Impact of Gaussian Noise on Machine Learning Models

Harrison Tolley, 47447842

# Introduction

## Background

For this report, we will be using the provided Drosophila dataset to implement and investigate machine learning models/algorithms/techniques. This dataset came from a study which investigated variation in body size and life history traits in Drosophila aldrichi and Drosophila buzzatii in eastern Australia published in July 2000. This dataset provides a rich source of information for exploring the robustness of various machine learning models trained with noisy data.

## Research Question

The research question formulated for this report is:

Can other machine learning models outperform neural networks in this noisy data context?

## Objectives

The objective of this study is to compare the performance of different machine learning models when trained on data with various levels of Gaussian noise. Specifically, we will:

* Add Gaussian noise to the dataset at levels of 1%, 2.5%, 5%, 10%, and 20%.
* Train several machine learning models on this noisy data. These models include support vector machines, neural networks, decision trees, random forests, and k-nearest neighbours.
* Evaluate and compare the performance of these models using relevant performance metrics.
* Analyse the robustness of each model to noise and discuss our findings.

By systematically investigating these aspects, we aim to determine which model/s are more robust and resilient to noise and provide insight into improving model performance when trained on noisy data.

# Data Description and Preprocessing

With the provided dataset, we have 3 comma-separated value sheets. The first 5 rows of each datasheet can be seen in the following images:

A screen shot of a computer

Description automatically generated*83\_Loeschcke\_et\_al\_2000\_Thorax\_&\_wing\_traits\_lab pops.csv:*

A screenshot of a computer

Description automatically generated*84\_Loeschcke\_et\_al\_2000\_Wing\_traits\_&\_asymmetry\_lab pops.csv:*

A screenshot of a computer screen

Description automatically generated*85\_Loeschcke\_et\_al\_2000\_Wing\_asymmetry\_lab\_pops.csv:*

These tables are generated using Python’s pandas library by reading the csv into dataframes and printing the dataframe’s head. The code for this can be seen in the appendix.

For this report, we need to determine which dataset would be most beneficial to train classification models on.

The first dataset (*83\_Loeschcke\_et\_al\_2000\_Thorax\_&\_wing\_traits\_lab pops.csv*) provides a diverse range of features, including morphological measurements which could prove to be important for classifying species and populations. As well as this, it provides a combination of geographic, and temporal data.

The second dataset (*84\_Loeschcke\_et\_al\_2000\_Wing\_traits\_&\_asymmetry\_lab pops.csv*) focuses on wing measurements and asymmetry, which can also be critical for species differentiation. This also includes geographic and temporal data.

The third dataset (*85\_Loeschcke\_et\_al\_2000\_Wing\_asymmetry\_lab\_pops.csv*) focused on asymmetry in various measurements. This *may* reveal subtle differences between species/populations that other features may not capture.

For our classification task, its most likely best to use a dataset which provides a diverse range of features, which, in this case, would be the first dataset. This provides a variety of morphological measurements, which are likely to be highly relevant for distinguishing between various species/populations. Furthermore, the inclusion of temporal, geographical, and sex data may provide more accurate classification.

Overall, the first dataset offers a rich dataset for training robust machine learning models, and thus will be used for training the models.

## Data Preprocessing

Various steps must be taken with this data to convert it into a usable form to train models on.

Firstly, we must drop the data that will not be useful for the classification task. We can use the pandas library to count how many distinct/unique values are in each column. The code and output from this can be seen in the appendix.

There are several important things to note from this table. Firstly, there are 2 unique species, with 5 unique populations, which results in 10 unique classes that our models will classify into ( ). Secondly, the starting and ending year only provide one unique value, and thus offer no differentiating features between species/population. These columns will be dropped (to be clear, the Year\_start and Year\_end columns). Thirdly, we will also drop the *Replicate* column, as it only has 3 unique values, which arguably will not provide much extra differentiation between species/population, and thus is redundant.

To reiterate what will be done:

* The species and population columns will be combined.
* The columns *Year\_start, Year\_end*, and *Replicate* will all be dropped from the dataset.

The code to achieve this can be seen in the appendix.

