Pattern Recognition Letters 33 (2012) 92–102

Contents lists available at SciVerse ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

A direct boosting algorithm for the *k*-nearest neighbor classifier via local warping of the distance metric

Toh Koon Charlie Neo, Dan Ventura [*]

Department of Computer Science, Brigham Young University, Provo, UT 84602, USA

ARTICLE INFO

Article history:
Received 17 May 2010
Available online 29 September 2011
Communicated by B. Kamgar-Parsi

Keywords:
Classification
*k*NN
Boosting
Distance measures

ABSTRACT

Though the *k*-nearest neighbor (*k*-NN) pattern classifier is an effective learning algorithm, it can result in large model sizes. To compensate, a number of variant algorithms have been developed that condense the model size of the *k*-NN classifier at the expense of accuracy. To increase the accuracy of these condensed models, we present a direct boosting algorithm for the *k*-NN classifier that creates an ensemble of models with locally modified distance weighting. An empirical study conducted on 10 standard databases from the UCI repository shows that this new Boosted *k*-NN algorithm has increased generalization accuracy in the majority of the datasets and never performs worse than standard *k*-NN.

1. Introduction

The *k*-nearest neighbor (*k*-NN) pattern classifier is an effective learner for general pattern recognition domains. *k*-NN classifiers are appealing because of their conceptual simplicity, which makes them easy to implement. *k*-NN classifiers allow new information to be easily included at runtime and are thus useful for applications that collect users' feedback. Moreover, it is proven that the asymptotic error rate of the *k*-NN rule has an upper bound that is at most twice that of the Bayes optimal error (under some continuity assumptions on the underlying distributions) (Cover and Hart, 1967). (Note: strictly speaking, this bound applies to the case $k = 1$; as k grows large, this bound approaches the Bayes optimal error rate.)

However, the *k*-NN classifier is not without its drawbacks. One of these is the need for a large amount of storage space due to the fact that it has to store all training instances as its model. This also leads to large computation requirements during classification. There are indexing techniques that would help in reducing the computation requirement, but not the storage need. Several methods have been proposed to reduce the model size of the *k*-NN classifier.

One year after *k*-NN was introduced, the first algorithm to reduce the model size was introduced: the Condensed Nearest Neighbor rule (CNN) (Hart, 1968). CNN iterates through the training instances, adding instances that are classified incorrectly to the selected set, until all the instances are correctly classified. The CNN algorithm has been improved upon by several other algorithms. One such algorithm is the Reduced Nearest Neighbor (RNN) rule (Gates, 1972). RNN is an extension of CNN which iteratively removes an instance in the CNN selected set and tests for consistency (all training instances being correctly classified) till none can be removed. Another algorithm that reduces the model size is the Selective Nearest Neighbor (SNN) decision rule (Ritter et al., 1975). SNN searches for the smallest possible consistent subset; however, SNN requires exponential runtime. In order to improve the CNN algorithm by keeping only instances that are close to the decision boundary, Tomek introduces two modifications of CNN using the concept of *Tomek Links*, which are pairs of nearest neighbor instances of different class (Tomek, 1976). Tomek's algorithm first computes all the *Tomek Links* for a dataset. Next, it iterates through the dataset adding an instance from the *Tomek Links* set to the condensed set until all the instances from the dataset are correctly classified. Tomek's concept was improved upon in the GKA algorithm, which uses the concept of mutual nearest neighborhood for selecting patterns close to the decision boundaries and results in a smaller selected set than Tomek (Gowda and Krishna, 1979).

Recently there has been a resurgence in attempts to reduce the model size for the *k*-NN classifier. The Modified Condensed Nearest Neighbor (MCNN) algorithm uses centroids (the training instance closest to the mean of all the instances of the same class) as the starting point and thus is order independent, unlike CNN (Devi and Murty, 2002). The Pairwise Opposite Class-Nearest Neighbor (POC-NN) algorithm is the first model size reduction algorithm that is able to improve accuracy over the CNN algorithm while

* Corresponding author.
E-mail addresses: charlie.neo@gmail.com (T.K.C. Neo), ventura@cs.byu.edu (D. Ventura).

0167-8655/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2011.09.028

needing significantly less training time (Raicharoen and Lursinsap, 2005). POC-NN recursively separates, analyzes and selects prototypes until all regions are correctly classified. Fast Condensed Nearest Neighbor (FCNN) is another algorithm that attempts to reduce the training time while producing an accurate reduced model (Angiulli, 2005). FCNN is order independent and has sub-quadratic worst-case time complexity, requires few iterations to converge, and is likely to select points very close to the decision boundary. All of these algorithms have had varied degrees of success in reducing model size for the *k*-NN algorithm.

Model size reducing algorithms have come a long way since being introduced; however, using a model reduction algorithm usually results in a reduction in accuracy. Perhaps the most common way to increase accuracy is to employ an ensemble of models. For example, Alpaydin suggests voting over multiple condensed nearest neighbors (CNN) to increase accuracy, exploring simple and weighted voting, and also bootstrapping and partitioning of the training set before training (Alpaydin, 1997). Alpaydin is successful in increasing accuracy over a single CNN but only partially successful in increasing accuracy over the original *k*-NN algorithm which uses the entire training set. Multiple Feature Subset (MFS) is another algorithm for combining multiple NN classifiers (Bay, 1998). In MFS, each nearest neighbor classifier has access to all the patterns in the original training set, but only to a random subset of the features. An empirical study showed that MFS was effective in improving accuracy on many datasets, and was even competitive with boosted decision trees on some, but performed much worse in others. However, both Alpaydin's algorithm and MFS use randomness to facilitate creating uncorrelated errors. There is no guarantee that these errors will be uncorrelated, but it does happen frequently.

Okun and Priisalu experimented, in the context of *k*-NN classifiers, with a multi-view classification methodology where patterns or objects of interest are represented by a set of different views (a view is a subset of features) rather than the union of all views (all features) (Okun and Priisalu, 2005). They suggest that by replacing feature selection with multiple views, it is possible to dramatically lower computational demands for combining classifiers. Empirical study shows accuracy improvement can be achieved in the protein fold recognition problem over current best results.

Another approach creates an ensemble of nearest neighbor classifiers by sampling feature subsets according to a probability distribution calculated by the ADAMENN algorithm which estimates feature relevance (Domeniconi and Yan, 2004). Experiments show that it has an advantage over random feature selection when the number of features is large. Zhou and Yu experimented with bootstrap sampling and injecting randomness to distance metrics in order to create diverse *k*-NN models for use in an ensemble (Zhou and Yu, 2005a). Later they added attribute filtering (removing irrelevant attributes) and attribute subspace selection (random feature subset) to their algorithms (Zhou and Yu, 2005b). Using majority voting on ensembles of size 100, experiments show best results are achieved using various combinations of the four ways to perturb the models.

Boosting is popular as a somewhat more principled approach to ensemble creation. It is an iterative approach which re-weights misclassified instances, increasing their importance to subsequent models, which are then more likely to fix previous errors. A well-known boosting algorithm for classifiers is AdaBoost (short for Adaptive Boosting) (Freund and Schapire, 1995). AdaBoost is a general purpose boosting algorithm that can be used in conjunction with many other learning algorithms to improve their performance. AdaBoost (and its multi-class extensions M1 and M2) has been shown to be practical and effective in reducing error when used with the C4.5 algorithm (Freund and Schapire, 1996). However, it is shown that AdaBoost does not work well with a standard nearest neighbor classifier, as accuracy is reduced instead of increased (Valverde and Ferri, 2005). One reason why *k*-NN does not work with AdaBoost is because *k*-NN is a stable algorithm with low variance, resulting in the production of hypotheses with correlated errors during each iteration of AdaBoost. As a result, some research has been done to attempt to adapt AdaBoost for the *k*-NN classifier or increase its accuracy through various algorithms and methodologies.

