­­­­

# Final Group Project for the Course

Applied Machine Learning:

**ML Classifier for IoT Intrusion Detection**

Section 1: Marco Palermo

Section 2: Canio Alberto Maggio

Section 3: Odise Vila

Section 4: Rabia Shah

Section 5: Delvin Monzon

Wordcount: 4059

January 2020

**Abstract**

This research deals with a Classification Problem, the famous supervised learning task of categorizing new and unseen objects into the desired classes. In particular we focused on building a classification model for detecting suspicious network traffic, in order to spot and prevent security breaches and other potential threats to a computer.

The goal of this work was to develop a very powerful classifier that could predict the difference between ‘normal’ and ‘intrusive’ traffic, with a high precision on the testing dataset.

To achieve this, first we studied the statistical and visual characteristics of the data, in order to better understand the problem we were facing. We also did some feature engineering work combined with feature selection, to reduce the dimensionality of our dataset while preserving as much information as possible.

Then, we explored and compared several different classification models analysing their trade-offs among accuracy, speed and interpretability. We selected the Multilayer Perceptron (MLP) as the most effective model on this specific dataset and proceeded to fine tune the model and find its optimal parameters using cross-validation techniques such as random search and grid search.

Finally we calculated and plotted several performance measures, in order to fully understand really how well the model performs.

Throughout the whole process we made sure to review the individual programming sections multiple times to reassess and improve each step of our work in combination with the others. This resulted in a very exciting and stimulating iterative optimization process, that allowed us to deeply understand how each component influences the subsequent pieces of code, often in a drastic manner.

**Keywords**

Machine Learning, Classification, Intrusion Detection, Dataset Visualization, Descriptive Statistics, Feature Engineering, Feature Selection, Algorithms Comparison, MLP, Hyperparameters Tuning, Performance Metrics.

**Index**

|  |  |
| --- | --- |
| [**Planning**](#r6p9js3nlzoa) | **Page 3** |
| [**1st Section - Pre-processing (Marco Palermo)**](#uz21wnkk9v2p) | **Page 3** |
| [**Dataset Cleaning**](#a84qy7d3rnbb) | **Page 3** |
| [**Descriptive Statistics**](#n3jdk94954hq) | **Page 4** |
| [**Dataset Visualization**](#ybhnz2o87zuf) | **Page 5** |
| [**Features Re-scaling**](#o97fq3loic74) | **Page 6** |
| [**2nd Section - Selecting Features (Canio Alberto Maggio)**](#ud9sgw44i309) | **Page 7** |
| [**Features Engineering**](#g4ge2rmfeuin) | **Page 7** |
| [**Features Selection**](#r3eas9qkvv2f) | **Page 8** |
| [**3rd Section - Exploring and Selecting ML Algorithms (Odise Vila)**](#fox55dufz79s) | **Page 10** |
| [**Training Accuracy**](#h0ye1l32agw1) | **Page 10** |
| [**Testing Accuracy**](#qzump2h5iah) | **Page 12** |
| [**4th Section - Refining Algorithms (Rabia Shah)**](#rl60jzla0kb4) | **Page 13** |
| [**Random Search**](#dcb0lr25eb85) | **Page 13** |
| [**Grid Search**](#qcnphdlhn5nc) | **Page 13** |
| [**Refining Process**](#rl60jzla0kb4) | **Page 13** |
| [**5th Section - Evaluating the Model and Analysing Results (Delvin Monzon)**](#lwvohdz7ef3) | **Page 14** |
| [**Evaluation Metrics**](#ehixna4ad70a) | **Page 14** |
| [**Benchmark Comparisons**](#tc2i32l06n6b) | **Page 17** |
| [**Future work**](#t3tafsehjyx) | **Page 18** |
| [**Resources**](#ej3k7wjv4210) | **Page 19** |

**Planning**

In our work, we aimed at maximizing the accuracy that we got on the testing data. To do this we decided to focus on feature construction, since we thought this method would prove crucial to improve the performance of our classification models.

In the time limit that was at our disposal, we tried to explore and test as many algorithms as possible to find the most efficient model on this specific dataset. We decided to fine-tune the model so that it would be specific enough to achieve the highest accuracy possible with the testing data that we were provided with. At the same time, we also wanted our code to be general enough so that it could be easily adapted to any dataset, thus we tried to always use functions and other good coding practices to make the code as reusable and scalable as possible.

We agreed on three deadlines. One week after the project was assigned, we had to come up with a minimum viable product. Having the basics down for our final script would allow us to test and refine our code in an iterative way during the whole process. The next deadline was the end of December and its goal was to have finished the python code. Once the programming part was completed, we could all focus on the final step which consisted in writing the report before the official deadline of the project: January 19th.

Before starting to code, we looked into the class materials and the research papers provided by our professor. We also analysed more of the existing literature by looking for additional research papers and website sources. Understanding the state of the art of the current studies on the dataset gave us a benchmark to test our final accuracy and allowed us to get an idea of which were the best practices that would work well on this specific data frame.

For what concerns our model trade-offs, we sought to find a balance between accuracy, speed, and interpretability, but in the end we decided to opt for the Multilinear Perceptron (MLP) classifier, a very powerful black box model, that sacrifices interpretability to achieve a better accuracy.

In accordance to the project requirements, each major task was assigned to a different individual. Despite this division of work, it was really important for us to go back and forth between all the sections, reviewing and optimising the code as a group, so that we could build a robust and efficient code where every single part interacts with the others in a synergistic manner.

**1st Section - Pre-processing (Marco Palermo)**

**Dataset Cleaning -** In this part, our main concern was trying to reduce the size of the dataset as much as possible, by removing all the redundant and irrelevant features. This process can be dangerous as the last thing we wanted was deleting a feature that could actually be useful for the classification, thus losing information right from the beginning. So, before making any changes to the data frame we created a backup copy of the dataset using the function df.copy(). The initial dataset presented no missing values (NAs) and had already been balanced between the two target classes, so that it presented the same distribution for ‘normal’ and ‘intrusive’ network traffic.

*Description of the initial datasets*

|  |  |
| --- | --- |
| Number of Training Samples | 97044 |
| Number of Testing Samples | 40158 |
| Number of Features | 152 features +  1 ‘target’ feature |
| Number of Target Classes | 2 |
| Classes | Normal Traffic (target = 0)  Intrusive Traffic (target = 1) |
| Class Distribution | 50% Normal Traffic  50% Intrusive Traffic |

By analysing the dataset using df.describe(), we saw that some columns that were exactly the same appeared multiple times in the dataset. By deleting these ‘duplicate’ and redundant columns we reduced the size of the dataset from 153 columns to 140. We also noticed that some features only presented value zero for the whole column, since they had both mean and standard deviation equal to zero. The same thing happened for some columns where the whole column was composed by ones. Removing these irrelevant features brought the number of features from 140 down to 57 (56 predictors and 1 target).

Finally, we renamed feature ‘155’ containing the 2 class labels, to ‘target’.

**Descriptive Statistics -** We use some exploratory data analysis tools to get an initial idea of the shape and characteristics of the dataset.

