International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Double Ensemble kNN: Towards an Enhanced k-Nearest-Neighbors

Dhruv Roongta (TISB)[1], Dr. Anasse Bari (NYU)[2]

[1, 2]Pioneer Academics

Abstract: ***Predictive Analytics, a branch of Data Science, has seen a surge in interest, attributed to the rise of Big Data and corporations' needs to identify consumer trends. Unsupervised Learning refers to a branch of Machine Learning in which unlabeled data is sorted into clusters, to identify trends and create target-segments. This can be used within Predictive Analytics to predict spam e-mails, which customers are likely to return, and more. Several unsupervised learning methods have been created, with K-means, an iterative clustering algorithm, being widely used due to its simplicity and stable nature. This paper creates three additional Ensemble methods using the K-Nearest-Neighbor as the base-learner. The Double-Ensemble, a novel method introduced here uses an Ensemble of kNNs with different 'k's on subsets of data with randomized features. Experimental results demonstrated that the Double Ensemble method outperforms the normal kNN, with an additional accuracy of 9.8%.***

Keywords: Predictive Analytics, K-Means++, K-Nearest-Neighbors, Randomized-KNN, Double Ensemble, Machine Learning

1. Introduction

Predictive Analytics refers to the process of using data mining to discover hidden patterns in data, and combining these patterns with business knowledge to extract value to an organization. It has been increasingly used both in the business world, as well in commercial applications, such as healthcare or the government. A key necessity for Predictive Analytics is a trend, which will allow a trained model to predict values in the future.

Predictive Analytics works in conjunction with Big Data, which is characterized by great variety, has large volume, and increasing velocity. The amount of available data has steadily increased, at an ever-higher rate, resulting in the need for both fast and efficient algorithms.

A method often used in Predictive Analytics is data clustering, which is a form of unsupervised learning, i. e. the data is not labeled. While this allows the computer to potentially find patterns in the data that may have escaped human analysis, it can also often lead to issues when clusters in the data are relatively similar. An optimal data clustering algorithm will create clusters that are different from each other, but contain similar data-points. This paper will discuss K-means and K-means++.

While data clustering can provide insight into hidden patterns within data, supervised learning is needed in order to make predictions. Supervised learning also relies on these patterns, but a key difference is that in this case the test-data is labeled, so unsupervised learning does not need to take place. Supervised learning aims to connect a new data point to the group, or data-point most similar to it. In order for data to be analyzed effectively, most data-points need pre-processing. For unstructured or semi-structured data, this primarily involves using techniques to convert the data into structured data. For instance, for text-based data, 'document frequency' is used to see how often a keyword is mentioned across documents. Feature-selection is also used to create new features, potentially through a combination of existing features, which can provide a stronger relationship with the prediction-variable.

Predictive Analytics has use-cases across all industries, but a key step is to use business knowledge for that particular domain when creating, evaluating, and optimizing models. Hence, this requires both a thorough analysis of the data, as well as an understanding of the industry for customer patterns, demographics, and more. For instance, a large-scale, unexpected event can often create an unexpected trend in the data; this was the case with the onset of the Covid-19 pandemic, which, as a black-swan event, led to the stock-market having a highly volatile period. [1]

"Twitter mood predicts the stock market" [2], published in 2011, was the first of a kind to use Predictive Analytics tools to analyze high-volume and high-noise data, such as the stock market. It used Natural Language Processing (NLP) to find the sentiment from tweets using OpinionFinder and GPOMS. It used Granger Causality to analyze the time-series data. This sparked further research into stock-prediction, with Natural Language Processing also expanding to news articles. In 2019, [3] used an Auto-Regressive-Integrated-Moving-Average model with a Recurrent Neural Network to predict the S&P-500 by training a model on historical stock data and news articles.

A common use case of the kNN is in Recommender Systems, where a new user or product is attempted to be classified into a category of user / product groups. This information can then be used to recommend users products that users in a similar group purchased. This is known as a collaborative filtering system. A key problem in collaborative filtering too is the initial choice of 'k'. The initial classification may be carried out using K-means or K-means++ through unsupervised learning. A new user may then be classified into a category using kNN. These systems often suffer from a lack of sufficient user data, a lack of business understanding, or users with a large number of interests. This may also be known as the 'cold-start' problem. A potential solution is to use content-based filtering, where attributes of the product are used to compare it to similar products, and hence recommend products to users.

Similarly, Predictive Analytics was also used by Google to build "Correlate" [4], a follow-up tool to the algorithms it built in its prediction of the flu-influenza using google-search-queries. Correlate uses a Nearest Neighbor approach using Pearson Similarity to generalize it to all search queries.

As can be seen, Predictive Analytics is used across a wide range of fields. Supervised learning, which predicts a variable from a set of data, has far-reaching scope. Hence, the development of supervised machine learning models that are fast, efficient, and accurate is highly needed. This paper creates such an algorithm, coined the "Double Ensemble kNN", which uses two Ensemble systems with kNNs as base-learners. Our research is motived by the following questions:

1) Can we enhance kNN by creating an Ensemble of learners with different k?
2) Can we enhance kNN by creating an Ensemble of learners trained on subsets of the original dataset?

We build our model in python 3.9 using the sci-kit learn library, and test it out on 9 datasets spanning a range of sizes and features. The results are compared to the traditional kNN, as well as a Single Ensemble, and a 'random-k kNN'.

2. Machine Learning Algorithms

2.1 K-Means

The K-Means algorithm is an iterative unsupervised clustering algorithm. It sorts the given data into 'k' groups, where 'k' has to be manually set beforehand.

The algorithm starts by picking 'k' initial centroids, or cluster representatives. This step is known as the "Assignment Step". The initial centroids are picked randomly. The initial centroids determine both the time taken for the algorithm to converge, as well the final result. Different initial centroids may have different results.

In the Assignment Step, all other data points are then assigned to a cluster based on the centroid they are most similar to. Similarity can be measured using several techniques, however, Euclidean distance and cosine similarity are the most common technique. Cosine Similarity is particularly effective during document-clustering.

In the recalculation step, centroids are recalculated as the average of the values of all data-points in a cluster, and the process is repeated. The algorithm continues until there is no change between two iterations, i. e. the centroids remain the same.

In Euclidean Distance, a simple distance is calculated between the data-point and the centroid: For a dataset with 'n' features, the Euclidean distance is the root of the sum of the squares of the difference between the centroid and the data-point. A smaller Euclidean Distance is better, as it indicates that the points are closer together. This can be mathematically represented as follows:

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_1)^2}$$

Cosine Similarity finds the cosine of the angle between the two data-points, when they are regarded as vectors. Because the data-points are regarded as vectors, their dot-product is divided by their scalars when multiplied. Unlike Euclidean Distance, a higher value indicates that two data-points are more similar for Cosine Similarity.

$$\cos(d_1, d_2) = \frac{(d_1 \cdot d_2)}{\|d_1\| \|d_2\|}$$

Cosine and Euclidean Similarity measures often perform relatively similarly, and are used in conjunction to provide a better overall similarity. However, Cosine may perform better with smaller 'k' values, while Euclidean may perform better for high-dimensional data. [5]

Advantages:
The K-means algorithm is one of the earliest unsupervised clustering models to be introduced, and is still relatively simple to implement. As a result, it can often be used as an 'initial measure', or when testing a dataset for clustering, as the algorithm does not heavy modification or personalization to be implemented.

