Nearest Neighbor Classifiers versus Random Forests and Support Vector Machines

Saket Sathe
IBM Research AI
Yorktown Heights, NY 10598
ssathe@us.ibm.com

Charu C. Aggarwal
IBM Research AI
Yorktown Heights, NY 10598
charu@us.ibm.com

***Abstract*—Recent experimental studies have shown that the support vector machine and the random forest are "stand out" models that outperform most other supervised learning methods in benchmarked evaluations on real-world data sets. Studies have also established that both these models are closely related to nearest neighbor classifiers. Due to this connection, one would expect a similar performance from a nearest neighbor classifier; however, the latter lags greatly behind the support vector machine and the random forest in most evaluations. On the contrary, the nearest neighbor classifier has great theoretical potential because of its ability to model complex decision boundaries with low bias. Furthermore, it theoretically guarantees an error of no more than twice the Bayes error rate on an infinite data set. In this paper, we examine this obvious disconnect between the great theoretical promise and empirical reality of the nearest neighbor classifier. An additional point is that the random forest and support vector machine can be considered eager forms of the (lazy) nearest neighbor classifier. Inspired by the connections between nearest neighbor classifiers, support vector machines and random forest models, we propose a nearest neighbor classifier that is competitive to these models.**

I. Introduction

A nearest neighbor classifier computes the k-nearest neighbors of a given instance, and reports the majority class among these nearest neighbors. Nearest-neighbor methods are also referred to as *lazy learners*. This is because most of the work is done at classification time, and very little work is done during training time, unlike *eager learners*, where the model is constructed in the training phase. Nearest neighbor methods can discover complex decision boundaries and can provide theoretically bounded error with an infinite amount of data. A particularly salient result is that the error of a 1-nearest neighbor classifier is at most twice the optimal Bayes error rate with an infinite amount of data. The optimal Bayes error rate is the lowest possible error of any classifier with a particular data distribution.

In spite of this theoretical promise, nearest neighbor classifiers have not come close to achieving the empirical performance of many other classifiers, such as support vector machines (SVMs) and random forests [5]. This is because random variations in sampled training instances in a finite data set will inevitably lead to errors; a lazy learner has no understanding of which training points are important for prediction. The resulting decision boundary (approximated using finite data) has high variance and thus leads to poor generalization performance. Therefore, when working with a finite data set, it is particularly important to properly identify relevant instances and dimensions in the nearest neighbor computation process. Unfortunately, nearest neighbor classifiers are lazy learners that do not perform any of these tasks up front. Interestingly, both support vector machines and random forests can be shown to be eager variations of nearest neighbor classifiers in which specific instances or dimensions are weighted with the use of an optimization model [1]. These eager variations show excellent performance improvements over the (purely lazy) nearest neighbor learners.

Although the connections of nearest neighbors to random forests and SVMs are well known, the importance of the corresponding insights is surprisingly underemphasized in the literature. In this paper, we examine the nearest neighbor classifier from the perspective of the related SVM and random forest models. We propose eager variations of nearest neighbor classifiers that are motivated by the key ideas in SVMs and random forests. We show that the much-maligned nearest neighbor classifier is on par with the SVM and the random forest when a proper eager variant is used.

II. Weighted Nearest Neighbors, SVMs and Random Forests

Let the data points be sorted in order of their distance to the test data point $\overline{Z}$. Let w_i be the weight of the ith data point $\overline{X_i}$. Then, the predicted label $\hat{y}(\overline{Z})$ of the data point $\overline{Z}$ in the weighted nearest neighbor method is computed as follows:

$$\hat{y}(\overline{Z}) = \text{sign}\left(\sum_{i=1}^{n} y_i w_i(\overline{Z})\right). \tag{1}$$

A k-nearest neighbor method can be viewed as a special case of a weighted nearest neighbor method [6] in which the weights of the k closest examples are set to 1, and all others are set to 0. As we will see below, different choices of the weights lead to SVMs and random forest.