For example, Freund and Schapire were the first to experiment with AdaBoost and a variant of a NN classifier (Freund and Schapire, 1996). They were able to speed up the classification by producing an ensemble of subsets of the training set sufficient to correctly label the whole training set. However, the method does not increase generalization accuracy. A somewhat similar approach, by Babu et al., uses AdaBoost and a prototype selection method based on clustering to select small subsets of data that are used to train ensemble members and does show good generalization results (Babu et al., 2004).

Instead of applying boosting directly, there are ways that boosting can indirectly benefit the *k*-NN algorithm. Athitsos and Sclaroff experimented with using boosting to learn a distance metric and applying it to a *k*-NN classifier (Athitsos and Sclaroff, 2005). This is achieved by using AdaBoost to learn a linear combination of a family of distance measures. Boosted Distance is another algorithm that also uses AdaBoost to learn a distance function for a *k*-NN classifier (Amores et al., 2006). Instead of a using a family of distance measures as inputs, Boosted Distance creates a new distance metric by comparing similarities between pairs of training instances. Empirical studies show that Boosted Distance is a superior implementation as it produces better results when compared with Athitsos and Sclaroff's algorithm, with AdaBoosted C4.5 and with a *k*-NN classifier employing other distance measures.

All the aforementioned algorithms attempt to increase the accuracy of the *k*-NN classifier either by creating *ad hoc* ensembles using randomness and/or feature manipulation, or by applying boosting indirectly by supplying the classifier with a boosted distance function. We present Boosted *k*-NN, a direct boosting algorithm specifically for the *k*-NN classifier. Boosted *k*-NN can increase generalization accuracy by creating ensembles of weighted instances, and it allows the user to control the trade-off between speed (model size) and accuracy.

2. Boosted *k*-NN

First we present the basic Boosted *k*-NN algorithm. Following the basic algorithm, we discuss 5 variants to the algorithm. Finally, empirical results of applying the basic Boosted *k*-NN algorithm and the 5 variants to several real datasets are presented.

2.1. The basic algorithm

The new Boosted *k*-NN algorithm follows the basic structure of AdaBoost by iterating through the training set to produce an ensemble of classifiers as the proposed hypothesis. However, during each iteration, instead of testing the current hypothesis with the whole set of training instances, Boosted *k*-NN holds out a training instance and classifies the held out instance using the rest of the training set. By using this "leave one out" method, each training instance will not help classify itself. This is important because by not leaving the instance out, the *k*-NN classifier (distance weighted) will always achieve 100% training set accuracy, and boosting is not possible. In traditional boosting, an instance that is misclassified would have its weight changed. However, in the case of the *k*-NN classifier, changing the weights of the misclassified instance will not help classify itself. Therefore, during each iteration and for each training instance that has been classified incorrectly, the algorithm will determine and modify the influence of its k-nearest neighbors.

At the start of the Boosted k-NN algorithm, a weight term w_i^0 is associated with each training instance, and the weight terms are all initialized to zero. Boosted k-NN then uses a k-NN classifier with distance weighting of $1/d$ to classify each instance using the rest of the training instances. Boosted k-NN will then modify the influence of the k nearest neighbors in the following manner during each iteration. If a query instance is classified incorrectly, Boosted k-NN will examine each of its k nearest neighbors, and modify their weights such that they are more likely to classify that instance correctly the next iteration. Thus, the modified weight term will increase the value of the vote for the correct class and decrease the value of the vote for the incorrect class. Each iteration through the training set, Boosted k-NN produces a model with modified weight terms. Boosted k-NN loops through the training set multiple times and returns an ensemble of models as the final hypothesis.

We begin with some preliminary definitions. Let S be a training set with n instances, and the ith instance s_i is described by (w_i^0, x_i, y_i) where w_i^0 is a weight term initialized to zero, x_i is a vector in the feature space X, and y_i is a class label in the label set Y. $d(x_1,x_2)$ is defined as the Euclidean distance between two instances s_1 and s_2 ($\|x_1 - x_2\|_2$). The distance between the query instance s_q and the ith instance is then defined as the function $D(s_q,s_i)$ where

$$D(s_q, s_i) = \frac{1}{\left(1 + e^{-w_i^t}\right) d(x_q, x_i)}$$

The distance function $D(s_q,s_i)$ is designed to be the product of a sigmoid function $1/\left(1 + e^{-w_i^t}\right)$ and the traditional distance function $1/d(x_q,x_i)$. When the weight term is set to the initial value of 0, the value of the distance function $D(s_q,s_i)$ is half that of the traditional distance function $1/d(x_q,x_i)$. Modifying the weight term will then change the value of the sigmoid function between 0 and 1 before it is multiplied with the traditional distance function. Constraining the weight term with a sigmoid function works well to prevent weights from modifying the distance function $D(s_q,s_i)$ too drastically and/or too quickly.

The pseudo code for the Boosted k-NN algorithm is shown in Algorithm 1. Given the training set S, number of iterations T, and weight update term λ, Boosted k-NN constructs an ensemble of up to T k-NN classifiers with modified weight terms. During each iteration t, a k-NN classifier is constructed by iterating through the weighted training set querying each instance against the rest of the training set. When an instance is misclassified, the weights of its k-nearest neighbors will be modified as follows. For each neighbor instance that belongs to a different class than the query instance, its weight term for the next iteration will be *decreased* by $\lambda/d(x_q,x_i)$, where $d(x_q,x_i)$ is defined as the Euclidean distance between the query instance and the nearest neighbor being modified. On the other hand, a neighbor that belongs to the same class as the query instance will have its weight for the next iteration *increased* by $\lambda/d(x_q,x_i)$. The modified weight term affects the distance function $D(s_q,s_i)$ by increasing the distance of neighboring opposite-class instances and decreasing the distance of neighboring same-class instances. The label of the query instance is the class that has the highest weighted sum of votes among its k nearest neighbors based on the distance function $D(s_q,s_i)$; therefore, modifying the weight terms in this way improves the chances of misclassified instances being correctly labeled the next iteration.

Algorithm 1. Boosted k-NN

```
1:  Inputs:
        $S = \{s_i\} = \{(w_i^0, x_i, y_i)\},\ T,\ \lambda$
2:  Initialize:
        $w_i^0 \leftarrow 0,\ i = 1 \ldots n$
        $S_0 \leftarrow S$
3:  for $t = 1$ to $T$ do
4:      $S_t \leftarrow S_{t-1}$
5:      for $s_q \in S_t$ do
6:        $N_q \leftarrow k$ nearest neighbors of $s_q$ using $D(s_q,s_i)$
7:        $\mathtt{label}(s_q) = \mathrm{argmax}_{y \in Y} \sum_{s_i \in N_q} D(s_q, s_i)\delta(y, y_i)$
8:        if $\mathtt{label}(s_q) \neq y_q$ then
9:          for $s_i \in N_i$ do
10:           if $y_i \neq y_q$ then
11:             $w_i^t \leftarrow w_i^t - \lambda/d(x_q, x_i)$
12:           else
13:             $w_i^t \leftarrow w_i^t + \lambda/d(x_q, x_i)$
14:     if $\mathtt{label}(s_q) = y_q,\ \forall s_q$ then
15:       break
16: return ensemble hypothesis $f(S_1,\ldots,S_t)$
```

We have considered that there are very rare cases where Boosted k-NN would not be able to perform modification of the weights. For example, if the training set is able to correctly label all its instance using the "leave one out" method during the first iteration, then no boosting is possible. In another rare case, all modifications of weights during an iteration may cancel each other out, and all weights return to their initialized value after an iteration through the training set. However, these cases are extremely rare and may occur only in trivial/toy problems.

