL _m	Roc-SVM	AGPS	S-EM	PosOnly	Bagging
SVM _{only}		0.05	0.99	0.67	0.99	0.59	0.32	0.002	0.99
Carter	0.96		0.99	1	1	0.99	0.92	0.45	1
PSoL _o	0.01	0.01		0.24	0.99	0.24	0.15	0.002	0.97
PSoL _m	0.37	0.002	0.78		1	0.21	0.10	0.06	1
Roc-SVM	0.006	0.002	0.01	0.002		0.01	0.006	0.002	0.02
AGPS	0.45	0.004	0.78	0.82	0.99		0.05	0.10	0.97
S-EM	0.72	0.10	0.87	0.91	0.99	0.96		0.32	0.97
PosOnly	1	0.58	1	0.95	1	0.91	0.71		1
Bagging	0.006	0.002	0.03	0.002	0.98	0.04	0.04	0.002	

PosOnly is another algorithm which performs poorly in our experiments. While it has an average F measure result, it has the lowest MCC. In our experiments we observed that PosOnly is not able to calculate the optimum value for $Pr(s = 1|y = 1)$ constant, even when we tried all three different estimators suggested in [17]. Since this constant is used for calculating unlabeled examples' probabilities of being positive, an inaccurate constant results in mostly false predictions.

Bagging and Carter has very close F -measure results. Since both algorithms use random subsets of the unlabeled set as negative, it is reasonable for them to get similar results. But since Bagging provide much more variety, it is expected for Bagging to be more accurate. As expected, MCC result of Bagging is much higher than Carter's. It has the second highest MCC result after RocSVM. Bagging is another algorithm which labels most of the unlabeled set as negative. Selecting negatives from U randomly and therefore using positives in the unlabeled set as negatives can be a reason for this.

Even if AGPS has the highest average F measure in our experiments, its MCC results are close to average of nine algorithms. While most of the algorithms perform unbalanced predictions on unlabeled examples, such as labelling most of them negative, AGPS makes more balanced predictions. Therefore this results higher F measure than other algorithms.

For data sets with low r , S-EM performs best. This may be because of the spy positives it places into the unlabeled set. They help S-EM to learn about the characteristics of positive examples when there are too few known positives. Removing examples from P to create the spy positive set decreases the number of positives in the training set. But this approach appears to be efficient at least for the data sets with low r .

For data sets with medium and high r , RocSVM achieved both the highest F measure and MCC results on the average. In our experiments we observed that its prototype technique has high accuracy (94%) when selecting

strong negatives. Since it starts iterations with these strong negatives, it is expected for RocSVM to get high results. Therefore it can be assumed that prototype technique and cosine measure is suitable for our type of data.

One-sided Wilcoxon signed rank test p values are given in Tables 4 and 6. These p values give a pairwise comparison of the algorithms. When we look at the p -values for F measure (Table 4), Roc-SVM and AGPS got better results than others. For the rank test of MCC, RocSVM seems to be better than others. Therefore, according to the rank tests, among two-step algorithms, AGPS and Roc-SVM seem to perform better, but for one step algorithms it is hard to reach a conclusion as no algorithm significantly outperforms others.

4 Conclusion

In this paper, we summarized the classification algorithms for domains with no known negative examples, where these algorithms are named as PU learning algorithms. We grouped the PU learning algorithms into two main classes: Two-step algorithms first find some negative examples in the unlabeled set and then use this new negative set with already known positive set to label the rest of unlabeled examples, while one-step algorithms use positive and unlabeled data directly to classify the given examples. For each algorithm, we explained how each algorithm works and differences in their methods.

Finally, we compared eight (plus a modified version for PSoL) important algorithms of PU learning by testing them with PPI network data sets. We showed and compared their results on the data sets we generated.

To the best of our knowledge, this is the first time PU learning algorithms are used for derivations of PPI networks. While these algorithms have been used mostly in areas like text classification, we showed how successful they are in network derivation using biological data. Some of them (especially Roc-SVM and AGPS) output

promising results in the sense that they can be further adapted for protein interaction predictions.

We have used microarray data as feature vector in our experiments. There are also other kinds of data types which can represent characteristics of proteins, like GO Terms, amino acid sequences, 3D structural data, etc. For future work, additional data types can be integrated to our setting to be used in the feature vector.

