[redacted]

COMP3009 Assignment

HUZAIFA F KHAN 20188850

CURTIN UNIVERSITY SEMESTER 2 2023

Introduction

Data mining is the study and practice of investigating large amounts of data in order to retrieve, analyse and learn new information. This objective of this assignment was to dive deep into a given dataset comprising of 82333 samples, illustrating vital information about 8 different malicious activities, as well as activities classified as normal. The aim of the report was to tackle a twofold problem; the first problem was to distinguish whether an activity is malicious or not, and the second problem was to distinguish what specific type of malicious activity has been detected.

The purpose of this analysis was to create and train the data, so a model is created capable of predicting whether an activity is malicious or not and what type of malicious activity it is. The goal was to achieve an accuracy greater that 75%. A programming approach was implemented to clean, analyse, and train the model. There were two datasets provided to us initially: a training set and a test set. This made our analysis streamline as we did not need to split the data using manual means and instead, we were already given the two datasets beforehand.

The analysis was split into two main parts: data preparation and data classification. During initial exploratory data analysis, we cleaned, transformed, and prepared the data for classification. This was done so that there are no anomalies and irrelevant data present, as this would have affected our accuracy rating. Once we had prepared our data, we implemented 3 different classification techniques to our processed datasets and cross validated these techniques using k-fold cross validation to test the accuracy and validity of our accuracy predictions.

In the data preparation part, irrelevant attributes were removed from the dataset to help increase the accuracy of the model, and certain attributes were changed to a state (numeric, categorical etc) that was better suited to it. The training data was then standardised using many different techniques such as “select_dtypes” to select only the numeric attributes, and StandardScaler() for standardising features and removing mean and standard deviation. The training data was also normalised using min-max scaling using the “MinMaxScaler” function. This was done for numeric values that had a range [0,1], and computed the minimum and maximum values required for scaling. The processed training data was then shuffled before splitting it into a test dataset. This was done to ensure that the order of the data is randomised, accounting for a better split in the test dataset.

In the classification part of the report, three techniques were implemented: k-Neighbours, Decision Tree, and Naïve Bayes classification models. These 3 models were implemented on the processed training dataset to identify which model would obtain the highest level of accuracy. Classification reports for each modelling technique were formed, and then predictions were made using the 2 models with the highest accuracy. The prediction was implemented to predict the missing class labels in the test data set that was provided, and these predictions were then outputted into a csv file using the “to_csv” function.

The report dives deep into how the analysis and model was implemented and shows how the data was prepared for classification. It explains the classification techniques used and why they were used, as well as illustrating the accuracy results obtained through the

analysis. From the accuracy ratings, a final decision is made as to which classifier was best suited to our dataset and which one provided us with the most accurate and precise predictions. It provides an insight into how data mining techniques can be applied in the real world, with realistic datasets to help identify patterns, trends and improvements that can be made to aid with the improvement of one's business.

Methodology

A programming approach was used to conduct this analysis, with Python being the choice of language to be implemented. Inbuilt models such as pandas and sklearn were used to help develop our data frames and classification models. We were given csv files that were already split into training and test datasets and used pandas "read_csv" function to store these datasets into our workspace and named them trainData and testData. These were then stored into a dataframe using the function pd.DataFrame() and were renamed traindf and testdf.

Data Preparation

Initially, the dataset was given in a csv file which was loaded into our Jupyter notebook working environment through the function "pd.read_csv" and after loading the data, it was stored as a dataframe and was named "traindf" & "testdf".

Attribute Statistics

To investigate the attribute statistics of the dataset, we first identify how may attributes and instances are present in the dataset. From the output of the respective code, we identify that the training dataset has 82332 instances and 45 attributes. We conduct further analysis to show how much of the data is numeric and how much is nominal, through the describe function.