A. Support Vector Machines as Weighted Nearest Neighbors

Support vector machines are weighted nearest neighbor classifiers, in which the value of $w_i(\overline{Z})$ is defined as follows:

$$w_i(\overline{Z}) = \lambda_i S(\overline{X_i}, \overline{Z}). \tag{2}$$

The similarity $S(\overline{X_i}, \overline{Z})$ reduces exponentially with increasing distance as in the case of the Gaussian kernel. However, the *only* difference is that we are using a global weight $\lambda_i \geq 0$ for each point, which is learned upfront from the training data, and tells us how "important" the point is to the classification process. Training points that are closer to points of other classes generally get more weight because it is assumed that they lie closer to the decision boundary, and most of the other points get zero weight. All this is achieved by using the following optimization model:

$$\text{Maximize } \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j S(\overline{X_i}, \overline{X_j}) y_i y_j$$
$$\text{subject to: } \sum_{i=1}^{n} \lambda_i y_i = 0 \text{ and } 0 \leq \lambda_i \leq C$$

The main term in the above objective function is:

$$-\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j S(\overline{X_i}, \overline{X_j}) y_i y_j, \tag{3}$$

which tends to increase the value of λ_i for instances near other classes, and tends to set λ_i to 0 otherwise. As a result, "easy instances" in regions containing a single class often have λ_i set to 0. On the other hand, "difficult" instances near the decision boundary or misclassified training instances have positive values of λ_i. Such instances are also referred to as support vectors. At its core, an SVM performs the weighted nearest neighbors only over the support vectors, because the remaining instances have a weight of 0. Text data can usually be classified very well with an SVM by using the simple dot product as the choice of the similarity function. This corresponds to the linear SVM.

B. Random Forests as Weighted Nearest Neighbors

Random forests are methods that learn similarity in the presence of many irrelevant attributes. A random forest samples bags of attributes at nodes of a decision tree and selects the best one among them for splitting. It uses the selected attribute to recursively partition the data in the nodes, until each node contains only the instances belonging to a particular class. Typically, locally relevant subsets of attributes define the data points in each node, and therefore the approach is particularly robust when there are many irrelevant attributes. However, since the depth of the tree is usually limited, the approach does not work very well when a combination of a large number of attributes are required to isolate a class. For example, in the case of text data, there are hundreds or thousands of weakly relevant attributes (words) of a particular class, and in such a case, the random forest does not work well. Interestingly, the SVM works quite well for text. Therefore, SVMs and random forests have different types of use cases.

Random forests can also be described as nearest neighbor methods, in which two points are considered similar if they end up in the same leaf node (described by a small, relevant subset of dimensions) in many different trees. Let $I_j(\overline{X}, \overline{Y})$ be a binary 0-1 indicator function that outputs 1 when $\overline{X}$ and $\overline{Y}$ are mapped to the same leaf node in the j-th randomized decision tree from a forest containing T trees. Let $N_j(\overline{X_i})$ be the number of training instances that are contained in the same leaf node as $\overline{X_i}$ for the j-th randomized decision tree. Then the weight of the training instance $\overline{X_i}$ is defined as follows:

$$w_i(\overline{Z}) = S(\overline{X_i}, \overline{Z}) = \frac{1}{T}\sum_{j=1}^{T} \frac{I_j(\overline{Z}, \overline{X_i})}{N_j(\overline{X_i})} \tag{4}$$

By substituting Equation 4 in Equation 1, and by changing the order of summation we obtain the following prediction rule:

$$\hat{y}(\overline{Z}) = \text{sign}\left\{\frac{1}{T}\sum_{j=1}^{T}\sum_{i=1}^{n} y_i \frac{I_j(\overline{Z}, \overline{X_i})}{N_j(\overline{X_i})}\right\}. \tag{5}$$