The next step for preprocessing the data is to encode numerical values for the combined species and population so that they can be used for training an evaluation metrics. For example, encoding the species of *D.\_aldrichi* with a population *Binjour* to a value of 0, meaning that it now becomes the class of ‘0’. This will be repeated for all unique combinations of species and populations, which as mentioned is a total of 10 unique classes (0 through to 9). This will be done as well on the sex column, converting the ‘female’ value to 0, and ‘male’ to 1. This can be done by using the library scikit-learn’s Label Encoder. The code for this can be seen in the appendix.

The distribution of these classes can be seen in the following graph:A graph of a number of species

Description automatically generated

This shows that there is a near equal distribution of the classes, which means that the provided data is not biased towards one class or another. This ensures that the training data for all classes is near equal. The code to generate this graph can be seen in the appendix.

Next, there were issues identified with the data that the column ‘*wing\_loading*’ and ‘*Thorax\_length*’ were of type object instead of the desired float or integer data types. The origin of these errors is unknown but are suspected to be caused by missing values in the dataset. To rectify these errors, we dropped all rows which didn’t have a number as a value. This resulted in a total of 2 rows being dropped. The code to perform this can be seen in the appendix.

The next precaution taken was to drop all rows which contained a 0 in a measurement field. If a 0 is measured, that likely meant that the dragonfly was deformed or missing a limb/wing/etc (besides human error), which negatively affects the quality of the training data. To rectify this, all rows which have 0 as a measurement are dropped. The code to perform this can be seen in the appendix.

## Adding Gaussian Noise

The next step is to create a series of CSV files that contain various percentages of added noise; for this report the percentages of added noise will be 1%, 2.5%, 5%, 10%, 20%. The process for adding noise will be to calculate the standard deviation of a column’s values. We multiply the standard deviation by the percentage of noise for that iteration (1%, 10%, etc.).

We then generate a 1-dimensional vector the size of the column, filled with noise sampled from a random normal distribution with a mean of 0 and a standard deviation equal to the column’s standard deviation multiplied by the noise percentage.

This process will be repeated for all numerical columns (excluding categorical columns) and saved to separate CSV files with their respective noise percentage.

The code used to calculate this can be seen in the appendix.

### Example

For the thorax length column, if the standard deviation is 0.1 mm and the added noise is 10%, the standard deviation for sampling will be 0.01 (0.1 mm × 10%). A 1D vector of the column's length will be generated with values from the normal distribution centred around 0 with the calculated standard deviation.

# Model Investigation

## Method

To investigate the models the process that will be undertaken is as follows:

1. Train the model on unprocessed (0% noise) data to gauge a benchmark and evaluate using an 80/20 training/testing data split. During this step, hyperparameter tuning will be conducted to ensure the model is optimised for performance. These hyperparameters will not be changed for future testing of the same model (i.e. the hyperparameters used for 0% noise data will be used for 20% noise data).
2. Train the model on various amounts of added noise and test their accuracy using original unprocessed data. Repeat for all noise percentages.
3. Repeat steps 1 and 2 for all models.
4. Tabulate and compare results between models.

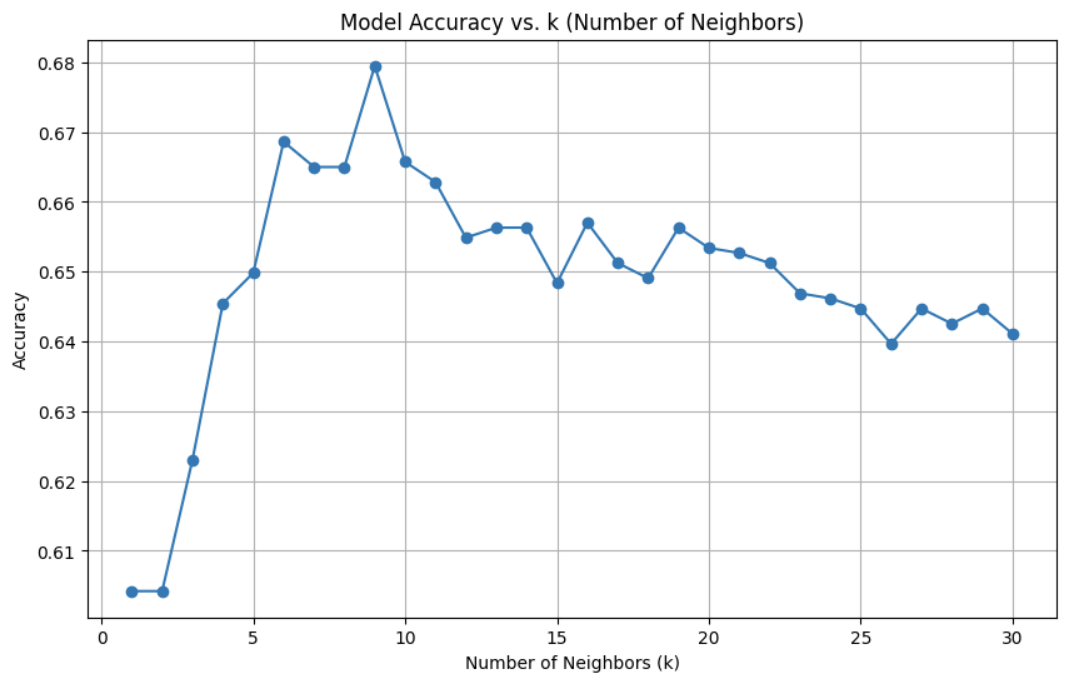
It should be emphasized that the models are being trained on noisy data but being assessed with the original data.

