Quarter 2 Project: Random-KNN

Arya Bharath, James Wright

Machine Learning - Dr. Yilmaz

2/03/25

Abstract

When we were brainstorming for our project and determining which algorithms stood out to test, we were drawn to ensemble learning and the Random Forest (RF) algorithm. Through previous experience comparing RF to other machine learning models and knowledge of ensemble learners, we were wondering if the Decision Tree truly was the best model for the Random Forest algorithm. We then decided that it could be interesting to swap out Decision Trees for the K-Nearest-Neighbors algorithm. While Decision Trees can work well on some datasets, they are constrained by the fact that you only can split on a single individual attribute at a time [1]. On the other hand, KNNs may be able to better capture nonlinear trends in the data - particularly when data from the same class tend to cluster in space [2].

Introduction

Ensemble learning models like Random Forest are far more accurate at the expense of additional computation. With this in mind, we want to attempt to solve this problem of computation with a novel approach to ensemble learning - Random KNN. In contrast to the Decision Tree algorithm, which requires training, the KNN algorithm instead calculates distances each time the algorithm runs, labeling it a “lazy learner.” When comparing the two algorithms (Decision Tree and KNN), KNN will suffer with larger datasets and higher dimensionality due to the exponentially larger number of distance calculations. However, it may have some advantages when data that shares labels is clustered in space, particularly with smaller to medium-sized datasets where the distance calculations will not slow the algorithm to as great of a degree [3]. On the other hand, the Decision Tree algorithm will overfit on the data at higher rates and be more prone to outliers.

This means that while both Random Forest and Random KNN would be slow on larger datasets, Random KNN presents some clear benefits that are worth exploring. As a result, the plan is to input several datasets to our Random KNN algorithm, which will produce results on the testing sets of these data.

Additionally, it is interesting to pose the question of why we chose to implement Random KNN as an ensemble learner. First, because we wanted to compare RKNN with RF, we wanted our ensemble learner to be supervised, like Random Forest is (with Decision Trees). This eliminates unsupervised clustering algorithms like k-means. Next, choosing among supervised algorithms, we believed that KNN made more sense than Naive Bayes due to the independence assumption of the NB algorithm, which could hinder its results and make it less effective as an ensemble learner.

Related Work

There are many published articles on KNN, but those concerning Random KNN are few and far between. The majority of the articles strictly on Random KNN have been written by Shengqiao Li, who seems to have created the algorithm [4]. In his studies, he notes that the primary advantage of the KNN algorithm over the Decision Tree algorithm is the lack of instability that DT has. For Random KNN, there are three parameters that the author addresses. First, *r*, or the number of KNNs used to predict. Second, *k*, or the number of nearest neighbors. Lastly, *m*, the number of features per random KNN. While we can test these hyperparameters in our own work, we are more interested in seeing what types of datasets the Random KNN algorithm excels at predicting on. In the work conducted by S. Li, Li concluded that *m* is best set as the square root of the dimensionality, *k* between 1 and 15, and *r* can be much larger, determined mathematically depending on the dataset [3].

Dataset and Features

We used synthetic sklearn datasets to contrast the performance of Random KNN with Random Forest on a variety of different data. Specifically, we created 10 artificial sklearn datasets with various numbers of samples, features, clusters, and standard deviations in order to compare Random KNN with Random Forest and see how the two algorithms compare. Because all of our datasets were synthetic, no discretization or normalization was needed to preprocess the data. The sklearn configurations we used were:

```
Dataset config: {'n_samples': 100, 'n_features': 5, 'n_clusters': 3, 'cluster_std': 1.0}
Dataset config: {'n_samples': 200, 'n_features': 5, 'n_clusters': 4, 'cluster_std': 1.5}
Dataset config: {'n_samples': 300, 'n_features': 6, 'n_clusters': 5, 'cluster_std': 0.8}
Dataset config: {'n_samples': 150, 'n_features': 4, 'n_clusters': 2, 'cluster_std': 2.0}
Dataset config: {'n_samples': 250, 'n_features': 3, 'n_clusters': 3, 'cluster_std': 0.5}
Dataset config: {'n_samples': 400, 'n_features': 7, 'n_clusters': 6, 'cluster_std': 1.2}
Dataset config: {'n_samples': 180, 'n_features': 4, 'n_clusters': 2, 'cluster_std': 1.8}
Dataset config: {'n_samples': 220, 'n_features': 5, 'n_clusters': 4, 'cluster_std': 0.7}
Dataset config: {'n_samples': 350, 'n_features': 6, 'n_clusters': 5, 'cluster_std': 1.1}
Dataset config: {'n_samples': 100, 'n_features': 3, 'n_clusters': 2, 'cluster_std': 1.5}
```

Each dataset configuration represents a unique, independent dataset to run the RKNN algorithm on. To visualize this, let us take the first dataset configuration, with {‘n_samples’: 100, ‘n_features’: 5, ‘n_clusters’: 3, ‘cluster_std’: = 1.0}. This input would create a dataset like the one shown below.