Data Type

The data type of an attribute refers to the nature that the attribute can take. For instance, the age of males and females can be considered to be numeric data and what colour hair an individual has can be considered to be categorical data. It was vital in our analysis that each attribute was assigned with the correct data type, as if they were incorrectly attributed, the analysis and accuracy of the models implemented would have been significantly affected. Panda has an inbuilt capacity to automatically assign datatypes to the given attributes in the dataset. This can be helpful, however more often than not transformations have to be done, as numerical values can be classified as categorical, and/or vice versa. Without addressing these issues, the accuracy from the classification models will be affected, as the data types would be meaningless to their attribute variable due to it being incorrectly classified. The table show the initial state of the variables in the dataset, with the dataset containing 82332 rows.

Attribute	Data Type
id	Int64
Dur	Float64
Proto	Object
service	Object
state	Object

Spkts	Int64
dpkts	Int64
Sbytes	Int64
Rate	Float64
Sttl	Int64
Dttl	Int64
Sload	Float64
Dload	Float64
Sloss	Int64
Dloss	Int64
Sinpkt	Float64
Dinpkt	Float64
Sjit	Float64
Djit	Int64
Swin	Int64
Stcpb	Int64
Dtcpb	Int64
Dwin	Int64
Tcprtt	Float64
Synack	Float64
ackdat	Float64
Smean	Int64
Dmean	Int64
Trans_depth	Int64
response_body_len	Int64
ct_srv_src	Int64
ct_state_ttl	Int64
ct_dst_ltm	Int64
ct_src_dport_ltm	Int64
ct_dst_sport_ltm	Int64
ct_dst_src_ltm	Int64
is_ftp_login	Int64
ct_ftp_cmd	Int64
ct_flw_http_mthd	Int64
ct_src_ltm	Int64
ct_srv_dst	Int64
is_sm_ips_ports	Int64
attack_cat	object
label	Int64

To identify which numeric attributes should be categorical, we first looked at the feature description that was provided to us, to see if we can identify any transformations that we need to do just by visual inspection. We can identify for numeric variables that should be categorical from the feature description. These 4 attributes are “is_ftp_login”, “ct_ftp_cmd”, “is_sReportm_ips_ports” and “label”. The code below confirms this visual inspection:

```
for col in trainData.select_dtypes(include=np.number):
    if trainData[col].nunique() < 6:
        print(col)
        trainData[col] = trainData[col].astype(object)
```

The following code iterates over columns within the training dataset that are considered to be numeric data types and checks whether the number of unique values in the columns that are assigned to a numeric data type are less than 6, and then prints the column name. After identifying these variables, we perform a transformation to convert the numerical data types to categorical as follows:

```
trainData['is_ftp_login'] = trainData["is_ftp_login"].astype('category')
trainData['ct_ftp_cmd'] = trainData["ct_ftp_cmd"].astype('category')
trainData['is_sm_ips_ports'] = trainData["is_sm_ips_ports"].astype('category')
trainData['label'] = trainData["label"].astype('category')
```

Furthermore, there are some object attributes that should also be categorical and need transformation. We can identify from the feature description that proto, service, state, and attack_cat are variables that should be categorical, as they illustrate a certain set of “items” the attribute can be. For example, attack_cat has a given set of names it can be, such as normal, generic or exploits. The code below ensures that the object variables are treated as categorical and not anything else:

```
for col in trainData.select_dtypes(include=object):
    print(col)
    trainData[col] = trainData[col].astype(object)
```

From the output of this code, we can identify 4 object variables that need to be converted to categorical (as mentioned above). To further ensure that all our object attributes are treated as categorical, we perform a manual transformation as follows:

```
trainData['proto'] = trainData["proto"].astype('category')
trainData['service'] = trainData["service"].astype('category')
trainData['state'] = trainData["state"].astype('category')
trainData['attack_cat'] = trainData["attack_cat"].astype('category')
```

Ensuring the data types of each attribute were correct was pivotal for the progression of our analysis, as if they were undealt with and not transformed, then the validity and accuracy of our classification models and predictions would be greatly affected, providing us with inaccurate and unprecise models.

Dealing with Missing Data

Given that the task was to simulate a real-world data mining problem, it was very plausible that the given dataset would contain missing values and entries. It was important to ensure that if the dataset contained missing values, they were dealt with in an appropriate manner.

