Data Exploration

First, we start by loading our train data in df_train and then we load our test data in df_test and then we merge all data (df_train first and then df_test) and make df_all. We make df_all because if we want to change any feature, we should do it on both df_train and df_test.

After loading data it's time to see how is our data and what values each feature has.

We see that there are:

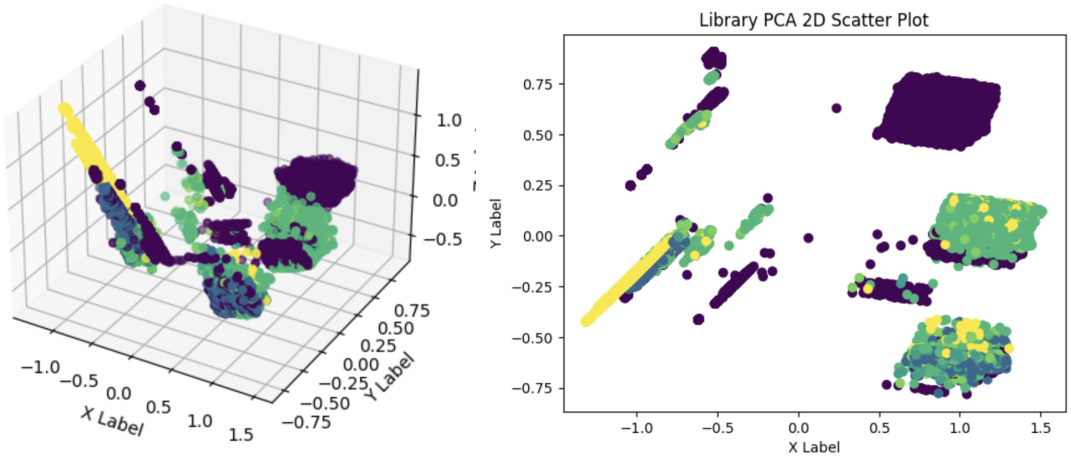
- 3 features that have categorical values
- 11 features that have decimal values
- 29 features that have integer values
- and there is a label feature which has a categorical value

	id	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	\
0	1	0.121478	tcp	-	FIN	6	4	258	172	
1	2	0.649902	tcp	-	FIN	14	38	734	42014	
2	3	1.623129	tcp	-	FIN	8	16	364	13186	
3	4	1.681642	tcp	ftp	FIN	12	12	628	770	
4	5	0.449454	tcp	-	FIN	10	6	534	268	
...	...	...	...	...	...	...	...	...	...	
175336	175337	0.000009	udp	dns	INT	2	0	114	0	
175337	175338	0.505762	tcp	-	FIN	10	8	620	354	
175338	175339	0.000009	udp	dns	INT	2	0	114	0	
175339	175340	0.000009	udp	dns	INT	2	0	114	0	
175340	175341	0.000009	udp	dns	INT	2	0	114	0	

	rate	...	ct_src_dport_ltm	ct_dst_sport_ltm	\
0	74.087490	...	1	1	
1	78.473372	...	1	1	
2	14.170161	...	1	1	
3	13.677108	...	1	1	
4	33.373826	...	2	1	

(Complete information in main.ipynb)

After that to get to know more about our data, we try to plot it, and to plot a high dimensionan data we should decrease it's dimantionality (with pca that we talk about later) and also factorize it's categorical features (which we speak much more completely further). After doing the requirements we have that:



And as it is obvious only two or three features are separated and others are very involved to each other.

Preprocessing

After we get to know our data a little bit, it's time to start our preprocessing.

First, we see how our categories are distributed between df_train and df_test and unfortunately, we see that there are categories that are only in df_test or in df_train. This can hurt our models very much but because we should assume we don't have the test dataset for our modeling, we ignore it and we factorize our categorical features into nominal ones.

```
proto
DIFF TRAIN WITH TEST:
[0.03030303 0.04545455]
DIFF TEST WITH TRAIN:
[]
######
service
DIFF TRAIN WITH TEST:
[]
DIFF TEST WITH TRAIN:
[]
######
state
DIFF TRAIN WITH TEST:
[0.3 0.6 0.7 0.8]
DIFF TEST WITH TRAIN:
[0.9 1. ]
######
attack_cat
DIFF TRAIN WITH TEST:
[]
DIFF TEST WITH TRAIN:
[]
######
```

After factorizing our categorical features we check if there are any missing values and fortunately, there aren't any.

