




**Department of
Aerospace Engineering**
Faculty of Engineering
& Architectural Science

Semester (Term, Year)	F2024
Course Code	AER 850
Course Section	2
Course Title	Intro to Machine Learning
Course Instructor	Dr. Reza Faieghi
Submission	Project Report
Submission Due Date	Oct. 20 th , 2024
Title	Project 1 Report
Submission Date	Oct. 20 th , 2024

Authors/Contributors:	Student Number (XXXX12345):	Signature*:
Ahad Razack	8108	

By signing the above you attest that you have contributed to this submission and confirm that all work you contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, and "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Academic Integrity Policy 60, which can be found at www.torontomu.ca/senate/policies/

Table of Contents

Introduction	2
Data Visualization.....	2
Correlation Analysis	5
Data Preparation	5
Model Selection and Tuning.....	6
Model Performance Analysis	7
Stacked Model Performance	9
Model Evaluation	10
Conclusion.....	10
GitHub Submission	11
References.....	12

Introduction

One key area where machine learning can be applied in the aerospace sector is through the use of augmented reality. Machine learning models can be used to make AR modules more efficient and accurate, allowing users to gain better insights while performing a variety of tasks, such as maintenance, assembly, and design.

The following report details the process for developing a machine learning model to predict the corresponding maintenance step for a part, given a set of X, Y, and Z coordinates. The data is organized into 4 columns, 3 for each of the coordinate axes, which are the features, and 1 for the maintenance step, which will be the label, or target. There are a total of 13 unique maintenance steps. This is an example of a supervised learning, multi-classification problem.

Data Visualization

Before any effort is made to train a model, it is first necessary to gain an understanding of the raw data. This was accomplished by loading the data contained in a .csv file into a pandas dataframe for effective manipulation and analysis.

The first thing to note is the integrity of the data. From the figure below, there are no missing or null values, and as such imputation will not be required.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 860 entries, 0 to 859
Data columns (total 4 columns):
#   Column  Non-Null Count  Dtype
---  -
0   X        860 non-null    float64
1   Y        860 non-null    float64
2   Z        860 non-null    float64
3   Step     860 non-null    int64
dtypes: float64(3), int64(1)
memory usage: 27.0 KB
```

Figure 1: Results of the df.info() method on the raw data

The following figure presents the distribution of the classes in the data.

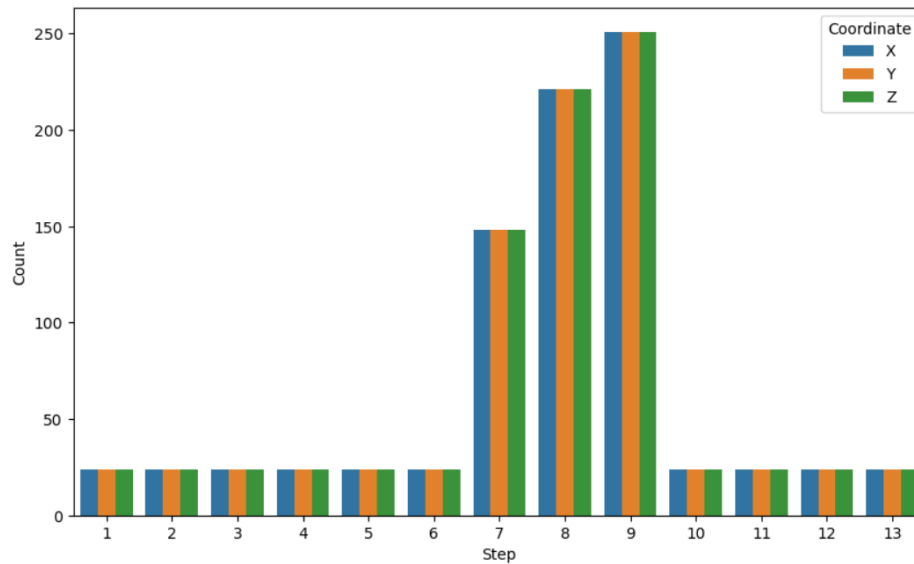


Figure 2: Distribution of each step in the dataset

From figure 2, it is readily apparent that steps 7, 8, and 9 occur far more frequently in the data than the others, which are all represented equally. This could imply that these steps are more complicated, and have many fine adjustments associated with them.

