Boosting for Comparison-Based Learning

Michaël Perrot[1] and **Ulrike von Luxburg**[1,2]
[1]Max-Planck-Institute for Intelligent Systems, Tübingen, Germany
[2]University of Tübingen, Department of Computer Science, Tübingen, Germany
michael.perrot@tuebingen.mpg.de, ulrike.luxburg@uni-tuebingen.de

Abstract

We consider the problem of classification in a comparison-based setting: given a set of objects, we only have access to triplet comparisons of the form "object A is closer to object B than to object C." In this paper we introduce TripletBoost, a new method that can learn a classifier just from such triplet comparisons. The main idea is to aggregate the triplets information into weak classifiers, which can subsequently be boosted to a strong classifier. Our method has two main advantages: (i) it is applicable to data from any metric space, and (ii) it can deal with large scale problems using only passively obtained and noisy triplets. We derive theoretical generalization guarantees and a lower bound on the number of necessary triplets, and we empirically show that our method is both competitive with state of the art approaches and resistant to noise.

1 Introduction

In the past few years the problem of comparison-based learning has attracted growing interest in the machine learning community [Agarwal *et al.*, 2007, Jamieson and Nowak, 2011, Tamuz *et al.*, 2011, Tschopp *et al.*, 2011, Van Der Maaten and Weinberger, 2012, Heikinheimo and Ukkonen, 2013, Amid and Ukkonen, 2015, Kleindessner and Luxburg, 2015, Jain *et al.*, 2016, Haghiri *et al.*, 2017, Kazemi *et al.*, 2018]. The motivation is to relax the assumption that an explicit representation of the objects or a distance metric between pairs of examples are available. Instead one only has access to a set of ordinal distance comparisons that can take several forms depending on the problem at hand. In this paper we focus on triplet comparisons of the form *object* x_i *is closer to object* x_j *than to object* x_k, that is on relations of the form $d(x_i, x_j) < d(x_i, x_k)$ where d is an unknown metric[1].

We address the problem of classification with noisy triplets that have been obtained in a passive manner: the examples lie in an unknown metric space, not necessarily Euclidean, and we are only given a small set of triplet comparisons — there is no way in which we could actively ask for more triplets. Furthermore we assume that the answers to the triplet comparisons can be noisy. To deal with this problem one can try to first recover an explicit representation of the examples, a task that can be solved by ordinal embedding approaches [Agarwal *et al.*, 2007, Van Der Maaten and Weinberger, 2012, Terada and von Luxburg, 2014, Jain *et al.*, 2016], and then apply standard machine learning approaches. However, such embedding methods assume that the examples lie in a Euclidean space and do not scale well with the number of examples: typically they are too slow for datasets with more than 10^3 examples. As an alternative, it would be desirable to have a classification algorithm that can work with triplets directly, without taking a detour via ordinal embedding. To the best of our knowledge, for the case of passively obtained triplets, this problem has not yet been solved in the literature.

Another interesting question in this context is that of the minimal number of triplets required to successfully learn a classifier. It is known that to exactly recover an ordinal embedding one needs of the order $\Omega(n^3)$ passively queried triplets in the worst case (essentially all of them), unless we make stronger assumptions on the underlying metric space [Jamieson and Nowak, 2011]. However, classification is a problem which seems simpler than ordinal embedding, and thus it might be possible to obtain better lower bounds.

In this paper we propose TripletBoost, a method for classification that is able to learn using only passively obtained triplets while not making any assumptions on the underlying metric space. To the best of our knowledge this is the first approach that is able to solve this problem. Our method is based on the idea that the triplets can be aggregated into simple *triplet classifiers*, which behave like decision stumps and are well-suited for boosting approaches [Schapire and Freund, 2012]. From a theoretical point of view we prove that our approach learns a classifier with low training error, and we derive generalization guarantees that ensure that its error on new examples is bounded. Furthermore we derive a new lower bound on the number of triplets that are necessary to ensure useful predictions. From an empirical point of view we demonstrate that our approach can be applied to datasets that are several order of magnitudes larger than the ones that can currently be handled by ordinal embedding methods. Furthermore we show that our method is quite resistant to noise.

[1]Note that this kind of ordinal information is sometimes just used as side information [Bellet *et al.*, 2015, Kane *et al.*, 2017], but, as the references in the main text, we focus on the setting where ordinal comparisons are the sole information available.

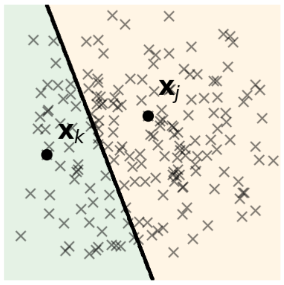


Figure 1: Example of a triplet classifier. Given two reference points x_j and x_k, the space is divided in two half-spaces: examples closer to x_j and examples closer to x_k. Triplet information is enough to reveal which half-space a point x_i is in.