However, it was surprising to identify that there were no missing values in the dataset. This is seen through the output of the training data set using the function .info(), where all the respective attributes count is “non-null”. Non-null refers to the condition of the attribute wherein a value exists. In regard to our dataset, it confirms that there is indeed no missing data entries present for each attribute. This was a benefit in our data preparation as it

ensured we did not need to remove or generate new values for any missing data, saving time and making our processed training dataset much more efficient and realistic.

Checking and Dealing with Duplicates

A duplicate is when data points are repeated. It is imperative to deal with duplicated instances and attributes as it may introduce anomalies and irregularities in our final classification model. For instance, if one attribute is duplicated many times, then the final accuracy and prediction will be biased towards that attribute and/or will have a major effect on the value of the accuracy.

To identify whether the dataset contains duplicates or not, the drop_duplicates() function was used to identify and remove any given duplicates.

```
trainData = trainData.drop_duplicates()
```

We can conclude that there were no duplicated columns in the dataset however, and this was done merely as a precaution. From visual inspection using the head() function, we can see that each column is different from the other and the values within are also different, ensuring no duplicates were present from the start.

Dealing with Irrelevant Attributes

An irrelevant attribute is something that does not provide any beneficial and relevant information to the dataset and is not relevant to the classification model and if implemented in the model, the accuracy and validity of the model can be greatly affected. Having data that is not necessary within the model is called “noise”. Increased noise in data has a negative effect of analysis and predictions, as it allows for anomalies to be present and for instances that have no relation to the desired outcome of the analysis to affect the models. Therefore, it was essential that attributes that serve no significant purpose to our classification models were discarded from the data set in the preparation stage of our analysis.

From visual inspection, we can see that “id” is just a row of numbers used as a label for the rest of the attributes and served no purpose for our classification and was hence discarded. As our aim for the project was to predict one what type of activity was recorded and two, if the recorded activity was malicious then what type of malicious activity was it, we removed the attributes “proto”, “service” and “state” as well. The reasons as to why these attributes were removed are mainly due to the fact that initially they were implemented in our classification models, and the models’ returned errors as well as a very low accuracy rating. From this, it was identified that these 4 attributes were greatly affecting our models, and were there discarded from our training data using the drop() function as follows:

```
trainData = trainData.drop(columns=['id', 'proto','service','state','label'])
```

Through removing the respective attributes, our dataset has decreased in size substantially. This is an advantage as we have cleared out any noise from the training data and made it adequate and ready for data classification.

Standardisation and Normalisation

The standardisation of data is essential for data to be accurately modelled in our classification models. It is a vital pre-processing step that aids with the performance, stability, and accuracy of our machine learning classification model. In our case, standardisation was only done to numeric as they are the ones involved in our analysis predominantly. The StandardScaler() function was used to scale the columns with a numeric data type and computes the mean and standard deviation necessary for standardisation. This ensures that for all our attributes, the data type is consistent with its content and format. The code below demonstrates how this was achieved:

```
trainData_numeric = trainData.select_dtypes(include='number')
scaled_trainData_numeric = StandardScaler().fit_transform(trainData_numeric)
trainData[trainData_numeric.columns] = scaled_trainData_numeric
```

Normalising in a broader sense is the process of organising data in a given dataset and is used to scale the contents of an attribute so that the range is decreased. Using the MinMaxScaler() function, we transform the numeric columns of the training dataset so that they are between the ranges 0 and 1, and the fit.transform() function identifies the minimum and maximum values for each attribute. The code that implements this process is shown below:

```
scaled_trainData_numeric = MinMaxScaler().fit_transform(trainData_numeric)
trainData[trainData_numeric.columns] = scaled_trainData_numeric
```

We also shuffled the rows within the training data set randomly, by using the “sample” function. This was done to randomise the order of the data before we are splitting the data into training and test data sets. Through the shuffling of the rows, it makes sure that the data is evenly distributed ensuring that there is less room for inaccuracy and bias in our classification.