Furthermore, K-means has linear time-complexity, represented in Big-O notation as O (n). While not ideal, this allows K-means to still be a relatively fast algorithm, particularly compared to Hierarchical Clustering. This is important for large datasets, which may have lots of data to be clustered. Big Data is also characterized by 'Velocity', and Predictive Analytics is often used in real-time, so efficient computation is key.

Disadvantages:
K-means algorithms are not well-suited for datasets that have a larger number of outliers, and the inclusion of outliers affects the value of the centroids. This is because all data-points are forced to be assigned to a cluster, and outliers are not calculated. Hence, pre-processing of data to remove both outliers, as well as data-points with missing values is highly important.

K-means' primary disadvantage is the need for manual selection of 'k'. This is often difficult to choose, because the number of clusters may not be known. Hence, it is difficult to predict the value for 'k', on which the algorithm so heavily relies. In order to find the optimal 'k', the algorithm may be run over several values of 'k' to find an optimum. However, this can often take more time.

The initial centroids also have a significant effect on the final result, and since they are picked randomly and can result in a range of accuracies for the algorithm. While the algorithm will converge in all cases, it will likely only reach a local minimum: i. e. the final solution will not be the ideal solution.

2.2 K-Means ++ Algorithm

While the K-means Algorithm continues to be a highly useful clustering algorithm, its primary disadvantage, as discussed above, is the effect the randomization of the picking of initial centroids can have. By leaving this initial decision up to chance, final results can often have varying results, and not be ideal.

Hence, the K-means ++ algorithm aims to improve K-means by changing the initial process. It works by choosing an initial random centroid.

Subsequently, for all other data points, the distance from that data-point to the nearest centroid is found.

A new centroid is then found using a weighted probability distribution that is proportional to the distance calculated squared. This ensures that centroids are more likely to be further from each other, so that inter-cluster distance is higher, while intra-cluster distance is lower.

The process is repeated until ‘k’ centroids are found.

Once all centroids are found, the rest of the K-means algorithm is applied. K-means++ also has several advantages and disadvantages, but has been shown to generally perform better than K-means.

A disadvantage is that the initial selection of centroids takes more time than K-means, but in general, the overall algorithm is faster. [6] first documents K-means++, finding a two-fold increase in the speed compared to K-means.

[7], compared K-means++ and K-means on a crime-domain dataset, and found that K-means++ with Cosine Similarity attained an F-measure of 0.910, while K-means with Cosine Similarity only attained 0.802.

2.3 K-Nearest-Neighbor

Explanation

The K-Nearest-Neighbor was first introduced by [8], to solve “the discrimination problem”. It is one of the earliest machine learning algorithms for supervised learning, i. e. where a new test-point has to be classified into a set of class-labels by a model trained on a data-set of training-points with features. Despite its simplicity, the kNN continues to be one of the most widely used Machine Learning algorithms, and is used across a variety of disciplines. kNN works by classifying a new test-point based on the ‘k’ closest, previously classified, data-points. The case in which k=1 is referred to as the Nearest Neighbor approach.

Similar to K-means, several dominant techniques exist for determination of the closest neighbor. This paper will primarily use Euclidean Distance, due to its simplicity and high accuracy. However, several approaches exist where a different similarity measure is used. For instance, Minkowski distance may be used for real-valued vector-spaces, i. e. distances are vectors and non-negative. Manhattan distance may also be used, which considers the absolute value difference between the Cartesian coordinates. [9] finds that the Euclidean Distance metric is best for categorical and numerical datasets, but not for mixed datasets.

It is important to standardize the features if Euclidean distance is used. If this is not done, features with a large range can have a disproportionate impact.

In Pseudocode, kNN can be described as follows:

```
function kNN { (dataset = S, data point = d, k) =>
 Calculate distance for all points in S from d.
 Sort the distances, smallest first.
 Find the k nearest data points, and their classes.
 Assign the majority class from the k nearest neighbors.
 }
```

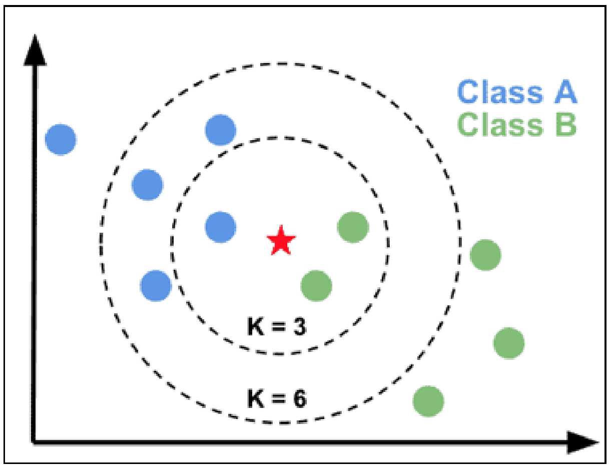

Figure 1: KNN Algorithm

As can be seen, increasing K increases the size of the ‘radius’. The dotted line indicates that distance is likely calculated using Euclidean metrics. The star is the data point to be classified. This is only a two-dimensional dataset. However, for larger datasets, the area would expand into all dimensions. This further shows the importance that the value of ‘k’ can have: For K=3, it is classified as Class B, while for K = 6, it is classified as Class A.

Furthermore, the kNN is a non-parametric model. A parametric model makes an assumption of the form of the data-relationship, on which a model is built. This limits the model, as it cannot be flexible. Furthermore, the relationship assumed may be false. On the other hand, a non-parametric model does not make any assumptions. While this makes them more flexible, it has the disadvantage of needing more data. [10]

Additionally, kNN is a “lazy learning” approach. This refers to a type of supervised learning model where the processing of training data points is carried out only when making predictions/inferences. Hence, the algorithm is not ‘trained’ per se. This is the opposite of eager learning. Lazy learning is better for Big Data, because the tendency of the data to continuously increase in size means an eager learning approach would have to be constantly retrained. However, it has the disadvantage of taking a lot of time to make a prediction. A lazy learning approach can also solve several problem simultaneously.

A Voronoi Diagram can be used to illustrate the Decision Boundary, particularly for Nearest Neighbors. The following example is taken from Raschka, Sebastian, "STAT 479: Machine Learning Lecture Notes [11]. For the case where k=1, a "decision boundary" can be established between two classes. At this decision boundary, there is a theoretical tie between the two-classes for classifying a new point. All points within a decision boundary will be classified to the training point within.

