Assignment 2

Instructions

How to run:

Python nids.py <filename.csv> <classification method>

Available classification names and descriptions:

1. lr_binary : binary Logistic regression
2. rf_binary : random Forest Binary
3. dt_binary : Decision Tree Binary
4. lr_multi : Logistic regression multi class
5. rf_multi : random Forest multi class
6. dt_multi : Decision Tree multi class

Initial Data processing

- drop columns: "Stime" and "Ltime" (time) , "srcip" and "dstip": unique constants – not relevant
- convert null / empty “attack_cat” to “non_attack”
- drop nan rows
- label encode the categorical columns

Data frame after pre-processing

Data shape: (123690, 45)

Data head:

sport dsport proto state dur sbytes dbytes sttl dttl ... ct_srv_src ct_srv_dst ct_dst_ltm ct_src_ltm ct_src_dport_ltm ct_dst_sport_ltm ct_dst_src_ltm attack_cat Label

0 55490 53 udp CON 0.001050 146 178 31 29 ... 3 4 3 2 1 1 1 None 0

1 26756 21 tcp FIN 4.264797 2934 3738 31 29 ... 5 5 15 11 5 5 8 None 0

5 56992 55107 tcp FIN 0.016463 2958 32408 31 29 ... 11 16 9 2 1 1 1 None 0

10 35681 45031 tcp FIN 0.032683 3614 45044 31 29 ... 8 2 2 1 1 1 1 None 0

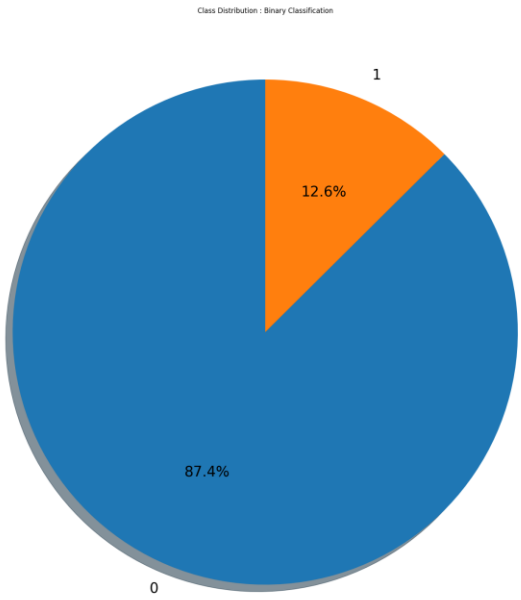
11 0 0 egp INT 0.000005 180 0 254 0 ... 8 6 5 4 4
4 4 Fuzzers 1

[5 rows x 45 columns]

Class Distribution

Done to visually understand the various class distributions under label and cat_attack

Label:

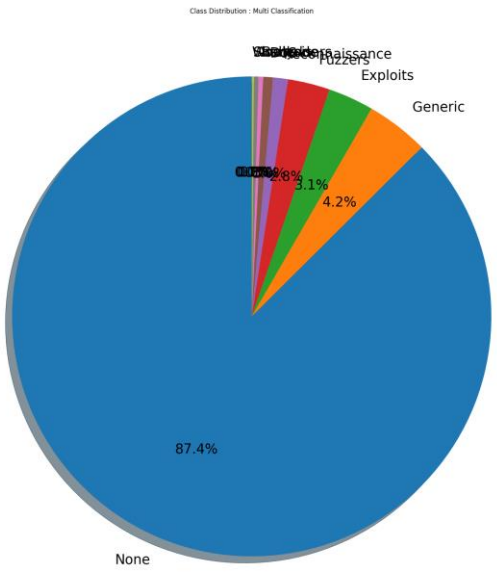


Normal (No Intrusion - 0 Class): 87.4%

Ab Normal (Intrusion - 1 Class): 12.6%

The Binary Data is IMBALANCED – in that the distribution is uneven

Attack_cat



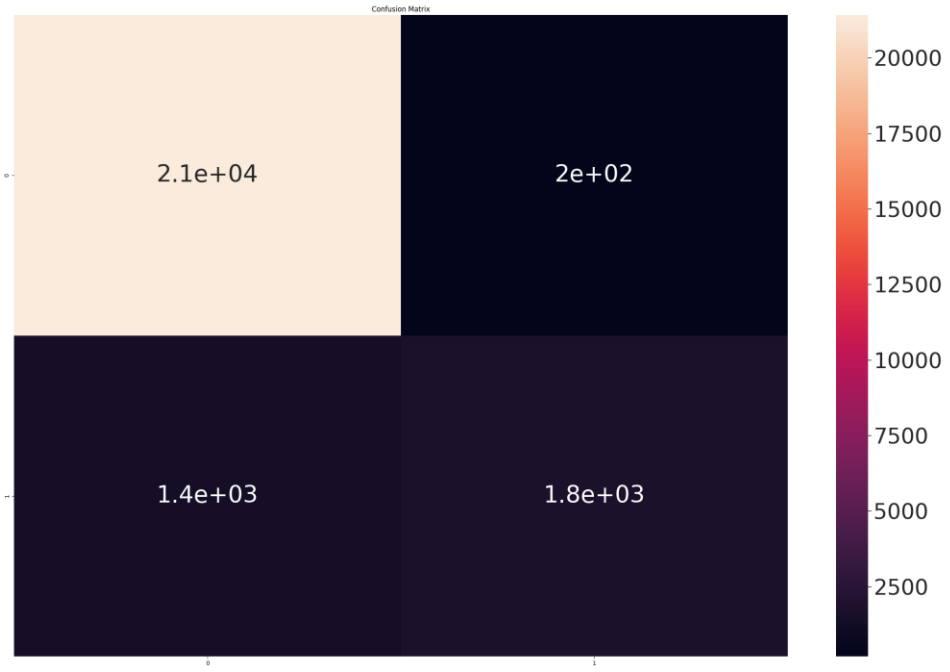
By plotting the various classes under attack_cat – we can see a severe lean towards no attack of 87% and the remaining 12% distributed over the 13 odd classes under attack cat

Feature selection

Logistic regression without feature selection

Examining the raw data without any transformation- in terms of logistic regression and its accuracy

Accuracy: 0.9372625111165009
Confusion Matrix:
[[21424 195]
[1357 1762]]



Classification Report:

	precision	recall	f1-score	support
0	0.94	0.99	0.97	21619
1	0.90	0.56	0.69	3119
accuracy			0.94	24738
macro avg	0.92	0.78	0.83	24738
weighted avg	0.94	0.94	0.93	24738

<u>Without Feature Selection, the accuracy (Micro-Avg) is 94%.</u>

The 3 feature selection methods I tried:

- chi2 feature selection
- mutual_info_classif feature selection
- f_classif feature selection

chi2 feature selection

This is done to identify independent variables.