Both these steps were crucial before implementing the classification models as it scales and standardises the attributes in the dataset, making the values consistent and in a standard format. We implemented both StandardScaler() and MinMaxScaler() to ensure that all of the different data types were accounted for, and that all of our data was scaled appropriately.

Data Classification

Training and Test Split

Following the data preparation stage of the analysis, the data was then split into training and test datasets. The target variable for our analysis is “attack_cat”; this variable contains the name of each category of attack (normal, fuzzers, analysis etc). The inbuilt function “train_test_split” was used to split the training data into two respective datasets that were named: trainData and testData. The split of the two datasets was 15%. This means that 15% of the data will be implemented in our testing and prediction phase, whereas 85% of the data will be implemented in training and modelling our classification techniques. The stratify

```
trainData, testData = train_test_split(trainData, test_size = 0.15, stratify = trainData["attack_cat"],

trainData["attack_cat"].value_counts()
```

function was used to make sure that the distribution of the data within “attack_cat” is even and approximately the same in both our datasets (training and test).

Furthermore, we conducted oversampling on our newly formed training and test datasets to separate “attack_cat”’s features from the rest of the data attributes. This is an essential step and ensures that our classification models will run correctly. Attack cat was assigned to Y_train and Y_test and was dropped from X_train and X_test. This division of features and labels is essential for the training and evaluation of the models we implemented and ensures that the two-fold problem that we are working to achieve is reached, as removing attack cat from one axis makes it easy to identify the label of the recorded activity, whether it is normal or malicious.

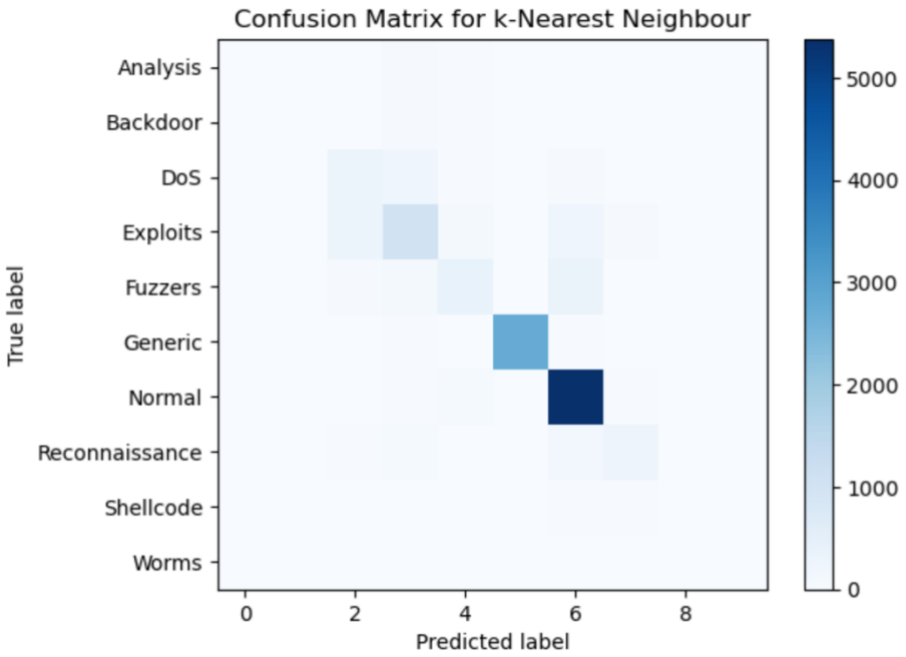
```
Y_train = trainData["attack_cat"]
X_train = trainData.drop("attack_cat",axis=1)
Y_test = testData["attack_cat"]
X_test = testData.drop("attack_cat",axis=1)
trainData.info()
```

k-Nearest Neighbour Classifier

The k-nearest neighbour classifier is a machine learning supervised classification model that uses how close a value is and uses this to make classifications about the target variable. Using the kNeighbours function, we implemented this classifier on the test data and evaluated its performance using accuracy and error rate. From the codes output, we can identify the error rate to be 18.70% and the accuracy to be 81.30%. To aid in visually seeing the output, we printed a classification report using the prediction of the kNeighbours classifier and Y_test, as shown below.

