2007-11-30

A Direct Algorithm for the K-Nearest-Neighbor Classifier via Local Warping of the Distance Metric

TohKoon Neo
Brigham Young University - Provo

A DIRECT BOOSTING ALGORITHM FOR THE K-NEAREST NEIGHBOR CLASSIFIER VIA LOCAL WARPING OF THE DISTANCE METRIC

by

Toh Koon Charlie Neo

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Toh Koon Charlie Neo

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Dan A. Ventura, Chair
Date	Christophe G. Giraud-Carrier
Date	Y. Dennis Ng

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Toh Koon Charlie Neo in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Dan A. Ventura Chair, Graduate Committee

Accepted for the Department

Date	Parris K. Egbert Graduate Coordinator

Accepted for the College

Date	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

A DIRECT BOOSTING ALGORITHM FOR THE K-NEAREST NEIGHBOR CLASSIFIER VIA LOCAL WARPING OF THE DISTANCE METRIC

Toh Koon Charlie Neo

Department of Computer Science

Master of Science

The k-nearest neighbor (k-NN) pattern classifier is a simple yet effective learner. However, it has a few drawbacks, one of which is the large model size. There are a number of algorithms that are able to condense the model size of the k-NN classifier at the expense of accuracy. Boosting is therefore desirable for increasing the accuracy of these condensed models. Unfortunately, there does not exist a boosting algorithm that works well with k-NN directly. 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.

ACKNOWLEDGMENTS

I would like to thank my wife, Chui Li, for believing in me and for her unceasing support throughout the long years of schooling.

Secondly I am very grateful to my advisor, Dr. Dan Ventura, for his guidance and help in completing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The k-nearest neighbor (k-NN) pattern classifier is an effective learner for general pattern recognition domains [1]. 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) [1].

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) [2]. CNN iterates through the training instances, adding instances that are classified incorrectly to the selected set. CNN iterates 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 [3]. 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 [4]. 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 [5]. 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 [6].

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 [7]. 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 needing significantly less training time [8]. 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 [9]. FCNN is order independent and has sub-quadratic worst-case time complexity, requires few iterations to converge, and it 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. One way to increase accuracy is to use boosting, which is a meta-learning algorithm. Boosting is based on a principle which states that current misclassified instances are re-weighted to be of higher importance. Therefore, subsequent models are likely to fix previous errors. Boosting therefore has the potential to restore the accuracy lost by a model reduction algorithm. A well-known boosting algorithm for classifiers is AdaBoost (short for Adaptive Boosting) [10]. 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 [11]. However, it is shown that AdaBoost does not work well with a standard nearest neighbor classifier, as accuracy is reduced instead of increased [12]. 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.M1 . A select few have tried to adapt AdaBoost.M1 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 [11]. 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.

Instead of using boosting to increase the accuracy of a k-NN classifier, a few have chosen to create ensembles of k-NN classifiers. Alpaydin suggests voting over multiple condensed nearest neighbors (CNN) to increase accuracy, exploring simple and weighted voting, and also bootstrapping (random sampling of instances with replacement) and partitioning of the training set before training [13]. 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 an algorithm for combining multiple NN classifiers [14]. 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 in many datasets, and was even competitive with boosted decision trees in some, but perform much worse in others. However, both Alpaydin's algorithm and MFS use randomness in hope of creating uncorrelated errors. There is no guarantee that these errors will be uncorrelated, but it does happen frequently.

Okun and Priisalu experimented 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) in the context of k-NN classifiers [15]. 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.

Ensemble of Nearest Neighbors in Weight-Driven Subspaces is an algorithm that 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 [16]. 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 [17]. Later they added attribute filtering (removing irrelevant attributes) and attribute subspace selection (random feature subset) to their algorithms [18]. 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.

Instead of applying boosting directly using a k-NN algorithm, 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 [19]. This is achieved by using AdaBoost to learn a weighted distance measure, that is, 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 [20]. 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, AdaBoost with C4.5 and a k-NN classifier with other distance measures.

All the aforementioned algorithms attempt to increase the accuracy of the k-NN classifier either by creating ensembles through 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.

Chapter 2

Boosted *k*-NN

We present a direct boosting algorithm for the k-NN algorithm call Boosted k-NN. First we present the basic Boosted k-NN algorithm. Following the basic algorithm, we discuss 5 variants to the algorithm. Experimental results of applying the basic Boosted k-NN algorithm and the 5 variants to several real datasets are presented in Chapter 3.

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 formally define the algorithm starting 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 $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$. The distance function $D(s_q, s_i)$ is design to be the product of a sigmoid function $\frac{1}{(1+e^{-w_i^t})}$ 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 $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$ 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.

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 modification 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.

Algorithm 1 Boosted k-NN

Inputs:

Training set S with n instances, where ith instance $= (w_i^0, x_i, y_i)$

T specifying the number of iterations

λ weight update term

Initialize:

All weight terms $w_i^0 = 0$, where $i = 1...n$

$S_0 = S$

for $t = 1$ to T **do**

set $S_t = S_{t-1}$

for s_q in S_t, query s_q where $q = 1...n$ **do**

Find k nearest neighbors using $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$, where $i = 1...n$, $s_q \neq s_i$, and $d(x_q, x_i) = \|x_q - x_i\|_2$

Return the label of $s_q = f(s_q) = \arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise

if $f(s_q) \neq y_q$ **then**

for $i = 1$ to k nearest neighbor **do**

if $y_i \neq y_q$ **then**

update $w_i^t = w_i^t - \lambda / d(x_q, x_i)$

else

update $w_i^t = w_i^t + \lambda / d(x_q, x_i)$

end if

end for

end if

end for

if all $f(s_q) = y_q$ (All labels are correct) **then**

$T = t$

break (EXIT loop)

end if

end for

return final hypothesis as an ensemble $f(S_1, ..., S_T)$

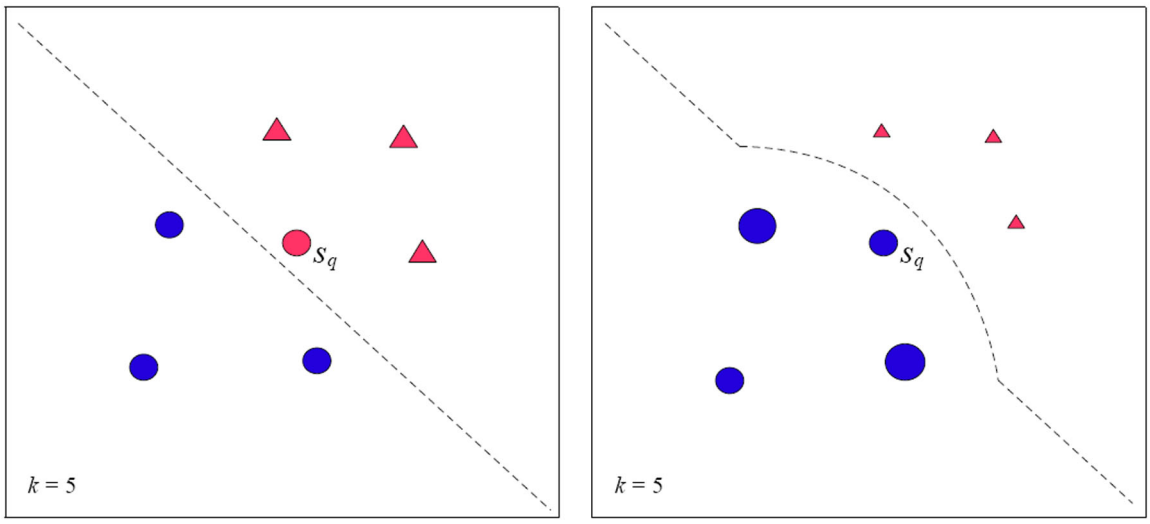

Figure 2.1: Boosted k-NN modifying the decision surface. (a) At the beginning of the algorithm, weights are initialized to zero. During an iteration, query s_q is incorrectly labeled. (b) After the algorithm modifies the weights of the 5 nearest neighbors ($k = 5$), the shape of the decision surface is changed, and subsequent queries to s_q will be correct.

A high-level example of what Boosted k-NN does is shown in Figure 2.1. In Figure 2.1(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 circle) by increasing their weights, and reduces the weight of the 3 nearest neighbors of the opposite class (red triangle). See Figure 2.1(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 $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$ 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 $\arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$.

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 the subsequent instance 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. The new Boosted k-NN with randomized data order shown in Algorithm 2 (changes from Algorithm 1 are underlined) is no longer data order dependent.

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 $\Delta w_i^t = \pm\lambda/d(x_q, x_i)$, and update $w_i^t = w_i^t + \Delta w_i^t$ after iterating through the entire training set. The pseudo code for Boosted k-NN with batch update is shown in Algorithm 3.

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

Algorithm 2 Boosted k-NN with randomized data order

Inputs:

Training set S with n instances, where ith instance $= (w_i^0, x_i, y_i)$

T specifying the number of iterations

λ weight update term

Initialize:

All weight terms $w_i^0 = 0$, where $i = 1...n$

$S_0 = S$

for $t = 1$ to T **do**

set $S_t = S_{t-1}$

randomize data order in S_t

for s_q in S_t, query s_q where $q = 1...n$ **do**

Find k nearest neighbors using $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$, where $i = 1...n$, $s_q \neq s_i$, and $d(x_q, x_i) = \|x_q - x_i\|_2$

Return the label of $s_q = f(s_q) = \arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise

if $f(s_q) \neq y_q$ **then**

for $i = 1$ to k nearest neighbor **do**

if $y_i \neq y_q$ **then**

update $w_i^t = w_i^t - \lambda / d(x_q, x_i)$

else

update $w_i^t = w_i^t + \lambda / d(x_q, x_i)$

end if

end for

end if

end for

if all $f(s_q) = y_q$ (All labels are correct) **then**

$T = t$

break (EXIT loop)

end if

end for

return final hypothesis as an ensemble $f(S_1, ..., S_T)$

Algorithm 3 Boosted k-NN with batch update

Inputs:

Training set S with n instances, where ith instance $= (w_i^0, x_i, y_i)$

T specifying the number of iterations

λ weight update term

Initialize:

All weight terms $w_i^0 = 0$, where $i = 1...n$

$S_0 = S$

for $t = 1$ to T **do**

set $S_t = S_{t-1}$

set $\Delta w_i^t = 0$, where $i = 1...n$

for s_q in S_t, query s_q where $q = 1...n$ **do**

Find k nearest neighbors using $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$, where $i = 1...n$, $s_q \neq s_i$, and $d(x_q, x_i) = \|x_q - x_i\|_2$

Return the label of $s_q = f(s_q) = \arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise

if $f(s_q) \neq y_q$ **then**

for $i = 1$ to k nearest neighbor **do**

if $y_i \neq y_q$ **then**

update $\Delta w_i^t = \Delta w_i^t - \lambda / d(x_q, x_i)$

else

update $\Delta w_i^t = \Delta w_i^t + \lambda / d(x_q, x_i)$

end if

end for

end if

end for

if all $f(s_q) = y_q$ (All labels are correct) **then**

$T = t$

break (EXIT loop)

end if

set $w_i^t = w_i^t + \Delta w_i^t$, where $i = 1...n$

end for

return final hypothesis as an ensemble $f(S_1, ..., S_T)$

iteration is then used to weight the vote for its corresponding k-NN classifier. Boosted k-NN with error-weighted voting is shown in Algorithm 4.

2.4 Condensing model size

In chapter 1, we discuss the need to reduce the model size for the k-NN classifier as it reduces both storage and computational requirements. However, Boosted k-NN increases the size of the model by creating an ensemble of weights. Therefore, 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 original 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 (Algorithm 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 [21] and single layer perceptrons [22]. 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. The new variant then returns only one model with the averaged weights. Algorithm 6 shows the pseudo code for Boosted k-NN with averaged weights.

Algorithm 4 Boosted k-NN with error-weighted voting

Inputs:

Training set S with n instances, where ith instance $= (w_i^0, x_i, y_i)$

T specifying the number of iterations

λ weight update term

Initialize:

All weight terms $w_i^0 = 0$, where $i = 1...n$

$S_0 = S$

for $t = 1$ to T **do**

set $S_t = S_{t-1}$

set $E_t = 0$

for s_q in S_t, query s_q where $q = 1...n$ **do**

Find k nearest neighbors using $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$, where $i = 1...n$, $s_q \neq s_i$, and $d(x_q, x_i) = \|x_q - x_i\|_2$

Return the label of $s_q = f(s_q) = \arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise

if $f(s_q) \neq y_q$ **then**

$E_t = E_t + 1$

for $i = 1$ to k nearest neighbor **do**

if $y_i \neq y_q$ **then**

update $w_i^t = w_i^t - \lambda / d(x_q, x_i)$

else

update $w_i^t = w_i^t + \lambda / d(x_q, x_i)$

end if

end for

end if

end for

$E_t = E_t / n$

if all $f(s_q) = y_q$ (All labels are correct) **then**

$T = t$

break (EXIT loop)

end if

end for

return final hypothesis as an ensemble $f(\{S_1, E_1\}, ..., \{S_T, E_T\})$

Algorithm 5 Boosted k-NN with optimal weights

Inputs:

Training set S with n instances, where ith instance $= (w_i^0, x_i, y_i)$

T specifying the number of iterations

λ weight update term

Initialize:

All weight terms $w_i^0 = 0$, where $i = 1...n$

$S_0 = S$

$\underline{accMax = 0}$

$\underline{optimal = 0}$

for $t = 1$ to T **do**

set $S_t = S_{t-1}$

$\underline{\text{set } acc = 0}$

for s_q in S_t, query s_q where $q = 1...n$ **do**

Find k nearest neighbors using $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$, where $i = 1...n$, $s_q \neq s_i$, and $d(x_q, x_i) = \|x_q - x_i\|_2$

Return the label of $s_q = f(s_q) = \arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise

if $f(s_q) \neq y_q$ **then**

for $i = 1$ to k nearest neighbor **do**

if $y_i \neq y_q$ **then**

update $w_i^t = w_i^t - \lambda / d(x_q, x_i)$

else

update $w_i^t = w_i^t + \lambda / d(x_q, x_i)$

end if

end for

else

$\underline{acc = acc + 1}$

end if

end for

if $\underline{acc > accMax}$ **then**

$\underline{accMax = acc}$

$\underline{optimal = t}$

end if

if all $f(s_q) = y_q$ (All labels are correct) **then**

$T = t$

break (EXIT loop)

end if

end for

return final hypothesis as an ensemble $\underline{f(S_{optimal})}$

Algorithm 6 Boosted k-NN with averaged weights

Inputs:

Training set S with n instances, where ith instance $= (w_i^0, x_i, y_i)$

T specifying the number of iterations

λ weight update term

Initialize:

All weight terms $w_i^0 = 0$, where $i = 1...n$

$S_0 = S$

for $t = 1$ to T **do**

set $S_t = S_{t-1}$

for s_q in S_t, query s_q where $q = 1...n$ **do**

Find k nearest neighbors using $D(s_q, s_i) = \frac{1}{(1+e^{-w_i^t})d(x_q,x_i)}$, where $i = 1...n$, $s_q \neq s_i$, and $d(x_q, x_i) = \|x_q - x_i\|_2$

Return the label of $s_q = f(s_q) = \arg\max_{y \in Y} \sum_{i=1}^{k} D(s_q, s_i)\delta(y, y_i)$

where $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise

if $f(s_q) \neq y_q$ **then**

for $i = 1$ to k nearest neighbor **do**

if $y_i \neq y_q$ **then**

update $w_i^t = w_i^t - \lambda / d(x_q, x_i)$

else

update $w_i^t = w_i^t + \lambda / d(x_q, x_i)$

end if

end for

end if

end for

if all $f(s_q) = y_q$ (All labels are correct) **then**

$T = t$

break (EXIT loop)

end if

end for

for $i = 1$ to n **do**

$weightTotal = 0$

for $t = 1$ to T **do**

$weightTotal = weightTotal + w_i^t$

end for

$w_i^1 = weightTotal/T$

end for

return final hypothesis as an ensemble $f(S_1)$

Chapter 3

Results and Analysis

In this chapter we present the experimental results of Boosted k-NN and its 5 variants. For the basic Boosted k-NN algorithm, we compare the results to 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 [20]. Therefore, only results for the sonar, ionosphere and liver dataset which are obtained from [20] 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 [23] to determine the effectiveness of our algorithm. Table 3.1 lists the datasets with the number of samples and number of attributes in each set. The datasets chosen are of various sizes and difficulty. As we are not interested in mapping discrete values into real values and inventing a distance function for them, 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 the 10-fold cross validation method to evaluate the performance of each algorithm including k-NN, CNN and Boosted Distance. In

Dataset	Number of Samples	Number of Attributes
sonar	208	60
ionosphere	351	34
wine	178	13
liver	345	6
vowel	990	10
segment	2310	19
vehicle	846	18
iris	150	4
glass	214	10
diabetes	768	8

Table 3.1: Datasets used in the experiments

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 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 grand total of 3600 experiments were run and each of them was 10-fold cross validated. Here we highlight various interesting empirical results. The complete results of these experiments are shown in Appendix A.