After checking for missing values it's time to check our features intervals. We see that they can vary a lot and if we don't scale them, some of our features will have more impact than others but we don't have any assumptions about our features so we should make them matter alike and we do a min-max scale on each of our features and make them to vary in interval of [0, 1].

```
dttl Max: 254 Min: 0
sload Max: 5988000256.0 Min: 0.
dload Max: 22422730.0 Min: 0.0
sloss Max: 5319 Min: 0
dloss Max: 5507 Min: 0
sinpkt Max: 84371.496 Min: 0.0
dinpkt Max: 57739.24 Min: 0.0
sjit Max: 1483830.917 Min: 0.0
djit Max: 463199.2401 Min: 0.0
swin Max: 255 Min: 0
stcpb Max: 4294958913 Min: 0
dtcpb Max: 4294881924 Min: 0
dwin Max: 255 Min: 0
tcprtt Max: 3.821465 Min: 0.0
synack Max: 3.226788 Min: 0.0
ackdat Max: 2.928778 Min: 0.0
```

(Complete information in main.ipynb)

After that, we see that one of our features is id, and id is a unique value for each of the samples and it can't help our model at all, so we drop that feature from our dataset.

After that, we try to see how imbalanced is our data, and we see that it is totally imbalance. Some of our labels are barely 1% of all of our data and if we keep them they will just confuse our models and make their predictions weaker, so we decide to cut them off the dataset.

```
#lable 0  : 93000
#lable 1  : 2329
#lable 2  : 2677
#lable 3  : 24246
#lable 4  : 1511
#lable 5  : 13987
#lable 6  : 44525
#lable 7  : 16353
#lable 8  : 174
#lable 9  : 58871
```

Now that we are done with our preprocessing, it's time to divide our data into df_train and df_test again. And then divide df_train into X_train and y_train and divide df_test into X_test and y_test.

Feature Processing

Now that we have baked our data, we start to make some feature extraction/reduction algorithms and make some datasets for our models to learn on them and we decide which one of them is better for which of our models.

Our first feature extraction/reduction method is to use autoencoders. We build an autoencoder with 2 encoding and 2 decoding layers (first encoding layer decrease number of our features from 42 to 30 and our second encoding layer decreases it from 30 to 21 and decoding layers convert them back to 42). And for each nodes activation function we set Relu function except for the last layer of our decoder, we choose sigmoid function because our variables are in interval of [0, 1]. And for our loss function and optimizer we have that:

- **Optimizer = Adam**: It combines the benefits of adaptive learning rates and momentum-based updates. Adam algorithm computes individual adaptive learning rates for each of our parameters and it allows efficient convergence and handles sparse gradients.
- **Loss function = Binary cross-entropy**: It measures the dissimilarity between our predicted outputs and the true binary targets, and it penalizes large discrepancies. And by maximizing the similarity between our reconstructed outputs and the original inputs, it encourages our autoencoder to learn meaningful binary representations.

With this configuration, we learn our autoencoder for 100 epochs and we end up with val_loss = 0.097 and we encode our train and test data for our models to use them in future.

Our second feature extraction/reduction method is to use PCA (Principal Component Analysis). We choose 2 feature sizes of 20 and 30. After that we start with fitting our pca on our data and we extract our 20d train and test data from it. We do the exact same again and extract our 30d train and test data too and we keep them for our models to use them in future.

Our third feature extraction/reduction method is to use supervised feature selection algorithms. For this one first we use RFE (Recursive Feature Elimination) method with RandomForestClassifier and we select 10 most important features from it. After that with considering our data we have that two labels "Normal" and "Generic" have the most samples and may affect our feature selection, so we remove those samples from our dataset and we find another 10 most important features with a random forest feature importance model. After that we combine them and we get 15 features that are expected to divide our labels good. After that we pick those 15 features from our dataset and keep them for our models to use them in future.