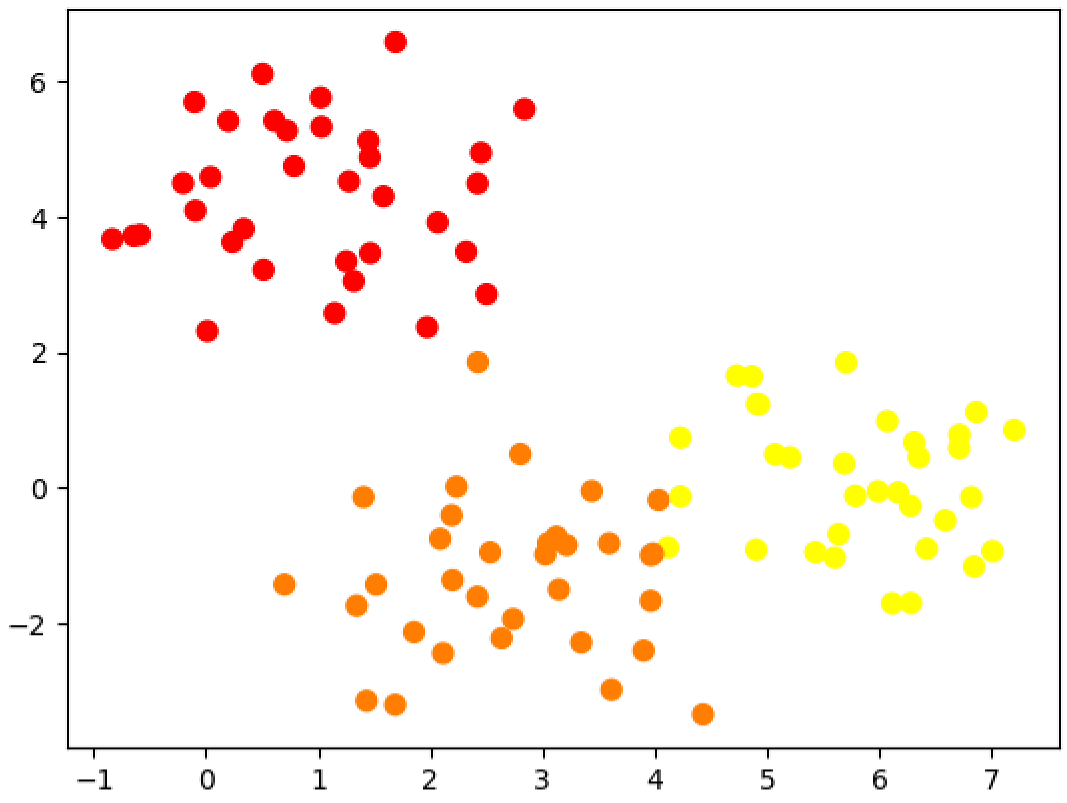

Figure 1. Sample dataset for Data config. 1.

Methods

In our quarter two project, we plan to explore the Random KNN algorithm. This combines the bagging element of Random Forest with the K-Nearest-Neighbors algorithm.

First, the Random Forest algorithm. In the RF algorithm, the dataset is used to generate *n* samples, where *n* represents the number of Decision Trees to be constructed. Each sample has *p* features, where *p* is usually the square root or logarithm of the dimensionality of the dataset. The features are selected without replacement. Then, some number of instances is selected for each sample, with instances chosen with replacement, randomly. At this point, we have *n* samples of data with *p* features each and some fixed number of instances in each sample. In a normal RF algorithm, each sample would be used to train a Decision Tree. In combination, these trees would create an ensemble learner where predictions would be made by aggregating the "votes" of each tree. In this way, the RF algorithm gets to fix the overfitting issue that Decision Trees typically face.

Second, the K-Nearest-Neighbor algorithm. In the KNN algorithm, there is no training stage. Instead, for each test point, a distance is calculated to all training points in the dataset through various means, depending on the type of data. For quantitative data, euclidean distance is often used. However, cosine similarity, manhattan distance, and several other methods exist to calculate distance in the KNN algorithm [1]. At this point, we use the hyperparameter *k* to decide how many neighbors to use to classify. We sort the training points by distance to the testing points and take the labels of the *k* closest points as votes for the class of the test point. In our specific algorithm, we set the *k* of each data sample to the square root of the size of the sample.

Finally, in this research, we plan to combine these two algorithms. We take the sampling method from the Random Forest algorithm and combine it with the KNN algorithm. Thus, we still create *n* samples of *p* features each through bagging, and then use the KNN algorithm on each subdataset.