3.2 Boosted k-NN

Figure 3.1 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 Figure 3.1 are results chosen from the best of all the experiments with various parameters (i.e. pick the best k value). From Figure 3.1 we observe that the results of Boosted k-NN are competitive with other algorithms for all ten datasets. Figure 3.2 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

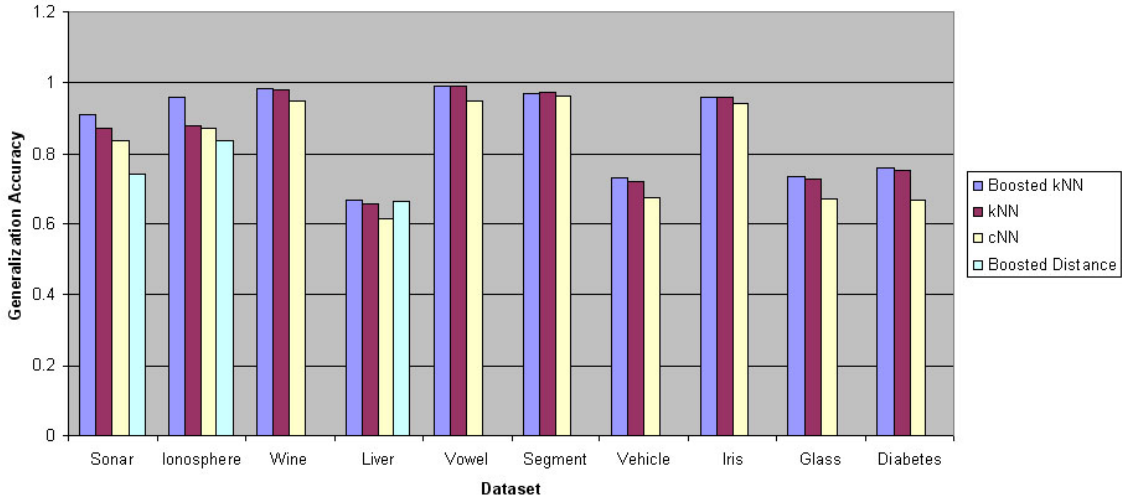

Figure 3.1: Boosted k-NN vs. other algorithms. This graph shows the generalization accuracy of Boosted k-NN, regular k-NN, CNN and (sometimes) Boosted Distance on 10 datasets.

gains (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 90's; therefore, it is more difficult to increase accuracy in these cases.

One feature that we observe about Boosted k-NN is that it is less sensitive to the selection of k. Figure 3.3 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. Figure 3.4 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

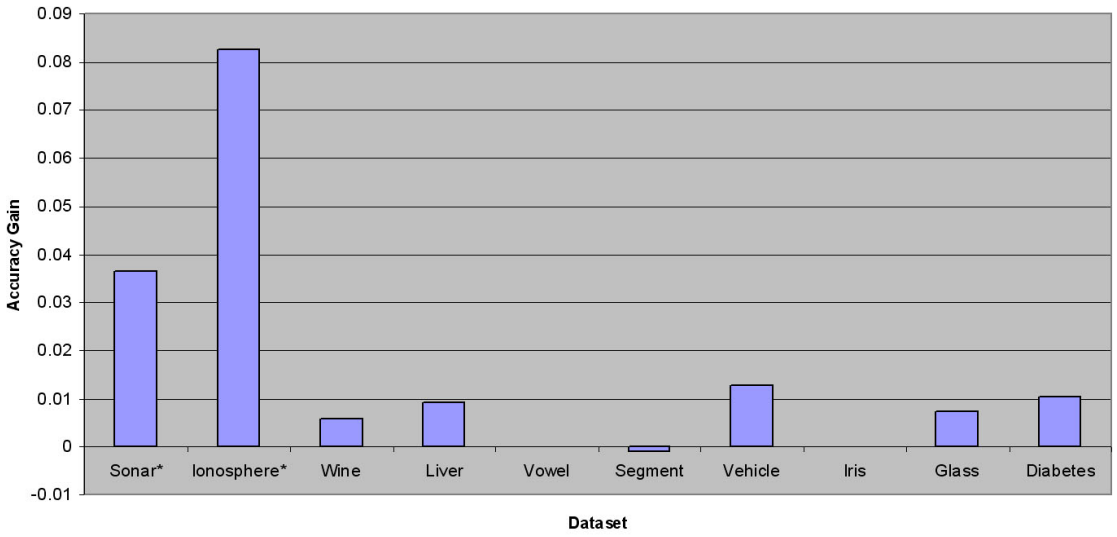

Figure 3.2: Absolute accuracy gain of Boosted k-NN over regular k-NN. Dataset names with an asterisk indicate statistically significant accuracy gains.

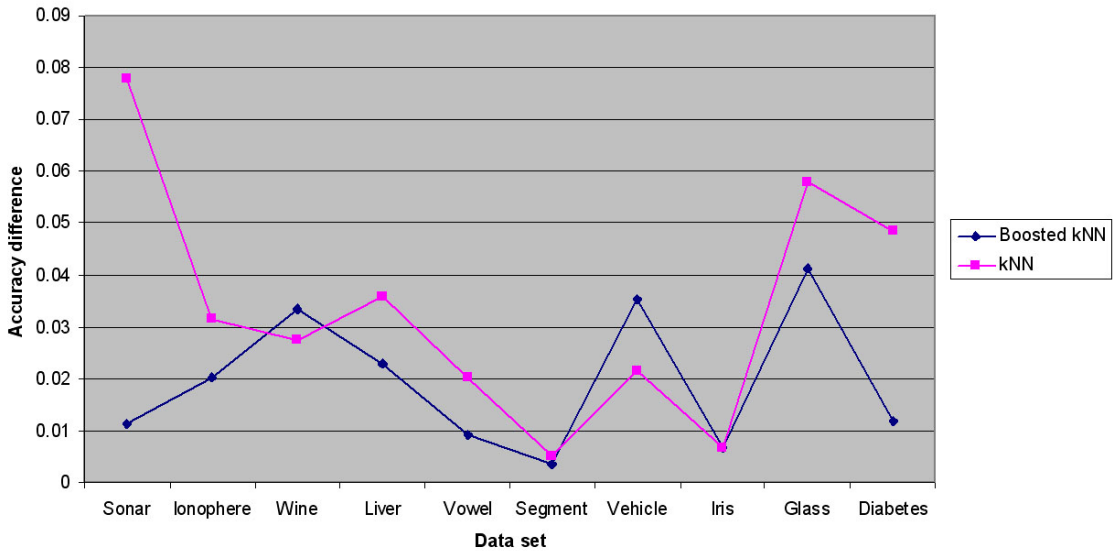

Figure 3.3: Accuracy range of Boosted k-NN and regular k-NN. This graph plots the accuracy difference for 10 datasets with k ranging from 1 to 15.

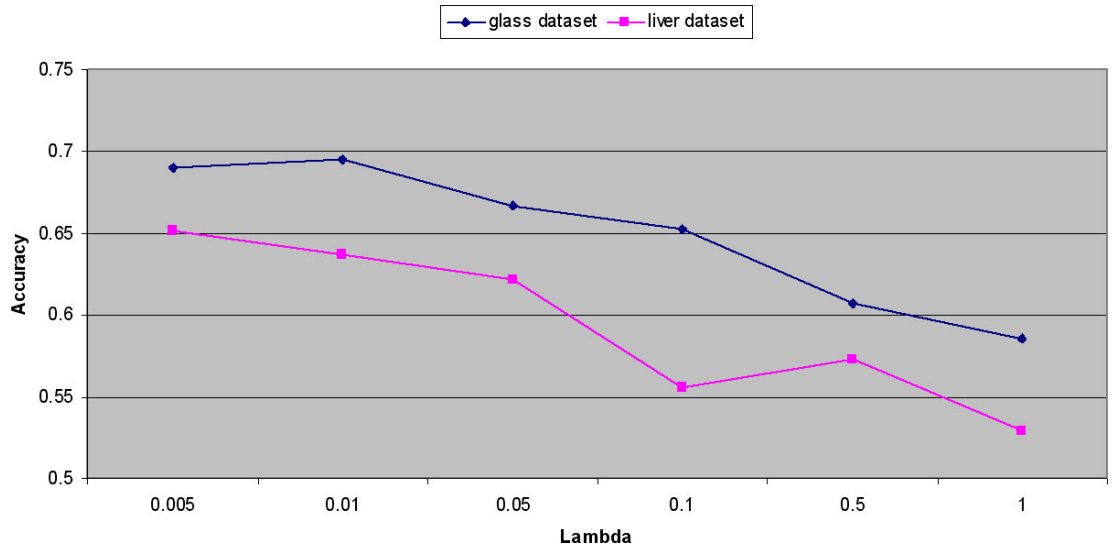

Figure 3.4: Large λ causes accuracy decrease. Choosing larger values for λ causes the accuracy of the liver and glass datasets to decrease.

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. Figure 3.5 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 their neighbors towards their respective desired local minimum; therefore, choosing a λ that is too large with respect to the average distance, will cause the weights to oscillate and not converge

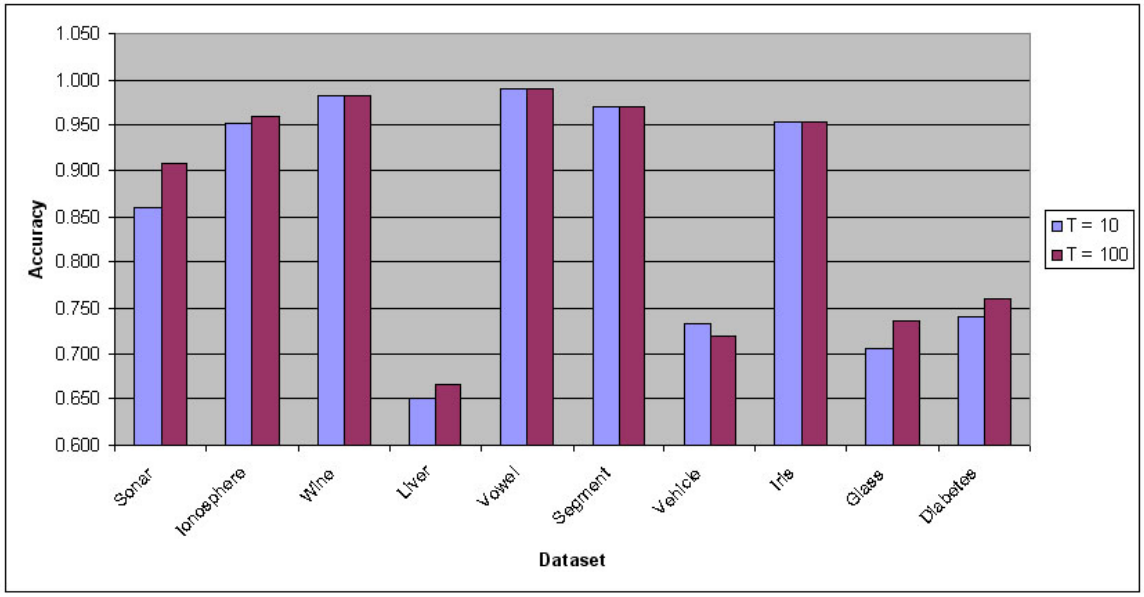

Figure 3.5: Increasing the value of T can increase accuracy if a suitable λ is chosen.

to a good solution, while choosing a λ too small will negatively impact the time to convergence.

3.3 Using hold-out set for selection of λ

One standard way we can choose λ is to use a hold-out set. This is similar to choosing k using a hold-out set in a regular k-NN algorithm. 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. Figure 3.6 shows the hold-out and test set accuracies of the experiment. The graph shows that the hold-out set accuracy follows closely the test set accuracy. This means that the hold out set accuracy is a good indicator for selecting a suitable λ that will likely produce good test set accuracy.

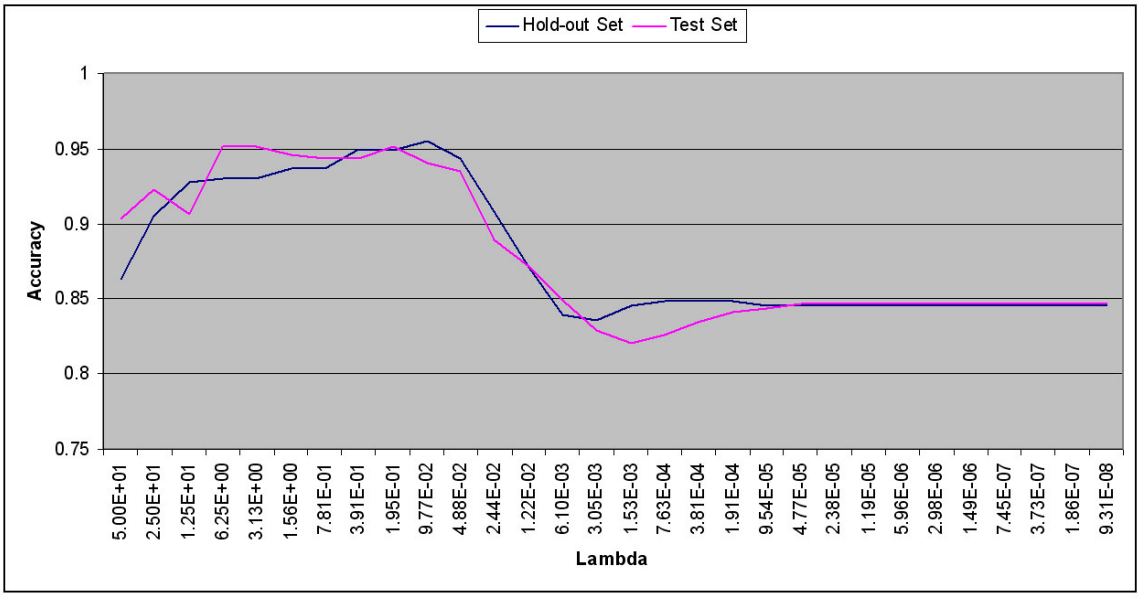

Figure 3.6: Using a hold-out set for selection of λ. Accuracy and the value of λ are compared for the hold-out and test sets on the ionosphere dataset. From the results, we can see that the hold-out set accuracy is a good indicator for picking a suitable λ.

The result of Figure 3.6 corresponds to the results we obtained from the experiment in section 3.2 (see Appendix A). 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.

3.4 Boosted *k*-NN with randomized data order

Figure 3.7 shows the experiment results for Boosted k-NN with randomized data order compared to the basic Boosted k-NN and 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.

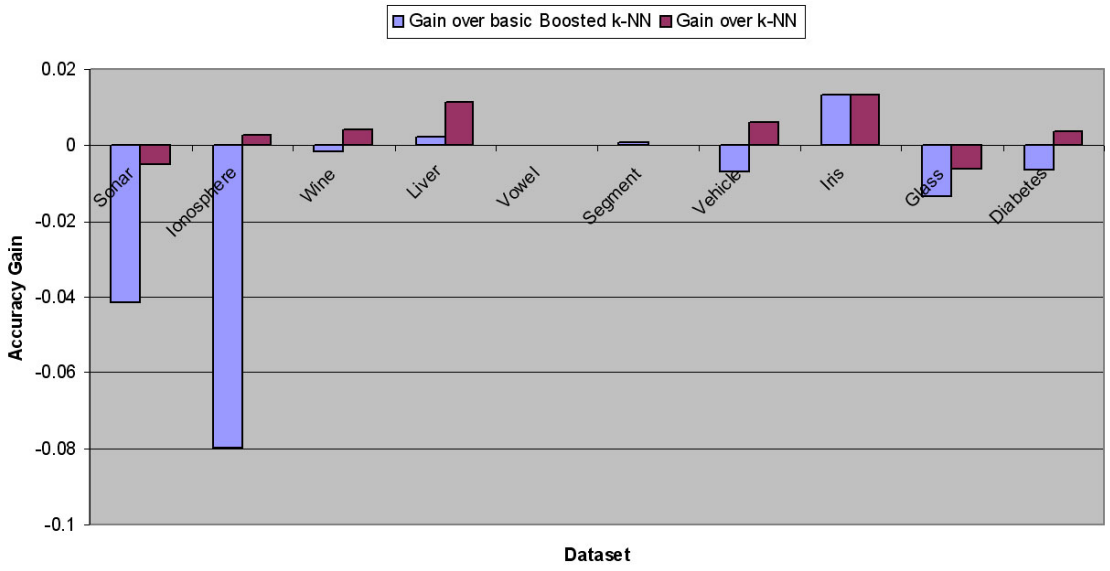

Figure 3.7: Boosted k-NN with randomized data order. The accuracy gain of Boosted k-NN with randomized data order vs. basic Boosted k-NN and regular k-NN. The randomized version does not compare favorably with the basic algorithm. In the sonar and ionosphere datasets, Boosted k-NN with randomized data order shows statistically significant accuracy loss when compared to the basic Boosted k-NN.