*Diagonal Correlation Matrix on the dataset after cleaning.*

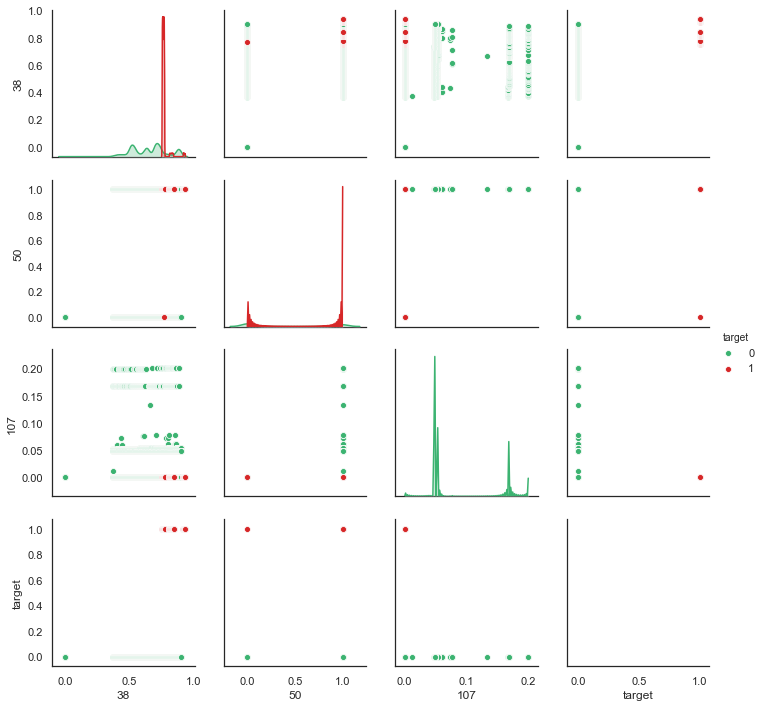
*Plot created using sns.diverging\_palette().*



We use the Diagonal Correlation Matrix to visualize the pairwise correlation coefficients between the variables in the dataset. The blue colours highlight a negative correlation between the features, while the red palette identifies a positive one. Finally the intensity of the colour shows how strong the correlation is (a darker blue tile is connected to a stronger negative correlation).

*Matrix showing the pairwise relationships between features 38, 50, 107 and target.*

*Plot created using sns.pairplot().*

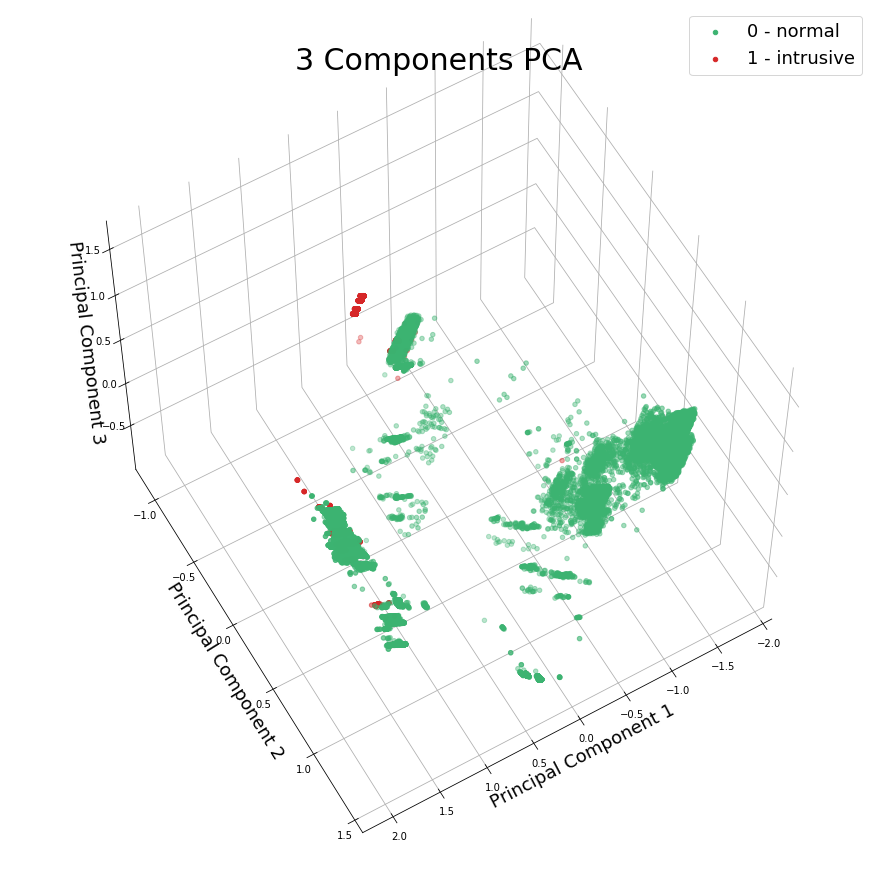
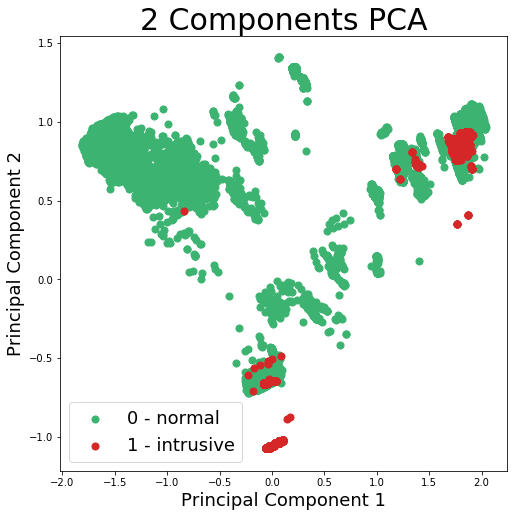


The pairwise relationships plot gives us a better idea of the statistical characteristics of the plotted features and their associations. The diagonal shows the distribution of the features, highlighting the distinction between the classes with different colours. The more separated the distribution of class 0 from the one of class 1 the more important the feature will be in predicting the target.

**Dataset Visualization -** In Python, Principal Component Analysis (PCA) is a simple but really powerful linear transformation technique. PCA is often used to emphasize variation and bring out strong trends in a data set. Here we use PCA to be able to visualize the data frame.