## K-Nearest Neighbours (K-NN)

The K-Nearest Neighbours (KNN) model requires tuning of several hyperparameters to optimize its performance:

* K Value: The number of neighbours considered in the prediction process. K values from 1 to 30 were tested.
* Weights: Determines the influence of neighbours. Two types were tested:
* Uniform: All neighbours have equal weight.
* Distance: Neighbours are weighted by the inverse of their distance, giving closer neighbours more influence.
* Metric: Defines how distances between points are calculated. Three metrics were used:
* Euclidean
* Manhattan
* Minkowski

A grid search was conducted across all K values from 1 to 30. The search determined that a K value of 9 resulted in the highest model accuracy of approximately 71% when using distance weighting with the Manhattan metric. The graph of accuracy versus K values (with distance and Manhattan parameters) is shown below:



The code used to grid search the hyperparameters and generate this graph can be seen in the appendix.

A diagram of a graph

Description automatically generated with medium confidenceThe confusion matrix for this trained model on the original data can be seen below:

Now it will be trained on various amounts of noise and tested against the original data.

The model accuracy when trained on various amounts of noise can be seen in the table below:

|  |  |
| --- | --- |
| Noise | Accuracy |
| 0% | 72.83% |
| 1% | 72.25% |
| 2.5% | 72.54% |
| 5.0% | 69.36% |
| 10% | 67.05% |
| 20% | 63.87% |

### Observations

The observation that the KNN model slightly decreases (or remains stable) as the noise level is expected. Although, it was expected that noise would more dramatically affect the results. However, there are several potential reasons why it didn’t:

1. Noise acts as regularization. Adding noise to the data can act as a form of regularization, preventing the model from overfitting to training data. This noise can help the model generalize better by smoothing out peculiarities in the data.
2. KNN robustness. As were using Manhattan distance and weighted distances, it can be expected that this makes the KNN particularly robust to variations in data. As the noise does not significantly alter the relative neighbourhood of points, the KNN’s predictions will not change much.
3. Scaling. As were scaling the values to a mean of 0 with a standard deviation of 1, this may be mitigating the effect of the noise.

## Neural Networks

### Background

We can employ neural networks to classify the Drosophila dataset. The primary structure of a network includes an input layer with as many neurons as input features, a series of hidden layers with varying amounts of neurons, and an output layer with varying amounts of neurons depending on the reason for use.

### Simple Neural Network

The simple network consists of two hidden layers. The first hidden layer has 512 neurons, and the second hidden layer has 256 neurons. Both layers use the ReLU activation function to introduce non-linearity into the model. The output consists of 10 neurons, corresponding to the number of classes in our dataset, with a SoftMax layer to handle multi-class classification.

#### Training Process

* Data preprocessing. As for all models, the input data has noise added and then is standardised to have zero mean and unit variance.
* Epochs. The network is trained for many epochs (200) to ensure the model has enough iterations to learn from the noisy data.
* Batch size. A batch size of 64 is used, which balances computational efficiency and model performance.
* Optimiser. Adam optimiser will be used, known for its effectiveness in training deep neural networks.
* Loss functions. Cross-entropy loss is used, which is standard for multiclass classification problems.