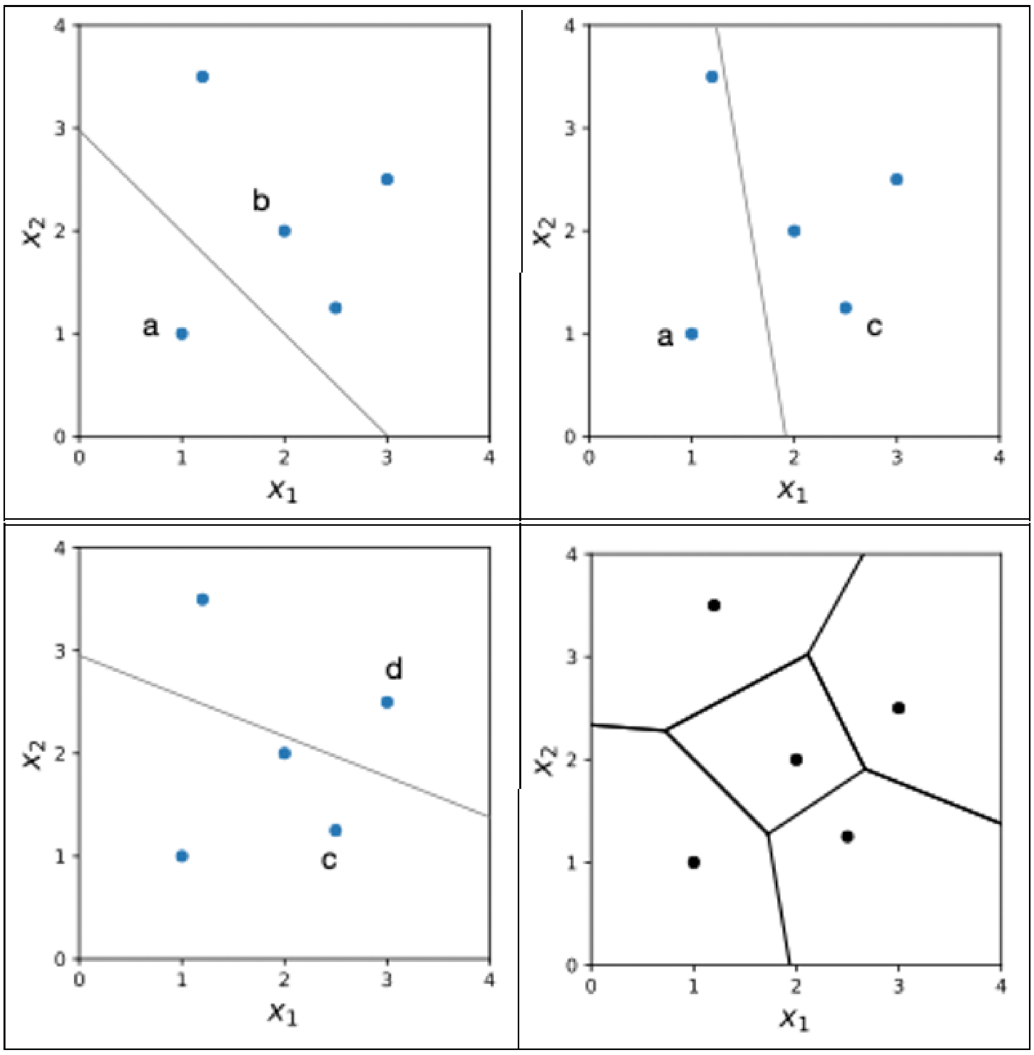

Figure 2: Voronoi Diagram

For each of these, by creating a decision boundary, one can easily see which class a new data point would be classified into. These may also be known as Voronoi-cells. Because Euclidean distance is used, a decision boundary is equidistant from two data points. A vertex is equidistant from three (or more) data points. When a class is added to each data-point, Voronoi cells can be combined to form decision boundaries for classes (as opposed to just cells). This is illustrated in the Figure below, which shows the final result of applying Voronoi tessellation using a Nearest Neighbor approach.

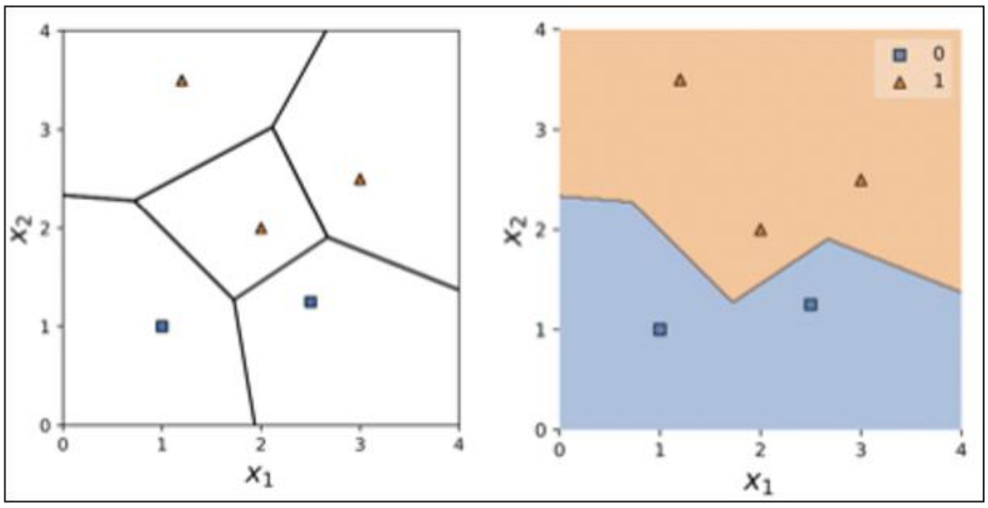

Figure 3: Voronoi Diagram part 2

Problems

There are three primary problems associated with the kNN:

1) The algorithm is relatively inefficient, because it has a time-complexity of O (n), where n is the number of training-points. This occurs because the classification of a new data-point requires the calculation of its distance to all training-points. Hence, the kNN cannot be pre-trained, and then shipped. In fact, there is no training required.

Several methods have been proposed to solve this. The Condensed Nearest Neighbor, proposed by [12], aims to solve this by removing training-data-points which are relatively similar to others. Hence, only a subset of the original training data-points are stored. However, some implementations of the model select samples to retain randomly. Hence, this may save samples that do not accurately classify test-points. This would particularly be the case if the samples are deep within a neighborhood, and not on the sample.

The reduced kNN (RNN) by [13] further works on the CNN by removing data-points that do not significantly affect the accuracy of the model. This can be known as "pruning", and involves two significant types: editing and prototype selection. In editing, a data-point is permanently removed if it does not play a significant role in classification. This is usually true of outliers. Because the data point does not play a role in classification, it can safely be removed.
In prototyping, several data points can be combined to a single data point. This is similar to K-means, where a centroid represents a class.

Research has also been conducted into improving the classification time for the kNN by using different techniques, such as hashing. [14] create a method for the kNN with time-complexity O (dlogn), using locally-sensitive hashing, whereby "the probability of collision is much higher for objects which are close to each other than those which are far apart".

2) The second problem associated with the kNN is the selection of 'k'. The intuitive technique of 'brute-forcing' the optimal value for 'k', i. e. testing out models with a range of values of k on a dataset and then choosing the most accurate value, is highly inefficient, as it would take a lot of time. This would be particularly ineffective for larger datasets.

The upper-bound for k is usually set at the square root of the number of samples. However, since data sets increasingly have a large number of samples, brute-forcing is not an effective technique.

Similarly, an optimal k would still be a local optimum: there exists the possibility of there being a different optimal value of k for each dataset. Selecting these would not be plausible, since for a real-world application, the class of a test-point will not be known. Hence, there exists the need for an accurate method to choose k.