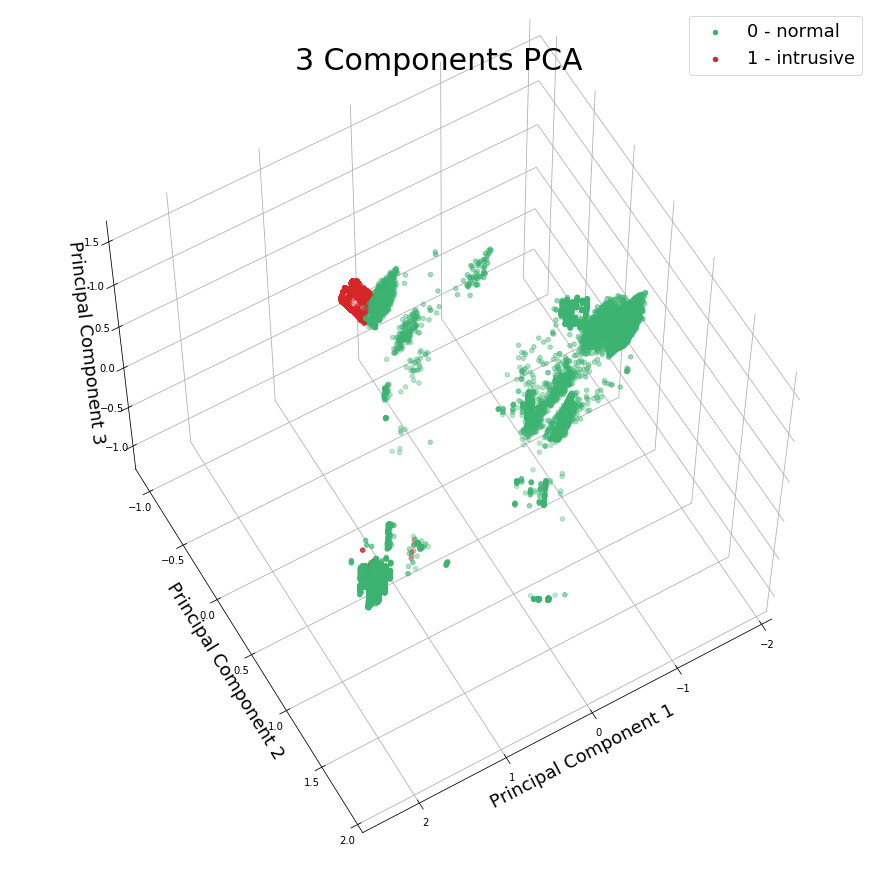
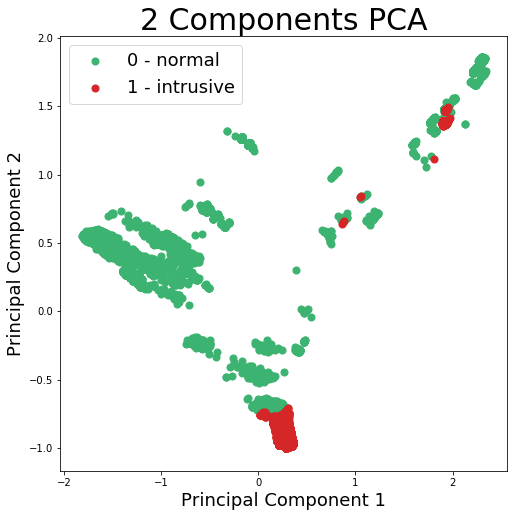
*Plot showing the 2 components and 3 components PCA on the training data.*

*Plots obtained using plot\_2D\_PCA(df) and plot\_3D\_PCA(df)*

****

*Plot showing the 2 components and 3 components PCA on the Testing data.*

*Plots obtained using plot\_2D\_PCA(df\_test) and plot\_3D\_PCA(df\_test)*

****

The 2 Components PCA plot shows that the clouds of points are not linearly separable in 2 dimensions, as the normal (target = 0) and intrusive (target = 1) classes look somewhat overlapping.

The 3 Components PCA, on the other hand shows a much better separation of the two classes, with the green ‘normal’ cluster occupying most of the plot, while the points belonging to the red ‘intrusive’ cluster are concentrated in the upper-left part of the graph.

**Features Re-scaling -** We set up several features re-scaling techniques that will be useful later on, after the feature engineering step.

Standardizing the range of the features in the dataset is a fundamental practice that prevents a single variable with a larger variance than the others, from dominating the classification. When all the variables are in the same range each feature contributes proportionately to the final distance.

**2nd Section - Selecting Features (Canio Alberto Maggio)**

**Feature engineering -** This phase consists in using the existing features to create new ones that might help in the classification process. In order to do so we utilize various dimensionality reduction techniques, to create condensed features that preserve as much information as possible about the data frame. These methods project the dataset onto a lower-dimensional features space while maintaining the class-discriminatory information in order to avoid overfitting and to reduce computational costs. The ones we have implemented are:

• Linear Discriminant Analysis (LDA), whose goal is to find a linear combination of features that characterizes or separates two or more classes of objects. This Turned out to be the most relevant feature construction technique, as the engineered feature ‘LDA1’ was picked by all the different feature selection methods we used.

• Principal Component Analysis (PCA), uses linear algebra to transform the dataset into a compressed form and finds the directions (principal components) that maximize the variance in a dataset. In our code we find the first 3 principal components, but they only explain 25.96% of the variance of the test data (14.21% for the first principal component, 7.39% for the second and 4.34% for the third).

• t-Distributed Stochastic Neighbour Embedding (t-SNE) calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space. It then tries to optimize these two similarity measures using a cost function. We discarded this feature construction method as it just took too long to process the dataset, even after the cleaning step.

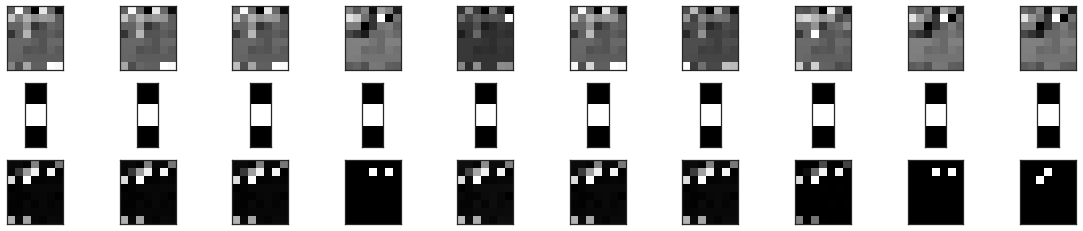
• Factor Analysis (FA) which analyses interrelationships between a large number of variables and to explain them in terms of their common underlying factors.

• The Autoencoder (AE), unsupervised artificial neural network that learns how to encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.

*From top to bottom: original features, AE compressed features and AE reconstructed features,*

*visualized as images in a black and white colour palette.*

*Image obtained using the function plt.imshow()*



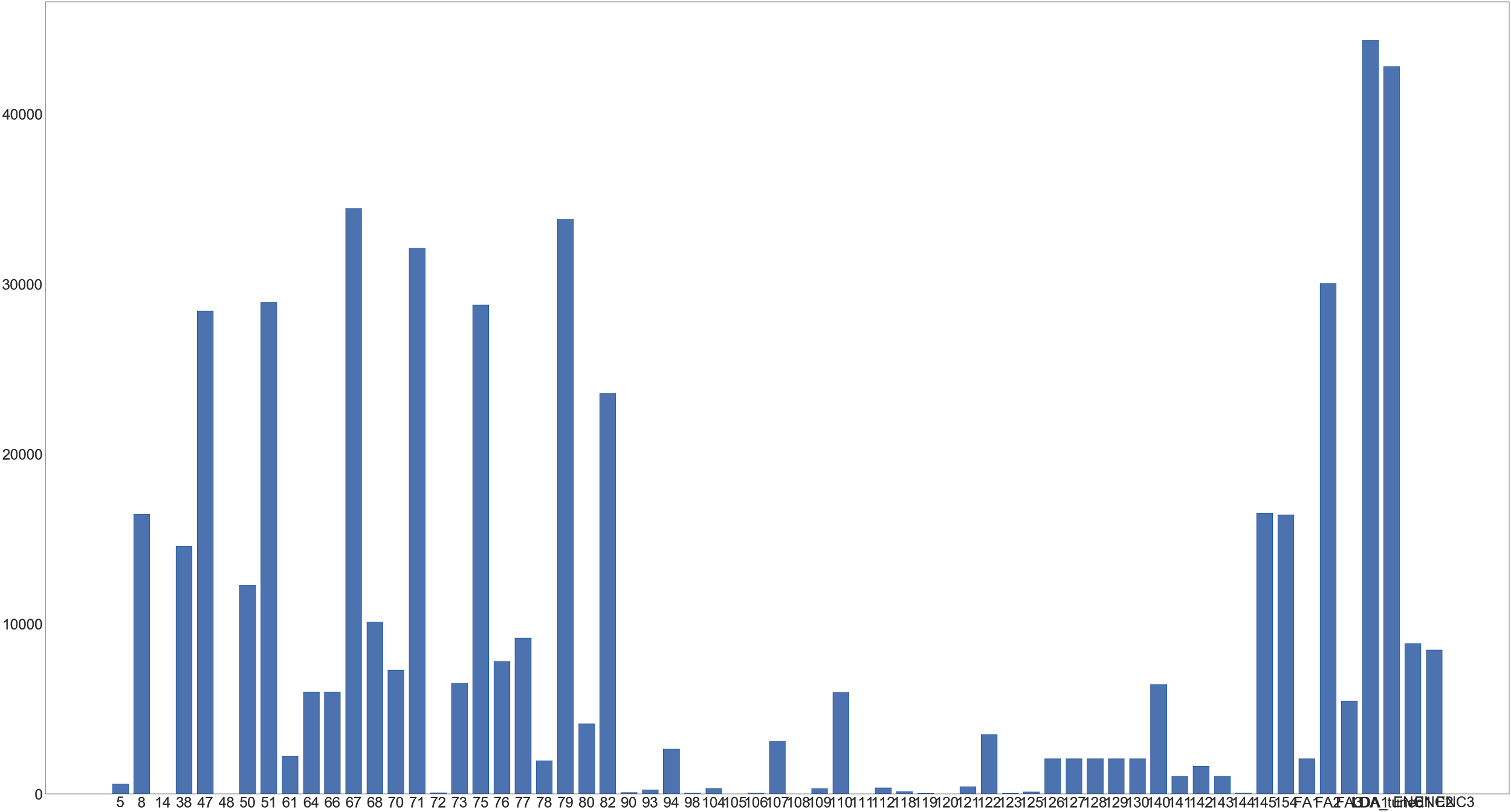
**Feature Selection -** We rescale all the features through a standardization so that they have the same range, in order to prevent one feature with a larger variance to be seen as disproportionately important. We now apply feature selection, a process that select the features in the data that contribute most to the prediction variable. This is a fundamental step because reducing the number of features might improve the performance of some classification algorithms.

