Data exploration and preprocessing

Data exploration

The first step is always to examine what your data looks like. The more you understand your data, the better decisions you can make. For this reason, we begin by inspecting a row of our data and its shape.

id	1
dur	0.121478
proto	tcp
service	-
state	FIN
spkts	6
dpkts	4
sbytes	258
dbytes	172
rate	74.08749
sttl	252
dttl	254
sload	14158.94238
dload	8495.365234
sloss	0
dloss	0
sinpkt	24.2956
dinpkt	8.375
sjit	30.177547
djit	11.830604
swin	255
stcpb	621772692
dtcpb	2202533631
dwin	255
tcprtt	0.0
synack	0.0
ackdat	0.0
smean	43
dmean	43
trans_depth	0
response_body_len	0
ct_srv_src	1
ct_state_ttl	0
ct_dst_ltm	1
ct_src_dport_ltm	1
dttl	254
sload	14158.94238
dload	8495.365234
sloss	0
dloss	0
sinpkt	24.2956
dinpkt	8.375
sjit	30.177547
djit	11.830604
swin	255
stcpb	621772692
dtcpb	2202533631
dwin	255
tcprtt	0.0
synack	0.0
ackdat	0.0
smean	43
dmean	43
trans_depth	0
response_body_len	0
ct_srv_src	1
ct_state_ttl	0
ct_dst_ltm	1
ct_src_dport_ltm	1
ct_dst_sport_ltm	1
ct_dst_src_ltm	1
is_ftp_login	0
ct_ftp_cmd	0
ct_flw_http_mthd	0
ct_src_ltm	1
ct_srv_dst	1
is_sm_ips_ports	0
attack_cat	Normal

Name: 0, dtype: object

```
(175341, 44)
```

After closely examining our data, we have identified several key points:

- We have categorical values within our data.
- The presence of IDs in the data can potentially introduce bias or corruption to our predictions.
- Our classes are also categorical.
- Some of the data entries have significantly larger values than others, indicating the need for normalization.

Additionally, upon examining the distribution of the training data among classes, we discovered that we are dealing with an imbalanced dataset. Some classes contain less than 1% of the data, while others comprise approximately one-third of the data.

```
print(df['attack_cat'].value_counts())
```

```
Normal            56000
Generic           40000
Exploits          33393
Fuzzers           18184
DoS               12264
Reconnaissance    10491
Analysis           2000
Backdoor           1746
Shellcode          1133
Worms               130
Name: attack_cat, dtype: int64
```

Preprocessing

Missing values

None were found.

In [7]:
```
# We check for missing values.
# Check if any rows have null values
null_rows = df.isnull().any(axis=1)
print('Number of rows with null values:', sum(null_rows))
```

```
Number of rows with null values: 0
```

Duplicate values

While many duplicated values were found in the data, it is important to consider the nature of the data. Each value represents a configuration of a network intrusion, and it is logical to have the same network intrusions occurring multiple times. Additionally, having more instances of a particular configuration suggests that this type of intrusion is more likely to happen. By including these duplicates in the data, we can improve our understanding and predictions related to these specific intrusion types.

In [8]:
```
duplicate_rows = df[df.duplicated()]
print("Number of duplicate rows: ", len(duplicate_rows))


# That's a lot of duplicated values! but they make sense as they will be discussed in the report
```

```
Number of duplicate rows:  66939
```

Excluding minority classes

As mentioned earlier, some of the classes in our data are extremely minority classes, comprising approximately 1% of the data. In line with the approach described in the paper, we will exclude these classes from our analysis. The excluded classes are 'Analysis', 'Backdoor', 'Shellcode', and 'Worms', which accounted for only 1.141%, 0.996%, 0.646%, and 0.074% of the training set, respectively.

Deleting id column

The inclusion of IDs in the dataset has the potential to introduce bias or corruption into our predictions.

Change categorical values to numeral

After analyzing the data, we have identified three categorical variables, excluding the classes that are directly mapped to numbers. The recommended approach for converting categorical values to numerical ones is to use one-hot encoding, where each category is represented in a binary form. One advantage of this approach is that the numerical representation of categories does not imply any numerical relationship. For example, assigning 'bad' as '1' and 'good' as '2' does not imply that 'good' is numerically greater than 'bad'.

However, when applying one-hot encoding using get_dummies, we found that due to the large number of categorical values, converting all categorical values to one-hot would require adding 155 additional columns. Considering our objective, which aligns with the paper's goal of feature reduction, adding such a significant number of columns would not be a wise decision. Therefore, we opted to assign integers to each unique categorical value within the column.

Normalization

As mentioned before, since our categorical values have been converted to numerical representations, we can proceed with normalization techniques. Considering the varying ranges of our columns, scaling is necessary to achieve our objective of mapping the data to a range of 0 to 1. To accomplish this, we will utilize Min-Max scaling.