A high-level example of what Boosted k-NN does is shown in Fig. 1. In Fig. 1(a), s_q is labeled incorrectly with k set to 5. Boosted k-NN then modifies the 2 nearest neighbors of the same class (blue[1] circle) by increasing their weights, and reduces the weight of the 3 nearest neighbors of the opposite class (red triangle), see Fig. 1(b). As a result, subsequent queries of s_q will return the correct label due to the new weighted distances. The Boosted k-NN algorithm locally modifies the decision surface by increasing or decreasing the influence of each instance in the model. The goal of modifying the weights is to alter the decision surface closer to the true solution.

After T ensembles have been created or all labels are correct during an iteration ($T = t$), Boosted k-NN returns T weighted k-NN classifiers as the final hypothesis. A voting mechanism (e.g. simple voting) is used on the ensemble to determine the class label for a query instance. Each k-NN classifier uses the same weighted distance function $D(s_q,s_i)$ to determine the k nearest neighbors, and the class with the highest sum of $D(s_q,s_i)$ will be returned as the label. That is, the class label for a query point s_q is calculated as $\mathrm{argmax}_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$.

2.2. Addressing sensitivity to data order

Since the original algorithm is sensitive to data ordering, we investigate two simple alternatives that ameliorate the problem. The original algorithm is sensitive to data ordering because as it iterates through the training instances w_i^t is updated so that the new weights affect the calculations for subsequent instances in same iteration. One way to eliminate the problem is to randomize the instances in the data set during each training iteration in the algorithm (between lines 4 and 5 in Algorithm 1).

[1] For interpretation of color in Fig. 1, the reader is referred to the web version of this article.

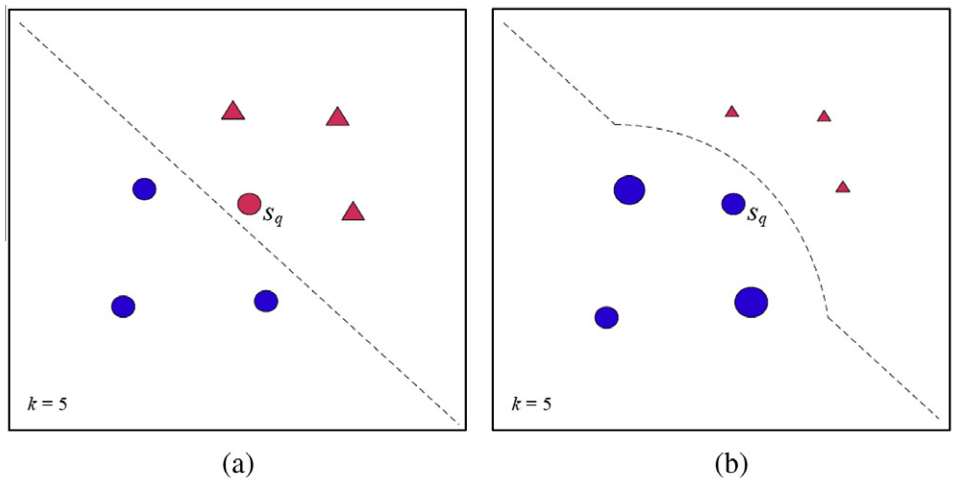

Fig. 1. Boosted k-NN modifying the decision surface. (a) At the beginning of the algorithm, weights are initialized to zero. During an iteration, query s_q is incorrectly labeled. (b) After the algorithm modifies the weights of the 5 nearest neighbors ($k = 5$), the shape of the decision surface is changed, and subsequent queries to s_q will be correct.

Another method for addressing the sensitivity to data order is to use a batch update method. Instead of updating the weight term w_i^t each iteration, we accumulate the changes in a weight change term $\Delta w_i^t = \pm\lambda/d(x_q, x_i)$ (replacing lines 11 and 13 in Algorithm 1), and update $w_i^t = w_i^t + \Delta w_i^t$ after iterating through the entire training set (after line 15). It is also necessary to initialize $\Delta w_i^t \leftarrow 0$ each time through the ensemble loop of line 3 (this initialization is done between lines 4 and 5).

2.3. Voting mechanism

In the original algorithm we suggest using simple voting where each k-NN classifier in the ensemble has an equal vote. An alternative to simple voting is error-weighted voting. The algorithm is modified such that the training accuracy for each iteration is recorded and returned together with the ensemble. The training accuracy for each iteration is then used to weight the vote for its corresponding k-NN classifier. This is accomplished by accumulating a count of misclassified instances (between lines 8 and 9 of Algorithm 1), converting this to a percentage of the total number of instances at the end of the iteration (before line 14) and including this percentage for each iteration in the final hypothesis (line 16). In addition, the count must be reinitialized each iteration (between lines 4 and 5).

2.4. Condensing model size

In Section 1, we discussed the need to reduce the model size for the k-NN classifier as it reduces both storage and computational requirements. Therefore, since Boosted k-NN increases the size of the model by creating an ensemble of weights, it is important that we investigate ways that can condense the model size of the Boosted k-NN algorithm while retaining the accuracy gain.

One of the ways that we can condense the model size of the Boosted k-NN algorithm is instead of using an ensemble, the algorithm could return the model with the optimal weights (weights giving the best training accuracy during an iteration). This is achieved by storing the best training accuracy seen so far while iterating up to T and returning only one model, $S_{optimal}$, where *optimal* is the loop with the best training accuracy. This is done by modifying Algorithm 1 to count each correctly classified instance (adding an ELSE statement to the IF of line 8) and adding functionality at the end of the iteration to compare this count with the best seen so far—if there is an improvement, the latest accuracy and iteration number are saved (new lines added before line 14). The hypothesis returned (line 16) is simply the best model seen in any of the iterations. Of course, the accuracy must be re-initialized each time through the loop (between lines 4 and 5).

Another way to condense model size is to average the weights of all the models in the ensemble and return just one k-NN classifier with the averaged weights. This is similar to the weight averaging concept in neural networks (Utans, 1996) and single layer perceptrons (Andersen and Martinez, 1999). After the algorithm has iterated up to T iterations and produced an ensemble of weights, the new Boosted k-NN with average weight algorithm then iterates through the ensemble, averaging the weights for each instance (new lines added just before line 16 of Algorithm 1). The new variant then returns only one model with the averaged weights (line 16).

3. Results and analysis

For the basic Boosted k-NN algorithm, we compare results to results obtained using regular k-NN, CNN and (for some datasets) the Boosted Distance algorithm. In the case of the Boosted Distance algorithm, we are not confident in our implementation of the algorithm, due to the fact that we found an error in the pseudo code in (Amores et al., 2006). Therefore, only results for the sonar, ionosphere and liver dataset which are obtained from Amores et al. (2006) are included in the comparison. (Please note that there are slight differences in the experiment setup.) For the variants of Boosted k-NN, we are interested in comparing them to the basic Boosted k-NN, and regular k-NN is included as a base line.

3.1. Experiment setup

We conducted an empirical study using 10 UCI datasets (Asuncion and Newman, 2007) to determine the effectiveness of our algorithm. Table 1 lists the datasets with the number of samples and number of attributes in each set. The datasets chosen are of various sizes and difficulty, and to avoid complicating distance calculations, we chose to select datasets with only real-valued attributes. In the case of the Vowel dataset, the speaker name and sex is removed.

For each experiment, we use 10-fold cross validation to evaluate the performance of each algorithm including k-NN, CNN and Boosted Distance. In the case where a dataset is pre-divided into training and test sets, they are combined into one pool before samples for each fold are randomly selected. We experimented with a range of values as input parameters to Boosted k-NN and its

Table 1
Datasets used in the experiments.

Dataset	Samples	Attributes	Classes	Distribution
Sonar	208	60	2	53/47
Ionosphere	351	34	2	63/37
Wine	178	13	3	Balanced
Liver	345	6	2	42/58
Vowel	990	10	11	Balanced
Segment	2310	19	7	Balanced
Vehicle	846	18	4	Balanced
Iris	150	4	3	Balanced
Glass	214	10	7	70/17/6/0/13/9/29
Diabetes	768	8	2	65/35

variants. For each dataset and each algorithm variant, we experimented with values of k from 1 to 15, λ from 1 to 0.005 and T set to 10 and 100. A total of 3600 experiments were run and each of them was 10-fold cross validated. Here we highlight various interesting empirical results.