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.

3.5 Boosted k-NN with batch update

In Figure 3.8, Boosted k-NN with batch update is being 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 badly in 3 of the datasets, better in 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

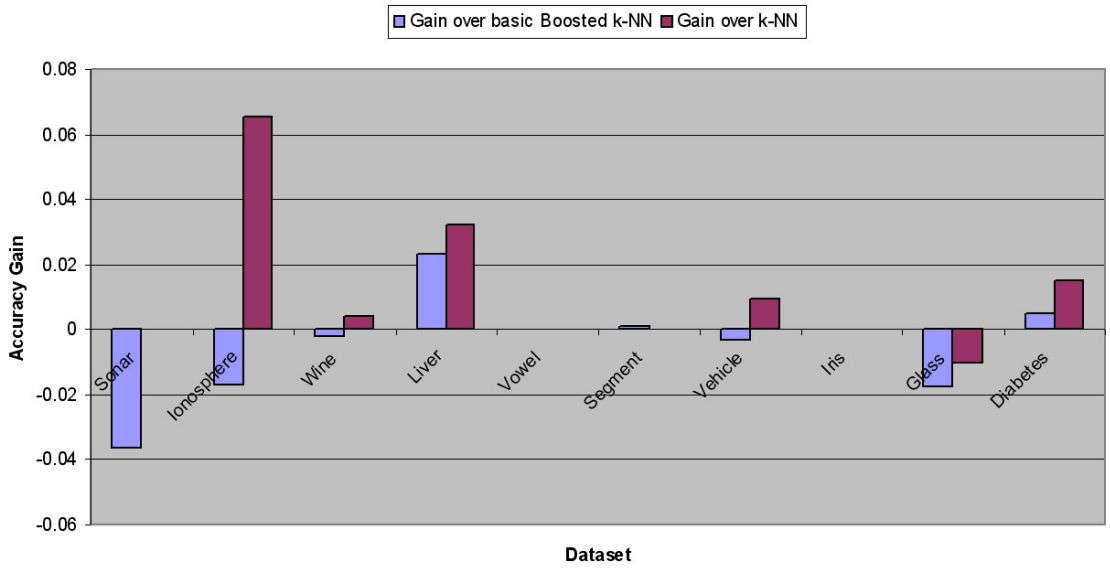

Figure 3.8: Boosted k-NN with batch update. The accuracy gain of Boosted k-NN with batch update vs. basic Boosted k-NN and regular k-NN. Boosted k-NN with batch update shows statistically significant accuracy loss in the sonar dataset when compared to the basic Boosted k-NN. However, Boosted k-NN with batch update still has statistically significant accuracy gain over regular k-NN in the ionosphere dataset.

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. 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 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 modifies 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.

3.6 Boosted k-NN with error-weighted voting

Figure 3.9 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 Figure 3.9; 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.

3.7 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 Figure 3.10. There are no sta-

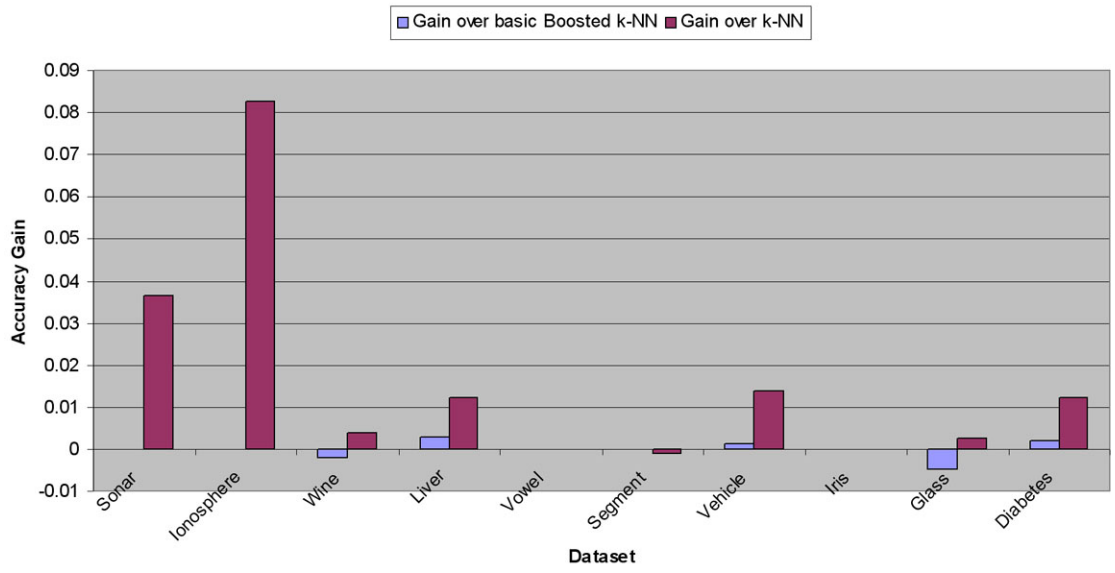

Figure 3.9: Boosted k-NN with error-weighted voting. The accuracy gain of Boosted k-NN with error-weighted voting vs. basic Boosted k-NN and regular k-NN. There are no statistically significant differences between Boosted k-NN with error-weighted voting and the basic Boosted k-NN.

tistically 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 in 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

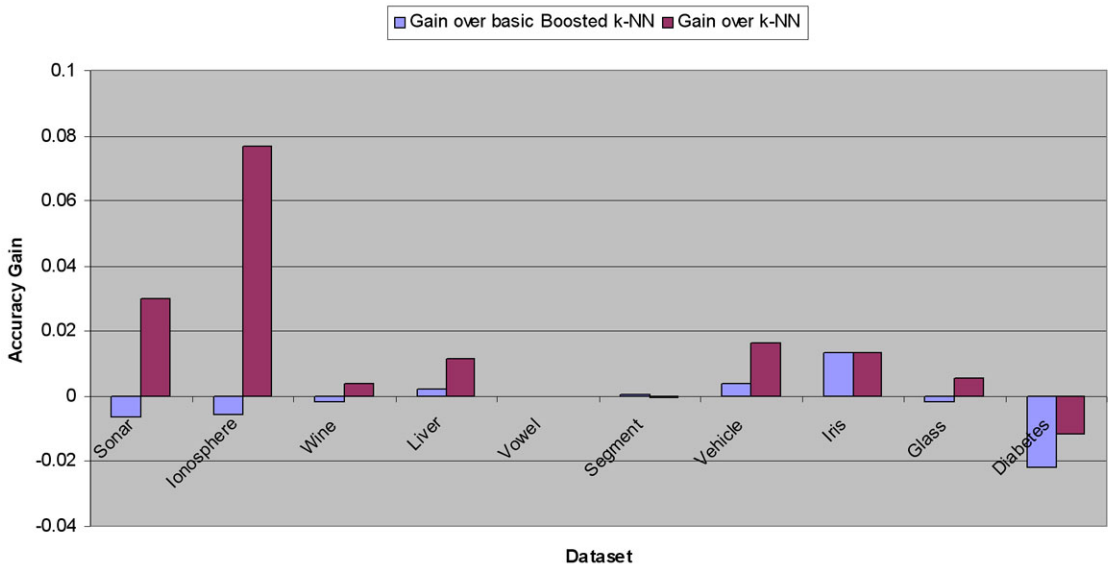

Figure 3.10: Boosted k-NN with optimal weights. The accuracy gain of Boosted k-NN with optimal weights vs. basic Boosted k-NN and regular k-NN. There are no statistically significant differences between Boosted k-NN with optimal weights and the basic Boosted k-NN. This variant reduces storage requirements while maintaining accuracy gains.

optimal weights is useful when boosting is needed but not at the expense of increased storage or query time.

3.8 Boosted k-NN with averaged weights

The last algorithm variant we experimented with is Boosted k-NN with averaged weights. Figure 3.11 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 problem with Boosted k-NN with averaged weights. The algorithm produces bad 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

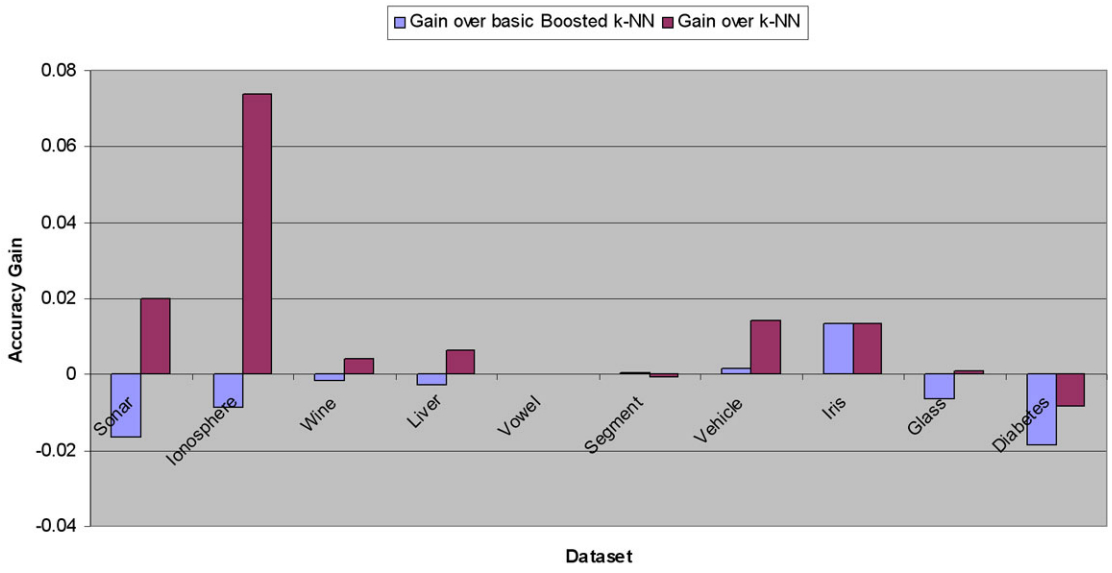

Figure 3.11: Boosted k-NN with average weights. The accuracy gain of Boosted k-NN with average weights vs. basic Boosted k-NN and regular k-NN. There are no statistically significant differences between Boosted k-NN with averaged weights and the basic Boosted k-NN. This variant can be effective for model size reduction but must be used judiciously to avoid undesirable side effects.

groups of models that are converging to different local minima. This also happens if the value of λ is large. By averaging the weights together 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.

Chapter 4

Conclusion

In this thesis we presented 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.

Experimental results show that the Boosted k-NN algorithm is able to increase generalization accuracy in 7 out of 10 datasets. In 2 of the datasets Boosted k-NN posts a statistically significant increase in generalization accuracy over the regular k-NN.

We also investigated 5 variants to the Boosted k-NN algorithm. Table 4.1 shows the high level comparison of performance between Boosted k-NN and its 5 variants. The paired values in column 2 and 3 represent the number of dataset(s) (out of 10 datasets) that the algorithm 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 in 3 datasets and worse in 1 dataset. Out of the 5 variants, Boosted k-NN with optimal

Algorithm	vs. k-NN	vs. bBk-NN	Space reduction
Boosted k-NN	(+5, -0)	NA	No
Boosted k-NN with randomized data order	(+2, -0)	(+1, -2)	No
Boosted k-NN with batch update	(+3, -1)	(+1, -3)	No
Boosted k-NN with error-weighted voting	(+5, -0)	(+0, -0)	No
Boosted k-NN with optimal weights	(+5, -1)	(+1, -1)	Yes
Boosted k-NN with average weights	(+4, -0)	(+1, -2)	Yes

Table 4.1: Comparing Boosted k-NN to its variants. The paired values in column 2 and 3 represent the number of dataset(s) (out of 10 datasets) that 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.)

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.

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.

Bibliography

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[2] P. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, pp. 515–516, 1968.

[3] W. Gates, "The reduced nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 18, pp. 431–433, 1972.

[4] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour, "An algorithm for a selective nearest neighbor decision rule," *IEEE Transactions on Information Theory*, vol. 21, pp. 665–669, 1975.

[5] I. Tomek, "Two modifications of cnn," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, pp. 769–772, 1976.

[6] K. C. Gowda and G. Krishna, "The condensed nearest neighbor rule using the concept of mutual nearest neighborhood," *IEEE Transactions on Information Theory*, vol. 25, pp. 488–490, 1979.

[7] F. Devi and M. Murty, "An incremental prototype set building technique," *Pattern Recognition*, vol. 35, pp. 505–513, 2002.

[8] T. Raicharoen and C. Lursinsap, "A divide-and-conquer approach to pairwise opposite class (poc) nearest neighbor algorithm," *Pattern Recognition Letters*, vol. 26, pp. 1554–1567, 2005.

[9] F. Angiulli, "Fast condensed nearest neighbor rule," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM Press, 2005, pp. 25–32.

[10] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*, 1995, pp. 23–37. [Online]. Available: citeseer.ist.psu.edu/freund95decisiontheoretic.html

[11] ——, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning*, 1996, pp. 148–156. [Online]. Available: citeseer.ist.psu.edu/freund96experiments.html

[12] M. Valverde and F. Ferri, "Experiments with adaboost and linear programming," in *III Taller de Minera de Datos y Aprendizaje (TAMIDA 2005)*, 2005, pp. 153–158.

[13] E. Alpaydin, "Voting over multiple condensed nearest neighbors," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 115–132, 1997. [Online]. Available: citeseer.ist.psu.edu/article/alpaydin97voting.html

[14] S. D. Bay, "Combining nearest neighbor classifiers through multiple feature subsets," in *Proc. 15th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1998, pp. 37–45. [Online]. Available: citeseer.ist.psu.edu/bay98combining.html

[15] O. Okun and H. Priisalu, "Multiple views in ensembles of nearest neighbor classifiers," in *Proceedings of Workshop on Learning with Multiple Views (in conjunction with the Twenty-Second International Conference on Machine Learning)*, Bonn, Germany, 2005, pp. 51–58.

[16] C. Domeniconi and B. Yan, "Nearest neighbor ensemble," in *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 228–231.

[17] Z. H. Zhou and Y. Yu, "Adapt bagging to nearest neighbor classifiers," *Journal of Computer Science and Technology*, vol. 20, pp. 48–54, 2005.

[18] ——, "Ensembling local learners through multimodal perturbation," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, pp. 725–735, 2005.

[19] V. Athitsos and S. Sclaroff, "Boosting nearest neighbor classifiers for multiclass recognition," in *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*. Washington, DC, USA: IEEE Computer Society, 2005, p. 45.

[20] J. Amores, N. Sebe, and P. Radeva, "Boosting the distance estimation," *Pattern Recognition Letters*, vol. 27, no. 3, pp. 201–209, 2006.

[21] J. Utans, "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*. Portland, OR: AAAI Press, 1996, pp. 133–138. [Online]. Available: citeseer.ist.psu.edu/utans96weight.html

[22] T. Andersen and T. Martinez, "The little neuron that could," in *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, vol. 3, Washington, DC, USA, 1999, pp. 1608–1613.

[23] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

Appendix A

Experiment Results

This section presents the experiment results that have we gathered for Boosted k-NN and the variants. There are a total of 60 tables in this appendix which comes from 6 algorithms, each with 10 datasets worth of experiments. Each table contains the regular k-NN results and 60 10-fold cross validated results of our algorithm using various values for T, λ and k. The p-value is the pair permutation test result of our algorithm's accuracy vs. regular k-NN.