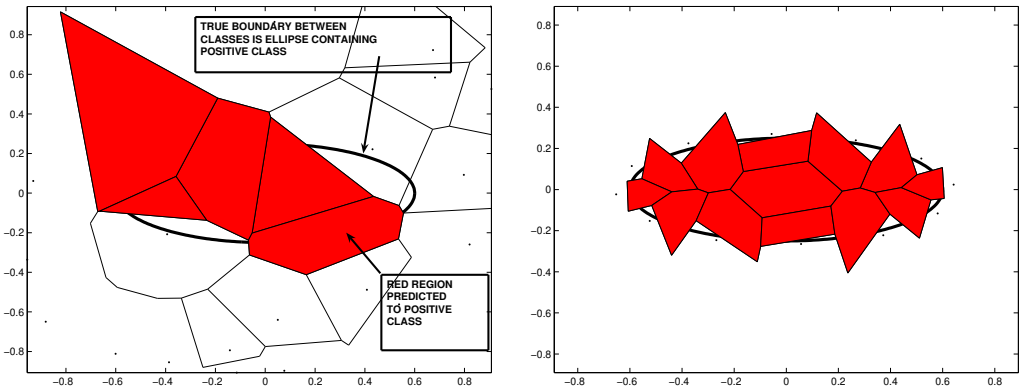
Observe that $\sum_{i=1}^{n} y_i \frac{I_j(\overline{Z}, \overline{X_i})}{N_j(\overline{X_i})}$ is the prediction obtained by an individual tree of a forest. This is because the numerator of $\frac{\sum_i y_i I_j(\overline{Z}, \overline{X_i})}{N_j(\overline{X_i})}$ is either $-p_t$ or $+p_t$ where p_t is the number of training instances in the leaf node that contains the test instance $\overline{Z}$ in the t-th tree. Furthermore, the denominator will always be p_t for all i in which $I_j(\overline{Z}, \overline{X_i})$ is non-zero. Therefore, the contribution of each tree to the prediction is either $+1$ or -1. Note that we are making the assumption (as is common in random forests) that the trees are grown until the leaf nodes contain instances only of a particular class. The final prediction of the forest is the majority vote across all trees. *Therefore, the basic structure of this prediction function is similar to that of a weighted nearest neighbor classifier as well as the SVM.*

In a recent comprehensive evaluation of classifiers [5], it was shown that the random forest and the SVM are among the most robust classifiers. In the same evaluation [5], it was shown that nearest neighbor learners perform quite poorly. A natural question arises as to whether one can directly modify nearest neighbor classifiers in a principled way to improve classification performance. This will be the focus of the following sections.

III. Nearest Neighbors like Random Forests and SVMs

We propose the Voronoi classifier, which is an adaptive nearest neighbor method, and has connections with both random forests and support vector machines. These connections explain its excellent performance.

The basic algorithm is extremely simple, and it can be implemented in a few lines of code. Just as a support vector machine isolates a few points (i.e., support vectors) in order to perform the classification, this approach also isolates a few points from the data set in order to perform classification. These points are referred to as *Voronoi anchors*, and they define a Voronoi partitioning of the data. Later, we will describe a version of the Voronoi classifier known as the *soft Voronoi classifier*, where the choice of the anchors is not restricted to the points from the data set. All test points within the Voronoi cell belonging to an anchor are classified



(a) Sample of 25 *randomly* chosen anchors (b) Sample of 25 *carefully* chosen anchors

Figure 1. [Best viewed in color] Effect of supervision on the decision boundary created by the anchors

to the same class as the training point. Note that this approach is implicitly a 1-nearest neighbor method on the anchors. However, a key difference lies in how the anchors are chosen. If the anchors are randomly chosen from the data, the predicted decision boundary will not match the true decision boundary because of the natural variability over different choices of training data sets. This situation is shown in Figure 1(a), in which 25 anchors are randomly chosen. In this case, the red region is predicted as class 2 (for which the true boundary is elliptical). Note that if 25 different random anchors are sampled, one obtains a completely different prediction. On the other hand, if the anchors are chosen in a more strategic way, as in Figure 1(b), then the predicted decision boundary matches the true decision boundary more accurately.