The feature selection methods are typically presented in three classes based on how they combine the selection algorithm and the model building: the Filter picks up the intrinsic relevance of the features measured via univariate statistics, the Wrapper measures the usefulness of features based on the classifier performance, the Embedded, which is similar to the wrapper but uses an intrinsic model building metric during learning. We implement a technique per every different method and try to select 4 relevant features for use in model construction.

*Histogram with the Feature Importance for the Chi-Squared selection method*

*Selected features are: ENC3, LDA1, LDA\_tuned1, 67.*

*Plot created using plt.bar()*

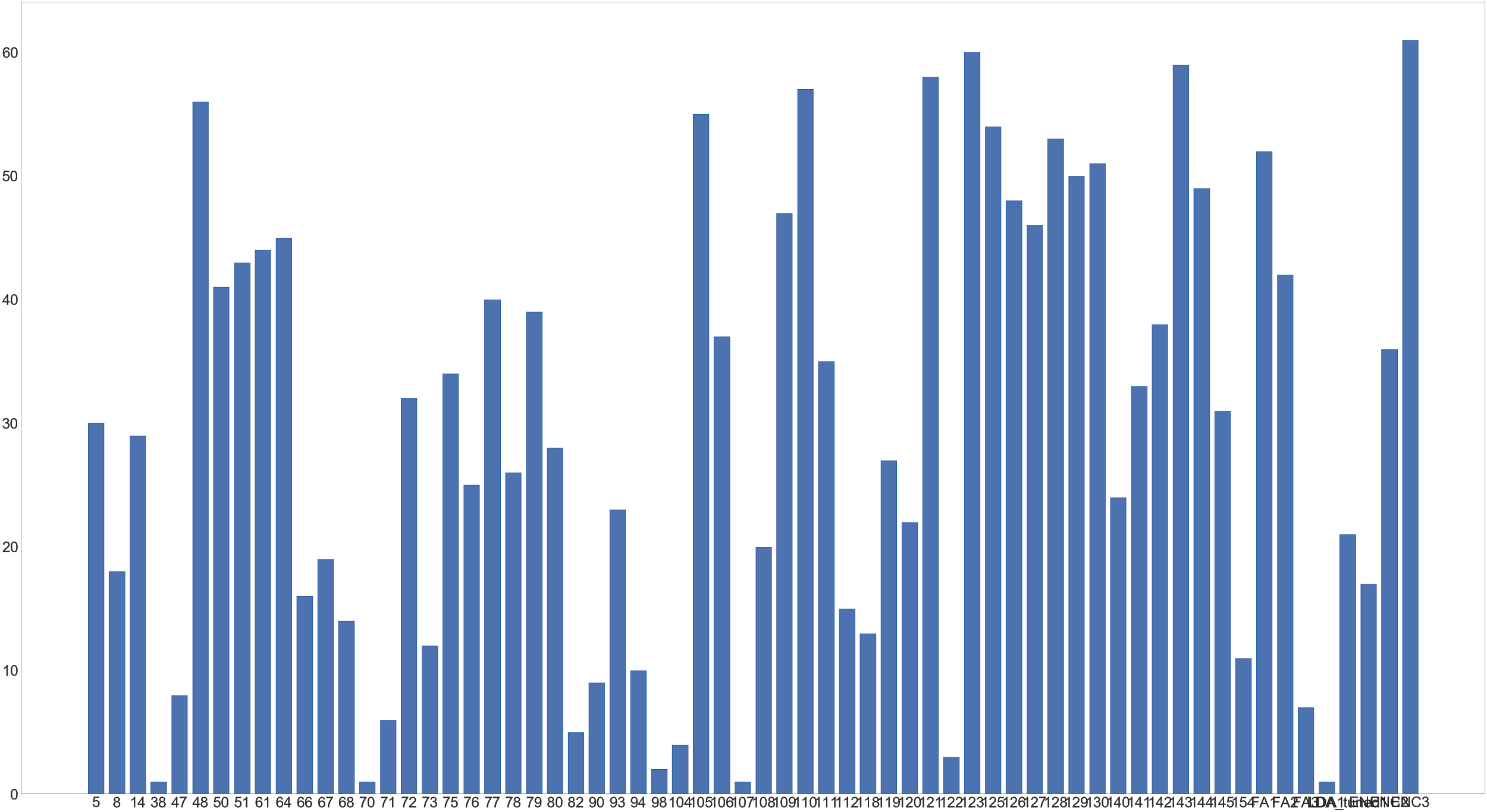


First we use chi-squared, a statistical hypothesis test that assumes (the null hypothesis) that the observed frequencies for a categorical variable match the expected frequencies for the categorical variable. It selects those features that have the strongest relationship with the output variable.