3.2. Boosted k-NN

Fig. 2 shows the experimental results of Boosted k-NN in comparison to other algorithms. The results for the Boosted k-NN and the original k-NN shown in Fig. 2 are results chosen from the best of all the experiments with various parameters (e.g. pick the best k value). From Fig. 2 we observe that the results of Boosted k-NN are competitive with other algorithms for all ten datasets. Fig. 3 shows the accuracy gain of Boosted k-NN over regular k-NN. Boosted k-NN is able to increase generalization accuracy over regular k-NN in seven of the ten datasets, and of those seven, two datasets (sonar, ionosphere) exhibit statistically significant accuracy gains (computed using a paired permutation test with a p-value less than 0.05). Only one dataset (segment) shows a small loss in generalization accuracy, and two other datasets (vowel and iris) show no gain. Note that accuracies for all three of these datasets are in the high 90s; therefore, it is more difficult to increase accuracy in these cases.

One observation we make about Boosted k-NN is that it is less sensitive to the selection of k. Fig. 4 shows that Boosted k-NN is less sensitive to the selection of k in seven of the datasets, no different in one dataset and sightly worse in two datasets.

On the other hand, Boosted k-NN is sensitive to the selection of T and λ. Choosing a λ that is too big can actually cause the generalization accuracy to decrease. Fig. 5 shows that for the glass and liver dataset, the upper range of λ chosen in our experiments is too large for the datasets and this results in the decrease of accuracy. This is representative of what can happen to any dataset if the value of λ is too large. This decrease in accuracy can be compounded by multiple iterations of the algorithm causing the generalization accuracy to decrease significantly. On the other hand, choosing a λ that is too small can fail to produce enough shift in the weights to cause any changes in the generalization accuracy. In general, if we pick a suitable λ, then by increasing T (more models in the ensemble) we improve the results at the cost of increased storage and query times. Fig. 6 shows that from our experiments, 5 datasets show accuracy increases when T is increased and only 1 dataset has a decrease.

From the experiments we did not discover a one-size-fits-all value for λ. This is because each dataset has different characteristics. Datasets with smaller average distances between instances will require the choice of a smaller λ than datasets with larger average distances. With a suitable λ the algorithm can then converge to a stable solution. This is similar to the step size of the gradient descent algorithm—picking a step size that is too large will cause the algorithm to skip across the local minimum. In Boosted k-NN, each training instance will modify weights of its neighbors towards its respective desired local minimum; therefore, choosing a λ that is too large with respect to the average distance, will cause the weights to oscillate rather than converging to a good solution, while choosing a λ too small will negatively impact the time to convergence.

3.2.1. Using hold-out set for selection of λ

One standard way we can choose λ is to use a hold-out set. However, this method is computationally expensive due to the large range for λ that needs to be tested. In order to demonstrate the idea, we conducted an experiment trying a range for λ from 10 to 10^{-8} using the ionosphere dataset with T set to 10. We use the 10-fold cross validation method to compute the accuracies, and for each fold we use 10% of the training set as the hold-out set. Fig. 7 shows the hold-out and test set accuracies of the experiment. The graph shows that the hold-out set accuracy closely tracks the test set accuracy, demonstrating that the hold out set accuracy is a good indicator for selecting a suitable λ that will likely produce good test set accuracy.

The result of Fig. 7 corroborates the results we obtained from the experiment described in Section 3.2. Both experiments indicate that a λ value of 0.1 produces good results and a λ value of 0.01 and smaller produces poorer results.

3.3. Boosted k-NN with randomized data order

Fig. 8 shows the experiment results for Boosted k-NN with randomized data order compared to the basic Boosted k-NN and

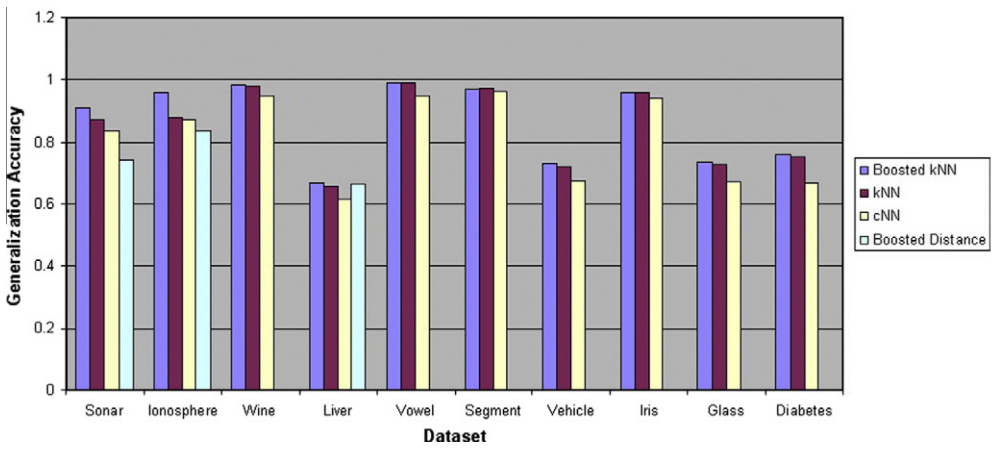

Fig. 2. Boosted k-NN vs. other algorithms. This graph shows the generalization accuracy of Boosted k-NN, regular k-NN, CNN and (sometimes) Boosted Distance on 10 datasets.

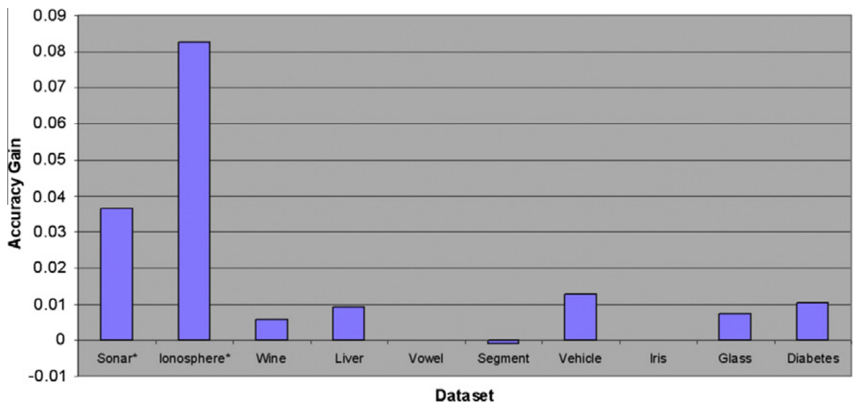

Fig. 3. Absolute accuracy gain of Boosted *k*-NN over regular *k*-NN. Dataset names with an asterisk indicate statistically significant accuracy gains.

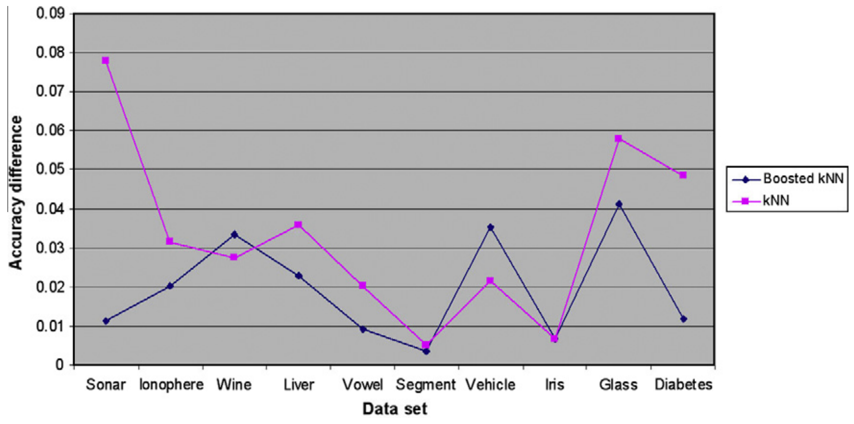

Fig. 4. Accuracy range of Boosted *k*-NN and regular *k*-NN. This graph plots the accuracy difference for 10 datasets with *k* ranging from 1 to 15.