It is noteworthy that the anchors selected in Figure 1(b) are selected strategically along the decision boundary between the two classes. A support vector machine also tends to select *support vectors* on the margin of a decision boundary in order to maximize accuracy. However, a problem is that misclassified points and noise are usually included among support vectors, which has a detrimental effect on accuracy. In the case of the Voronoi classifier, the points are chosen so as to maximize accuracy on the training data. Another difference is that an SVM picks these points and creates a relatively stable classifier that does not significantly improve in accuracy by averaging the predictions over different samples of the training data. However, the Voronoi classifier, similar to a random forest, allows overfitting in each ensemble component, but the overall accuracy is enhanced due to variance reduction obtained by combining the predictions of multiple ensemble components. In this sense, even though the Voronoi classifier tends to use points on the natural boundaries between classes, on average it tends not to include misclassified points or noise in the selected sample.

A. The Voronoi Classifier

The basic algorithm for the Voronoi classifier (denoted as VC) is fairly simple, and we will first discuss the process of creating a single component of this classifier. As we will discuss later, this classifier is closely related to all three: nearest neighbor classifier, SVM, and random forest. In the following, it will be assumed that the data set contains d feature variables and n points.

Let $f \in (0, 1)$ be a parameter that is chosen by the user and is an input to the classification[1] algorithm. This parameter is referred to as the *conciseness* parameter, which regulates the trade-off between bias and variance for the model, just as the choice of the Gaussian kernel bandwidth in an SVM also regulates the trade-off between bias and variance. Small values of f lead to concise models with low variance but high bias, and vice versa. Each individual component of VC attempts to find a set A consisting of $\lfloor f \cdot n \rfloor$ anchors from the data, and the anchors naturally define a Voronoi partitioning of the data. Any test instance that lies in a particular region is classified to the same label as the anchor that defines the region. These anchors are chosen carefully by the learning algorithm in order to maximize accuracy, just as support vectors are chosen by SVMs to maximize accuracy. However, the criteria used to select the anchors is quite different from what is used in a support vector machine.

Rather than using the approach of finding the points in the margin region, the Voronoi classifier directly chooses anchors to maximize accuracy. Although this will often result in overfitting, the main advantage of the Voronoi classifier lies in its ability to reduce this effect by using multiple ensemble components like a random forest. However, the approach is different from a random forest because the data points are partitioned with an emphasis on optimizing anchor choice in order to create a good Voronoi partitioning.

A single component of the Voronoi classifier is presented in Algorithm 1. The idea of this approach is to capture a *concise set* of anchor points (regulated by the parameter f), and each Voronoi region is assigned to the class of its anchor point. The approach works iteratively, in which we start with a random set of anchors and continually interchange non-anchors with anchors in order to improve classification accuracy on the training data. The candidate points $\overline{X}$ to

[1]As in all supervised learning algorithms, such parameters are chosen by cross-validation.

Algorithm 1 Single component of the Voronoi classifier.

Input: Data points and labels: $\mathcal{D}$, Sample size: s, Conciseness: f, Maximum number of iterations: max_iter,
Output: Anchor points A

$S, y_s =$ Sample of s points and their labels from $\mathcal{D}$
Initialize A to $r = \lfloor f \cdot n \rfloor$ random points from S
Predict label of each data point in S as the label of its closest anchor in A. Let these predicted labels be y'_s
Compute accuracy $\alpha = \text{accuracy}(y_s, \overline{y}_s)$
for $0 \leq iter < max\_iter$ **do**
    Select a misclassified point $\overline{X}$ from training sample S ▷ (Misclassified points like $\overline{X}$ are non-anchors)
    Select a random anchor point $\overline{Y}$ in A
    Predict labels $\overline{y}_s$ using anchor set $\{A - \{\overline{Y}\}\} \cup \{\overline{X}\}$
    if accuracy$(y_s, \overline{y}_s) > \alpha$ **then**
        $A \Leftarrow \{A - \{\overline{Y}\}\} \cup \{\overline{X}\}$, $\alpha \Leftarrow$ accuracy$(y_s, \overline{y}_s)$
    end if
    if no improvement in L consecutive tries **then**
        break
    end if
end for
return A