A.1 Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.871		0.846		0.854		0.826		0.793	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.866	1.000	0.856	1.000	0.854	1.000	0.830	1.000	0.810	0.250
10	0.01	0.861	0.500	0.856	1.000	0.844	0.750	0.835	0.688	0.819	0.094
10	0.05	0.867	1.000	0.871	0.250	0.825	0.188	0.850	0.125	0.832	0.180
10	0.1	0.877	0.688	0.876	0.188	0.840	0.633	0.859	0.094	0.844	0.063
10	0.5	0.896	0.313	0.897	0.063	0.865	0.625	0.877	0.031	0.861	0.023
10	1	0.881	0.813	0.862	0.609	0.874	0.500	0.887	0.016	0.869	0.023
100	0.005	0.857	0.375	0.876	0.125	0.840	0.613	0.845	0.219	0.847	0.078
100	0.01	0.872	1.000	0.871	0.188	0.849	0.969	0.859	0.094	0.844	0.070
100	0.05	0.881	0.750	0.882	0.156	0.874	0.422	0.854	0.094	0.867	0.016
100	0.1	0.881	0.750	0.879	0.281	0.867	0.688	0.844	0.375	0.872	0.016
100	0.5	0.896	0.313	0.894	0.055	0.892	0.172	0.877	0.016	0.907	0.008
100	1	0.886	0.688	0.887	0.156	0.897	0.078	0.897	0.016	0.872	0.008

Table A.1: Sonar: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.878		0.863		0.858		0.852		0.846	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.889	0.125	0.869	0.625	0.841	0.063	0.852	1.000	0.844	1.000
10	0.01	0.889	0.219	0.872	0.531	0.858	1.000	0.858	0.656	0.861	0.422
10	0.05	0.898	0.063	0.900	0.023	0.909	0.016	0.920	0.004	0.940	0.002
10	0.1	0.906	0.094	0.912	0.008	0.920	0.016	0.946	0.002	0.943	0.002
10	0.5	0.917	0.063	0.937	0.002	0.940	0.002	0.949	0.002	0.935	0.002
10	1	0.920	0.023	0.935	0.002	0.929	0.004	0.943	0.002	0.952	0.002
100	0.005	0.892	0.234	0.906	0.016	0.909	0.008	0.917	0.006	0.932	0.002
100	0.01	0.906	0.074	0.920	0.008	0.929	0.004	0.932	0.002	0.943	0.002
100	0.05	0.917	0.051	0.937	0.002	0.940	0.002	0.943	0.002	0.946	0.002
100	0.1	0.912	0.090	0.934	0.002	0.943	0.002	0.935	0.002	0.949	0.002
100	0.5	0.935	0.023	0.940	0.002	0.937	0.002	0.937	0.002	0.943	0.002
100	1	0.940	0.020	0.949	0.002	0.946	0.002	0.955	0.002	0.960	0.002

Table A.2: Ionosphere: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.951		0.961		0.955		0.963		0.978	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.951	1.000	0.961	1.000	0.961	1.000	0.968	1.000	0.978	1.000
10	0.01	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.984	1.000
10	0.05	0.951	1.000	0.965	1.000	0.967	0.500	0.978	0.250	0.978	1.000
10	0.1	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
10	0.5	0.951	1.000	0.976	0.250	0.978	0.250	0.976	0.500	0.982	1.000
10	1	0.951	1.000	0.971	0.500	0.982	0.125	0.971	0.750	0.965	0.500
100	0.005	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.978	1.000
100	0.01	0.951	1.000	0.971	0.500	0.967	0.500	0.976	0.500	0.982	1.000
100	0.05	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
100	0.1	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
100	0.5	0.951	1.000	0.976	0.250	0.978	0.250	0.976	0.500	0.982	1.000
100	1	0.951	1.000	0.971	0.500	0.982	0.125	0.971	0.750	0.965	0.500

Table A.3: Wine: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.638		0.644		0.621		0.630		0.657	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.635	1.000	0.655	0.691	0.658	0.172	0.666	0.297	0.651	0.875
10	0.01	0.638	1.000	0.638	0.836	0.640	0.539	0.637	0.828	0.637	0.391
10	0.05	0.624	0.695	0.637	0.777	0.607	0.676	0.624	0.898	0.622	0.125
10	0.1	0.597	0.195	0.598	0.219	0.590	0.305	0.622	0.867	0.555	0.016
10	0.5	0.619	0.336	0.594	0.066	0.580	0.117	0.583	0.318	0.573	0.004
10	1	0.626	0.793	0.600	0.188	0.591	0.281	0.547	0.027	0.529	0.004
100	0.005	0.644	0.875	0.619	0.402	0.659	0.219	0.652	0.529	0.666	0.656
100	0.01	0.624	0.688	0.635	0.836	0.654	0.340	0.646	0.656	0.649	0.813
100	0.05	0.609	0.350	0.610	0.273	0.629	0.793	0.598	0.432	0.571	0.039
100	0.1	0.588	0.098	0.603	0.219	0.635	0.688	0.623	0.871	0.591	0.035
100	0.5	0.600	0.227	0.585	0.172	0.591	0.242	0.614	0.621	0.591	0.111
100	1	0.606	0.305	0.594	0.156	0.597	0.313	0.588	0.223	0.594	0.039

Table A.4: Liver: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.990		0.984		0.982		0.978		0.970	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.990	1.000	0.984	0.750	0.985	0.500	0.982	0.359	0.979	0.109
10	0.01	0.990	1.000	0.987	0.500	0.986	0.250	0.983	0.188	0.979	0.148
10	0.05	0.990	1.000	0.982	0.781	0.988	0.094	0.986	0.070	0.977	0.406
10	0.1	0.987	1.000	0.980	0.188	0.982	0.844	0.980	0.797	0.973	0.734
10	0.5	0.987	1.000	0.976	0.094	0.960	0.008	0.929	0.002	0.888	0.002
10	1	0.987	1.000	0.978	0.391	0.957	0.023	0.918	0.004	0.786	0.002
100	0.005	0.990	1.000	0.984	0.750	0.986	0.250	0.984	0.109	0.980	0.086
100	0.01	0.990	1.000	0.987	0.500	0.987	0.250	0.982	0.359	0.981	0.102
100	0.05	0.989	1.000	0.982	0.781	0.988	0.094	0.984	0.230	0.977	0.422
100	0.1	0.988	1.000	0.981	0.500	0.984	0.750	0.982	0.453	0.975	0.547
100	0.5	0.976	0.031	0.931	0.004	0.868	0.002	0.842	0.002	0.862	0.002
100	1	0.971	0.063	0.711	0.002	0.601	0.002	0.556	0.002	0.553	0.002

Table A.5: Vowel: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.971		0.969		0.967		0.968		0.966	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.971	1.000	0.970	0.688	0.969	0.188	0.969	0.750	0.967	0.883
10	0.01	0.968	0.383	0.966	0.141	0.966	0.813	0.965	0.332	0.962	0.340
10	0.05	0.965	0.008	0.968	0.941	0.961	0.125	0.965	0.141	0.964	0.391
10	0.1	0.965	0.070	0.966	0.500	0.961	0.102	0.959	0.018	0.957	0.086
10	0.5	0.960	0.002	0.960	0.025	0.956	0.039	0.951	0.002	0.951	0.012
10	1	0.960	0.002	0.961	0.078	0.960	0.094	0.953	0.004	0.949	0.020
100	0.001	0.970	0.336	0.967	0.406	0.968	0.789	0.968	1.000	0.965	0.602
100	0.01	0.970	0.719	0.966	0.203	0.966	0.875	0.963	0.109	0.963	0.531
100	0.05	0.963	0.055	0.963	0.063	0.964	0.453	0.961	0.105	0.961	0.191
100	0.1	0.960	0.031	0.966	0.500	0.961	0.117	0.957	0.012	0.961	0.289
100	0.5	0.960	0.002	0.960	0.025	0.956	0.039	0.951	0.002	0.951	0.012
100	1	0.960	0.002	0.961	0.078	0.960	0.094	0.953	0.004	0.949	0.020

Table A.6: Segment: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.699		0.707		0.716		0.720		0.703	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.693	0.672	0.706	0.969	0.728	0.336	0.732	0.469	0.733	0.086
10	0.01	0.689	0.406	0.705	0.930	0.721	0.566	0.725	0.844	0.729	0.080
10	0.05	0.689	0.441	0.709	0.914	0.727	0.227	0.709	0.344	0.720	0.211
10	0.1	0.685	0.270	0.708	0.953	0.714	0.938	0.716	0.715	0.719	0.445
10	0.5	0.637	0.014	0.636	0.020	0.626	0.004	0.620	0.002	0.641	0.025
10	1	0.611	0.006	0.533	0.002	0.532	0.002	0.540	0.002	0.540	0.002
100	0.005	0.693	0.500	0.718	0.539	0.716	1.000	0.718	0.875	0.720	0.223
100	0.01	0.698	0.904	0.717	0.492	0.713	0.750	0.715	0.699	0.724	0.215
100	0.05	0.676	0.117	0.706	0.879	0.703	0.219	0.713	0.625	0.717	0.434
100	0.1	0.655	0.006	0.696	0.477	0.702	0.313	0.691	0.078	0.704	1.000
100	0.5	0.630	0.012	0.497	0.002	0.438	0.002	0.505	0.002	0.545	0.002
100	1	0.597	0.004	0.362	0.002	0.483	0.002	0.488	0.002	0.527	0.002

Table A.7: Vehicle: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.953		0.953		0.953		0.960		0.953	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.947	1.000	0.953	1.000	0.947	1.000	0.947	0.500	0.947	1.000
10	0.01	0.947	1.000	0.960	1.000	0.947	1.000	0.953	1.000	0.953	1.000
10	0.05	0.953	1.000	0.960	1.000	0.947	1.000	0.947	0.625	0.947	1.000
10	0.1	0.953	1.000	0.953	1.000	0.947	1.000	0.953	1.000	0.947	1.000
10	0.5	0.953	1.000	0.960	1.000	0.953	1.000	0.960	1.000	0.960	1.000
10	1	0.953	1.000	0.960	1.000	0.953	1.000	0.947	0.625	0.947	1.000
100	0.005	0.947	1.000	0.953	1.000	0.953	1.000	0.953	1.000	0.953	1.000
100	0.01	0.947	1.000	0.947	1.000	0.947	1.000	0.953	1.000	0.953	1.000
100	0.05	0.953	1.000	0.947	1.000	0.947	1.000	0.947	0.625	0.953	1.000
100	0.1	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.625	0.947	1.000
100	0.5	0.947	1.000	0.947	1.000	0.940	0.750	0.947	0.500	0.940	0.750
100	1	0.940	1.000	0.933	0.250	0.913	0.250	0.907	0.063	0.920	0.188

Table A.8: Iris: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.697		0.715		0.729		0.705		0.671	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.697	1.000	0.706	0.688	0.709	0.273	0.691	0.359	0.690	0.555
10	0.01	0.730	0.289	0.706	0.750	0.709	0.336	0.691	0.430	0.695	0.375
10	0.05	0.667	0.270	0.673	0.328	0.672	0.023	0.671	0.215	0.667	0.906
10	0.1	0.666	0.422	0.665	0.090	0.644	0.016	0.681	0.188	0.652	0.266
10	0.5	0.573	0.031	0.581	0.016	0.578	0.004	0.630	0.031	0.607	0.063
10	1	0.492	0.004	0.431	0.002	0.418	0.002	0.437	0.002	0.586	0.047
100	0.005	0.686	0.734	0.709	0.844	0.690	0.154	0.695	0.719	0.695	0.438
100	0.01	0.667	0.313	0.736	0.406	0.699	0.258	0.695	0.688	0.694	0.375
100	0.05	0.601	0.078	0.694	0.461	0.681	0.055	0.670	0.098	0.651	0.430
100	0.1	0.559	0.031	0.648	0.098	0.652	0.031	0.671	0.289	0.638	0.313
100	0.5	0.403	0.002	0.507	0.014	0.571	0.004	0.493	0.004	0.624	0.461
100	1	0.396	0.002	0.333	0.002	0.359	0.002	0.346	0.002	0.575	0.047

Table A.9: Glass: Boosted k-NN

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.703		0.730		0.734		0.736		0.751	
Boosted k-NN											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.710	0.441	0.720	0.281	0.732	0.930	0.727	0.529	0.729	0.059
10	0.01	0.720	0.162	0.725	0.645	0.737	0.766	0.740	0.578	0.732	0.125
10	0.05	0.724	0.188	0.738	0.686	0.741	0.594	0.742	0.559	0.730	0.090
10	0.1	0.738	0.031	0.746	0.178	0.736	0.910	0.748	0.203	0.719	0.016
10	0.5	0.750	0.016	0.754	0.137	0.749	0.434	0.751	0.422	0.740	0.531
10	1	0.750	0.039	0.703	0.389	0.734	0.961	0.742	0.676	0.734	0.297
100	0.005	0.712	0.539	0.709	0.102	0.711	0.023	0.715	0.125	0.733	0.109
100	0.01	0.719	0.371	0.728	0.926	0.716	0.250	0.715	0.156	0.735	0.250
100	0.05	0.744	0.016	0.735	0.766	0.727	0.652	0.739	0.875	0.732	0.203
100	0.1	0.755	0.004	0.745	0.172	0.748	0.406	0.744	0.510	0.746	0.813
100	0.5	0.750	0.029	0.717	0.508	0.732	0.908	0.757	0.277	0.760	0.617
100	1	0.752	0.057	0.698	0.148	0.736	0.930	0.761	0.152	0.748	0.859

Table A.10: Diabetes: Boosted k-NN

A.2 Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.871		0.846		0.854		0.826		0.793	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.866	1.000	0.846	1.000	0.852	1.000	0.816	0.500	0.788	1.000
10	0.01	0.866	1.000	0.846	1.000	0.844	0.500	0.816	0.500	0.791	0.875
10	0.05	0.861	0.500	0.851	1.000	0.842	0.250	0.820	0.563	0.786	0.875
10	0.1	0.856	0.375	0.859	0.250	0.849	0.875	0.835	0.625	0.788	0.875
10	0.5	0.846	0.125	0.827	0.289	0.861	0.625	0.836	0.766	0.809	0.453
10	1	0.857	0.609	0.862	0.500	0.842	0.375	0.824	0.813	0.803	0.727
100	0.005	0.861	0.500	0.846	1.000	0.854	1.000	0.830	1.000	0.784	0.625
100	0.01	0.861	0.500	0.851	1.000	0.852	1.000	0.834	0.750	0.798	0.875
100	0.05	0.851	0.313	0.846	1.000	0.857	0.875	0.834	0.750	0.813	0.250
100	0.1	0.865	0.844	0.856	1.000	0.852	0.969	0.830	1.000	0.804	0.375
100	0.5	0.859	0.500	0.852	0.750	0.834	0.500	0.831	1.000	0.803	0.633
100	1	0.866	1.000	0.844	0.980	0.814	0.219	0.819	0.797	0.809	0.461

Table A.11: Sonar: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.878		0.863		0.858		0.852		0.846	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.878	1.000	0.863	1.000	0.855	1.000	0.855	1.000	0.835	0.250
10	0.01	0.880	1.000	0.855	0.250	0.855	0.500	0.849	1.000	0.835	0.250
10	0.05	0.875	1.000	0.849	0.500	0.838	0.063	0.846	0.625	0.812	0.016
10	0.1	0.863	0.313	0.846	0.305	0.838	0.234	0.807	0.004	0.795	0.055
10	0.5	0.860	0.156	0.744	0.004	0.764	0.004	0.750	0.008	0.704	0.004
10	1	0.861	0.281	0.736	0.002	0.735	0.002	0.715	0.004	0.673	0.002
100	0.005	0.872	0.750	0.852	0.313	0.846	0.250	0.846	0.750	0.829	0.250
100	0.01	0.869	0.531	0.846	0.188	0.832	0.031	0.838	0.250	0.815	0.063
100	0.05	0.858	0.094	0.770	0.008	0.778	0.002	0.733	0.002	0.724	0.004
100	0.1	0.823	0.016	0.724	0.002	0.690	0.002	0.687	0.002	0.678	0.002
100	0.5	0.798	0.008	0.681	0.004	0.681	0.002	0.676	0.002	0.653	0.002
100	1	0.772	0.002	0.692	0.002	0.670	0.002	0.673	0.002	0.653	0.002