It's important to highlight that Gaussian normalization is not applicable in this scenario. When using averaging or mean values for normalization, the frequency of categorical values becomes relevant. The cumulative effect of multiple occurrences of the same categorical value will indeed be reflected in the normalization process. Therefore, in our case, Min-Max scaling is the appropriate choice for achieving our normalization goals.

Visualization

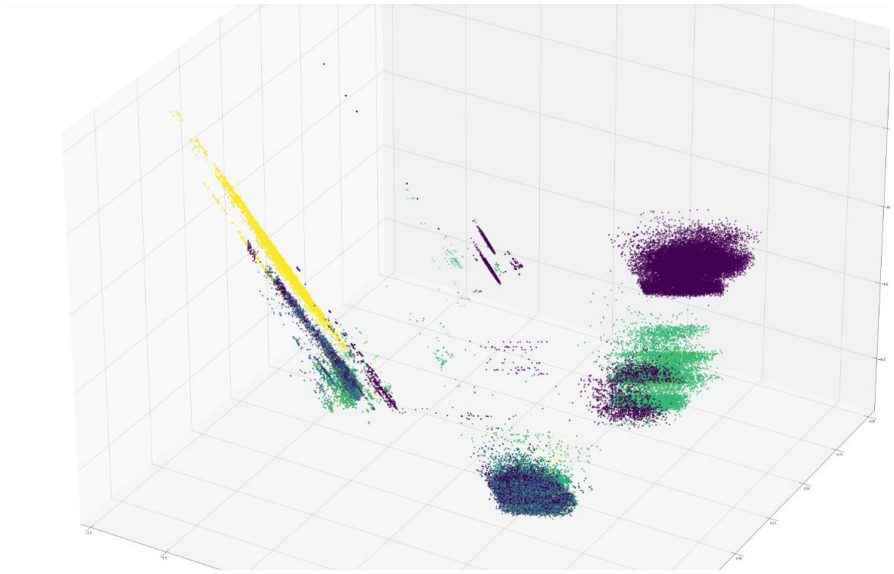
After performing the aforementioned preprocessing steps, our data will be as follows.

```
Name: attack_cat, Length: 170332, dtype: int64
                dur     proto   service  state     spkts     dpkts    sbytes  \
0       2.024634e-03  0.000000  0.000000  0.000  0.000520  0.000364  0.000018
1       1.083170e-02  0.000000  0.000000  0.000  0.001352  0.003463  0.000054
2       2.705215e-02  0.000000  0.000000  0.000  0.000728  0.001458  0.000026
3       2.802737e-02  0.000000  0.083333  0.000  0.001144  0.001093  0.000046
4       7.490901e-03  0.000000  0.000000  0.000  0.000936  0.000547  0.000039
...              ...       ...       ...    ...       ...       ...       ...
175335  1.000000e-07  0.007576  0.500000  0.125  0.000104  0.000000  0.000007
175336  1.500000e-07  0.007576  0.500000  0.125  0.000104  0.000000  0.000007
175338  1.500000e-07  0.007576  0.500000  0.125  0.000104  0.000000  0.000007
175339  1.500000e-07  0.007576  0.500000  0.125  0.000104  0.000000  0.000007
175340  1.500000e-07  0.007576  0.500000  0.125  0.000104  0.000000  0.000007

          dbytes      rate      sttl  ...  ct_dst_ltm  ct_src_dport_ltm  \
0       0.000012  0.000074  0.988235  ...        0.00              0.00
1       0.002867  0.000078  0.243137  ...        0.00              0.00
2       0.000900  0.000014  0.243137  ...        0.02              0.00
3       0.000053  0.000014  0.243137  ...        0.02              0.00
4       0.000018  0.000033  0.996078  ...        0.02              0.02
...          ...       ...       ...  ...         ...               ...
175335  0.000000  0.166667  0.996078  ...        0.64              0.64
175336  0.000000  0.111111  0.996078  ...        0.46              0.46
175338  0.000000  0.111111  0.996078  ...        0.04              0.04
175339  0.000000  0.111111  0.996078  ...        0.58              0.58
175340  0.000000  0.111111  0.996078  ...        0.58              0.58

        ct_dst_sport_ltm  ct_dst_src_ltm  is_ftp_login  ct_ftp_cmd  \
0               0.000000        0.000000          0.00        0.00
1               0.000000        0.015625          0.00        0.00
2               0.000000        0.031250          0.00        0.00
3               0.000000        0.031250          0.25        0.25
4               0.000000        0.609375          0.00        0.00
...                  ...             ...           ...         ...
175335          0.355556        0.687500          0.00        0.00
175336          0.266667        0.359375          0.00        0.00
```

To visualize our data, we employ PCA (Principal Component Analysis) to reduce its dimensionality to three. Subsequently, we use matplotlib's plt module to plot the data.



From the visualization, it is evident that certain classes are well-separated, while others pose challenges in terms of distinctiveness. It is important to note that the visualization represents a three-dimensional representation of our data, and the actual data may exhibit different characteristics.