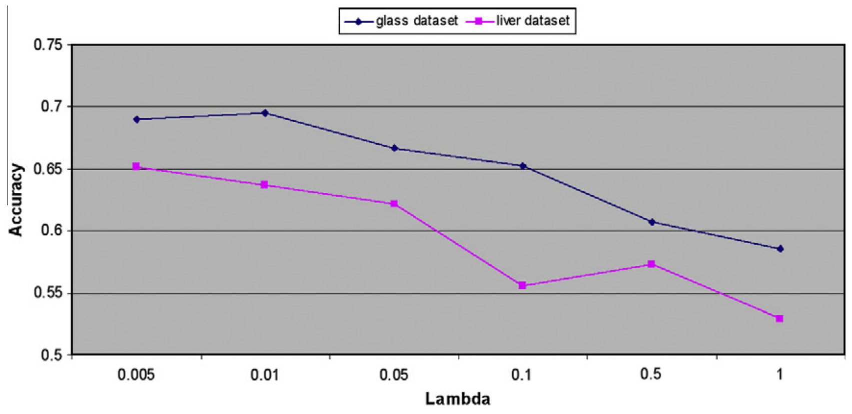

Fig. 5. Large λ causes accuracy decrease. Choosing larger values for λ causes the accuracy of the liver and glass datasets to decrease.

regular *k*-NN. From the graph we can see that Boosted *k*-NN with randomized data order performs worse than the basic algorithm in 6 of the datasets (statistically significant accuracy lost on sonar and ionosphere datasets) and posts insignificant or no gain on 3 of the datasets. On the iris dataset Boosted *k*-NN with randomized data order performs 1.3% better than the basic algorithm.

The experiment results show that it is not advisable to use Boosted *k*-NN with randomized data order over the basic algorithm. Randomizing the data order does not seem to help the algorithm improve generalization accuracy. In fact, for most of the datasets it actually prevents the algorithm from converging to a good solution.

3.4. Boosted k-NN with batch update

In Fig. 9, Boosted *k*-NN with batch update is compared with the basic Boosted *k*-NN and regular *k*-NN. The graph shows that in comparison with the basic algorithm, Boosted *k*-NN with batch update performs poorly on 3 of the datasets, better on 1 dataset, and performs the same for the rest. Although Boosted *k*-NN with batch update performs better than Boosted *k*-NN with randomized data order, it is still not as consistent as the basic algorithm. Boosted *k*-NN with batch update shows statistically significant accuracy loss on the sonar dataset when compared to the basic Boosted *k*-NN. However, Boosted *k*-NN with batch update still has statistically

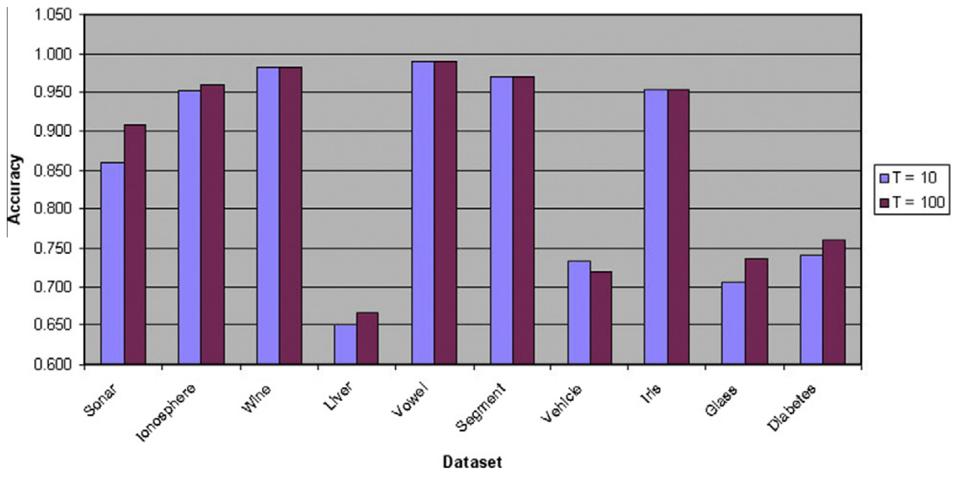

Fig. 6. Increasing the value of T can increase accuracy if a suitable λ is chosen.

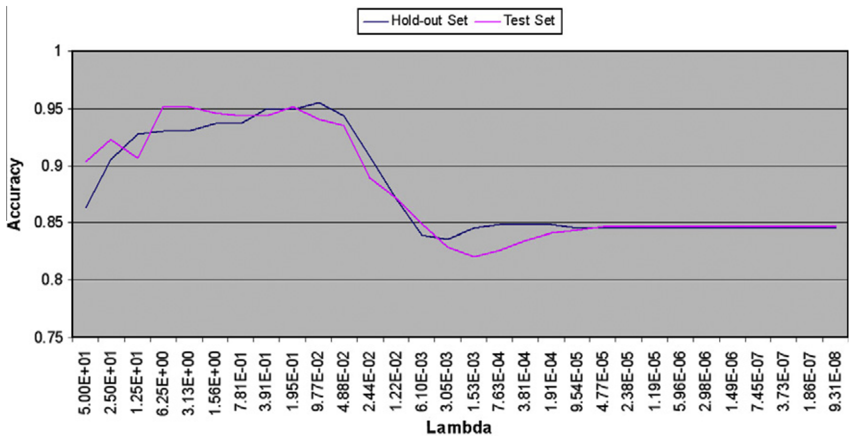

Fig. 7. Using a hold-out set for selection of λ. Accuracy and the value of λ are compared for the hold-out and test sets on the ionosphere dataset. From the results, we can see that the hold-out set accuracy is a good indicator for picking a suitable λ.

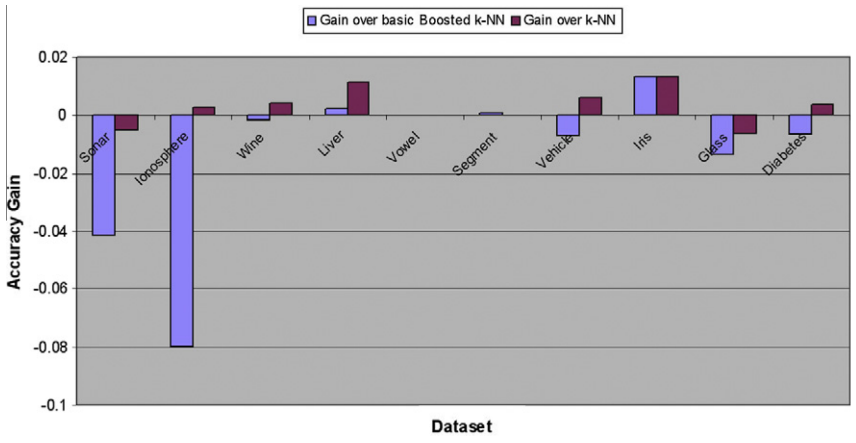

Fig. 8. Boosted k-NN with randomized data order. The accuracy gain of Boosted k-NN with randomized data order vs. basic Boosted k-NN and regular k-NN. The randomized version does not compare favorably with the basic algorithm. In the sonar and ionosphere datasets, Boosted k-NN with randomized data order shows statistically significant accuracy loss when compared to the basic Boosted k-NN.

significant accuracy gain over regular k-NN in the ionosphere dataset. Although not statistically significant, k-NN with batch update also performs much better in the liver dataset, achieving 3.2% better generalization accuracy than regular k-NN and 2.3% better than the basic Boosted k-NN algorithm.

