Ensemble Learning with Biased Classifiers: The Triskel Algorithm

Andreas Heß, Rinat Khoussainov and Nicholas Kushmerick

University College Dublin, Ireland
{andreas.hess, rinat, nick}@ucd.ie

Abstract. We propose a novel ensemble learning algorithm called Triskel, which has two interesting features. First, Triskel learns an ensemble of classifiers that are biased to have high precision (as opposed to, for example, boosting, where the ensemble members are biased to ignore portions of the instance space). Second, Triskel uses weighted voting like most ensemble methods, but the weights are assigned so that certain pairs of biased classifiers outweigh the rest of the ensemble, if their predictions agree. Our experiments on a variety of real-world tasks demonstrate that Triskel often outperforms boosting, in terms of both accuracy and training time.

1 Introduction

Ensemble techniques have been demonstrated to be an effective way to reduce the error a base learner across a wide variety of tasks. The basic idea is to vote together the predictions of a set of classifiers that have been trained slightly differently for the same task. There is a strong body of theory explaining why ensemble techniques work.

Nevertheless, it is straightforward to construct learning tasks that confound existing ensemble techniques. For example, consider a synthetic“layer cake” binary learning task shown in Fig. 1. SVM with a linear kernel learns a decision surface with a large error. Boosting SVM does not help: at each iteration, the classifier is unable to stop making mistakes on the middle two regions; these regions then get even more weight on the next iteration, and eventually boosting gives up because it can not find a classifier with error less than 0.5.

However, Fig. 1(b) shows that ensemble methods are in principle well suited to this task: when combined with a simple unweighted vote, the set of three linear decision surfaces yields an ensemble that has zero error.

Motivated by this sort of learning task, we propose a novel ensemble learning algorithm called Triskel, which has two interesting features. First, Triskel learns an ensemble of classifiers that are biased to have high precision for one particular class. For example, in Fig. 1(b), one of the “outer” classifiers is biased to (i.e. has high precision, albeit mediocre recall, for) the positive class, and the other classifier is biased for the negative class. In contrast, most existing ensemble techniques feature ensemble members that are biased to focus on various regions of the instance space. For example, at each round, boosting focuses on instances that were classified incorrectly in previous rounds; and bagging simply involves hiding some of the training data from each ensemble member.

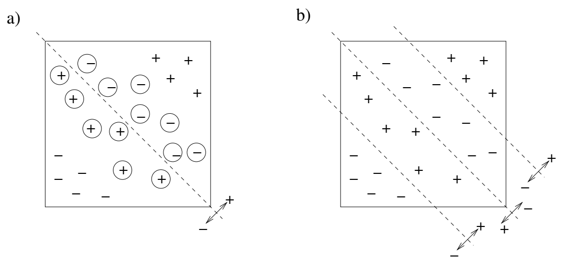

Fig. 1. The “layer cake” task: a) decision surface learned by a single SVM with linear kernel (circled instances are classified incorrectly); b) an ensemble of three linear SVMs that as zero training error when combined with a simple majority vote.

The second interesting feature is the manner in which Triskel assigns weights to the ensemble members. Triskel uses weighted voting like most ensemble methods, but the weights are assigned so that certain pairs of biased classifiers outweigh the rest of the ensemble, if their predictions agree. For example, in Fig. 1(b), the two “outer” classifiers dominate the vote if they agree, but if they disagree then the “inner” classifier casts the deciding vote. Our algorithm is named Triskel after a Celtic spiral design with three branches. In its simplest incarnation, Triskel uses an ensemble of three classifiers: one classifier biased for the positive class, one classifier biased for the negative class, and one unbiased classifier to make predictions when the others disagree.

We make the following contributions. First, we motivate and describe Triskel, our novel approach to ensemble learning, and describe various ways to construct the biased classifiers on which Triskel relies. Second, we discuss how Triskel represents a middle ground between covering and ensemble techniques such as boosting. Finally, we evaluate Triskel on a variety of real-world tasks, and demonstrate that our method often outperforms boosting, in terms of both accuracy and training time.