Experiments/Results/Discussion

Our results can be seen in *Table 1*.

Dataset #	n_samples	n_features	n_clusters	cluster_std	RKNN	RF
1	100	5	3	1.00	0.63	1.00
2	200	5	4	1.5	1.00	1.00
3	300	6	5	0.8	0.99	1.00
4	150	4	2	2.0	1.00	1.00
5	250	3	3	0.5	1.00	0.97
6	400	7	6	1.2	0.99	1.00
7	180	4	2	1.8	1.00	1.00
8	220	5	4	0.7	1.00	1.00
9	350	6	5	1.1	1.00	1.00
10	100	3	2	1.5	1.00	1.00

Table 1. Displays results for 10 dataset configurations.

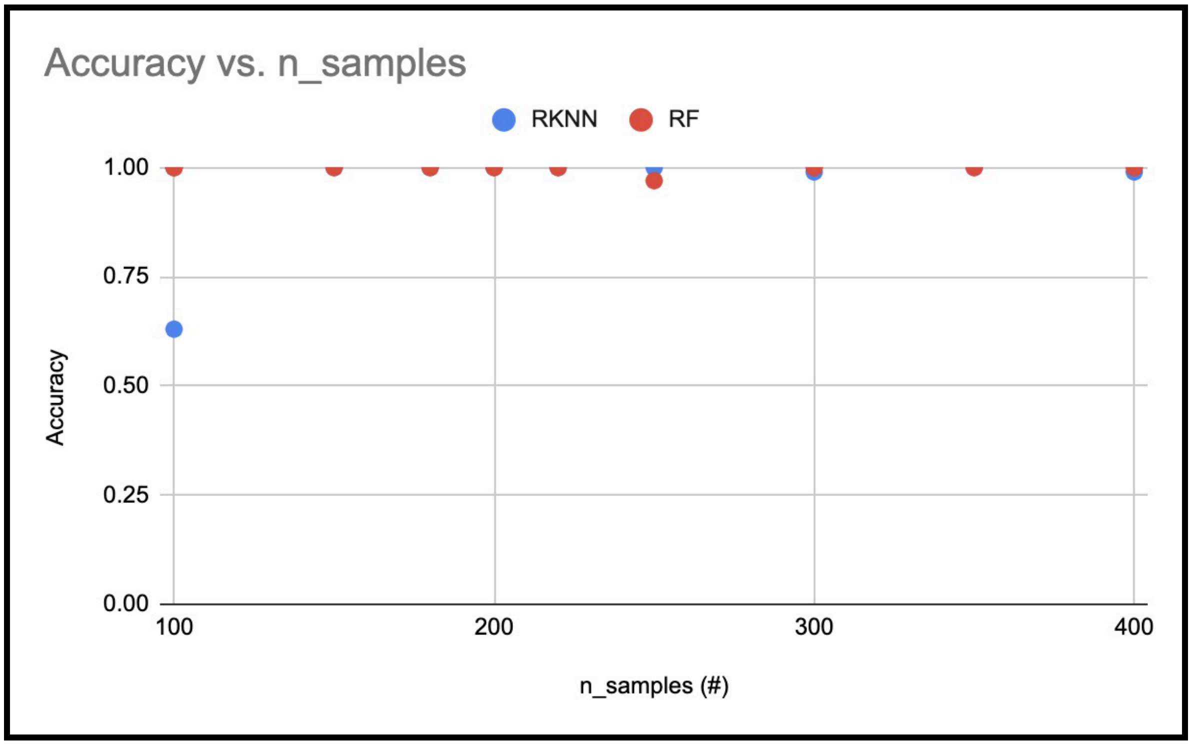

Figure 2. Accuracy vs. n_samples for 10 dataset configurations.