2 The TripletBoost Algorithm

In this paper we are interested in multi-class classification problems. Let $(\mathcal{X}, d)$ be an unknown and general metric space, typically not Euclidean. Let $\mathcal{Y}$ be a finite label space. Let $S = \{(x_i, y_i)\}_{i=1}^n$ be a set of n examples drawn i.i.d. from an unknown distribution $\mathcal{D}_\mathcal{S}$ defined over $\mathcal{X} \times \mathcal{Y}$. Note that we use the notation x_i as a convenient way to identify an object; it does not correspond to any explicit representation that could be used by an algorithm (such as coordinates in a vector space). Let $T = \{(x_i, x_j, x_k) : (x_i, y_i), (x_j, y_j), (x_k, y_k) \in S, x_j \neq x_k\}$ be a set of m triplets. Each ordered tuple $(x_i, x_j, x_k) \in T$ encodes the following relation between the three examples:

$$d(x_i, x_j) < d(x_i, x_k)\,. \tag{1}$$

Given the triplets in T and the label information of all points, our goal is to learn a classifier. We make two main assumptions about the data. First, we assume that the triplets are uniformly and independently sampled from the set of all possible triplets. Second, we assume that the triplets in T can be noisy (that is the inequality has been swapped, for example $(x_i, x_k, x_j) \in T$ while the true relation is $d(x_i, x_j) < d(x_i, x_k)$), but the noise is uniform and independent from one triplet to another. In the following $\mathbb{I}_a$ denotes the indicator function returning 1 when a property a is verified and 0 otherwise, $\mathcal{W}_c$ is an empirical distribution over $S \times \mathcal{Y}$, and $w_{c,x_i,y}$ is the weight associated to object x_i and label y.

2.1 Weak Triplet Classifiers

Rather than considering triplets as individual pieces of information we propose to aggregate them into decision stumps that we call *triplet classifiers*. The underlying idea is to select two reference examples, x_j and x_k, and to divide the space into two *half-spaces*: examples that are closer to x_j and examples that are closer to x_k. This is illustrated in Figure 1. In principle, this can be achieved with triplets only. However, a major difficulty in our setting is that our triplets are passively obtained: for most training points x_i we do not know whether they are closer to x_j or x_k. In particular, it is impossible to evaluate the classification accuracy of such a simple classifier on the whole training set. To deal with this problem we propose to use an abstention scheme where a triplet classifier abstains if it does not know on which side of the hyperplane the considered point lies. Given a set T of triplets and two reference points x_j and x_k, we define a triplet classifier as:

$$h_{j,k}(x) = \begin{cases} o_j & \text{if } (x, x_j, x_k) \in T, \\ o_k & \text{if } (x, x_k, x_j) \in T, \\ \vartheta & \text{otherwise.} \end{cases}$$

In our multi-class setting, o_j and o_k will be sets of labels, that is $o_j, o_k \subseteq \mathcal{Y}$. In Section 2.2 we describe how we choose them in a data dependent fashion to obtain classifiers with minimal error on the training set. The prediction ϑ simply means that the triplet classifier abstains on the example. Let $\mathcal{H}$ denote the set of all possible triplet classifiers.

The triplet classifiers are very simple and, in practice, we do not expect them to perform well at all. But we prove in Section 3.1 that, for appropriate choices of o_j and o_k, they are at least as good as random predictors. This is all we need to ensure that they can be used successfully in a boosting framework. The next section describes how this works.

Algorithm 1 TripletBoost: boosting with triplet classifiers

```
Input: $S = \{(x_i, y_i)\}_{i=1}^n$ a set of $n$ examples,
       $T = \{(x_i, x_j, x_k) : x_j \neq x_k\}$ a set of $m$ triplets.
Output: $H(\cdot)$ a strong classifier.
1: Let $\mathcal{W}_1$ be the empirical uniform distribution:
   $\forall (x_i, y_i) \in S, \forall y \in \mathcal{Y}, w_{1,x_i,y} = \frac{1}{n|\mathcal{Y}|}$.
2: for $c = 1, \ldots, C$ do
3:     Choose a triplet classifier $h_c$ according to $\mathcal{W}_c$ (Equation (2)).
4:     Compute the weight $\alpha_c$ of $h_c$ according to its performance on $\mathcal{W}_c$ (Equation (3)).
5:     Update the weights of the examples to obtain a new distribution $\mathcal{W}_{c+1}$ (Equation (4)).
6: end for
7: return $H(\cdot) = \arg\max_{y \in \mathcal{Y}} \left( \sum_{c=1}^{C} \alpha_c \mathbb{I}_{h_c(\cdot) \neq \vartheta \wedge y \in h_c(\cdot)} \right)$
```

2.2 TripletBoost

Boosting is based on the insight that weak classifiers (that is classifiers marginally better than random predictors) are usually easy to obtain and can be combined in a weighted linear combination to obtain a strong classifier. This weighted combination can be obtained in an iterative fashion where, at each iteration, a weak-classifier is chosen and weighted so as to minimize the error on the training examples. The weights of the points are then updated to put more focus on hard-to-classify examples [Schapire and Freund, 2012]. In this paper we use a well-known boosting algorithm called AdaBoost.MO [Schapire and Singer, 1999, Schapire and Freund, 2012]. This method can handle multi-class problems with a one-against-all approach, works with abstaining classifiers and is theoretically well founded. Algorithm 1 summarizes the main steps of our approach that we detail below.