At this point, the raw data was understood enough to create separate train and test sets. To do this, stratified sampling was utilized to maintain the distribution of steps shown in figure 2.

Further exploration was conducted using the separated training set. Presented next are the boxplots of each coordinate, as well as their distribution.

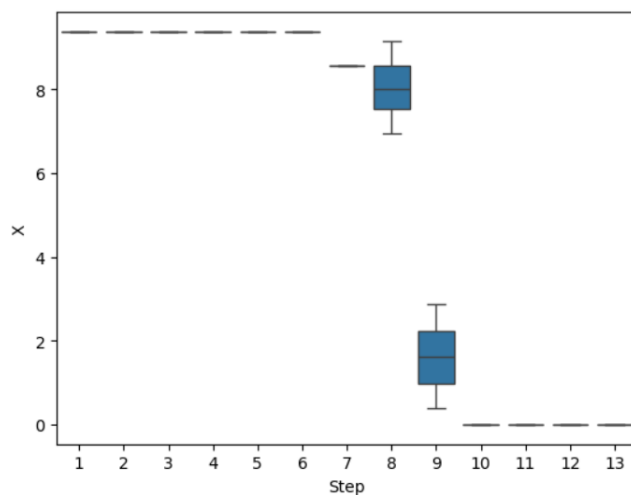


Figure 3: X-coordinate boxplot

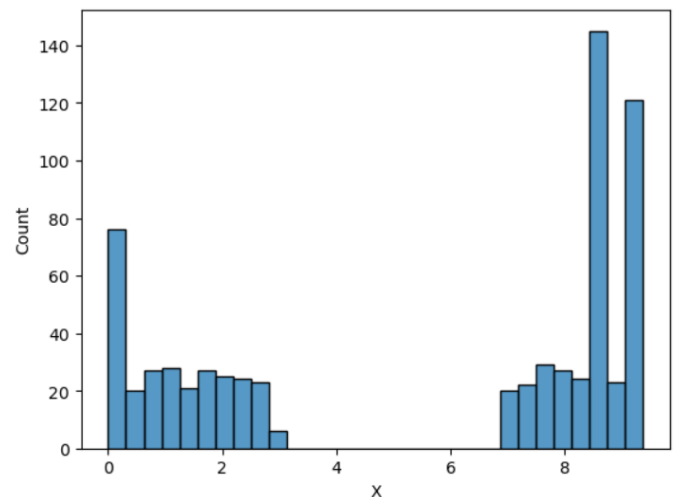


Figure 4: X-coordinate distribution

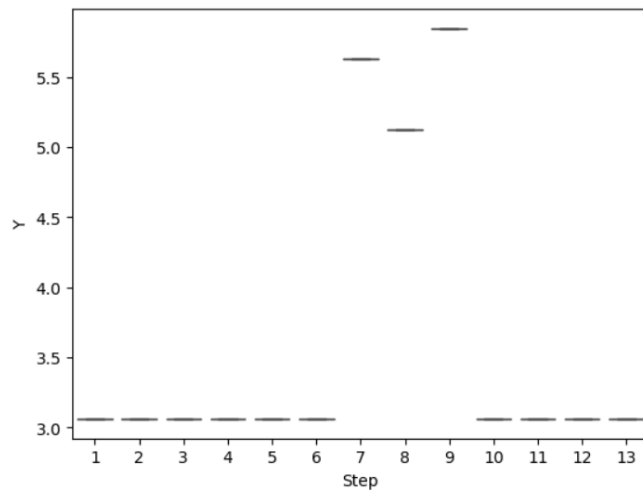


Figure 5: Y-coordinate boxplot

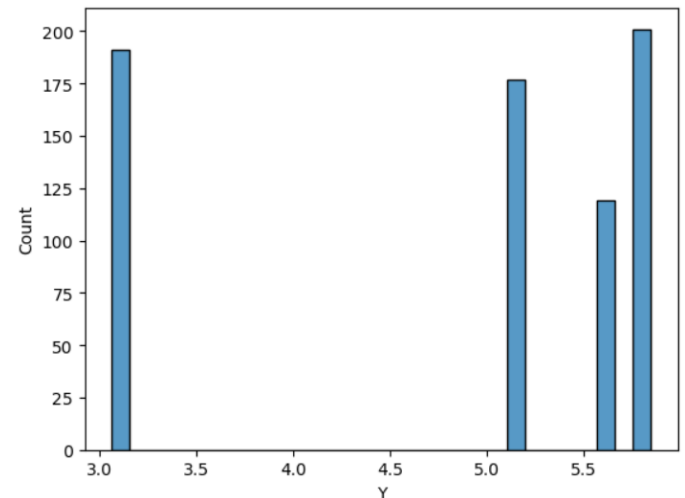


Figure 6: Y-coordinate distribution

