The Utility of Merging k-Nearest Neighbor Classifier and Classification Tree

Tor Nitayanont TOR_N@BERKELEY.EDU
Department of Industrial Engineering and Operations Research

Abstract

Classifiers that rely on pairwise similarity such as k-nearest neighbor classifier often struggle with computation time issue, especially when the data is high-dimensional. Their performances are also dependent on the quality of distance metrics that capture the relationship between samples. In this work, we introduce the use of classification tree into the framework of k-nearest neighbor classifier to handle both issues. We use the tree to partition samples into groups before applying the k-nearest neighbor classifier. This pre-processing step can reduce the computation time of the classifier by a significant amount. Moreover, we leverage the classification tree further by using the information about the feature importance from the tree to develop a better distance metric that results in accuracy improvement of the k-nearest neighbor classifier.

1. Introduction

k-nearest neighbor classifier or kNN is among popular machine learning methods that are both simple in terms of interpretation and implementation, and often yield impressive results. kNN does not have many hyperparameters. It can capture nonlinear decision boundaries. However, kNN classifier still suffer from several pitfalls.

First, kNN often has a huge computational cost due to its nature as a method that relies on pairwise similarity computation. Hence, the runtime is quadratic in the number of samples. Techniques like kd-tree and ball tree have been introduced and improved the computation time significantly. However, they cannot be applied to some custom distance metrics, and there are still several cases where the tree data structure fails to deliver notable time reduction. The computation time does not only increase with the size of the dataset but also with the dimension of the data. Distance computation between two samples requires more arithmetic operation in high-dimensional data.

Another point about the disadvantage, or to be more precise, the sensitivity of kNN classifier is the choice of distance metric. The issue of choosing a proper distance metric such that the distance reflects the similarity accurately has been studied and still active as a separate research field such as the works done by Yang and Jin (2006), Chomboon et al. (2015) and Abu Alfeilat et al. (2019). Different datasets from different domains work well with different distance metrics. One of the common distance metrics that are widely used is the Euclidean distance (and any functions of Euclidean distance). However, the Euclidean distance gives equal weights to all features, which is not necessarily the most appropriate approach. In this case, Mahalanobis distance can be introduced to adjust the weights of the features and also take into account the correlations.

In this paper, we propose a solution to the addressed issues of kNN classifier by bringing in the classification tree or CART, which is another well-known machine learning model, which is also very intuitive. The proposed model is called *kNN-tree classifier*. In the kNN-tree classifier, We use CART as a preprocessing step to partition data into smaller subsets so that the computation time of the neighbor search is reduced. Note that CART itself runs quickly and does not add much time to the entire algorithm. Moreover, we utilize the feature importance computed in the construction of the tree to compute the distance between samples in the way that reduces the distance computation time between two samples.

Most importantly, the tree classifier and the kNN classifier have some properties regarding where they work well that are complementary to each other. The kNN classifier is capable of detecting nonlinear boundaries between samples with different labels whereas the tree classifier partitions samples according to a single feature at a time, unless feature engineering is performed but this is a different problem. The kNN classifier is also more competent at local levels when we examine a small number of samples that are close to one another whereas the tree classifer is effective when we attempt to separate samples at higher levels, or when we have a large number of samples all over the place. The contrasting yet supportive behaviors of the two baseline models motivate their integration in this work.

The description of the proposed algorithm is discussed in section 3 after the overview of the baseline models in section 2. After that, in section 4, we perform the experiment by using the proposed model to solve binary classification problem on several datasets.

This work is relevant to topics covered in this class such as generalization and data representation. We adopt k-fold cross validation technique to tune parameters so that the models are generalized to the unseen data. Moreover, we use the tree structure to represent the data.

2. Brief overview of the baseline models

In this paper, we will only discuss our model and all relevant subjects in the context of binary classification. Suppose we have n labeled samples: $X = \{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^d$ and their labels $Y = \{y_1, y_2, ..., y_d\} \subseteq \{0, 1\}^n$. We would like to construct a classifier F, based on the given X and Y as training data, that maximizes the accuracy of the predicted labels of some unlabeled samples or testing samples.