Boosted k-NN with batch update does a better job than the basic algorithm at avoiding local minimum, moving instead toward a global minimum. This is because when weights are updated incrementally, there is a higher chance that it will be trapped in a local minimum, whereas batch update sums changes to weights from all

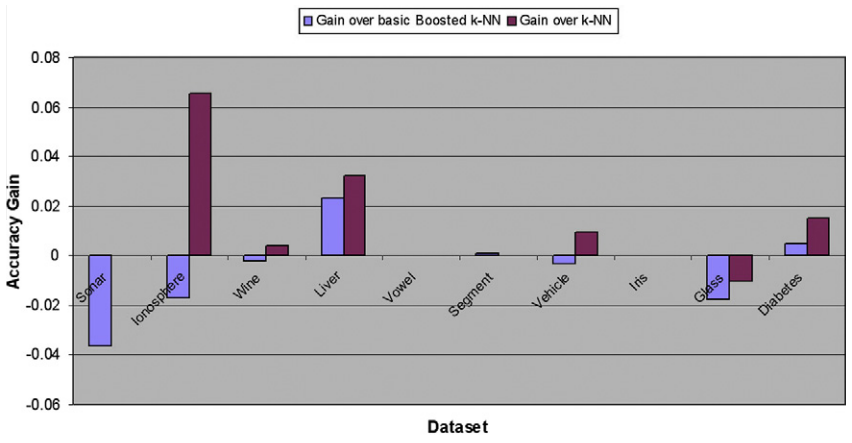

Fig. 9. Boosted *k*-NN with batch update. The accuracy gain of Boosted *k*-NN with batch update vs. basic Boosted *k*-NN and regular *k*-NN. Boosted *k*-NN with batch update shows statistically significant accuracy loss in the sonar dataset when compared to the basic Boosted *k*-NN. However, Boosted *k*-NN with batch update still has statistically significant accuracy gain over regular *k*-NN in the ionosphere dataset.

training instances thus moving towards a more global solution. However, it requires more iterations to get to a stable solution as instances often "resist" each other, making the update slower than if it were an incremental update. Increasing λ may speed up the process; however, it may also cause problems when multiple iterations modify a weight in the same direction causing it to overshoot the desired solution. Boosted *k*-NN with batch update is perhaps worth trying if training time and model size is not an issue.

3.5. Boosted k-NN with error-weighted voting

Fig. 10 shows the experimental result for Boosted *k*-NN with error-weighted voting. We see from the graph that Boosted *k*-NN with error-weighted voting performs comparably with the basic Boosted *k*-NN algorithm on all ten datasets. This raises the question of why error-weighted voting does not improve performance of Boosted *k*-NN over simple voting. One of the reasons is because the variance of the error rates between the models is small (due to the fact that all interior instances are always classified correctly; therefore, only instances close to the decision surface affect the error rate). Another reason is the fact that a lot of parameter optimization is done on the experiment shown in Fig. 10; therefore, training accuracies are stable and do not vary much.

However, Boosted *k*-NN with error weighted voting would be useful in situations where the training accuracy is fluctuating a lot. For example, choosing an unsuitable λ will cause the training accuracies to fluctuate quite a bit between models. Thus by weighting the vote of each model by its training accuracy, the impact bad models have on the overall accuracy would be reduced.

3.6. Boosted k-NN with optimal weights

The experimental results for Boosted *k*-NN with optimal weights compared to the basic Boosted *k*-NN and regular *k*-NN are shown in Fig. 11. There are no statistically significant differences between Boosted *k*-NN with optimal weights and the basic Boosted *k*-NN. From the graph we see that Boosted *k*-NN with optimal weights, in 8 of the datasets, exhibits less than 1% difference in generalization accuracy compared to the basic Boosted *k*-NN algorithm. Boosted *k*-NN with optimal weights performs worse on only the diabetes dataset, and performs somewhat better on the iris dataset.

The results are encouraging considering that Boosted *k*-NN with optimal weights uses only one set of weights rather than an ensemble. Boosted *k*-NN with optimal weights reduces both the storage requirement and query time over the basic algorithm. The space requirement for Boosted *k*-NN with optimal weights is $O(nm)$, which is the same as regular *k*-NN. In contrast, the basic Boosted *k*-NN algorithm requires $O(nm + nT)$ space, where n is the number of instances, m is the number of features and T is the number of training iterations. Therefore, Boosted *k*-NN with optimal weights is useful when boosting is needed but not at the expense of increased storage or query time.

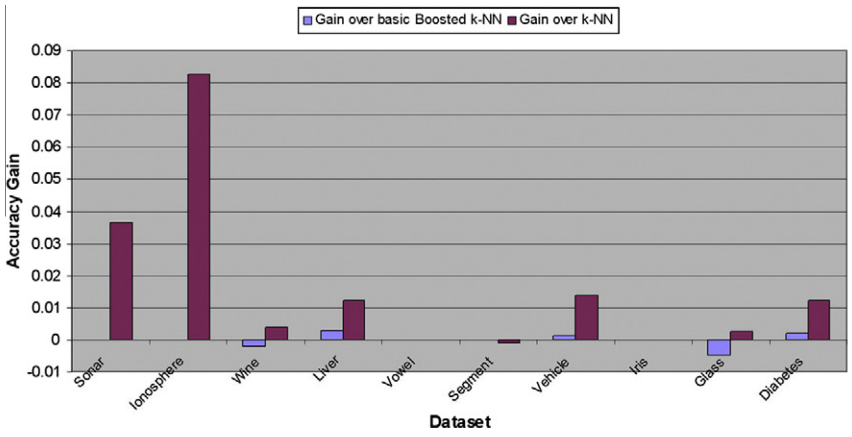

Fig. 10. Boosted *k*-NN with error-weighted voting. The accuracy gain of Boosted *k*-NN with error-weighted voting vs. basic Boosted *k*-NN and regular *k*-NN. There are no statistically significant differences between Boosted *k*-NN with error-weighted voting and the basic Boosted *k*-NN.

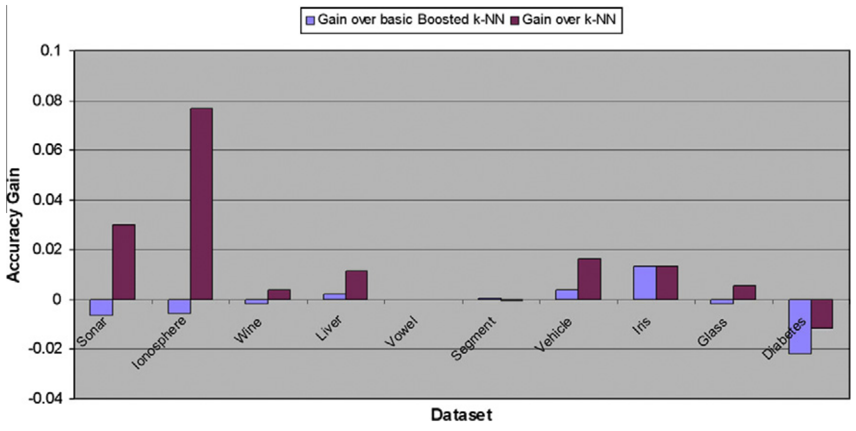

Fig. 11. Boosted k-NN with optimal weights. The accuracy gain of Boosted k-NN with optimal weights vs. basic Boosted k-NN and regular k-NN. There are no statistically significant differences between Boosted k-NN with optimal weights and the basic Boosted k-NN. This variant reduces storage requirements while maintaining accuracy gains.