The models ‘effectiveness’ will be evaluated using accuracy as the primary metric.

#### Visualisations

Graphs of the training loss over the number of epochs can be seen as follows:

A group of graphs showing the growth of a stock market

Description automatically generated with medium confidence

A graph with a line

Description automatically generatedThe accuracy of the model across various noise percentages can be seen as follows:

### Complex Neural Network

The complex neural network is designed to offer increased robustness and performance under noisy conditions compared to the simpler network model. This network incorporates additional layers and techniques aimed at improving generalisation and reducing overfitting. While there is no guarantee that performance will be notably improved between model iterations (simple vs complex), it would be expected that for a larger dataset the testing accuracy would improve, as there is not a lot of data to train on where regularisations and batch normalisations would be highly effective.

The model architecture consists of 2 hidden layers with 512 and 256 neurons respectively, as well as batch normalisations followed by dropout layers between the and first and second hidden, as well as between the second hidden and output layer.

#### Training Process

The training process will be identical. Experimentations were conducted to find a potential better optimiser, but across all tests, Adam proved to have the highest accuracy consistently.

#### Visualisations

A group of graphs showing different types of data

Description automatically generated with medium confidenceThe training loss over epochs graph can be seen as follows for the various datasets.

A graph with a line and a point

Description automatically generatedThe accuracy over the different datasets can be seen as follows:

### Results

The results from the 2 neural networks can be seen tabulated as follows:

|  |  |  |
| --- | --- | --- |
| Noise | Simple Network Accuracy | Complex Network Accuracy |
| 0% | 81.21% | 82.37% |
| 1% | 82.95% | 83.53% |
| 2.5% | 83.24% | 82.95% |
| 5% | 84.97% | 81.21% |
| 10% | 81.21% | 78.90% |
| 20% | 75.43% | 73.99% |

### Improvements

Both validation sets and K-fold should be implemented to prevent overfitting and to monitor the networks performance during training, but due to time constraints these could not be implemented.

### Observations

Interestingly, the models trained with 1% and 2.5% noise data achieve higher accuracy than those trained on clean data. This phenomenon may be due to the regularization effect of noise, preventing overfitting and helping the models generalize better. However, as expected, the accuracy decreases with higher noise levels, highlighting the models' sensitivity to noise.

As expected, the complex network was slightly more accurate than the simple model on less noise datasets.

## Decision Tree

For this study, we will use a decision tree classifier to classify the Drosophila dataset. The primary hyperparameters we will tune using grid search for a decision tree include:

* Max depth: The maximum depth of the tree. Limiting this can prevent overfitting. We will grid search all depths from 0 to 50 (although lower depths will most likely have terrible performance)
* Criterion: The function to measure the quality of the tree branch. We will measure Gini impurity and information gain (entropy).
* Minimum samples split: The minimum number of samples required to split an internal node.
* Minimum samples leaf: The minimum number of samples required to be at a leaf node.
* Max features: The number of features to consider when looking for the best split.

The grid search conducted determined that a max depth of 20 nodes, with a minimum sample split of 6 and a minimum samples leaf of 2 with no max features provided the highest performance for the clean dataset.

### Results

|  |  |
| --- | --- |
| Noise | Accuracy |
| 0% | 75.43% |
| 1% | 71.09% |
| 2.5% | 70.23% |
| 5% | 73.12% |
| 10% | 65.89% |
| 20% | 64.45% |

### Observations

Interestingly, the model's performance at 5% noise shows a slight improvement (73.12%) compared to 2.5% noise (70.23%). This could be due to the model's sensitivity to specific noise patterns, suggesting that certain levels of noise might help in regularizing the model, preventing it from overfitting to the training data.

The general trend is a decline in accuracy as noise levels increase, which is expected as the decision tree model becomes more sensitive to noisy features and makes less reliable splits.

## Random Forests

We will now implement a random forest classifier to the dataset. The hyperparameters we tuned using grid search include:

* Number of trees: The number of trees in the forest. We tested values of 50, 100, 150, and 200.
* Maximum depth: The maximum depth of each tree. We tested values of 0, 10, 20, and 30.
* Minimum samples required to split an internal node: The minimum number of samples required to be a leaf node. We tested values of 2, 5, and 10.
* Minimum samples required to be a leaf node: The minimum number of samples required to be at a leaf node. We tested values of 1, 2, and 4.
* Number of features to consider for each split: The number of features to consider when looking for the best split. We tested values of the square root of features as well as the base 2 log.