2 Related work

Various explanations for the success of ensemble methods have been proposed. For example, [1] presents two necessary and sufficient conditions for an ensemble to be more accurate than any of its member classifiers: the classifiers should be accurate (better than guessing randomly) and diverse (they make different—ideally, independent—mistakes on new data). A simple probabilistic argument shows that if the classifiers’ errors are independent and their error rates are less than that of guessing randomly, then the probability that the majority of classifiers is wrong on a new instance is less than the error of any individual. Thus, combining the decisions using a majority vote always has lower error than any of the members.

Of course, these assumptions may not hold in practice, but [2] discusses three fundamental ways in which an ensemble can achieve better performance: statistical, computational and representational. The statistical analysis starts with the observation that any learning algorithm tries to find a hypothesis that has a good accuracy on the training data. When the amount of the training data is small, there may be many different hypotheses that all give the same accuracy on the training data. However, not all of these hypotheses may be correct for a given new data instance. Constructing an ensemble out of all these accurate classifiers can allow the algorithm to reduce the risk of choosing the wrong hypothesis.

The computational argument is that many learning algorithms perform some sort of local search in the hypotheses space that may get stuck in a local optimum. Examples include gradient-based search in neural networks and greedy search in decision trees. An ensemble constructed by running the local search from multiple different starting points may result in a better approximation to the true hypothesis. Finally, the representational analysis follows from the fact that a learning algorithm may not be capable of representing the true function either because it is outside of its hypothesis space or because it does not have sufficient training data to explore all of its hypothesis space to find it (e.g. the classifier would stop searching once it finds a hypothesis that fits the training data). By combining several different hypotheses (e.g. using a weighted sum) it may be possible to expand the space of representable functions.

Perhaps the best-known ensemble methods are bagging [3], and boosting [4, 5], in particular AdaBoost [6]. Bagging generates different training sets by drawing randomly with replacement from the original data set. The classifiers' decisions are combined using the majority vote. AdaBoost performs several learning iterations on the same training set. However, in each iteration it adjusts the weights of the training instances to emphasize the examples that were misclassified by the last learned classifier. The decisions of the classifiers in the ensemble are combined using weighted voting, where the weights depend on the error of the classifier on the training set.

Since bagging constructs its training sets (and, hence, its ensemble members) independently from the others, it mainly addresses the statistical and, to a lesser extent, computational problems. AdaBoost constructs each new hypothesis to eliminate remaining errors and, thus, is directly trying to address the representational problem. It has been shown that by focusing on incorrectly classified instances AdaBoost minimises a particular error function of the ensemble on the training data called the negative exponential of the margin [6, 2].

3 The Triskel Algorithm

Motivation. One of the problems with AdaBoost is that in each subsequent iteration the base learner is presented with more and more difficult problems. The redistribution of instance weights is based on the errors of the last learned hypothesis on the training data. Over multiple iterations, this can result in weight distributions that are too complex for the base learner to handle. For example, suppose we would like to boost a Support Vector Machine [7] with a linear kernel on a synthetic data set shown in Figure 1a. The Figure shows the decision surface that an SVM would learn on this data in the first iteration. We can see that the distribution of errors is such that a linear decision surface will do a poor job on such task. Specifically, the weight distribution will switch in this case between inner and outer instances after each boosting iteration without improvements to the resulting ensemble accuracy.

Nonetheless, the example in Figure 1a can be handled perfectly by an ensemble of three linear separators shown in Figure 1b combined using a majority vote. One classifier separates a part of the positive instances from the rest of positives and negatives, one classifier separates a part of the negative instances, and the remaining classifier handles the instances where the first two classifiers disagree.

An analogy between this approach and set covering can be drawn. Essentially, one classifiers covers the data instances that can be confidently classified as positive ("easy" positives), one classifier covers the data that can be confidently classified as negatives ("easy" negatives), and the last classifier is used to handle the remaining "hard" instances. Our Triskel algorithm is inspired by this idea of exploring a middle ground between ensemble and set covering methods.