I performed a chi-square test on the x_train data frame

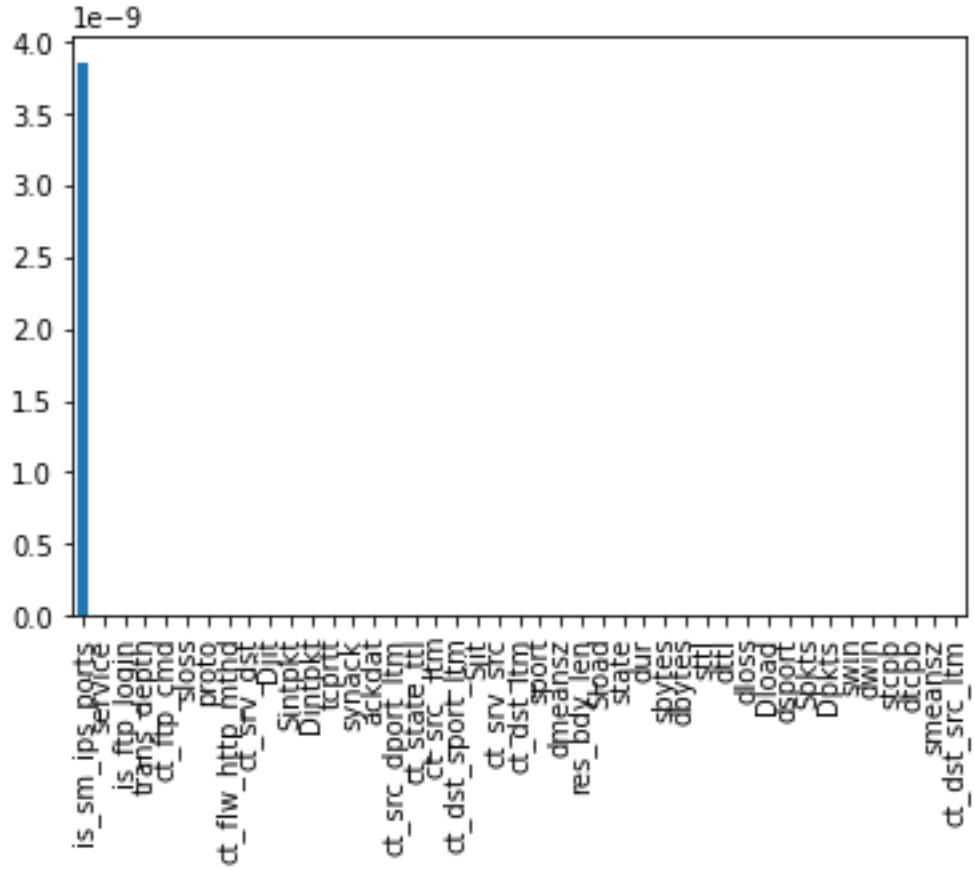
```
chi_scores = chi2(X_train, y_train)
```

```
chi_scores
```

```
p_values = pd.Series(chi_scores[1],index = X.columns)
p_values.sort_values(ascending = False , inplace = True)
```

```
p_values.plot.bar()
```

And found one independent variable come up in the results –



Logistic regression with Chi2

Accuracy: 0.9372625111165009
Confusion Matrix:
[[21424 195]
[1357 1762]]

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.99	0.97	21619
1	0.90	0.56	0.69	3119
accuracy			0.94	24738
macro avg	0.92	0.78	0.83	24738
weighted avg	0.94	0.94	0.93	24738

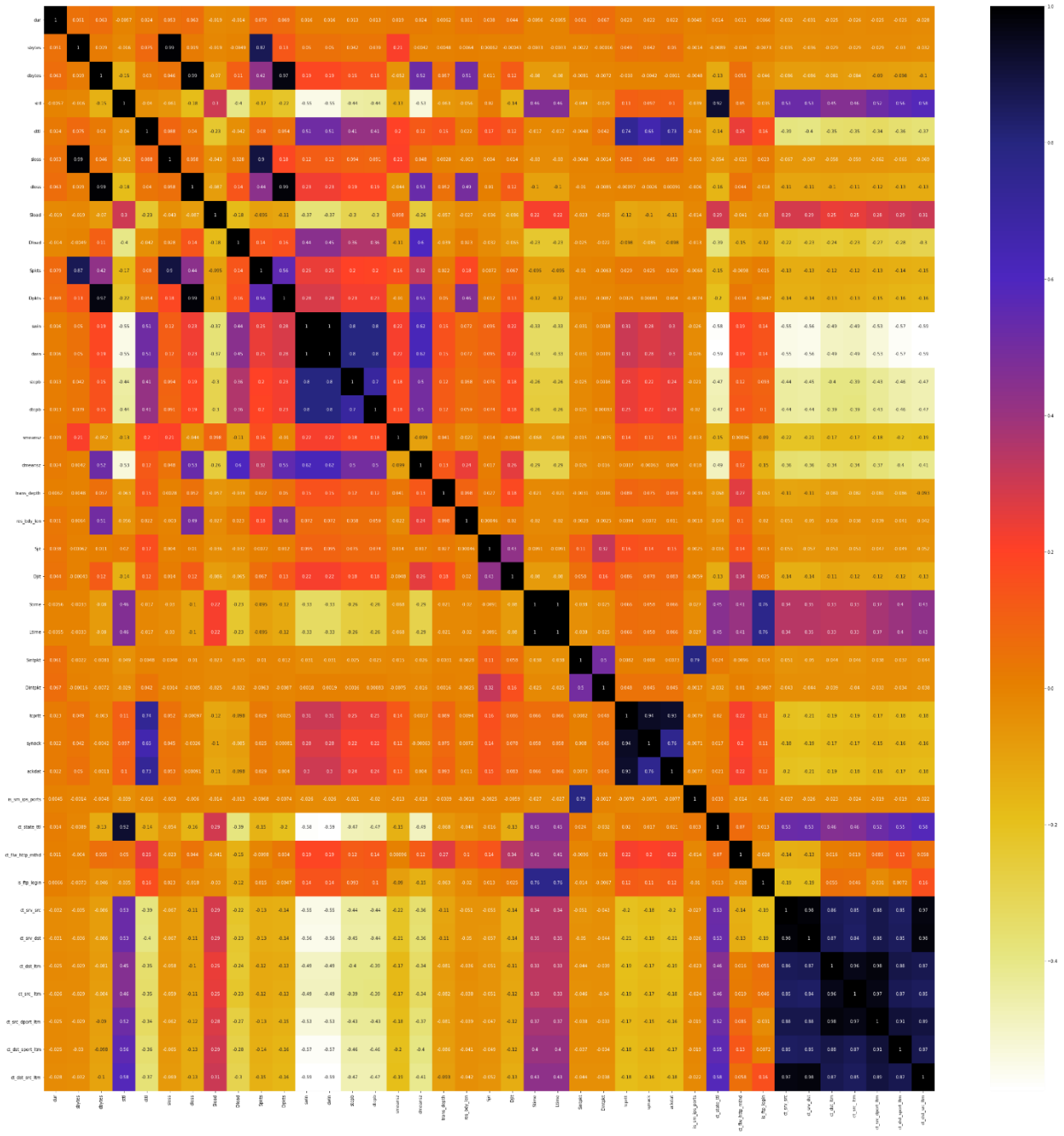
With Chi Square Test with Logistic Regression, The Accuracy is 94%, same as the without selection one. However, here the number of features was reduced to 25 from 45.

