Improved KNN through Hyperplane Division and Instance Removal

Quarter 2 Project

Kaushik Siruvuri and Liam Baird

Thomas Jefferson High School for Science and Technology

Yilmaz Machine Learning Period 1

1/9/23

# Table of Contents

[**Abstract**](#_lmgoc8ytmqm0) **3**

[**Introduction**](#_1ke0jwfffukg) **3**

[**Related Works**](#_igbh5zc8a01r) **5**

[**Dataset and Features**](#_rx8xiegeb8aa) **6**

[Dataset](#_u11wk848dl2h) 6

[Figure 4.1](#_2axsdxz8ae1m) 6

[Figure 4.2](#_tlz8h5fl208) 7

[Preprocessing](#_xclcumeamjn) 7

[Figure 4.3](#_9bwfyh1njlp4) 8

[**Methods**](#_8ri74il0xp3u) **8**

[K Nearest Neighbors](#_qskjovqx05te) 8

[Spatial Division](#_5rqod11he26v) 9

[Hyperplanes](#_rzoy0et85ez7) 9

[Figure 5.1](#_ui5ip72qoogi) 10

[Hyperplane Division](#_vm5oyed5bus) 11

[Figure 5.2](#_tdp1kk2oxmki) 11

[Aggregation](#_k06urxihrb3v) 11

[K-Means Pre-Clustering](#_n1eq8dn02gi8) 12

[Random mutation hill climbing](#_2xs3mi611b78) 12

[Improved KNN Algorithm](#_la2izzio4mg0) 13

[**Experiments/Results/Discussion**](#_igrjh7gcd0fl) **14**

[Metrics](#_3gzmd1ikfjp8) 14

[Aggregation](#_pgven4jx9brh) 14

[K-Means](#_b91i4qfyyu2c) 14

[Random Mutation Hill Climbing](#_58e7z6g9ytxs) 15

[Figure 6.1](#_myxwqrpzou70) 15

[Figure 6.2](#_2rbs5sme4ip5) 16

[Figure 6.3](#_snwkwsckn3ji) 17

[Figure 6.4](#_w7l3chbb0i08) 17

[Hyperplane Division Performance](#_ye7856q5nua2) 18

[Figure 6.5](#_heec1jt98fyt) 18

[Figure 6.6](#_s1kwj0sqy7ok) 19

[KNN Performance](#_980720eg9knj) 19

[Custom Algorithm Performance](#_75oa92a977mr) 19

[Table 6.7](#_f6tmjet020h0) 21

[Figure 6.8 Confusion Matrices](#_icmu2g8fubad) 22

[Figure 6.9 Additional Confusion Matrices](#_r3tysljdtqct) 23

[Figure 6.10](#_rmjtu4pft1ol) 26

[Conclusion and Future Works](#_lthiot9ebe1o) **26**

[**Contributions**](#_mejtcxvolb3e) **27**

[Liam](#_j14xax9fbnnn) 27

[Kaushik](#_8xak74nqt6dp) 27

[**References**](#_pgwosw5n9aui) **28**

# Abstract

The KNN algorithm is a lazy learning algorithm requiring the computer to iterate through all data points in a training data set. For datasets with multiple dimensions and a high instance count, this is a very high cost in both time and memory, making KNN unviable.

To solve this issue, we propose reducing the training data to make KNN more time and space efficient. To accomplish this, we plan to add two steps before the lazy learning process. First, data should be aggregated and reduced in size and space to remove unneeded data. Next, the data should be randomly split by hyperplanes, after which KNN will run. The testing point would only be compared to the aggregated points inside the same group of points split by the hyperplane. This leads to far fewer distance comparisons needing to be made, thus improving the speed of subsequent classifications. The combination of both of these methods should lead to a noticeable decrease in time and space complexity, especially on larger datasets.

# Introduction

K-nearest neighbors (KNN) is a classification algorithm that sorts testing points into different class labels through distance. It is classified as a lazy learning algorithm, which means that there is no model built. Instead, all training data points are iterated through at the time of testing/classification. Although this ensures that no data is lost through a model being built, it also has major drawbacks: KNN has high space and time complexity. For small datasets, this is not an issue, and KNN works fine. However, datasets with a large instance and feature count can be near impossible to run KNN on, especially when there are numerous test/classification points.

This paper aims to improve the general KNN algorithm universally to be able to run on large datasets. To do this, a data reduction plan was developed to reduce the size of data KNN needs to work with. Although this removes some data and adds overhead time before testing can begin, the proposed algorithm should be able to significantly reduce time and space complexity with minimal losses to accuracy as compared with the normal KNN algorithm. To verify the efficacy of this algorithm, we have decided to use a large dataset with numerical inputs and three class variables (See [**Dataset and Features**](#_rx8xiegeb8aa)). Our improved KNN algorithm will be compared to the normal KNN algorithm using the same input dataset.

Our improved algorithm has two main features: data reduction and hyperplane division. First, the instance count of the data is significantly reduced to prevent the KNN algorithm from iterating through too many points. Afterwards, the remaining points are split into different subsets randomly, via a hyperplane. When a sample point is classified using KNN, it only uses the subset the sample point belongs to as its training dataset. This method takes advantage of the curse of dimensionality, and assumes that randomly splitting data will ensure that similar data points land in the same subset randomly. Combined, these two features aim to significantly reduce KNN’s time and space complexity through data reduction and a slight modification of the KNN algorithm.

# Related Works

Wang et al. (2022) use PK-means++ as an alternative, improved clustering, and aggregation algorithm. [[2](#_pgwosw5n9aui)] This works by weighting samples by distance to a centroid and randomly picking one as a new centroid. This method optimizes the problem of k-mean’s dependence on starting values through local probability, which is more likely to choose accurate clusters. However, this still is very time taking for a large k-value and is most likely not going to be used for aggregation.