A k that is too small may result in noisy data having a larger impact, while a larger k would result in the algorithm being affected by data points not in the data-class / from another cluster. This is also why k is capped at the square root of n: a k that is larger would likely simply lead to the data-point being primarily affected by clusters that have a large number of data points. If k = n, any new data-point would be classified into the largest class.

3) The "curse of dimensionality" refers to high-dimensional data with a small training size, also known as the "small N, large P" problem. This is a significant problem for the kNN because the volume of hyperspace that must be captured increases. Hence, the neighbors become more dissimilar, in both practice and theory. This is because they are now further apart in several dimensions from the test-point.

2.4 Random KNN

Several papers have investigated kNN Ensembles: i. e. a group of sub-kNNs that act together to form a larger model, based on a form of voting. Ensemble techniques have included bagging, boosting, and more. A large variety of Ensemble techniques surrounding the kNN have been shown to outperform both the base kNN, as well as other models.

To begin with, [15] proposes Random KNN feature selection as an alternative to Random Forests, which work on bagging.

It applies this for “small n, large p” problems, which refers to datasets where there are a large number of features, but limited number of data-points to train models on. This may also be known as high-dimensional data, where dimensions refer to the features. Methods used to address ‘small n, large p’ problems focus on pre-processing of the data-set to reduce noisy data and features which have minimal impact, so that the model can be trained on a limited number of features. By reducing the number of features, models can better understand each feature, with the “samples per feature” ratio increasing.

In their paper, Li. et al, [15] create a random KNN by creating a set of ‘r’ kNNs, trained on r subsets of the input features. This acts as a method to address high-dimensional data, as individual kNNs have a high samples-per-feature ratio, while the overall algorithm is trained on all features. The final classification of the model is done by majority vote of the individual kNNs. This works particularly well on datasets that have a large number of features, as these would allow for a large subset of individual kNNs to be formed, which can be trained on different variables.

However, this does not solve the problem of redundant features being removed. They are equally likely to be chosen, and affect the final classification. A potential method to avoid this may be using cross-validation to create a weighted Ensemble.

On their comparison of 12 datasets, on average, a Random Forest attained an accuracy of 83.0%, a R-1NN an accuracy of 89.9%, and a R-3NN an accuracy of 87.0%. This shows that the R-1NN performed best (and had a lower standard deviation too).

This is contrary to Breiman’s prediction of bagging and bagging-like methods not being highly effective on stable classifiers such as the kNN. [16] states that the “vital element is the instability of the prediction method”. This refers to the impact that changing the data-set can have on the base-model. A model that is highly sensitive to the base-dataset would likely have a higher improvement when bagging is applied to it. In fact, [16] further states that bagging can “slightly degrade the performance of stable procedures”

Bagging, introduced by [16], is a method to create Ensemble Learners by Bootstrapping and Aggerating. Two main methods to combine individual learners are proposed to receive the final classification: for labeled data-sets, voting is suggested, while for numerical data, averaging is suggested. Breiman further showed that accuracy tangents after 25 sub-learners, with more sub-learners not further improving the performance.

[17] applies bagging to a kNN, similar to the method used by [15]. However, a key difference is that they focus on small datasets, because the kNN would be more unstable in these. Hence, bagging is approached from improving an unstable classifier, as opposed to the prior, which used it to reduce high-dimensional data.

K is set to the square root of N, and 25 base-learners are used. They find that for small data-sets, there is minimal improvement when applying bagging to a kNN. In fact, it performed worse than an individual kNN for a data-set of size 12. However, for data-sets that are medium in size, the bagged kNN offered slight accuracy improvements, in the range ~ (0.01-0.06). Furthermore, it was shown that voting was the best form of combining base-learners for a final classification.

[18] constructs an ensemble of kNNs, where a subset of features is used to train individual kNNs. However, only a subsection of trained kNNs are chosen, based on validation from part of the training set. kNNs that have an accuracy above the upper quartile are chosen to be in the Ensemble. The benchmark value for accuracy can be changed based on the wanted number of final kNNs in the ensemble. When compared to other models, it is found that their model, ES*k*NN, is most accurate on 8 datasets, and the Random Forest is most accurate for 9 datasets. The random-KNN is only most accurate for 1 dataset, which suggests that data-set selection can play a large role, with a random-KNN working well only on specific types of datasets.

3. Methodology

While much work has been conducted on a subset of kNNs that use a subset of the training data, relatively less research has been probed into the construction of a kNN that uses base-learners of kNNs with different ‘k’s. This has the potential to provide a more robust structure for kNNs that removes Problem II, the problem of choosing an appropriate k value.

Several different Ensemble methods are created and analyzed in this paper.

3.1 Random-k kNN

The first is the Random-k KNN, where the value of ‘k’ is randomly chosen between the range of $(1, \sqrt{N})$. This is carried out within an ensemble of 10 KNNs. An ensemble method is applied here because a randomly chosen k would naturally not return a high accuracy, as evidenced by research previously conducted into optimal ‘k’s. However, by choosing ‘k’ randomly for an Ensemble, the likely effect may be that KNNs of varying sizes will develop to avoid being disproportionately affected by noisy data or large classes.

Pseudocode:

```
Function Random-k kNN {
for counter in range (0, 10):
k = random integer between 1 and $\sqrt{N}$
create kNN
predict Class
majority vote on Class
}
```

This is a relatively novel technique of a k-NN ensemble, and most prior research into selection of ‘k’ has chosen to use a mathematical-optimization, or brute-forcing approach to select the optimal ‘k’. This approach has several advantages, including having faster selection of ‘k’, since brute-forcing requires testing on all ‘k’ values, which, particularly for larger datasets, can represent significant computational effort.

Furthermore, the randomized selection of ‘k’ for kNNs within an Ensemble may lead to the avoidance of bias. Noisy data continues to represent a key struggle in Big Data, particularly as data continues to grow and become noisier. An ensemble with randomized 'k' could bypass highly inaccurate results due to noisy data by 'regressing to the mean', i. e. being affected by noisy data from several classes, such that the final result would be closer to the true value.

However, a randomized 'k' does not provide a significant mathematical suggestion for improved performance. The use of a k closer to $\sqrt{N}$ may allow for a better overall suited kNN, and the creation of an Ensemble may simply increase computational complexity. If errors from individual kNNs are too large, they will ultimately affect the final result. Hence, it is important that errors from individual base-learners are canceled, not compounded.

3.2 Double Ensemble

Furthermore, a Double-Ensemble is created: The training set is divided into randomly chosen sub-training sets made out of N rows and M features, where N is the original number of rows and M=5, either 5, 10, 15, 20 times, and for each sub-training set, an Ensemble of KNNs for k={1 to 5} is created. This Double Ensemble aims to use random feature-selection to avoid overfitting on a specific class, while several KNNs are used to further gauge the data from multiple angles.

Pseudocode:

```
Function Double-Ensemble { Num1 = [5.10, 15, 20]
For counter in range (0, Num1):
Select subset of Dataset randomly. (of size features=5)
For k in range (1, 5):
Train kNN (for k)
Predict *Class*
Majority vote on *Class*
}
```

While this model represents far higher complexity, both algorithmically and computationally, than the traditional kNN, it has the potential to solve several issues that currently plague the kNN.