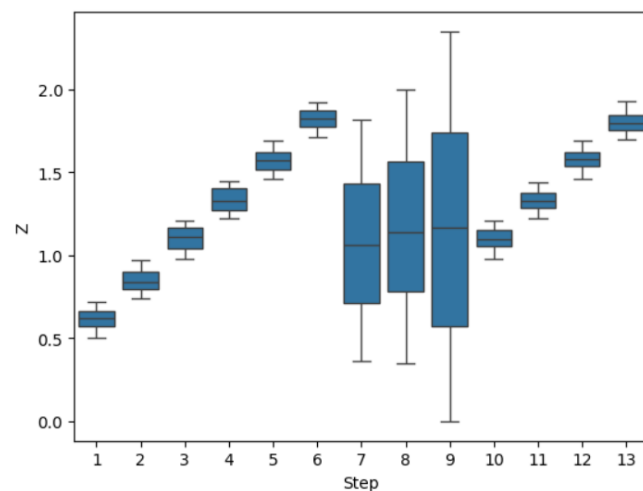


Figure 7: Z-coordinate boxplot

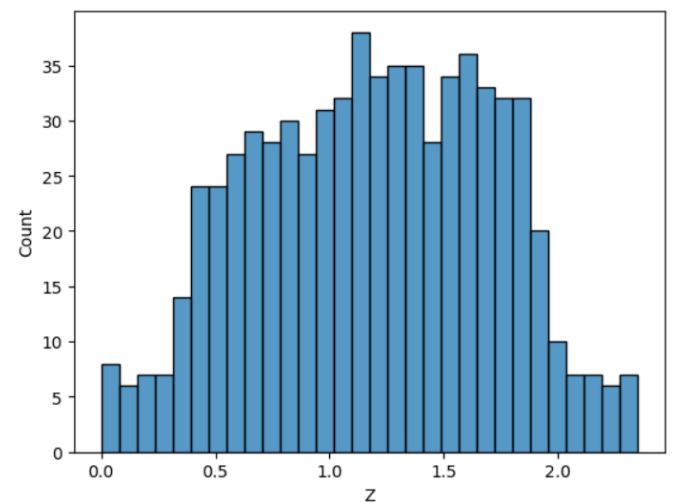


Figure 8: Z-coordinate distribution

From figures 3, 5, and 7, a few interesting observations can be made. Firstly, for each step, the range of the X and Y coordinates is very small, often 0. By contrast, there exists more “movement” the Z axis, especially for steps 7, 8, and 9. This could imply poor correlation between the Z-coordinate and the corresponding step, as there is a wide range of possibilities for each step. This will be confirmed in the correlation analysis in the proceeding section.