Table A.12: Ionosphere: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.951		0.961		0.955		0.963		0.978	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.951	1.000	0.961	1.000	0.955	1.000	0.968	1.000	0.978	1.000
10	0.01	0.951	1.000	0.961	1.000	0.961	1.000	0.963	1.000	0.978	1.000
10	0.05	0.951	1.000	0.963	1.000	0.949	1.000	0.963	1.000	0.972	1.000
10	0.1	0.951	1.000	0.955	1.000	0.961	1.000	0.982	0.250	0.978	1.000
10	0.5	0.951	1.000	0.957	1.000	0.965	0.500	0.941	0.219	0.976	1.000
10	1	0.945	1.000	0.961	1.000	0.949	1.000	0.957	1.000	0.957	0.250
100	0.005	0.957	1.000	0.957	1.000	0.955	1.000	0.974	0.500	0.972	1.000
100	0.01	0.951	1.000	0.961	1.000	0.955	1.000	0.968	1.000	0.972	1.000
100	0.05	0.957	1.000	0.959	1.000	0.965	0.500	0.976	0.500	0.978	1.000
100	0.1	0.961	0.500	0.972	0.500	0.972	0.500	0.965	1.000	0.972	1.000
100	0.5	0.965	0.500	0.971	0.500	0.967	0.625	0.982	0.250	0.965	0.500
100	1	0.959	1.000	0.951	0.500	0.967	0.500	0.961	1.000	0.906	0.016

Table A.13: Wine: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.638		0.644		0.621		0.630		0.657	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.627	0.719	0.633	0.734	0.618	0.906	0.651	0.250	0.634	0.234
10	0.01	0.607	0.230	0.618	0.320	0.635	0.672	0.653	0.242	0.602	0.117
10	0.05	0.641	0.875	0.594	0.055	0.610	0.730	0.580	0.270	0.614	0.039
10	0.1	0.606	0.141	0.594	0.109	0.617	0.910	0.595	0.260	0.598	0.031
10	0.5	0.631	0.797	0.593	0.180	0.563	0.152	0.567	0.117	0.609	0.098
10	1	0.627	0.746	0.565	0.031	0.549	0.076	0.615	0.680	0.615	0.141
100	0.005	0.603	0.102	0.659	0.602	0.661	0.055	0.649	0.430	0.661	0.852
100	0.01	0.635	1.000	0.638	0.877	0.646	0.459	0.629	1.000	0.655	1.000
100	0.05	0.662	0.313	0.606	0.246	0.610	0.734	0.588	0.223	0.611	0.117
100	0.1	0.668	0.188	0.618	0.391	0.600	0.418	0.565	0.051	0.611	0.070
100	0.5	0.635	1.000	0.571	0.070	0.592	0.414	0.538	0.035	0.612	0.107
100	1	0.594	0.162	0.594	0.188	0.600	0.340	0.566	0.098	0.612	0.102

Table A.14: Liver: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.990		0.984		0.982		0.978		0.970	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.989	1.000	0.982	0.500	0.981	1.000	0.978	1.000	0.973	0.500
10	0.01	0.990	1.000	0.984	1.000	0.982	1.000	0.974	0.500	0.971	0.813
10	0.05	0.989	1.000	0.975	0.031	0.974	0.063	0.970	0.063	0.971	0.961
10	0.1	0.986	0.125	0.974	0.078	0.964	0.008	0.970	0.250	0.941	0.016
10	0.5	0.985	0.125	0.964	0.004	0.946	0.002	0.898	0.002	0.820	0.002
10	1	0.987	1.000	0.969	0.016	0.942	0.002	0.864	0.002	0.697	0.002
100	0.005	0.980	0.031	0.983	0.938	0.978	0.438	0.972	0.391	0.966	0.500
100	0.01	0.984	0.063	0.982	0.875	0.972	0.047	0.972	0.609	0.967	0.516
100	0.05	0.986	0.500	0.936	0.002	0.958	0.023	0.954	0.004	0.944	0.004
100	0.1	0.981	0.031	0.938	0.002	0.937	0.002	0.930	0.004	0.925	0.004
100	0.5	0.982	0.063	0.828	0.002	0.748	0.002	0.727	0.002	0.769	0.002
100	1	0.978	0.008	0.838	0.002	0.668	0.002	0.620	0.002	0.595	0.002

Table A.15: Vowel: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.971		0.969		0.967		0.968		0.966	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.971	1.000	0.968	0.844	0.968	0.422	0.967	0.250	0.966	1.000
10	0.01	0.969	0.125	0.968	0.500	0.966	0.922	0.966	0.094	0.965	0.836
10	0.05	0.969	0.203	0.967	0.391	0.963	0.125	0.962	0.086	0.961	0.119
10	0.1	0.968	0.047	0.965	0.066	0.961	0.074	0.963	0.016	0.952	0.008
10	0.5	0.967	0.094	0.966	0.109	0.962	0.141	0.957	0.006	0.942	0.002
10	1	0.968	0.063	0.961	0.012	0.959	0.031	0.955	0.004	0.932	0.002
100	0.005	0.968	0.430	0.964	0.059	0.968	0.859	0.963	0.020	0.959	0.059
100	0.01	0.968	0.250	0.959	0.016	0.962	0.172	0.959	0.055	0.952	0.008
100	0.05	0.968	0.385	0.950	0.006	0.946	0.002	0.948	0.002	0.941	0.008
100	0.1	0.968	0.313	0.939	0.002	0.940	0.002	0.937	0.002	0.930	0.002
100	0.5	0.958	0.008	0.907	0.002	0.898	0.002	0.900	0.002	0.906	0.002
100	1	0.962	0.016	0.858	0.002	0.868	0.002	0.887	0.002	0.881	0.002

Table A.16: Segment: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.699		0.707		0.716		0.720		0.703	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.700	0.906	0.703	0.656	0.718	0.875	0.726	0.625	0.713	0.133
10	0.01	0.693	0.527	0.703	0.703	0.712	0.695	0.722	0.891	0.700	0.766
10	0.05	0.693	0.590	0.690	0.094	0.694	0.047	0.697	0.170	0.680	0.125
10	0.1	0.708	0.590	0.690	0.264	0.694	0.100	0.663	0.008	0.665	0.004
10	0.5	0.678	0.203	0.583	0.004	0.618	0.004	0.567	0.002	0.583	0.002
10	1	0.664	0.074	0.473	0.002	0.500	0.002	0.522	0.002	0.515	0.002
100	0.005	0.700	0.879	0.712	0.602	0.719	0.691	0.714	0.516	0.709	0.656
100	0.01	0.705	0.461	0.709	0.750	0.721	0.547	0.711	0.496	0.704	1.000
100	0.05	0.719	0.184	0.695	0.414	0.710	0.516	0.707	0.359	0.686	0.180
100	0.1	0.722	0.090	0.693	0.523	0.665	0.002	0.637	0.002	0.640	0.004
100	0.5	0.699	1.000	0.515	0.002	0.514	0.002	0.537	0.002	0.552	0.002
100	1	0.712	0.434	0.465	0.002	0.495	0.002	0.489	0.002	0.565	0.002

Table A.17: Vehicle: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.953		0.953		0.953		0.960		0.953	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.960	1.000	0.953	1.000	0.947	1.000	0.947	0.500	0.947	1.000
10	0.01	0.953	1.000	0.960	1.000	0.947	1.000	0.960	1.000	0.953	1.000
10	0.05	0.953	1.000	0.953	1.000	0.960	1.000	0.960	1.000	0.953	1.000
10	0.1	0.933	0.250	0.960	1.000	0.953	1.000	0.960	1.000	0.940	0.500
10	0.5	0.933	0.250	0.940	0.500	0.940	0.500	0.920	0.125	0.953	1.000
10	1	0.953	1.000	0.960	1.000	0.960	1.000	0.953	1.000	0.940	0.750
100	0.005	0.953	1.000	0.967	0.500	0.960	1.000	0.947	0.500	0.953	1.000
100	0.01	0.947	1.000	0.920	0.125	0.960	1.000	0.947	0.500	0.953	1.000
100	0.05	0.960	1.000	0.947	1.000	0.933	0.250	0.947	0.500	0.953	1.000
100	0.1	0.953	1.000	0.927	0.500	0.973	0.500	0.953	1.000	0.960	1.000
100	0.5	0.953	1.000	0.933	0.875	0.893	0.500	0.873	0.023	0.927	0.500
100	1	0.947	1.000	0.940	0.688	0.920	0.375	0.913	0.375	0.940	0.750

Table A.18: Iris: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.697		0.715		0.729		0.705		0.671	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.670	0.414	0.693	0.313	0.723	0.836	0.677	0.266	0.677	0.750
10	0.01	0.678	0.500	0.649	0.043	0.658	0.051	0.639	0.023	0.653	0.313
10	0.05	0.650	0.031	0.620	0.031	0.608	0.002	0.605	0.027	0.567	0.004
10	0.1	0.661	0.188	0.563	0.004	0.602	0.008	0.566	0.012	0.569	0.008
10	0.5	0.614	0.016	0.473	0.004	0.538	0.008	0.535	0.004	0.535	0.029
10	1	0.633	0.172	0.492	0.008	0.435	0.004	0.468	0.008	0.497	0.002
100	0.005	0.691	0.785	0.654	0.102	0.653	0.055	0.668	0.078	0.649	0.422
100	0.01	0.696	0.992	0.596	0.008	0.647	0.016	0.657	0.051	0.605	0.043
100	0.05	0.678	0.547	0.578	0.031	0.505	0.008	0.517	0.002	0.582	0.016
100	0.1	0.667	0.336	0.393	0.004	0.501	0.004	0.527	0.018	0.544	0.008
100	0.5	0.564	0.029	0.356	0.002	0.414	0.002	0.471	0.002	0.539	0.020
100	1	0.336	0.004	0.433	0.004	0.444	0.004	0.487	0.004	0.606	0.105

Table A.19: Glass: Boosted k-NN with randomized data order

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.703		0.730		0.734		0.736		0.751	
Boosted k-NN with randomized data order											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.694	0.293	0.718	0.086	0.729	0.707	0.726	0.320	0.748	0.754
10	0.01	0.707	0.699	0.734	0.781	0.737	0.813	0.738	0.855	0.755	0.641
10	0.05	0.719	0.365	0.721	0.578	0.737	0.875	0.719	0.313	0.720	0.129
10	0.1	0.724	0.148	0.717	0.281	0.702	0.111	0.722	0.379	0.682	0.002
10	0.5	0.722	0.371	0.658	0.002	0.691	0.092	0.675	0.004	0.680	0.004
10	1	0.713	0.441	0.660	0.020	0.682	0.047	0.669	0.002	0.661	0.002
100	0.005	0.714	0.094	0.731	1.000	0.742	0.297	0.739	0.766	0.751	0.988
100	0.01	0.709	0.234	0.744	0.180	0.741	0.398	0.747	0.375	0.739	0.242
100	0.05	0.741	0.002	0.721	0.652	0.674	0.014	0.663	0.004	0.659	0.002
100	0.1	0.744	0.020	0.661	0.002	0.653	0.002	0.654	0.002	0.653	0.002
100	0.5	0.750	0.008	0.651	0.002	0.651	0.002	0.653	0.002	0.653	0.002
100	1	0.749	0.043	0.651	0.002	0.650	0.002	0.661	0.004	0.653	0.002

Table A.20: Diabetes: Boosted k-NN with randomized data order

A.3 Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.871		0.846		0.854		0.826		0.793	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.871	1.000	0.846	1.000	0.844	0.500	0.825	0.875	0.821	0.063
10	0.01	0.871	1.000	0.846	1.000	0.849	1.000	0.825	0.781	0.819	0.250
10	0.05	0.871	1.000	0.861	0.375	0.854	1.000	0.834	0.797	0.844	0.047
10	0.1	0.871	1.000	0.867	0.328	0.840	0.406	0.859	0.223	0.831	0.156
10	0.5	0.871	1.000	0.843	1.000	0.791	0.090	0.819	0.680	0.823	0.250
10	1	0.871	1.000	0.794	0.109	0.769	0.008	0.799	0.309	0.813	0.250
100	0.005	0.871	1.000	0.856	0.750	0.854	1.000	0.830	1.000	0.849	0.063
100	0.01	0.871	1.000	0.862	0.453	0.846	0.781	0.859	0.223	0.831	0.156
100	0.05	0.871	1.000	0.829	0.500	0.801	0.031	0.829	1.000	0.848	0.023
100	0.1	0.871	1.000	0.819	0.250	0.786	0.008	0.809	0.531	0.831	0.031
100	0.5	0.871	1.000	0.818	0.188	0.761	0.008	0.796	0.188	0.844	0.016
100	1	0.871	1.000	0.789	0.109	0.760	0.008	0.789	0.055	0.780	0.668

Table A.21: Sonar: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.878		0.863		0.858		0.852		0.846	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.878	1.000	0.869	0.500	0.855	1.000	0.838	0.375	0.827	0.438
10	0.01	0.878	1.000	0.875	0.125	0.867	0.250	0.847	0.813	0.855	0.578
10	0.05	0.878	1.000	0.886	0.031	0.881	0.031	0.889	0.016	0.895	0.008
10	0.1	0.878	1.000	0.892	0.031	0.912	0.004	0.920	0.002	0.917	0.002
10	0.5	0.878	1.000	0.860	0.846	0.903	0.188	0.940	0.002	0.923	0.004
10	1	0.878	1.000	0.826	0.176	0.877	0.578	0.917	0.008	0.923	0.004
100	0.005	0.878	1.000	0.886	0.031	0.883	0.063	0.886	0.016	0.889	0.016
100	0.01	0.878	1.000	0.895	0.031	0.906	0.016	0.912	0.004	0.920	0.002
100	0.05	0.878	1.000	0.917	0.004	0.943	0.002	0.938	0.002	0.929	0.002
100	0.1	0.878	1.000	0.920	0.004	0.937	0.002	0.935	0.002	0.926	0.006
100	0.5	0.878	1.000	0.883	0.400	0.932	0.002	0.937	0.002	0.937	0.004
100	1	0.878	1.000	0.832	0.277	0.892	0.191	0.917	0.006	0.935	0.002

Table A.22: Ionosphere: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.951		0.961		0.955		0.963		0.978	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.951	1.000	0.961	1.000	0.961	1.000	0.972	0.500	0.978	1.000
10	0.01	0.951	1.000	0.961	1.000	0.961	1.000	0.978	0.250	0.972	1.000
10	0.05	0.951	1.000	0.965	1.000	0.971	0.250	0.982	0.250	0.972	1.000
10	0.1	0.951	1.000	0.965	1.000	0.971	0.250	0.982	0.250	0.976	1.000
10	0.5	0.951	1.000	0.965	1.000	0.976	0.125	0.976	0.500	0.959	0.500
10	1	0.951	1.000	0.971	0.750	0.982	0.125	0.976	0.500	0.959	0.500
100	0.005	0.951	1.000	0.965	1.000	0.965	0.500	0.978	0.250	0.978	1.000
100	0.01	0.951	1.000	0.965	1.000	0.965	0.500	0.982	0.250	0.976	1.000
100	0.05	0.951	1.000	0.965	1.000	0.971	0.250	0.982	0.250	0.976	1.000
100	0.1	0.951	1.000	0.965	1.000	0.971	0.250	0.982	0.250	0.976	1.000
100	0.5	0.951	1.000	0.965	1.000	0.976	0.125	0.976	0.500	0.959	0.500
100	1	0.951	1.000	0.971	0.750	0.982	0.125	0.976	0.500	0.959	0.500

Table A.23: Wine: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.638		0.644		0.621		0.630		0.657	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.638	1.000	0.647	0.941	0.651	0.367	0.649	0.504	0.662	0.902
10	0.01	0.638	1.000	0.636	0.832	0.640	0.469	0.646	0.525	0.660	1.000
10	0.05	0.638	1.000	0.659	0.594	0.645	0.441	0.658	0.426	0.635	0.492
10	0.1	0.638	1.000	0.676	0.252	0.633	0.734	0.657	0.383	0.654	0.906
10	0.5	0.638	1.000	0.670	0.320	0.640	0.492	0.657	0.387	0.651	0.797
10	1	0.638	1.000	0.675	0.180	0.631	0.758	0.649	0.516	0.648	0.703
100	0.005	0.638	1.000	0.644	1.000	0.639	0.563	0.666	0.326	0.689	0.203
100	0.01	0.638	1.000	0.651	0.730	0.648	0.359	0.689	0.063	0.663	0.922
100	0.05	0.638	1.000	0.647	0.895	0.618	1.000	0.663	0.234	0.666	0.852
100	0.1	0.638	1.000	0.670	0.281	0.622	1.000	0.666	0.219	0.671	0.717
100	0.5	0.638	1.000	0.659	0.613	0.628	0.840	0.647	0.660	0.680	0.525
100	1	0.638	1.000	0.660	0.547	0.622	1.000	0.650	0.508	0.665	0.859