Being disproportionately affected by a certain feature can be solved through feature-selection. However, as a model that often does not undergo pre-processing, the option of choosing a subset of features represents a potential alternative to feature-selection: by choosing features randomly, a certain feature's potentially misleading relationship with Y (i. e. the class to be predicted) could be diminished.

Creating the second Ensemble further allows the model to minimize being affected by outliers, while points closer to the dataset are given more importance. This occurs because the point closest to the data-point is considered 5 times, the point second-closest 4 times, and so on. Hence, a weakness of the traditional kNN, the lack of weighting for points closer to the test-point, is also addressed.

3.2 Single Ensemble

Additionally, a Single Ensemble is created similar to the Double-Ensemble, except the second Ensemble is avoided: The model creates a subset of training data sets chosen randomly, on which a kNN for k=1 is applied.

Pseudocode:

```
Function Double-Ensemble { Num1 = [5.10, 15, 20]
For counter in range (0, Num1):
Select subset of Dataset randomly (of size features = 5)
Train kNN (k=1)
Predict *Class*
Majority vote on *Class*
}
```

This Ensemble is built similar to the Double Ensemble, except the second Ensemble is replaced with a singular kNN with k=1. By noting the differences in performance between both algorithms, an assessment of whether the second Ensemble helps will be established. Even if there are minor improvements through the Double Ensemble, there is a trade-off with the added time needed to compute the second Ensemble.

4. Datasets

To have a well-defined and generalized conclusion, we test our models on several datasets of various sizes and types. This has several practical advantages, including finding certain scenarios in which certain models are better, finding potential flaws in models, and gaining a deeper understanding of the improvement a certain model offers.

8 Datasets are used in this paper, including multi-and binary-classification. All datasets used are open-source, and links to their sources are included in the References section. The datasets span a variety of fields, sizes, and types.

Table 1: Dataset Description

Dataset Name	Total Number of Samples	Number of Features
Credit	690	14
Diabetes	768	8
Breast Cancer	699	9
Ionosphere	351	34
Ecoli	336	7
Wine	178	13
Iris	150	4
Lung Cancer	32	56

As can be seen, the datasets have a variety of sizes, however, Lung Cancer is by far the smallest, with only 32 data-samples. This represents a common "small N, large P" problem, because it has 56 features, which represents a feature-sample ratio of 7: 4. Hence, it might be likely that the randomized Ensembles perform better on this dataset. The limited number of data-samples might act as a bottleneck in testing, limiting the testing sample to only 6 samples. (All datasets use a test-sample size of 20% of the total data). Hence, even if the Ensemble models are better optimized to perform, we may not be able to distinguish them with other models due to the limited availability to assess them.

Furthermore, datasets with a smaller number of features may be better suited to individual kNNs, because the use of a randomized sub-set of data might result in kNNs being trained on the same dataset again. This could lead to randomized bias towards a particular data point if it was selected twice (or more).

Because of this reason, it is likely that the Ensembles will perform better on Datasets with both a higher number of Samples and a higher number of Features; particularly, it will be interesting to note how the models perform on the Credit and Ionosphere datasets.

Lastly, the datasets used here have been frequently used in the academic community to test models, and several pre-processing tasks have been implemented to optimize machine learning models. However, to provide a general overview of the models, as well as assess how well they perform on raw data, without pre-processing techniques such as Feature-selection of Hyper-parameter-selection, this paper has not used any pre-processing techniques; this excludes cleaning data, and filling *null* values with the mean of the column.

5. Results

All algorithms were tested on the Dataset mentioned, as well as three traditional kNNs for k = 1, 3 and $\sqrt{N}$ to test how large the improvements in accuracy for the models were. The following table summarizes the results. Accuracy was calculated using the following formula:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions} * 100\%$$

Table 2: Experimental Results

Dataset	KNN			Random KNN	Double Ensemble				Single Ensemble				Max Accuracy	Max Accuracy Model
	1	3	$\sqrt{N}$		5	10	15	20	5	10	15	20		
Credit	72%	67%	67%	67%	70%	70%	78%	79%	68%	72%	81%	77%	81%	SE-15
Diabetes	68%	65%	68%	68%	66%	69%	68%	71%	67%	69%	69%	69%	71%	DE-20
Breast Cancer	78%	76%	78%	78%	94%	96%	97%	96%	95%	94%	95%	96%	97%	DE-15
Ionosphere	82%	83%	82%	82%	82%	92%	92%	92%	93%	90%	**94%**	90%	94%	SE-15
Ecoli	84%	**87%**	84%	63%	56%	60%	60%	59%	54%	46%	53%	57%	87%	3-KNN
Wine	95%	94%	95%	95%	96%	**98%**	**98%**	**98%**	96%	**98%**	**98%**	**98%**	98%	DE, SE-[10, 15, 20]
Iris	**93%**	**93%**	**93%**	67%	67%	67%	67%	67%	67%	67%	67%	67%	93%	1, 3, sqrt (N)-KNN
Lung Cancer	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	**92%**	92%	All

To begin with, it can be seen that in 5 out of 8 Datasets, the Double Ensemble or Single Ensemble outperforms the traditional kNN. Furthermore, this is often by a significant margin, on average, 9.8%. This is a relatively high difference, and further shows that the Ensemble techniques outperform the traditional kNN for several types of datasets.

It is also noted that in both the Double Ensemble and the Single Ensemble, a larger value for the base-learners is associated with a generally larger accuracy. However, this is not a significant linear trend, but rather, a generalized one. The key observation is that there is often a significant difference in accuracy between the number of base-learners from 5 to 10, and a resultant diminished increase. Hence, it is suggested that an optimal value for the base-learns in both Double and Single Ensemble be set to 10.

Interestingly, the Random-k kNN performs rather poorly. While it underperforms with the Ensembles, it performs on-par with the traditional kNN. Hence, the random KNN is likely not a fit option, and will have probably 'compounded' the individual errors.

The Lung Cancer dataset had all models perform with a 92% accuracy, which is likely due to all models predicting the same set of classes. The likely cause behind this relatively rare event would be a lack of training data and limited testing data, leading to the same answers, as randomization would not have enough space to have significant effects.

The Iris and Ecoli datasets act as exceptions, with the traditional kNNs outperforming all Ensembles, and by a high margin. A further investigation into these datasets might likely yield a reason for this. However, a likely potential cause could be a high number of outliers, as well as all features playing a significant role in classification-determination.

On average, the Double Ensemble outperforms the Single Ensemble. However, the margin of error is relatively high, because the outperformance is minimal. Hence, while the Double Ensemble outperforms the Single Ensemble, it is recommended that the Single Ensemble be used, because of the trade-off between accuracy and prediction-time.

6. Double and Single Ensemble – Analysis

In response to the data results, it is clear that the Double and Single Ensemble can regularly outperform the traditional kNN. However, a thorough analysis of their advantages and disadvantages is needed to assess their implications in a real-world setting.

Advantages

A key advantage of the Double and Single Ensemble are their reduction of bias. Bias refers to when a machine learning model makes systematic errors, i. e. has a lower accuracy, because of inaccurate trend-identification in the data. This particularly occurs when a model underfits, which may occur if there is a lack of many data points. The Ensembles reduce bias by not being trained on all data-features.