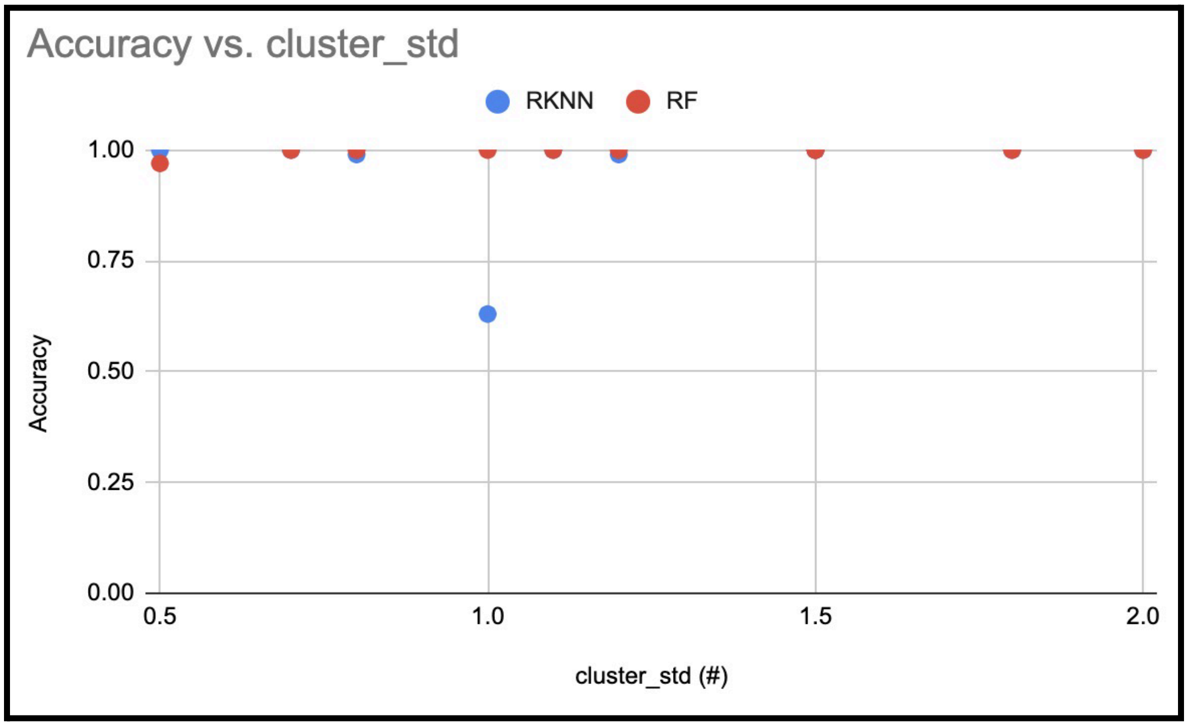

Figure 3. Accuracy vs. cluster_std for 10 dataset configurations.

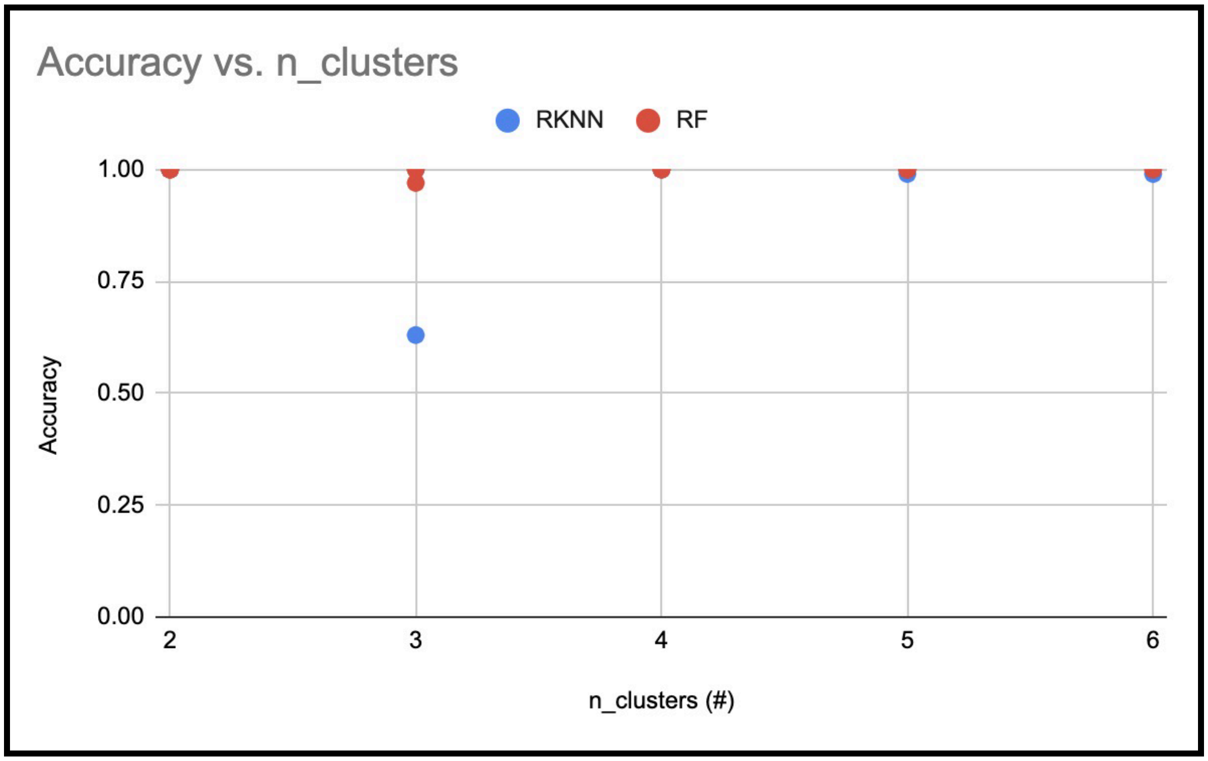

Figure 4. Accuracy vs. n_clusters for 10 dataset configurations.