Condensed Nearest Neighbors is a popular alternative to KNN that reduces data size by generating a subset of the training data and removing points that do not affect the classification of points in the training data. This approach will retain ‘noisy’ data and amplify its effects thus reducing accuracy but improving speed. (Wilson & Martinez,2009) [[8](#_pgwosw5n9aui)]

Yong et al.(2009) use K-means clustering to generate a new training dataset. This approach addresses the multi-peak nature of text classification datasets by removing data points near the boundaries of clusters. This approach is not designed to improve the speed of KNN but rather the accuracy. They use k-means to cluster thus suffering from time and space issues at higher instance counts. [[3](#_pgwosw5n9aui)]

Bhaya (2017) lists clustering as a method for outlier detection - points that fall outside a cluster by having a significantly higher distance to the nearest cluster than other points in the cluster can be removed as outliers. This approach is a general preprocessing technique and will remove relatively few points. [[7](#_pgwosw5n9aui)]

Zhu et al.(2013) cluster each class of training data into clusters then combine similar samples. This approach improved the speed of KNN and retained similar accuracy. [[4](#_r3nptbxy2k8a)]

Scikit-learn offers ball trees as an algorithm to compute nearest neighbors. “A ball tree is a complete binary tree in which a ball is associated with each node in such a way that an interior node’s ball is the smallest which contains the balls of its children.” Omohundro (1989). [[9](#_pgwosw5n9aui)]

Scikit-learns also offers KD-trees, KD-trees pick a random dimension then split the data into blocks divided by the median value, each of these blocks are children of the original node. Bentley (1975). [[10](#_pgwosw5n9aui)]

Due to similarities with splitting space and data, KD-trees and ball-trees offer the best comparison for our algorithm.

# Dataset and Features

## Dataset

Our main dataset criteria was a high instance count. Our proposed algorithm would only show its effects when a dataset had an incredibly high number of instances such that iterating through all of them would be unfeasible for KNN. We elected to choose data from a sensor array, as it had 928991 instances and a decent number of dimensions (10). The sensor array consisted of eight metal oxide sensors measuring gas concentration, one temperature sensor, and one humidity sensor. These sensors were exposed to three types of stimuli: background, banana, wine, the latter two by placing them close to the sensors. The input data is a set of floats, being the values read by the sensors, while the output is the type of stimuli the sensors were exposed to. [[6](#_r3nptbxy2k8a)]

### 

### Figure 4.1

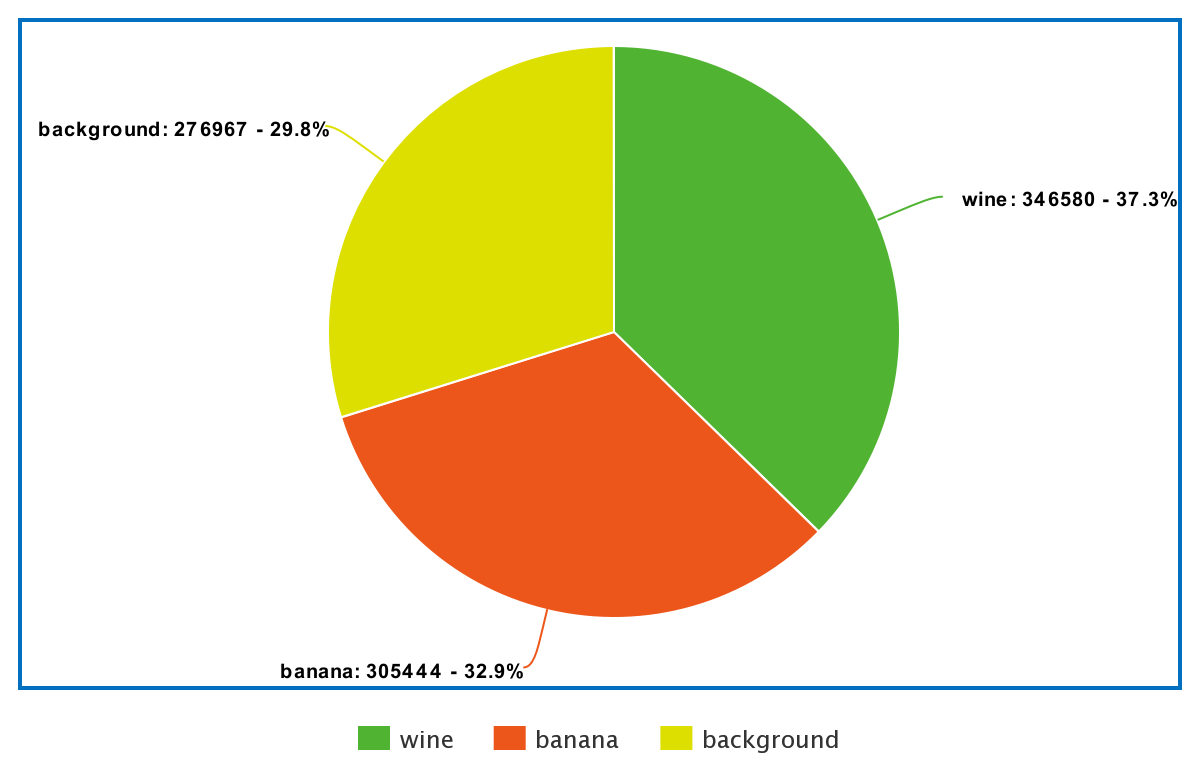
Sample Entry

| iid | ttime | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | Temp. | Humidity |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | -0.99975 | 12.8621 | 10.3683 | 10.4383 | 11.6699 | 13.4931 | 13.3423 | 8.04169 | 8.73901 | 26.2257 | 59.0528 |

id corresponds to a stimulus - banana, wine, background

### Figure 4.2

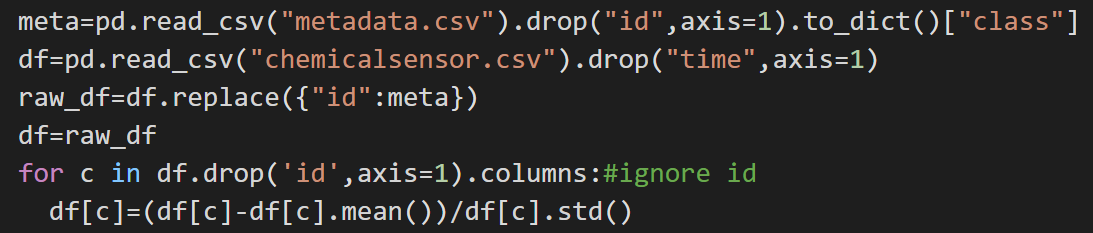
Class Imbalance



## Preprocessing

Our dataset has no missing values and it is all continuous numeric data. The class imbalance is small. Additionally, the original dataset had several numbers for the class values that corresponded to the three actual classes, the key to which was in a separate header file. We had to map these into a pandas DataFrame to work with the data. Figure 4.3 showcases the code used to accomplish this.