Secondly, the scale of the 3 axes differ. For example, the X-values range from about 0 to 9, whereas the Z-values only from 0 to 2.5. Due to these differences, scaling of the features will be required. The decision to use a StandardScaler or a MinMax scaler will be in a future section on data preprocessing.

Correlation Analysis

To perform the correlation analysis, the `df.corr()` method of was used, and the results were visualized using seaborn's heatmap, presented next.

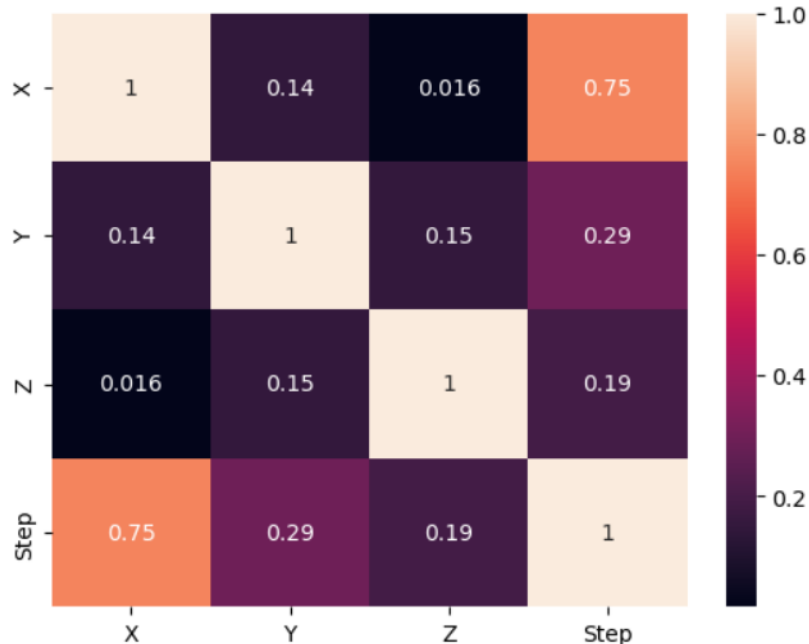


Figure 9: Correlation matrix of the dataset

Interpreting this heatmap highlights the strong correlation between the X-coordinate and the corresponding maintenance step. It also shows that despite the Y and Z coordinates showing a weaker correlation with the target, none of the features demonstrate strong correlation with each other, implying that they are linearly independent. As such, no features will be dropped from the training data, to provide the model with more information to make predictions.

Data Preparation

As mentioned previously, the data will need to be scaled before it can be used to train a model. The two most common methods for data scaling are standardization, or normalization. Standardization is typically used when the data exhibits a gaussian, or bell-shaped distribution. From figures 5, 7, and 9, it is evident that only the Z-coordinates exhibit this behavior. For simplicity, a min/max scaler was used for all 3 features. The decision to scale the features differently could be a potential hyperparameter for further model tuning but will not be explored in this project.

Model Selection and Tuning

The following section details the selection and tuning of 5 machine learning models from the scikit-learn library. The first 3 were developed using grid search cross-validation, one utilized random search cross-validation, and the final model is a stacked classifier using logistic regression as the final estimator. The ranges of the hyperparameters were adjusted until the best parameters lay somewhere in the middle of the given range, indicating that the range of optimal parameters had been found. Before detailed analysis of the performance of each model was conducted, the `best_score_` parameter was utilized to gain a preliminary understanding of the accuracy of each model. For each model, 5 folds were utilized.

The first model chosen was a random forest classifier. This model was chosen for its native ability to handle multiclass classification problems such as this one. The parameters used were `n_estimators`, `max_depth`, `min_samples_split`, and `min_samples_leaf`. After tuning, the model was able to achieve an accuracy of 98.98%.