Logistic Regression with Mutual info selection method

Feature selection using pearson correlation

I used the **dataframe.corr()** to find the correlation between various columns. By default uses the pearson method – standard correlation coefficient. I then tried to plot it using seaborn to get a visual representation of the results. Because of 47 columns – the result was hard to identify and difficult to read.

See image below



I then applied a function borrowed from the sklearn library pages– to list the number of columns that fall below a certain threshold –

For 0.8 – I had 20 columns as highly correlated and for 0.7 I had 5 columns as highly correlated

For 0.9 – 13 columns shows as highly correlated

Select K Best

On further research I then used selectKbest – available In the scikit library- providing an implementation of the mutual information for feature selection with numeric input and categoric output variables.

I limited my k to 25.

I also chose to try out another way while performing feature selection – as understood from this blog - https://towardsdatascience.com/what-and-why-behind-fit-transform-vs-transform-in-scikit-learn-78f915cf96fe

As I understand it – fit_transform is used to scale the training data while transform() is used on the test data to to use mean and variances to transform the test data.

Accuracy: 0.9295415959252971

Confusion Matrix:
[[21461 158]
[1585 1534]]

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.99	0.96	21619
1	0.91	0.49	0.64	3119
accuracy			0.93	24738
macro avg	0.92	0.74	0.80	24738
weighted avg	0.93	0.93	0.92	24738

The accuracy decreased to 93%.

ANOVA f Test Feature Selection Method

The f classif() function in the scikit-learn machine library implements the ANOVA f-test. This function could be used in conjunction with the SelectKBest class to select the top k most relevant attributes (biggest values).

I used ANOVA (Analysis of Variance) to help identify the optimal features for this experiment.

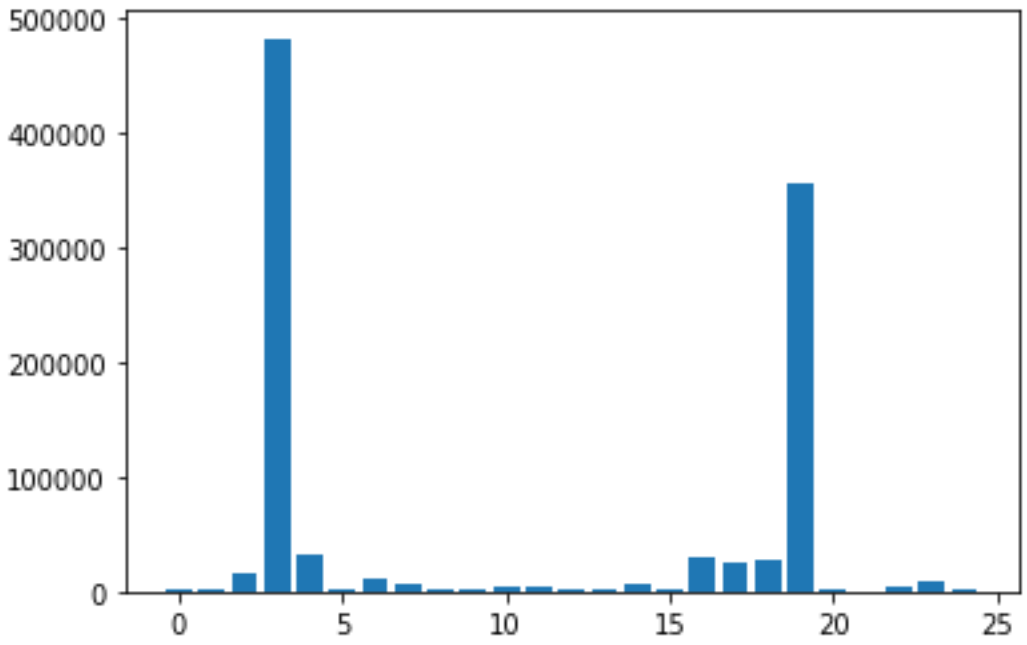
To my understanding - In a variable, variance is a measure of how far apart the numbers are. Each value in a variable is compared to the mean to see how distant it is from the average.

How much a characteristic influences the response variable depends on its variance. If the variation is low, then this feature has no effect on the response, and the opposite is true.

When two or more groups have significantly different mean values, a statistical approach called Analysis of Variance (ANOVA) can be used to compare them.

I conducted a quick study on which features showed the highest scores:

```
Feature 0: 1295.330621
Feature 1: 1014.324101
Feature 2: 16091.844290
Feature 3: 482678.138481
Feature 4: 32892.714878
Feature 5: 714.216130
Feature 6: 10799.874169
Feature 7: 5705.186532
Feature 8: 834.423135
Feature 9: 1044.669904
Feature 10: 4295.932406
Feature 11: 4266.484119
Feature 12: 1943.235464
Feature 13: 2011.015424
Feature 14: 6976.059043
Feature 15: 854.378580
Feature 16: 29432.989991
Feature 17: 24516.386778
Feature 18: 28066.613430
Feature 19: 356503.788143
Feature 20: 1081.372412
Feature 21: 667.661550
Feature 22: 5277.282197
Feature 23: 7924.164967
Feature 24: 1624.090131
```



This showed that perhaps feature 3, 19 are the most relevant because of their high scores.

Accuracy: 0.9332201471420487
Confusion Matrix:
[[21334 285]
[1367 1752]]

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.99	0.96	21619
1	0.86	0.56	0.68	3119
accuracy			0.93	24738
macro avg	0.90	0.77	0.82	24738
weighted avg	0.93	0.93	0.93	24738

The ANOVA selection method also resulted in 93% accuracy.

Final Selection Method based on Accuracy

Best Feature Selector: chi2

Best Feature Set:

- sport
- dsport
- sbytes
- dbytes
- sttl
- dttl
- dloss
- Sload
- Dload
- Spkts
- Dpkts
- swin
- dwin
- stcpb
- dtcpb
- smeansz
- dmeansz
- res_bdy_len
- Sjit
- Djit
- Sintpkt
- Dintpkt
- ct_state_ttl
- ct_src_dport_ltm
- ct_dst_sport_ltm

Label Classification

Keeping only the selected columns as X and “Label” column as y, I did a classification analysis with the following classifiers.