### Figure 4.3



Additionally, we used z-score normalization to reduce computations of which region the point lies in by simplifying the hyperplane generation process, without normalizing the values the hyperplanes will tend to divide the data points into fewer regions with the vast majority of the regions having 0 points. By normalizing the values we center the points near the origin which is effectively the same as generating the planes about the center of the points.

# Methods

## K Nearest Neighbors

K Nearest Neighbors (KNN) is a supervised machine learning algorithm that sorts data based on distance. KNN is a lazy learner, meaning it only iterates through training data when a testing point is given. A typical KNN algorithm follows these steps:

***Given training data set S and testing point I***

*for every training data point Si in S:*

*Calculate distance from Si to I*

*Create set A containing k smallest calculated distances*

*Return the majority classification of set A*

While KNN is easy to implement and versatile without the need to train a model, being a lazy learner makes it difficult to use on large datasets. Additional problems include difficulty in selecting the correct k-value and a large time and space complexity. KNN has a time complexity of O(n\*m) where n is the number of points in the training set and m is the number of points to classify. In this paper, we hope to explore and mitigate some of the disadvantages of lazy learning while still retaining the essential features of the KNN algorithm.

## Spatial Division

Spatial division can be used to reduce data size. By organizing data into subsets based on the spatial location of the data, comparisons can be made easier and more efficient. In simple terms, data far away from a test point will not be used, as it would likely not help the model. Our proposed spatial division method is random hyperplane division.

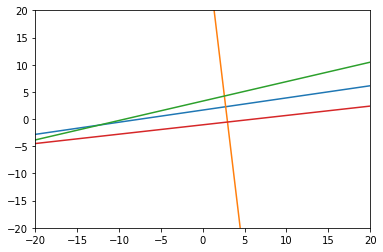
### Hyperplanes

Hyperplanes are geometrical objects that can exist in any dimension. They are categorized as always having a dimensionality one less than the space they exist in. For example, a hyperplane in two dimensions would be a line, while a hyperplane in three dimensions would be a regular plane. The equation of a hyperplane we plan to use in this paper is as follows:

ii

where C is a constant, Ai is a coefficient of a variable, and Xi is a distinct variable that characterizes the space. An example of a three-dimensional hyperplane would be 5 + 3x + 4y + 7z = 0. By nature, each hyperplane splits a space into two regions. Combining hyperplanes would result in multiple regions of space. Figure 2.1 shows a 2d space with four randomly generated hyperplanes (lines) splitting the region into eleven smaller regions, with nine being pictured.

#### Figure 5.1

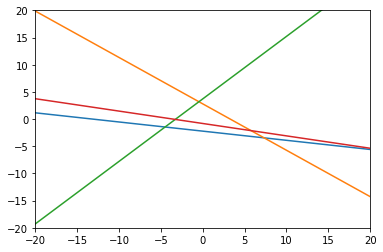


The maximum number of regions n hyperplanes can split an m-dimensional space into can be found using this function: . However, this value is unlikely to appear through randomly generated hyperplanes as it requires a specific orientation that is nigh impossible to generate randomly. To generate hyperplanes randomly, the Ai term in the first formula above needs to be modified or randomly selected. Since any number between 0 and infinity can be selected, this tends to skew the coefficients towards smaller numbers. In 2 dimensional space, this would look like multiple lines with small slopes. To counteract this, we instead use angles and convert that to coefficients. is used to achieve an even range of values without skew.

### Hyperplane Division

To generate hyperplanes given n dimensions, we generate n values ranging from 0 to 𝛑 and use the tangent function to convert to cartesian coefficients (*A*) and achieve an even distribution of lines from these values. We then add an additional shifting value C ranging from -1 to 1 (as we use z-score normalization the standard deviation is equal to 1). We generate m planes where m is the natural log of the number of instances rounded up.

#### Figure 5.2



To assign a point a region number, create a binary number where each digit indicates which side of a plane the region lies on. We do this because integers take up less memory than tuples of booleans. (For 14 planes a tuple of booleans takes up 152 bytes and an equivalent integer takes up 28 bytes)

where P is a Point, H is the list of hyperplanes, *f* is a function that takes a point and a hyperplane and determines which side of the hyperplane the point lies on, R is the assigned region number.

## Aggregation

In order to further decrease the number of comparisons that need to be made we use an aggregation technique to reduce the number of data points KNN will run on. The aggregation algorithm needs to have better performance than the k-means algorithm in order to improve performance over KNN.

### K-Means Pre-Clustering

K-means is a clustering algorithm that moves k centroids and maps them to various sections of data. It is an algorithm that establishes linear boundaries between data through distance calculations and is an unsupervised learning algorithm. It is particularly resource intensive and has a high space and time complexity. We intended to test k-means as a form of data reduction, where a high k value could be used to aggregate data while preserving the general location of removed points through means.

### Random mutation hill climbing

Random mutation hill climbing is a data reduction algorithm that has a high level of customization. It emulates Darwinian evolution and makes random changes to a subset of the full training dataset repeatedly. The resulting dataset is typically much smaller and can even be more accurate, as it also removes outliers. The pseudocode for such an algorithm is as follows [[1](#_r3nptbxy2k8a)]:

***Given training data set S, number of iterations n, and wanted subset size m:***

*Sample S to create subset S’ of size m*

*for n iterations:*

*Replace random point P in S’ with random point Q in S.*

*Evaluate the accuracy of S’*

*if the accuracy has decreased, revert S’ to include point P*

*Return S’*

This algorithm allows for both time and space complexity to be directly customized by the end user, but makes it difficult to find an acceptable size for both m and n. Additionally, evaluating the accuracy of S’ is often difficult to do and maybe resource intensive, especially when run n times. In this paper, the accuracy of S’ prime is determined by running KNN. A subset of S not in S’ is used as test data, while S’ is used as training data.

## Improved KNN Algorithm

Utilizing the above methods, the improved KNN algorithm would first include data reduction using random mutation hill climbing, and then hyperplane generation and division. Afterwards, a slight modification is made to the general KNN algorithm to allow for significant speedups, shown in the pseudocode below.