3.7. Boosted k-NN with averaged weights

The last algorithm variant we experimented with is Boosted k-NN with averaged weights. Fig. 12 shows the results. There are no statistically significant differences between Boosted k-NN with averaged weights and the basic Boosted k-NN. Only on 2 datasets (sonar and diabetes) did it lose generalization accuracy of more than 1%, and it performs just as well on the rest of the datasets.

However, there is a potential for performance decreases with Boosted k-NN with averaged weights. The algorithm produces poor results when the input T is large. This is because when the number of models produced during training is large, it is very likely to produce groups of models that are converging to different local minima. This also happens if the value of λ is large. By averaging the weights in these scenarios, the algorithm can produce results that are not close to any correct solution. Therefore, it is not advisable to use Boosted k-NN with averaged weights unless both T and λ are small enough that the algorithm will not likely produce models converging to different local minima.

4. Scaling up

Because changing instance weights may result in neighborhood modification, the Boosted k-NN algorithm can be computationally expensive because it must (in theory) consider the possibility that any instance may (eventually) be the neighbor of any other instance (with enough weight modification). However, in practice, it does not seem likely that such extremes will occur. Therefore, to save computation time, we consider a modification, called *throttling*, in which we limit the set of possible neighbors for each instance. In the most extreme case, we set the throttle limit $n = k$ and an instance may never change it's original k neighbors (though, of course, their weights will still be changed during boosting). If this limit is relaxed, the set of possible neighbors grows until n reaches the size of the data set and we have the original Boosted k-NN algorithm.

Table 2 shows the effect on performance for different throttling levels for Boosted k-NN and compares this with unthrottled Boosted k-NN. Notice that even fairly severe throttling does not incur too significant a performance penalty (see also Table 3, which compares with traditional k-NN). Notice that even the throttled boosted results are never worse than k-NN and are often somewhat better.

Finally, we applied the throttled Boosted k-NN algorithm to the MAGIC Gamma telescope data set (19,020 instances, 10 attributes, 2 classes, 65/35 distribution), which involves the discrimination of primary gamma ray events from background hadron events. The best throttled boosted accuracy for the MAGIC data set was 0.846 (classical k-NN produces the same result). The best unthrottled boosted accuracy was 0.848.

5. Discussion

Experimental results show that compared to traditional k-NN, the Boosted k-NN algorithm maintains or increases generalization accuracy in 9 out of 10 datasets (the loss in the remaining dataset

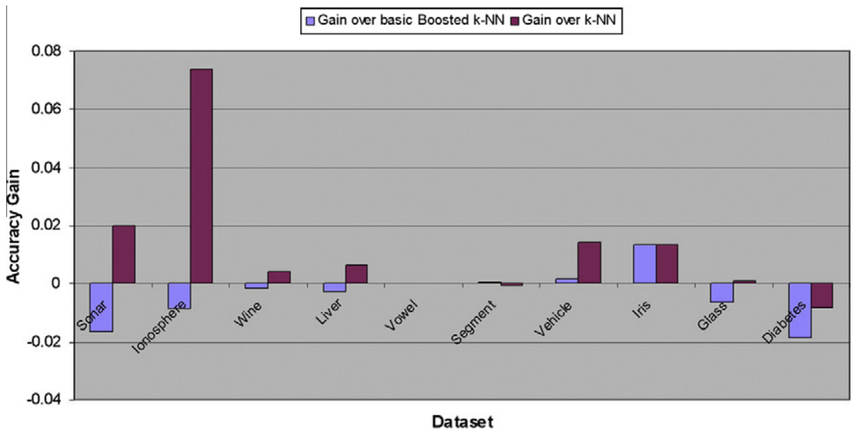

Fig. 12. Boosted k-NN with average weights. The accuracy gain of Boosted k-NN with average weights vs. basic Boosted k-NN and regular k-NN. There are no statistically significant differences between Boosted k-NN with averaged weights and the basic Boosted k-NN. This variant can be effective for model size reduction but must be used judiciously to avoid undesirable side effects.

Table 2
Limiting the number of potential neighbors can result in significant computational savings for large data sets without incurring a significant performance penalty. The first three columns represent different throttling levels with the rightmost column the full Boosted k-NN for comparison.

Dataset	Throttled boosted ($n = k$)	Throttled boosted ($n = 2k$)	Throttled boosted ($n = 5k$)	Boosted k-NN
Sonar	0.852	0.881	0.889	0.907
Liver	0.650	0.666	0.657	0.666
Vowel	0.990	0.990	0.990	0.990
Wine	0.988	0.984	0.982	0.984
Diabetes	0.754	0.753	0.757	0.761
Iris	0.967	0.960	0.967	0.960
Ionosphere	0.952	0.949	0.949	0.960
Vehicle	0.719	0.732	0.729	0.733
Segment	0.971	0.971	0.971	0.971
Glass	0.709	0.725	0.714	0.736

Table 3
Best throttled results (over all throttling levels) compared with classical k-NN. Even with fairly tight throttling, there is no loss of performance and in many cases some modest improvement.

Dataset	k-NN	Best throttled boosted ($n = k, 2k, 5k$)
Sonar	0.871	0.889
Liver	0.657	0.666
Vowel	0.990	0.990
Wine	0.978	0.988
Diabetes	0.751	0.757
Iris	0.960	0.967
Ionosphere	0.878	0.952
Vehicle	0.720	0.732
Segment	0.971	0.971
Glass	0.729	0.725

Table 4
Comparing Boosted k-NN to its variants. The paired values in columns 2 and 3 represent the number of datasets (out of 10 datasets) on which the algorithm does better or worse than regular k-NN or basic Boosted k-NN by at least 1% difference in generalization accuracy (e.g. Boosted k-NN with batch update when compared to regular k-NN, does better in 3 datasets and worse in 1 dataset).

Algorithm	vs. k-NN	vs. bBk-NN	Compression
basic Boosted k-NN	(+5, −0)	NA	No
bBk-NN + rand. data order	(+2, −0)	(+1, −2)	No
bBk-NN + batch update	(+3, −1)	(+1, −3)	No
bBk-NN + weighted voting	(+5, −0)	(+0, −0)	No
bBk-NN + optimal weights	(+5, −1)	(+1, −1)	Yes
bBk-NN + average weights	(+4, −0)	(+1, −2)	Yes

is insignificant). For 2 of the datasets, Boosted k-NN posts a statistically significant increase in generalization accuracy over the regular k-NN.

It should be noted that these two data sets both exhibit class imbalance (see Table 1), and that the performance of traditional k-NN is known to be susceptible to degradation in the presence of class imbalance (Japkowicz and Stephen, 2002; Hand and ronica, 2003).

At the same time, boosting has been shown to be a robust solution for many pathological data scenarios, including the case of class imbalance (Guo and Viktor, 2004; Sun et al., 2006). Our results here agree, and provide additional motivation for a direct method for boosting k-NN—directly incorporating boosting into the k-NN algorithm makes it more robust in the presence of class imbalance. Consider the class distributions shown in Table 1 and note from Figs. 2 and 3 that most of the exhibited improvement for Boosted k-NN occurs on unbalanced sets, with lesser improvements shown on the balanced data sets (vehicle being a bit of an exception). Additionally, the three data sets not improved (vowel, segment and iris) are all balanced, suggesting that a significant gain for Boosted k-NN is its robustness to class imbalance (inherent in the boosting).

In fact, it may be possible to make further improvements by combining Boosted k-NN with additional ideas for handling class imbalance, such as additional weighting schemes (Liu and Chawla, 2011) or oversampling the minority class(es) (Guo and Viktor, 2004; Sun et al., 2006).

And, very recent work suggests that in very noisy environments class imbalance may best be ameliorated using bagging rather than boosting (Khoshgoftaar et al., 2011), suggesting that perhaps an analogical directly bagged version of k-NN might be developed.