2.1 k-Nearest Neighbor Classifier

Given a distance metric $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and the number of neighbors k, $I^{knn}(v)$ is the set of indices of k nearest neighbors (among X) of a given query v if

$$|X^{knn}| = k \text{ and } d(x_i, v) \leq d(x_j, v) \text{ for all } i \in I^{knn}(v) \text{ and } j \in [n] \setminus I^{knn}(v)$$

Then, we can assign a label to the given query based on some rule such as the majority voting rule:

$$\widehat{y}(v) = \mathbb{1}\{ \sum_{i \in I^{knn}(v)} \mathbb{1}\{x_i = 1\} \geq \sum_{i \in I^{knn}(v)} \mathbb{1}\{x_i = 0\}\}$$

Alternatively, we can use the "weighted" voting rule as follows:

$$\widehat{y}(v) = \mathbb{1}\{\sum_{i \in I^{knn}(v)} w(x_i, v)\mathbb{1}\{x_i = 1\} \geq \sum_{i \in I^{knn}(v)} w(x_i, v)\mathbb{1}\{x_i = 0\}\}$$

2.2 Classification Tree

Classification tree partitions the data on each dimension (or feature) at each step. Given an impurity function $Imp(\cdot)$ tat measures the heterogeneity of a set, the tree finds the feature p and the threshold value θ such that the split performs on the data at that node using p and θ results in the maximum impurity reduction. For the ease of notation, suppose all features are non-categorical and we are at a particular node in the tree with samples denoted as X^C, we want to find the best split:

$$p^*, \theta^* = \arg\max_{p,\theta} |X^C| \cdot Imp(X^C) - |X^L| \cdot Imp(X^L) - |X^R| \cdot Imp(X^R)$$

where the sample set in the left node $X^L = \{x_i \in X^{current} | x_{i,p} < \theta\}$, the sample in the right node $X^R = \{x_i \in X^C | x_{i,p} \geq \theta\}$ and $|\cdot|$ is the set cardinality.

Regarding the impurity function, there are several options such as the Gini impurity and the entropy function. In this work, we use the Gini impurity function: $Gini(X) = p_0(1 - p_0) + p_1(1 - p_1)$ where p_i denotes the fraction of samples of class i in X. A smaller Gini impurity implies better homogeneity.

For a fully grown classification tree, we repeat this process on each branch of the tree. A node becomes a leaf, on which we no longer perform a split, if the leaf only contains samples with the same label. However, full depth trees often overfit on the training data. We will use a regularized classification tree in our work where we stop branching out the tree after it reaches some depth or if the number of samples in the nodes fall below some threshold.

After the tree construction is completed, we take a test sample and see which leaf node it falls into. To label this test sample, there are multiple ways but the most common method is to take the majority label of the training samples in the leaf. Alternatively, we can compute the probability that the test sample belongs to each class by computing the fraction of samples from each class in that leaf node, instead of assigning a binary label.

2.2.1 Feature Importance

One key value (or set of values) computed in a decision tree is the feature importance. Conceptually, it quantifies the power of each feature in determining the labels of the samples. The computation of the feature importance of each feature is done by the (weighted) impurity reduction brought by that feature over all nodes in the tree that split on the feature. We denote the feature importance of feature f by $FI(f)$, the set of non-leaf nodes in the tree by $V_{\text{non-leaf}}$, the set of non-leaf nodes that split on f by $V_{\text{non-leaf}}(f)$ and the impurity reduction of the split at node v (into the left node v_L and the right node v_R) by $ImpRed(v)$.

n represents the number of samples.

$$\begin{aligned} FI(f) &= \sum_{v \in V_{\text{non-leaf}}(f)} WeightedImpRed(v) \\ &= \sum_{v \in V_{\text{non-leaf}}(f)} \frac{|v|}{n} \cdot Imp(v) - \frac{|v_L|}{n} \cdot Imp(v_L) - \frac{|v_R|}{n} \cdot Imp(v_R) \end{aligned}$$

This impurity-based feature importance is known to be biased in favor of categorical variables with high cardinality Breiman (2001).