Now with 5 sets of dataset, we start to learn different models and find best parameters from each of them to get the best outcome from them.

Model Selection and Training and Evaluation:

To find best parameters and best dataset for each model, we use 5-fold cross validation and we get accuracy and f1-score and recall and precision of each fold of each set of parameters and we pick best one of them from average of their 5 f1-score from each fold of it.

First we start to learn best parameters for SVM with 5-fold classification and our learning parameter is:

A. Kernel type:
    a. Linear: Forms linear decision boundaries in the input space.
    b. Poly: Allows for curved decision boundaries using polynomial functions and we choose it's degree for more specification.
    c. Rbf: Creates non-linear decision boundaries using Gaussian-like functions.
    d. Sigmoid: Maps the input space into a hyperbolic tangent function for non-linear classification.

After learning we find that best parameters are:

```
best_Kernel : poly | best_Degree : 4 | best_data : Encoded | best_score : 0.8077070265248129
best_Kernel_f1 : poly | best_Degree_f1 : 4 | best_data_f1 : Encoded | best_score_f1 : 0.6619730270908294
```

We pick our best parameters based on f1-score and accuracy but we only build the model with best average f1-score. Here it shows that our encoded data (with autoencoder) and polynomial kernel with degree of 4 have had the best f1-score which shows that reducing our features can help us in classifying with SVM and that our labels may have a polynomial relation with each other.

After that we learn our model on the all of the train data (because of computation limits we have found our best parameters only on 20% of each dataset but we have kept distribution between our labels).

```
accuracy 0.7076255424674519
accuracy_train 0.7893819129699645
precision 0.6396203971627599
precision_train 0.7457143453012275
recall 0.7281625365843877
recall_train 0.7880471196701274
f1 0.6429117674094137
f1_train 0.7379195740057597
confusion [[22507 12232  1062  1190     9     0]
 [   41  3994   808    89  1130     0]
 [    2   123  2974    72   324     1]
 [   34   739   747  6603  3009     0]
 [    8   220   215   797  2848     1]
 [    4   188    53   416    66 18144]]
confision_train [[45215  9823   407   546     9     0]
 [   26 15659   842   190  1467     0]
 [    8   299  8306    66  1812     0]
 [  111  1864  1922 16359 13137     0]
 [   19   432   297  1697  9819     0]
 [    7   157    37   373   327 39099]]
```

Here we can see that our accuracy is not very bad but because of the data's imbalancement our f1-score is low. And from confusion matrices we can see that forth label (*Exploits*) is not classifying good but first and last label (*Normal and Generic*) are classifying very good because of their repetition.

Second we try to learn best parameters for Random Forest with 5-fold classification and our learning parameters are:

A. Criterion: criterion is a parameter that specifies the function used to measure the quality of a split. and my options were:
    a. Gini
    b. Entropy
    c. Log loss
B. Number of estimators: number of decision trees in the forest. and my options were:
    a. 75
    b. 100
    c. 125
C. Max depth: maximum depth of each tree. and my options were:
    a. None, it means that nodes are expanded until all leaves are pure
    b. 10
    c. 20

After learning we find that best parameters are:

```
best_criterion: gini | best_n_estimators: 100 | best_max_depth: 20 | best_data: Selected | best_score: 0.8460418
638687965
best_criterion_f1: entropy | best_n_estimators_f1: 100 | best_max_depth_f1: None | best_data_f1: Selected | best_
score_f1: 0.7570210378265279
```

We pick our best parameters based on f1-score and accuracy but we only build the model with best average f1-score. Here it shows that our feature-selected data and entropy criterion with 100 estimators have had the best f1-score which shows that selecting our features can help us in classifying with Random Forest.
After that we learn our model on the all of the train data (because of computation limits we have found our best parameters only on 50% of each dataset but we have kept distribution between our labels).

```
accuracy 0.7894730316181029
accuracy_train 0.8995021487448043
precision 0.7187908614595638
precision_train 0.8965518306699106
recall 0.7547822959728064
recall_train 0.8888878323875101
f1 0.7175547170134807
f1_train 0.865250103047276
confusion [[28524  7715    10   692    51     8]
 [ 1010  3359     6   405  1273     9]
 [   13    56  2777   349   301     0]
 [  103   311   188  7524  2992    14]
 [   19   144    25   783  3105    13]
 [    8    34     3   350    94 18382]]
confision_train [[55520   477     1     2     0     0]
 [   44 16660     8    41  1431     0]
 [    0     0  8679    49  1763     0]
 [    3    30    45 20502 12812     1]
 [    0     3    19    71 12171     0]
 [    0     3     0    27   288 39682]]
```