	precision	recall	f1-score	support
Analysis	0.00	0.00	0.00	16
Backdoor	0.00	0.00	0.00	7
DoS	0.45	0.40	0.42	676
Exploits	0.61	0.62	0.62	1629
Fuzzers	0.40	0.56	0.47	643
Generic	0.97	0.99	0.98	2777
Normal	0.97	0.87	0.92	6196
Reconnaissance	0.51	0.67	0.58	395
Shellcode	0.09	0.45	0.15	11
Worms	0.00	0.00	0.00	0
accuracy			0.81	12350
macro avg	0.40	0.46	0.41	12350
weighted avg	0.85	0.81	0.83	12350

The classification report provides an extensive summary of our model’s performance by each of the names of the attack_cat data attribute. We can see that the classifier achieved a very high accuracy score on the test data, indicating that the model is running at optimal rate and that our pre-processing of the data was well suited to this type of classifier.



From the k-Nearest Neighbour confusion matrix above, we can that most of our predictions lie withing the labels 4-6 and are either generic or normal for the type of activity.

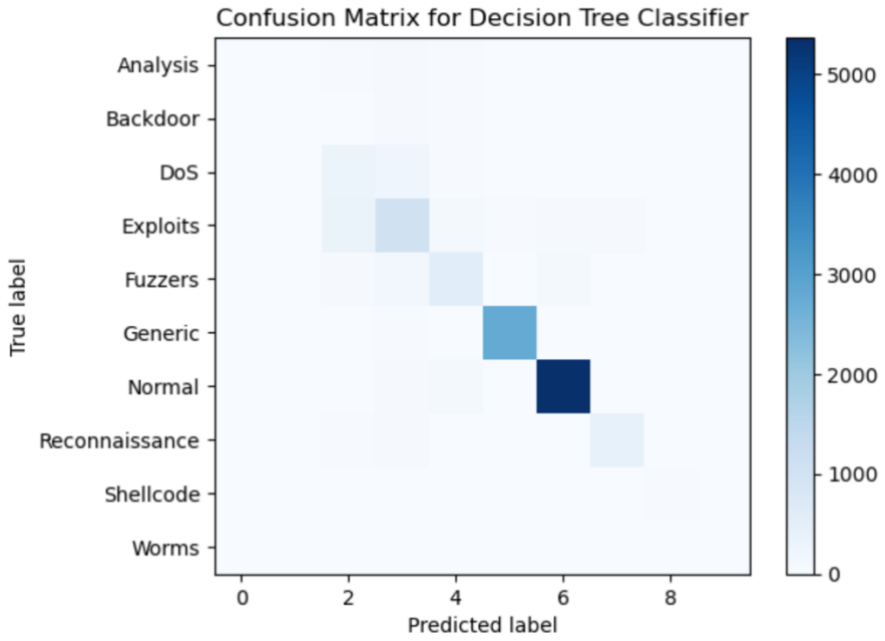
Decision Tree Classifier

The decision tree classifier is a versatile classification model and partitions the given training and test data based on the target variables attributes. In our model, the decision tree classifier is made to a random state of 50 and is then modelled on the training data set. Following the training of the data, the classifier is then used to make predictions on the test data set, with the error rate being 14.60% and the accuracy being 85.40%. The output was then printed in a classification report to help view the ratings of the model more clearly.

	precision	recall	f1-score	support
Analysis	0.08	0.35	0.13	23
Backdoor	0.01	0.06	0.02	16
DoS	0.52	0.39	0.45	813
Exploits	0.64	0.63	0.63	1694
Fuzzers	0.64	0.67	0.66	873
Generic	0.98	0.98	0.98	2820
Normal	0.97	0.96	0.97	5556
Reconnaissance	0.79	0.87	0.83	479
Shellcode	0.42	0.37	0.39	65
Worms	0.00	0.00	0.00	11
accuracy			0.85	12350
macro avg	0.51	0.53	0.51	12350
weighted avg	0.86	0.85	0.86	12350

We can see that the macro average of the model is 0.53, and the weighted average is the same as the accuracy. The accuracy of the model returned to be a high value, indicating that the implementation of the model was done correctly, and the data was processed adequately. We can also conclude that our data was better suited to this type of classifier

compared to the k-Nearest neighbour classifier, as the accuracy rating is much higher for the decision tree classifier.