Choosing a triplet classifier. To choose a triplet classifier we proceed in two steps. In the first step, we select two reference points x_j and x_k such that $y_j \neq y_k$. This is done by randomly sampling from an empirical distribution $\mathcal{W}_{c,\mathcal{X}}$ on the examples. Here $\mathcal{W}_{c,\mathcal{X}}$ denotes the marginal distribution of $\mathcal{W}_c$ with respect to the examples. This distribution is updated at each iteration to put more focus on those parts of the space that are hard to classify while promoting triplet classifiers that are able to separate different classes (see Equation 4).

Choosing the predicted labels. In the second step, we choose o_j and o_k, the sets of labels that should be predicted for each half space of the triplet classifier. Given one of the half spaces, we propose to add a label to the set of predicted labels if the weight of examples of this class is greater than the weight of examples of different classes. Formally, with $w_{c,x_i,y}$ defined as in Algorithm 1, we construct o_j as follows:

$$o_j = \left\{ y : \sum_{\substack{(x_i,y_i)\in S,\\ (x_i,x_j,x_k)\in T}} \left(\mathbb{I}_{y=y_i} - \mathbb{I}_{y\neq y_i}\right) w_{c,x_i,y} > 0 \right\}. \quad (2)$$

We construct o_k in a similar way. The underlying idea is that adding h_c to the current combination of triplet classifiers should improve the predictions on the training set as much as possible. In Section 3.1 we show that this strategy is optimal and that it ensures that the selected triplet classifier is either a weak classifier or has a weight α_c of 0.

Computing the weight of the triplet classifier. To choose the weight of the triplet classifier h_c we start by computing $W_{c,+}$ and $W_{c,-}$, the weights of correctly and incorrectly classified examples:

$$W_{c,+} = \sum_{\substack{(x_i,y_i)\in S,\\ h_c(x_i)\neq\vartheta}} \left(\mathbb{I}_{y_i\in h_c(x_i)} w_{c,x_i,y_i} + \sum_{y\neq y_i} \mathbb{I}_{y\notin h_c(x_i)} w_{c,x_i,y} \right),$$
$$W_{c,-} = \sum_{\substack{(x_i,y_i)\in S,\\ h_c(x_i)\neq\vartheta}} \left(\mathbb{I}_{y_i\notin h_c(x_i)} w_{c,x_i,y_i} + \sum_{y\neq y_i} \mathbb{I}_{y\in h_c(x_i)} w_{c,x_i,y} \right). \quad (3)$$

We then set $\alpha_c = {}^{\log\left(\frac{W_{c,+}+\frac{1}{n}}{W_{c,-}+\frac{1}{n}}\right)}/_2$. The term $\frac{1}{n}$ is a smoothing constant [Schapire and Singer, 2000]: in our setting with few, passively queried triplets it helps to avoid numerical problems that might arise when $W_{c,+}$ or $W_{c,-} = 0$. In Theorem 2 we show that this choice of α_c leads to a decrease in training error as the number of iterations increases.

Updating the weights of the examples. In each iteration of our algorithm, a new triplet classifier h_c is added to the weighted combination of classifiers, and we need to update the empirical distribution $\mathcal{W}_c$ over the examples for the next iteration. The idea is (i) to reduce the weights of correctly classified examples, (ii) to keep constant the weights of the examples for which the current triplet classifier abstains, and (iii) to increase the weights of incorrectly classified examples. The weights are then normalized by a factor Z_c so that $\mathcal{W}_{c+1}$ remains an empirical distribution over the examples. Formally, $\forall (x_i,y_i) \in S, \forall y \in \mathcal{Y}$, if $h_c(x_i) = \vartheta$ then $w_{c+1,x_i,y} = \frac{w_{c,x_i,y}}{Z_c}$ and if $h_c(x_i) \neq \vartheta$ then

$$w_{c+1,x_i,y} = \frac{w_{c,x_i,y}}{Z_c} \exp\left\{-\alpha_c \left(\mathbb{I}_{y=y_i} - \mathbb{I}_{y\neq y_i}\right) \times \left(\mathbb{I}_{y\in h_c(x_i)} - \mathbb{I}_{y\notin h_c(x_i)}\right)\right\}. \quad (4)$$