In order to identify instances that can be confidently classified as positive or negative, we make use of biased classifiers. A classifier that is biased towards predicting positives will usually have a high precision on negative instances and vice versa. We train a biased classifier for each class. All instances where the biased classifiers agree are considered "easy", all other instances are "hard". The third classifier, the *arbiter*, is then trained only on those "hard" instances. The intuition is that the feature patterns among the "hard" instances may be different from those among the "easy" training examples. By separating away the "easy" instances and training the arbiter only on the "hard" ones, we make the learning problem for the arbiter easier since it only has to deal with a supposedly more regular subset of the data.

Like AdaBoost, we are trying to improve (boost) the classification accuracy of the base classifier on the training set by increasing the representational power using the ensemble. However, the expectation is that we can achieve better results by splitting one hard classification problem into a series of easier ones instead of progressively constructing more difficult problems as in AdaBoost.

The Algorithm. Consider first the Triskel algorithm for a binary classification problem. Assume that a classifier is a function mapping data instances onto a binary set of classes: $h : X \to \{-1, +1\}$. Similarly to AdaBoost, Triskel is an iterative algorithm. In each iteration, we train a pair of biased classifiers: one classifier biased towards the positive class, and one classifier biased towards the negative class. For a discussion of different ways of biasing classifiers, see section 3. Next, we evaluate the biased classifiers on the training data and obtain two sets of instances: "easy" examples, where the biased classifiers agree; and "hard" ones, where the biased classifiers disagree. To obtain the training set for the next iteration, the weights of the "easy" instances are reduced and the weights of the "hard" instances are increased. The training set obtained after the last iteration is used to train the *arbiter* classifier. Algorithm 1 shows the details.

To combine the decisions of the learned classifiers, we use a conventional weighted voting scheme, with the weights set in such a way that some ensemble members' votes can dominate the others. Specifically, we use a sequence of exponentially decreasing weights such that if two biased classifiers from a given iteration agree on the label of a new instance, then their combined vote outweighs the votes of the classifiers from all subsequent rounds. Essentially, in each iteration we classify and separate the "easy" instances and then use the ensemble members from subsequent iterations to handle the remaining "hard" instances in a recursive way.

Algorithm 1 Triskel

```
/* To train on $\{\ldots,(x_i,y_i),\ldots\}$ ($y_i = \pm 1$) */
Choose the method of weight adjustment:
$W_{easy} = 0; W_{hard} = 1$, or /* "separation" */
$W_{easy} = 1/2; W_{hard} = 2$ /* "soft covering" */
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
    $h_t^+ =$ Learn with weights $D_{t-1}$, biased for class +1
    $h_t^- =$ Learn with weights $D_{t-1}$, biased for class -1
    $\alpha_t = 2^{K-t}$
    for each instance $i$ do
        $\Delta_{t,i} = \begin{cases} W_{easy}, & \text{if } h_t^+(x_i) = h_t^-(x_i) = y_i \\ W_{hard}, & \text{otherwise} \end{cases}$
        $D_t(i) = D_{t-1}(i) \cdot \Delta_{t,i}$ and normalise
    end for
end for
$h_{K+1} =$ Learn with weights $D_K$, unbiased
$\alpha_{K+1} = 1$
/* To classify instance $x$ */
return $y = \text{sign}\left[\sum_{t=1}^{K+1} \alpha_t h_t^*(x)\right]$, where $h_t^*(x) = h_t^+(x) + h_t^-(x)$ for $t \leq K$, and $h_t^*(x) =$
$h_{K+1}(x)$ for $t = K + 1$.
```

There are two principle ways in which the instance weights can be adjusted during training. One way is to set the weights of the "easy" instances to zero, leaving the weights of the "hard" instances unchanged. In this case, the classifiers in each subsequent iteration are trained on a shrinking subset of the training data. This method is more similar to the set covering idea, since after each iteration (the covered) part of the training instances is completely removed from consideration. The problem with this method is that it may quickly "run out of instances". That is, the number of instances left in consideration may quickly become too small to train a sensible classifier.

Therefore, the second way to adjust the instance weights is more similar to boosting, when the weights of "easy" instances are reduced, while the weights of "hard" instances are increased. In our experiments, we increase or reduce the weights by the factor of 2 (see Algorithm 1).