It is important to note that of all the models trained, the random forest took by far the most time every time it was fit. The biggest influence seemed to be the `n_estimators` parameter, where higher values led to longer runtimes.

The next model selected was a support vector classifier. In contrast to the random forest classifier, a support vector classifier is just a binary classifier, and to handle a multiclass problem, it essentially trains multiple binary classifiers using a one vs rest scheme [1]. The hyperparameters used in this model were `C`, `kernel`, and `gamma`. In contrast to the random forest, this model only took a few seconds to train. After tuning, the accuracy was also 98.98%.

The last model trained using grid search was a complement naïve-bayes classifier. This model was selected upon inspecting the following flowchart from sci-kit learn. It suggests a naïve-bayes classifier for classification problems with < 100K samples. It was further suggested on the Scikit-Learn documentation that the complement classifier was better-suited to imbalanced datasets.

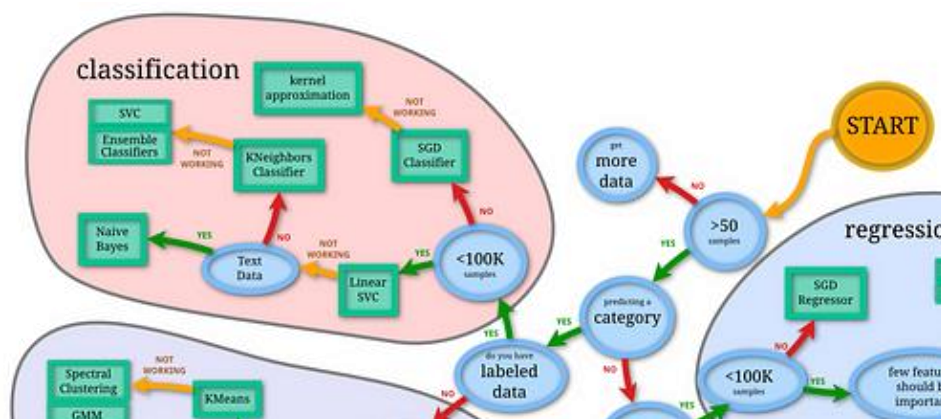


Figure 10: Sci-kit learn model selection flowchart

The amount of hyperparameters available for this model is a lot lower than for the previous models, so the tuning process was relatively quick. Due to the limited hyperparameter space, and low accuracy of 70%, this model was quickly dismissed as not being ideal. This was a surprising result, as it seemed to be the one best suited for the task.

The next model evaluated was another naïve-bayes classifier, this time the gaussian classifier. There were two main reasons for selecting this model. Firstly, it was to compare the two naïve-bayes classifiers and determine if the interpretation of the use case for the complement nb classifier was wrong. Secondly, this model only has one tunable hyperparameter, that has type float. As such, it was best suited for randomized search cross validation. This model also provided an accuracy score of 98.98%. This does confirm that the reason for initially trying a complement nb classifier was wrong.

Model Performance Analysis

After training 4 models, it was then necessary to evaluate their performance against each other. To do this, 3 parameters were used. The first, as mentioned before, was accuracy. Accuracy is a very straightforward metric, which just takes the number of correct predictions and divides it by the total number of datapoints. This can sometimes be a problematic metric when the dataset is highly unbalanced, as the model can achieve high accuracy by just always predicting the most represented class in the dataset. As the dataset in this project is evenly distributed, with only a few steps with significantly more instances than the others, this is a good metric to evaluate the models on.

The next metric is precision, which is a measure of how many positive predications are actually positive [2]. Precision should be prioritized for cases where false positives are more costly. In this project, this can be conceptualized as when the model makes a prediction for a step, it is very confident in its decision for that step, meaning its criteria for making the prediction is very narrow.

