https://github.com/kikutabryan/AER850_Project.git

AER850 PROJECT 1

Bryan Kikuta

501039179

Abstract

This project focused on the implementation of various machine learning models for prediction of maintenance and manufacturing steps for Augmented Reality utilization in the aerospace industry. Multiple different models were selected and their respective hyperparameters were tuned. Various metrics were used to determine the best performing models, followed by the selection of the best models to be used in a stacked classification model. Finally, the final model was put to the test in terms of predicting various coordinate points.

Project GitHub Link: https://github.com/kikutabryan/AER850_Project.git

Table of Contents

Table of Figures

List of Tables

1 Introduction

This project focused on implementing machine learning models to predict maintenance and manufacturing steps in the aerospace industry, specifically for Augmented Reality applications. The primary objective was to develop a robust model capable of accurately classifying specific X, Y, and Z coordinates to their respective steps. The first component of this project focused on gaining an understanding of the data, this was achieved through the visual representation of the data by utilizing histograms, in addition to a 3D scatter plot. The next component of this project was to identify any correlations within the dataset that should be removed. This was followed by the development of machine learning models after the dataset was appropriately stratified by the Steps into training and testing datasets. Models were created for the following classifiers: Support Vector Machine, Logistic Regression Classifier, Random Forest Classifier, Decision Tree Classifier, K Nearest Neighbours Classifier. The Decision Tree Classifier in addition to the K Nearest Neighbours Classifiers were both stacked together to create a new prediction model after the model performance of each model had been analyzed. The final step consisted of predicting a set of provided data points with the stacked model, and ensuring they were accurate.

2 Results

2.1 Step 2

Perform statistical analysis on the dataset and visualize the dataset behaviour within each class. This will provide an initial understanding of the raw data behaviour. You are required to include the plots and explain the findings.

To gain a better understanding of the dataset, histograms were created for each of the columns of the provided dataset. The feature columns are as follows: X, Y, and Z. The output column is the Step. The histograms for each of these are seen in the following figures. Each of the histograms shows the frequency of the specific variable of study with respect to a range of values. For example, the histogram of column X indicates that X occurred at higher frequencies for values less than 4, and greater than 6, with the interval between 4 and 6 not having any occurrences. Based on the histograms of X, Y, and Z, it is challenging to gain a proper understanding of what the data represents, aside from knowing the spread of each specific variable.

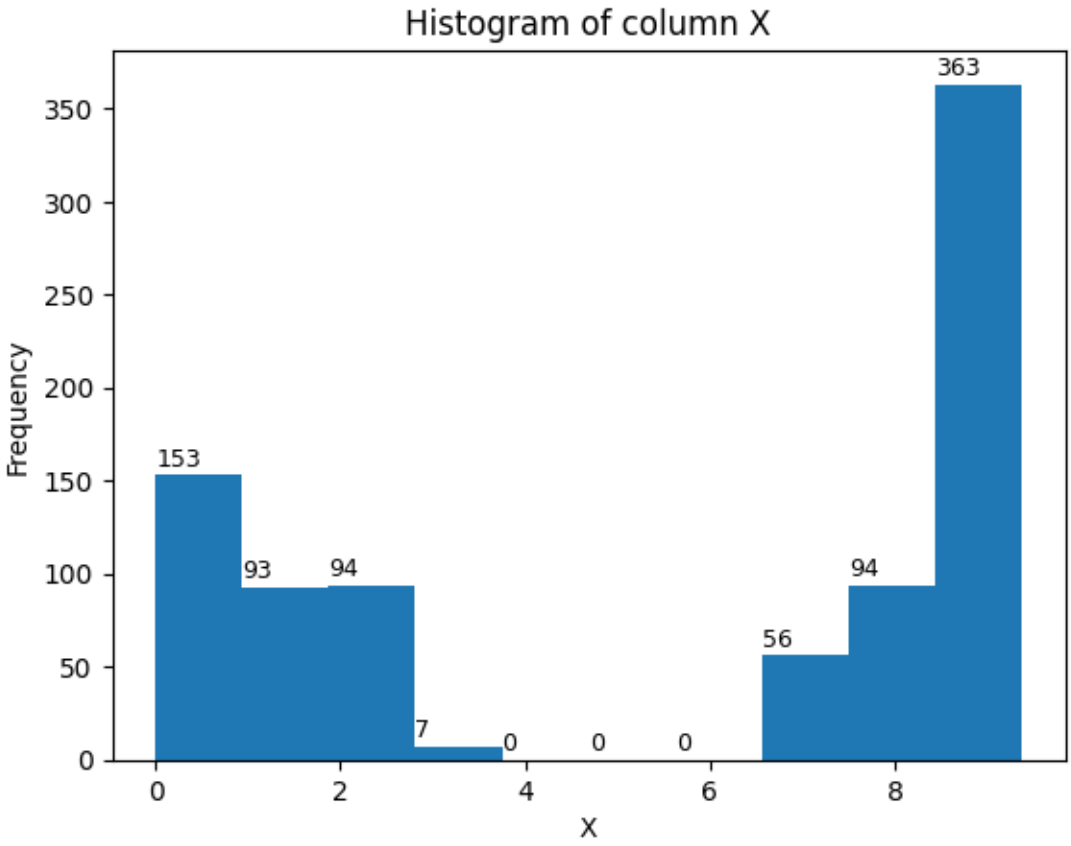

Figure 1: Histogram of X coordinate.

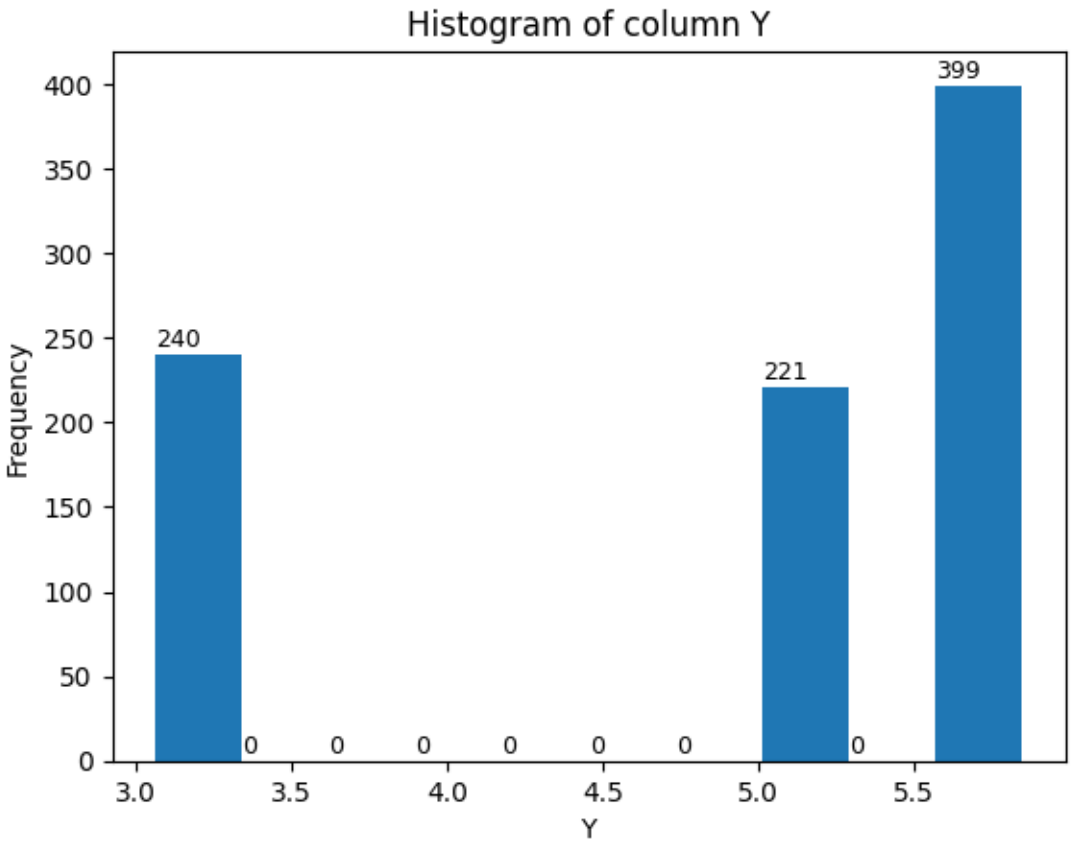

Figure 2: Histogram of Y coordinate.