Furthermore, they reduce variance, which refers to how deeply a change in the training data affects the model's prediction. This is particularly important for data-sets where the training data may not be fully accurate, or have many missing values. The models reduce variance, like many other Ensemble models, by not being trained on the complete dataset.

While several effective Ensemble Machine Learning models exist, the kNN double and single Ensemble create a highly applicable instance, through a combination of the relative simplicity of kNNs and error-reducing ensemble methods, reducing some of the kNN's major disadvantages.

For instance, when compared to a Random Forest, which applies bagging to Decision Trees, an Ensemble k-NN as created in this paper might prove more useful for recommendation engines. This is because a kNN works by creating user-neighborhoods, while a Random Forest, as an eager learner, attempts to understand and build patterns. These might be unnecessarily complicated, and potentially inaccurate. By using an ensemble, a user may also be mapped to better sub-categories, i. e. areas where neighborhoods overlap due to shared interests. Experimentally evaluating this hypothesis might be highly useful for the development of recommendation engines.

A key advantage of the Double Ensemble over the Single Ensemble is weighting the nearest neighbor: this is a problem with the traditional kNN too, which does not place greater importance on neighbors closer to the test-point, but rather, values all k-nearest-neighbors equally. Since the double Ensemble considers the closer neighbors more often, their weightage increases.

The total number of neighbors considered is, where T is the number of base-learners.

$$\frac{(T)(T+1)}{2}$$

It is important to note that T = k_{max}, i. e. the furthest neighbor considered. If T =5, the 5th base-learner will consider the 5 nearest neighbors, while the 4th base learner will only consider the 4 nearest neighbor. Hence, the Nth base-learner will consider the N-nearest-neighbor.

The number of instances of the nth neighbor is: $T+1-n$

Hence, to find the weightage, the number of instances is divided by the total number of instances. Therefore:

$$W(n)=\frac{2(T+1-n)}{(T)\ (T+1)}$$

The below graph illustrates that the weight decreases linearly. This is because T is a constant, while 'n' increases. In this specific example, 'T' is taken as 3.

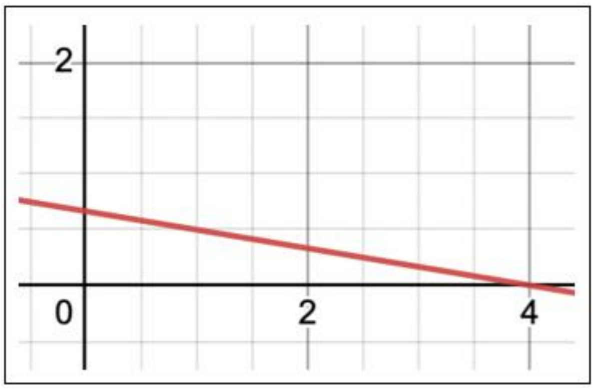

Figure 4: Demonstration of Weighting

Table 3: Sample Weight Calculation

'n' value	Weightage
1	0.5
2	0.333
3	0.167
Total	1

This represents an advantage over the traditional kNN and Single-Ensemble kNN, where each neighbor is given a weightage of 1/k.

Disadvantages

A key disadvantage of kNNs is the high training-time, because the model has to parse all data-points prior to making a prediction, as a lazy-learner. By creating an Ensemble approach, the computational complexity increases multi-fold, relatively by the number of base-learners created. This effect is squared for the Double-Ensemble, since there are two Ensemble methods within each other.

Furthermore, optimizing the Ensemble kNNs would be harder due to the introduction of a random effect, which removes the kNN's traditional stability: the same data point is always classified into the same class. Because kNNs may be used for medical applications, or other important fields, a lack of stability can be a major drawback, and create uncertainty. Finding the optimal 'k', or number of base-learners, would also be harder.

Lastly, the data results showed that datasets with a small number of features (such as the Iris Dataset) are not well suited for the Ensemble kNNs, because the randomization of data by splitting features does not effectively work.

7. Conclusion

This paper has investigated the design and applications of a novel kNN model, coined the 'Double Ensemble kNN', built on top of a 'Single Ensemble kNN' that uses randomized feature-splitting to increase accuracy and take advantage of the kNNs traditional stability and Ensemble methods reduction of bias and variance to create a net model that performs more accurately than the traditional kNN.

While the Ensemble methods have several disadvantages, the higher accuracies, combined with other advantages, provide a defining case for their use in real-world applications, particularly datasets that contain a large number of features and data points.

Further research may be conducted on the selection of an optimal number of base-learners for both Ensembles, that provides a midpoint for the tradeoff between accuracy and computational-time. This might push the model closer to Bayes Error, which is regarded as the lowest possible error rate that can be traditionally achieved.

References

[1] Ning Zhang, Aiqun Wang, Naveed-Ul-Haq & Safia Nosheen (2022) The impact of COVID-19 shocks on the volatility of stock markets in technologically advanced countries, Economic Research-Ekonomska Istraživanja, 35: 1, 2191-2216, DOI: 10.1080/1331677X.2021.1936112

[2] Bollen et al., (2010), *Twitter Mood Predicts the Stock Market, https: //arxiv. org/pdf/1010.3003. pdf*

[3] Mohan, Saloni; Mullapudi, Sahitya; Sammeta, Sudheer; Vijayvergia, Parag; Anastasiu, David C. (2019). *Stock Price Prediction Using News Sentiment Analysis,* doi: 10.1109/BigDataService.2019.00035

[4] Mohebbi et al, (2011), *Google Correlate Whitepaper, https: //static. googleusercontent. com/media/research. google. com/en//pubs/archive/41695. pdf*

[5] Baisantry, M., & Shukla, D., (2017), *COMPARISON OF DIFFERENT SIMILARITY MEASURES FOR SELECTION OF OPTIMAL, INFORMATION-CENTRIC BANDS OF HYPERSPECTRAL IMAGES, https: //www.asprs. org/wp-content/uploads/2018/04/Baisantry_M. pdf*

[6] Arthur, D., & Vassilvitskii, S., *k-means++: the advantages of careful seeding, https: //theory. stanford. edu/~sergei/papers/kMeansPP-soda. pdf*

[7] Aubaidan et al., (2014), *COMPARATIVE STUDY OF K-MEANS AND K-MEANS++ CLUSTERING ALGORITHMS ON CRIME DOMAIN, https: //thescipub. com/pdf/jcssp.2014.1197.1206. pdf*

[8] E. Fix and J. L. Hodges (1951): *An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation* https: //www.jstor. org/stable/1403796

[9] Hu, Li-Yu; Huang, Min-Wei; Ke, Shih-Wen; Tsai, Chih-Fong (2016). *The distance function effect on k-nearest neighbor classification for medical datasets.* SpringerPlus, 5 (1), 1304–. doi: 10.1186/s40064-016-2941-7

[10] Ahmed, M.,, (2021), *Parametric and Non-parametric Machine Learning Algorithm,* https: //faun. pub/parametric-and-nonparametric-machine-learning-algorithm-6134d7155cc