We also investigated five variants to the Boosted k-NN algorithm. Table 4 shows the high level comparison of performance between Boosted k-NN and its 5 variants. The paired values in columns 2 and 3 represent the number of dataset(s) (out of 10 datasets) for which the algorithm (variant) does better or worse than regular k-NN or basic Boosted k-NN by at least 1% difference in generalization accuracy. For example, Boosted k-NN with batch update when compared to regular k-NN, does better on three datasets and worse on one dataset. Out of the five variants, Boosted k-NN with optimal weights is the most interesting as it produces comparable results to the basic Boosted k-NN algorithm while reducing storage requirements and query time. Boosted k-NN with optimal weights requires only a single model with assigned weights instead of the ensemble of models produced by the basic Boosted k-NN algorithm. However, when the model size is not an important factor in choosing the algorithm, the basic Boosted k-NN is the algorithm of choice because it is less complicated than its variants, and it produces the best results.

6. Conclusion

We present an innovative algorithm, Boosted k-NN, that can boost the generalization accuracy of the k-nearest neighbor algorithm. This is accomplished via local warping of the distance metric using modified weights assigned to each instance. The weights are trained by iterating through the training set and classifying each instance against the rest of the training set. Incorrectly classified instances then update the weights of their neighbors so that they are more likely to correctly classify the instance during the following iteration. With the addition of these trained weights, the Boosted k-NN algorithm modifies the decision surface, producing a better solution.

We have several ideas for future research directions. One area we can explore is the use of a decay term for λ. It would be interesting to see if reducing the value of λ after each iteration through the training set would help eliminate the algorithm's sensitivity to the selection of λ. Having a decaying λ also might improve the algorithm's convergence to the desired solution.

Another improvement we would like to make to Boosted k-NN is to assign one weight to each feature in each instance instead of just one weight per instance. By assigning weights to features, we can modify the relationship between instances in each dimension in proportion to that dimension's importance. It would be interesting to see if assigning weights to features would help solve the feature selection problem or help eliminate *k*-NN's sensitivity to irrelevant features.

Last, we would like to explore using the trained weights in other applications. One application might be to use the weights for noise reduction. Trained weights that have their value reduced significantly might be a good indication that the associated data point is noise. Another possibility is to use the weights to label boundary instances or interior instances. With this labeling, we can perform model reduction by removing interior instances, reducing storage requirements and speeding up classification.

References

Alpaydin, E., 1997. Voting over multiple condensed nearest neighbors. Artif. Intell. Rev. 11 (1–5), 115–132 <citeseer.ist.psu.edu/article/alpaydin97voting.html>.

Amores, J., Sebe, N., Radeva, P., 2006. Boosting the distance estimation. Pattern Recognition Lett. 27 (3), 201–209.

Andersen, T., Martinez, T., 1999. The little neuron that could. In: Internat. Joint Conf. on Neural Networks, 1999, IJCNN '99, Washington, DC, USA, vol. 3, pp. 1608–1613.

Angiulli, F., 2005. Fast condensed nearest neighbor rule. In: ICML '05: Proc. 22nd Internat. Conf. on Machine Learning. ACM Press, New York, NY, USA, pp. 25–32.

Asuncion, A., Newman, D., 2007. UCI Machine Learning Repository. <http://www.ics.uci.edu/mlearn/MLRepository.html>.

Athitsos, V., Sclaroff, S., 2005. Boosting nearest neighbor classifiers for multiclass recognition. In: CVPR '05: Proc. 2005 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR'05) – Workshops. IEEE Computer Society, Washington, DC, USA, p. 45.

Babu, T.R., Murty, M.N., Agrawal, V., 2004. Adaptive boosting with leader based learners for classification of large handwritten data. In: Fourth Internat. Conf. on Hybrid Intelligent Systems, pp. 326–331.

Bay, S.D., 1998. Combining nearest neighbor classifiers through multiple feature subsets. In: Proc. 15th Internat. Conf. on Machine Learning. Morgan Kaufmann, San Francisco, CA, pp. 37–45 <citeseer.ist.psu.edu/bay98combining.html>.

Cover, T., Hart, P., 1967. Nearest neighbor pattern classification. IEEE Trans. Inform. Theory 13, 21–27.

Devi, F., Murty, M., 2002. An incremental prototype set building technique. Pattern Recognition 35, 505–513.

Domeniconi, C., Yan, B., 2004. Nearest neighbor ensemble. ICPR '04: Proc. Pattern Recognition, 17th Internat. Conf. on (ICPR'04), vol. 1. IEEE Computer Society, Washington, DC, USA, pp. 228–231.

Freund, Y., Schapire, R.E., 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In: European Conf. on Computational Learning Theory, pp. 23–37. <citeseer.ist.psu.edu/freund95decisiontheoretic.html>.

Freund, Y., Schapire, R.E., 1996. Experiments with a new boosting algorithm. In: Internat. Conf. on Machine Learning, pp. 148–156. <citeseer.ist.psu.edu/freund96experiments.html>.

Gates, W., 1972. The reduced nearest neighbor rule. IEEE Trans. Inform. Theory 18, 431–433.

Gowda, K.C., Krishna, G., 1979. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. IEEE Trans. Inform. Theory 25, 488–490.

Guo, H., Viktor, H.L., 2004. Learning from imbalanced data sets with boosting and data generation: The databoost-im approach. SIGKDD Explor. Newslett. 6, 30–39.

Hand, D.J., ronica, Vinciotti, 2003. Choosing *k* for two-class nearest neighbour classifiers with unbalanced classes. Pattern Recognition Lett. 24 (9–10), 1555–1562.

Hart, P., 1968. The condensed nearest neighbor rule. IEEE Trans. Inform. Theory 14, 515–516.

Japkowicz, N., Stephen, S., 2002. The class imbalance problem: A systematic study. Intell. Data Anal. J. 6 (5), 429450.

Khoshgoftaar, T.M., Hulse, J.V., Napolitano, A., 2011. Comparing boosting and bagging techniques with noisy and imbalanced data. IEEE Trans. Systems Man Cybernet. A: Systems Humans 41 (3), 552–568.

Liu, W., Chawla, S., 2011. Class confidence weighted *k*NN algorithms for imbalanced data sets. In: Proc. 15th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Part II, pp. 345-356.

Okun, O., Priisalu, H., 2005. Multiple views in ensembles of nearest neighbor classifiers. In: Proc. Workshop on Learning with Multiple Views (in Conjunction with the 22nd Internat. Conf. on Machine Learning), Bonn, Germany, pp. 51–58.

Raicharoen, T., Lursinsap, C., 2005. A divide-and-conquer approach to pairwise opposite class (poc) nearest neighbor algorithm. Pattern Recognition Lett. 26, 1554–1567.

Ritter, G., Woodruff, H., Lowry, S., Isenhour, T., 1975. An algorithm for a selective nearest neighbor decision rule. IEEE Trans. Inform. Theory 21, 665–669.

Sun, Y., Kamel, M.S., Wang, Y., 2006. Boosting for learning multiple classes with imbalanced class distribution. In: Proc. 6th Internat. Conf. on Datamining, pp. 592–602.

Tomek, I., 1976. Two modifications of cnn. IEEE Trans. Systems Man Cybernet. 6, 769–772.

Utans, J., 1996. Weight averaging for neural networks and local resampling schemes. In: AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms. AAAI Press, Portland, OR, pp. 133–138 <citeseer.ist.psu.edu/utans96weight.html>.

Valverde, M., Ferri, F., 2005. Experiments with adaboost and linear programming. In: III Taller de Minera de Datos y Aprendizaje (TAMIDA 2005), pp. 153–158.

Zhou, Z.H., Yu, Y., 2005a. Adapt bagging to nearest neighbor classifiers. J. Comput. Sci. Technol. 20, 48–54.

Zhou, Z.H., Yu, Y., 2005b. Ensembling local learners through multimodal perturbation. IEEE Trans. Systems Man Cybernet. B, 725–735.