- Logistic regression
- Random Forest
- Decision Tree

I also chose the best feature analysis in the code – and then applied the classification

The results and analysis are shown below:

Logistic Regression Binary Classification

On running the actual model these are the summary statistics:

Accuracy: 0.9372625111165009
Confusion Matrix:
[[21424 195]
[1357 1762]]

Classification Report:Binary_Logistic_Regression_with_best_feature_selection

	precision	recall	f1-score	support
0	0.94	0.99	0.97	21619
1	0.90	0.56	0.69	3119
accuracy			0.94	24738
macro avg	0.92	0.78	0.83	24738
weighted avg	0.94	0.94	0.93	24738

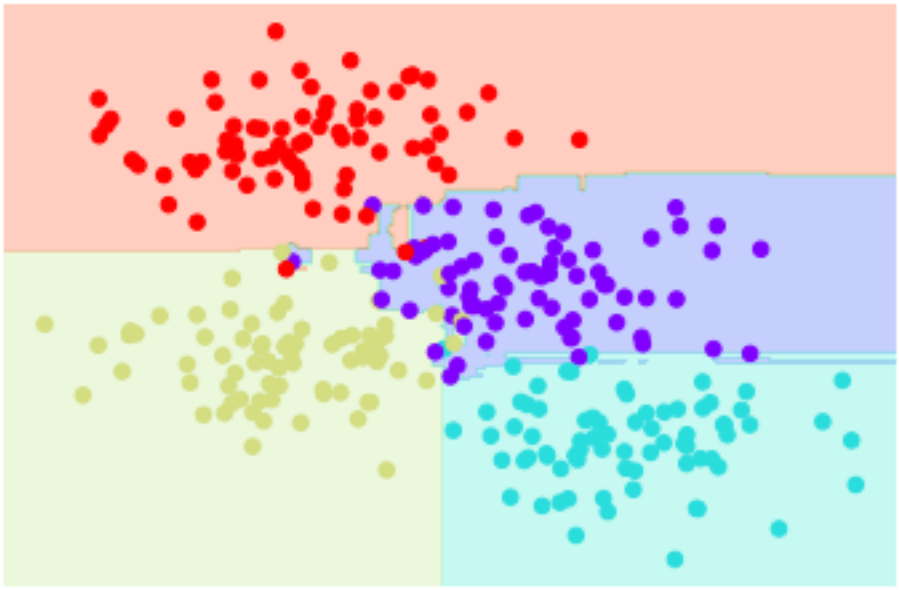
Random Forest Binary Classification

This was hard to visualize – There were many blogs online suggesting creating box whisker plots for running a complex series of runs taking the means for each run and plotting the accuracy with the number of features considered. But I felt it was beyond the effort and time I could give to this assignment. I was considering implementing from here - https://machinelearningmastery.com/random-forest-ensemble-in-python/

I did find a blog which went to great lengths to visualize random forests and decision trees - https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html

And I used one of their functions to initially visualize my model. It's a basic scatter plot – that tries to fit a number of decision trees. I ran the sample code which used the n_estimator = 100 i.e number of trees = 100 – as well as the criterion as entropy .

To my understanding – This exercise was to understand parameters space and various models and their prediction.



On running the actual model these are the summary statistics:

Accuracy: 0.9967256851806937
Confusion Matrix:
[[21549 70]
[11 3108]]

Classification Report:Binary_Random_Forest_with_best_feature_selection

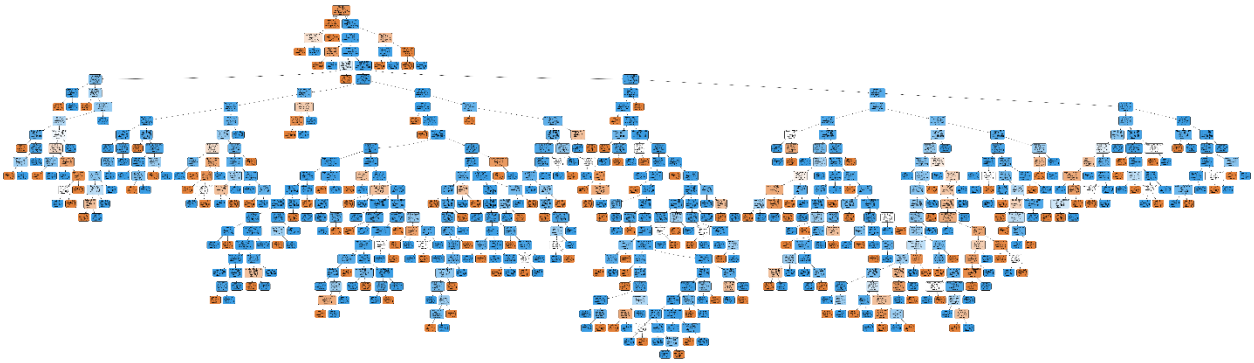
	precision	recall	f1-score	support
0	1.00	1.00	1.00	21619
1	0.98	1.00	0.99	3119
accuracy			1.00	24738
macro avg	0.99	1.00	0.99	24738

weighted avg 1.00 1.00 1.00 24738

Decision Tree Binary Classification

On applying decision tree classification – I visualized the results using graphviz :

The tree was quite a lot to read, but it helped me understand how my feature selection had helped in really narrowing down on selecting features that mattered – reducing redundancy



On running the actual model these are the summary statistics:

Accuracy: 0.9958363651063141
Confusion Matrix:
[[21564 55]
[48 3071]]

Classification Report:Binary_Decision_Tree_with_best_feature_selection

	precision	recall	f1-score	support
0	1.00	1.00	1.00	21619
1	0.98	0.98	0.98	3119
accuracy			1.00	24738
macro avg	0.99	0.99	0.99	24738
weighted avg	1.00	1.00	1.00	2473

BEST MODEL BASED ON ACCURACY

Both Decision tree and random Forest has 100% accuracy on the binary classification problem. Logistic regression had an accuracy of 93%.

Attack_cat MultiClass Classification

Problems faced and remedy

Data Imbalance

Since the data set is highly skewed toward the “None” class, I thought doing an UnderSampling to balance the data. Otherwise, the accuracy would be very high on few classes, and absolutely abysmal on the other classes. The dataset is preprocessed first, such that after under sampling, the class with the highest class is reduced in number to just double the number of rows in the class with the minimum number of rows.

To my understanding - A dataset is considered imbalanced if the dominant class has less than one example for every hundred or thousand cases of the minority class.

Machine learning algorithms can be influenced by this bias as in training dataset, leading some to completely overlook the minority class. Due to the fact that predictions are often most relevant for those in the minority, this is an issue

Class imbalance can be alleviated by resampling the training dataset at random. Oversampling and under sampling are two of the most common methods for resampling an imbalanced dataset, and they both have advantages and disadvantages.

It's possible to rebalance the class distribution of an imbalanced dataset by using random resampling. Oversampling the training dataset by using random oversampling might lead to overfitting for the some models. As an example, when the majority class is under sampled, a model may lose important data.

Random under sampling is the process of removing randomly selected examples from the training data from the majority class. As a result, the training dataset's majority class has fewer samples in the modified version. It is possible to repeat this process until the required class distribution is reached, such as an equivalent number of samples for each class.

In data when there has been a class imbalance, this technique may be more appropriate, but a sufficient number of cases in the minority class can match such a helpful model. Samples from majority class may be informative, significant, or even essential to fitting a solid decision boundary, but under sampling has this drawback: they are omitted from the sample. There is no method to identify or preserve "excellent" instances from majority class because examples are removed at random.