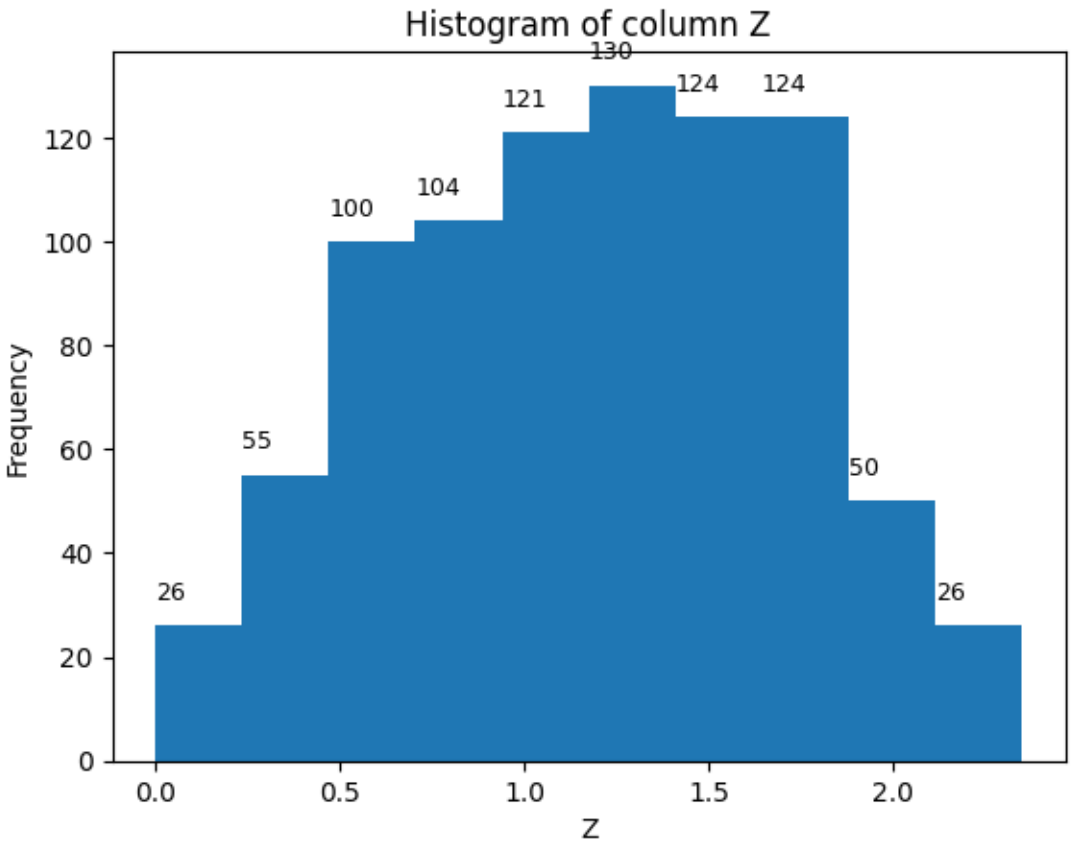

Figure 3: Histogram of Z coordinate.

The final histogram focuses on the number of occurrences of each Step in the dataset. This histogram is slightly more useful than the prior ones, as it indicates where the majority of the datapoints lie. From the histogram, it can be seen that the majority of the datapoints are of steps numbered 7 and 8. Given that the majority of steps lie within steps 7 and 8, it is probable that the predictions for steps 7 and 8 will be significantly better than the predictions of the others steps such as steps 2 through 6, and 9 through 12 as the number of occurrences of these are quite low. When training a machine learning model, the more data you have, the better the results will typically be. It is also important to properly stratify the data when splitting it into training and testing sets by the steps and ensure that each of the sets maintains the same proportions as seen in the histogram below.

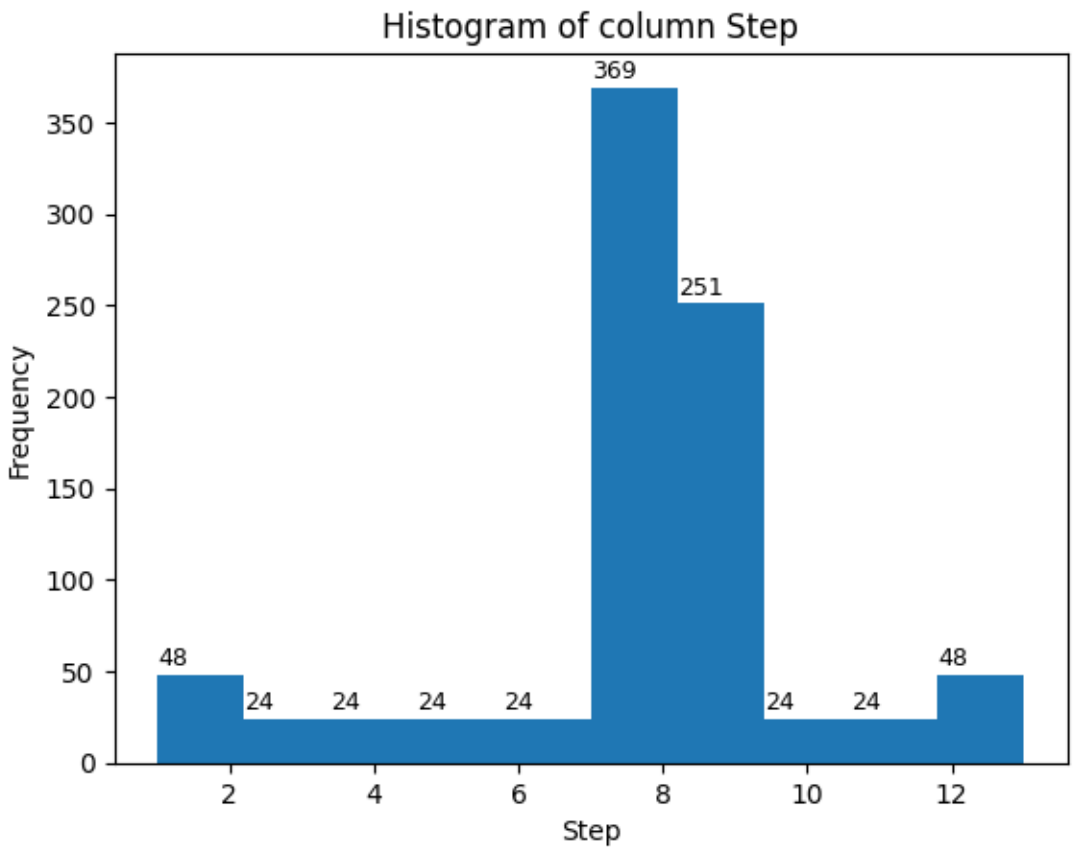

Figure 4: Histogram of Step numbers.