Table A.24: Liver: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.990		0.984		0.982		0.978		0.970	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.990	1.000	0.982	0.500	0.985	0.500	0.982	0.359	0.977	0.238
10	0.01	0.990	1.000	0.982	0.500	0.986	0.250	0.984	0.109	0.980	0.107
10	0.05	0.990	1.000	0.981	0.500	0.986	0.281	0.983	0.234	0.978	0.273
10	0.1	0.990	1.000	0.978	0.125	0.983	0.719	0.976	0.867	0.971	0.965
10	0.5	0.990	1.000	0.976	0.063	0.979	0.531	0.964	0.074	0.963	0.281
10	1	0.990	1.000	0.977	0.125	0.979	0.531	0.963	0.031	0.961	0.234
100	0.005	0.990	1.000	0.982	0.500	0.986	0.250	0.984	0.188	0.981	0.070
100	0.01	0.990	1.000	0.982	0.500	0.986	0.250	0.985	0.148	0.978	0.164
100	0.05	0.990	1.000	0.981	0.500	0.986	0.281	0.983	0.320	0.976	0.461
100	0.1	0.990	1.000	0.978	0.125	0.983	0.719	0.974	0.633	0.969	0.934
100	0.5	0.990	1.000	0.977	0.063	0.979	0.531	0.967	0.203	0.960	0.164
100	1	0.990	1.000	0.978	0.125	0.978	0.359	0.964	0.039	0.958	0.105

Table A.25: Vowel: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.971		0.969		0.967		0.968		0.966	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.971	1.000	0.966	0.156	0.965	0.375	0.965	0.215	0.968	0.816
10	0.01	0.971	1.000	0.965	0.125	0.964	0.156	0.963	0.094	0.967	0.875
10	0.05	0.971	1.000	0.965	0.125	0.962	0.063	0.962	0.055	0.961	0.336
10	0.1	0.971	1.000	0.965	0.188	0.960	0.008	0.958	0.008	0.964	0.625
10	0.5	0.971	1.000	0.965	0.016	0.961	0.094	0.957	0.008	0.965	0.695
10	1	0.971	1.000	0.965	0.031	0.962	0.086	0.960	0.031	0.965	0.813
100	0.005	0.971	1.000	0.965	0.125	0.964	0.250	0.961	0.027	0.961	0.123
100	0.01	0.971	1.000	0.963	0.078	0.962	0.078	0.962	0.063	0.964	0.391
100	0.05	0.971	1.000	0.965	0.094	0.961	0.031	0.963	0.016	0.961	0.250
100	0.1	0.971	1.000	0.965	0.125	0.960	0.031	0.960	0.004	0.961	0.328
100	0.5	0.971	1.000	0.965	0.016	0.959	0.063	0.957	0.004	0.960	0.109
100	1	0.971	1.000	0.966	0.125	0.959	0.016	0.959	0.039	0.959	0.031

Table A.26: Segment: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.699		0.707		0.716		0.720		0.703	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.699	1.000	0.714	0.508	0.719	0.855	0.726	0.729	0.729	0.154
10	0.01	0.699	1.000	0.703	0.781	0.717	0.906	0.723	0.859	0.720	0.389
10	0.05	0.699	1.000	0.699	0.656	0.699	0.160	0.700	0.150	0.696	0.703
10	0.1	0.699	1.000	0.689	0.377	0.693	0.035	0.693	0.123	0.692	0.563
10	0.5	0.699	1.000	0.670	0.051	0.697	0.203	0.690	0.125	0.699	0.709
10	1	0.699	1.000	0.674	0.105	0.689	0.109	0.687	0.094	0.713	0.488
100	0.005	0.699	1.000	0.695	0.352	0.715	0.998	0.707	0.479	0.698	0.832
100	0.01	0.699	1.000	0.693	0.285	0.712	0.734	0.705	0.258	0.688	0.314
100	0.05	0.699	1.000	0.696	0.625	0.692	0.146	0.691	0.061	0.617	0.002
100	0.1	0.699	1.000	0.693	0.469	0.693	0.121	0.684	0.051	0.671	0.096
100	0.5	0.699	1.000	0.675	0.070	0.693	0.188	0.687	0.148	0.707	0.723
100	1	0.699	1.000	0.676	0.094	0.685	0.141	0.693	0.148	0.715	0.242

Table A.27: Vehicle: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.953		0.953		0.953		0.960		0.953	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.953	1.000	0.947	1.000	0.940	0.500	0.947	0.500	0.947	1.000
10	0.01	0.953	1.000	0.947	1.000	0.947	1.000	0.953	1.000	0.953	1.000
10	0.05	0.953	1.000	0.947	1.000	0.940	0.625	0.953	1.000	0.953	1.000
10	0.1	0.953	1.000	0.947	1.000	0.940	0.625	0.947	0.625	0.960	1.000
10	0.5	0.953	1.000	0.947	1.000	0.947	1.000	0.947	0.625	0.953	1.000
10	1	0.953	1.000	0.947	1.000	0.947	1.000	0.947	0.625	0.953	1.000
100	0.005	0.953	1.000	0.947	1.000	0.947	1.000	0.947	0.500	0.953	1.000
100	0.01	0.953	1.000	0.947	1.000	0.947	1.000	0.953	1.000	0.947	1.000
100	0.05	0.953	1.000	0.947	1.000	0.940	0.625	0.953	1.000	0.947	1.000
100	0.1	0.953	1.000	0.953	1.000	0.940	0.625	0.953	1.000	0.953	1.000
100	0.5	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.625	0.947	1.000
100	1	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.625	0.947	1.000

Table A.28: Iris: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.697		0.715		0.729		0.705		0.671	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.697	1.000	0.709	0.813	0.682	0.125	0.697	0.688	0.686	0.406
10	0.01	0.697	1.000	0.705	0.813	0.668	0.063	0.686	0.359	0.690	0.313
10	0.05	0.697	1.000	0.701	0.641	0.657	0.047	0.682	0.297	0.677	0.859
10	0.1	0.697	1.000	0.701	0.633	0.652	0.016	0.700	0.914	0.701	0.234
10	0.5	0.697	1.000	0.696	0.453	0.672	0.063	0.672	0.191	0.682	0.703
10	1	0.697	1.000	0.686	0.230	0.696	0.188	0.672	0.336	0.687	0.627
100	0.005	0.697	1.000	0.719	0.906	0.663	0.063	0.710	0.875	0.640	0.453
100	0.01	0.697	1.000	0.701	0.656	0.662	0.125	0.686	0.563	0.658	0.648
100	0.05	0.697	1.000	0.696	0.547	0.643	0.055	0.649	0.141	0.663	0.773
100	0.1	0.697	1.000	0.701	0.633	0.649	0.055	0.658	0.191	0.678	0.844
100	0.5	0.697	1.000	0.691	0.375	0.678	0.117	0.696	0.828	0.678	0.875
100	1	0.697	1.000	0.682	0.191	0.682	0.137	0.690	0.602	0.687	0.641

Table A.29: Glass: Boosted k-NN with batch update

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.703		0.730		0.734		0.736		0.751	
Boosted k-NN with batch update											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.703	1.000	0.724	0.500	0.707	0.082	0.728	0.559	0.717	0.035
10	0.01	0.703	1.000	0.719	0.313	0.711	0.078	0.731	0.797	0.731	0.109
10	0.05	0.703	1.000	0.726	0.719	0.723	0.664	0.727	0.563	0.748	0.867
10	0.1	0.703	1.000	0.727	0.770	0.737	0.852	0.746	0.531	0.751	0.988
10	0.5	0.703	1.000	0.740	0.316	0.732	0.904	0.758	0.109	0.760	0.531
10	1	0.703	1.000	0.737	0.477	0.739	0.793	0.766	0.023	0.763	0.277
100	0.005	0.703	1.000	0.721	0.355	0.727	0.596	0.718	0.332	0.723	0.070
100	0.01	0.703	1.000	0.717	0.199	0.717	0.313	0.727	0.576	0.737	0.424
100	0.05	0.703	1.000	0.723	0.543	0.723	0.570	0.734	0.957	0.733	0.215
100	0.1	0.703	1.000	0.726	0.742	0.737	0.832	0.743	0.676	0.750	0.984
100	0.5	0.703	1.000	0.734	0.563	0.745	0.539	0.758	0.105	0.756	0.773
100	1	0.703	1.000	0.733	0.641	0.728	0.734	0.758	0.180	0.753	0.879

Table A.30: Diabetes: Boosted k-NN with batch update

A.4 Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.871		0.846		0.854		0.826		0.793	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.866	1.000	0.856	1.000	0.854	1.000	0.830	1.000	0.815	0.125
10	0.01	0.861	0.500	0.856	1.000	0.844	0.750	0.830	1.000	0.829	0.047
10	0.05	0.872	1.000	0.871	0.250	0.825	0.188	0.855	0.070	0.837	0.129
10	0.1	0.877	0.688	0.877	0.188	0.835	0.539	0.859	0.094	0.852	0.031
10	0.5	0.896	0.313	0.882	0.188	0.865	0.625	0.882	0.027	0.861	0.023
10	1	0.876	1.000	0.857	0.750	0.869	0.594	0.882	0.031	0.864	0.023
100	0.005	0.857	0.375	0.876	0.125	0.840	0.613	0.845	0.219	0.847	0.078
100	0.01	0.872	1.000	0.871	0.188	0.849	0.969	0.859	0.094	0.844	0.070
100	0.05	0.881	0.750	0.877	0.188	0.874	0.422	0.854	0.094	0.867	0.016
100	0.1	0.881	0.750	0.879	0.281	0.867	0.688	0.844	0.375	0.872	0.016
100	0.5	0.896	0.313	0.894	0.055	0.892	0.172	0.877	0.016	0.907	0.008
100	1	0.876	1.000	0.887	0.156	0.897	0.078	0.892	0.016	0.872	0.008

Table A.31: Sonar: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.878		0.863		0.858		0.852		0.846	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.889	0.125	0.869	0.625	0.847	0.406	0.855	0.813	0.847	0.914
10	0.01	0.889	0.219	0.872	0.531	0.861	0.906	0.861	0.516	0.864	0.328
10	0.05	0.898	0.063	0.912	0.004	0.909	0.016	0.920	0.004	0.943	0.002
10	0.1	0.906	0.094	0.915	0.008	0.926	0.016	0.946	0.002	0.943	0.002
10	0.5	0.917	0.063	0.935	0.004	0.937	0.002	0.940	0.002	0.935	0.002
10	1	0.923	0.008	0.935	0.002	0.929	0.004	0.937	0.002	0.949	0.002
100	0.005	0.892	0.234	0.906	0.016	0.909	0.008	0.917	0.006	0.932	0.002
100	0.01	0.906	0.074	0.920	0.008	0.932	0.004	0.932	0.002	0.943	0.002
100	0.05	0.917	0.051	0.937	0.002	0.940	0.002	0.943	0.002	0.946	0.002
100	0.1	0.912	0.090	0.934	0.002	0.943	0.002	0.935	0.002	0.949	0.002
100	0.5	0.937	0.016	0.940	0.002	0.937	0.002	0.937	0.002	0.943	0.002
100	1	0.940	0.020	0.946	0.002	0.946	0.002	0.955	0.002	0.960	0.002

Table A.32: Ionosphere: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.951		0.961		0.955		0.963		0.978	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.951	1.000	0.961	1.000	0.961	1.000	0.968	1.000	0.978	1.000
10	0.01	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.978	1.000
10	0.05	0.951	1.000	0.965	1.000	0.967	0.500	0.978	0.250	0.978	1.000
10	0.1	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
10	0.5	0.951	1.000	0.976	0.250	0.978	0.250	0.976	0.500	0.982	1.000
10	1	0.951	1.000	0.971	0.500	0.982	0.125	0.971	0.750	0.965	0.500
100	0.005	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.978	1.000
100	0.01	0.951	1.000	0.971	0.500	0.967	0.500	0.976	0.500	0.982	1.000
100	0.05	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
100	0.1	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
100	0.5	0.951	1.000	0.976	0.250	0.978	0.250	0.976	0.500	0.982	1.000
100	1	0.951	1.000	0.971	0.500	0.982	0.125	0.971	0.750	0.965	0.500

Table A.33: Wine: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.638		0.644		0.621		0.630		0.657	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.638	1.000	0.658	0.619	0.658	0.133	0.669	0.307	0.657	1.000
10	0.01	0.635	1.000	0.644	1.000	0.643	0.547	0.643	0.754	0.644	0.477
10	0.05	0.644	0.828	0.637	0.742	0.600	0.379	0.639	0.830	0.616	0.168
10	0.1	0.603	0.129	0.615	0.398	0.581	0.156	0.610	0.605	0.572	0.023
10	0.5	0.608	0.328	0.600	0.094	0.577	0.156	0.583	0.227	0.579	0.004
10	1	0.637	1.000	0.597	0.180	0.585	0.156	0.556	0.023	0.552	0.016
100	0.005	0.635	1.000	0.619	0.402	0.659	0.219	0.652	0.529	0.666	0.656
100	0.01	0.624	0.688	0.635	0.836	0.651	0.371	0.649	0.605	0.649	0.813
100	0.05	0.609	0.350	0.619	0.375	0.629	0.793	0.607	0.586	0.568	0.031
100	0.1	0.585	0.066	0.606	0.234	0.635	0.709	0.626	0.934	0.591	0.033
100	0.5	0.600	0.227	0.588	0.172	0.591	0.242	0.626	0.910	0.597	0.158
100	1	0.600	0.221	0.594	0.156	0.594	0.313	0.588	0.223	0.602	0.064

Table A.34: Liver: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.990		0.984		0.982		0.978		0.970	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.990	1.000	0.984	0.750	0.985	0.500	0.982	0.359	0.979	0.109
10	0.01	0.990	1.000	0.987	0.500	0.986	0.250	0.983	0.188	0.979	0.148
10	0.05	0.989	1.000	0.982	0.781	0.988	0.094	0.986	0.070	0.977	0.438
10	0.1	0.987	1.000	0.980	0.188	0.982	0.844	0.980	0.797	0.973	0.766
10	0.5	0.987	1.000	0.980	0.313	0.960	0.008	0.931	0.002	0.895	0.002
10	1	0.987	1.000	0.978	0.391	0.963	0.016	0.929	0.008	0.815	0.002
100	0.005	0.990	1.000	0.984	0.750	0.986	0.250	0.984	0.109	0.980	0.086
100	0.01	0.990	1.000	0.987	0.500	0.987	0.250	0.982	0.359	0.981	0.102
100	0.05	0.989	1.000	0.982	0.781	0.988	0.094	0.984	0.230	0.977	0.422
100	0.1	0.988	1.000	0.981	0.500	0.984	0.750	0.982	0.453	0.975	0.547
100	0.5	0.976	0.031	0.956	0.008	0.905	0.002	0.863	0.002	0.866	0.002
100	1	0.975	0.031	0.971	0.008	0.890	0.006	0.726	0.002	0.602	0.002

Table A.35: Vowel: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.971		0.969		0.967		0.968		0.966	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.969	0.336	0.968	0.938	0.970	0.375	0.968	1.000	0.968	0.699
10	0.01	0.968	0.223	0.967	0.500	0.968	0.750	0.964	0.180	0.965	0.848
10	0.05	0.969	0.125	0.970	0.703	0.964	0.512	0.965	0.125	0.967	0.930
10	0.1	0.965	0.063	0.968	0.709	0.968	0.586	0.964	0.172	0.960	0.188
10	0.5	0.961	0.008	0.961	0.078	0.955	0.039	0.955	0.004	0.954	0.051
10	1	0.962	0.008	0.959	0.004	0.959	0.141	0.950	0.004	0.944	0.004
100	0.005	0.970	0.375	0.967	0.316	0.967	1.000	0.964	0.242	0.965	0.891
100	0.01	0.971	0.805	0.968	0.570	0.966	0.859	0.964	0.141	0.965	0.617
100	0.05	0.965	0.070	0.965	0.203	0.964	0.359	0.966	0.352	0.962	0.395
100	0.1	0.956	0.004	0.961	0.012	0.963	0.234	0.963	0.168	0.961	0.359
100	0.5	0.906	0.002	0.918	0.002	0.920	0.002	0.925	0.002	0.933	0.004
100	1	0.861	0.002	0.900	0.002	0.901	0.002	0.906	0.002	0.906	0.002