Using H for prediction. Given a new example x, TripletBoost predicts its label as

$$H(x) = \underset{y\in\mathcal{Y}}{\arg\max} \left(\sum_{c=1}^{C} \alpha_c \mathbb{I}_{h_c(x)\neq\vartheta \wedge y\in h_c(x)} \right) \quad (5)$$

that is the label with the highest weight as predicted by the weighted combination of selected triplet classifiers. However, recall that we are in a passive setting, and thus we assume that we are given a set of triplets $T_x = \{(x,x_j,x_k) : (x_j,y_j),(x_k,y_k) \in S, x_j \neq x_k\}$ associated with the example x (but there is no way to choose them). Hence, some of the triplets in T_x correspond to triplet classifiers in H (that is $(x,x_j,x_k) \in T_x$ and the reference points for h_c were x_j and x_k) and some do not. In particular, it might happen that none of the triplets in T_x corresponds to a triplet classifier in H and, in this case, H can only randomly predict a label. In Section 3.3 we provide a lower bound on the number of triplets necessary to avoid this behaviour. The main computational bottleneck when predicting the label of a new example x is to check whether the triplets in T_x match a triplet classifier in H. A naive implementation would compare each triplet in T_x to each triplet classifier, which can be as expensive as $O(|T_x|\,C)$. Fortunately, by first sorting the triplets and the triplet classifiers, a far more reasonable complexity of $O(|T_x|\log(|T_x|) + C\log(C))$ can be achieved.

3 Theoretical Analysis

In this section we show that our approach is theoretically well founded. First we prove that the triplet classifiers with non-zero weights are weak learners: they are slightly better than random predictors (Theorem 1). Building upon this result we show that, as the number of iterations increases, the training error of the strong classifier learned by TripletBoost is decreased (Theorem 2). Then, to ensure that TripletBoost does not over-fit, we derive a generalization bound showing that, given a sufficient amount of training examples, the test error is bounded (Theorem 3). Finally, we derive a lower bound on the number of triplets necessary to ensure that TripletBoost does not learn a random predictor (Theorem 4).

3.1 Triplet Classifiers and Weak Learners

We start this theoretical analysis by showing that the strategy to choose the predicted labels of triplet classifiers described in Equation (2) is optimal: it ensures that their error is minimal on the training set (compared to any other labelling strategy). We also show that the triplet classifiers are never worse than random predictors and in fact, that only those triplets classifiers that are weak classifiers (strictly better than random classifiers) are affected a non-zero weight. This is summarized in the next theorem.

Theorem 1 (Triplet classifiers and weak learners). *Let $\mathcal{W}_c$ be an empirical distribution over $S \times \mathcal{Y}$ and h_c be the corresponding triplet classifier chosen as described in Section 2.2. It holds that:*

1. *the error of h_c on $\mathcal{W}_c$ is at most the error of a random predictor and is minimal compared to other labelling strategies,*

2. *the weight α_c of the classifier is non-zero if and only if, h_c is a weak classifier, that is is strictly better than a random predictor.*

Proof. Given in Perrot and von Luxburg [2018]. □

3.2 Boosting Guarantees

From a theoretical point of view the boosting framework has been extensively investigated and it has been shown that most AdaBoost-based methods decrease the training error at each iteration [Freund and Schapire, 1997]. Another question that has attracted a lot of attention is the problem of generalization. It is known that when the training error has been minimized, AdaBoost-based methods often do not over-fit and it might even be beneficial to further increase the number of weak learners. A popular explanation is the margin theory which says that as the number of iterations increases, the confidence of the algorithm in its predictions increases and thus, the test accuracy is improved [Schapire *et al.*, 1998, Breiman, 1999, Wang *et al.*, 2011, Gao and Zhou, 2013]. TripletBoost is based on AdaBoost.MO [Schapire and Freund, 2012], and thus it inherits the theoretical guarantees presented above. In this section, we provide two theorems which show that (i) TripletBoost reduces the training error as the number of iterations increases, and (ii) it generalizes well to new examples.

The following theorem shows that, as the number of iterations increases, TripletBoost decreases the training error.

Theorem 2 (Reduction of the training error). *Let S be a set of n examples and T be a set of m triplets (obtained as described in Section 2). Let $H(\cdot)$ be the classifier obtained after C iterations of TripletBoost (Algorithm 1) using S and T as input. It holds that:*

$$\underset{(x,y)\in S}{\mathbb{P}}\left[H(x)\neq y\right]\leq\frac{|\mathcal{Y}|}{2}\prod_{c=1}^{C}Z_c$$

with $Z_c = (1 - W_{c,+} - W_{c,-}) + (W_{c,+}) \cdot \sqrt{\frac{W_{c,-}+\frac{1}{n}}{W_{c,+}+\frac{1}{n}}} + (W_{c,-}) \cdot \sqrt{\frac{W_{c,+}+\frac{1}{n}}{W_{c,-}+\frac{1}{n}}} \leq 1.$

Proof. This result, inherited from AdaBoost.MO, is proven in Perrot and von Luxburg [2018]. □

The next theorem shows that the true error of a classifier learned by TripletBoost can be bounded by a quantity related to the confidence of the classifier on the training examples, with respect to a margin, plus a term which decreases as the number of examples increases. The confidence of the classifier on the training examples is also bounded and decreases for sufficiently small margins.