While the histograms are good at depicting the general spread of the data, and where it lies, it is beneficial to find other means of representing that data, particularly ways that show how it relates to one another. As the data provided consisted of X, Y, and Z coordinates with their associated steps, it was logical to generate a 3D plot of all the data alongside their correlated steps. The figure below depicts a 3D scatter plot of all the data, with colour coding for each of the specific steps. Based on the scatter plot, the steps appear in linearly aligned groups, and the points for a specific step lie beside one another. Step 9 appears to have the longest range of data points, and steps one through 6 all like beside one another on the same vector. Overall, the 3D plot provides a far better understanding of the general structure and position of each of the steps.

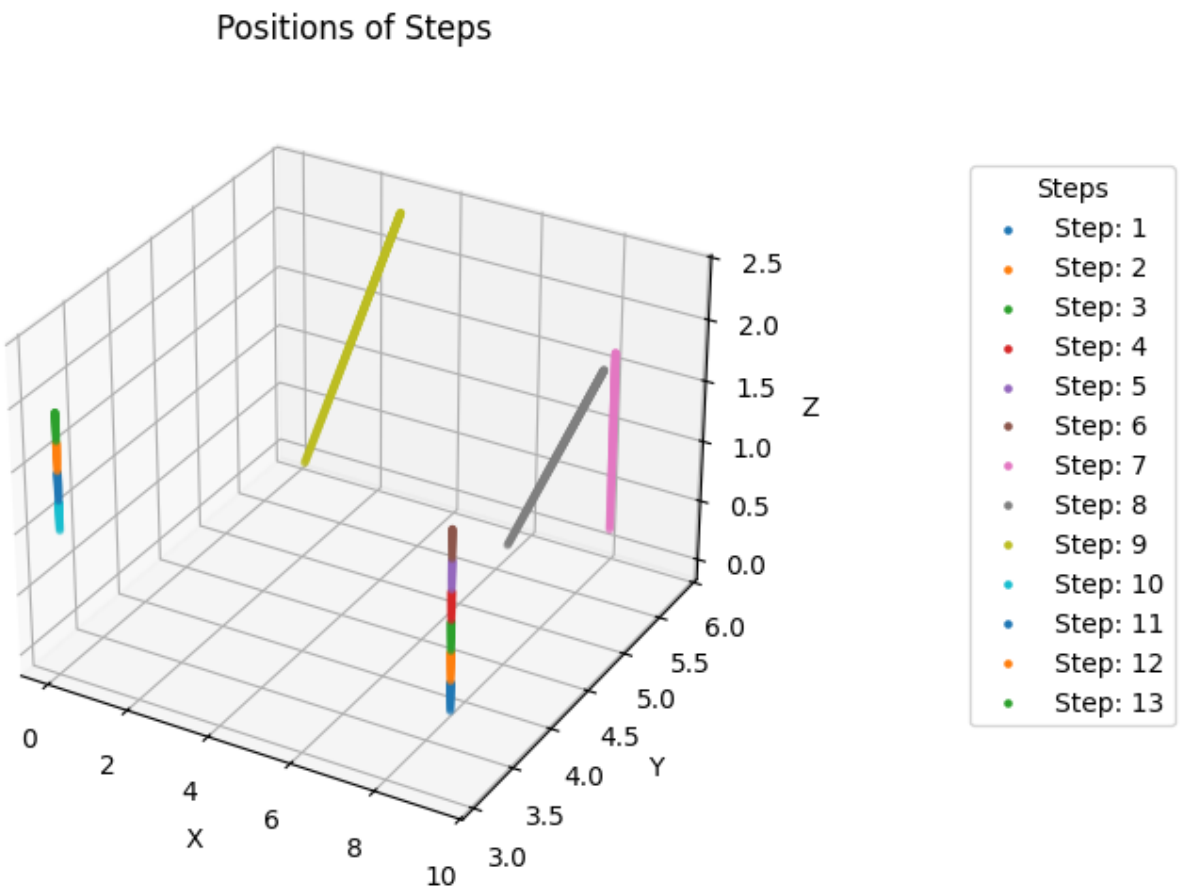

Figure 5: 3D scatter plot of data points.

2.2 Step 3

Assess the correlation of the features with the target variable. A correlation study provides an understanding of how the features impact the target variable. A common correlation method used is *Pearson Correlation*. You are required to include the correlation plot, and explain the correlation between the features and the target variables, along with the impact it could have on your predictions.

An important step of data processing consists of the elimination of strong correlations between feature variables. Oftentimes in datasets, certain features will be correlated with one another, to

assess if this is the case, correlation matrices can be utilized. The figure below depicts a correlation matrix of the input variables in addition to the output step variable. The reason for leaving the output variable named “Step” within the correlation matrix was that in the even there are two variables which are strongly correlated, the one which has a lower correlation with the output variable would be dropped from the dataset. As can be seen by the colours of the grids, the non-diagonal grid between the input variables has relatively dark colours, indicating that there is little correlation between any of the features. On a different note, the bottom left and top right squares are brighter, indicating a correlation value in the 0.7 range. This means that there is a relatively strong correlation between the input variable X and the related step of a specific coordinate.

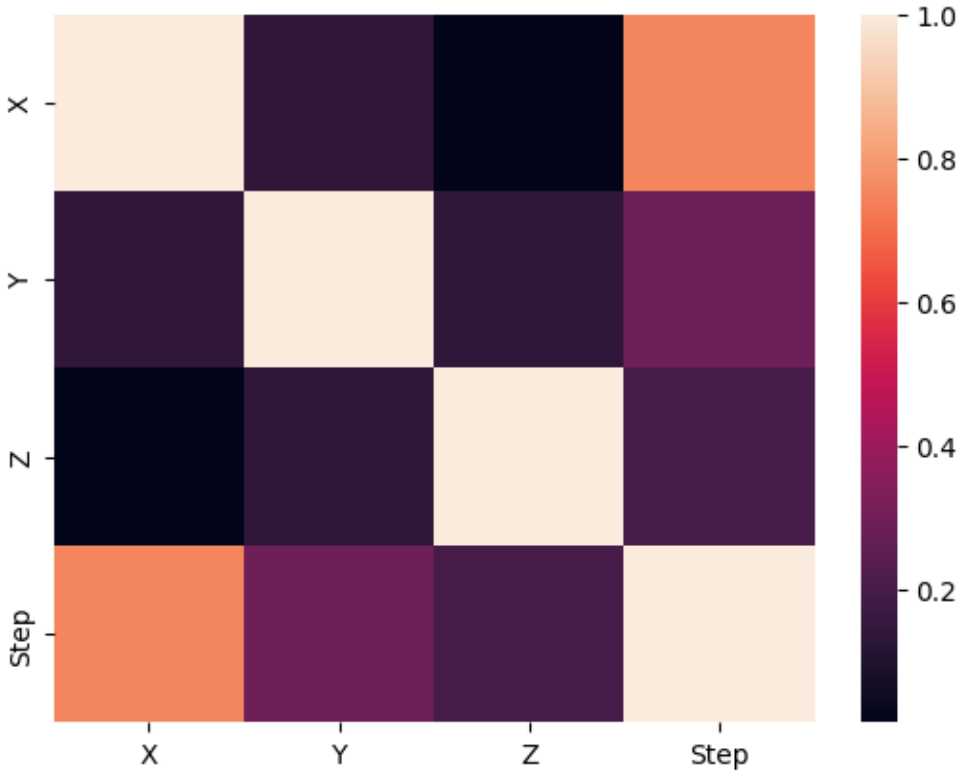

Figure 6: Correlation matrix of all data.