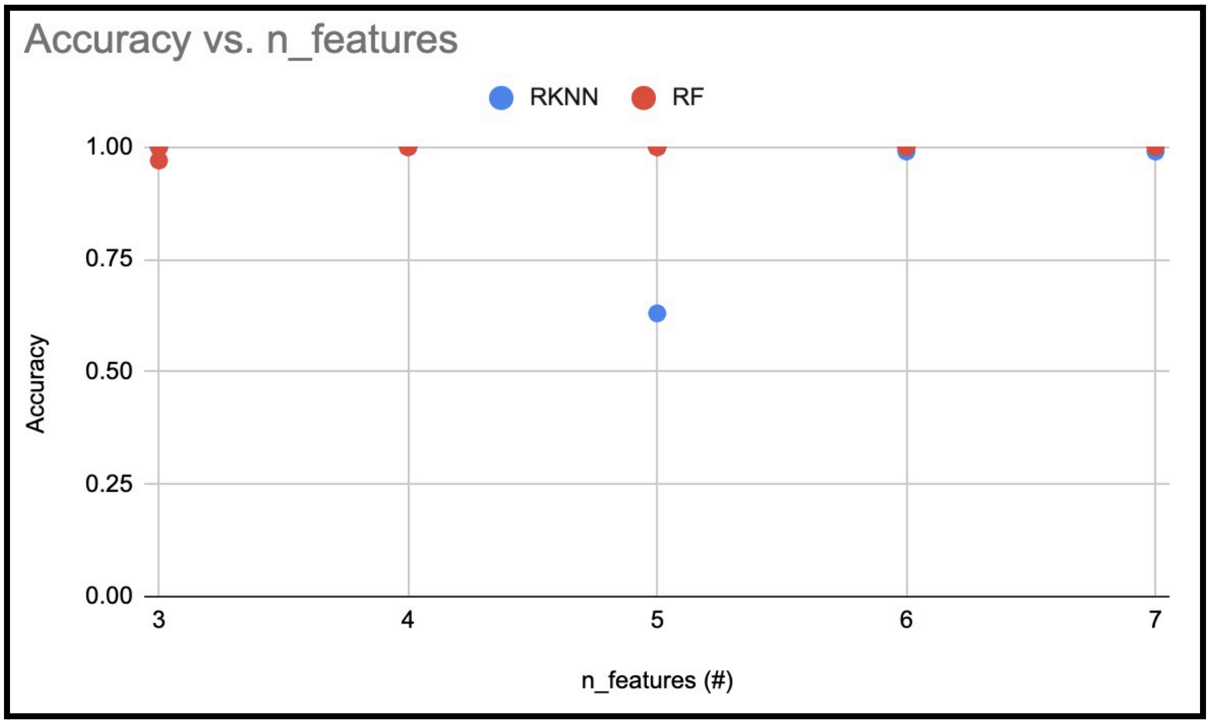

Figure 5. Accuracy vs. n_features for 10 dataset configurations.