Here we can see that our accuracy is good but because of the data's imbalancement our f1-score is not as well as our accuracy. And from confusion matrices we can see that again forth label (*Exploits*) is not classifying good (model is overfitted on the forth label) but first and last label (*Normal and Generic*) are classifying very good because of their repetition.

Third we try to learn best parameters for KNN (K-Nearest Neighbors) with 5-fold classification and our learning parameters are:

A. Number of neighbors: it shows that our model looks for how many neighbors to classify.
    a. 3
    b. 5
    c. 7
    d. 11
B. Weights:
    a. Uniform: All neighboring points have an equal contribution to the classification decision.
    b. Distance: Neighboring points have weights inversely proportional to their distance from the query point, giving closer points more influence.

After learning we find that best parameters are:

```
best_N_neighbor:  11 | best_Weight:  distance | best_data:  Selected | best_score:  0.828405679101493
best_N_neighbor_f1:  7 | best_Weight_f1:  distance | best_data_f1:  Selected | best_score_f1:  0.7350304250213828
```

We pick our best parameters based on f1-score and accuracy but we only build the model with best average f1-score. Here it shows that our feature-selected data and distance weight function with 7 nearest neighbors have had the best f1-score which shows that selecting our features and weighting them based on their distance from our test point can help us in classifying with K-Nearest Neighbors.
After that we learn our model on the all of the train data (because of computation limits we have found our best parameters only on 50% of each dataset but we have kept distribution between our labels).

```
accuracy 0.7737259764414135
accuracy_train 0.9154827043655919
precision 0.6519077294540995
precision_train 0.8777093053181296
recall 0.6741341097346872
recall_train 0.8426953497587979
f1 0.6518569710021623
f1_train 0.8570019844205564
confusion [[28425  7280   197   902   166    30]
 [ 1370  2701   589  1040   358     4]
 [   42   104  2739   526    80     5]
 [  221   512   260  9140   962    37]
 [   56   255    51  2675  1037    15]
 [   27    81    18   331    55 18359]]
confision_train [[55822   178     0     0     0     0]
 [  285 16439    14  1044   402     0]
 [    0     8  8698  1235   550     0]
 [    0    37   170 29923  3263     0]
 [    0     4   105  6782  5373     0]
 [    0     3     4   204   108 39681]]
```

Here we can see that our accuracy is good but because of the data's imbalancement our f1-score is not. And from confusion matrices we can see that second label (*Fuzzers*) is not classifying good but first and last label (*Normal and Generic*) are classifying very good because of their repetition.

Forth we try to learn best parameters for MLP (Multileyer Perceptron) with 5-fold classification and our learning parameters are:

A. Hidden layers: here we fix number of layers and size of them, but again because of computation limits we choose only two layer option of (100, 25) .
B. Learning rates:
    a. Constant: Uses a constant learning rate throughout the training process, which may lead to slower convergence or overshooting in some cases.
    b. Adaptive: Adjusts the learning rate dynamically based on the progress of the training, such as reducing the learning rate when approaching a minimum or increasing it for faster initial learning
    c. Invscaling: Gradually decreases the learning rate over time based on the inverse of a scaling factor, aiding convergence and stability during training.
C. Activation functions:
    a. Logistic: Maps the input to a range between 0 and 1, resembling a logistic function, commonly used in binary classification problems and here because of the computational limits and it's best performance in binary classification we ignore it.
    b. Identity: Provides a linear mapping where the output is equal to the input, often used in regression problems or as a pass-through activation function and here because of the computational limits and it's best performance in regression problems we ignore it. .
    c. Relu: Sets all negative inputs to zero and keeps positive inputs unchanged, commonly used in deep learning models to introduce non-linearity.
    d. Tanh: Rescales the input to a range between -1 and 1, similar to the logistic function but symmetric around zero.
D. Batch size: Refers to the number of training examples used in each forward and backward pass to update the model's parameters but because of computation limits we have avoided it.
E. Maximum number of iterations: Because of computation limits we have avoided it.