Now that we have explained the two baseline models to the extent that is necessary in this work, we proceed to the description of the proposed kNN-tree classifier.

3. kNN-tree Classifier

The process of the kNN-tree model can be listed out in several steps as follow:

1. First, we build a regularized classification tree. The tree is regularized by our choice of the minimum number of samples in leaf nodes. This threshold is a tunable parameter.
2. Within each leaf, we build a kNN classifier using the training samples in that leaf.
3. For each testing sample, we find the leaf node that the testing sample falls in, and use the kNN classifier in that leaf node to classify the testing sample.

In last section, we attribute the need to regularize the tree to the inclination of the tree to overfit. It is important to add here that we regularize the tree not only because we want to avoid overfitting but also because we want the kNN classifier to do its job in the leaf node. The kNN classifier could prove useful if we maintain some number of samples in the node. In this project, we would like to examine the mutual benefits of the two baseline models as much as possible.

Additionally, We also introduce a variant of kNN-tree classifier that relies on the computation of the impurity-based feature importance of the built classification tree. We call this variant *kNN-Tree-FI*. In the kNN-Tree-FI classifier, we follow the same procedure as listed above. The only diffference lies in the distance computation in the kNN step. In kNN-tree classifier, we simply compute the Euclidean distance between samples. However, in kNN-Tree-FI, we weigh each feature in the Euclidean distance computation by its importance.

$$Dist(x, y) = \sqrt{\sum_{f=1}^{m} FI(f) \cdot (x_f - y_f)^2}$$

4. Experiments

We will test our proposed model, k-NN-Tree classifier, and its variant kNN-Tree-FI classifier, along with the baseline models on binary classification tasks. The datasets that we use include both real datasets and synthetic datasets.

4.1 Data

The synthetic datasets are generated by the *make_classification* function from Scikit-Learn developed by Pedregosa et al. (2011). We generated three sets of datasets. Each set contains 20 datasets. In all 60 datasets, we have 10000 samples and 50 features. The samples are split into positive and negative classes equally. The difference between the three sets of datasets is that the number of features that are relevant vary from 25% or 13 features, 50% or 25 features to 75% or 37 features. The motivation behind this variation is that we would like to see the performances of our models with the existence of noisy features (features that are irrelevant ot the labels.)

In each dataset, we sample 70% of the data to be training samples, and the remaining 30% are the testing samples.

4.2 Parameters

There are several parameters in our models that need to be tuned.

- Minimum fraction of samples required at leaf nodes: $0.001, 0.002, 0.005, 0.01$
  If the fraction of samples in a node falls below this parameter, we go back to its parental node and make it a leaf node.
- Impurity function: Gini impurity
- Number of neighbors in kNN classifier: $4, 8, 12, 16, 20$
- Distance function: Euclidean distance
- Weight in kNN: Inverse of distance $w(u, v) = \frac{1}{d(u,v)}$

To tune the parameters, we perform 5-fold stratified cross validation.

4.3 Models

There are 5 models that we include in our experiment: 1) the classification tree 2) the k-nearest neighbor classifier or kNN 3) the k-nearest neighbor classifier with feature importance-weighted distance or kNN-FI 4) the proposed kNN-Tree classifier and 5) the variant of the proposed classifier, which is the kNN-Tree-FI classifier.

4.4 Results

4.4.1 Accuracy

In this section, we will refer to the models by their abbreviations. We examine both accuracy and run time of the models. First, we look at the histogram of the accuracy. In each of the three plots in figure 1, we have 5 histograms, one for each classifier, that reflects the distribution of the accuracy of the classifier on 20 datasets generated as explained earlier.