*Histogram with the Feature Importance for the RFE feature selection method*

*Selected features are: 38, 70, 107, LDA1.*

*Plot created using plt.bar()*

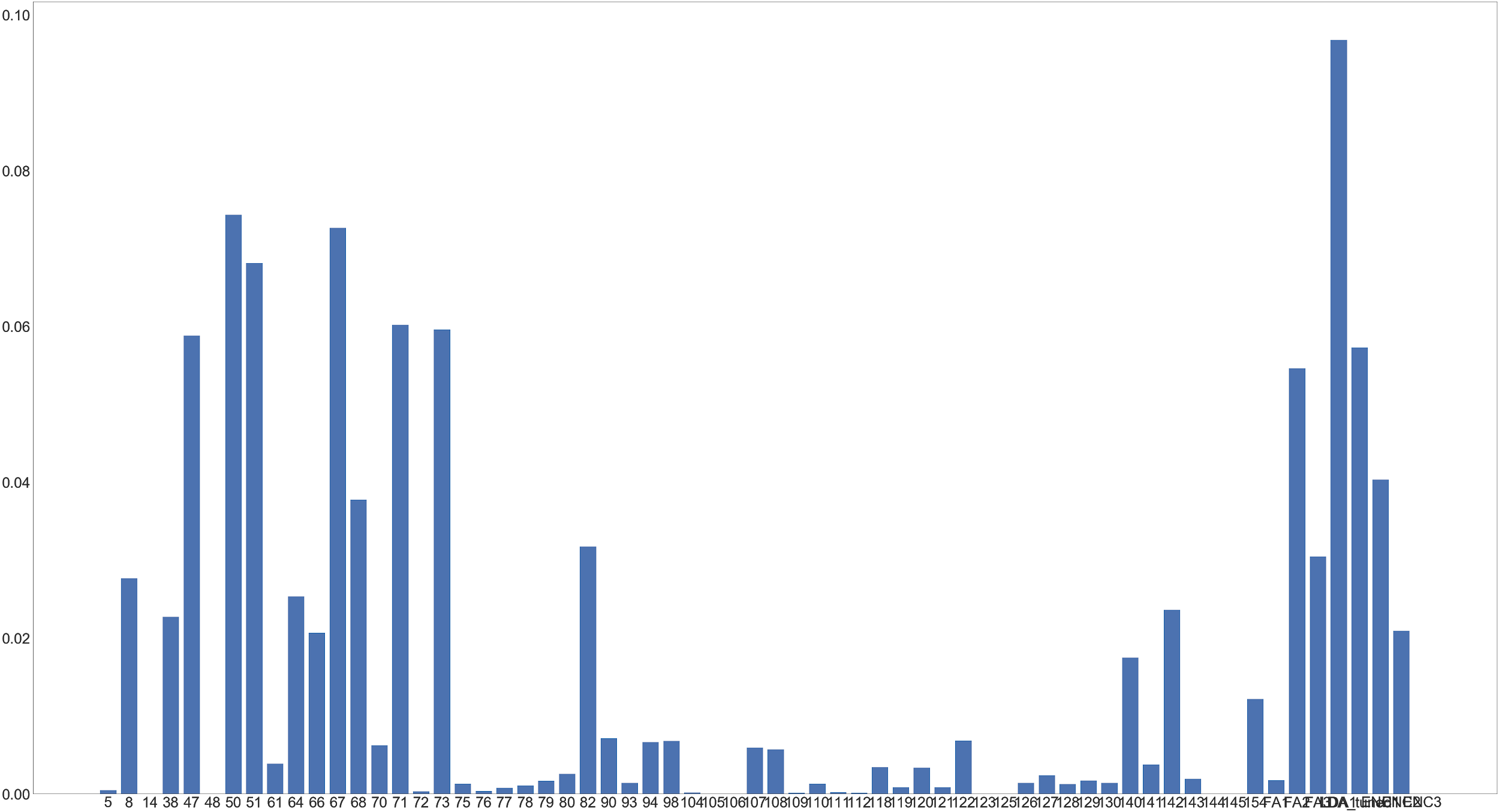


Next, we implement a feature extraction with the recursive feature elimination (RFE) method that works by recursively removing attributes and building a model on those attributes that remain. First, the estimator is trained on the initial set of features and their importance. Then, the least important features are pruned from the current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

*Histogram with the Feature Importance for Feature Extraction with Extra Trees*

*Selected features are: 51, 67, 50, LDA1.*

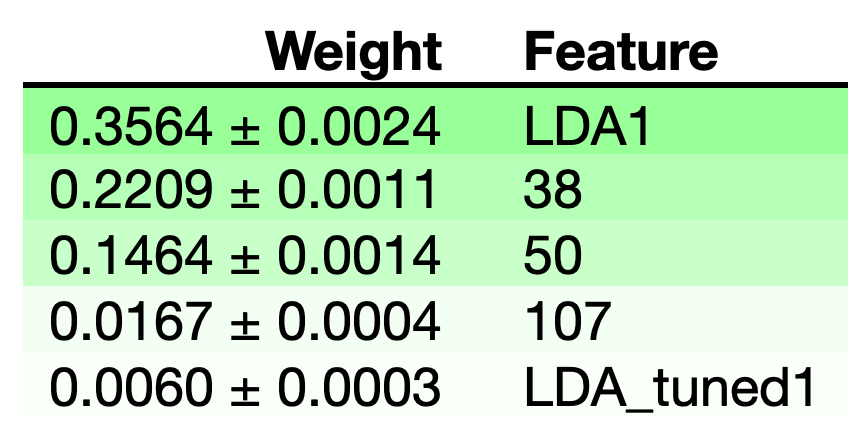
*Plot created using plt.bar()*



Finally, we apply a Feature Extraction with Extra Trees, a method that is used to estimate the feature importance by fitting randomised decision trees (extra-trees) on various sub-samples of the dataset and using averaging to improve the predictive accuracy and control over-fitting.

*The 5 features we selected for our final model, the MLP classifier,*

*with their respective Weight in the classifier.*



We selected the best features for our final model through a trial and error process, running the three feature selection methods listed above multiple times, testing the accuracy of the classifiers on different sets of features.

In the end we found that the optimal number of features for our classifier is k=5 and the best features to optimize the performance are: LDA, 38, 50, 107, LDA\_tuned1.These were obtained with a very “lucky draw” of the Random Feature Extraction method. (the features selected somewhat changed at every run because of some randomness in the feature engineering process).