Table A.36: Segment: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.699		0.707		0.716		0.720		0.703	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.692	0.656	0.707	0.996	0.725	0.457	0.731	0.551	0.733	0.090
10	0.01	0.687	0.305	0.703	0.875	0.721	0.570	0.722	0.922	0.734	0.068
10	0.05	0.690	0.531	0.709	0.945	0.728	0.164	0.715	0.586	0.721	0.145
10	0.1	0.688	0.289	0.708	1.000	0.714	0.906	0.719	0.885	0.715	0.586
10	0.5	0.648	0.016	0.643	0.021	0.631	0.002	0.636	0.002	0.644	0.008
10	1	0.621	0.008	0.567	0.002	0.568	0.002	0.556	0.002	0.544	0.002
100	0.005	0.693	0.500	0.717	0.594	0.716	1.000	0.718	0.875	0.719	0.250
100	0.01	0.698	0.904	0.717	0.523	0.713	0.750	0.715	0.730	0.723	0.227
100	0.05	0.674	0.098	0.706	0.879	0.705	0.238	0.713	0.625	0.717	0.434
100	0.1	0.656	0.008	0.693	0.328	0.703	0.363	0.694	0.104	0.708	0.883
100	0.5	0.630	0.012	0.506	0.002	0.440	0.002	0.507	0.002	0.546	0.002
100	1	0.598	0.004	0.385	0.002	0.495	0.002	0.494	0.002	0.528	0.002

Table A.37: Vehicle: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.953		0.953		0.953		0.960		0.953	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.947	1.000	0.953	1.000	0.947	1.000	0.947	0.500	0.947	1.000
10	0.01	0.947	1.000	0.960	1.000	0.947	1.000	0.953	1.000	0.953	1.000
10	0.05	0.953	1.000	0.960	1.000	0.947	1.000	0.947	0.625	0.947	1.000
10	0.1	0.953	1.000	0.960	1.000	0.947	1.000	0.947	0.625	0.947	1.000
10	0.5	0.953	1.000	0.960	1.000	0.953	1.000	0.960	1.000	0.960	1.000
10	1	0.953	1.000	0.960	1.000	0.947	1.000	0.947	0.625	0.947	1.000
100	0.005	0.947	1.000	0.953	1.000	0.953	1.000	0.953	1.000	0.953	1.000
100	0.01	0.947	1.000	0.947	1.000	0.947	1.000	0.953	1.000	0.953	1.000
100	0.05	0.953	1.000	0.947	1.000	0.947	1.000	0.947	0.625	0.953	1.000
100	0.1	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.625	0.947	1.000
100	0.5	0.947	1.000	0.947	1.000	0.940	0.750	0.947	0.500	0.940	0.750
100	1	0.940	1.000	0.933	0.250	0.913	0.250	0.907	0.063	0.920	0.188

Table A.38: Iris: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.697		0.715		0.729		0.705		0.671	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.697	1.000	0.697	0.406	0.700	0.125	0.691	0.359	0.695	0.438
10	0.01	0.720	0.500	0.701	0.672	0.704	0.234	0.691	0.430	0.694	0.375
10	0.05	0.672	0.313	0.673	0.336	0.695	0.156	0.681	0.281	0.662	0.797
10	0.1	0.705	0.902	0.670	0.152	0.653	0.031	0.671	0.031	0.652	0.375
10	0.5	0.647	0.219	0.604	0.041	0.611	0.008	0.615	0.047	0.620	0.086
10	1	0.620	0.047	0.577	0.016	0.489	0.004	0.469	0.004	0.586	0.047
100	0.005	0.682	0.656	0.709	0.844	0.695	0.203	0.695	0.719	0.695	0.438
100	0.01	0.667	0.313	0.732	0.500	0.699	0.258	0.695	0.688	0.694	0.375
100	0.05	0.605	0.078	0.690	0.359	0.681	0.055	0.666	0.078	0.651	0.430
100	0.1	0.564	0.033	0.662	0.119	0.652	0.031	0.671	0.289	0.652	0.625
100	0.5	0.435	0.002	0.509	0.004	0.580	0.008	0.507	0.006	0.629	0.469
100	1	0.405	0.002	0.470	0.002	0.409	0.002	0.432	0.002	0.583	0.055

Table A.39: Glass: Boosted k-NN with error-weighted voting

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.703		0.730		0.734		0.736		0.751	
Boosted k-NN with error-weighted voting											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.712	0.387	0.720	0.246	0.729	0.811	0.727	0.453	0.732	0.141
10	0.01	0.716	0.301	0.725	0.652	0.733	0.992	0.740	0.578	0.734	0.078
10	0.05	0.727	0.125	0.737	0.732	0.735	1.000	0.737	0.879	0.731	0.094
10	0.1	0.734	0.055	0.740	0.512	0.726	0.633	0.748	0.188	0.716	0.012
10	0.5	0.744	0.031	0.763	0.031	0.755	0.203	0.744	0.635	0.745	0.828
10	1	0.742	0.066	0.709	0.490	0.729	0.816	0.736	1.000	0.728	0.086
100	0.005	0.714	0.502	0.709	0.102	0.712	0.027	0.713	0.125	0.735	0.150
100	0.01	0.719	0.371	0.726	0.734	0.716	0.250	0.715	0.156	0.736	0.281
100	0.05	0.744	0.016	0.732	0.914	0.728	0.695	0.740	0.820	0.732	0.205
100	0.1	0.755	0.004	0.744	0.250	0.749	0.336	0.744	0.512	0.746	0.805
100	0.5	0.753	0.018	0.720	0.625	0.737	0.832	0.757	0.303	0.760	0.637
100	1	0.753	0.049	0.692	0.094	0.737	0.891	0.760	0.172	0.749	0.938

Table A.40: Diabetes: Boosted k-NN with error-weighted voting

A.5 Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.871		0.846		0.854		0.826		0.793	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.871	1.000	0.846	1.000	0.854	1.000	0.816	0.500	0.801	0.500
10	0.01	0.866	1.000	0.856	1.000	0.839	0.500	0.835	0.656	0.829	0.094
10	0.05	0.872	1.000	0.872	0.250	0.840	0.563	0.855	0.109	0.854	0.023
10	0.1	0.882	0.531	0.882	0.250	0.829	0.313	0.864	0.031	0.844	0.063
10	0.5	0.896	0.313	0.881	0.250	0.877	0.250	0.847	0.406	0.855	0.047
10	1	0.876	1.000	0.841	0.922	0.874	0.453	0.879	0.031	0.894	0.016
100	0.001	0.866	1.000	0.856	1.000	0.839	0.500	0.835	0.656	0.834	0.094
100	0.01	0.881	0.688	0.862	0.500	0.852	0.984	0.862	0.094	0.862	0.008
100	0.05	0.886	0.531	0.877	0.250	0.871	0.422	0.850	0.188	0.886	0.012
100	0.1	0.886	0.531	0.859	0.750	0.857	0.977	0.864	0.141	0.872	0.027
100	0.5	0.896	0.313	0.901	0.078	0.887	0.344	0.897	0.008	0.889	0.023
100	1	0.876	1.000	0.889	0.063	0.882	0.219	0.831	1.000	0.852	0.031

Table A.41: Sonar: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.878		0.863		0.858		0.852		0.846	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.880	1.000	0.863	1.000	0.858	1.000	0.846	0.500	0.838	0.500
10	0.01	0.889	0.359	0.878	0.375	0.875	0.313	0.863	0.516	0.880	0.031
10	0.05	0.889	0.219	0.912	0.008	0.932	0.002	0.926	0.002	0.940	0.004
10	0.1	0.906	0.109	0.912	0.008	0.932	0.008	0.949	0.002	0.949	0.002
10	0.5	0.912	0.072	0.932	0.020	0.926	0.004	0.937	0.002	0.932	0.002
10	1	0.932	0.008	0.932	0.002	0.921	0.002	0.935	0.002	0.929	0.008
100	0.001	0.889	0.359	0.875	0.500	0.872	0.250	0.866	0.227	0.880	0.031
100	0.01	0.915	0.055	0.917	0.004	0.935	0.004	0.935	0.002	0.943	0.002
100	0.05	0.909	0.102	0.943	0.002	0.946	0.002	0.940	0.002	0.946	0.002
100	0.1	0.917	0.063	0.929	0.004	0.946	0.002	0.937	0.002	0.949	0.002
100	0.5	0.923	0.039	0.935	0.004	0.929	0.002	0.943	0.002	0.940	0.002
100	1	0.926	0.035	0.932	0.002	0.920	0.008	0.940	0.002	0.954	0.002

Table A.42: Ionosphere: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.951		0.961		0.955		0.963		0.978	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.951	1.000	0.961	1.000	0.955	1.000	0.963	1.000	0.978	1.000
10	0.01	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.972	1.000
10	0.05	0.951	1.000	0.965	1.000	0.967	0.500	0.982	0.250	0.978	1.000
10	0.1	0.951	1.000	0.971	0.500	0.978	0.250	0.982	0.250	0.976	1.000
10	0.5	0.945	1.000	0.976	0.250	0.982	0.125	0.976	0.500	0.982	1.000
10	1	0.945	1.000	0.971	0.500	0.982	0.125	0.976	0.500	0.965	0.500
100	0.001	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.972	1.000
100	0.01	0.951	1.000	0.965	1.000	0.967	0.500	0.976	0.500	0.976	1.000
100	0.05	0.951	1.000	0.971	0.500	0.972	0.500	0.982	0.250	0.976	1.000
100	0.1	0.945	1.000	0.971	0.500	0.978	0.250	0.982	0.250	0.976	1.000
100	0.5	0.945	1.000	0.976	0.250	0.982	0.125	0.976	0.500	0.982	1.000
100	1	0.945	1.000	0.971	0.500	0.982	0.125	0.976	0.500	0.965	0.500

Table A.43: Wine: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.638		0.644		0.621		0.630		0.657	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.623	0.234	0.659	0.602	0.641	0.400	0.632	1.000	0.668	0.438
10	0.01	0.629	0.805	0.626	0.584	0.643	0.473	0.638	0.855	0.627	0.250
10	0.05	0.626	0.754	0.624	0.514	0.568	0.090	0.582	0.203	0.599	0.031
10	0.1	0.628	0.770	0.582	0.102	0.549	0.117	0.581	0.145	0.587	0.043
10	0.5	0.624	0.664	0.556	0.020	0.567	0.090	0.556	0.176	0.520	0.004
10	1	0.624	0.664	0.622	0.676	0.530	0.094	0.575	0.258	0.538	0.002
100	0.001	0.626	0.438	0.644	1.000	0.640	0.508	0.640	0.764	0.659	1.000
100	0.01	0.607	0.316	0.627	0.695	0.635	0.672	0.644	0.678	0.607	0.104
100	0.05	0.614	0.482	0.627	0.623	0.582	0.227	0.602	0.391	0.595	0.078
100	0.1	0.628	0.770	0.608	0.344	0.557	0.105	0.561	0.131	0.630	0.418
100	0.5	0.624	0.664	0.556	0.020	0.570	0.162	0.573	0.105	0.600	0.039
100	1	0.624	0.664	0.622	0.676	0.556	0.156	0.531	0.016	0.547	0.002

Table A.44: Liver: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.990		0.984		0.982		0.978		0.970	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.990	1.000	0.984	1.000	0.982	1.000	0.982	0.125	0.977	0.125
10	0.01	0.990	1.000	0.987	0.500	0.987	0.250	0.983	0.297	0.983	0.039
10	0.05	0.989	1.000	0.982	0.781	0.987	0.188	0.984	0.230	0.976	0.500
10	0.1	0.987	1.000	0.981	0.375	0.982	0.859	0.981	0.656	0.973	0.813
10	0.5	0.986	0.500	0.976	0.031	0.960	0.010	0.945	0.031	0.901	0.002
10	1	0.986	0.500	0.980	0.469	0.966	0.023	0.942	0.008	0.884	0.002
100	0.001	0.990	1.000	0.986	0.500	0.986	0.250	0.983	0.188	0.979	0.094
100	0.01	0.990	1.000	0.987	0.500	0.986	0.375	0.983	0.297	0.983	0.039
100	0.05	0.989	1.000	0.981	0.578	0.988	0.094	0.984	0.184	0.974	0.703
100	0.1	0.988	1.000	0.979	0.156	0.982	1.000	0.981	0.578	0.976	0.500
100	0.5	0.987	1.000	0.978	0.094	0.960	0.010	0.945	0.031	0.901	0.002
100	1	0.987	1.000	0.980	0.469	0.966	0.023	0.942	0.008	0.884	0.002

Table A.45: Vowel: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.971		0.969		0.967		0.968		0.966	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.971	1.000	0.970	0.688	0.969	0.188	0.969	0.750	0.967	0.883
10	0.01	0.968	0.383	0.966	0.141	0.966	0.813	0.965	0.332	0.962	0.340
10	0.05	0.965	0.008	0.968	0.941	0.961	0.125	0.965	0.141	0.964	0.391
10	0.1	0.965	0.070	0.966	0.500	0.961	0.102	0.959	0.018	0.957	0.086
10	0.5	0.960	0.002	0.960	0.025	0.956	0.039	0.951	0.002	0.951	0.012
10	1	0.960	0.002	0.961	0.078	0.960	0.094	0.953	0.004	0.949	0.020
100	0.001	0.970	0.336	0.967	0.406	0.968	0.789	0.968	1.000	0.965	0.602
100	0.01	0.970	0.719	0.966	0.203	0.966	0.875	0.963	0.109	0.963	0.531
100	0.05	0.963	0.055	0.963	0.063	0.964	0.453	0.961	0.105	0.961	0.191
100	0.1	0.960	0.031	0.966	0.500	0.961	0.117	0.957	0.012	0.961	0.289
100	0.5	0.960	0.002	0.960	0.025	0.956	0.039	0.951	0.002	0.951	0.012
100	1	0.960	0.002	0.961	0.078	0.960	0.094	0.953	0.004	0.949	0.020

Table A.46: Segment: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.699		0.707		0.716		0.720		0.703	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.700	1.000	0.709	0.711	0.716	1.000	0.719	0.914	0.715	0.328
10	0.01	0.676	0.109	0.708	1.000	0.707	0.473	0.719	0.879	0.727	0.125
10	0.05	0.694	0.711	0.701	0.754	0.719	0.844	0.707	0.373	0.701	0.867
10	0.1	0.670	0.082	0.680	0.207	0.696	0.094	0.682	0.070	0.694	0.695
10	0.5	0.656	0.016	0.650	0.004	0.661	0.002	0.664	0.004	0.617	0.004
10	1	0.657	0.016	0.636	0.035	0.669	0.004	0.664	0.025	0.592	0.002
100	0.001	0.687	0.375	0.726	0.143	0.719	0.801	0.721	1.000	0.736	0.055
100	0.01	0.696	0.654	0.702	0.754	0.707	0.313	0.692	0.156	0.722	0.203
100	0.05	0.690	0.592	0.705	0.906	0.711	0.746	0.694	0.145	0.712	0.750
100	0.1	0.670	0.082	0.680	0.207	0.698	0.094	0.690	0.078	0.698	0.906
100	0.5	0.656	0.016	0.650	0.004	0.661	0.002	0.664	0.004	0.617	0.004
100	1	0.657	0.016	0.636	0.035	0.669	0.004	0.664	0.025	0.592	0.002

Table A.47: Vehicle: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.953		0.953		0.953		0.960		0.953	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.960	1.000	0.953	1.000	0.947	1.000	0.953	1.000	0.947	1.000
10	0.01	0.947	1.000	0.960	1.000	0.947	1.000	0.947	0.500	0.947	1.000
10	0.05	0.967	0.500	0.967	0.625	0.947	1.000	0.953	1.000	0.947	1.000
10	0.1	0.967	0.500	0.953	1.000	0.953	1.000	0.947	0.625	0.947	1.000
10	0.5	0.967	0.500	0.960	1.000	0.947	1.000	0.953	1.000	0.960	1.000
10	1	0.967	0.500	0.953	1.000	0.940	0.500	0.933	0.125	0.947	1.000
100	0.001	0.947	1.000	0.960	1.000	0.953	1.000	0.947	0.500	0.947	1.000
100	0.01	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.500	0.953	1.000
100	0.05	0.967	0.500	0.947	1.000	0.947	1.000	0.947	0.625	0.947	1.000
100	0.1	0.973	0.250	0.960	1.000	0.940	0.750	0.947	0.625	0.947	1.000
100	0.5	0.967	0.500	0.953	1.000	0.940	0.500	0.947	0.500	0.953	1.000
100	1	0.967	0.500	0.953	1.000	0.940	0.500	0.933	0.125	0.940	0.688