After learning we find that best parameters are:

```
best_Hidden_layer_size:  (100, 25) | best_Learning_rate:  constant | best_Activation:  relu | best_data:  Selected |
best_score:  0.8236416843418745
best_Hidden_layer_size_f1:  (100, 25) | best_Learning_rate_f1:  invscaling | best_Activation_f1:  relu | best_data_f
1:  Main | best_score_f1:  0.7203744428524955
```

We pick our best parameters based on f1-score and accuracy but we only build the model with best average f1-score. Here it shows that our main data and invscaling learning rate withrelu activation function have had the best f1-score which shows that feature extraction/reduction is not helping us in MLP but invscaling can help our model to classify better.

After that we learn our model on the all of the train data (because of computation limits we have found our best parameters only on 20% of each dataset but we have kept distribution between our labels).

```
accuracy 0.7782021078735276
accuracy_train 0.8448383157598103
precision 0.6443920657307128
precision_train 0.7932559942019001
recall 0.6681645046402531
recall_train 0.7309612491772005
f1 0.6360463518986866
f1_train 0.7341986538345932
confusion [[28353  6776   526  1296    31    18]
 [ 1231  2870   225  1382   351     3]
 [   12    43  2765   672     4     0]
 [  125   264   135 10102   481    25]
 [   36   101    48  3466   425    13]
 [   10    83    26   422    83 18247]]
confision_train [[51673  3722   132   458    13     2]
 [ 3519 12637   221  1699    96    12]
 [   20    22  7761  2509   179     0]
 [  278   650   348 31197   868    52]
 [   58   184    82 10540  1383    17]
 [   12    63    27   579    67 39252]]
```

Here we can see that our accuracy is good but because of the data's imbalancement our f1-score is not. And from confusion matrices we can see that fifth label (*Dos*) is not classifying good but first and last label (*Normal and Generic*) are classifying very good because of their repetition.

Fifth we try to learn best parameters for a little more complicated method, Ada Boost, with 5-fold classification and our learning parameters are:

A. Estimator: We use it's default estimator which is decision trees because so far decision trees have done best for our data.
B. Number of estimators:
    a. 30
    b. 50
    c. 100

After learning we find that best parameters are:

```
best_n_estimator:  50 | best_data:  Main | best_score:  0.7559706175349302
best_n_estimator_f1:  30 | best_data_f1:  Main | best_score_f1:  0.6336310549014645
```

We pick our best parameters based on f1-score and accuracy but we only build the model with best average f1-score. Here it shows that our main data and 30 estimators have had the best f1-score which shows that feature extraction/reduction and increasing number of estimators are not helping us in Ada Boosting.
After that we learn our model on the all of the train data (because of computation limits we have found our best parameters only on 50% of each dataset but we have kept distribution between our labels).

```
accuracy 0.7193676379417235
accuracy_train 0.7454911584435103
precision 0.646581641228412
precision_train 0.6715655044451146
recall 0.6210946262880989
recall_train 0.6516006524553355
f1 0.595560398256201
f1_train 0.6466509826008724
confusion [[26296  8185    34  2441    42     2]
 [ 1563  3885    64   205    46   299]
 [  128   515  2521   265    43    24]
 [  600  2525   423  6816   298   470]
 [   88  2049   172  1117   320   343]
 [   38   110    13   511    20 18179]]
confision_train [[48498  5993    54  1234    35   186]
 [ 4610 11437   201  1228   347   361]
 [  196  1749  7467   582   380   117]
 [  668  8801  1650 18266  3248   760]
 [  169  6127   460  2729  2176   603]
 [   25   259    22   492    65 39137]]
```