**3rd Section - Exploring and Selecting ML Algorithms (Odise Vila)**

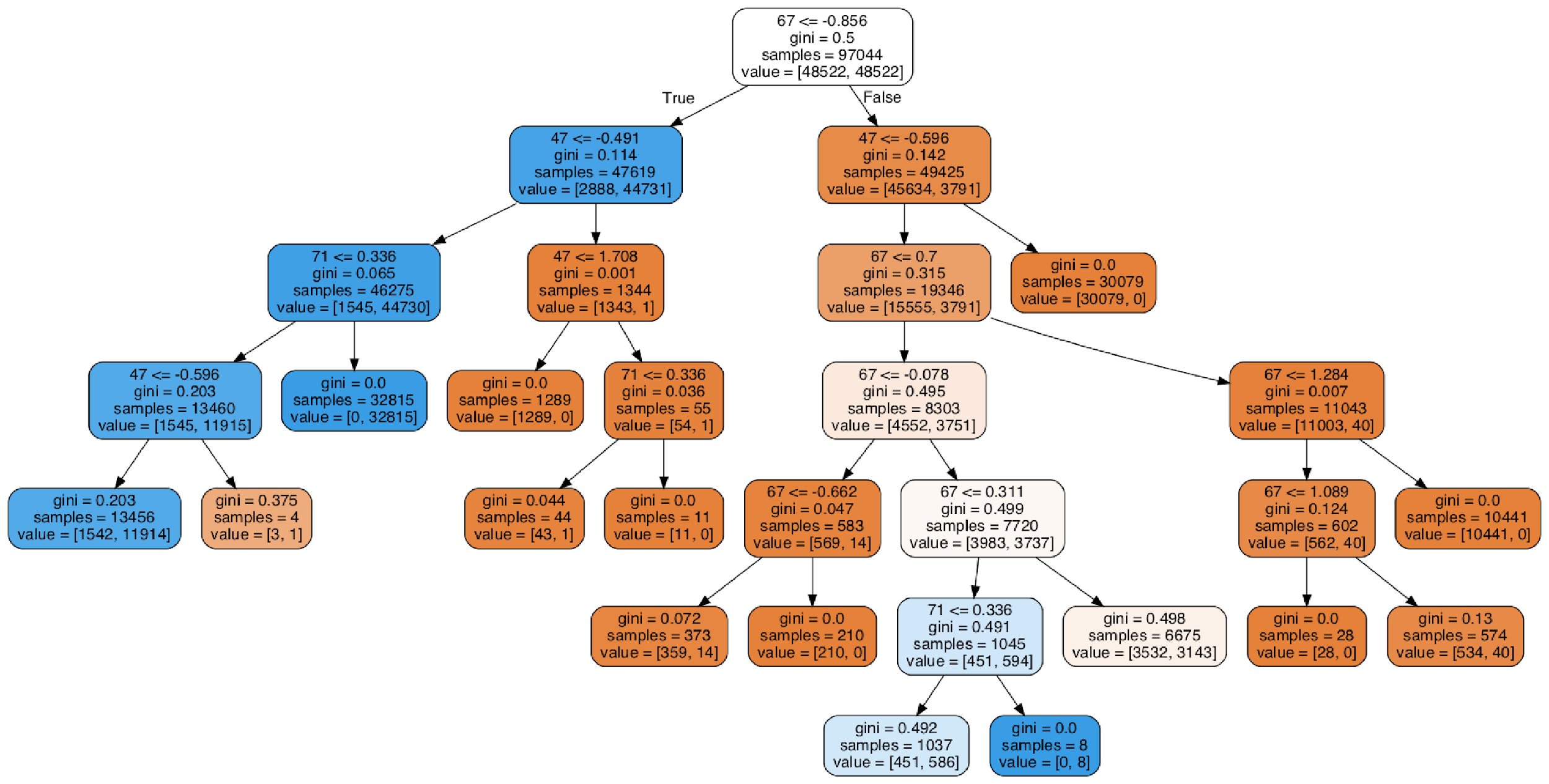
**Training Accuracy -** Prior to the selection of the candidate algorithms, the model trade-offs had to be laid out. After careful consideration as a group, it was decided that the focus would be placed on three main areas: speed, accuracy, and interpretability. With the model trade-offs established, we elected to have a balance between algorithms that we assumed would perform well under at least one of the above-mentioned areas. Therefore, the initial algorithms were:

Logistic Regression (LR), Linear Discriminant Analysis (LDA), Decision Tree (DT), K-Nearest Neighbours (KNN), GaussianNB (NB), Random Forest (RF), Extra Trees (ET), MultiLayer Perceptron (MLP),

Support Vector Machine (SVM).

*Decision Tree that was created with the features; 71, 51, 67, 47*

*Graph created using the function export\_graphviz()*



The next step in the selection process was to test the baseline accuracy of the models on the training data. The initial results showed promise as most algorithms made predictions between the ranges of 94% - 97% (see table below). The three algorithms that stood out in this initial phase were the Random Forest, the Decision Tree, and the Support Vector Machine. The Random Forest had the highest accuracy (98.49%), while the Decision Tree had the lowest (76.08%). The Support Vector Machine was ruled out as its computational time was too high with regard to the other models, in some cases taking longer than one hour to execute.

*Result of the “compare\_models()” function on the initial training data frame.*

|  |  |  |
| --- | --- | --- |
| Algorithm | Train Acc | Standard Deviation |
| LR | 0.947115 | 0.101375 |
| LDA | 0.964304 | 0.065782 |
| KNN | 0.977617 | 0.045288 |
| CART | 0.760810 | 0.371787 |
| NB | 0.956431 | 0.069650 |
| RF | 0.984903 | 0.041403 |
| EX | 0.975082 | 0.049481 |
| MLP | 0.956204 | 0.088015 |

**Testing Accuracy -** Having established the baseline accuracies for every selected algorithm, it was time to evaluate said models on the testing data. To determine their performance, it was deemed necessary to define a function, “class\_acc\_report()”. In a simplistic explanation, this function would train the models and test them against different data frames that were created after an array of feature selection methods had been applied, returning a report with the respective accuracies of the models. This proved to be constructive as it gave us insight into which models worked best with what data frames.

From the different set of combinations between model and data frame, it was visible that there existed several algorithms that could achieve accuracies of at least 98%. The data frame that was modified using the Chi-Squared feature selection method proved to have the most models with accuracies above 98% (see table below).

*Result of the “class\_acc\_report()” function on the data frame, reduced to 4 features using the Chi-Squared feature selection method (selected features are:* LDA\_1, LDA\_tuned1, 67, 79*).*

|  |  |
| --- | --- |
| Algorithm | Test Acc |
| LR | 0.984697 |
| LDA | 0.984790 |
| KNN | 0.984161 |
| NB | 0.984718 |
| MLP | 0.984615 |

Albeit the Chi-Squared feature selection model providing the data frame that would yield the highest number of algorithms with accuracies above 98%, it would not be this data frame that would output the highest accuracy possible. The data frame that would, consisted of features: '38', '50', '107', 'LDA1', and 'LDA\_tuned1'. These features were found from the Recursive Feature Elimination method. Said features applied to the Multilayer Perceptron proved to return the best accuracy achievable at this stage, that accuracy being 99.81%.

Thus, it became evident to us that the MLP classifier, in combination with the features mentioned above, showed the most promising for the upcoming fine-tuning phase. This, aided by the fact that more interpretable and faster models than the MLP classifier obtained lower accuracies, led to our final decision to proceed forward with the MLP classifier for the next stage.

**4th Section - Refining Algorithms (Rabia Shah)**

Tuning is referred to as the shaping of the model architecture using the available space within the model, therefore searching for the right parameters for optimisation. If parameters are not optimal, training time can be exponential, especially for high dimensional data, local minima may not be achieved, therefore precision and accuracy at an optimal point cannot be achieved. Tuning is essentially optimising the model for a higher precision and accuracy rate using hyperparameters. The two most widely used techniques are the Grid Search and Random Search.

**Random Search -** this hyperparameters tuning technique uses random combinations of the parameters to find the best result of accuracy for the built model. In general, it has been considered to produce better results in comparison to the Grid Search as random combinations are considered for every iteration, it is highly likely that the whole space of the model is reached at a quicker speed than that of the Grid Search. However, it tends to generate a higher variance and since the combinations are randomly generated, no logic is used behind the selection. Due to the randomness and omission of logic, this search works best under the assumption that not all hyperparameters are equally important and the chances of finding the optimal parameters are comparatively higher at a quicker speed.

**Grid Search** - This technique involves building a model for every combination of parameters and then evaluating the model. Whichever model presents the highest level of accuracy will be considered the best. The Grid Search is referred to as such, due to the pattern it follows where all the values are placed in the form of a matrix through which the values are iterated. As a result, the evaluation of the parameters with growing dimensionality is adversely affected by the exponential factor.

**Refining Process -** When tuning algorithms, its optimisation depends on the values chosen for the parameters. We can manually change these, create lists of values or choose default values and re-run the model. Other than the Grid and Random Search, we have run the Linear Discriminant Analysis and also tuned the MLP using the Random Search.

The hyperparameters found for the LDA were singular value decomposition (svd) solver and a tolerance of 6.0e-1. Other solvers, such as least squares (lsqr) and Eigenvalue were tested but the latter was not compatible with the data and out of the svd and lsqr, shrinkage was not supported by the former, yet it gave a better result and higher level of accuracy than the lsqr. The results of this tuning created a new tuned featured, LDA\_tuned1, which was then incorporated with the model and produced better results and a higher level of accuracy on the testing data.