In the next phase, we will introduce various dimension reduction techniques and compare their effectiveness.

Feature processing

In this section, we will propose various supervised feature extraction methods. These methods align with the suggestions provided in the paper.

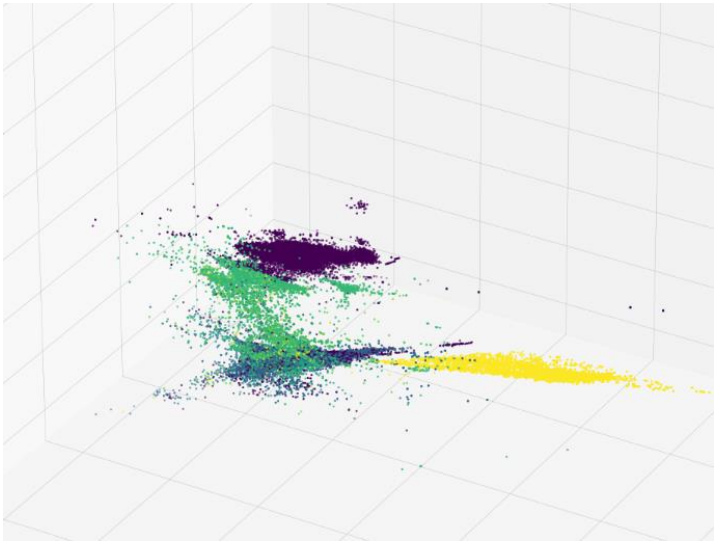
We will explore and visualize two primary categories of techniques: supervised dimension reduction approaches and feature selection approaches. The visualizations will play a crucial role in aiding our decision-making process, as they will provide insights into the effectiveness of these methods.

Supervised dimension reduction

Linear discriminant analysis (LDA)

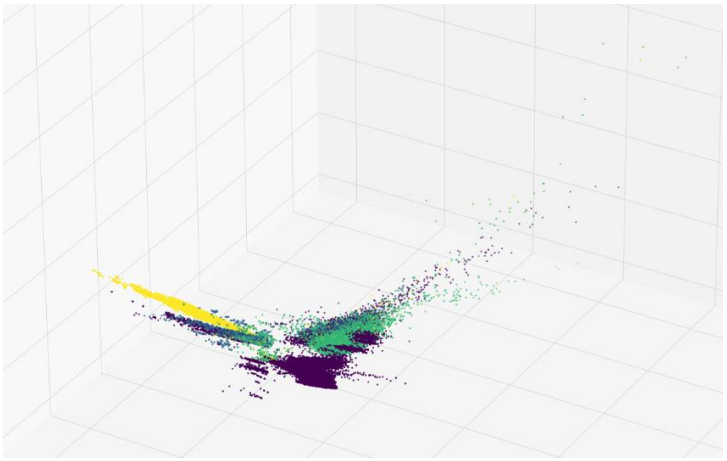
The first method we employed is Linear Discriminant Analysis (LDA). LDA is a dimensionality reduction technique that aims to find linear combinations of features that maximize the separation between different classes while minimizing the variation within each class.

In our LDA implementation, we obtained four components. These components are computed based on the class labels of the data. The visualization of the LDA results shows that the green dots are better distinguished compared to the original data. This improvement in separation is a result of LDA's ability to project the data onto a lower-dimensional space that optimally discriminates between the classes.



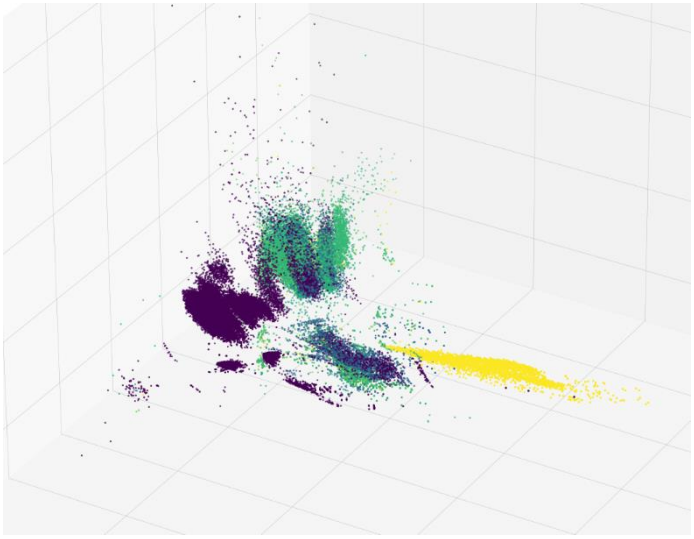
Partial least square (PLS)

The next method we utilized is Partial Least Squares (PLS). PLS is a multivariate regression technique that aims to find linear combinations of features that maximize the covariance between the predictors (features) and the response variable (class labels). In our PLS implementation, we obtained six components. The visualization of the PLS results showcases the distinct characteristics of this method compared to LDA.