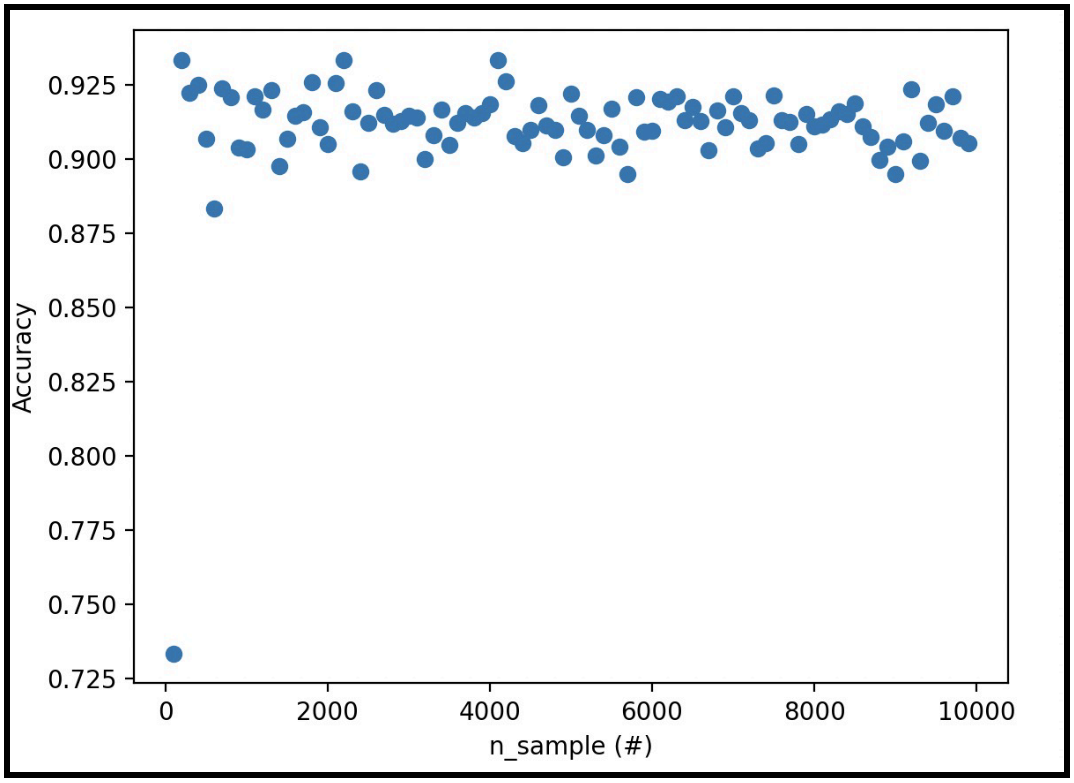

Figure 6. Accuracy vs. n_sample for an increasing number of samples.

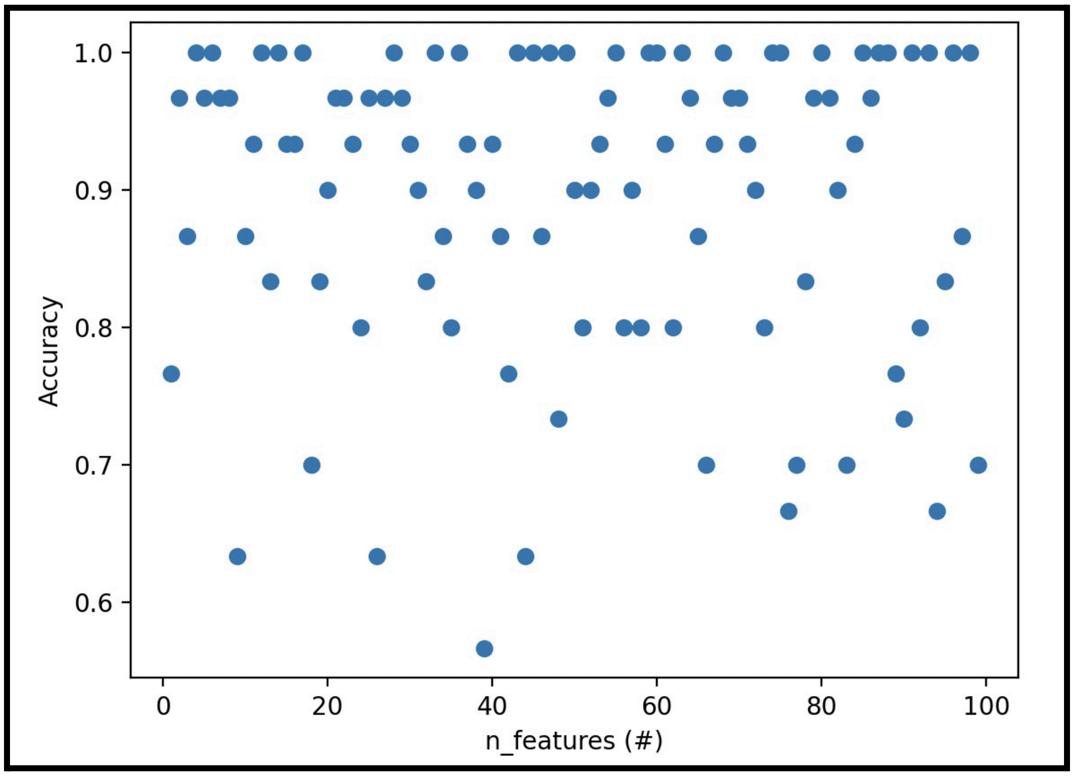

Figure 7. Accuracy vs. n_features for an increasing number of features.