Theorem 3 (Generalization guarantees). *Let $\mathcal{D}_\mathcal{S}$ be a distribution over $\mathcal{X}\times\mathcal{Y}$, let S be a set of n examples drawn i.i.d. from $\mathcal{D}_\mathcal{S}$, and let T be a set of m triplets (obtained as described in Section 2). Let $H(\cdot)$ be the classifier obtained after C iterations of TripletBoost (Algorithm 1) using S and T as input. Let $\mathcal{H}$ be a set of triplet classifiers as defined in Section 2.1. Then, given a margin parameter $\theta > \sqrt{\frac{\log|\mathcal{H}|}{16|\mathcal{Y}|^2 n}}$ and a measure of the confidence of $H(\cdot)$ in its predictions $\theta_H(x,y)$, with probability at least $1-\delta$, we have that*

$$\underset{(x,y)\sim\mathcal{D}_\mathcal{S}}{\mathbb{P}}\left[H(x)\neq y\right]\leq\underset{(x,y)\in S}{\mathbb{P}}\left[\theta_H(x,y)\leq\theta\right] + \mathcal{O}\left(\sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{n}+\log\left(\frac{|\mathcal{Y}|^2 n\theta^2}{\log|\mathcal{H}|}\right)\frac{\log|\mathcal{H}|}{n\theta^2}}\right)$$

Furthermore we have that

$$\underset{(x,y)\in S}{\mathbb{P}}\left[\theta_H(x,y)\leq\theta\right]\leq\frac{|\mathcal{Y}|}{2}\prod_{c=1}^{C}Z_c\sqrt{\left(\frac{W_{c,+}+\frac{1}{n}}{W_{c,-}+\frac{1}{n}}\right)^{\theta}}.$$

Proof. This result, inherited from AdaBoost.MO, is proven in Perrot and von Luxburg [2018]. □

At a first glance it seems that this bound does not depend on m, the number of available triplets. However, this dependency is implicit: m impacts the probability that the training examples are well classified with a large margin θ. If the number of triplets is small, the probability that the training examples are well classified with a given margin is small. This probability increases when the number of triplets increases (as illustrated in Perrot and von Luxburg [2018]).

To prove a bound that explicitly depends on m would be of significant interest. However this is a difficult problem, as it requires to use an explicit measure of complexity for general metric spaces, which is beyond the scope of this paper.

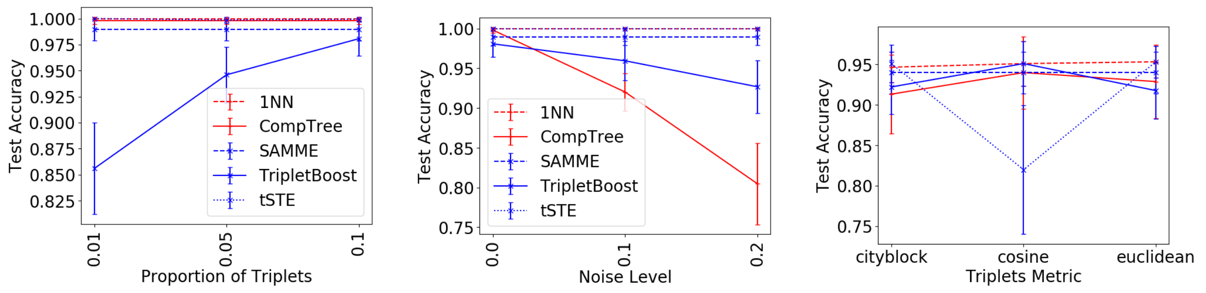
3.3 Lower Bound on the Number of Triplets

In this section we investigate the minimum number of triplets that are necessary to ensure that our algorithm performs well. Ideally, we would like to obtain a lower bound on the number of triplets that are necessary to achieve a given accuracy. In this paper we take a first step in this direction by deriving a bound on the number of triplets that are necessary to ensure that the learned classifier does not abstain on any unseen example. Theorems 4 and 5 show that it abstains with high probability if it is learned using too few triplets or if it combines too few triplet classifiers.

Theorem 4 (Lower bound on the probability that a strong classifier abstains). *Let $n\geq 2$ be the number of training examples, $p=\frac{2n^k}{n^3}$ with $k\in\left[0,3-\frac{\log(2)}{\log(n)}\right)$ be the probability that a triplet is available in the triplet set T and $C=\frac{n^\beta}{2}$ with $\beta\in\left[0,1+\frac{\log(n-1)}{\log(n)}\right]$ be the number of classifiers combined in the learned classifier. Let $\mathcal{A}$ be any algorithm learning a classifier $H(\cdot)=\arg\max_{y\in\mathcal{Y}}\left(\sum_{c=1}^{C}\alpha_c\mathbb{I}_{y\in h_c(\cdot)}\right)$ that combines several triplet classifiers using some weights $\alpha_c\in\mathbb{R}$. Assume that triplet classifiers that abstain on all the training examples have a weight of 0 (that is if $h_c(x_i)=\vartheta$ for all the examples $(x_i,y_i)\in S$ then $\alpha_c=0$). Then the probability that H abstains on a test example is bounded as follows:*

$$\underset{(x,y)\sim\mathcal{D}_\mathcal{S}}{\mathbb{P}}\left[H(x)=\vartheta\right]\geq\left(1-p+p\left(1-p\right)^n\right)^C. \quad (6)$$