[11] Raschka, S. (2018), *STAT 479: Machine Learning Lecture Notes* https: //sebastianraschka. com/pdf/lecture-notes/stat479fs18/02_knn_notes. pdf

[12] Hart, P. (1968). *The condensed nearest neighbor rule (Corresp.)., 14 (3), 515–516.* doi: 10.1109/tit.1968.1054155

[13] G. Gates.: The Reduced Nearest Neighbour Rule. IEEE Transactions onInformation Theory, 18, 431-433, (1972)

[14] Har-peled et al., (2012), *Approximate Nearest Neighbor: Towards removing the curse of dimensionality, https: //theoryofcomputing. org/articles/v008a014/v008a014. pdf*

[15] Li, S., Harner, E. J. & Adjeroh, D. A. Random KNN feature selection-a fast and stable alternative to Random Forests. *BMC Bioinformatics* 12, 450 (2011). https: //doi. org/10.1186/1471-2105-12-450

[16] Breiman, L. Bagging predictors. *Mach Learn* 24, 123–140 (1996). https: //doi. org/10.1007/BF00058655

[17] AlBaghdadi, Amer & Alkoot, Fuad. (2005*). Bagging KNN Classifiers using Different Expert Fusion Strategies.* .219-224.

[18] Gul et al., (2014), *Ensemble of a subset of kNN Classifiers, https: //link. springer. com/content/pdf/10.1007/s11634-015-0227-5. pdf*

[19] R. D. KING, C. FENG & A. SUTHERLAND (1995) STATLOG: *COMPARISON OF CLASSIFICATION ALGORITHMS ON LARGE REAL-WORLD PROBLEMS*, Applied Artificial Intelligence, 9: 3, 289-333, DOI: 10.1080/08839519508945477

[20] Michie, D. & Spiegelhalter, D. & Taylor, Charles. (1999). *Machine Learning, Neural and Statistical Classification.* Technometrics.37.10.2307/1269742.

[21] Ha, Sang & Nam, Nguyen & Nhan, Nguyen. (2016). *A Novel Credit Scoring Prediction Model based on Feature Selection Approach and Parallel Random Forest.* Indian Journal of Science and Technology.9.10.17485/ijst/2016/v9i20/92299.

[22] Bouaguel, Waad & Mufti, Ghazi & Limam, Mohamed. (2013). *A three-stage feature selection using quadratic programming for credit scoring.* Applied Artificial Intelligence.27.721-742.10.1080/08839514.2013.823327.

[23] García-Pedrajas, N., & Ortiz-Boyer, D. (2009), *Boosting k-nearest neighbor classifier by means of input space projection, http: //www.stat. yale. edu/~lc436/papers/KNN/Garcia-Pedrajas_2009. pdf*

[24] Ebrahimpour, H., & Kouzani, A., *Face Recognition using Bagging KNN, http: //users. cecs. anu. edu. au/~ramtin/ICSPCS/ICSPCS%2707/papers/198. pdf*

[25] *Farrelly, C., KNN Ensembles for Tweedie Regression: The Power of Multiscale Neighborhoods, https: //arxiv. org/pdf/1708.02122. pdf*

[26] Domeniconi, C., & Yan., B., (2004), *Nearest Neighbor Ensemble, https: //citeseerx. ist. psu. edu/viewdoc/download?doi=10.1.1.331.2665&rep=re p1&type=pdf*

[27] S. Shi, Y. Liu, Y. Huang, S. Zhu &Y. Liu, (2008) *Active Learning for kNN Based on Bagging Features,* 2008 Fourth International Conference on Natural Computation, 2008, pp.61-64, *doi: 10.1109/ICNC.2008.868.*

[28] Yu, Q., & Lendasse, A., (2009) *Ensemble KNNs for Bankruptcy Prediction, https: //www.researchgate. net/publication/255669581_Ensemble_KNNs_for_Ban kruptcy_Prediction*

[29] Uddin, S., et al., (2022), Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction, https: //www.nature. com/articles/s41598-022-10358-x. pdf?origin=ppub

[30] Tahir, A., Morison, G., Skelton, D. A. *et al.* (2020) *A Novel Functional Link Network Stacking Ensemble with Fractal Features for Multichannel Fall Detection*. *Cogn Comput* 12, 1024–1042 (2020). *https: //doi. org/10.1007/s12559-020-09749-x*

[31] Hassasnat, A., et al., (2014), Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach, https: //arxiv. org/pdf/1409.0919. pdf

[32] Gowda, K.; Krishna, G. (1979). *The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (Corresp.)., 25 (4), 488–490.* doi: 10.1109/tit.1979.1056066

Appendix 1 – Dataset Sources:

Credit: https: //archive. ics. uci. edu/ml/datasets/default+of+credit+card+clients
Diabetes: https: //www.kaggle. com/datasets/uciml/pima-indians-diabetes-database
Breast Cancer Wisconsin: https: //archive. ics. uci. edu/ml/datasets/breast+cancer+wisconsin+ (diagnostic)
Ionosphere: https: //archive. ics. uci. edu/ml/datasets/ionosphere
Ecoli: https: //archive. ics. uci. edu/ml/datasets/ecoli
Wine: https: //archive. ics. uci. edu/ml/datasets/wine
Iris: https: //www.kaggle. com/datasets/uciml/iris
Lung https: //archive. ics. uci. edu/ml/datasets/lung+cancer