be considered anchors are always misclassified points in the training data, under the assumption that their locality is not properly represented in the current anchor set. Since anchors are always assigned to themselves, misclassified candidate points like $\overline{X}$ are guaranteed to be non-anchors. We first test whether replacing a randomly chosen anchor $\overline{Y} \in A$ with the misclassified non-anchor candidate $\overline{X}$ leads to an improvement in accuracy. If this is indeed the case, then the switch is made permanent. This interchange approach is repeated to convergence or until a maximum number of iterations are reached. Convergence is declared if classification accuracy does not improve in L consecutive iterations.

In essence, this approach amounts to using 1-nearest neighbor on the *concise sample that is learned in a supervised way*. Therefore, the approach is an eager nearest neighbor learner. Since the focus is on reducing the misclassification of training examples, it is possible for each ensemble component to overfit the data much like a single tree in a random forest. The conciseness parameter f also has an impact on the level of overfitting and the shape of the decision boundary. Larger values of f can lead to a greater level of nonlinearity in the decision boundary. As we will discuss later, the lowest (reasonable) choice of f leads to a *linear* decision boundary between classes much like a linear SVM.

B. The Soft Voronoi Classifier

The VC approach follows a "hard" replacement strategy, wherein the anchors A have to be from the sample set S. The problem with this setting arises especially when we have limited data. This is because a good anchor point may not be present in the sample set S. Therefore, we propose another version of the Voronoi classifier known as the soft Voronoi classifier (denoted as SVC). In the SVC classifier an accurate set of anchors is learned on-the-fly. These points may not be from the set S. Similar to Algorithm 1, suppose $\overline{X}$ is a misclassified point and we are evaluating whether replacing an anchor point $\overline{Y}$ with $\overline{X}$ will improve accuracy. The VC algorithm directly interchanges $\overline{Y}$ with $\overline{X}$ and checks if accuracy improves. On the contrary, the SVC approach computes a new point $\overline{Z}$, which is a random linear combination of $\overline{Y}$ and $\overline{X}$ as follows:

$$\overline{Z} = \beta\overline{X} + (1 - \beta)\overline{Y}. \tag{6}$$

Here, β is a uniform random number between 0 and 1. The point $\overline{Z}$ is assigned $\overline{X}$'s label if $\beta \geq 0.5$, otherwise it is assigned $\overline{Y}$'s label. $\overline{Y}$ is then replaced with $\overline{Z}$ in the anchor set A; this replacement is made permanent only if it improves accuracy. As the name suggests, the SVC is a "soft" replacement version of VC. In the SVC approach, the decision boundary is gradually expanded (or contracted) as several values of β are tried. In the experimental section, we show that SVC is superior to VC in terms of its accuracy on many data sets. This superior performance can be attributed to SVC's tendency to overfit more as compared to VC. Recall that when the ensemble components overfit the data (similar to a decision tree in a random forest), the overall performance of the ensemble is superior due to variance reduction obtained from averaging (or majority voting) the predictions.

Lastly, we train T ensemble components of either VC or SVC. After all the models have been constructed, a test instance is classified by the ith ensemble component (for each $i \in \{1, \ldots, T\}$), by determining the label of its closest neighbor in A_i. Then the dominant vote among all the T classifications is returned as the relevant label.