The figure below depicts the correlation matrix without the output variable included. This gives a better representation of the lack of any existing correlations given that all the non-diagonal squares are all dark coloured indicating there are little to no correlations between the input variables.

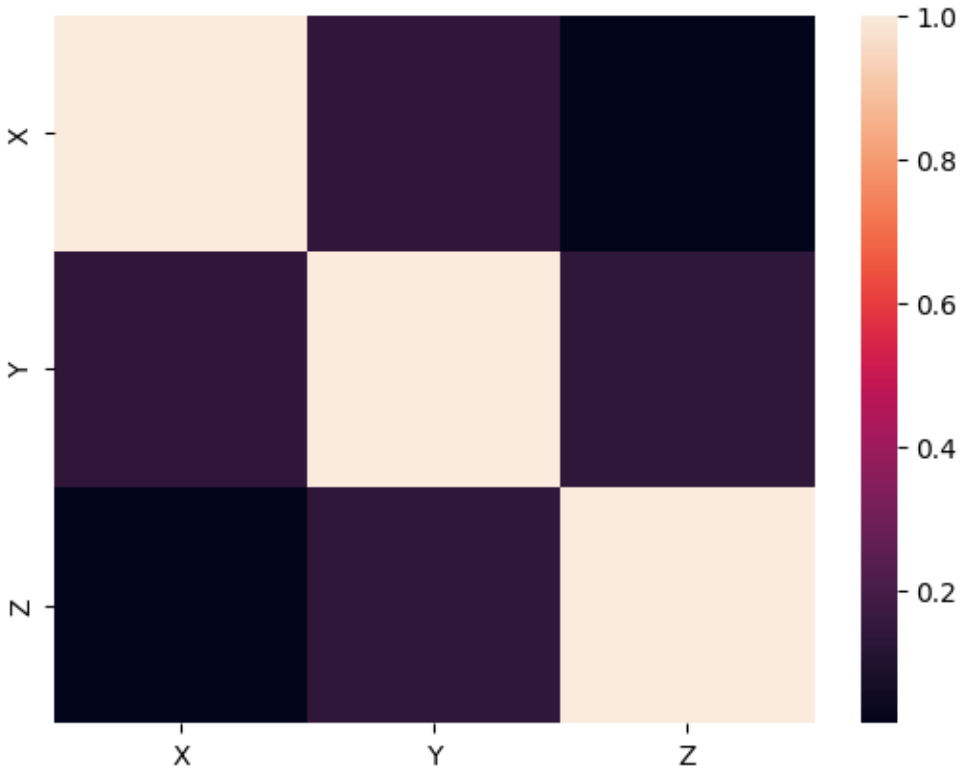

Figure 7: Correlation matrix of coordinates.

2.3 Step 4

Prepare the data to create three classification models (based on ML algorithms). The dataset needs to be split into training and testing categories to develop the models. For each ML model, utilize grid search cross-validation to assess the hyperparameters that give you the best results. You are required to explain your selected choice of classification algorithms. In addition to the three classification models with grid search cross-validation, you must make one model based on using *RandomizedSearchCV*. This will provide another method of determining the best hyperparameters to optimize your results.

2.3.1 Support Vector Classifier

The first model which was utilized was the Support Vector Machine (SVM) Classifier. SVMs function by identifying a hyperplane which best sorts the data points into different classes. Typically, SVMs are only sufficient for binary classification, that is a classification problem where there are only two classes, either positive or negative. Given that this problem had a total of 13 different classes, each being a different step, the traditional SVM could not be used. Scikit-learn handles this challenge by implementing a one versus one approach to allow for SVMs to handle multiclass classification.

Various parameters were set within the parameter grid of the SVM for selecting the best hyperparameters with GridSearchCV. The parameters which were selected were influenced by the course lecture slides in addition to documentation from the Scikit-learn website. The code utilized to run the select the best hyperparameters for the model can be seen below:

```
# Support Vector Classifier (SVC)
svc = SVC(random_state=42)
param_grid_svc = {
    "C": [0.1, 1, 10, 100],
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "degree": [2, 3, 4],
    "gamma": ["scale", "auto"],
}

grid_search_svc = GridSearchCV(svc, param_grid_svc, cv=5, scoring="f1_micro", n_jobs=-
1)
grid_search_svc.fit(X_train, y_train)
svc_model = grid_search_svc.best_estimator_
print(f"Best SVC Model: {svc_model}")
```

The hyperparameters of the best SVC model were as follows:

```
Best SVC Model: SVC(C=1, degree=2, gamma='auto', kernel='poly', random_state=42)
```

2.3.2 Logistic Regression Classifier

The next model which was selected was the Logistic Regression (LR) Classifier. The logistic regression classifier functions by making a linear combination and passing it through a sigmoid function to set the output to a range from 0 to 1. The LR model predicts a probability which if above a certain threshold, indicates a positive value, and if below, indicates a negative value for a specific class. The code alongside the hyperparameters which were to be tuned using GridSearchCV can be seen below:

```
# Logistic Regression Classifier
lrc = LogisticRegression(random_state=42)
param_grid_lrc = {
    "C": [0.1, 1, 10, 100],
    "max_iter": [200]
}
grid_search_lrc = GridSearchCV(lrc, param_grid_lrc, cv=5, scoring="neg_log_loss",
n_jobs=-1)
grid_search_lrc.fit(X_train, y_train)
lrc_model = grid_search_lrc.best_estimator_
print(f"Best LRC Model: {lrc_model}")
```

The best performing LR model had the following hyperparameters:

```
Best LRC Model: LogisticRegression(C=100, max_iter=200, random_state=42)
```

2.3.3 Random Forest Classifier

The next model which was selected was the Random Forest (RF) Classifier. The RF model consist of multiple decision trees. To make the predictions, each tree in the forest provides a prediction and the random forest takes the majority prediction among all trees as the final prediction. The code alongside the hyperparameters which were to be tuned using GridSearchCV can be seen below:

```
# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)
```

```
param_grid_rfc = {
    "n_estimators": [10, 20, 50],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2"]
}
grid_search_rfc = GridSearchCV(rfc, param_grid_rfc, cv=5, scoring="f1_micro", n_jobs=-
1)
grid_search_rfc.fit(X_train, y_train)
rfc_model = grid_search_rfc.best_estimator_
print(f"Best RFC Model: {rfc_model}")
```

The best performing RF model had the following hyperparameters:

```
Best RFC Model: RandomForestClassifier(n_estimators=10, random_state=42)
```

2.3.4 Decision Tree Classifier

The final model which was tuned with GridSearchCV was a Decision Tree (DT) Classifier. Decision Tree Classifiers operate by having numerous decisions in a tree like structure, where decisions lead to other decision nodes which eventually lead to a lead node providing the final prediction. The code alongside the hyperparameters which were to be tuned using GridSearchCV can be seen below:

```
# Decision Tree Classifier
dtc = DecisionTreeClassifier(random_state=42)
param_grid_dtc = {
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}
grid_search_dtc = GridSearchCV(dtc, param_grid_dtc, cv=5, scoring="f1_micro", n_jobs=-
1)
grid_search_dtc.fit(X_train, y_train)
dtc_model = grid_search_dtc.best_estimator_
print(f"Best DTC Model: {dtc_model}")
```

The best performing DT model had the following hyperparameters:

```
Best DTC Model: DecisionTreeClassifier(random_state=42)
```