Generating Biased Classifiers. Biasing techniques have been previously used for improving performance of neural net classifiers on imbalanced datasets [8] and for adaptive voting in the ensembles of classifiers for incremental learning [9].

Some machine learning algorithms have an inherent way of setting a bias. Bayesian classifiers, for example, output a probability distribution. The class with the highest posterior probability as calculated by the classifier is predicted. It is easy to bias a Bayesian classifier by either modifying the prior probabilities or to impose biased thresholds on the posterior probabilities. Support Vector Machines also use a confidence value threshold.

There are, however, more generic ways to bias classifiers. Resampling techniques have been used in literature to address the problem of imbalance in the training set. But resampling can of course also be used to create an imbalance, which is what we need for Triskel. Akbani et al. found in [10] that for imbalanced datasets undersampling the majority class to eliminate the bias leads to good performance, although some of the training examples are discarded.

In preliminary experiments, we tried over- and undersampling to create biased classifiers. We found that creating the bias through undersampling does not hurt the overall performance of Triskel, even if as little as 10% of the training instances of one class are kept. For some datasets, the performance was even slightly better than the approach with oversampling. Additionally, because we drop 90% of the instances for one class, training becomes faster. Therefore we decided to use undersampling with a 10% undersampling rate for our final experiments.

Relation to Covering. There is a loose relationship between Triskel and rule covering algorithms (e.g. [11]). A covering algorithm tries to identify rules with high precision that cover a large number of (ideally uniformly positive or negative) training examples. These training examples are then removed from the training set, as they are covered by the rule, and rule learning continues until all examples are covered. In Triskel, identifying easy instances using biased classifiers could be seen as covering positive and negative instances, as these instances are then removed from the training set from which the arbiter is learned.

Comparison with Boosting. Shapire's original boosting algorithm [4] uses three classifiers: The first one is trained on the original dataset, the training set for the second classifier consists equally of instances that were classified correctly by the first classifier and instances that were incorrectly classified. A third classifier is trained on instances where the first two classifiers disagree. The predictions are combined by voting. In our algorithm we follow up on this idea, however the way we create the first two classifiers is fundamentally different. Also unlike the original boosting, we can use multiple iterations. This results in an ensemble containing more than three classifiers similar to AdaBoost.

Both AdaBoost [6] and Triskel try to enhance the decision surface of the ensemble by focusing on hard instances. The main difference between the two algorithms is, however, how the hard instances are defined. In AdaBoost, the hard instances are defined as the instances where the base classifier makes mistakes. In Triskel, the hard instances are defined as the instances that cannot be classified "confidently", where we assume that we can classify an instance "confidently", if the biased classifiers agree on its label.

4 Experimental Results

We evaluated Triskel on several multi-class datasets from the well-known UCI repository. Because of its very good accuracy, we chose AdaBoost as the benchmark ensemble algorithm for our experiments. We used SMO [12] as a base classifier, again because of its good performance. However, when comparing ensemble methods, accuracy is not the only important factor. The reduced error of ensemble algorithms comes at the price of a greater computational effort. Therefore, time and memory consumption has to be compared as well. Both are usually related to the ensemble size.