#### Training Process

Using grid search, the best hyperparameters for the forest were determined. These were no max depth (somewhat expected), no max features, 1 minimum samples required to be a leaf node, 5 minimum samples required to split an internal node, with 150 trees.

These hyperparameters were then used to train the model on the clean and noisy datasets, and the performance was evaluated.

### Results

|  |  |
| --- | --- |
| Noise | Accuracy |
| 0% | 78.32% |
| 1% | 77.16% |
| 2.5% | 76.30% |
| 5% | 76.58% |
| 10% | 74.85% |
| 20% | 73.12% |

### Observations

The random forest model shows a strong baseline accuracy of 78.32% on the clean dataset, indicating good performance.

This data reports that the random forest is quite robust to noise, with only a 5.20% drop in accuracy between the clean data and 20% noise data. This suggests that the nature of random forests helps mitigate the impact of noise. As expected, a gradual decline in performance over increasing amounts of noise, but still retained a semi high level of accuracy.

The slight jumps in accuracy between the 2.5% and 5.0% noise dataset may be due to the regularisation effect of noise.

## Support Vector Machines

We will now implement a support vector machine (SVM) to classify the Drosophila dataset. These work by finding the hyperplane that best separates the data points of different classes in a high dimensional space.

We again employed hyperparameter tuning via grid search to optimise the hyperparameters of the model without any added noise. The parameters tuned:

* C (Regularisation): This controls the trade off between achieving a low error on the training data and minimising the model complexity. A smaller value of C smooths the decision boundary, while a large value aims to classify all training examples correctly.
* Gamma: Controls the influence of a single training example. A higher value means that only nearby points are considered, and a lower value means points further away from the boundary will also be considered.
* Kernel: The kernel function allows the SVM to better handle the relationships by transforming the data into a higher dimensional space.

The grid search concluded that for our case using a C value of 100, with a gamma value of 0.01, and a radial basis function resulted in the highest accuracy for the model.

### Results

|  |  |
| --- | --- |
| Noise | Accuracy |
| 0% | 81.50% |
| 1% | 81.21% |
| 2.5% | 82.08% |
| 5% | 80.64% |
| 10% | 81.21% |
| 20% | 77.46% |

A graph with a line

Description automatically generated

### Observations

The SVM model demonstrates a strong accuracy of 81.50% on the clean dataset - one of the highest measured so far. The model, as expected, showed a slight decrease in accuracy as noise increased. SVM’s are known to be sensitive to noise, so it’s rather surprising that it was managed to retain a large amount of accuracy when noise was added.

Interestingly, the model's accuracy at 2.5% noise was higher than at 0% noise, achieving 82.08%. This improvement could be attributed to the regularization effect of noise, which helps prevent overfitting and allows the model to generalize better.

# Performance Comparison

The raw results of each model can be seen tabulated as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Models Accuracy | | | | | |
| Noise | K-NN | Simple N.N. | Complex N.N. | Decision Tree | Random Forest | SVM |
| 0% | 72.83% | 81.21% | 82.37% | 75.43% | 78.32% | 81.50% |
| 1% | 72.25% | 82.95% | 83.53% | 71.09% | 77.16% | 81.21% |
| 2.5% | 72.54% | 83.24% | 82.95% | 70.23% | 76.30% | 82.08% |
| 5% | 69.36% | 84.97% | 81.21% | 73.12% | 76.58% | 80.64% |
| 10% | 67.05% | 81.21% | 78.90% | 65.89% | 74.85% | 81.21% |
| 20% | 63.87% | 75.43% | 73.99% | 64.45% | 73.12% | 77.46% |

## Visualisations

A line graph of the accuracy of the noise levels can be seen as follows. As expected, there is evidently a trend of accuracy decreasing as noise level increases.

A graph of different noise levels

Description automatically generated

A box plot can be seen below which helps visualise where each models lays relative to the other models.

A diagram of a box diagram

Description automatically generated

A graph of accuracy retention of each model can be seen as follows. This depicts how much the models retain their accuracy as the noisiness of the data increases. The models start at 100% accuracy, as it is assumed that they train best on the original data (which was not always the case).