2.3.5 K Neighbors Classifier (RandomizedSearchCV)

The final model selected was the K Nearest Neighbors (KNN) Classifier. This model was to be tuned using RandomizedSearchCV. The KNN Classifier operates by finding neighbouring data points, and deciding the class based on most of the neighbouring points. The KNN model utilizes the training data points as the original neighbouring points which it uses to gather the class of the prediction point. The optimization of this model is different from the previous ones as this uses RandomizedSearchCV. One of the distinctions between the two is that this hyperparameter selection method randomly selects different combinations up to the number of iterations and selects the best model from those combinations. This allows for a greater range of values to be specified, and faster model selection times in comparison to the GridSearchCV method. The code alongside the hyperparameters which were tuned using RandomizedSearchCV can be seen below:

```python
# K Neighbors Classifier - Randomized Search CV
knc_rand = KNeighborsClassifier()
param_grid_knc_rand = {
    "n_neighbors": np.arange(1, 100),
    "weights": ["uniform", "distance"],
    "algorithm": ["ball_tree", "kd_tree", "brute"],
    "leaf_size": np.arange(1, 100),
}
random_search_knc_rand = RandomizedSearchCV(
    knc_rand,
    param_grid_knc_rand,
    n_iter=100,
    cv=5,
    scoring="f1_micro",
    n_jobs=-1,
    random_state=42,
)
random_search_knc_rand.fit(X_train, y_train)
knc_rand_model = random_search_knc_rand.best_estimator_
print(f"Best KNC Model: {knc_rand_model}")
```

In the code, it can be seen that some of the parameters use the NumPy “arange” method which lists out a range of numbers over a specified interval with a default spacing of 1. The randomized search will randomly select a number from this range to use in combination with other randomly selected hyperparameters.

The best performing KNN model had the following hyperparameters:

```
Best KNC Model: KNeighborsClassifier(algorithm='ball_tree', leaf_size=np.int64(80),
                    n_neighbors=np.int64(77), weights='distance')
```

2.4 Step 5

Compare the overall performance of each model based on f1 score, precision and accuracy. You are required to provide an explanation to what these metrics mean and which metric to prioritize for this use-case. Based on the selected model, create a confusion matrix to visualize the performance of your model. Include the confusion matrix in the report as well.

In order to evaluate the performance of the different machine learning models, the metrics of precision, recall, accuracy and F1 score were utilized. Each of the individual sections below dedicated for a specific machine learning model contains a classification report which outlines each of these metrics. It is important to have an understanding of what each of these metrics details, thus, the definitions shall be outlined. Accuracy is a metric which focuses on the total number of correct predictions divided by the total number of data points. The equation for accuracy is as follows:

$$Accuracy = \frac{\#\ of\ correct\ predictions}{\#\ of\ all\ data\ points}$$

The next metric is precision, which studies how well a model was able to classify positives. That is, out of a group of values which were assigned positive, how many of them are true positives. The equation for precision is as follows:

$$Precision = \frac{\#\ of\ true\ positives}{\#\ of\ true\ positives + \#\ of\ false\ positives}$$

The next individual metric is recall. Recall looks at a specific value or in this case a Step, and how well the model was able to correctly classify what it is. Recall will be high if all the instances of a specific Step were assigned correctly but will be low if they were not assigned correctly, resulting in numerous false negatives. The equation for recall is as follows:

$$Recall = \frac{\#\ of\ true\ positives}{\#\ of\ true\ positives + \#\ of\ false\ negatives}$$

The final metric is the F1 score. The F1 score considers both the precision and the recall of the model to provide a more balanced understanding of the model. Recall fails to penalize false positives, and precision fails to penalize false negatives, thus, f1 score is used to take into account both recall and precision. The equation for f1 score is as follows:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

In the case of this project, the F1 score was prioritized for the determination of the best models due to it considering both recall and precision to find a well-balanced model. Simply considering only precision or only recall would suffer a lot of short comings of each equation.

Confusion matrices are displayed for each of the models. The vertical axis indicates which label a prediction was, and the horizontal axis indicates what the prediction was. For example, in the

confusion matrix for the Support Vector Classifier, there is a 1 at position (2, 1). This indicates that the correct label value was Step 1, but the model predicted it as Step 2.

2.4.1 Support Vector Classifier

The weighted average F1 score for the Support Vector Machine Classifier (SVM) is 99%. Based on the confusion matrix, the SVM model only got 2 predictions wrong indicated by the summation of the non-diagonal values.

Table 1: Support Vector Classifier classification report.

	precision	recall	f1-score	support
1	1.00	0.80	0.89	5
2	0.83	1.00	0.91	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	0.83	1.00	0.91	5
6	1.00	0.75	0.86	4
7	1.00	1.00	1.00	29
8	1.00	1.00	1.00	44
9	1.00	1.00	1.00	50
10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	5
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	5
accuracy			0.99	172
macro avg	0.97	0.97	0.97	172
weighted avg	0.99	0.99	0.99	172

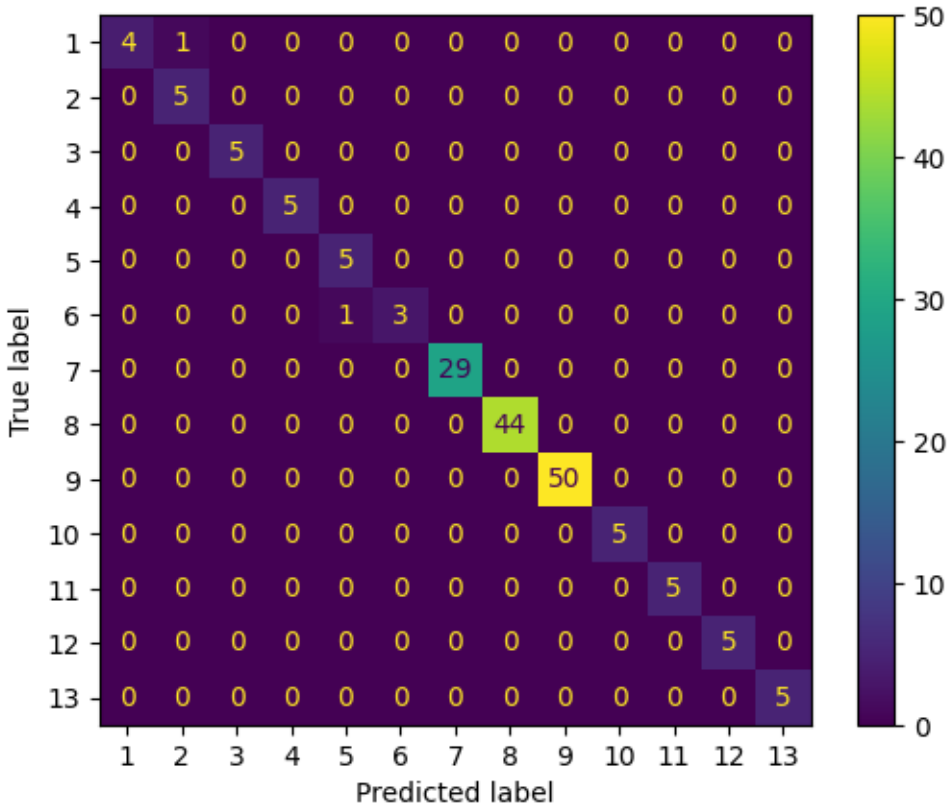

Figure 8: Support Vector Classifier confusion matrix.