Algorithm 2 Comparison of Triskel (left) to AdaBoost (right)

```
/* To train on $\{\ldots,(x_i,y_i),\ldots\}$ */
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
    $h_t^+ =$ Learn(weights $D_{t-1}$, biased +1)
    $h_t^- =$ Learn(weights $D_{t-1}$, biased -1)
    $\alpha_t = 2^{K-t}$
    for each instance $i$ do
        $\Delta_{t,i} =$
        $= \begin{cases} W_{easy} & \text{if } h_t^+(x_i) = h_t^-(x_i) = y_i \\ W_{hard} & \text{otherwise} \end{cases}$
        $D_t(i) = D_{t-1}(i) \cdot \Delta_{t,i}$ and normalise
    end for
end for
$h_{K+1} =$ Learn(weights $D_K$, unbiased)
$\alpha_{K+1} = 1$
/* To classify instance $x$ */
return $y = \text{sign}\left[\sum_{t=1}^{K+1} \alpha_t h_t^*(x)\right]$,
```

```
/* To train on $\{\ldots,(x_i,y_i),\ldots\}$ */
$D_0(i) = 1/N$ for each instance $i$
for $t = 1, 2, \ldots, K$ do
    $h_t =$ Learn(weights $D_{t-1}$, unbiased)
    $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$,
    where $\epsilon_t = \sum_i D_{t-1}(i)[\![y_i \neq h_t(x_i)]\!]$
    for each instance $i$ do
        $\Delta_{t,i} =$
        $= \begin{cases} \epsilon_t/(1-\epsilon_t) & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
        $D_t(i) = D_{t-1}(i) \cdot \Delta_{t,i}$ and normalise
    end for
end for
/* To classify instance $x$ */
return $y = \text{sign}\left[\sum_{t=1}^{K} \alpha_t h_t(x)\right]$
```

Because SMO can only handle binary problems, we had to choose a mode of splitting the multi-class problems into binary classification tasks. In all but one configurations we decided to use a one-against-one scheme: A binary classifier is contructed for all pairwise combinations of two classes. This means that for a dataset with k classes it is necessary to train $\frac{k(k-1)}{2}$ classifiers. Note that on datasets with more than 3 classes, this setup is computationally more expensive than a one-against-all scheme, but generally leads to a much better performance.

In conjunction with Triskel it is possible to use a compromise between one-vs-all and one-vs-one methods. We call this extension Triskel-M. For each class, a binary problem is created in order to separate this class ('positive instances') from all others ('negative instances'). These classifiers are biased towards high precision on the positive class and used similar as in binary Triskel: If exactly one of the biased classifiers predicts positive, this prediction is returned. If more than one or none of the biased classifiers predict positive, the prediction of the arbiter is returned. The arbiter is trained in one-vs-one mode to achieve a better accuracy. In our experiments, we used Triskel-M with $W_{easy} = 0$ and 1 round (denoted as Triskel-M1).

For AdaBoost, boosting the binary classifiers individually yielded a better performance than using AdaBoost-M1 [6].

We used a standard SMO as baseline. We used three different AdaBoost-ensembles with 3, 10 and 50 rounds. We compared these against standard Triskel with 1 round and discarding easy instances for the arbiter ($W_{easy} = 0$) (Triskel-1) and against Triskel with weighting ($W_{easy} = 1/2; W_{hard} = 2$) with 2 and 4 rounds (denoted as Triskel-W2 and Triskel-W4). Note that for a (binary) Triskel the actual ensemble size is twice the number of rounds plus one.

We used the Weka framework [13] to conduct our experiments. We evaluated all algorithms using 10-fold cross-validation with 10 randomized repetitions for statistical significance testing, using a corrected resampled t-test as implemented in the Weka experimenter. [1]

The experiments show that AdaBoost with 50 rounds does not improve the accuracy over AdaBoost with 10 rounds when using SMO as a base classifier. Triskel-W4 outperforms AdaBoost with 3 significant wins out of the 15 datasets used. This quality improvement comes at the price of higher training cost when compared to AdaBoost-10. However, it is still faster than AdaBoost-50. Triskel-W2 (i.e. with an ensemble size of 5 classifiers) achieves a performance that is comparable to AdaBoost-10 (2 wins, 2 losses), but is significantly faster.

As expected, the M1 setup for Triskel is both the least accurate but also the fastest ensemble method. Although the biased classifiers are only trained in a one-against-all mode, the ensemble can still sigificantly outperform the base SMO in one-against-one mode on the anneal.ORIG, hypothyroid and segment datasets. Because of its one-against-all nature, this setup of Triskel can even be faster than one-against-one SMO, especially on large datasets (here on the audiology, hypothyroid and soybean datasets), while not hurting accuracy.