Proof. Given in Perrot and von Luxburg [2018]. □



(a) Moons, Metric: Euclidean, Noise Level: 0%

(b) Moons, Metric: Euclidean, Proportion of Triplets: 10%

(c) Iris, Proportion of Triplets: 10%, Noise Level: 0%

Figure 2: Moons is a small scale dataset with 500 examples and 2 dimensions. We fix the triplets metric to the Euclidean distance. In Figure 2a we consider the noise free setting and we vary the proportion of available triplets from 1 to 10% of all the triplets. In Figure 2b we fix this proportion to 10% and we vary the noise level from 0 to 20%. Iris is a small scale dataset with 150 examples and 4 dimensions. In Figure 2c we fix the proportion of available triplets to 10%, the noise level to 0% and we vary the metric considered to generate the triplets.

To understand the implications of this theorem we consider a concrete example.

Example 1. *Assume that we build a linear combination of all possible triplet classifiers, that is $C = \frac{n(n-1)}{2}$. Then we have*

$$\lim_{n \to +\infty} \underset{(x,y)\sim\mathcal{D}_S}{\mathbb{P}} [H(x) = \vartheta] \geq \begin{cases} 1 & \text{if } k < \frac{3}{2}, \\ \exp(-2) & \text{if } k = \frac{3}{2}, \\ 0 & \text{if } \frac{3}{2} < k, \end{cases} \quad (7)$$

where k is the parameter that controls the probability p that a particular triplet is available in the triplets set T. The bottom line is that when $k < \frac{3}{2}$, that is when we do not have at least $\Omega(n\sqrt{n})$ random triplets, the learned classifier abstains on all the examples.

Proof. Given in Perrot and von Luxburg [2018]. □

Theorem 4 shows that when p and C are too small, then the strong classifier abstains with high probability. However, the theorem does not guarantee that the strong classifier does not abstain when p and C are large. The next theorem takes care of this other direction under slightly stronger assumptions on the weights learned by the algorithm.

Theorem 5 (Exact bound on the probability that a strong classifier abstains). *In Theorem 4, further assume that each triplet classifier that does not abstain on at least one training example has a weight different from 0 (if for at least one example $(x_i, y_i) \in S$ we have that $h_c(x_i) \neq \vartheta$ then $\alpha_c \neq 0$). Then equality holds in Equation (6).*

Proof. Given in Perrot and von Luxburg [2018]. □

Theorem 5 implies that equality holds in Example 1, thus when $C = \frac{n(n-1)}{2}$ we need at least $k > \frac{3}{2}$, that is at least $\Omega(n\sqrt{n})$ random triplets, to obtain a classifier that never abstains. In Perrot and von Luxburg [2018] we extend Example 1 and we study the limit as $n \to \infty$ for general values of C and p. We also provide a graphical illustration of the bound and a discussion on how this lower bound compares to existing results [Ailon, 2012, Jamieson and Nowak, 2011, Jain *et al.*, 2016].

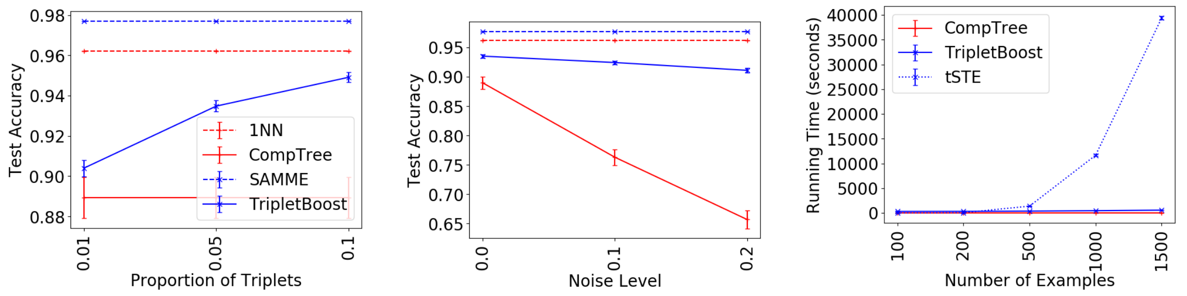
4 Experiments

We propose an empirical evaluation of TripletBoost. We consider six datasets of varying scales and four baselines.