***Given training data set S, set of hyperplanes H and testing point I***

*Calculate the hyperspace region that I is in using H*

*Find S’ such that it is a subset of S that exists within the same hyperspace region as I*

*for every training data point Si in S’:*

*Calculate distance from Si to I*

*Create set A containing k smallest calculated distances*

*Return the majority classification of set A*

This simple change retains most of the benefits of the original KNN algorithm while greatly reducing run times.

# Experiments/Results/Discussion

## Metrics

Our primary metrics are runtime and accuracy. Since the main goal is to reduce time-complexity, we will measure the runtime of all combinations of algorithms and compare them with other algorithms. Additionally, we want to ensure that a drop in runtime does not equate to a large drop in performance, so we will use accuracy as a standard metric. Since the dataset is well balanced, accuracy is a good overall standard measure of performance. We ran all the code on the same machine with the same random state to ensure fair comparisons. We used a jupyter notebook to more easily edit and change code.

## Aggregation

### K-Means

Originally, we tried to use k-means with a high centroid count to cluster points that were close to each other together, with the intention of reducing the instance count and thereby reducing space and time complexity when KNN is run. However, we found that running k-means on such a large dataset would take too long, and the time K-means clustering would save would not show its benefits in terms of time saved unless the testing dataset was extremely large and KNN needed to be run countless times.

### Random Mutation Hill Climbing

Contrary to k-means, random mutation hill climbing proved to be very successful. We tested a range of m (subset size) and n (iterations) values, and almost all of them significantly reduced data size while actually increasing accuracy. For the following test cases, we split our dataset into 90% training data and 10% testing data, and then ran the random mutation hill climbing algorithm on the training data. Afterwards, we tested KNN on both the full training dataset and the modified training data set created through random mutation hill climbing. An example can be found below in Figure 6.1, where m was set to 10,000 and n was set to 100.

#### Figure 6.1

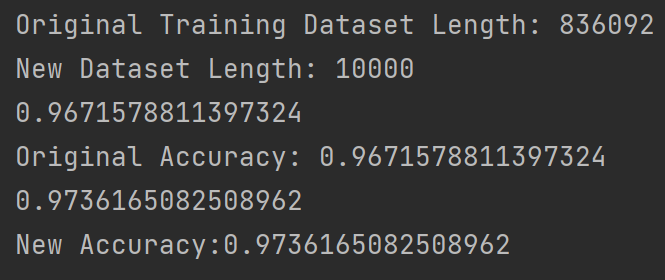
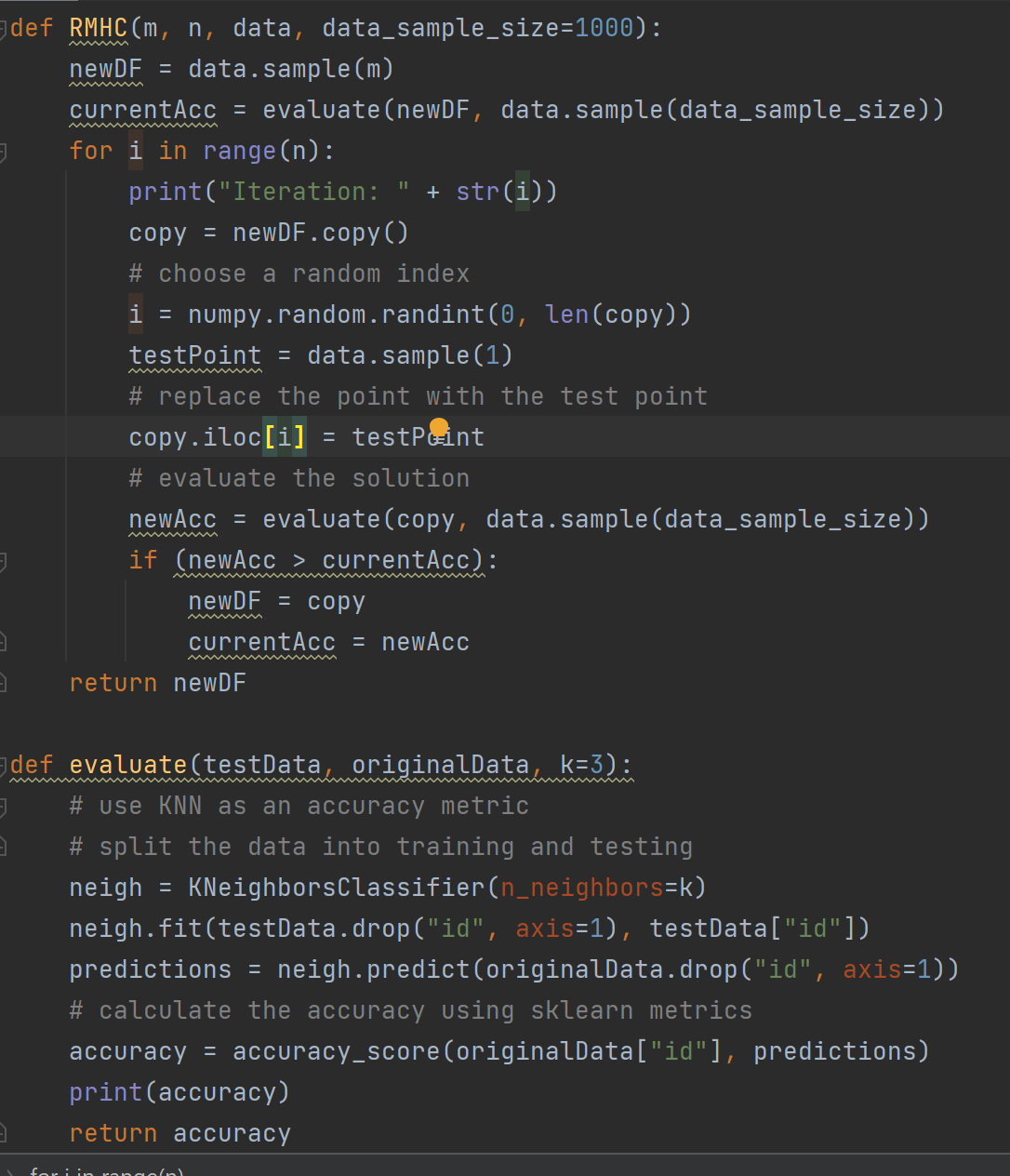


Figure 6.2 showcases our implementation of this algorithm.