Figure 2 illustrates the relation between training time and accuracy for the algorithms on four typical datasets. The data points on the Triskel line correspond to (from fastest to slowest) Triskel-M1, -1, -W2 and -W4, while the data points for AdaBoost show the setup for 3, 10 and 50 rounds. Note that in most cases the line for Triskel is above the AdaBoost line, indicating that Triskel offers a better trade-off between accuracy and speed. Triskel achieves greater accuracy in the same time, and the same accuracy can be reached faster. Furthermore, note that the highest accuracy for Triskel is usually above the highest accuracy for AdaBoost, indicating that, given enough time, Triskel can typically outperform any setting of AdaBoost.

5 Conclusion

Current and Future Work. We are currently researching many aspects of Triskel that we could only sketch shortly in this paper. For example, note that for the experiments presented in this paper, we have used undersampling as the only method of generating the biased classifiers. In future work we would like to explore the space of possible methods for generating bias, such as setting the bias of the classifier directly by means of thresholding, or to use oversampling with artificially created instances as used in the SMOTE algorithm [14].

We are currently working on a covering-inspired way of generating the biased classifiers. To train a classifier that is biased towards high precision on positive instances, we train multiple versions of a base classifier while iteratively removing instances from the training set that the base classifier predicts as negative (i.e. instances that are covered). On classification time, an instance is predicted as positive only if all ensemble

[1] Due to space restrictions we cannot present all our experimental results in this paper. Our WEKA-based implementation of Triskel and more experimental results are available from our website `http://moguntia.ucd.ie/projects/triskel/`

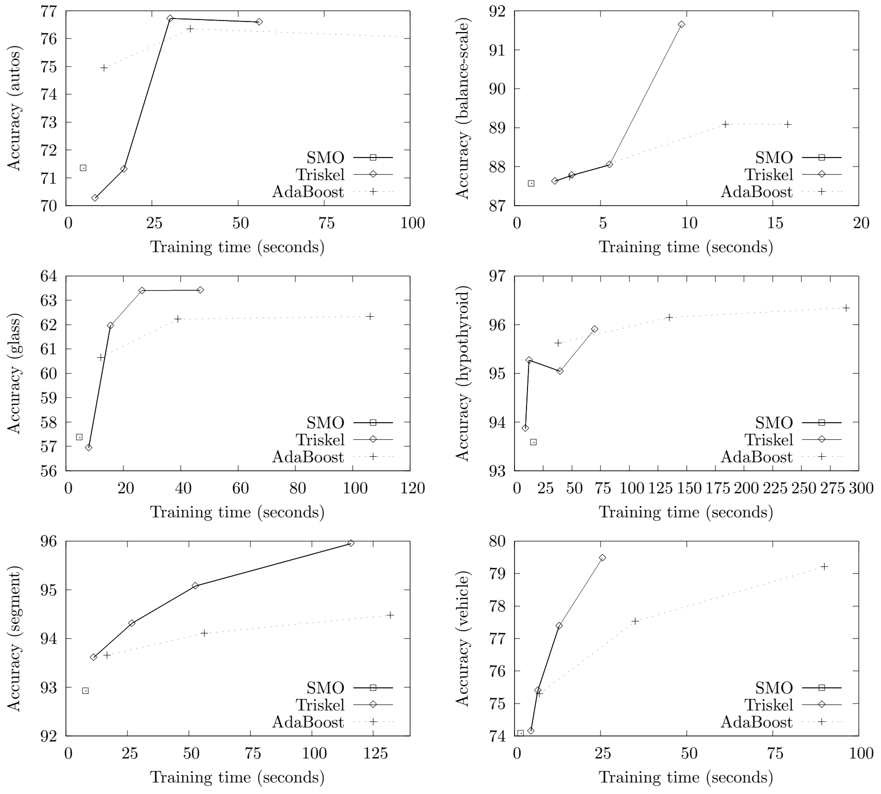

Fig. 2. Accuracy and training time for SMO, Triskel-M1, Triskel-1, -W2, -W4, AdaBoost-3, -10 and -50 on the (from left to right and top to bottom) “autos”, “balance-scale”, “glass”, “hypothyroid”, “segment” and “vehicle” datasets.

members classify it as positive. Achieving high precision on negative examples is symmetric, and the generalisation towards multi-class datasets is straightforward. Fig. 3 shows the covering-based biased classifier in pseudo-code.