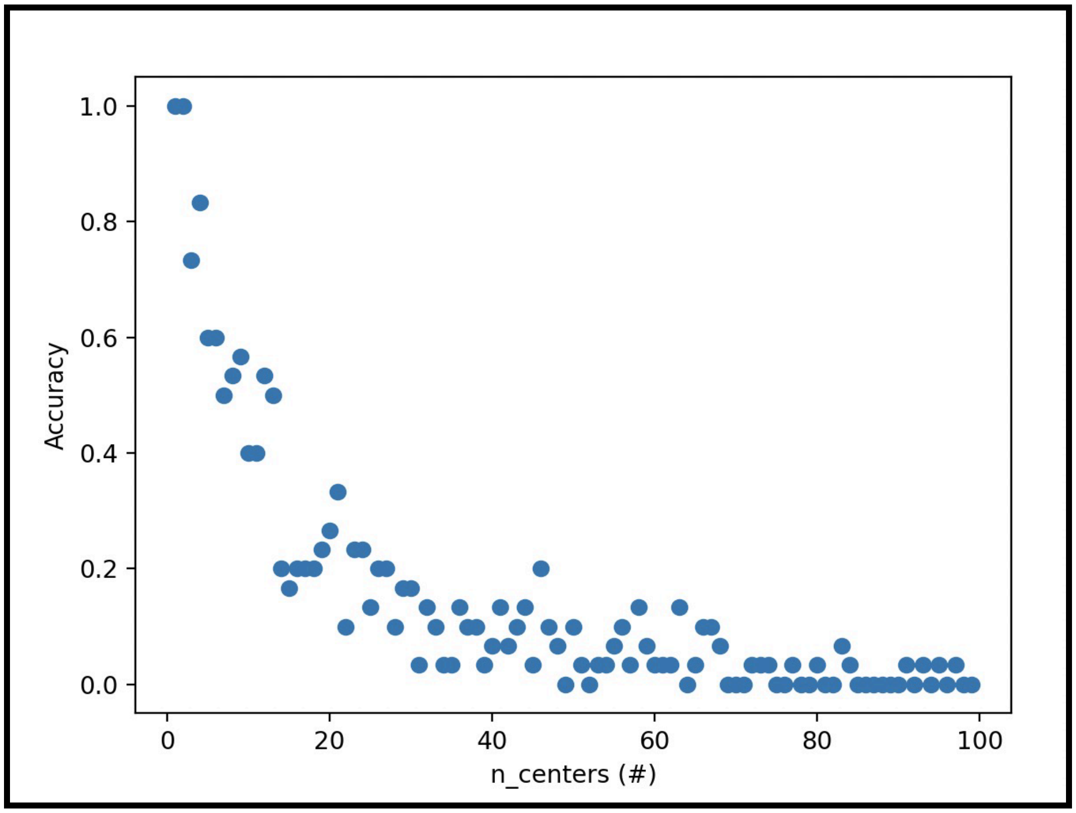

Figure 8. Accuracy vs. n_centers for an increasing number of centers and classes.

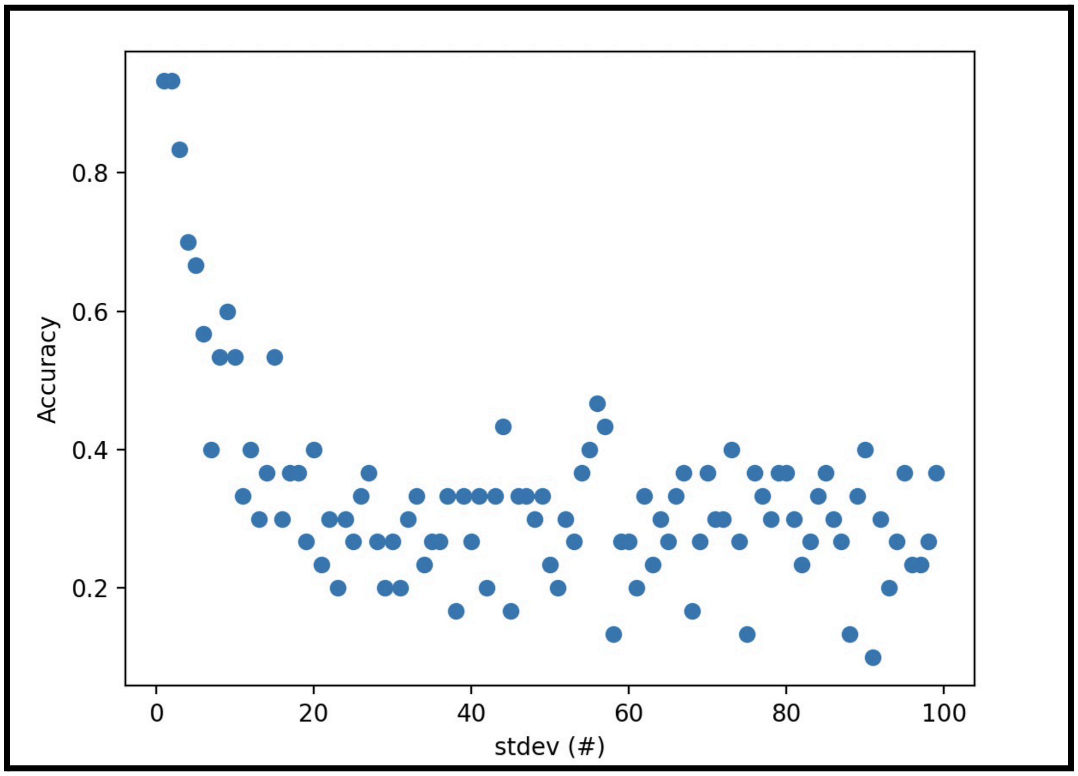

Figure 9. Accuracy vs. standard deviation for an increasing standard deviation.

