Student ID 13166549 Siddharth Sengupta Preprocessing

Student ID 13405463 Alex Bates Feature Selection

Student ID 12819393 Alex Gorton – Algorithm Comparison

Student ID 13157188 Konstantin Orlovskiy – Algorithm Tuning

Student ID 13159064 Scott Tasker – Model Evaluation

Applied Machine Learning:

*Birkbeck University: Professor Paul Yoo*

Applied Machine Learning Group Coursework:

IoT Intrusion Detection Competition using Machine Learning

**Planning**

**Project initiation**

We formed a group and held our initial meeting on the 10th of November 2019. At this meeting we made a series of decisions for the project:

1. Assignment of sections to group members
2. Workflow methodology
3. Project principles
4. Agreed a work schedule
5. Established communication
6. Programming conventions
7. Collectively this was our first experience using machine learning, so rather than prioritizing experience, we discussed the aspects and requirements of each section. We then individually volunteered for sections based on personal preferences.
8. We identified that each of the data handling steps are sequential, requiring input from the preceding stage. For example, the input of the feature selection step is the pre-processed data. This constraint necessitates a waterfall-like workflow.
9. Due to time constraints and our relative inexperience, we determined that our project should favor speed and simplicity over complexity, for example using available libraries like scikit-learn rather than attempting to build our own tools from scratch.
10. Factoring 10 weeks to complete the project, we agreed to attempt to complete 2 full iterations prior to the Christmas break. The planned schedule is depicted in figure 1.
11. To enable distributed development of the project, we set up a WhatsApp group for communication and created a Github repository to store project files.
12. Further to point 5, we agreed that all programming should be performed in the Python programming language, and each section implemented as a function with documented inputs and outputs to allow easy use by the other group members.



Figure 1. Planned work schedule.

**Project review**

We held two review sessions, the first on the 12/12/2019 and a final review session on the 09/01/2020.

In the first session, we reported the conclusions of the first two iterations, and identified critical areas for improvement:

* Trying different scaling methods such as MinMax, StandardScaler
* Exploring dimensionality reduction techniques for feature selection like PCA
* Reviewing the algorithm selection, adding in K-nearest neighbours and Naïve Bayes Classifier
* Expanding the tuning parameters search size.
* Exploring further evaluation parameters such as F1 score, False Alarm rate, Michaels Correlation Coefficient

In the final review session, we discussed the conclusions of the project and assigned responsibilities for the planning section and the future of the report and code clean up.

**Summary**

* We delivered the project closely to the planned schedule, with several high-performance models developed by the Christmas break, allowing further research than anticipated.
* The WhatsApp group and Github repository were highly effective tools for sharing knowledge.
* By undertaking this project, we have all gained an insight into the application of machine learning algorithms to a real-world application.

**Preprocessing**

Any real-world data, in principle, is generally unclean, incomplete, unscaled, and overall messy. The pre-processing stage is essential to the machine learning pipeline, in order to prepare the data before further analysis and model building.

A pre-processing generally involves some data transformation, filling missing observations with suitable values, scaling various features to a standard scale, among other things.

The following operations were performed on the raw data set during this phase –

1. Metadata transformation – The raw data set provided to us has got column numbers instead of the actual column names. Given the number of features, at later stages, it would be next to impossible to identify the features. To overcome that, we have assigned the actual column names to the features instead of the numbers. The list of attributes is available Aegean data set web page. In the data set provided to us, the frame.time\_epoch and frame.time\_relative are excluded, so the feature names have been assigned accordingly.
2. Data cleaning – Next we checked the data for missing values. For each feature, we checked if there are any missing observations, and filled them with the median of that feature. The reason for choosing the median, instead of mean is that, in case the overall mean of a feature is very large, it might introduce unnecessary bias in the data, while a median would just assign the most common observation, which should be fair enough.
3. Data transformation – One of the crucial steps in this phase is to ensure that all features are numeric, since ML models generally work with numbers, and Python, unlike R, cannot implicitly handle the categorical features. To do that, first we identified the categorical features, and then used LabelEncoder to assign numbers to the observations. Then, using OneHotEncoder, we transformed those observations into dummy variables. This gives a binary variable for each of the categories.
4. Standardization – The last step we performed on this phase is to standardize the data. The features in any data set are likely to have very different variance with respect to each other, and feeding them to an ML model without scaling them might unfairly tip it in favour of few specific features with greater variance.