Logistic Regression MultiClass Classification

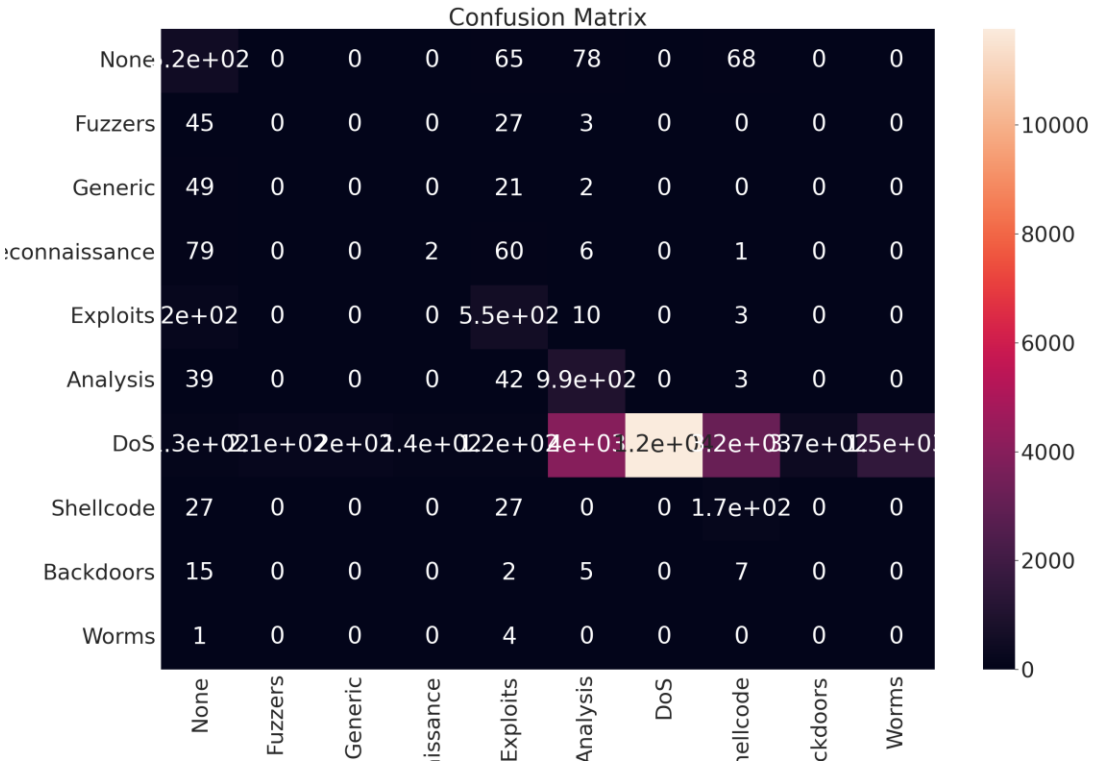
Performed using the LogisticRegression classifier available in sklearn.

Feature selection was performed on selecting the best features discovered in the previous phase – Chi2 method is used.

Accuracy: 0.5345217883418223

Confusion Matrix:

```
[[ 520    0    0    0   64   79    0   63    0    0]
 [  45    0    0    0   27    3    0    0    0    0]
 [  49    0    0    0   21    2    0    0    0    0]
 [  79    0    0    2   60    6    0    1    0    0]
 [ 200    0    0    0  553    9    0    3    0    0]
 [  39    0    0    0   42  990    0    3    0    0]
 [ 180  246   22  126 4224 4542 10989  761   42  487]
 [  30    0    0    0   26    0    0  169    0    0]
 [  16    0    0    0    2    5    0    6    0    0]
 [   1    0    0    0    4    0    0    0    0    0]]
```



Classification Report:Multi_Class_Logistic_Regression_with_best_feature_selection

	precision	recall	f1-score	support

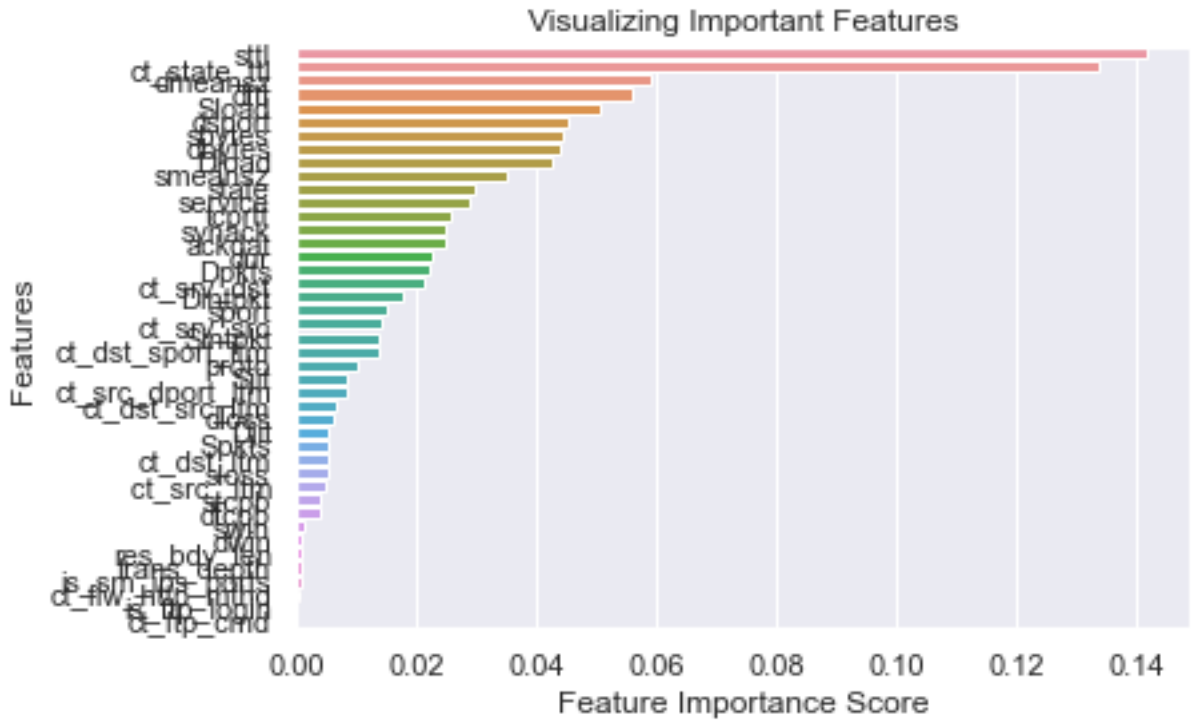
0	0.33	0.71	0.45	726
1	0.00	0.00	0.00	75
2	0.00	0.00	0.00	72
3	0.01	0.01	0.01	148
4	0.10	0.72	0.18	765
5	0.18	0.92	0.30	1074
6	1.00	0.50	0.67	21619
7	0.23	0.75	0.36	225
8	0.00	0.00	0.00	29
9	0.00	0.00	0.00	5
accuracy			0.86	24738
macro avg	0.49	0.46	0.46	24738
weighted avg	0.90	0.53	0.62	24738

Random Forest MultiClass Classification

Implementing this classification using the RandomForestClassifier found in the sklearn library – it fits a number of decision trees on sub samples of datasets.

I looked at the feature importance without any feature selection – by accessing the feature_importance attribute available on the classifier.

On plotting the features we see that column – sttl as well as ct_state_ttl show up as highly important.