Union of LDA and PLS

For the next step, we combined the LDA components and PLS components to create a 10-dimensional dataset. We then visualized this dataset and observed that it exhibits significantly improved distinctiveness compared to the previous visualizations. This enhanced distinctiveness gives us hope that utilizing this new dataset will lead to better predictions. Consequently, we will consider this augmented dataset as one of our experimental data and employ it in our modeling process.



Feature selection:

The next category of feature extraction methods we will explore is feature selection. Feature selection techniques aim to identify a subset of relevant features from the original dataset while discarding irrelevant or redundant ones. These methods consider the predictive power of individual features or their ability to contribute to the classification task.

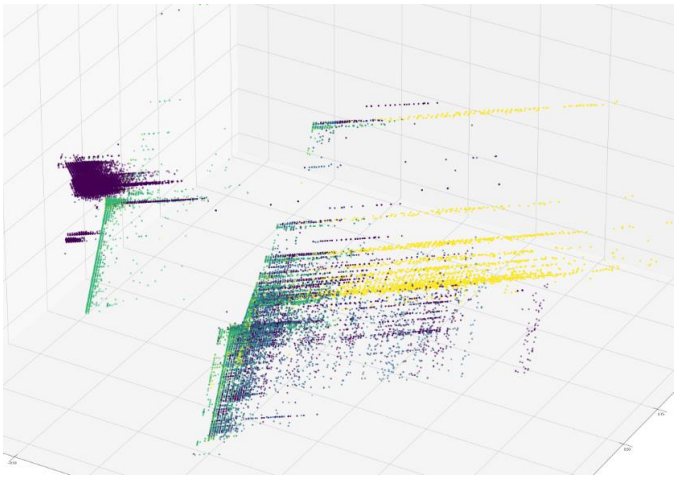
The paper's utilization of feature selection methods for feature reduction serves as an encouragement for us to follow suit. These methods help to identify the most informative features, potentially improving the model's efficiency and generalization performance.

Recursive feature elimination (RFE)

For the first method of this category, we employed Recursive Feature Elimination (RFE) as our feature selection model. RFE is a technique that recursively selects features by training a model and eliminating the least important features until a desired number of features is reached.

In our implementation, we used the Random Forest classifier as our estimator. The RFE algorithm evaluated the importance of each feature based on the performance of the Random Forest classifier. We obtained the indices of the 10 most effective columns through the RFE process, representing the selected features and then visualized it.

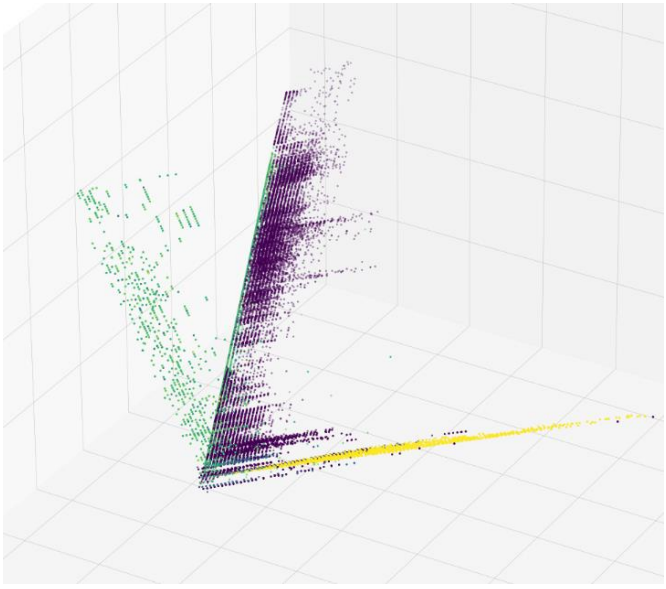
The selected indices were [6, 8, 9, 12, 26, 31, 33, 34, 35, 40]. Upon visualizing the data based on these selected features, we observed a completely different pattern compared to the previous visualizations from LDA and PLS.



SelectFromModel

The next feature selection method we employed was SelectFromModel. SelectFromModel is a technique that selects features based on the importance scores assigned by a specific model.

In our implementation, we used logistic regression as our estimator. The SelectFromModel algorithm evaluated the importance scores assigned by the logistic regression model to each feature. After applying SelectFromModel, we identified the 10 highly scored features with indices [1, 3, 12, 13, 24, 27, 28, 33, 34, 40]. Subsequently, we visualized the data based on these selected features. The visualization demonstrated that these features exhibit a similar essence to those selected by RFE, but with more distinct patterns and separability.