2.4.2 Logistic Regression Classifier

The weighted average F1 score for the Logistic Regression Classifier (LR) is 99%. Based on the confusion matrix, the LR model only got 1 prediction wrong indicated by the summation of the non-diagonal values.

Table 2: Logistic Regression Classifier classification report.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	5
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	0.83	1.00	0.91	5
6	1.00	0.75	0.86	4
7	1.00	1.00	1.00	29
8	1.00	1.00	1.00	44
9	1.00	1.00	1.00	50

10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	5
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	5
accuracy			0.99	172
macro avg	0.99	0.98	0.98	172
weighted avg	1.00	0.99	0.99	172

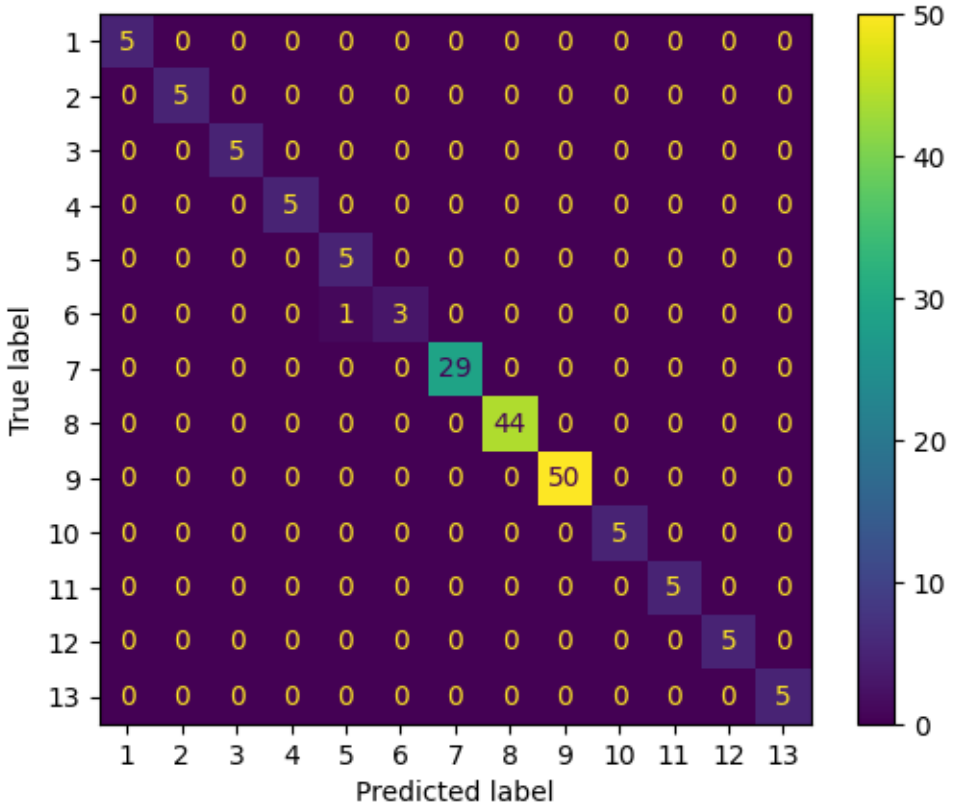

Figure 9: Logistic Regression Classifier confusion matrix.

2.4.3 Random Forest Classifier

The weighted average F1 score for the Random Forest Classifier (RF) is 99%. Based on the confusion matrix, the RF model only got 2 predictions wrong indicated by the summation of the non-diagonal values.

Table 3: Random Forest Classifier classification report.

	precision	recall	f1-score	support
1	1.00	0.80	0.89	5
2	0.83	1.00	0.91	5

3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	0.83	1.00	0.91	5
6	1.00	0.75	0.86	4
7	1.00	1.00	1.00	29
8	1.00	1.00	1.00	44
9	1.00	1.00	1.00	50
10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	5
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	5
accuracy			0.99	172
macro avg	0.97	0.97	0.97	172
weighted avg	0.99	0.99	0.99	172

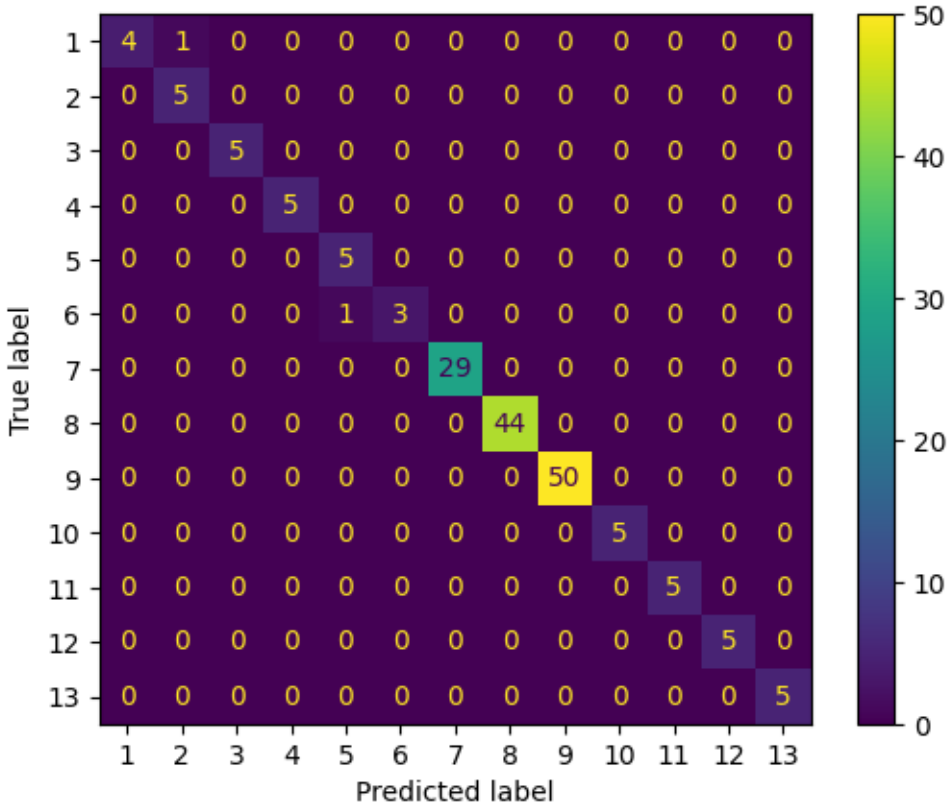

Figure 10: Random Forest Classifier confusion matrix.

2.4.4 Decision Tree Classifier

The weighted average F1 score for the Decision Tree Classifier (DT) is 99%. Based on the confusion matrix, the DT model only got 1 prediction wrong indicated by the summation of the non-diagonal values.