Table A.48: Iris: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.697		0.715		0.729		0.705		0.671	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.701	1.000	0.735	0.141	0.712	0.375	0.715	0.594	0.724	0.047
10	0.01	0.697	1.000	0.701	0.555	0.681	0.109	0.695	0.563	0.699	0.344
10	0.05	0.697	1.000	0.663	0.166	0.656	0.023	0.653	0.070	0.631	0.219
10	0.1	0.692	0.922	0.654	0.070	0.677	0.047	0.643	0.031	0.631	0.287
10	0.5	0.673	0.438	0.646	0.078	0.633	0.014	0.629	0.047	0.594	0.111
10	1	0.667	0.250	0.647	0.117	0.662	0.008	0.587	0.016	0.606	0.063
100	0.001	0.697	0.992	0.693	0.453	0.716	0.531	0.682	0.383	0.705	0.188
100	0.01	0.711	0.688	0.713	0.977	0.697	0.359	0.700	0.883	0.680	0.844
100	0.05	0.697	1.000	0.678	0.387	0.616	0.004	0.655	0.148	0.655	0.516
100	0.1	0.692	0.922	0.644	0.070	0.634	0.008	0.639	0.023	0.607	0.051
100	0.5	0.673	0.438	0.646	0.078	0.633	0.014	0.629	0.047	0.633	0.289
100	1	0.667	0.250	0.647	0.117	0.662	0.008	0.587	0.016	0.576	0.039

Table A.49: Glass: Boosted k-NN with optimal weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.703		0.730		0.734		0.736		0.751	
Boosted k-NN with optimal weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.001	0.713	0.313	0.734	0.813	0.721	0.305	0.724	0.359	0.729	0.008
10	0.01	0.707	0.771	0.718	0.352	0.726	0.563	0.732	0.766	0.722	0.090
10	0.05	0.722	0.234	0.709	0.313	0.734	0.980	0.718	0.414	0.725	0.127
10	0.1	0.724	0.129	0.725	0.680	0.711	0.098	0.739	0.773	0.676	0.008
10	0.5	0.703	0.986	0.731	0.992	0.716	0.283	0.695	0.094	0.733	0.539
10	1	0.703	0.986	0.726	0.791	0.723	0.547	0.714	0.051	0.681	0.004
100	0.001	0.708	0.590	0.717	0.285	0.721	0.316	0.713	0.025	0.721	0.023
100	0.01	0.712	0.629	0.724	0.578	0.717	0.406	0.706	0.063	0.716	0.023
100	0.05	0.723	0.141	0.709	0.275	0.709	0.051	0.688	0.025	0.706	0.018
100	0.1	0.722	0.344	0.727	0.785	0.705	0.355	0.725	0.525	0.696	0.037
100	0.5	0.703	0.986	0.728	0.961	0.729	0.813	0.707	0.084	0.737	0.619
100	1	0.703	0.986	0.723	0.637	0.736	0.922	0.682	0.361	0.717	0.070

Table A.50: Diabetes: Boosted k-NN with optimal weights

A.6 Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.871		0.846		0.854		0.826		0.793	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.861	0.500	0.856	1.000	0.849	1.000	0.830	1.000	0.801	0.625
10	0.01	0.861	0.500	0.856	1.000	0.844	0.750	0.835	0.688	0.824	0.094
10	0.05	0.857	0.375	0.876	0.125	0.839	0.563	0.855	0.109	0.837	0.129
10	0.1	0.872	1.000	0.886	0.125	0.834	0.469	0.849	0.375	0.846	0.109
10	0.5	0.886	0.594	0.867	0.500	0.869	0.375	0.861	0.250	0.872	0.012
10	1	0.886	0.625	0.867	0.500	0.884	0.063	0.872	0.078	0.875	0.035
100	0.005	0.862	0.625	0.876	0.125	0.839	0.594	0.850	0.188	0.844	0.047
100	0.01	0.867	1.000	0.871	0.188	0.834	0.406	0.855	0.188	0.854	0.070
100	0.05	0.876	1.000	0.882	0.156	0.869	0.531	0.852	0.188	0.862	0.016
100	0.1	0.876	1.000	0.874	0.375	0.862	0.813	0.847	0.406	0.867	0.055
100	0.5	0.891	0.406	0.884	0.109	0.874	0.500	0.855	0.109	0.864	0.023
100	1	0.886	0.625	0.866	0.625	0.861	0.844	0.887	0.023	0.824	0.250

Table A.51: Sonar: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.878		0.863		0.858		0.852		0.846	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.889	0.125	0.869	0.625	0.849	0.563	0.849	1.000	0.838	0.813
10	0.01	0.889	0.219	0.875	0.344	0.858	1.000	0.855	0.813	0.861	0.383
10	0.05	0.883	0.531	0.903	0.004	0.897	0.047	0.917	0.006	0.935	0.002
10	0.1	0.906	0.094	0.906	0.008	0.926	0.004	0.937	0.002	0.940	0.002
10	0.5	0.923	0.043	0.906	0.129	0.929	0.002	0.935	0.004	0.929	0.002
10	1	0.923	0.031	0.912	0.037	0.917	0.051	0.895	0.035	0.914	0.012
100	0.005	0.889	0.359	0.900	0.023	0.906	0.016	0.912	0.006	0.923	0.008
100	0.01	0.903	0.094	0.909	0.016	0.920	0.008	0.929	0.002	0.943	0.002
100	0.05	0.915	0.066	0.935	0.002	0.932	0.002	0.937	0.002	0.951	0.002
100	0.1	0.909	0.117	0.940	0.002	0.923	0.020	0.937	0.002	0.937	0.008
100	0.5	0.935	0.016	0.926	0.004	0.934	0.002	0.920	0.010	0.929	0.002
100	1	0.923	0.035	0.880	0.625	0.889	0.188	0.866	0.627	0.906	0.008

Table A.52: Ionosphere: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.951		0.961		0.955		0.963		0.978	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.951	1.000	0.961	1.000	0.961	1.000	0.968	1.000	0.978	1.000
10	0.01	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.978	1.000
10	0.05	0.951	1.000	0.965	1.000	0.961	1.000	0.978	0.250	0.978	1.000
10	0.1	0.951	1.000	0.971	0.500	0.978	0.250	0.982	0.250	0.976	1.000
10	0.5	0.945	1.000	0.976	0.250	0.982	0.125	0.982	0.250	0.982	1.000
10	1	0.945	1.000	0.971	0.500	0.982	0.125	0.976	0.500	0.971	0.500
100	0.005	0.951	1.000	0.965	1.000	0.961	1.000	0.968	1.000	0.978	1.000
100	0.01	0.951	1.000	0.971	0.500	0.967	0.500	0.976	0.500	0.982	1.000
100	0.05	0.951	1.000	0.971	0.500	0.967	0.500	0.982	0.250	0.976	1.000
100	0.1	0.951	1.000	0.971	0.500	0.978	0.250	0.982	0.250	0.976	1.000
100	0.5	0.945	1.000	0.976	0.250	0.982	0.125	0.982	0.250	0.982	1.000
100	1	0.945	1.000	0.971	0.500	0.982	0.125	0.976	0.500	0.971	0.500

Table A.53: Wine: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.638		0.644		0.621		0.630		0.657	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.642	0.859	0.652	0.758	0.653	0.375	0.651	0.547	0.652	0.824
10	0.01	0.641	0.891	0.661	0.539	0.652	0.297	0.641	0.777	0.641	0.566
10	0.05	0.609	0.465	0.575	0.033	0.522	0.012	0.591	0.281	0.540	0.002
10	0.1	0.576	0.078	0.543	0.029	0.448	0.002	0.523	0.094	0.473	0.002
10	0.5	0.598	0.209	0.547	0.047	0.541	0.068	0.497	0.023	0.486	0.002
10	1	0.572	0.078	0.538	0.020	0.530	0.111	0.452	0.004	0.449	0.002
100	0.005	0.609	0.133	0.624	0.594	0.663	0.219	0.609	0.555	0.655	0.969
100	0.01	0.630	0.871	0.614	0.375	0.650	0.416	0.637	0.859	0.635	0.359
100	0.05	0.560	0.039	0.544	0.008	0.505	0.004	0.485	0.006	0.454	0.002
100	0.1	0.603	0.164	0.538	0.023	0.464	0.002	0.443	0.002	0.478	0.002
100	0.5	0.588	0.129	0.496	0.027	0.524	0.039	0.562	0.039	0.449	0.002
100	1	0.565	0.068	0.530	0.078	0.488	0.016	0.470	0.004	0.472	0.004

Table A.54: Liver: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.990		0.984		0.982		0.978		0.970	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.990	1.000	0.985	0.500	0.985	0.500	0.982	0.359	0.980	0.086
10	0.01	0.990	1.000	0.986	0.500	0.986	0.250	0.982	0.313	0.978	0.207
10	0.05	0.989	1.000	0.983	1.000	0.986	0.281	0.982	0.391	0.976	0.438
10	0.1	0.988	1.000	0.979	0.063	0.981	1.000	0.978	0.891	0.974	0.652
10	0.5	0.984	0.250	0.972	0.039	0.944	0.008	0.890	0.002	0.847	0.002
10	1	0.980	0.063	0.956	0.004	0.914	0.002	0.780	0.002	0.701	0.002
100	0.005	0.990	1.000	0.985	0.500	0.986	0.250	0.983	0.188	0.979	0.109
100	0.01	0.990	1.000	0.986	0.500	0.987	0.250	0.982	0.313	0.980	0.145
100	0.05	0.989	1.000	0.983	1.000	0.986	0.281	0.983	0.270	0.978	0.359
100	0.1	0.988	1.000	0.980	0.313	0.982	1.000	0.985	0.156	0.975	0.578
100	0.5	0.897	0.031	0.726	0.002	0.711	0.002	0.717	0.002	0.727	0.002
100	1	0.779	0.031	0.381	0.002	0.402	0.002	0.449	0.002	0.493	0.002

Table A.55: Vowel: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.971		0.969		0.967		0.968		0.966	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.970	0.594	0.968	0.844	0.969	0.492	0.968	0.988	0.968	0.672
10	0.01	0.968	0.227	0.968	0.516	0.967	0.891	0.963	0.109	0.967	0.994
10	0.05	0.969	0.180	0.968	0.516	0.964	0.377	0.963	0.133	0.967	0.945
10	0.1	0.965	0.023	0.962	0.133	0.962	0.385	0.963	0.219	0.963	0.367
10	0.5	0.955	0.014	0.937	0.002	0.936	0.004	0.935	0.002	0.935	0.006
10	1	0.942	0.002	0.919	0.002	0.897	0.002	0.902	0.002	0.880	0.002
100	0.005	0.970	0.375	0.968	0.367	0.969	0.510	0.966	0.488	0.963	0.484
100	0.01	0.971	0.949	0.965	0.227	0.967	1.000	0.963	0.125	0.962	0.469
100	0.05	0.965	0.109	0.962	0.203	0.961	0.109	0.959	0.078	0.961	0.188
100	0.1	0.958	0.004	0.948	0.002	0.951	0.021	0.954	0.004	0.962	0.512
100	0.5	0.812	0.002	0.882	0.002	0.888	0.002	0.887	0.002	0.883	0.002
100	1	0.642	0.002	0.834	0.002	0.847	0.002	0.852	0.002	0.841	0.002

Table A.56: Segment: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.699		0.707		0.716		0.720		0.703	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.694	0.777	0.710	0.781	0.727	0.344	0.730	0.609	0.723	0.184
10	0.01	0.692	0.596	0.714	0.508	0.719	0.783	0.727	0.723	0.734	0.094
10	0.05	0.691	0.559	0.702	0.758	0.721	0.645	0.713	0.578	0.707	0.836
10	0.1	0.683	0.314	0.697	0.547	0.704	0.391	0.688	0.053	0.696	0.711
10	0.5	0.540	0.002	0.529	0.002	0.507	0.002	0.475	0.002	0.531	0.004
10	1	0.415	0.002	0.436	0.002	0.444	0.002	0.462	0.002	0.506	0.002
100	0.005	0.697	0.836	0.717	0.469	0.714	0.594	0.720	1.000	0.707	0.781
100	0.01	0.697	0.914	0.716	0.512	0.709	0.516	0.716	0.781	0.723	0.148
100	0.05	0.670	0.031	0.676	0.105	0.698	0.266	0.693	0.117	0.719	0.340
100	0.1	0.631	0.002	0.634	0.006	0.676	0.016	0.648	0.012	0.657	0.105
100	0.5	0.447	0.002	0.329	0.002	0.329	0.002	0.412	0.002	0.468	0.002
100	1	0.355	0.002	0.267	0.002	0.362	0.002	0.387	0.002	0.394	0.002

Table A.57: Vehicle: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.953		0.953		0.953		0.960		0.953	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.947	1.000	0.953	1.000	0.947	1.000	0.947	0.500	0.947	1.000
10	0.01	0.947	1.000	0.967	0.500	0.947	1.000	0.947	0.500	0.953	1.000
10	0.05	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.625	0.947	1.000
10	0.1	0.967	0.500	0.967	0.500	0.960	1.000	0.947	0.625	0.947	1.000
10	0.5	0.967	0.500	0.960	1.000	0.960	1.000	0.967	1.000	0.960	1.000
10	1	0.967	0.500	0.973	0.250	0.940	0.750	0.940	0.250	0.953	1.000
100	0.005	0.947	1.000	0.953	1.000	0.953	1.000	0.953	1.000	0.947	1.000
100	0.01	0.947	1.000	0.953	1.000	0.960	1.000	0.947	0.500	0.953	1.000
100	0.05	0.953	1.000	0.953	1.000	0.947	1.000	0.947	0.625	0.953	1.000
100	0.1	0.967	0.500	0.947	1.000	0.940	0.625	0.947	0.500	0.947	1.000
100	0.5	0.900	0.500	0.927	0.313	0.933	0.500	0.913	0.125	0.920	0.375
100	1	0.880	0.500	0.913	0.250	0.947	1.000	0.887	0.031	0.933	0.656

Table A.58: Iris: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.697		0.715		0.729		0.705		0.671	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.693	0.938	0.678	0.125	0.696	0.063	0.691	0.438	0.690	0.516
10	0.01	0.701	0.984	0.696	0.688	0.709	0.273	0.700	0.938	0.690	0.406
10	0.05	0.686	0.656	0.686	0.492	0.670	0.070	0.666	0.188	0.624	0.141
10	0.1	0.647	0.201	0.677	0.344	0.611	0.004	0.624	0.016	0.626	0.156
10	0.5	0.284	0.002	0.558	0.016	0.540	0.002	0.536	0.004	0.536	0.008
10	1	0.255	0.002	0.431	0.002	0.402	0.002	0.473	0.002	0.519	0.004
100	0.005	0.705	0.828	0.701	0.664	0.696	0.250	0.686	0.551	0.700	0.422
100	0.01	0.700	0.996	0.730	0.582	0.686	0.199	0.700	0.969	0.700	0.313
100	0.05	0.446	0.002	0.618	0.063	0.632	0.012	0.641	0.047	0.617	0.219
100	0.1	0.387	0.002	0.508	0.004	0.580	0.004	0.554	0.008	0.577	0.051
100	0.5	0.317	0.002	0.486	0.004	0.516	0.004	0.438	0.004	0.549	0.164
100	1	0.326	0.002	0.299	0.002	0.410	0.008	0.299	0.002	0.483	0.004

Table A.59: Glass: Boosted k-NN with averaged weights

Algorithm		10 fold cross validation accuracy									
		k=1		k=3		k=5		k=7		k=15	
k-NN		0.703		0.730		0.734		0.736		0.751	
Boosted k-NN with averaged weights											
T	λ	k=1	p-value	k=3	p-value	k=5	p-value	k=7	p-value	k=15	p-value
10	0.005	0.712	0.328	0.718	0.250	0.724	0.584	0.723	0.227	0.728	0.023
10	0.01	0.714	0.322	0.724	0.625	0.726	0.508	0.731	0.621	0.728	0.037
10	0.05	0.722	0.189	0.730	0.986	0.743	0.672	0.717	0.250	0.691	0.012
10	0.1	0.737	0.086	0.728	0.867	0.687	0.063	0.702	0.059	0.585	0.002
10	0.5	0.725	0.303	0.507	0.002	0.536	0.002	0.541	0.006	0.476	0.002
10	1	0.737	0.123	0.449	0.002	0.572	0.012	0.526	0.004	0.457	0.002
100	0.005	0.708	0.688	0.708	0.148	0.717	0.219	0.708	0.016	0.721	0.016
100	0.01	0.712	0.586	0.722	0.580	0.702	0.102	0.694	0.010	0.685	0.002
100	0.05	0.735	0.148	0.702	0.484	0.597	0.006	0.624	0.008	0.556	0.002
100	0.1	0.743	0.043	0.657	0.029	0.534	0.010	0.512	0.008	0.509	0.002
100	0.5	0.729	0.164	0.641	0.004	0.678	0.006	0.644	0.059	0.595	0.004
100	1	0.689	0.582	0.679	0.129	0.674	0.010	0.675	0.029	0.542	0.002

Table A.60: Diabetes: Boosted k-NN with averaged weights