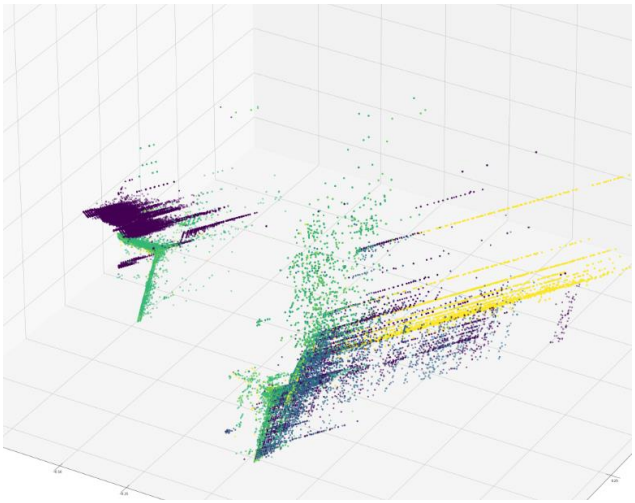


Union of RFE and SelectFromModel

For our last feature extraction method, we followed the essence of the paper and took the union of the columns selected by RFE and SelectFromModel. Since there were some common features identified by both methods, the resulting dataset consists of 16 dimensions.

Due to the improved visualization and alignment with the paper's approach, we decided to utilize this extracted feature set as our training data. We will apply our models on this dataset to make predictions.

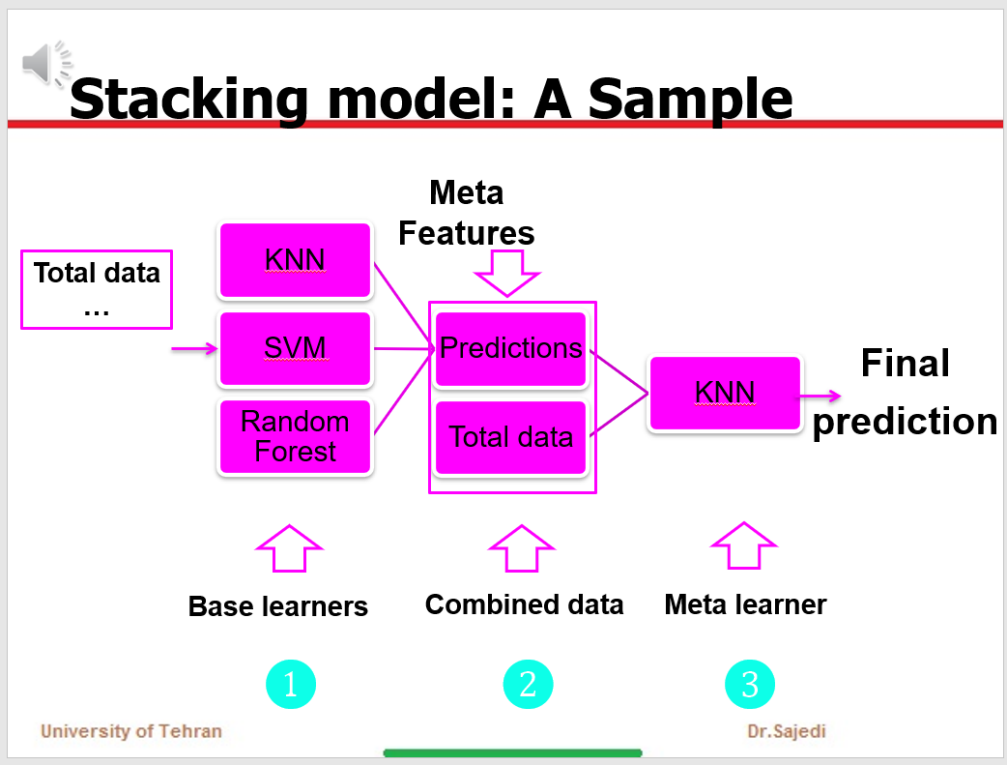
By combining the strengths of RFE and SelectFromModel and incorporating the shared informative features, we aim to benefit from their respective feature selection capabilities. This integrated feature set will enable us to perform classification tasks in line with the approach presented in the paper.



In the next section, we will provide an overview of the models we trained and explain our approach, including the reasons for selecting these specific models.

Model selection and training

In this section, our main focus is on training a stacking model. We have chosen three base learners for our stacking model: Random Forest Classifier, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Here's why we selected these models:



- Random Forest Classifier:
  - Random Forests have shown good performance in various tasks, including handling imbalanced data.
  - They are capable of capturing complex relationships between features and are robust against overfitting.

- K-Nearest Neighbors (KNN):
  - The visualizations have revealed that instances of the same class tend to be located close to each other in the data space.
  - KNN, as a proximity-based algorithm, can leverage this characteristic to make accurate predictions.

- Support Vector Machine (SVM):
  - SVMs offer a unique advantage of finding the optimal margin-based decision boundary between classes.
  - They can handle both linear and non-linear relationships between features.

To train the stacking model, we follow the following steps:

1. Hyperparameter Tuning:
    - We use cross-validation (CV) to find the best hyperparameters for each base learner model (Random Forest, SVM, and KNN).
    - By searching through a predefined parameter space, we aim to identify the optimal hyperparameters that maximize performance.
    - Considering the imbalanced nature of our data, we utilize the F1 score as the evaluation metric to compare the hyperparameters.