Table 4: Decision Tree Classifier classification report.

	precision	recall	f1-score	support
1	1.00	0.80	0.89	5
2	0.83	1.00	0.91	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	1.00	1.00	1.00	5
6	1.00	1.00	1.00	4
7	1.00	1.00	1.00	29
8	1.00	1.00	1.00	44
9	1.00	1.00	1.00	50
10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	5
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	5
accuracy			0.99	172
macro avg	0.99	0.98	0.98	172
weighted avg	1.00	0.99	0.99	172

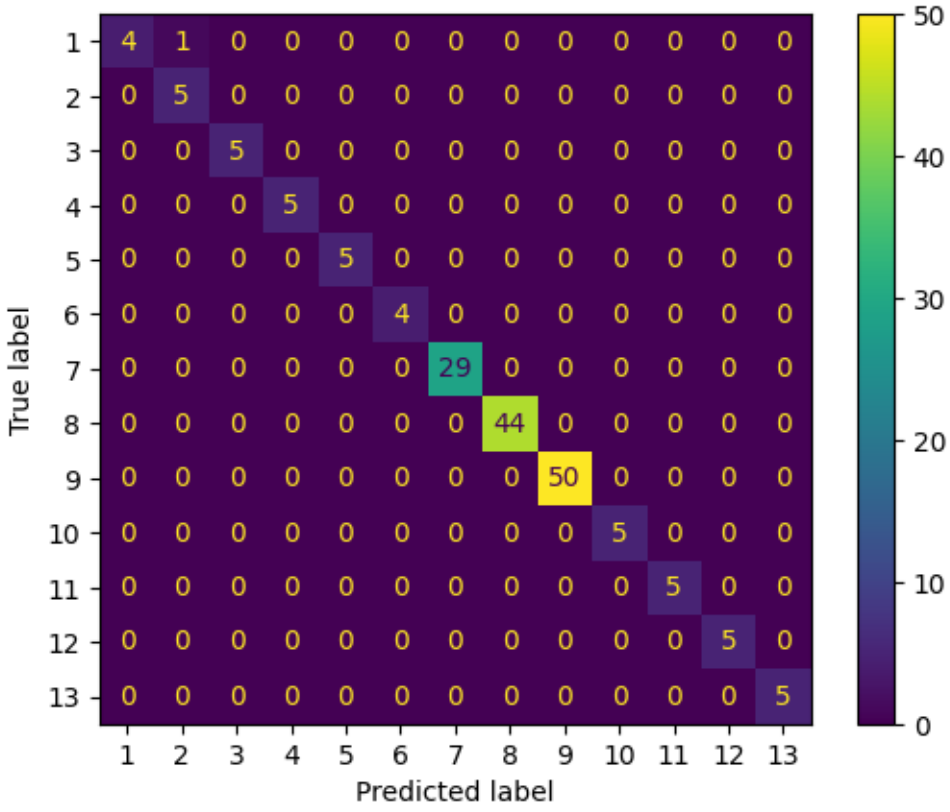

Figure 11: Decision Tree Classifier confusion matrix.

2.4.5 K Neighbors Classifier (RandomizedSearchCV)

The weighted average F1 score for the K Neighbors Classifier (KN) is 99%. Based on the confusion matrix, the KN model only got 1 prediction wrong indicated by the summation of the non-diagonal values.

Table 5: K Neighbors Classifier (RandomizedSearchCV) classification report.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	5
2	1.00	1.00	1.00	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	0.83	1.00	0.91	5
6	1.00	0.75	0.86	4
7	1.00	1.00	1.00	29
8	1.00	1.00	1.00	44
9	1.00	1.00	1.00	50
10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	5
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	5
accuracy			0.99	172
macro avg	0.99	0.98	0.98	172
weighted avg	1.00	0.99	0.99	172

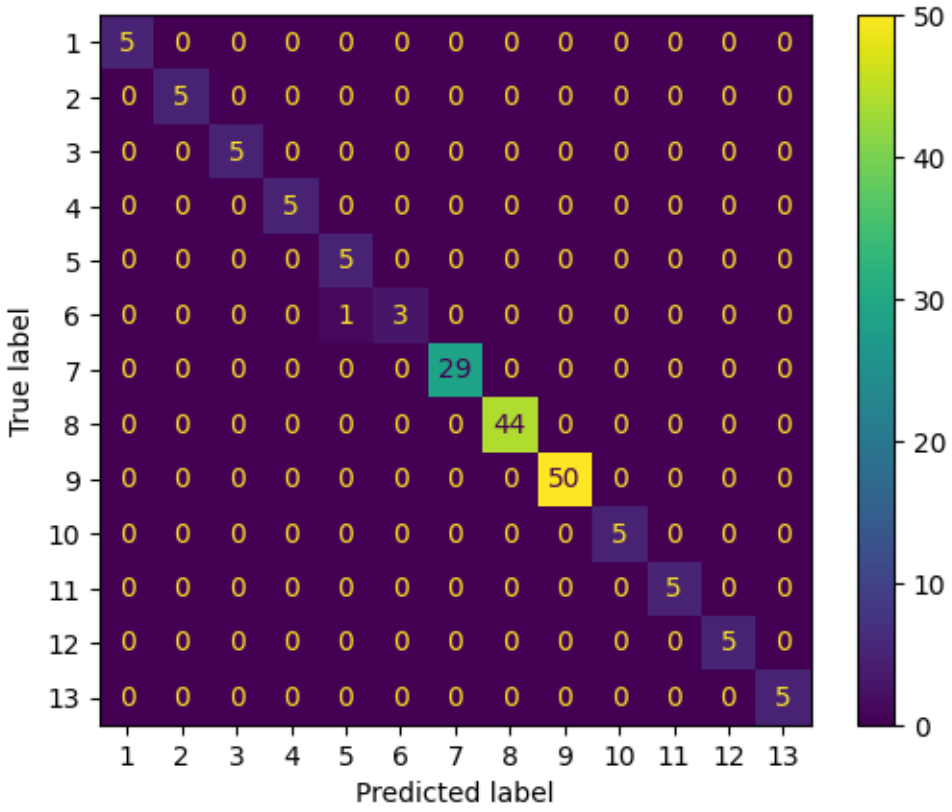

Figure 12: K Neighbors Classifier (RandomizedSearchCV) confusion matrix.

2.4.6 Remarks

Based upon the confusion matrices of each of the different models, the best performing models were the Logistic Regression Classifier, Decision Tree Classifier, and the K Neighbors Classifier, all of which only got a single prediction wrong on the testing data set.

2.5 Step 6

Using scikit-learn's *StackingClassifier*, combine two of the previously trained models to analyze the impact of model stacking on overall performance. Evaluate the performance of the stacked model based on their f1 score, precision and accuracy along with a confusion matrix to provide a clear visual representation. Include this confusion matrix in the report for detailed performance analysis. If a significant increase in accuracy is observed, discuss

how combining complementary strengths of the models contributed to the improvement. Conversely, if the change is minimal, explain why you think the stacking models had limited effectiveness.

For the stacked model, the K Neighbor Classifier and the Decision Tree Classifiers were stacked together as they were in the set of best performing models analyzed in the previous section. The weighted average F1 score of the stacked model was also 99% like the other models had been. There was no clear increase in the performance of the model. The reasoning for this can be attributed to the already excellent performance of the previous models having F1 scores of also 99%, leaving little room for improvement. Upon observation, the Steps which suffered from incorrect predictions were either steps 1 or 6, both which fall under the large cluster of different steps linearly aligned. There is the possibility that the points which were being incorrectly predicted were points on the edge neighbouring steps 2 or 6. This would result in ambiguity of the predictions, especially if the training data set did not have the edge points (which might be the ones in the testing dataset) resulting in incorrect predictions. Further analysis of this could be conducted. The confusion matrix for the stacked model indicated that step 1 was incorrectly predicted to be step 2, this was the only incorrect prediction by the stacked model.