Appendix 2– Python Code

```
from sklearn. neighbors import KNeighborsClassifier
from sklearn. model_selection import train_test_split
from sklearn. datasets import load_iris
from sklearn. metrics import accuracy_score
import pandas as pd
import numpy as np
import matplotlib. pyplot as plt
from sklearn. preprocessing import LabelEncoder
from sklearn. preprocessing import MinMaxScaler
import math
import random
from sklearn import datasets

class RunTest:
 def __init__ (self):
 self. num_columns = 1

 def compile (self):
 X_train, X_test, y_train, y_test = self. Ionosphere ()
 self. normal_KNN (X_train, X_test, y_train, y_test)
 self. random_K (X_train, X_test, y_train, y_test)
 self. DoubleEnsemble (X_train, X_test, y_train, y_test)
 self. SingleEnsemble (X_train, X_test, y_train, y_test)

 def LungCancer (self):
 df = pd. read_csv ('/Users/dhruvroongta/Downloads/lung_cancer_examples. csv')
 df = df. drop (['Name', 'Surname'], axis=1)
 X = df. drop ("Result", axis=1)
 y = df ['Result']
 X = X. fillna (0)
 X_train, X_test, y_train, y_test = train_test_split (
 X, y, test_size=0.2, random_state=42)
 return (X_train, X_test, y_train, y_test)
 def Iris (self):
 dataset = datasets. load_iris ()
 X, y = dataset. data, dataset. target
 #X = X [: , [0, 2]]
 X_train, X_test, y_train, y_test = train_test_split (
 X, y, stratify=y, test_size=0.7, random_state=42
)
 return (X_train, X_test, y_train, y_test)
 def Wine (self):
```

```
df = pd. read_csv ('/Users/dhruvroongta/Downloads/winequalityN. csv')
df ['type'] = df ['type']. astype ("category"). cat. codes
X = df. drop ("type", axis=1)
y = df ['type']
X = X. fillna (0)
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)
return (X_train, X_test, y_train, y_test)
def Ecoli (self):
df = pd. read_csv ('/Users/dhruvroongta/Downloads/ecoli. csv')
le = LabelEncoder ()
le. fit (df ["SITE"])
df ["SITE"] = le. transform (df ["SITE"])
df = df. drop (columns= ["SEQUENCE_NAME"])
y = df. iloc [: , 7]
X = df. iloc [: , 0: 6]
X = X. fillna (0)
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)
return (X_train, X_test, y_train, y_test)
def LiverData (self):
df = pd. read_csv ('/Users/dhruvroongta/Downloads/indian_liver_patient. csv')
df ["Gender"] = df ["Gender"]. map ({"Male": 0, "Female": 1})
indices_to_keep = ~df. isin ([np. nan, np. inf,-np. inf]). any (1)
df = df [indices_to_keep]. astype (np. float64)
df = df. dropna ()
y = df ["Dataset"]
y = y. map ({"1": 0, "2": 1})

print (df)
X = df
X = X. drop (columns= ["Dataset"])
X = X. fillna (0)
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)
return (X_train, X_test, y_train, y_test)
def Ionosphere (self):
df = pd. read_csv ('/Users/dhruvroongta/Downloads/ionosphere_data_kaggle. csv')
y = df ["label"]
y = y. map ({"g": 1, "b": 0})
X = df
X = X. drop (columns= ["label"])
X = X. fillna (0)
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)
return (X_train, X_test, y_train, y_test)
def WisconsinBreastCancer (self):

df = pd. read_csv ('/Users/dhruvroongta/Downloads/BreastCancerWisconsin. csv')
y = df ["diagnosis"]
y = y. map ({"M": 1, "B": 0})
X = df
X = X. drop (columns= ["diagnosis"])
X = X. fillna (0)
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)

return (X_train, X_test, y_train, y_test)
def IndianPrimaDiabetes (self):
df = pd. read_csv ('/Users/dhruvroongta/Downloads/diabetes. csv')
df = df. replace ("?", np. NaN)
df = df. fillna (df. mean ())
X = df. drop ('Outcome', axis=1)
```

```
y = df ['Outcome']
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)
return (X_train, X_test, y_train, y_test)
def creditloading (self):
df = pd. read_csv ('/Users/dhruvroongta/Downloads/cc_approvals. csv')

df = df. replace ("?", np. NaN)
df = df. fillna (df. mean ())
for col in df. columns:
# Check if the column is of object type
if df [col]. dtypes == 'object':
# Impute with the most frequent value
df [col] = df [col]. fillna (df [col]. value_counts (). index [0])
le = LabelEncoder ()
self. num_columns = int (len (df. columns))
for col in df. columns:
# Compare if the dtype is object
if df [col]. dtype=='object':
# Use LabelEncoder to do the numeric transformation
df [col]=le. fit_transform (df [col])


df = df. drop ([df. columns [10], df. columns [13]], axis=1)
df = df. values
X, y = df [: , 0: 13], df [: , 13]
scaler = MinMaxScaler (feature_range= (0, 1))
rescaledX = scaler. fit_transform (X)
X_train, X_test, y_train, y_test = train_test_split (
X, y, test_size=0.2, random_state=42)
return (X_train, X_test, y_train, y_test)

def normal_KNN (self, X_train, X_test, y_train, y_test):
knn_normal = KNeighborsClassifier (n_neighbors=1)
knn_normal. fit (X_train, y_train)
y_pred_normal = (knn_normal. predict (X_test))
print ("KNN-1: ", accuracy_score (y_test, y_pred_normal))

knn_normal = KNeighborsClassifier (n_neighbors=3)
knn_normal. fit (X_train, y_train)
y_pred_normal = (knn_normal. predict (X_test))
print ("KNN-3: ", accuracy_score (y_test, y_pred_normal))

knn_normal = KNeighborsClassifier (n_neighbors=int (math. sqrt (self. num_columns)))
knn_normal. fit (X_train, y_train)
y_pred_normal = (knn_normal. predict (X_test))
print ("KNN-Sqrt (n) = ", int (math. sqrt (self. num_columns)), " ", accuracy_score (y_test, y_pred_normal))

def random_K (self, X_train, X_test, y_train, y_test):
overallArr = []
for counter in range (1, 10):
knn = KNeighborsClassifier (n_neighbors=random. randint (1, int (math. sqrt (self. num_columns))))
knn. fit (X_train, y_train)
y_pred = (knn. predict (X_test))
overallArr. append (list (y_pred))
results = pd. DataFrame (overallArr, columns=list (range (0, len (y_test))))
finalResults = []
for column in results:
sum = (results [column]. sum ())
if sum >= (len (results. index)) / 2:
finalResults. append (1)
else:
finalResults. append (0)
```

```
acc = (accuracy_score (y_test, finalResults))
print ("Random-K: ", acc)

def SingleEnsemble (self, X_train, X_test, y_train, y_test):
results = pd. DataFrame ()
xpoints, ypoints = [], []
for num1 in range (1, 5):
overallArr = []
num2 = 5
# Actual r-knn starts here
for counter in range (0, 5 * num1):
X_train_sub = pd. DataFrame (X_train). sample (n=4, axis='columns')
X_test_sub = pd. DataFrame (X_test) [X_train_sub. columns]
y_predArr = []
k = 1
knn = KNeighborsClassifier (n_neighbors=k)
knn. fit (X_train_sub, y_train)

y_pred = (knn. predict (X_test_sub))
y_predArr. append (y_pred)
overallArr. append (list (y_pred))

results = pd. DataFrame (overallArr, columns=list (range (0, len (y_test))))
finalResults = []
for column in results:
sum = (results [column]. sum ())
if sum >= (len (results. index)) / 2:
finalResults. append (1)
else:
finalResults. append (0)
acc = (accuracy_score (y_test, finalResults))
print ("Single-Ensemble: ", 5 * num1, " ", acc)
xpoints. append (num1)
ypoints. append (acc)

def DoubleEnsemble (self, X_train, X_test, y_train, y_test):
results = pd. DataFrame ()
xpoints, ypoints = [], []
for num1 in range (1, 5):
overallArr = []
num2 = 5
#Actual r-knn starts here
for counter in range (0, 5*num1):
X_train_sub = pd. DataFrame (X_train). sample (n = 4, axis = 'columns')
X_test_sub = pd. DataFrame (X_test) [X_train_sub. columns]
y_predArr = []

for k in range (1, 1+num2):
knn = KNeighborsClassifier (n_neighbors=k)
knn. fit (X_train_sub, y_train)

y_pred = (knn. predict (X_test_sub))
y_predArr. append (y_pred)
overallArr. append (list (y_pred))

results = pd. DataFrame (overallArr, columns =list (range (0, len (y_test))))
finalResults = []
for column in results:
sum = (results [column]. sum ())
if sum >= (len (results. index)) /2:
finalResults. append (1)
else:
finalResults. append (0)
```

```
 acc = (accuracy_score (y_test, finalResults))
 print ("Double-Ensemble: ", 5*num1, " ", acc)
 xpoints. append (num1)
 ypoints. append (acc)




to_run = RunTest ()
to_run. compile ()
```