2. Training Base Learners:
    - Once we have the best hyperparameters, we train each base learner model on the training data using these hyperparameters.
    - This step involves fitting the models to the training data and learning their respective decision boundaries.

3. Generating Predictions:
    - We then use each trained base learner to predict the test data.
    - The predictions from the base learners are treated as new features and added to our dataset.

4. Meta Learner - K-Nearest Neighbors (KNN):
    - Finally, we employ KNN as the meta learner, utilizing the original features and the predictions from the base learners.
    - KNN learns to make predictions by considering both the original features and the added predictions from the base learners.

This approach will be applied to two best-extracted datasets: the union of LDA and PLS features and the union of RFE and SelectFromModel features. Each step will be explained thoroughly and executed to construct the stacking model effectively.

Base learners

Random forest classifier

In the hyperparameter search for our Random Forest Classifier, we employed cross-validation (CVfold) to assess the performance of different hyperparameter combinations. Here's an explanation for the chosen hyperparameters:

- n_estimators: We searched for values between 100 and 150. The number of estimators determines the number of decision trees in the Random Forest ensemble. A higher number of estimators can potentially improve the model's performance by reducing overfitting.

- criterion: We considered two options, 'gini' and 'entropy', for the criterion. This parameter defines the function used to measure the quality of a split in the decision tree.

- min_samples_split: We searched for values of 2 and 5. This parameter determines the minimum number of samples required to split an internal node. A higher value can prevent overfitting by ensuring a minimum number of samples in each split.

Based on cross-validation using the F1 score as the evaluation metric, we identified the best hyperparameters for the Union of LDA and PLS as follows:

```
1
1
Best Hyperparameters: {'n': 100, 'criterion': 'entropy', 'min_samples_split': 5}
Best Score: 0.7677819799759543
```

And for the Union of RFE and SelectFromModel, the best hyperparameters were:

```
Best Hyperparameters: {'n': 150, 'criterion': 'gini', 'min_samples_split': 5}
Best Score: 0.7899104466891338
```

KNN:

In the hyperparameter search for our K-Nearest Neighbors (KNN) model, we explored different values for the hyperparameters k and weights. Here's an explanation for the chosen hyperparameters:

- k (number of neighbors): We explored three different values for the number of neighbors, namely 3, 5, and 7. A higher value of k leads to smoother decision boundaries On the other hand, a lower value of k makes the model more sensitive to outliers.

- weights: We evaluated two options, 'uniform' and 'distance', for the weight parameter. 'Uniform' assigns equal weights to all neighbors, while 'distance' weights the neighbors inversely proportional to their distance. Choosing between these options allows us to assess the impact of different weighting strategies on model performance.

Union of LDA and PLS:

```
Best Score 0.7649825285126755 with 7 neighbors and weights='uniform'
```

Union of RFE and SelectFromModel:

```
Best Score 0.779328686119985 with 7 neighbors and weights='uniform'
```

SVM

Due to computational constraints, we examined only a subset of hyperparameters for SVM model. we explored different values for the hyperparameters kernel and degree. Here's an explanation for the chosen hyperparameters:

- Linear Kernel: This kernel uses a linear function to create a decision boundary. It is suitable for linearly separable data.

- Polynomial Kernel: We tested the polynomial kernel with degrees of 3 and 4. The degree parameter determines the complexity of the polynomial function used to map the data. By testing both an odd (degree 3) and an even (degree 4) value, we aimed to cover a broader spectrum of polynomial complexities.

- RBF Kernel: The RBF kernel uses a Gaussian function to create decision boundaries. It is suitable for data with non-linear relationships.

Union of LDA and PLS:

```
Best Hyperparameters: {'kernel': 'rbf',
Best Score: 0.7668182418627131
```

Union of RFE and SelectFromModel:

```
Best Hyperparameters: {'kernel': 'rbf',
Best Score: 0.7697991699359734
```

Conclusion

For both datasets, the Random Forest Classifier emerged as the best model during the training phase.

Training

Now that we have identified the best hyperparameters, it is time to train our training datasets using these hyperparameters on the selected models and make predictions on the test data. However, before proceeding with the training and prediction process, we need to transform the test data into two different data spaces: the union of LDA and PLS features and the union of RFE and SelectFromModel features.

By applying the same feature extraction techniques used on the training data to the test data, we ensure consistency and enable fair comparisons of the models' performance on the transformed test data.

To facilitate further analysis and evaluation of our models, we will save the predictions obtained from each model. These prediction results will be saved separately and can be later added as additional features to our transformed test datasets.

Test set transformation

For the transformation of the test data into the Union of LDA and PLS space, we apply the same LDA and PLS transformations used on the training data. This involves computing the LDA and PLS components for the test data separately and then merging the resulting dataframes to create the transformed test data in the Union of LDA and PLS space.