Baselines. First, we consider an embedding approach. We use tSTE [Van Der Maaten and Weinberger, 2012] to embed the triplets in a Euclidean space and we use the 1-nearest neighbour algorithm for classification. We also would like to compare to alternative approaches able to learn directly from triplets (without embedding as a first step). However, to the best of our knowledge, TripletBoost is the only method able to do classification using only passively obtained triplets. The only option is to choose competing methods that have access to more information (providing them an unfair advantage). We settled for a method that uses actively chosen triplets to build a comparison-tree to retrieve nearest neighbours (CompTree) [Haghiri *et al.*, 2017]. Finally, to put the results obtained in the triplet setting in perspective, we consider two methods that use the original Euclidean representations of the data, the 1-nearest neighbour algorithm (1NN) and AdaBoost.SAMME (SAMME) [Hastie *et al.*, 2009].

Implementation details. For tSTE we used the implementation distributed on the authors' website and we set the embedding dimension to the original dimension of the data. This method was only considered for small datasets with less than 10^3 examples as it does not scale well to bigger datasets (Figure 3c). For CompTree we used our own implementation and the leaf size of the comparison tree is set to 1 as this is the only value for which this method can handle noise. For 1NN and SAMME we used sk-learn [Pedregosa *et al.*, 2011]. The number of boosting iterations for SAMME is set to 10^3. Finally for TripletBoost we set the number of iterations to 10^6.

Datasets and performance measure. We consider six datasets: Iris, Moons, Gisette, Cod-rna, MNIST, and kMNIST. For each dataset we generate some triplets as in Equation (1) using three metrics: the Euclidean, Cosine, and Cityblock distances (See Perrot and von Luxburg [2018] for details). Given a set of n examples there are $n^2(n-1)/2$ possible triplets. We consider three different regimes where 1%, 5% or 10% of them are available, and we consider three noise levels



(a) Gisette, Metric: Euclidean, Noise Level: 0%

(b) Gisette, Metric: Euclidean, Proportion of Triplets: 5%

(c) Moons, Metric: Euclidean, Proportion of Triplets: 10%, Noise Level: 10%

Figure 3: The Gisette dataset has 7000 examples and 5000 dimensions. In Figure 3a we consider the noise free setting and we vary the proportion of triplets available from 1 to 10% of all the triplets. In Figure 3b we fix the proportion of available triplets to 5% and we vary the noise level from 0 to 20%. Figure 3c presents the training time of the triplet-based methods with training samples of increasing sizes on the Moons dataset. The proportion of triplets and the noise level were both set to 10%. The results were obtained on a single core @3.40GHz.

where 0%, 10% or 20% of them are incorrect. We measure performances in terms of test accuracy (higher is better). For all the experiments we report the mean and standard deviation of 10 repetitions. Since the results are mostly consistent across the datasets we present some representative ones here and defer the others to Perrot and von Luxburg [2018].

Small scale regime. We first consider the datasets with less than 10^3 training examples (Figure 2). In this setting our method does not perform well when the number of triplets is too small, but gets closer to the baselines when the number of triplets increases (Figure 2a). This behaviour can be easily explained: when only 1% of the triplets are available, the triplet classifiers abstain on all but 3 or 4 examples on average and thus their performance evaluations are not reliable. Consequently their weights cannot be chosen in a satisfactory manner. This problem vanishes when the number of triplets increases. With increasing noise levels (Figure 2b) one can notice that TripletBoost is more robust than CompTree. Indeed, CompTree generates a comparison-tree based on individual triplet queries, and the greedy decisions in the tree building procedure can easily be misleading in the presence of noise. Finally, our approach is less sensitive than tSTE to changes in the metric that generated the triplets (Figure 2c). Indeed, tSTE assumes that this metric is the Euclidean distance while our approach does not make any assumptions.

Large scale regime. On larger datasets (Figure 3), our method does not reach the accuracy of 1NN and SAMME, who exploit a significant amount of extra information. Still, it performs quite well and is competitive with CompTree, the method that uses active rather than passive queries. The ordinal embedding methods cannot compete in this regime, as they are too slow to even finish (Figure 3c). Once again TripletBoost is quite resistant to noise (Figure 3b).

5 Conclusion

In this paper we proposed TripletBoost to address the problem of comparison-based classification. It is particularly designed for situations where triplets cannot be queried actively, and we have to live with whatever set of triplets we get. We do not make any geometric assumptions on the underlying space. From a theoretical point of view we have shown that TripletBoost is well founded and we proved guarantees on both the training error and the generalization error of the learned classifier. Furthermore we derived a new lower bound showing that to avoid learning a random predictor, at least $\Omega(n\sqrt{n})$ triplets are needed. In practice we have shown that, given a sufficient amount of triplets, our method is competitive with state of the art methods and that it is quite resistant to noise.