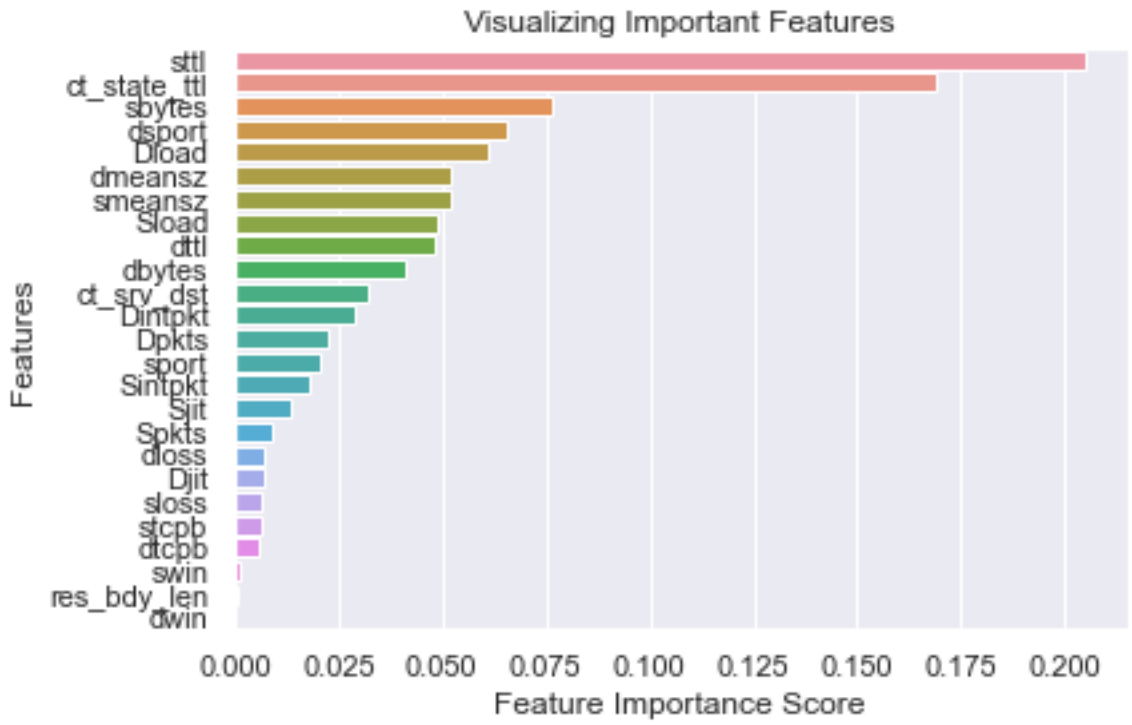
The entire list is here - sttl 1.418728e-01

ct_state_ttl	1.337243e-01
dmeansz	5.888537e-02
dttl	5.570663e-02
Sload	5.057821e-02
dsport	4.530443e-02
sbytes	4.430053e-02
dbytes	4.394782e-02
Dload	4.248876e-02
smeansz	3.507637e-02
state	2.975267e-02
service	2.857227e-02
tcprtt	2.540220e-02
synack	2.460967e-02
ackdat	2.452355e-02
dur	2.259714e-02
Dpkts	2.222862e-02
ct_srv_dst	2.136237e-02
Dintpkt	1.741254e-02
sport	1.492954e-02
ct_srv_src	1.396237e-02
Sintpkt	1.361964e-02
ct_dst_sport_ltm	1.354834e-02
proto	1.001299e-02
Sjit	8.072558e-03
ct_src_dport_ltm	8.042969e-03
ct_dst_src_ltm	6.446380e-03
dloss	5.937639e-03
Djit	5.321010e-03

```
Spkts                5.209612e-03
ct_dst_ltm           5.111522e-03
sloss                5.094018e-03
ct_src_ ltm          4.897569e-03
stcpb                3.819857e-03
dtcpb                3.783662e-03
swin                 9.798096e-04
dwin                 7.633315e-04
res_bdy_len          6.344228e-04
trans_depth          5.325763e-04
is_sm_ips_ports      4.680215e-04
ct_flw_http_mthd     4.625760e-04
is_ftp_login         2.319892e-06
ct_ftp_cmd           9.452699e-07
```

Feature selection was performed on selecting the best features discovered in the previous phase – Chi2 method is used.

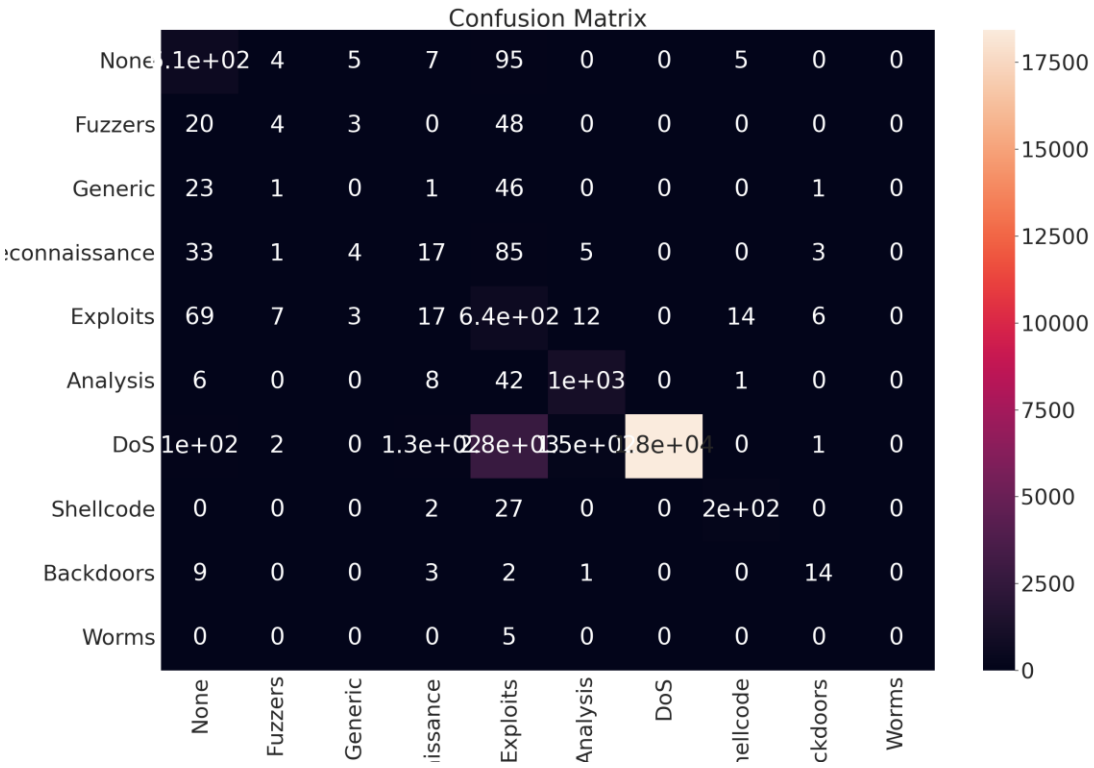
and plotting the importance features after feature selection and modelling showed us –



Showing us not much change .