Here we can see that our accuracy is not very bad but because of the data's imbalancement our f1-score is. And from confusion matrices we can see that most of the labels are not classifying good but first and last label (*Normal and Generic*) are classifying very good because of their repetition.
At last we try learn a stacking model based on predictions we have so far. We pick our top 3 models base on their train f1-scores (Random Forest and KNN and MLP) and we add our predictions of these samples as a feature to our sampled train and test dataset (we use sampled data set so that only important features exist and they doesn't take our new features importance and doesn't confuse our model) and after that we learn a KNN and a Random forest and a MLP model on these new data sets and we calculate their evaluation metrics and we have that:
For KNN we have:

```
accuracy 0.7794172349659021
accuracy_train 0.9153359321795083
precision 0.6720739102498627
precision_train 0.8778244622397312
recall 0.6927259143446811
recall_train 0.844781226885511
f1 0.6695161287917749
f1_train 0.8587006318332118
confusion [[28269  7867   192   600    68     4]
 [ 1213  3216   213  1050   367     3]
 [   36    71  2786   524    79     0]
 [  110   513   313  9166  1018    12]
 [   23   231    84  2631  1108    12]
 [    9    57    22   335   133 18315]]
confision_train [[55815   185     0     0     0     0]
 [  257 16470     8  1013   436     0]
 [    0    12  8692  1195   592     0]
 [    0    57   117 29632  3587     0]
 [    0    13    80  6550  5621     0]
 [    0     4     2   199   114 39681]]
```

Which is better than it's own metrics but still is not very good. And for Random Forest we have:

```
accuracy 0.7790948543087415
accuracy_train 0.9000481412770355
precision 0.7050808213023174
precision_train 0.8972342467223471
recall 0.7423337163165606
recall_train 0.8893394828425949
f1 0.7052896008699437
f1_train 0.8658242827072087
confusion [[28122  7950    30   852    40     6]
 [ 1245  3231   154   221  1208     3]
 [   32    83  2766   326   288     1]
 [  133   519   198  7349  2921    12]
 [   22   257    35   747  3014    14]
 [   14    79     4   316   106 18352]]
confision_train [[55591   409     0     0     0     0]
 [   41 16673     8    34  1428     0]
 [    0     0  8686    48  1757     0]
 [    0    37    47 20504 12804     1]
 [    0     2    21    70 12171     0]
 [    0     3     0    27   288 39682]]
```

which is good but surprisingly random forest works better on it's own. And for MLP we have:

```
accuracy 0.7758586484810911
accuracy_train 0.9185590493859052
precision 0.6681549075125347
precision_train 0.9222977489181795
recall 0.667973790712623
recall_train 0.8249131412530764
f1 0.6369834198892298
f1_train 0.8401969148354996
confusion [[28203  7713   247   731   102     4]
 [ 1249  3013   436  1345    16     3]
 [   25   123  2740   585    23     0]
 [  112   612   282  9841   273    12]
 [   17   278    49  3281   448    16]
 [    8    75    17   345    98 18328]]
confision_train [[55665   335     0     0     0     0]
 [  116 16598     8  1415    47     0]
 [    0     0  8683  1714    94     0]
 [    0    35    47 32926   385     0]
 [    0     4    18  9334  2906     2]
 [    0     3     0   312     3 39682]]
```

Which is a little better but still not good enough.

Comparison

At last our best model was our random forest with f1-score of **71.7%** and accuracy of **79%**. Also there is a point i must mention that we have calculated evaluation metrics both for test data and train data for all of our models, but we have only made decisions based on evaluation matrics of train data and evaluation matrics achieved from 5-fold cross validation of our train data. But this point is not regarded by authors of the paper. They have used their own test data for many things. First, they have changed their train data based on their test data so that their model predict more like test data which is using test data for adjusting their model. Second, they have used half of the test data for validation and this means they have used test data for adjusting their model.

For feature extraction we used 4 strong methods which are PCA, Autoencoders, RFE(we used random forest for estimator and they have used MLP for their estimator) and Random Forest feature importance(we used it to find better features for less common lables but they have used it for all of their labels). And for classfication we have used SVM, KNN, Random Forest, MLP(like paper), Ada Boost and Stacking model.

In Conclusion we can say we have tried more and stronger models but because of the computational limits we were not able to catch paper's best results (f1-score of **82.85%** and accuracy of **84.24%**) from our data.

Ashkan Zarkhah
610399196