The final metric is f1 score, which is the harmonic average of precision and recall [2]. Recall is a measure of how many correct positive predictions were made [2]. Recall should be prioritized in cases where false negatives are more costly. In this case, it can be thought of as the model not identifying a set of coordinates as a maintenance step, even if they correspond to one.

In this case, accuracy and precision should be the most important metrics for evaluating model performance. Accuracy, as the model should be able to make correct predictions. Precision was prioritized over recall, as its more important for the model to be confident in its predictions than it missing predictions. For this project, its better for the model to not predict a maintenance step at all, than to make an incorrect prediction. Missing a step just means the model isn't as effective as it could be, but an incorrect prediction could lead to worse training, as the technicians don't really learn anything from being given false information.

With the choice of metrics outlined, the results for the 4 models are presented below. These values were generated by making predictions on the test set.

Table 1: Classification metrics for random forest classifier

	Accuracy	Precision	F1-score
Macro average	98.8372%	97.4359%	96.6478%
Weighted average		99.0310 %	98.8162%

Table 2: Classification metrics for support vector machine classifier

	Accuracy	Precision	F1-score
Macro average	100%	100%	100%
Weighted average		100%	100%

Table 3: Classification metrics for complement naïve-bayes classifier

	Accuracy	Precision	F1-score
Macro average	70.3488%	84.2782%	26.3142%
Weighted average		83.2200 %	63.6541%

Table 4: Classification metrics for gaussian naïve-bayes classifier

	Accuracy	Precision	F1-score
Macro average	99.4186%	98.7179%	98.2018%
Weighted average		99.5155%	99.4035%

A few noteworthy results. Firstly, it appears that the support vector machine was able to achieve 100% in all metrics. Far from indicating a perfect model, this result can be explained by the small size of the test set, with only 172 observations. When the classification report is generated for the training set, which has 4x the number of observations, the model still performs the best, just not as spectacularly as these values indicate.

As a result of this evaluation, the support vector machine has been selected as the most optimal model. Below is a confusion matrix of the test set predictions for this model.

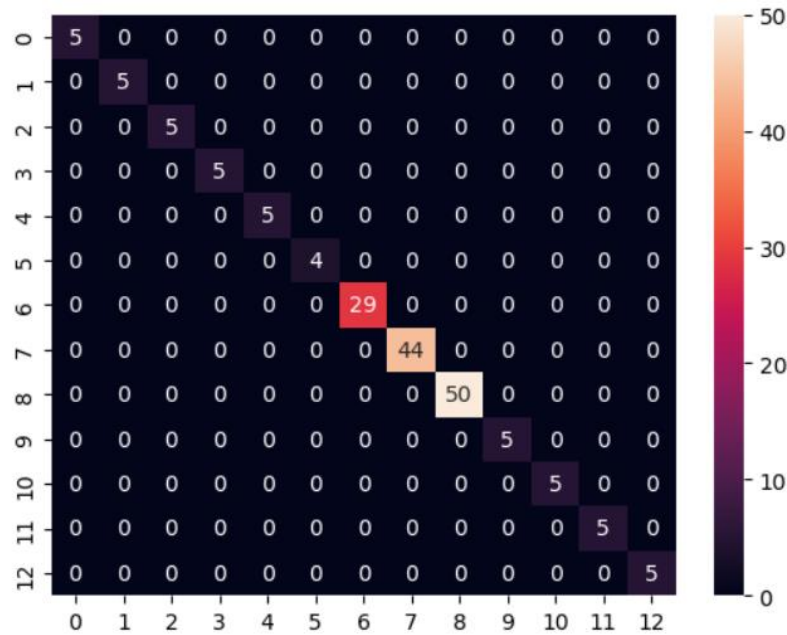


Figure 11: Confusion matrix for support vector machine

From the above figure, the model made no incorrect predictions, but it also just didn't make a lot of predictions, with only 5 for most classes. However, it did perform just as well for more represented classes, indicating that its effectiveness could scale for more observations.