To the best of our knowledge, TripletBoost is the first algorithm that is able to handle large scale datasets using only passively obtained triplets. This means that the comparison-based setting could be considered for problems which were, until now, out of reach. As an illustration, consider a platform where users can watch, comment and rate movies. It is reasonable to assume that triplets of the form *movie m_i is closer to m_j than to m_k* can be automatically obtained using the ratings of the users, their comments, or their interactions. In this scenario, active learning methods are not applicable since the users might be reluctant to answer solicitations. Similarly, embedding methods are too slow to handle large numbers of movies or users. However, we can use TripletBoost to solve problems such as predicting the genres of the movies. As a proof of concept we considered the 1m movielens dataset [Harper and Konstan, 2016]. It contains 1 million ratings from 6040 users on 3706 movies. We used the users' ratings to obtain some triplets about the movies and TripletBoost to learn a classifier able to predict the genres of a new movie (details are given in Perrot and von Luxburg [2018]). Given a new movie, in ∼83% of the cases the genre predicted as the most likely one is correct and, on average, the 5 genres predicted as the most likely ones cover ∼92% of the true genres.

Acknowledgments

Ulrike von Luxburg acknowledges funding by the DFG through the Institutional Strategy of the University of Tübingen (DFG, ZUK 63) and the Cluster of Excellence EXC 2064/1, project number 390727645.

References

[Agarwal *et al.*, 2007] Sameer Agarwal, Josh Wills, Lawrence Cayton, Gert Lanckriet, David Kriegman, and Serge Belongie. Generalized non-metric multidimensional scaling. In *Artificial Intelligence and Statistics*, pages 11–18, 2007.

[Ailon, 2012] Nir Ailon. An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity. *Journal of Machine Learning Research*, 13(Jan):137–164, 2012.

[Amid and Ukkonen, 2015] Ehsan Amid and Antti Ukkonen. Multiview triplet embedding: Learning attributes in multiple maps. In *International Conference on Machine Learning*, pages 1472–1480, 2015.

[Bellet *et al.*, 2015] Aurélien Bellet, Amaury Habrard, and Marc Sebban. Metric learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 9(1):1–151, 2015.

[Breiman, 1999] Leo Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[Freund and Schapire, 1997] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[Gao and Zhou, 2013] Wei Gao and Zhi-Hua Zhou. On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18, 2013.

[Haghiri *et al.*, 2017] Siavash Haghiri, Debarghya Ghoshdastidar, and Ulrike von Luxburg. Comparison-based nearest neighbor search. In *Artificial Intelligence and Statistics*, pages 851–859, 2017.

[Harper and Konstan, 2016] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19, 2016.

[Hastie *et al.*, 2009] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[Heikinheimo and Ukkonen, 2013] Hannes Heikinheimo and Antti Ukkonen. The crowd-median algorithm. In *First AAAI Conference on Human Computation and Crowdsourcing*, 2013.

[Jain *et al.*, 2016] Lalit Jain, Kevin G Jamieson, and Rob Nowak. Finite sample prediction and recovery bounds for ordinal embedding. In *Neural Information Processing Systems*, pages 2711–2719, 2016.

[Jamieson and Nowak, 2011] Kevin G Jamieson and Robert D Nowak. Low-dimensional embedding using adaptively selected ordinal data. In *Conference on Communication, Control, and Computing*, pages 1077–1084, 2011.

[Kane *et al.*, 2017] Daniel M Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active classification with comparison queries. In *Foundations of Computer Science*, pages 355–366, 2017.

[Kazemi *et al.*, 2018] Ehsan Kazemi, Lin Chen, Sanjoy Dasgupta, and Amin Karbasi. Comparison based learning from weak oracles. *arXiv preprint arXiv:1802.06942v1*, 2018.

[Kleindessner and Luxburg, 2015] Matthäus Kleindessner and Ulrike Luxburg. Dimensionality estimation without distances. In *Artificial Intelligence and Statistics*, pages 471–479, 2015.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Perrot and von Luxburg, 2018] Michaël Perrot and Ulrike von Luxburg. Boosting for comparison-based learning. *arXiv preprint arXiv:1810.13333*, 2018.

[Schapire and Freund, 2012] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

[Schapire and Singer, 1999] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

[Schapire and Singer, 2000] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2-3):135–168, 2000.

[Schapire *et al.*, 1998] Robert E Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of statistics*, pages 1651–1686, 1998.

[Tamuz *et al.*, 2011] Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. Adaptively learning the crowd kernel. In *International Conference on Machine Learning*, pages 673–680, 2011.

[Terada and von Luxburg, 2014] Yoshikazu Terada and Ulrike von Luxburg. Local ordinal embedding. In *International Conference on Machine Learning*, pages 847–855, 2014.

[Tschopp *et al.*, 2011] Dominique Tschopp, Suhas Diggavi, Payam Delgosha, and Soheil Mohajer. Randomized algorithms for comparison-based search. In *Neural Information Processing Systems*, pages 2231–2239, 2011.

[Van Der Maaten and Weinberger, 2012] Laurens Van Der Maaten and Kilian Weinberger. Stochastic triplet embedding. In *Machine Learning for Signal Processing*, pages 1–6, 2012.

[Wang *et al.*, 2011] Liwei Wang, Masashi Sugiyama, Zhaoxiang Jing, Cheng Yang, Zhi-Hua Zhou, and Jufu Feng. A refined margin analysis for boosting algorithms via equilibrium margin. *Journal of Machine Learning Research*, 12(Jun):1835–1863, 2011.