#### Figure 6.2



Below in Figure 6. 3 is a plot of accuracy as m changed from 10,000 to 90,000. N was held constant at 25 iterations, and the original model accuracy was 0.9671578811397324, where KNN was run on the full dataset. In all of these instances, the new data subset had a higher accuracy than the original dataset.

#### Figure 6.3

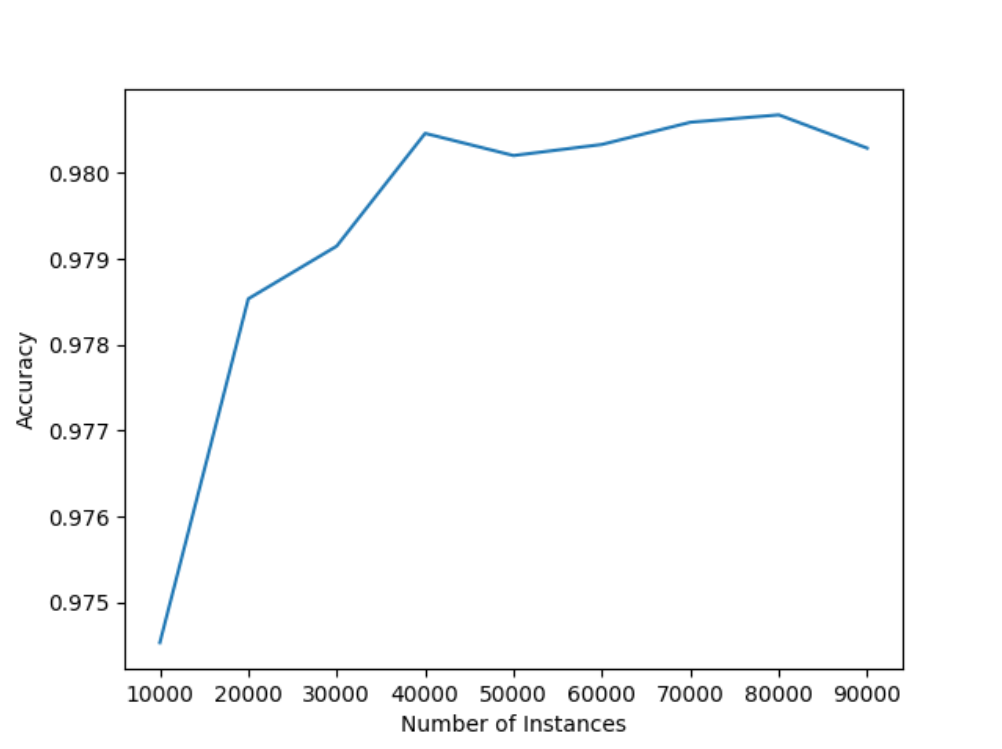
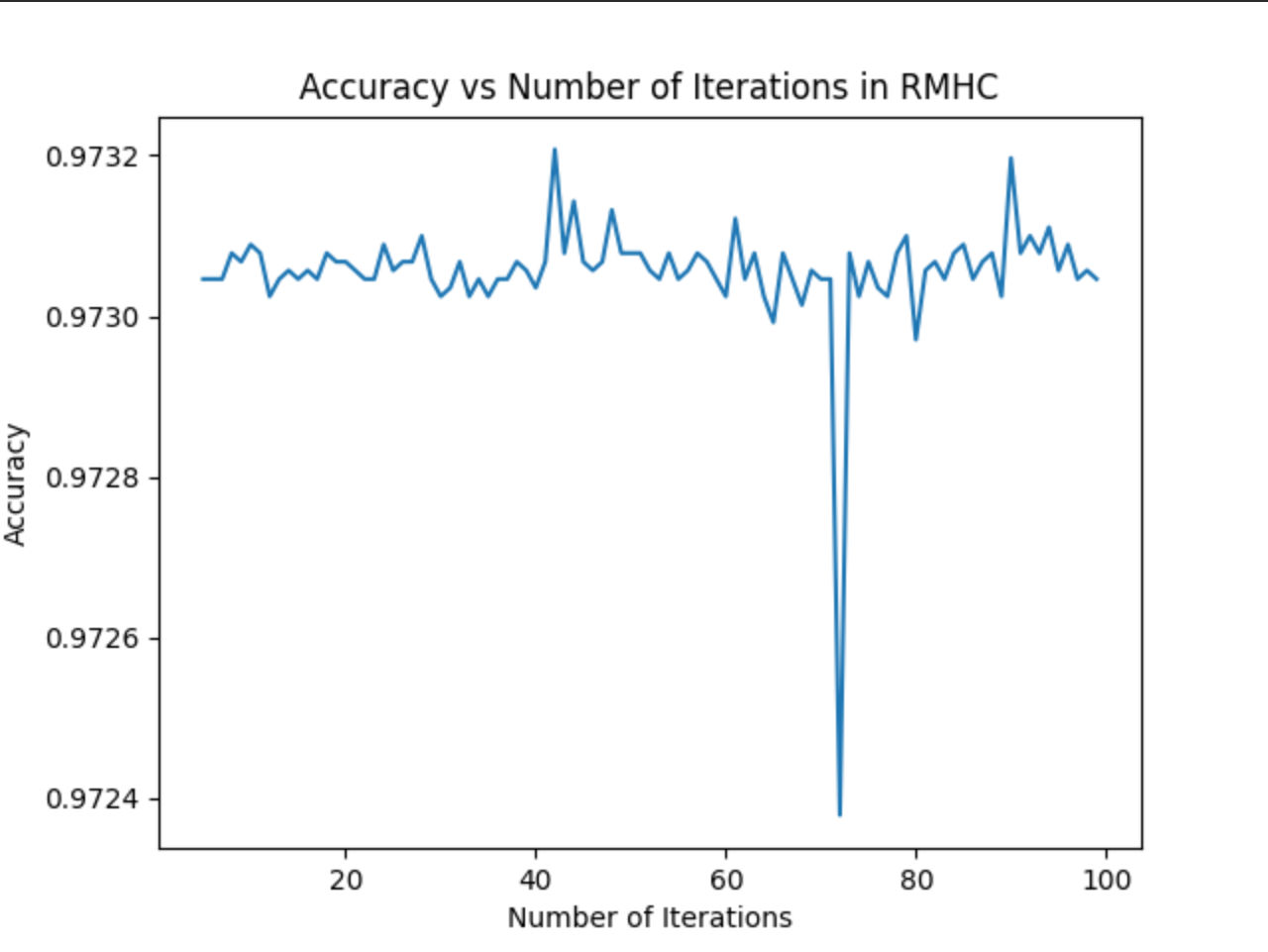


Figure 6.4 shows a plot of accuracy where m is held constant at 10,000, while n is changed from 1 to 100. The original model accuracy was still 0.9671578811397324. The randomness of this graph is probably due to the random starting sets. In the future, we will update this graph with one that compares accuracy over n iterations with the same starting subset.