For the Random Search, the best accuracy of 0.9985% had been obtained tuning the parameters of the MLP to the following values: {activation: ‘relu’, alpha: 0.0001, batch\_size: auto, beta\_1:0.9, beta\_2:0.999, early\_stopping: False, epsilon:1e-08, hidden\_layer\_sizes:(100), learning\_rate:’constant’, learning\_rate\_init:0.01, max\_iter:250, momentum:0.1, n\_iter\_no\_change:10, nesterovs\_momentum: True, power\_t:0.5, random\_state: None, shuffle: True, solver:Adam, tol:0.0001, validation\_fraction:0.0, warm\_start: False}

After having changed the values of the parameters, the best accuracy on the test data had been obtained consecutively within a 4 decimal place threshold, the changes being down to a random factor which has also been reduced by setting the seed. Therefore, can conclude the hyperparameters for the tuning as per these values.

A range of activation functions had been used including identity, logistic, tanh, ReLu- the former being one which returns the value as per the argument and the latter producing non-linear results as per the multiple node inputs. Different learning rates, alpha values and validation fractions had also been used, however, they did not obtain optimal results.

An adaptive learning rate had also been used- which is an ideal approach to the gradient descent where the vector is updated for each step during the descent for optimisation, however, the constant learning rate obtained a better accuracy.

An important aspect to continuously review is the level of dimensionality. The dimensions for this model have been reduced using various methods of feature selection in order to present and run a more efficient model and avoid the expense of exponentially running time. Tuning also created a new tuned feature which improved the accuracy results and could be used as a feature for the model. Therefore, a high dimensional space must be reduced to produce efficient and accurate results with a model of lower complexity

Another way of achieving a reduction in the layers of the network is by dropping out the layers. Generally a good value is between 0.5 and 0.8, which indicates a general dropout of 30% of the nodes in order to obtain a well-fitting model as opposed to an overfit due to the excess of layers. In the parameters, there are 100 hidden layers used to run the tuning models which gives enough space to drop any layers accordingly whilst maintaining optimal results.

**5th Section - Evaluating the Model and Analysing Results (Delvin Monzon)**

**Evaluation Metrics -** For all model training and testing we used a MacBook Pro with 2.3GHZ dual core i5 CPU, 8gb RAM running on macOS Catalina (10.15.2). With regards to the evaluation metrics, we opted to use Detection rate (DR), false alarm rate (FAR), detection accuracy (Acc), Matthews correlation coefficient (Mcc) to measure the quality of binary classification, time to build the model (TTB), time to test model (TTM). We also calculated the area under the ROC curve (AUC), to discriminate between positive and negative classes and the harmonic mean between precision and DR (F1). In turn we believed that these metrics would strengthen our classification and model evaluation. Ultimately, we were aiming for high Acc, DR, Mcc, AUC and F1 whilst maintaining a low FAR, FNR, TTB and TTM. The majority of these metrics were used in papers that used DFES methods and/or other methods using a reduced Aegean Wi-Fi Intrusion Dataset (AWID). As a result, a more credible comparison of our chosen model with benchmarks can be conducted.

After choosing our best 4 features via RFE, training and testing, it appeared that MLP was consistent with producing some of the best results in each evaluation metric. Namely, Acc (99.77%), MCC (99.54%), AUC (99.91%). We also believe that FNR is one of the most important metrics due to the possibility of an attack going undetected which could have detrimental effects, thus being a bigger factor to consider when choosing our algorithm; MLP produces the second lowest score of 0.06%. At this point, LDA was also a contender due to its even lower FNR and TTB. However, the deciding factor was the Acc between MLP and LDA, 99.77% and 95.76% respectively.

*Evaluation Metrics for the models we selected*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Classifier*** | ***Acc*** | ***DR*** | ***TNR*** | ***FAR*** | ***FNR*** | ***F1*** | ***MCC*** | ***AUC*** | ***TTB(s)*** | ***TTM(s)*** |
| **MLP** | 0.9977 | 0.9977 | 0.996 | 0.0039 | 0.00064 | 0.9977 | 0.9954 | 0.9991 | 12.8229 | 0.0328 |
| DT | 0.9336 | 0.8508 | 0.9976 | 0.0024 | 0.1492 | 0.9238 | 0.8577 | 0.9242 | 0.2202 | 0.0018 |
| RF | 0.9051 | 0.7697 | 0.9981 | 0.0019 | 0.2302 | 0.8824 | 0.7887 | 0.9984 | 0.517 | 0.0207 |
| ET | 0.9139 | 0.8006 | 0.9958 | 0.0042 | 0.1994 | 0.8972 | 0.8119 | 0.9953 | 0.1706 | 0.0226 |
| LR | 0.9686 | 0.9945 | 0.9399 | 0.06 | 0.0054 | 0.9672 | 0.9358 | 0.9585 | 0.1778 | 0.0008 |
| Gaussian | 0.25 | 0 | 1 | 0 | 1 | 0.3333 | 0 | 0.4971 | 0.0311 | 0.0035 |
| SVM | 0.9006 | 0.8409 | 0.9507 | 0.0493 | 0.1591 | 0.8955 | 0.7964 | 0.9597 | 26.808 | 0.7834 |
| LDA | 0.9576 | 0.9999 | 0.9075 | 0.0925 | 4.98E-05 | 0.9536 | 0.9113 | 0.976 | 0.08815 | 0.0088 |
| **Red = ranking 1** | **Green = ranking 2** | |  |  |  |  |  |  |  |  |

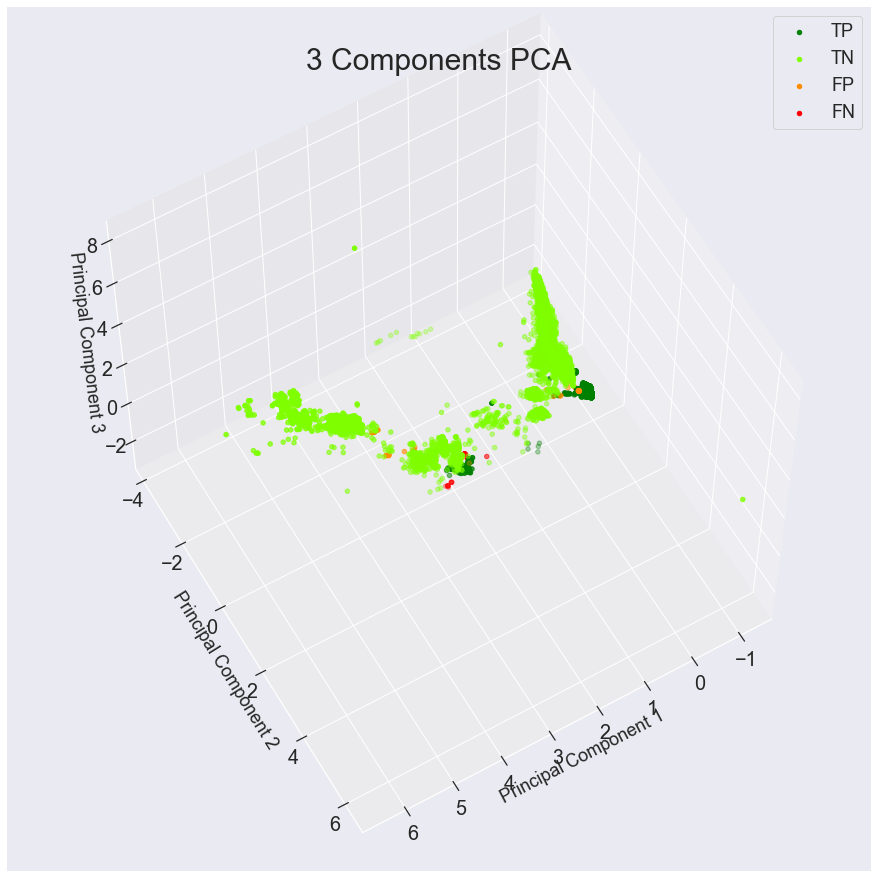
A strength of MLP is that random training iterations may result in better classification, unfortunately this would increase the total training time which seems to be an already expensive computational model. Also, the user has to set the number of hidden neurons initially, the disadvantage is that a value too low can lead to underfitting and a value too high may result in overfitting. Additionally, the use of a sigmoid function is required for classification when using MLP. This could lead to long-term information being corrupted and causing vanishing gradients – making the network harder to train.