For the parameters of our algorithm, we had to choose a number of samples to generate from the initial data and a *k* to run each sample on. For the second of these, the *k* to determine the number of nearest neighbors to check, we decided to use the square root of the sample size due to the results of our research. We believe that it is not possible to individually choose a good k-value for each sample, as it does not make sense to fine-tune a hyperparameter on some potentially large number of samples [3]. However, if we were to use a standard *k* that adjusts for the size of each sample, it could still produce decent results. We believe that one of the most substantial weaknesses of this algorithm is that the *k* is not always accurately picked because of this. In some cases, a square root of the sample size is not adequate to produce accurate results, which would explain the outlier 0.63 accuracy in the *Table 1* results.

Additionally, it is interesting to note several of our conclusions. As seen in *Figures 2, 3, 4*, and *5*, Random Forest and RKNN perform at relatively equivalent levels in general, with an occasional exception. Because of this, we chose to isolate each variable in the dataset to see the relationship between that variable and the accuracy of the RKNN model. For example, in *Figure 6*, we inspect the impact of varying the sample size of the dataset (n_samples) on the accuracy of the RKNN model. We find that after a preliminary initial increase in the sample size during which the accuracy increases, the accuracy stabilizes. Another interesting finding can be seen in *Figure 8*. As we increase the number of centers, and therefore the number of classes, the accuracy drops off, as expected. However, we may note that the algorithm seems to perform decently at even relatively high class counts. For example, when there are approximately 20 classes, the accuracy hovers around 0.20, which is four times greater than the accuracy of a model with no information, which would classify randomly at around an accuracy of 0.05. Finally, *Figure 9* shows that higher cluster standard deviations decrease accuracy, in general, which makes sense considering the fact that standard deviation is positively correlated with outliers.

In this sense, we believe that RF and RKNN are, on average, very similar in results - both performing extremely well in many cases. However, because there is a chance that the *k* is not a good fit for the dataset, RKNN has a chance of severely underperforming as well. That said, this could just be a dataset that RKNN happens to perform poorly on, so we cannot generalize this conclusion with complete confidence. Finally, we do not believe that our algorithm overfit on the training data because the RandomForest bagging technique works to combat overfitting by creating distinct samples that prevent DecisionTrees from overfitting.

Conclusion/Future Work

Through our research, we have determined that Random KNN has the potential to perform well on specific types of datasets and data, potentially even outperforming Random Forest. However, in order to conclusively determine the qualities of the algorithm, more testing on a wider variety of data is necessary. There are many possible future experiments that could be completed with Random KNN. First, hyperparameter tuning could be researched with the *r*, *k*, and *m* parameters mentioned in the Related Works section of this paper. Second, it is undoubtedly possible that Random KNN and Random Forest (as well as other ensemble learners) perform favorably on specific types of data and datasets. It would be interesting to explore these further and see where Random Forest is better than Random KNN and vice versa.

Contributions

Both group members collaborated to code and write the final paper.

Bibliography

[1] A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm

Saravanan Thirumuruganathan, "A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm," Machine Learning Blog, May 17, 2010. [Online]. Available: https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/. [Accessed: Jan. 26, 2025].

[2] KNN vs Decision Tree in Machine Learning

"KNN vs Decision Tree in Machine Learning," GeeksforGeeks, 2025. [Online]. Available: https://www.geeksforgeeks.org/knn-vs-decision-tree-in-machine-learning/. [Accessed: Jan. 26, 2025].

[3] Srisuradetchai & Suksrikan, "Random Kernel K-Nearest Neighbors Regression"

P. Srisuradetchai and K. Suksrikan, “Random Kernel K-Nearest Neighbors Regression,” Frontiers in Big Data, vol. 4, pp. 1–12, Jun. 2024. [Online]. Available: https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2024.1402384/full. [Accessed: Jan. 26, 2025].

[4] Shengqiao Li and James E. Harner, “Random KNN”

S. Li and J. E. Harner, “Random KNN,” ResearchGate, 2025. [Online]. Available: https://www.researchgate.net/publication/282233805_Random_KNN. [Accessed: Jan. 26, 2025].

[5] Scikit-Learn Documentation

“scikit-learn: Machine Learning in Python,” scikit-learn.org, 2025. [Online]. Available: https://scikit-learn.org/stable/.