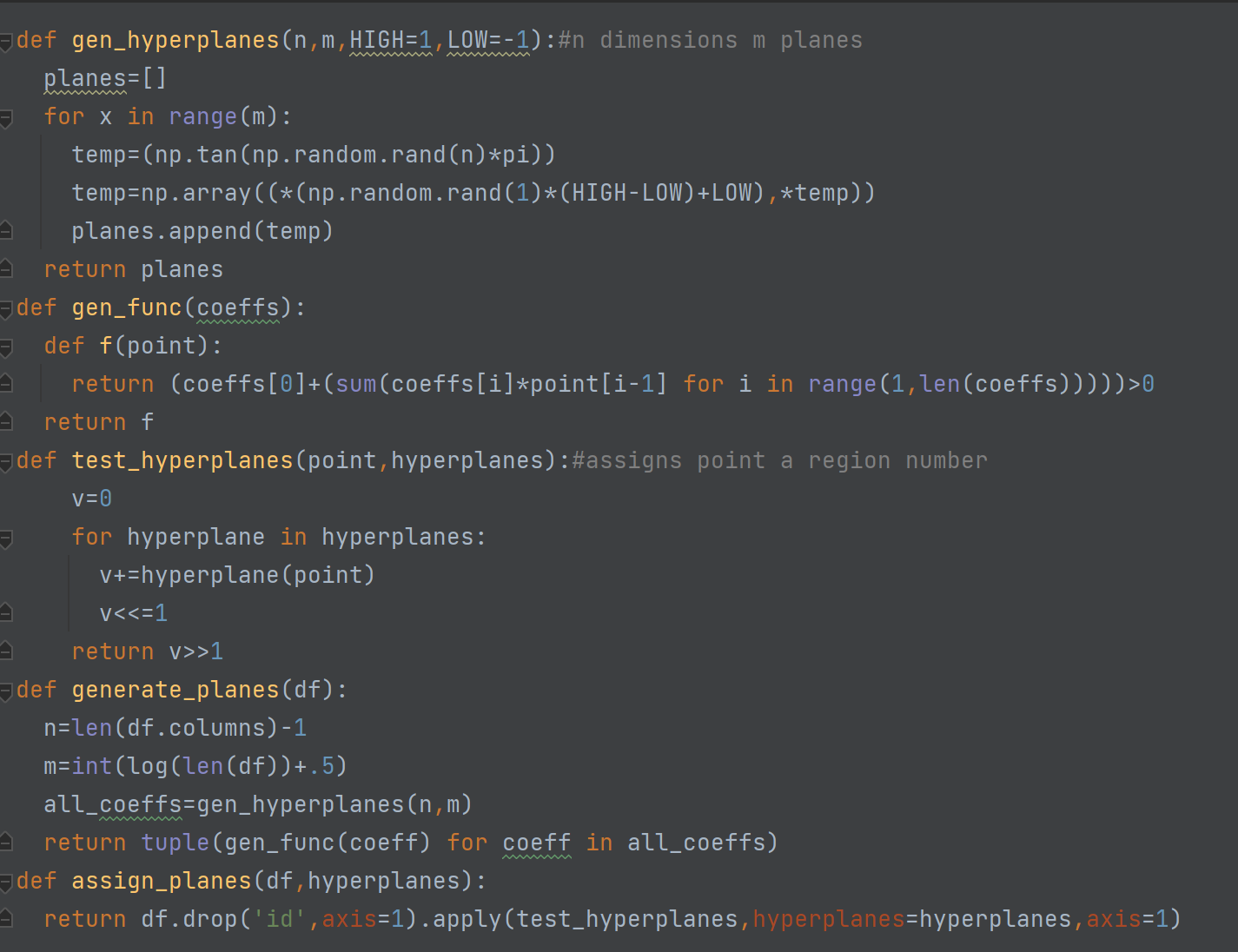
#### Figure 6.4



## Hyperplane Division Performance

We decided to use ln(n) random hyperplanes to split data in our algorithm. However, this number can be modified as needed. Dividing our dataset of 928991 instances took 360 seconds when dividing on 14 planes (ln(928991) rounds to 14). We have yet to run KNN on these hyperplane divisions. Figure 6.5 shows our hyperplane generation and division methods, while Figure 6.6 shows a sample run of this algorithm.

#### Figure 6.5



#### Figure 6.6

A sample run:

| Number of distinct regions | 1121.000000 |
| --- | --- |
| Average points per region | 828.716325 |
| Standard Deviation of points per region | 2338.598255 |
| Minimum points per region | 1 |
| Maximum points per region | 30260 |

## KNN Performance

We ran all KNN algorithms with a standard k of 5, as it was believed to be a good generic k-value for most datasets. Running KNN over the whole dataset took 4 minutes and 27 seconds, and had an accuracy of 0.9999784712429628. Although the accuracy is high, especially for a dataset of this size, we want to drastically reduce the runtime while keeping roughly the same accuracy. We used scikit-learn’s KNeighborsClassifier to run KNN, and made sure to select the “brute” option in the algorithms parameter. This ensures that the typical KNN algorithm is run on the full training dataset.

## Custom Algorithm Performance

Our algorithm begins by running RMHC on the full dataset. Figure 6.10 shows the python implementation. We arbitrarily chose to restrict the size of the dataset to 20%, so we chose an *m* value of 200,000 and an *n* value of 100. This process took 47.5 seconds. Although this is quite high, it permanently reduces runtime when running numerous testing instances in the future. Running KNN on this new dataset formed from RMHC took 1 minute and 58 seconds as opposed to the previous 4.5 minutes, and resulted in an accuracy of 0.9998735185524064. This is significantly faster, and only has a very slight accuracy drop.