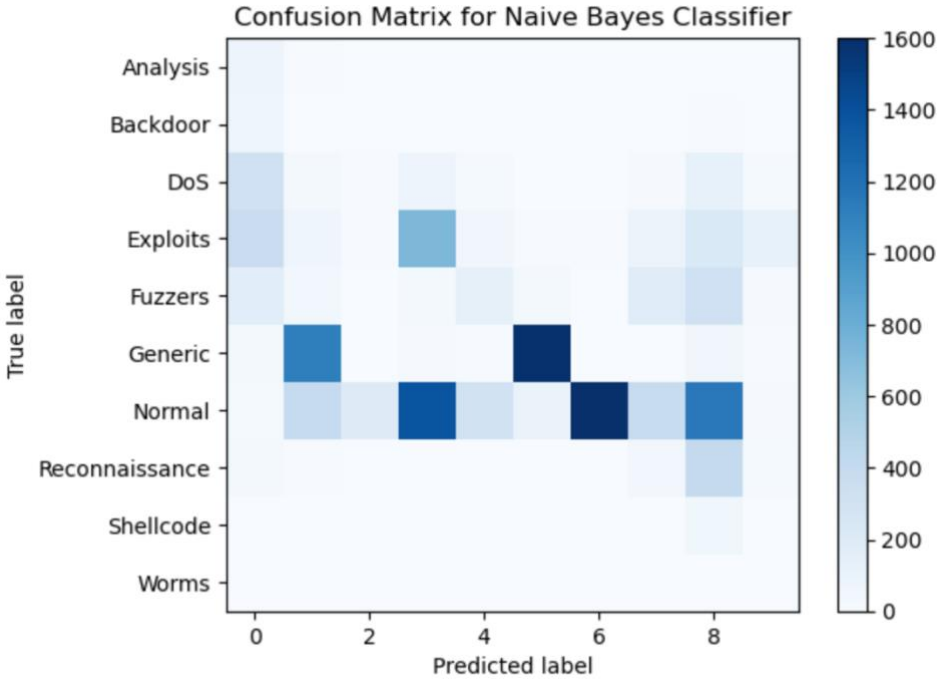
We can see that the confusion matrix for the decision tree classifier is quite similar to the Nearest Neighbours one. This makes sense as the accuracy ratings are also quite similar, meaning the predictions would also be around the same; this was confirmed by the confusion matrix.

Naïve Bayes Classifier

The naïve bayes classifier is based on the bayes theorem and calculates the probability of a given event occurring based on historical knowledge. The assumption "naïve" in this classification method means that the attributes are independent from each other which makes the calculation of the model simple. By using the "GaussianNB()" function, the training data is trained using the X_train and Y_train labels, with the output being the error rate which was 65.60% and the accuracy being 34.40%. The output was then printed in a classification report for ease of viewing.

	precision	recall	f1-score	support
Analysis	0.79	0.07	0.14	1097
Backdoor	0.02	0.00	0.00	1663
DoS	0.01	0.04	0.02	215
Exploits	0.44	0.33	0.37	2231
Fuzzers	0.15	0.25	0.18	531
Generic	0.56	0.91	0.70	1738
Normal	0.29	0.99	0.45	1621
Reconnaissance	0.09	0.07	0.08	721
Shellcode	1.00	0.02	0.05	2328
Worms	0.43	0.01	0.03	205
accuracy			0.34	12350
macro avg	0.38	0.27	0.20	12350
weighted avg	0.48	0.34	0.26	12350

From this we can see that the macro average is 27% and the accuracy of the model is quite low at 34%. The low accuracy value indicated that the data was not processed correctly to fit this type of classifier to the best of its ability.