C. Efficient Implementation

A tricky issue is that when an anchor $\overline{Y}$ is to be replaced with a non-anchor $\overline{X}$, the new anchor $\overline{X}$ may attract points assigned to other anchors. How should one reassign? Observe that the distance of each point to $\overline{X}$ could be computed to check whether or not the re-assignment should be done. Furthermore, all points assigned to the original anchor $\overline{Y}$ may need re-assignment, and therefore one has to compute the distances of these points to other anchors. Although it is possible in principle for $O(s)$ points to be assigned to $\overline{Y}$, this is not the case in practice. One way of making this approach efficient is to keep track for each point in S which is its closet point in A, and the distance of each point in S to its closest anchor point. Next, when we want to add $\overline{X}$ to the anchor set. First, we compute the distance between $\overline{X}$ and all points in S. Then for each point $\overline{Z}$ in S we check whether $\overline{X}$ is closer than $\overline{Z}$'s current nearest neighbor. If it is found to be closer, then we update $\overline{X}$ as $\overline{Z}$'s nearest neighbor along with the distance to $\overline{X}$ as its current nearest neighbor distance. This type of approach often requires $\theta(s)$ *empirical* time per iteration for values of f that are within a constant factor of s. In practice, the value of s is quite small ($\lfloor 0.20 \times n \rfloor$) and the values of f are often chosen around 0.1 to 0.2.

D. Termination and Convergence

Each improvement in the objective function's value reduces the misclassified points by at least one, and the total number of

such improvements is at most equal to the number of training points s in the sample. Furthermore, at least one improvement must occur in L consecutive iterations. This means that the algorithm is guaranteed to terminate in $O(L \cdot s)$ iterations. The value of L is typically chosen to be a small value such as 50. We have often noticed that even if we set the maximum number of iterations (max_iter) to small values (such as 200), an ensemble component rarely needs all the iterations to converge.

E. Effect of Conciseness on Bias-Variance Trade-off

One interesting aspect of this method is that the conciseness parameter f has a crucial effect on the shape of the discovered decision boundary, just as the choice of the Gaussian kernel bandwidth influences the shape of the decision boundary in a kernel SVM. It is noteworthy that linear SVMs can have as few as two support vectors, whereas Gaussian kernels with small bandwidth could contain all points as support vectors. In this sense, the anchor set can be viewed as the analog of the set of support vectors discovered by a support vector machine.

If $(f \cdot n)$ is 1, then the classifier becomes a simple majority voting classifier. In other words, all points will be voted to the majority class in the sample. This is a highly biased classifier with very little variance in predictions across multiple samples. Increasing $(f \cdot n)$ to 2 creates a linear decision boundary that is perpendicular to the line joining the two anchors. By using the hill-climbing approach, one essentially chooses the orientation of the hyperplane to optimize the accuracy, much like *linear* SVMs (albeit with a very different loss function). Rather than choosing the distance from the margins as the loss, the approach directly optimizes the accuracy of the discovered hyperplane. Unlike an SVM, the Voronoi classifier is not heavily influenced by strongly misclassified points or mislabeled outliers in the data.

When the value of $(f \cdot n)$ is larger than 2 but less than n, the approach constructs piecewise linear boundaries in the data space. It is noteworthy that the random forest also constructs piecewise linear boundaries, although each tree is constructed quite myopically with a split along a single attribute, whereas the Voronoi classifier has a more refined way of partitioning the data space with a hill-climbing approach. When the value of $(f \cdot n)$ is chosen to be n (i.e., $f = 1$), the approach gives 100% accuracy on the training data due to overfitting, since each point in S is assigned to itself. However, the performance of the ensemble is still reasonably good because it is at least as good as a bagged/subsampled 1-nearest neighbor learner with sample size s. Therefore, the approach is a flexible technique, where different values of the conciseness parameter provide different trade-offs between bias and variance, similar to tuning the kernel bandwidth in a SVM.

IV. Experimental Results

In this section, we present extensive experimental results comparing SVC and VC with other well-known classifiers. We use classification data sets for all the experiments in this section. We normalize our data sets to zero mean and unit variance. All of our data sets are obtained from either the UCI Machine Learning Repository[2], LibSVM repository[3] or from the data sets shared by the authors of [5].