References

- Carter RJ, Dubchak I, Holbrook SR (2001) A computational approach to identify genes for functional RNAs in genomic sequences, Oxford Univ Press. *Nucleic Acids Res* 29(19):3928–3938
- Wang C, Ding C, Meraz RF, Holbrook SR (2006) PSoL: a positive sample only learning algorithm for finding non-coding RNA genes. *Bioinformatics* 22(21):2590–2596
- Yu H, Han J, Chang KC-C (2004) PEBL: web page classification without negative examples. *IEEE Trans Knowl Data Eng* 16(1):70–81
- Liu B, Lee WS, Yu PS, Li X (2002) Partially supervised classification of text documents. In: Proceedings of the nineteenth international conference on machine learning (ICML).
- Li X, Liu B (2003) Learning to classify texts using positive and unlabeled data. In: IJCAI'03: Proceedings of the 18th international joint conference on artificial intelligence (2003), pp 587–592
- Fung GPC, Yu JX, Lu H, Yu PS (2005) Text classification without labeled negative documents. In: ICDE '05: Proceedings of the 21st international conference on data engineering, pp 594–605
- Fung GPC, Yu JX, Lu H, Yu PS (2006) Text classification without negative examples revisit. *IEEE Trans Knowl Data Eng* 18(1):6–20
- Li X, Liu B (2004) Dealing with different distributions in learning from positive and unlabeled web qata. In: WWW Alt. '04 Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, pp 440–441
- Li X-L, Liu B, Ng S-K (2007) Learning to identify unexpected instances in the test set. In: Proceedings of the IJCAI'07 proceedings of the 20th international joint conference on artificial intelligence, pp 2802–2807
- Wang X, Xu Z, Sha C, Ester M, Zhou A (2010) Semi-supervised learning from only positive and unlabeled data using entropy. In: WAIM'10 proceedings of the 11th international conference on Web-age information management, pp 668–679
- Zhao X-M, Wang Y, Chen L, Aihara K (2008) Gene function prediction using labeled and unlabeled data. *BMC Bioinformatics* 9:57
- Denis F, Gilleron R, Tommasi M (2002) Text classification from positive and unlabeled examples. In: IPMU'02, 9th international conference on information processing and management of uncertainty in knowledge-based systems
- Denis F, Laurent A, Gilleron R, Tommasi M (2003) Text classification and co-training from positive and unlabeled examples. In: ICML workshop on the continuum from labeled to unlabeled data, pp 80–87
- Zhang D, Lee WS (2005) A simple probabilistic approach to learning from positive and unlabeled examples. In: Proceedings of the 5th annual UK workshop on computational intelligence (UKCI), London
- Lee WS, Liu B (2003) Learning with positive and unlabeled examples using weighted logistic regression. In: Proceedings of the twentieth international conference on machine learning (ICML)
- Mordelet F, Vert J-P (2010) A bagging SVM to learn from positive and unlabeled examples.
- Elkan C, Noto K (2008) Learning classifiers from only positive and unlabeled data. In: KDD '08: Proceeding of the 14th ACM SIGKDD international conference on knowledge discovery and data mining, New York: ACM 2008:213–220
- Cerulo L, Elkan C, Ceccarelli M (2010) Learning gene regulatory networks from only positive and unlabeled data. *BMC Bioinformatics* 11(1): 228
- Blum A, Mitchell T (1998). Combining labeled and unlabeled data with co-training. In: Proceedings of 11th annual conference on computational learning theory. ACM Press, New York, pp 92–100
- Joachims T (1997). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In: Proceedings of the 14th international conference on machine learning (ICML). Nashville, Tennessee, pp 143–151
- Faith et al (2008) Many microbe microarrays database: uniformly normalized affymetrix compendia with structured experimental metadata. *Nucleic Acids Res* 36 (Database issue):D866D870, Jan 2008.
- Chang C-C, Lin C-J (2011) LIBSVM : a library for support vector machines. *ACM Trans Intell Syst Technol*, 2:27:1–27:27
- Joachims T (1999) Making large-Scale SVM Learning Practical. In: Learning, B. Scholkopf, Burges C, Smola A (eds) *Advances in Kernel methods—Support Vector*, MIT-Press, Pages 169–184
- Ding C, Peng H (2005) Minimum redundancy feature selection from microarray gene expression data. In: CSB '03 Proceedings of the IEEE computer society conference on bioinformatics, p 523
- Kerrien S, Aranda B, Breuza L, Bridge A, Broackes-Carter F, Chen C, Duesbury M, Dumousseau M, Feuermann M, Hinz U, Jandrasits C, Jimenez RC, Khadake J, Mahadevan U, Masson P, Pedruzzi I, Pfeifferberger E, Porras P, Raghunath A, Roechert B, Orchard I S, Hermjakob H (2011) The IntAct molecular interaction database in 2012. *Nucleic Acids Res* 40(1):D841–D846 doi: [10.1093/nar/gkr1088](https://doi.org/10.1093/nar/gkr1088)
- Liu B, Dai Y, Li X, Lee WS, Yu PS (2003) Building text classifiers using positive and unlabeled examples. In: ICDM '03 proceedings of the third IEEE international conference on data mining, p 179
- Zhu X (2005) Semi-supervised learning literature survey. Computer Sciences Technical Report 1530, University of Wisconsin-Madison
- Zhang B, Zuo W (2008) Learning from positive and unlabeled examples: a survey. In: Proceedings of the 2008 international symposiums on information processing
- Bennett KP, Demiriz A (1998). Semi-supervised support vector machines. *Adv Neural Inform Proces Syst (NIPS)* 12:368–374
- Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of the sixteenth international conference on machine learning (ICML 1999), Morgan Kaufmann, Bled, Slovenia, pp 200–209
- Joachims T (2006) Transductive support vector machines. In: B. Scholkopf, A. Zien (eds) *Semi-supervised learning by editors olivier chapelle*. MIT press, pp 105–117