The next step in the process is to significantly reduce runtime through hyperplane division. Running the hyperplane division and assignment algorithm took 1 minute and 36 seconds on the reduced data set, and 3 minutes 25 seconds on the full dataset. Finally, our modified KNN algorithm ran in 19.1 seconds, and resulted in an accuracy of 0.9792059117966824. For reference, running just the hyperplane division on the full dataset ran in 38.1 seconds and resulted in an accuracy of 0.9777311919396333.

Overall, our improved KNN model resulted in a significantly reduced runtime, while sacrificing only a slight portion of the accuracy. Table 6.7 shows the times used for KNN to run through and classify all test points, as well as the corresponding accuracy. The parentheses indicate any additional time taken during preprocessing. Note that this is a constant time given the same training dataset. For larger testing sets, the major reduction in data size would result in a sizable total speed up with a minimal drop in accuracy. In our case, with a 60-40 train test split, the time to run our algorithm was faster than the other four variations we tried totaling 162.6 seconds. Additionally, all of these runs were done with the same random state to ensure accurate comparisons.

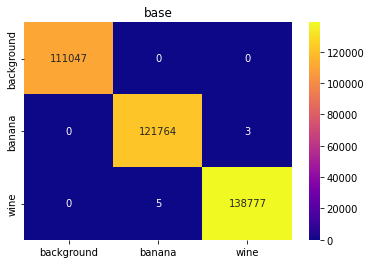
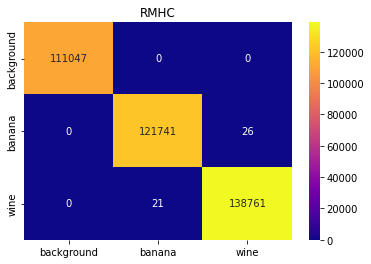
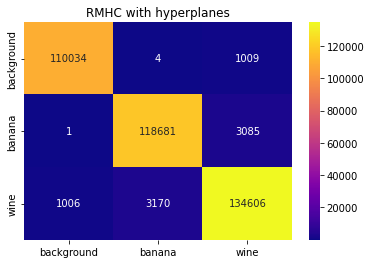
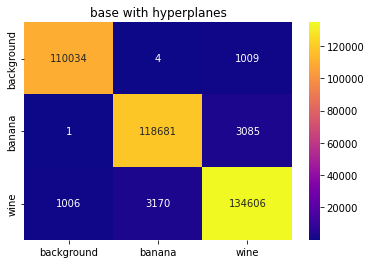
It is also possible to subdivide regions further. Scikit-learn provides KD-trees and Ball Trees as a way to optimize nearest neighbor calculations. We also tested running KD-trees and Ball Trees in addition to RMHC and hyperplane division.

### Table 6.7

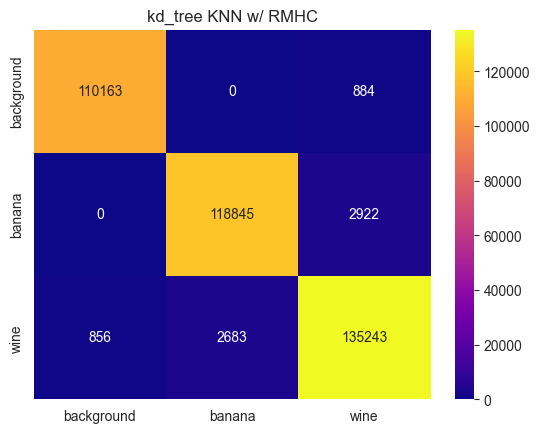
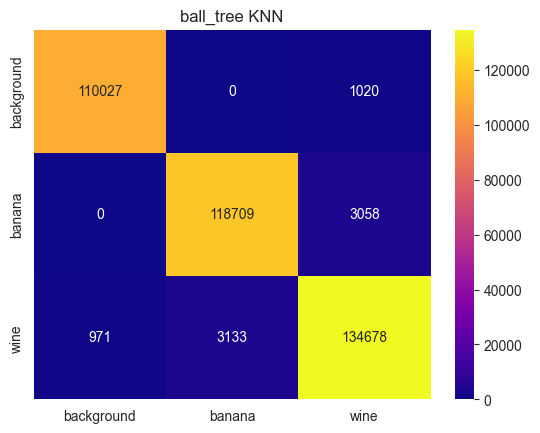
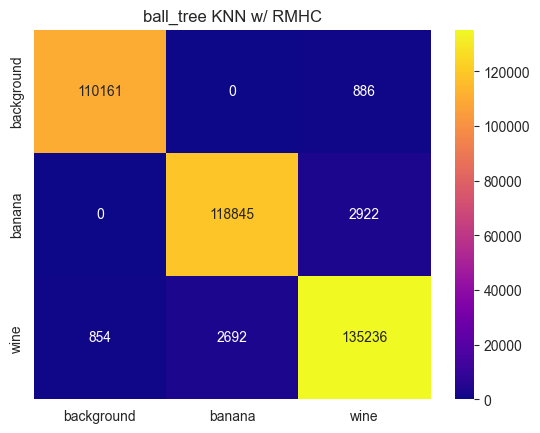
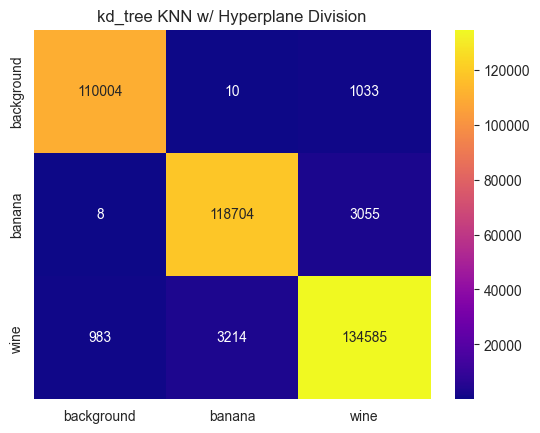
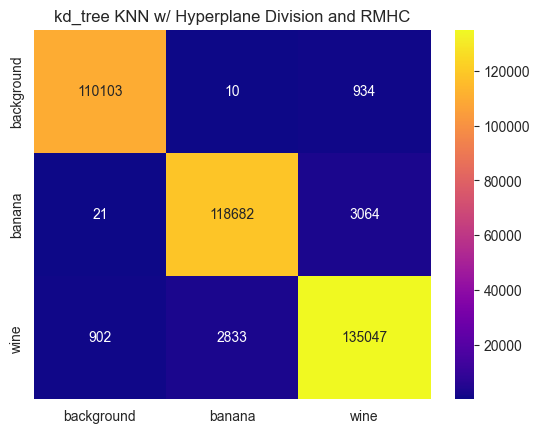
| Algorithm | Time to Run (seconds) | Accuracy |
| --- | --- | --- |
| KNN | 267 | 0.9999784712429628 |
| KNN on Reduced Dataset through RMHC | 118 (+47.5) | 0.9998735185524064 |
| KNN with Hyperplane Division | 38.1 (+205) | 0.9777311919396333 |
| KNN with Reduced Dataset through RMHC and Hyperplane Division | 19.1 (+143.5) | 0.9792059117966824 |
| CNN | 60 (+19695) | 0.6157640985728808 |
| Kd\_tree KNN | 7.15 | 0.9779518616992648 |
| Kd\_tree KNN on Reduced Dataset through RMHC | 5.66 (+47.5) | 0.9802339099452093 |
| Kd\_tree KNN with Hyperplane Division | 10.3 (+148) | 0.9776558412900032 |
| Kd\_tree KNN with Reduced Dataset through RMHC and Hyperplane Division | 6.7 (+145.5) | 0.9791063412953853 |
| Ball\_tree KNN | 288 | 0.977981463740191 |
| Ball\_tree KNN on Reduced Dataset through RMHC | 222 (+47.5) | 0.9802096900935424 |
| Ball\_tree KNN with Hyperplane Division | 14 (+220) | 0.9772333394331478 |
| Ball\_tree KNN with Reduced Dataset through RMHC and Hyperplane Division | 7.52 (+150.5) | 0.9790982680114964 |