Table I
Accuracy comparison (%) among various classifiers and data sets. The top-2 methods are shown in boldface.

Data set	SVC	VC	RF	SVM	KNN
Parkinsons	**71.79**	**71.79**	71.54	63.08	**73.85**
Congress-Voting	**60.29**	58.20	59.77	**58.39**	54.92
Breast-Cancer	**96.93**	96.86	**97.58**	95.97	95.40
Annealing	**92.22**	92.10	**95.97**	91.97	92.00
Optical	97.65	97.63	**97.88**	**98.37**	97.65
a1a	**83.17**	82.61	**83.04**	82.78	80.65

A. Evaluation Strategy

Before performing experiments we split the data set into a 80-20 split. This split is consistently performed such that each algorithm is trained and tested on the same data set. In experiments where cross-validation is performed, we keep a further 10% of the data for cross-validation. Here too we maintain consistency: each algorithm gets the same cross-validation data set for tuning the parameters. We measure model performance using *accuracy*. Accuracy is measured as the percentage of correctly classified test data points. We compare the proposed methods, SVC and VC, with random forests (RF) [2], kernel support vector machines (SVM) [3] and k-nearest neighbor methods (KNN) [4]. For the competitors we use an existing implementation available in the popular SciKit-learn package[4].

B. Accuracy Analysis

In this section we show by way of extensive experiments that SVC and VC are more accurate than the competitors. For all the experiments in this section, we train the classifiers using cross validation for 20 iterations and report the mean accuracy. This is done for obtaining a robust measurement of accuracy. In each iteration the parameters for the various algorithms are tuned using cross validation as follows.

In all our experiments we use SVMs with the Gaussian kernel. The penalty parameter C is varied between $\log(-3)$ to $\log(3)$ in four steps of equal size and the kernel coefficient (which controls the kernel bandwidth and is popularly denoted as γ) is varied between 0.001-1 in four geometric steps with a multiplicative factor of 10. Random forests were tuned by varying the maximum number of features (referred to as $max\_features$) used at each node of a RF decision tree for splitting the data. The recommended value for $max\_features$ is $\sqrt{d}$ for a d-dimensional data set. We further fine tune $max\_features$ from $0.5 \cdot \sqrt{d}$ to $1.5 \cdot \sqrt{d}$ in four equal steps. Lastly, for the KNN classifier we tune the number of nearest neighbors (k) by executing the classifier for $k = \{2, 4, 6, 8\}$. For both SVC and VC the parameter s is not tuned and is

[2]http://archive.ics.uci.edu/ml/index.php
[3]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
[4]http://scikit-learn.org/

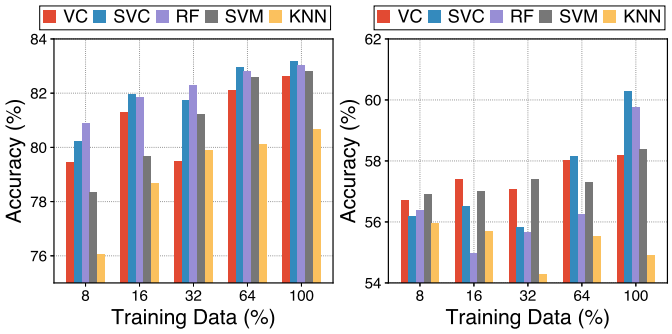
always fixed at 20% of the data size. The rational behind fixing s stems from the fact that a minimum number of points are needed to enable the computation of a reasonable decision boundary. On the other hand, if s is too high then the predictions of the ensemble components become correlated, which typically hurts the overall performance of any ensemble-centric method. Initially, during our experiments we tried to tune s using cross validation but obtained only minimal gains. Therefore, we choose to follow a compromise strategy of fixing s to a reasonable value. The conciseness parameter f is varied between 4 and $\lfloor s \times n \rfloor$ in three linear steps. In each pass SVC and VC are executed for 200 iterations and the processing is discontinued if no progress is made in 50 consecutive iterations. We use 300 ensemble components for SVC, VC, and RF. The best set of parameters found during cross-validation are retained and the test accuracy is based on this optimized setting.