From the confusion matrix above, we can see that there is no real trend and pattern for the predictions. This reflects the accuracy value correctly as if the value was high, there would be a distinct pattern on the confusion matrix, indicating that the predictions are correct.

Cross Validation

After implementing the classification models, we conducted cross validation of all 3 of the models to test each model's performance. This is a vital step in classifier comparison and selection as after cross validation, the models will be compared and the best two will be selected for the prediction step.

All 3 classification models underwent stratified k-Fold cross validation to evaluate the performance of each respective model. The dataset is divided into 10 stratified folds. This makes sure that each fold is consistent with the class distribution as the original training data. Then, the respective classifier is trained within a loop iterating over the specified number of stratified folds, while the validation set assesses the performance of the model. Once this is done, the accuracy scores of each model are printed 10 times. As this validation type iterates through all 10 folds, it is a comprehensive validation technique and allows for the model's performance to be assessed rigorously. Below are the results of the cross validation for each of the 3 classification models.

k-Nearest Neighbour	Decision Trees	Naïve Bayes
80.85%	86.25%	35.41%
81.85%	87.12%	35.34%
80.28%	85.09%	34.79%
81.28%	85.43%	35.73%
81.36%	85.08%	35.39%

81.51%	84.93%	33.79%
80.91%	84.03%	34.72%
81.13%	84.33%	35.96%
80.86%	84.31%	33.81%
80.89%	89.89%	34.89%

From the results of the cross validation, we can see that all 3 models have a similar accuracy rating as the one we got from their model accuracy rating. This means that as the validation process iterates through all the folds across the different subsets of the data, the accuracy rating stays the same validating the initial accuracy rating we obtained.

Classifier Comparison and Selection

After implementing the three different classification models and cross validating them through stratified kFold cross validation, we compared the accuracy and error rates of the models to come to a final decision as to which two classifier models to choose for our prediction evaluation. The accuracy and error rates of the respective models are listed below.

	k-Nearest Neighbours	Decision Tree	Naïve Bayes
Accuracy Score	81.00%	85.40%	34.40%
Error Rate	18.70%	14.60%	65.60%

From the above table, we can conclude that the best two models to implement our prediction evaluation with are: k-Nearest Neighbour and Decision Tree classifiers, with decision tree being the classifier with the highest accuracy rating and naïve bayes being the lowest.

Prediction

From the two csv files given to us initially, the test data file was 2 columns smaller than the original training data file. From exploratory data analysis, we can identify the two attributes missing to be attack_cat and label respectively. One of the aims of our project was to predict the missing class labels in the test dataset from the models we had implemented in the data classification phase.

In the first phase of the prediction evaluation, we added an empty column named “attack_cat” into the original test data set, and then did the transformed the variables that were changed in the training dataset to make our prediction consistent with the processed training data. We then created X_testFinal and Y_testFinal, with X_testFinal not containing the column attack_cat, as was done to the training data in the data preparation stage. This is shown below.

```
testDataFinal["attack_cat"] = ""
testDataFinal = testDataFinal.drop(columns=['id', 'proto','service','state'])
testDataFinal['attack_cat'] = testDataFinal["attack_cat"].astype('category')
testDataFinal['is_ftp_login'] = testDataFinal["is_ftp_login"].astype('category')
testDataFinal['ct_ftp_cmd'] = testDataFinal["ct_ftp_cmd"].astype('category')
testDataFinal['is_sm_ips_ports'] = testDataFinal["is_sm_ips_ports"].astype('category')
```

Creating X & Y Test

```
Y_testFinal = testDataFinal["attack_cat"]
X_testFinal = testDataFinal.drop("attack_cat", axis = 1)
testDataFinal.info()
```

The two classifiers being used for the prediction are k-Nearest Neighbours and Decision Tree, as they both had the highest accuracies out of the three, with 81% and 85% respectively. To create the prediction evaluation models for the two classifiers, we need to create a new prediction variable. This new predict variable is the printed to output the predicted labels alongside the true labels from the Y_testFinal set. This therefore allows us to compare between the model's predictions and the actual values of the model.