Table 6: Stacked model classification report.

	precision	recall	f1-score	support
1	1.00	0.80	0.89	5
2	0.83	1.00	0.91	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	1.00	1.00	1.00	5
6	1.00	1.00	1.00	4
7	1.00	1.00	1.00	29
8	1.00	1.00	1.00	44
9	1.00	1.00	1.00	50
10	1.00	1.00	1.00	5
11	1.00	1.00	1.00	5
12	1.00	1.00	1.00	5
13	1.00	1.00	1.00	5

accuracy			0.99	172
macro avg	0.99	0.98	0.98	172
weighted avg	1.00	0.99	0.99	172

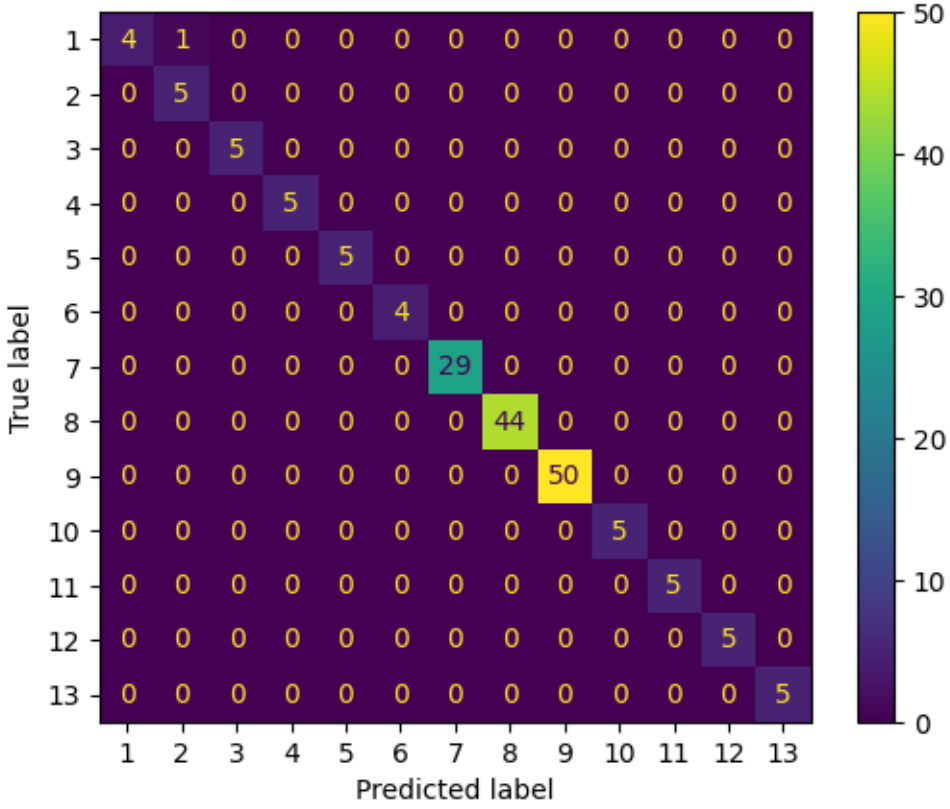

Figure 13: Stacked model confusion matrix.

2.6 Predictions

The final component of this project was to predict the steps of a provided set of coordinates. The coordinates to predict the step of were as follows:

[9.375,3.0625,1.51], [6.995,5.125,0.3875], [0,3.0625,1.93], [9.4,3,1.8], [9.4,3,1.3]

The associated predictions of each coordinate were as follows:

- For position [9.375, 3.0625, 1.51], the step is: 5
- For position [6.995, 5.125, 0.3875], the step is: 8
- For position [0, 3.0625, 1.93], the step is: 13

- For position [9.4, 3, 1.8], the step is: 6
- For position [9.4, 3, 1.3], the step is: 4

Finally, the predictions were plotted on the original plot to ensure that they were in their appropriate locations, and upon visual inspection, it is evident that the stacked model was successful in predicting the classes of each of these points.

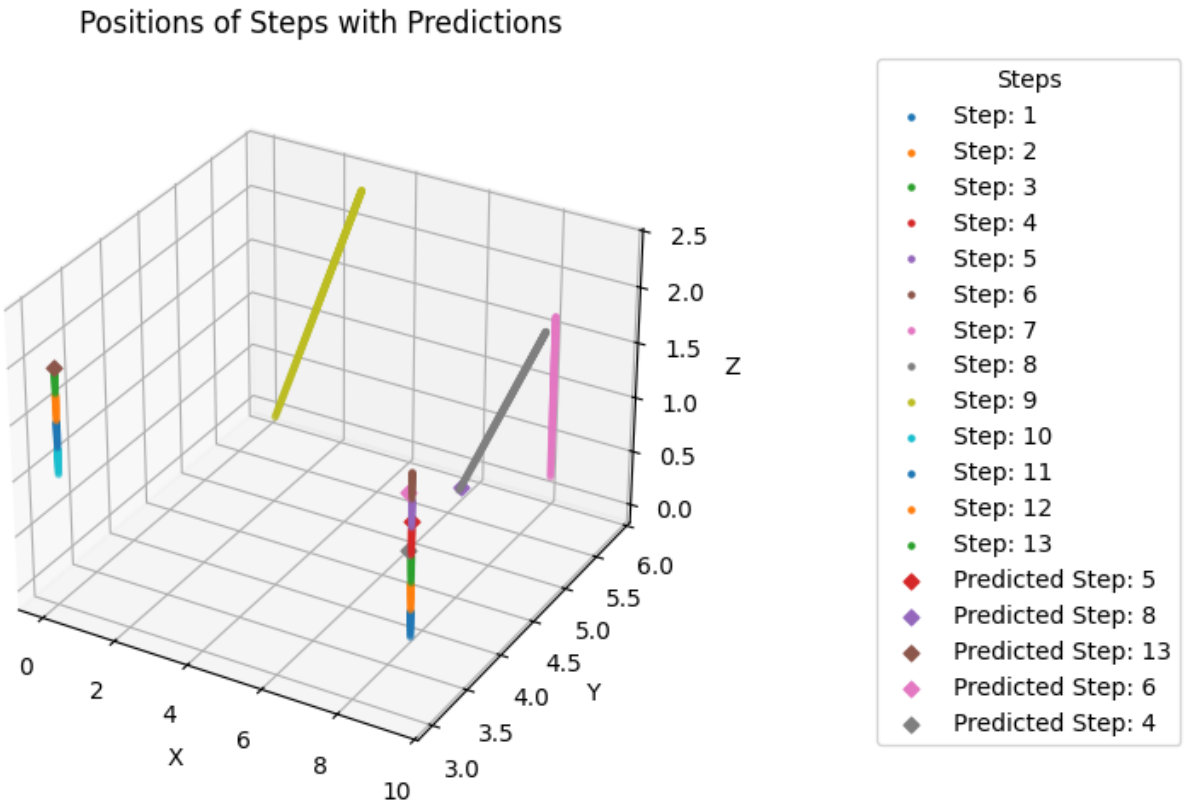

Figure 14: 3D scatter plot of original data with predicted data points.

3 Conclusions

With the completion of this project, a better understanding of the various different machine learning models was gained, in addition to how to properly implement data pre-processing techniques and data visualization techniques. The first components of this project were to visualize

the data, where it was learned from a 3D scatter plot that the points were in linear groups beside one another and were relatively neatly placed. This provided insight into the general structure of the data. The next steps consisted of identifying correlations, where it was learned there were no strong correlations between the feature variables, but there was a moderately strong correlation between the X variable and the Step output. Following the correlation analysis, the data was split into training and testing data sets by utilizing stratified sampling to ensure the proportions of the Step variable were identical in the training and testing data sets avoiding inadequate numbers of specific points in each set. With the data having been processed, different models were implemented and assessed. The best performing models were the Decision Tree Classifier and the K Nearest Neighbours Classifier each having only predicted one step incorrectly in the testing data sets. These models were utilized to build the stacked model, which performed the same as the original models. The stacked model was finally used to predict a provided list of data points, which based upon visual inspection, it successfully accomplished.