Despite MLP having one of the highest TTB (12.8229s), we decided that due to its excellent performance in several other metrics, as mentioned previously, we would select this to algorithm as our candidate. Once MLP has been tuned, the table above shows that the tuned version improves upon every evaluation metric we used. Most notable is the reduction of around 41% in TTB from 12.8229s to 7.6272s, which increases favourability to be used in real time IoT detection scenarios. Additionally, as seen in the confusion matrices, FNR has also decreased by 46%, illustrating less attacks going undetected thus less chance of intrusion.

|  |  |
| --- | --- |
| *Confusion Matrix of the un-tuned MLP* | *Confusion Matrix of the tuned MLP* |
| A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| *ROC Curve for of the un-tuned MLP* | *ROC Curve of the tuned MLP* |
|  |  |

*3 Components PCA Plot showing the result of the Classification on the test set, for the tuned MLP*

*(the different colours are used to highlight which points are TP, TN, FP and FN)*



*Evaluation Metrics for the tuned and untuned MLP*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Classifier*** | ***Acc*** | ***DR*** | ***TNR*** | ***FAR*** | ***FNR*** | ***F1*** | ***MCC*** | ***AUC*** | ***TTB(s)*** | ***TTM(s)*** |
| **MLP (tuned)** | 0.9984 | 0.9997 | 0.9972 | 0.0028 | 0.0003 | 0.9984 | 0.9968 | 0.9994 | 7.6272 | 0.0286 |
| MLP | 0.9977 | 0.9977 | 0.996 | 0.0039 | 0.0006 | 0.9977 | 0.9954 | 0.9991 | 12.8229 | 0.0328 |

**Benchmark Comparisons -** Comparison of our tuned and untuned MLP algorithm with benchmark depicts promising progress: the Acc of our MLP is only beaten by SVM variants from Parker et al and Aminanto et al 2018 as seen in (4). The same models also produce a lower FAR of 0.01% and 0.012% respectively and also higher F1 and Mcc values. However, the use of feature extraction with RFE has allowed us to utilise only 4 features, substantially less than the DFES method used by Parker et al 2018. As a result, despite the slight loss in the previously mentioned values, the TTB for our MLP model is significantly quicker at 7.6272s compared to their 12073s and would not fare as well in an environment where speed may be paramount. Lastly, our tuned MLP model performed the best DR of 99.97%.

*Performance comparison between our MLP model and algorithms from other papers*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Classifier*** | ***Acc (%)*** | ***DR (%)*** | ***FAR (%)*** | ***F1 (%)*** | ***MCC (%)*** | ***TTB (s)*** |
| **MLP (tuned)** | 99.84 | 99.97 | 0.28 | 99.84 | 99.68 | 7.6272 |
| MLP | 99.77 | 99.77 | 0.39 | 99.77 | 99.54 | 12.8229 |
| Parker et al 2019 (DETEReD RBFC) | 98.04 | 99.07 | 2.96 | 98.01 | 96.09 | 603.33 |
| Parker et al 2019 (DFES SVM) | 99.97 | 99.92 | 0.01 | 99.94 | 99.92 | 12073 |
| Aminanto et al 2018 | 99.97 | 99.92 | 0.01 | 99.94 | 99.92 | 12073 |
| Aminanto et al 2017 | 97.6 | 85 | 2.36 | NF | NF | 350 |
| Kolias et al 2015 | 96.26 | 96.3 | 43.68 | 94.8 | NF | 568.92 |
| **Red = ranking 1** | **Green = ranking 2** | | **\*NF = Not Found\*** | |  |  |

**Future Work**

**What are the strengths and weaknesses of your work?**We believe that the variety of candidate algorithms and evaluation metrics allowed us to thoroughly dissect the performance of each classifierto come to an informed decision about what we regard top qualities from an IoT intrusion detector. One weakness would be the lack of resources and extra time to optimally tune our model.

**What do you think are the next steps to take?**Regarding future work, we hope to test the capabilities and adaptabilities of our algorithm on varying attacks that may possibly appear in an IoT network environment. Real world scenarios and architectures would be the ultimate test of performance for our algorithm and other benchmarked models.

Additionally, exploring other tuning algorithms and incorporating other tuned features accordingly would also be a path; it was not within the scope of our project due to the endless possibilities of combinations.

**What other questions do your results raise?**The speed of our MLP algorithm comparable to benchmarked algorithms seems to demonstrate a stark contrast; from our results perhaps more emphasis needs to be invested in finding an optimal (near perfect) feature selection, tuning algorithms and tuned features to get the most out of the classifiers widely used.

**Do you think certain paths seem to be more promising than others?**

We noticed that reducing the number of features both reduces the computational time and increases the accuracy of most algorithms. More research could be carried out in order to find the optimal number and specific set of features to use in the classification.

**Resources**

• J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” *The Mathematical Intelligencer*, vol. 27, no. 2, 2005.

• Y. L. Pavlov, “Random Forests,” 2000.

• M. Sugiyama, “Statistical Machine Learning,” *Introduction to Statistical Machine Learning*, 2016.

• M. Jändel, “A neural support vector machine,” *Neural Networks*, vol. 23, no. 5, 2010.

• J. Tolles and W. J. Meurer, “Logistic Regression,” *Jama*, vol. 316, no. 5, Feb. 2016.

• LR Parker, PD Yoo, TA Asyhari, L Chermak, Y Jhi, and K Taha DEMISe: Interpretable Deep Extraction and Mutual Information Selection Techniques for IoT Intrusion Detection, The 14th ACM International Conference on Availability, Reliability and Security (ARES), 26-29 Aug. 2019, U.K.

• ME Aminanto, R Choi, HC Tanuwidjaja, PD Yoo and K Kim (2018) Deep abstraction and weighted feature selection for Wi-Fi impersonation detection, IEEE Transactions on Information Forensics and Security, 13(3), 621–636.

• ME Aminanto and K Kim (2017) Detecting impersonation attack in WiFi networks using deep learning approach, Information Security Applications 17th International Workshop. Jeju Island, South Korea, 25-27 August 2016, 136–147.

• C Kolias, G Kambourakis, A Stavrou and S Gritzalis (2016a) Intrusion detection in 802.11 networks : empirical evaluation of threats and a public dataset, IEEE Communication Surveys and Tutorials, 18(1), 184–208.

• P. Yoo. (2019). Regression-based ML Algorithms (LR and NN) [Pdf]. Available:<https://moodle.bbk.ac.uk/mod/folder/view.php?id=637295>.

• C-F Wang, “The Vanishing gradient Problem”, Towardsdatascience.com.<https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>

(accessed Jan. 16th, 2020).

• N.Gillian, “MLP”, nickgillian.com.

<https://www.nickgillian.com/wiki/pmwiki.php/GRT/MLP> (date accessed Jan 17th, 2020).