Accuracy: 0.8859649122807017
Confusion Matrix:
[[606 3 5 10 97 0 0 5 0 0]

```
[  20    5    3    0   47    0    0    0    0    0]
[  23    1    0    1   46    0    0    0    1    0]
[  34    1    4   17   83    6    0    0    3    0]
[  70    7    3   14  639   13    0   15    4    0]
[   6    0    0    8   41 1018    0    1    0    0]
[ 103    3    0   76 2001   16 19418    1    1    0]
[   0    0    0    2   24    0    0  199    0    0]
[   8    0    0    3    2    1    0    0   15    0]
[   0    0    0    0    5    0    0    0    0    0]]
```



F1-Score:

Class 0: 75.93984962406014
Class 1: 10.526315789473685
Class 2: 0.0
Class 3: 12.186379928315413
Class 4: 34.08
Class 5: 95.67669172932331
Class 6: 94.63654750590929
Class 7: 89.237668161435
Class 8: 56.60377358490567
Class 9: 0.0

Precision-Recall Curve:

Class 0: Precision: 69.6551724137931 Recall: 83.47107438016529 F1-Score: 75.93984962406014
Class 1: Precision: 25.0 Recall: 6.666666666666667 F1-Score: 10.526315789473685
Class 2: Precision: 0.0 Recall: 0.0 F1-Score: 0.0

Class 3: Precision: 12.977099236641221 Recall: 11.486486486486488 F1-Score: 12.186379928315413
Class 4: Precision: 21.407035175879397 Recall: 83.52941176470588 F1-Score: 34.08
Class 5: Precision: 96.58444022770398 Recall: 94.78584729981378 F1-Score: 95.67669172932331
Class 6: Precision: 100.0 Recall: 89.8191405707942 F1-Score: 94.63654750590929
Class 7: Precision: 90.04524886877829 Recall: 88.44444444444444 F1-Score: 89.237668161435
Class 8: Precision: 62.5 Recall: 51.724137931034484 F1-Score: 56.60377358490567
Class 9: Precision: 0.0 Recall: 0.0 F1-Score: 0.0

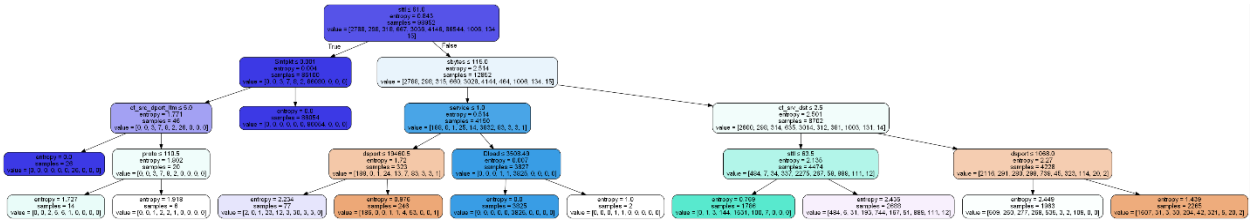
Classification Report:Multi_Class_Random_Forest_with_best_feature_selection

	precision	recall	f1-score	support
0	0.70	0.83	0.76	726
1	0.25	0.07	0.11	75
2	0.00	0.00	0.00	72
3	0.13	0.11	0.12	148
4	0.21	0.84	0.34	765
5	0.97	0.95	0.96	1074
6	1.00	0.90	0.95	21619
7	0.90	0.88	0.89	225
8	0.62	0.52	0.57	29
9	0.00	0.00	0.00	5
accuracy			0.89	24738
macro avg	0.48	0.51	0.47	24738
weighted avg	0.95	0.89	0.91	24738

Decision Tree MultiClass Classification

On running this without any feature selection – I tried to visualize the tree – and clearly that was too big. The notebook said it had to scale the image twice to be able to render it.

On trimming the decision tree at max depth = 3 we get :

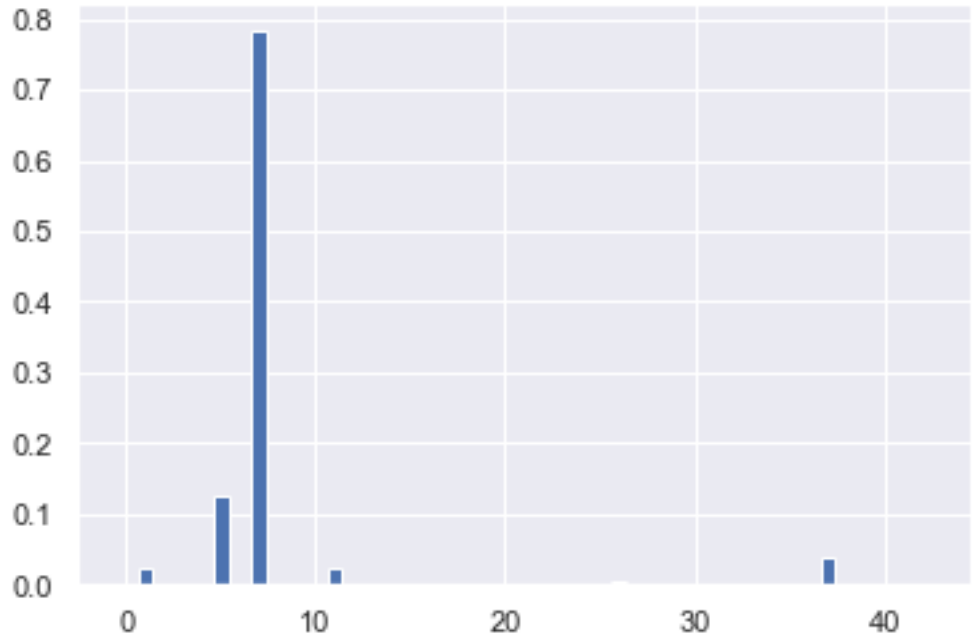


I did this so I can comprehend what the entropy values are at the beginning – how many samples were considered as well as what features were at play.

On printing the feature importance I was able to see :

```
('sport', 0.0)
('dsport', 0.02500101151480843)
('proto', 5.061238336326172e-06)
('state', 0.0)
('dur', 0.0)
('sbytes', 0.1256316262279313)
('dbytes', 0.0)
('sttl', 0.7827277317760183)
('dttl', 0.0)
('sloss', 0.0)
('dloss', 0.0)
('service', 0.023182459620194194)
('Sload', 0.0)
('Dload', 0.00036874962215776024)
('Spkts', 0.0)
('Dpkts', 0.0)
('swin', 0.0)
('dwin', 0.0)
('stcpb', 0.0)
('dtcpb', 0.0)
('smeansz', 0.0)
('dmeansz', 0.0)
('trans_depth', 0.0)
('res_bdy_len', 0.0)
('Sjit', 0.0)
('Djit', 0.0)
('Sintpkt', 0.0033584010013555255)
('Dintpkt', 0.0)
('tcprtt', 0.0)
('synack', 0.0)
('ackdat', 0.0)
('is_sm_ips_ports', 0.0)
('ct_state_ttl', 0.0)
('ct_flw_http_mthd', 0.0)
('is_ftp_login', 0.0)
('ct_ftp_cmd', 0.0)
('ct_srv_src', 0.0)
('ct_srv_dst', 0.039046358549922125)
('ct_dst_ltm', 0.0)
('ct_src_ ltm', 0.0)
('ct_src_dport_ltm', 0.0006786004492758795)
('ct_dst_sport_ltm', 0.0)
('ct_dst_src_ltm', 0.0)
```

Proto column seems to have the highest importance In this algorithm. Proto = transactional protocol telling me the type of protocol used is considered highly relevant.