ID	Prediction 1: k-Neighbour	Prediction 2: Decision Tree
200	Normal	Fuzzers
201	Normal	Fuzzers
202	Normal	Fuzzers
203	DoS	Fuzzers
204	Normal	Fuzzers
205	DoS	Fuzzers
206	Normal	Fuzzers
207	Normal	Fuzzers
208	Normal	Fuzzers
209	Normal	Fuzzers
210	Normal	Fuzzers
211	Normal	Fuzzers
212	Normal	Fuzzers
213	Normal	Fuzzers
214	Normal	Fuzzers
215	Normal	Fuzzers
216	Normal	Fuzzers
217	Normal	Fuzzers
218	Normal	Fuzzers
219	Normal	Fuzzers
220	Normal	Fuzzers
221	Normal	Fuzzers
222	Normal	Fuzzers
223	Normal	Fuzzers
224	Normal	Fuzzers
225	Normal	Fuzzers
226	Normal	Fuzzers
227	DoS	Fuzzers
228	Normal	Fuzzers
229	Normal	Fuzzers
230	Normal	Fuzzers
231	Normal	Fuzzers
232	DoS	Fuzzers
233	Normal	Fuzzers
234	DoS	Fuzzers
235	DoS	Fuzzers
236	Normal	Fuzzers
237	Normal	Fuzzers

238	DoS	Fuzzers
239	Normal	Fuzzers
240	Normal	Fuzzers
241	Normal	Fuzzers
242	Normal	Fuzzers
243	Normal	Fuzzers
244	Normal	Fuzzers

Conclusion

The two stages conducting to implement the classification models were data preparation and data classification. In the data preparation phase, exploratory data analysis was conducted such as the removal of irrelevant attributes, transformation of the data types of certain attributes, dealing with missing and duplicate entries and standardisation and normalisation of the training data. Once the data was properly processed and ready for classification, three classification models were implemented: k-Nearest Neighbours, Decision Tree, and Naïve Bayes classifiers. From these three classifiers, the highest accuracy obtained was from the Decision Tree classifier, with it being 85%, followed by k-Nearest classifier with it being 81% respectively. The assignment specification stated that the most optimal desired accuracy rating was 75%, and the two out of the three models that we implemented achieved an accuracy rating higher than 75%. This indicates that our data was processed and made ready for classification correctly for most of our classification models. To further improve our accuracy, more extensive data preparation could have been implemented so that the Naïve Bayes model also achieved an accuracy higher than the desired.

References

1. Ray, S. (2017, September 11). *Naive Bayes Classifier Explained: Applications and Practice Problems of Naive Bayes Classifier*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

2. Gandhi, R. (2018, May 5). *Naive Bayes Classifier - Towards Data Science*. Medium; Towards Data Science. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

3. *What are Naive Bayes classifiers? | IBM*. (2023). Ibm.com. https://www.ibm.com/topics/naive-bayes

4. Raj, A. (2021, October 27). *An Exhaustive Guide to Decision Tree Classification in Python 3.x*. Medium; Towards Data Science. https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f#:~:text=The%20Decision%20Tree%20algorithm%20uses,used%20to%20train%20the%20algorithm.

5. Avinash Navlani. (2018, December 28). *Decision Tree Classification in Python Tutorial*. Datacamp.com; DataCamp. https://www.datacamp.com/tutorial/decision-tree-classification-python

6. *Python Machine Learning Decision Tree*. (2023). W3schools.com. https://www.w3schools.com/python/python_ml_decision_tree.asp

7. Simplilearn. (2020, July 14). *The Best Guide On How To Implement Decision Tree In Python*. Simplilearn.com; Simplilearn. https://www.simplilearn.com/tutorials/machine-learning-tutorial/decision-tree-in-python

8. *What is the k-nearest neighbors algorithm? | IBM*. (2023). Ibm.com. https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point.

9. Srivastava, T. (2018, March 25). *A Complete Guide to K-Nearest Neighbors (Updated 2023)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

10. Christopher, A. (2021, February 2). *K-Nearest Neighbor - The Startup - Medium*. Medium; The Startup. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4