From the first plot of figure 1 where most of the features are irrelevant and only 13 out of 50 features are relevant, we can see that the models that rely on feature importance computed from CART, achieve the highest performance. The gap between the best model, which is kNN-FI and the second best, which is kNN-Tree-FI, is quite noticeable. Among

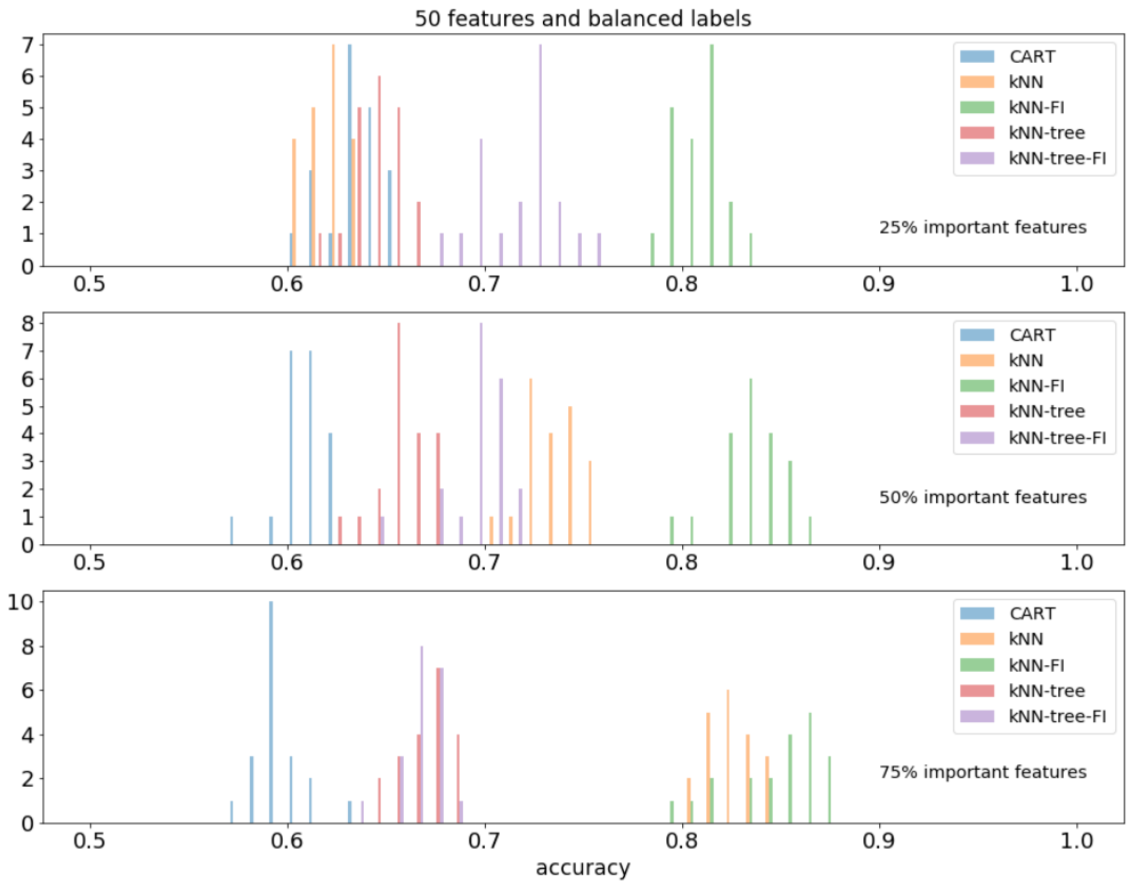

Figure 1: Histograms of the models' accuracy for different number of relevant features

the three models with lower accuracy, kNN-tree is the best performing classifier with a small margin over the other two. The baseline models, which are CART and kNN, have the lowest accuracy. This result shows that with the existence of many noisy features, all of our proposed models (kNN-Tree, kNN-Tree-FI and kNN-FI) are able to improve the classification performance.

However, in the second plot when more features are relevant (fewer noisy features), we can see that kNN-Tree-FI does not maintain the accuracy level. On the contrary, kNN-Tree and especially kNN gain some accuracy improvement.

Finally, in the last plot where most of the features are relevant, kNN and kNN-Fi have roughly the same performance. Both of them have much higher accuracy than kNN-Tree and kNN-Tree-FI, which perform slightly better than CART.