Regarding the transformation using RFE and SelectFromModel, we extract the subset of 16 columns that were found to be the most useful in the Union of RFE and SelectFromModel. These 16 columns, which demonstrated high importance in feature selection, are selected from the test data to create the transformed test data in the Union of RFE and SelectFromModel space.

By performing these transformations, we ensure that the test data is aligned with the respective feature extraction approaches used on the training data. This consistency allows for accurate and fair evaluation of the models' performance on the transformed test data.

Stacking model

After training our six models (Random Forest, SVM, and KNN for both the train_lda_pls and train_selection datasets), we proceed to make predictions on both the corresponding training data and the test data.

We utilize the predictions obtained from the training data to create a new column in the train dataset, representing the model's prediction for each instance. Similarly, we use the predictions obtained from the test data to evaluate the performance of the models and add them as additional columns to the respective test datasets.

Following these steps, we obtain new transformed training data and test data. These transformed datasets serve as inputs for our meta-learner, which is a KNN model. We train the KNN model on the transformed training data and use it to predict the outcomes for the transformed test data.

The predictions generated by the KNN model serve as our final predictions for the classification task. By employing this stacking ensemble approach, we leverage the collective wisdom of multiple models to enhance the accuracy and robustness of our final prediction.

Now it's time to evaluate our models.

Performance evaluation and comparison

First, we evaluate the performance of our base learner models using various metrics. It is indeed worth noting that when dealing with imbalanced datasets, using accuracy as the sole evaluation metric may not provide an accurate representation of model performance. This is because accuracy can be misleading when the class distribution is highly skewed, and the model tends to favor the majority class.

In such cases, metrics like F1 score, precision, recall, and the confusion matrix offer more comprehensive insights into the model's performance.

LDA_PLS transformed data

Random forest regression

```
Accuracy: 0.7577557346559206
Precision: 0.8204269126911993
Recall: 0.7577557346559206
F1 Score: 0.7705476707613479
[[27103  8227   624   987    52     7]
 [ 1162  3680   177   878   159     6]
 [   77   245  2527   627    17     3]
 [  229  1086   213  9284   306    14]
 [   79   421    89  3116   374    10]
 [   10   149    15   501    51 18145]]
```

The F1 score we obtained is even better than the one we observed during cross-validation on our training data. The high accuracy and F1 score, as well as the examination of the confusion matrix, indicate that we have not encountered bias towards the larger class.

However, upon closer inspection of the confusion matrix, we notice that certain classes, such as DoS, Fuzzer, and Reconnaissance, are frequently misclassified as Exploits. This observation aligns with the findings mentioned in the paper, where these classes are known to be challenging to classify accurately.

KNN

```
print(cm)
```

```
Accuracy: 0.7595164290142592
Precision: 0.8084912616737542
Recall: 0.7595164290142592
F1 Score: 0.7748479022255792
[[27876  7452   645   948    74     5]
 [ 1728  3167   174   715   263    15]
 [  140   214  2512   526   101     3]
 [  440   788   404  8524   952    24]
 [  148   274   110  2566   972    19]
 [   32   131    31   415    58 18204]]
```

```
[47]: # performing SVM with best parameters for lda and
```

The K-Nearest Neighbors (KNN) model has exhibited slightly better results compared to the Random Forest Regressor. This is evident in the evaluation metrics, which indicate improved performance for the KNN model.

SVM

```
Accuracy: 0.7121636701797892
Precision: 0.8472314720370288
Recall: 0.7121636701797892
F1 Score: 0.7255466712712556
[[22961 11302  1293  1442     0     2]
 [  320  4464   189  1081     0     8]
 [    4   424  2256   805     0     7]
 [  163  1039   348  9570     2    10]
 [   66   333   206  3445    19    20]
 [    3   195    39   468     0 18166]]
```

While the Support Vector Machine (SVM) model may not perform as well as other models overall, it can still be considered acceptable in terms of its performance. It's worth noting that SVM demonstrates higher precision compared to other models, indicating a lower false positive rate.

However, it is important to consider, it may be possible that a more intricate SVM model, with a different kernel or parameter configuration, could potentially yield improved results. Unfortunately, due to computational constraints, we were unable to explore these more complex SVM variations

Stacking model

```
Accuracy: 0.7574333539987601
Precision: 0.8229103887746132
Recall: 0.7574333539987601
F1 Score: 0.7771378098855
[[27152  8180   618   989    55     6]
 [ 1168  3652   231   670   339     2]
 [   77   246  2530   541    99     3]
 [  229  1068   318  8562   944    11]
 [   80   415   181  2367  1037     9]
 [   11   117    81   442    66 18154]]
```

Indeed, the stacking model, which was our primary focus, has demonstrated the best results among the models evaluated. This success validates the effectiveness of the stacking approach in improving the predictive performance.