This covering-like approach to biasing classifiers is more expressive than simple resampling approaches, because it is an ensemble itself. A Triskel classifier with covering-based biased classifiers is able to learn the correct hypothesis for a “slanted checkerboard” dataset (see Fig. 4), which is another example of a dataset that confounds many other algorithms. In preliminary experiments on real-world datasets, covering-like biasing improved classification accuracy slightly. On the other hand, this expressiveness is bought with the need for more ensemble members.

Summary. We have presented a novel ensemble learning algorithm called Triskel that makes use of biased classifiers to separate “easy” and “hard” instances. In its iterative nature, it is similar in style to Boosting methods, while the way Triskel separates easy and hard instances is loosely related to covering algorithms.

```
$D_0(i) = 1/N$ for each instance $i$
**for** $t = 1, 2, \ldots, K$ **do**
  $h_t$ = learn(instances weights $D_{t-1}$)
  **for** each instance $i$ **do**
    $D_t(i) = \begin{cases} 1 \text{ if } h_t(x_i) = 1 \\ 0 \text{ otherwise} \end{cases}$
  **end for**
**end for**
/* To classify instance $x$ */
**for** $t = 1, 2, \ldots, K$ **do**
  **if** $h_t(x) = -1$ **then**
    halt and return $-1$
  **end if**
**end for**
return 1
```

Fig. 3. Cover Negatives Algorithm

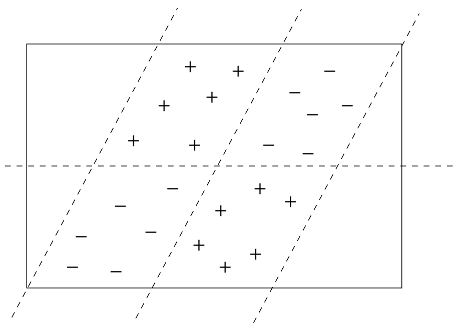

Fig. 4. “Slanted checkerboard” data set

Empirical results suggest that, compared to AdaBoost, Triskel offers a better trade-off between accuracy and speed. Furthermore, the experiments show the maximum accuracy that can be achieved with Triskel is higher than the accuracy of AdaBoost.

References

1. Hansen, L.K., Salamon, P.: Neural Network Ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990) 993–1001
2. Dietterich, T.G.: Ensemble Methods in Machine Learning. In: First Int. Workshop on Multiple Classifier Systems, New York (2000)
3. Breiman, L.: Bagging predictors. Machine Learning **24** (1996) 123–140
4. Shapire, R.E.: The Strength of Weak Learnability. Machine Learning **5** (1990) 197–227
5. Freund, Y.: Boosting a Weak Learning Algorithm by Majority. Information and Computation **121** (1995) 256–285
6. Freund, Y., Shapire, R.E.: A Decision-Theoretic Generalization of On-Line Learning. Journal of Computer and System Sciences **55** (1997) 119–139
7. Vapnik, V.N.: Vosstanovlenije Zavisimostej po Empiricheskim Dannym. Nauka (1979) In Russian.
8. Murphey, Y.L., Guo, H., Feldkamp, L.A.: Neural learning from unbalanced data. Appl. Intell. **21** (2004) 117–128
9. Muhlbaier, M., Topalis, A., Polikar, R.: Learn++.MT: A New Approach to Incremental Learning. In: 5th Int. Workshop on Multiple Classifier Systems, Cagliari, Italy (2004)
10. Akbani, R., Kwek, S., Japkowicz, N.: Applying Support Vector Machines to Imabalanced Datasets. In: 15th European Conference on Machine Learning (ECML), Pisa (2004)
11. Fürnkranz, J.: Separate and Conquer Rule Learning. Art. Intell. Review **13** (1999) 3–54
12. Platt, J.C.: 12. In: Fast Training of Support Vector Machines using Sequential Minimal Optimization. MIT Press (1999) 185–208
13. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (1999)
14. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: Synthetic Minority Over-sampling TEchnique. Journal of Artificial Intelligence Research **16** (2002) 341–378