Inspite of doing feature selection on the best selected features in the label process – the decision tree printed was far too large for any comprehension

I was also able to print out feature importance after feature selection

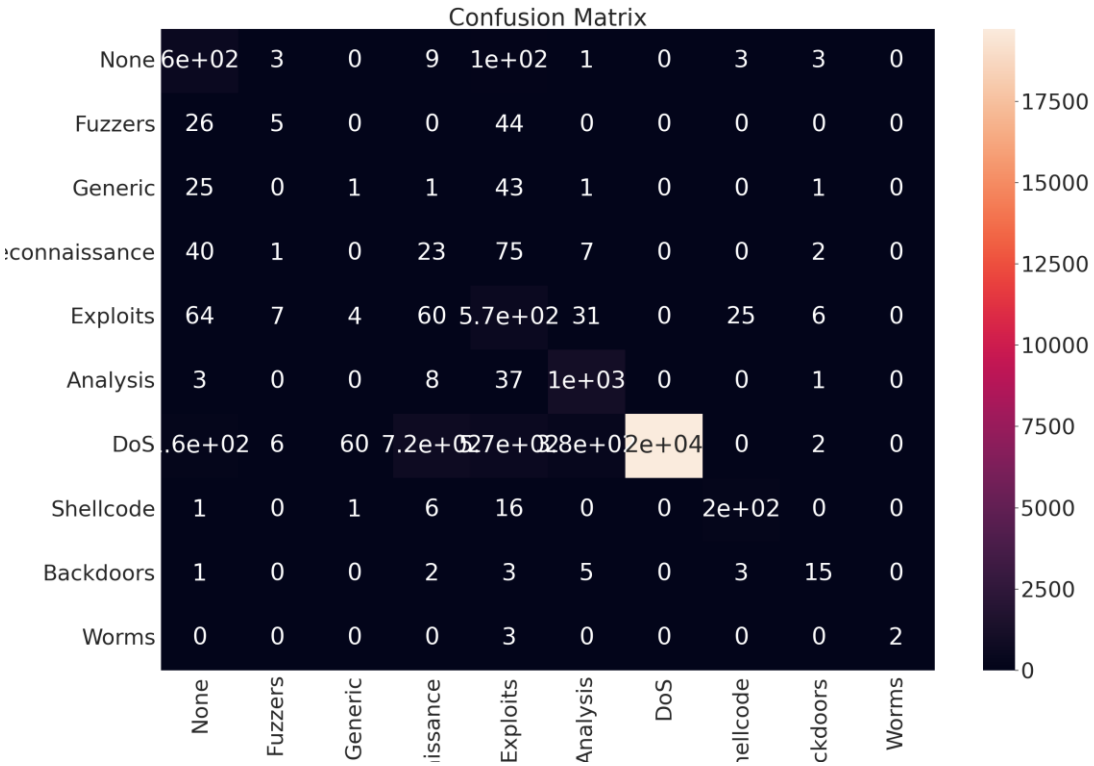
[('sport', 0.015898281446408535), ('dsport', 0.2866864195476979), ('proto', 0.3909398170597857), ('state', 0.03095922312715115), ('dur', 0.11488840899070608), ('sbytes', 0.00024954940177421156), ('dbytes', 0.0037470430277677436), ('sttl', 0.01581182676232269), ('dttl', 0.01108346023454924), ('sloss', 0.007315045179124909), ('dloss', 0.0009694274453689575), ('service', 0.00037432410266131737), ('Sload', 0.0), ('Dload', 0.010674958951820792), ('Spkts', 0.011827607121574006), ('Dpkts', 0.027581123595836584), ('swin', 0.011599420139162412), ('dwin', 0.002175469517227921), ('stcpb', 0.006860314753372154), ('dtcpb', 0.01113235576893027), ('smeansz', 0.01595410654945366), ('dmeansz', 0.011197768700588159), ('trans_depth', 0.0011379806934798506), ('res_bdy_len', 0.010297584568456555), ('Sjit', 0.0006384833147790001)]

To me – this indicates – proto and ds port having the most significance during a decision tree modelling.

Accuracy: 0.83446519524618

Confusion Matrix:

```
[[ 606    3    0   11   99    1    0    3    3    0]
 [  26    5    0    0   44    0    0    0    0    0]
 [  25    0    1    2   43    0    0    0    1    0]
 [  40    1    0   20   75   11    0    0    1    0]
 [  65    7    4   59  573   30    0   22    5    0]
 [   3    0    0    9   34 1026    1    0    1    0]
 [ 155    6   20  373 1722 1149 18192    0    2    0]
 [   0    0    0    5   16    0    0  204    0    0]
 [   1    0    0    3    3    5    0    3   14    0]
 [   0    0    0    0    3    0    0    0    0    2]]
```



F1-Score:

Class 0: 73.58834244080144
Class 1: 10.309278350515463
Class 2: 2.0618556701030926
Class 3: 6.349206349206349
Class 4: 33.93544566183003
Class 5: 62.25728155339806
Class 6: 91.38953079473525
Class 7: 89.27789934354486
Class 8: 50.0
Class 9: 57.14285714285715

Precision-Recall Curve:

Class 0: Precision: 65.79804560260585 Recall: 83.47107438016529 F1-Score: 73.58834244080144

Class 1: Precision: 22.727272727272727 Recall: 6.666666666666667 F1-Score: 10.309278350515463
Class 2: Precision: 4.0 Recall: 1.3888888888888888 F1-Score: 2.0618556701030926
Class 3: Precision: 4.149377593360995 Recall: 13.513513513513514 F1-Score: 6.349206349206349
Class 4: Precision: 21.937212863705973 Recall: 74.90196078431373 F1-Score: 33.93544566183003
Class 5: Precision: 46.17461746174617 Recall: 95.53072625698324 F1-Score: 62.25728155339806
Class 6: Precision: 99.99450338042104 Recall: 84.14820296961007 F1-Score: 91.38953079473525
Class 7: Precision: 87.93103448275862 Recall: 90.66666666666666 F1-Score: 89.27789934354486
Class 8: Precision: 51.85185185185185 Recall: 48.275862068965516 F1-Score: 50.0
Class 9: Precision: 100.0 Recall: 40.0 F1-Score: 57.14285714285715

Classification Report:Multi_Class_Decision_Tree_with_best_feature_selection

	precision	recall	f1-score	support
0	0.66	0.83	0.74	726
1	0.23	0.07	0.10	75
2	0.04	0.01	0.02	72
3	0.04	0.14	0.06	148
4	0.22	0.75	0.34	765
5	0.46	0.96	0.62	1074
6	1.00	0.84	0.91	21619
7	0.88	0.91	0.89	225
8	0.52	0.48	0.50	29
9	1.00	0.40	0.57	5
accuracy			0.83	24738
macro avg	0.50	0.54	0.48	24738
weighted avg	0.93	0.83	0.87	24738