4.4.2 Computation time

Next, we examine the computation time of the models. First of all, we need to note that the computation times reported in the plots only capture the time of the prediction phase

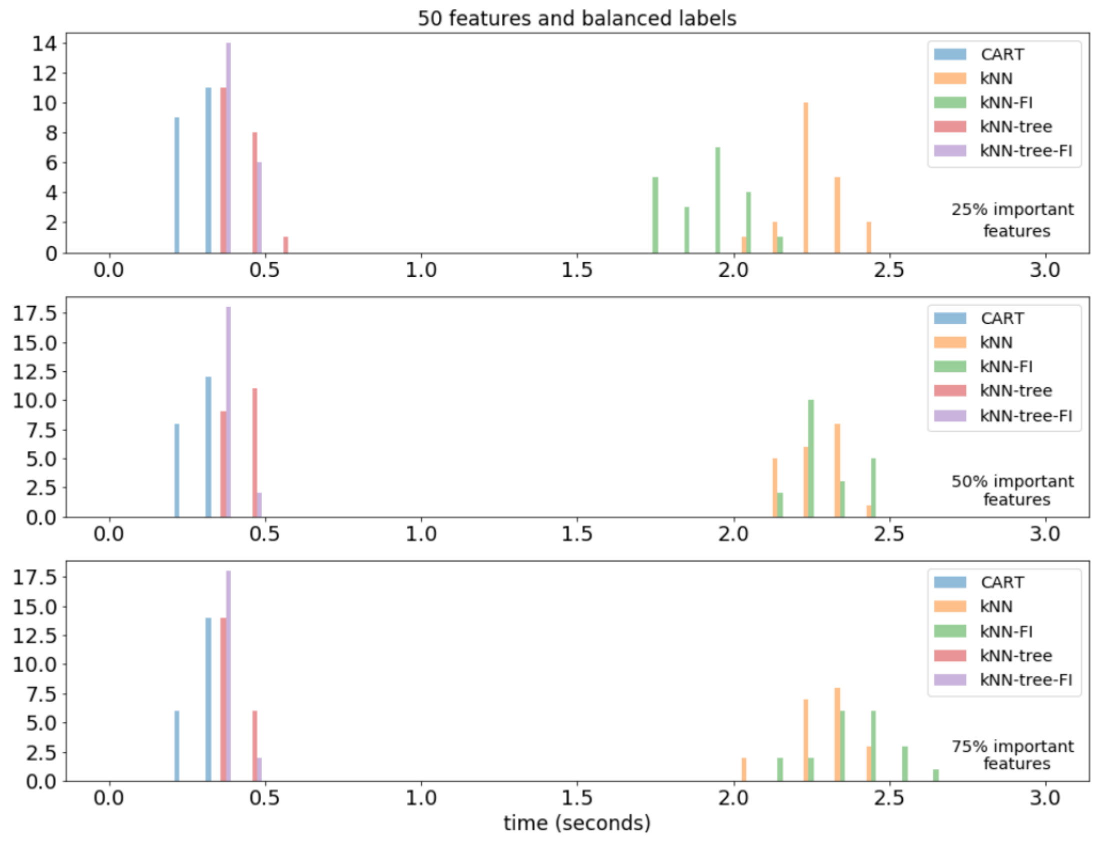

Figure 2: Histograms of the models' computation time for different number of relevant features

but exclude the time spent on hyperparameter tuning. The results are consistent across all noise levels in the features.

Models that rely on the structure of classification trees, which include CART, kNN-Tree and kNN-Tree-FI, take much smaller time than kNN and kNN-FI classifier. The computation cost of CART is slightly smaller than kNN-Tree and kNN-Tree-FI. Comparing the two models that are more computationally expensive, the runtime of kNN classifier is more stable over all plots than kNN-FI. kNN-FI runs more slowly as the number of relevant features increases.

4.4.3 Analysis

First, we will analyze the accuracy results. Before that, we need to discuss the feature importance computed by CART. Since the computation of the importance of each feature is done at the splits where the feature is the criterion, the number of features with computed importance is at most the number of splits in the tree. Moreover, it is usually the case that some features are used at multiple branches of tree. Therefore, the number of features with computed importance can be quite small, and smaller than the actual number of relevant features. In other words, the feature importance array can be quite sparse.