A graph of different noise levels

Description automatically generated

## Analysis

SVM model shows strong accuracy on clean detail and retains this high accuracy even as noise levels increase, with the highest retention rate among the models tested. Interestingly, the SVM’s accuracy at 2.5% is better than the model trained on the original data, suggesting that noise in this case may help in regularisation.

Both the simple and complex neural networks demonstrated high accuracy on the clean data, while the complex slightly outperforming the simple on the first couple iterations of data. They prove to be quite good at retaining their performance as the noise increases. Interestingly, the simple network has an unexpected increase in accuracy for the first few iterations of noise addition; again, this may also be due to be the regularisation effect of noise.

The random forest model shows quite robust performance with a strong baseline accuracy of 78.32%, with the expected gradual decline in accuracy as noise increased. The total drop between 0% noise and 20% noise was only 5.20% in accuracy, indicating that the model is particularly robust (and can be seen performing second in the accuracy retention graph).

The simpler models (K-NN and decision tree) have lower performance and retention rates compared to the more complex models (as expected). K-NN and the decision tree’s accuracy decreases steadily with added noise (besides a slight jump at 5% noise for the decision tree).

Overall, the SVM and neural networks demonstrated high robustness and performance retention across the various noise levels, with SVM having the best retention rate. The simpler models showed weaker performance against noise - as expected. The random forest lies in the middle ground between complexity and robustness, showing good accuracy retention but moderate performance overall.

# Future Improvements

There are several improvements that can be made for future studies which would make these results more rigorous and well tested.

Firstly, K-fold cross validation would provide better results when testing the robustness of the model, ensuring that the results are not dependent on a single train-test split. For the purpose of this study, the same random seed was used for all train-test splitting, and all random number generators needed. This ensured fair testing across models. K-fold validation was to be implemented, but due to time constraints with the report, it unfortunately could not be.

Secondly, while a grid search was performed for each model, expanding the range and combination of hyperparameters could lead to discovering more optimal settings. This includes grid searching the hyperparameters for each level of added noise, which would likely result in better accuracy for each iteration, and thus better accuracy retention.

Furthermore, more extensive grid search could have been performed on the neural network’s architecture, such as the number of hidden layers, neurons in the hidden layers, activation functions, and optimisers. This again, would likely increase their performance by a few percentage points.

Further experimentation could be done with varying regularisation techniques, such as dropout rates or L1/L2 regularisation, may help in prevent overfitting and improve the generalisation.

A validation set should have been utilised with the neural networks to ensure the networks were not overfitting with the data. Furthermore, more experimentation could be done with the number of epochs the networks were trained for. All of these would likely result in an increase of a few percentage points of accuracy, and thus the retention rate.

# Conclusion

In conclusion, the SVM proved to be the most robust against noise, as well as the 2nd highest performance model in model accuracy. The neural networks are similar in performance and robustness but just fall short; and the simpler models are not complex enough to effectively classify the Drosophila dataset. It is theorised that future improvements to the neural networks training and architecture would likely outperform the SVM, but rigorous testing must be conducted to identify where this increase in performance lies. Further improvements include implementing K-fold validation, more extensive hyperparameter tuning, and as mentioned, experimentation with neural network architecture.

Appendix

A screen shot of a computer code

Description automatically generatedCode to read a CSV file and print the first 5 rows of it. The code snippet also includes commented out lines which provide further information about that nature of the data.

A screenshot of a computer

Description automatically generatedCode and output from printing the number of unique values in each column of the original dataset.

Code to combine species and population as well as drop columns not being used is as follows:

A computer screen with text on it

Description automatically generated

A screen shot of a computer code

Description automatically generatedCode to encode every variation of species/population and gender into a unique class:

Code to drop NaN values within the dataset.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generatedCode to remove any rows which contained 0 as a measurement.

The function to add noise to the dataset can be seen below.

A screen shot of a computer code

Description automatically generated

The code to generate the class distribution graph.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generatedThe code used to perform a grid search of the hyperparameters of the KNN model.

A screen shot of a computer program

Description automatically generatedThe code used to generate the graph of the accuracy vs K value of the graph.

Example code used to grid search hyperparameters (in this case for the SVM).

A screen shot of a computer program

Description automatically generated

The entire project codebase can be found at the following link.