The comparison of accuracy obtained on various data sets is shown in Table I. Clearly, SVC is able to consistently beat the competitors on most of the data sets. When it comes second, it is often very close to the other methods. It is instructive to note that adding a minor form of supervision has such a catalyzing effect on the accuracy of nearest neighbor methods. Observe that the KNN method performs poorly as compared to RF and SVM, which is a well-known fact. However, with the modifications that we have proposed to the KNN approach (i.e., the Voronoi based methods SVC and VC) the improvements in performance are phenomenal. This experiment further establishes what we discussed in the previous sections regarding nearest neighbor methods; if modified appropriately, these methods can be used for highly accurate classification. Secondly, nearest neighbor methods have a tendency to overfit the data. This makes them a good candidate for use in ensemble-centric approaches (such as SVC and VC). This is because when predictions from a sufficiently large number of such ensemble components are averaged, it is possible to capture extremely complex decision boundaries, which cannot be modeled by each of the underlying ensemble components.

C. Effect of Increasing Data Size

We investigate the effect of varying the training data size on SVC, VC and the other competitors. A key challenge with the nearest neighbor methods is that they need large amounts of data for estimating an accurate decision boundary (and exhibit Bayes optimal behavior for infinite data). On the other hand, random forests and SVMs perform much better, because they use supervision up front to (implicitly) focus on important points or important attributes. In the following experiments, we will show that SVC and VC also have such favorable characteristics despite being nearest neighbor methods.

In Figure 2, we show the performance of all the classifiers for increasing data set size. For each classifier, we perform at least 20 trials for all the data sizes and report the mean performance at all levels. For lower data set sizes (i.e., when $\leq$ 32% data is used) we perform at least 40 trials to obtain a much more stable estimate of accuracy. In each of the trials, cross-validation is first used to estimate the optimal value of the parameters. The parameter ranges for the various algorithms are similar to the ones mentioned in Section IV-B. Observe that the proposed SVC and VC methods perform far better with increasing amount of data. Between SVC and VC, SVC has favorable performance as a function of data set size. This is because the decision boundaries produced by SVC are more accurate for even a small amount of data, since anchors are not restricted to the points form the data set. Overall, for the SVC method, the average improvement in accuracy obtained over RF, SVM, and KNN is 0.59%, 0.67% and 3.78%, respectively. This reinforces the fact that the proposed supervised nearest neighbor methods can overcome the challenges associated with limited data availability.



(a) A1A (b) CONGRESS-VOTING

Figure 2. [Best viewed in color] Comparing the performance of SVC and VC with the other classifiers as the data set size increases. Note the scale on the x-axis.

V. CONCLUSION

This paper explores the relationship between SVMs, random forests, and nearest neighbor classifiers. Although nearest neighbor classifiers generally lag the performance of SVMs and random forests, this paper shows that it is possible for eager versions of nearest neighbor classifiers to match and even outperform random forests or SVMs. This follows directly from the fact that SVMs and random forests are themselves eager variations of nearest neighbor classifiers.

REFERENCES

[1] C. Aggarwal. Machine learning for text. *Springer*, 2018.

[2] L. Breiman. Random Forests. *Journal Machine Learning archive*, 45(1), pp. 5–32, 2001.

[3] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3), 273-297, 1995.

[4] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27, 1967.

[5] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?. *The Journal of Machine Learning Research*, 15(1), pp. 3133–3181, 2014.

[6] R. Samworth. Optimal weighted nearest neighbour classifiers. *The Annals of Statistics*, 40(5), pp. 2733–2763, 2012.