This phenomenon is reflected through the accuracy drop of kNN-Tree-FI. We can see that kNN-Tree-FI is competitive when the number of relevant features is small. It is likely that CART's feature importance can capture all the important features, whereas the feature importance of other features remain zero. As the number of relevant features increase, the sparsity of feature importance vector remains high. There are some features that are left behind. Hence, the distance computed based on feature importance does not represent the similarity between samples accurately. The histogram of the accuracy of kNN-Tree-FI then moves to the lower side of the plot.

On the other hand, the performance of kNN improves when more features become relevant to the labels of the data. This change is also driven by the relevance of the features. Since the Euclidean distance takes all features into consideration when computing distance, the computed distance can be misleading when most features are in fact irrelevant. This error becomes weaker when most features determine the sample labels.

Regarding the computation time, the results are expected. The tree partitioning before applying kNN help reduce the runtime by a significant factor. Both kNN-Tree and kNN-Tree-FI take much less time than their non-tree counterparts. The main reason is that the numbers of pairwise distances to be computed in kNN-Tree and kNN-Tree-FI are considerably fewer than those in kNN and kNN-FI.

Regarding the computation time of kNN and kNN-FI, their difference also comes from the sparsity of feature importance vector. When only a few features are relevant, the feature importance of many features are zero, making the distance computation faster. As the sparsity becomes smaller, the computation time of kNN-FI increases and converges to that of the traditional kNN.

5. Conclusion

The tree structure can reduce the runtime of k-nearest neighbor classifier significantly. The k-nearest neighbor classifier with tree partitioning only takes a slightly longer computation time than the classification tree. At the same time, the information about the feature importance can enhance of the performance of the classifier, especially with the existence of noisy features. If we focus on the proposed kNN-tree model with feature importance-weighted distance computation, we can see that it is quite competent when few features are relevant. Even though the accuracy yielded by the model is not as high as that of the k-nearest neighbor classifier with feature importance-weighted distance, the time reduction is significant. Therefore, it could be worthwhile to apply the proposed classifier when working on data from domains that are high-dimensional or domains that are known to have noisy information.

6. Future Works

There are many directions that we can extend our work. One of them is to apply the similar idea to ensemble methods like random forest. Most of the time, random forest can enhance the performance of a single classification tree significantly. Shallower trees that rely on subsets of features and bootstrapped data help prevent overfitting. This aspect of random forest also fix one drawback of the classification tree, which is that one bad split close to the root can ruin the rest of the tree. Some features are never used as splits at all. In random forest, most of the features, if they are at least of some significance, would likely be tested. Hence, the feature importance measured by random forest would be more accurate than that given by a single classification tree. The accuracy of feature importance is crucial since we use it to compute the distance between samples, which are the key ingredient of our methods which rely heavily on similarities or dissimilarities between samples.

Another direction of work that we can focus on is to modify the tree. The tree structure can remain helpful in terms of the reduction in computation time of the k-nearest neighbor classifier. However, at each split in the tree, we do not necessarily select the feature of which split results in the maximum impurity decrease. The purpose of the maximal impurity reduction is to make the leaf nodes as homogeneous as possible. In our case, we have a k-nearest neighbor classifier within each leaf. Instead of having homogeneous leaves, we might prefer leaves in which samples of different labels are well separated.

References

Haneen Arafat Abu Alfeilat, Ahmad BA Hassanat, Omar Lasassmeh, Ahmad S Tarawneh, Mahmoud Bashir Alhasanat, Hamzeh S Eyal Salman, and VB Surya Prasath. Effects of distance measure choice on k-nearest neighbor classifier performance: a review. *Big data*, 7(4):221–248, 2019.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Kittipong Chomboon, Pasapitch Chujai, Pongsakorn Teerarassamee, Kittisak Kerdprasop, and Nittaya Kerdprasop. An empirical study of distance metrics for k-nearest neighbor algorithm. In *Proceedings of the 3rd international conference on industrial application engineering*, pages 280–285, 2015.

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

Liu Yang and Rong Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2(2):4, 2006.