Feature selection transformed data

Random forest classifier:

```
print(cm)
```

```
Accuracy: 0.7942839429634222
Precision: 0.8498636393971275
Recall: 0.7942839429634222
F1 Score: 0.8049082541829925
[[28621  7484    16   796    74     9]
 [  874  3413     2  1674    87    12]
 [   11    54  2790   594    47     0]
 [  102   313   173 10171   355    18]
 [   17   155    30  3153   725     9]
 [    6    40     3   427    56 18339]]
```

```
73]: # performing KNN with best parameters for dat
```

Our Random Forest model has showcased excellent performance, surpassing all the other models trained on the LDA_PLS transformed data, including the stacking model. This observation suggests that the feature selection method employed in our approach outperforms the supervised dimension reduction technique in terms of extracting informative features.

Moreover, the Random Forest model's performance is quite comparable to the results mentioned in the paper, and in some cases, it even outperforms the metrics reported in the paper. This outcome is particularly notable in terms of precision, where our model demonstrates superior results. The overall performance metrics achieved by our Random Forest model are also quite close to the metrics reported in the paper.

KNN

```
Accuracy: 0.7280471171729697
Precision: 0.7968456416146755
Recall: 0.7280471171729697
F1 Score: 0.750871531484838
[[28738  6831   285   824   179   143]
 [ 1217  2578   762  1134   346    25]
 [   32    65  2782   516   100     1]
 [  205   350   257  9232  1057    31]
 [   69   165    61  2644  1137    13]
 [ 1414  2630    85   425    67 14250]]
```

Although the K-Nearest Neighbors (KNN) model did not perform as well as the Random Forest model, it still demonstrated quite good results.

SVM

```
Accuracy: 0.6581897086174829
Precision: 0.8299644605994723
Recall: 0.6581897086174829
F1 Score: 0.6851810362798204
[[22712 10103  1956  2147     0    82]
 [  192  4296   384  1173     0    17]
 [    5   313  2641   537     0     0]
 [  122   777   460  9756     1    16]
 [   20   280   246  3530     5     8]
 [  116  4485   121   475     1 13673]]
```

```
76]: y_predict_Rf_Selectedtrain = RandomForest_se
```

The lower results obtained with the SVM model, specifically its struggle to predict the 5th class accurately, highlight a limitation or challenge in its performance. The consistently lower performance of the SVM model, compared to other models such as the Random Forest and KNN, suggests that the SVM algorithm might not be well-suited for capturing the underlying patterns and relationships present in our data.

Stacking model

```
Accuracy: 0.796751394916305
Precision: 0.8477254562570529
Recall: 0.796751394916305
F1 Score: 0.8118765727140949
[[28947  7158    16   797    74     8]
 [  890  3400    13  1367   380    12]
 [   11    58  2790   529   108     0]
 [  104   338   196  9482   994    18]
 [   17   166    54  2542  1301     9]
 [    7    40     4   419    63 18338]]
```

The stacking model for our feature-selected dataset stands out as the highlight of our project. It demonstrates superior performance in terms of accuracy, precision, recall, F1 score, and the confusion matrix when compared to all other models we have evaluated. These results highlight the effectiveness of the stacking approach in combining the strengths of multiple models to achieve optimal predictions.

Furthermore, our stacking model's performance is quite comparable to the model described in the paper. It even surpasses the reported precision metrics. This achievement further validates the reliability and competitiveness of our stacking model.

Conclusion and comparison

	Accuracy	Precision	Recall	F1
LDA_PLS RF	75.77	82.04	75.77	77.05
LDA_PLS KNN	75.95	80.84	75.95	77.48
LDA_PLS SVM	71.21	84.72	71.21	72.55
LDA_PLS Stacking	75.74	82.29	75.74	77.71
Selected RF	79.42	84.98	79.42	80.49
Selected KNN	72.80	79.6	72.80	75.08
Selected SVM	65.81	82.99	65.81	68.51
Selected Stacking	79.67	84.77	79.67	81.18
BEST of paper	84.24	83.60	84.24	82.85

1. The overall modeling efforts have yielded positive outcomes, with satisfactory performance across various models and techniques.

2. The performance of our feature-selected data has proven to be superior to that of supervised dimension reduction methods, demonstrating the effectiveness of our feature selection approach.

3. Among the base learners, the Random Forest model stood out as the best performer, showcasing its strength in handling the classification task.

4. The stacking model we implemented has been a success, surpassing the performance of its base learners and demonstrating promising results. Its performance is also comparable to the results reported in the paper, indicating its effectiveness in predicting the target variable.

5. Our performance is comparable to the paper's results for several reasons:
    a. By achieving such performance with classical models rather than neural networks, we have shown that our approach is highly effective.
    b. Unlike the paper, we did not oversample the data to match the test data ratio. This ensures the generalizability and reliability of our approach, as we did not manipulate the data by using information from the test set for training.

6. It's worth noting that despite having a smaller ratio of the data, our results remain comparable to those obtained by other researchers who had access to larger datasets. This indicates the robustness and generalizability of our approach.