Stacked Model Performance

Having selected an optimal singular model, a stacked classifier was developed, to investigate if combining different models would increase accuracy. It was decided to combine the worst-performing models, as this would make the improvement due to model stacking more apparent. The models were the random forest classifier, and the cnb classifier. The classification results are tabulated below.

Table 5: Classification metrics for gaussian naïve-bayes classifier

	Accuracy	Precision	F1-score
Macro average	99.4186%	98.7179%	98.2018%
Weighted average		99.5155%	99.4035%

From table 5, it seems the performance of the stacked model is identical to that of the random forest classifier. In this case, it seems as if model stacking doesn't increase accuracy, but just performs as well as the best performing model in the stack.

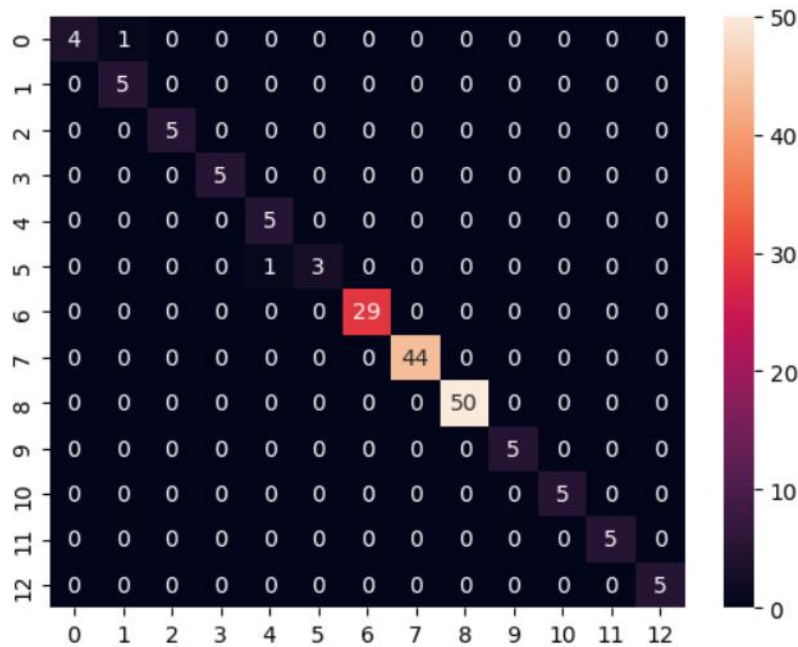


Figure 12: Confusion matrix for stacked model

Model Evaluation

Having decided on a final model, it was then packaged in a joblib file, and loaded into an empty notebook and used to make predictions on a given evaluation set. These results are summarized in the table below.

Table 6: Final model predictions on provided coordinates

Coordinates	Predicted Step
[9.375, 3.0625, 1.51]	5
[6.995, 5.125, 0.3875]	8
[0, 3.0625, 1.93]	13
[9.4, 3, 1.8]	6
[9.4, 3, 1.3]	4

Conclusion

In conclusion, this project was very successful in its goal of garnering a comprehensive understanding of the machine learning model development pipeline. It demonstrated the importance of preliminary data analysis to gain an understanding of the raw data landscape. This aids in identifying important distributions that needs to be preserved across data splits, as well as the correlation between features and the target, and amongst the features as well. The project also illustrated the effectiveness of hyperparameter tuning to generate a highly accurate model. Lastly, through the use of classification reports and confusion matrices, the project showed how various model performance metrics can be compared.

GitHub Submission

Below is a link to the repository for this project. All of the code is located in 'Project 1.ipbny'. The final model is loaded and evaluated in 'Final Predictions.ipbny'.

<https://github.com/CaptainAhad/Project-1>

References

[1] A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed. Sebastopol, CA: O'Reilly, 2019.

[2] R. Faieghi, "L5 - Classification Problems." Toronto Metropolitan University, Toronto, 2024