[Formatted Comparison](https://docs.google.com/spreadsheets/d/18qvQkfSrFbvyKfMbuZ0CEGSVx0ybysDJ8riMgiP3ZsI/edit?usp=sharing)

### Figure 6.8 Confusion Matrices



### Figure 6.9 Additional Confusion Matrices

### Figure 6.10



# Conclusion and Future Works

While we use Random Mutation Hill Climbing in our experiments, any aggregation algorithm or clustering algorithm with better performance than k-means clustering would be viable to use. We also made arbitrary selections for the number of instances to keep from Random Mutation Hill Climbings but the number of instances would depend on the dataset and the value of accuracy versus speed. Future work would involve finding a better way to calculate these values.

Additionally, this entire process was entirely dependent on randomness, in both data reduction and hyperplane division. This means that training models vary wildly, and can be inconsistent with each other. In the future, we may want to find a reliable way of generating hyperplanes to split data and choose points for data reduction that isn’t completely reliant on chance.

Applying different algorithms to divide the dataset allows for greater control of accuracy versus speed. In terms of both metrics, running kd\_tree KNN with RMHC seemed to work very fast while still retaining much of the accuracy. Although our algorithm takes longer and is slightly less accurate, it still succeeds in being able to significantly reduce runtime and space complexity while retaining accuracy.

# Contributions

## Liam

* Graphics
* CNN
* Hyperplane Division Code
* Report and Presentation
* Dataset and Preprocessing

## Kaushik

* RMHC
* Ran Code and Testing Instances for Timings and Accuracy
* Aggregation Testing
* Report and Presentation
* Algorithm Idea

# References

[[1]](https://drive.google.com/file/d/1HappiOxjDgtSV5-2ZVye3KTj1p9QQ-HE/view?usp=share_link) Si, L., Yu, J., Wu, W., Ma, J., Wu, Q., & Li, S. (2017). RMHC-MR: Instance selection by random mutation hill climbing algorithm with MapReduce in big data. *Procedia computer science*, *111*, 252-259.

[[2]](https://drive.google.com/file/d/1TfwjpQDEEDgrfIiisHTwA7Gj95h5RsAs/view?usp=share_link) Wang, H., Xu, P., & Zhao, J. (2022). Improved KNN algorithms of spherical regions based on clustering and region division. *Alexandria Engineering Journal*, *61*(5), 3571-3585.

[[3]](https://drive.google.com/file/d/15RVBYrtw7HRwiGnnQKIRw6msJzepEso4/view?usp=share_link)Yong, Z., Youwen, L., & Shixiong, X. (2009). An improved KNN text classification algorithm based on clustering. *Journal of computers*, *4*(3), 230-237.

[[4]](https://drive.google.com/file/d/1C5B3tDRbckzRt4D9S3nW1HmhHwaParMX/view?usp=share_link)Jing, Y., Gou, H., & Zhu, Y. (2013, June). An improved density-based method for reducing training data in KNN. In *2013 International Conference on Computational and Information Sciences* (pp. 972-975). IEEE.

[[5]](https://drive.google.com/file/d/1rHjvhhwWlJF_H3pRQbC5jnF13PuuN1ou/view?usp=share_link)Wang, H., Xu, P., & Zhao, J. (2021). Improved KNN Algorithm Based on Preprocessing of Center in Smart Cities. Complexity, 2021.

[[6]](http://archive.ics.uci.edu/ml/datasets/gas+sensors+for+home+activity+monitoring) Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml/datasets/gas+sensors+for+home+activity+monitoring>]. Irvine, CA: University of California, School of Information and Computer Science.

[[7]](https://drive.google.com/file/d/13ZxQ7CI4HFuPS2-_j3S7bSbd3GCgdNQk/view?usp=share_link) Alasadi, S. A., & Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. Journal of Engineering and Applied Sciences, 12(16), 4102-4107.

[[8]](https://drive.google.com/file/d/1Go3UbebGYRvzsN3wYNeSesUsJM9aOesJ/view?usp=share_link) Wilson, D.R., Martinez, T.R. Reduction Techniques for Instance-Based Learning Algorithms. Machine Learning 38, 257–286 (2000). <https://doi.org/10.1023/A:1007626913721>

[[9]](https://drive.google.com/file/d/1CZYKdulM-4PfPAYJ-5JsW_BpG2VKXFul/view?usp=share_link) Omohundro, S. M. (1989). *Five balltree construction algorithms* (pp. 1-22). Berkeley: International Computer Science Institute.

[[10]](https://drive.google.com/file/d/1MTl6kTnccntl2v6fHdoO27tdgNPZIcjZ/view?usp=share_link) Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509-517.