StandardScaler and MinMaxScaler are two of the most popular scalers available. However, both these scalers are sensitive to outliers. In this data set, we did not handle the outliers explicitly because we do not possess enough domain knowledge to determine whether a particular outlier is a human error, or a genuinely useful observation. Also, since these are all signal data, and we trying to detect unusual activities through this model, some of the outliers might contain the most useful observation. Therefore, we did not touch the outliers. However, while scaling, we need to keep the outliers in mind as well. For that purpose, we used a different scaler called the RobustScaler. Unlike StandardScaler, which removes the mean and scales to unit variance, RobustScaler removes the median and scales according to the quantile range. Therefore, it is more robust to outliers than any of the other scalers.

A couple of things to keep in mind here,

* The data set provided to us is quite clean, as we couldn’t find any missing values in any of the features. However, in our code, we still handled the missing observation condition as a contingency.
* There were no categorical variables either. There were binary features, but those are most likely signal values. Again, we handled the conversion of categorical to dummy variables, just in case.

**Feature Selection**

There are various techniques used for selecting features for machine learning models. ‘Filter’ methods select features based on their intrinsic properties. ‘Wrapper’ methods measure the worth of a feature using a classifier performance. Embedded methods are like wrapper methods but where the feature selection is embedded with the learning algorithm. Sometimes, just domain knowledge can be used to pick relevant features [1].

The AWID dataset contains 152 features to consider. Initially, features with zero variance were removed, as they can have no predictive power on the target. This left 78 features for consideration.

Due to the way the pipeline was organised, with different individuals working remotely on different stages, embedded feature selection methods were deemed to be too difficult to implement. Instead, various wrapper methods were investigated to consolidate the features to a reasonable number for the algorithmic stage from the 78 candidates.

One common technique from the academic literature is the use of an auto-encoder to reduce dimensionality in the dataset and generate new features [1]. This technique was not considered initially due to its complexity and the time constraints on the project, but could be investigated in future. Simpler methods, with greater interpretability, were first considered instead.

One key question is to decide how many features to select to train the model. Aminanto et. al [2] found, when training an artificial neural network on the AWID dataset, that there was little improvement in the F1 score from increasing the number of features beyond 10. In light of this, 10 features were selected in the first iteration.

On this first iteration, recursive feature elimination (‘RFE’) with a logistic regression classifier was used as the primary feature selection method. This gave a satisfactory result on several algorithms. The same method with just 5 features gave worse performance, suggesting that the model is not over-fitting due to the number of features. Using a different classifier with RFE resulted in broadly similar features being selected.

The first alternative to RFE considered was to use a randomised decision tree classifier and pick the top 10 features by feature importance. This approach yielded a very different set of features to RFE with only two common features between the approaches. Furthermore, the top two features by feature importance in the decision tree approach were not selected by any of the RFE algorithms tested. The decision tree approach yielded good results when used with a KNN algorithm in the learning phase. However, this set of features did not generally do as well in the learning phase as the features selected through RFE.

The final method considered was Principal Component Analysis. This method has the advantage of specifically capturing the variance in the *entire* dataset. However, by reducing dimensionality and combining features, some interpretability is lost. In our initial analysis (on the dataset using the Robust Scaler approach) the first 5 principal components explained over 98% of the variance in the dataset. On this basis, the top 5 components were selected for the learning phase but the results were not very encouraging and the method was discarded.

The best model evaluated in the project used RFE with 10 features. The final feature selection process can be summarised in the following flow chart:

Filter out zero variance features

152 AWID features

72 features with variance

10 features for learning

Recursive feature elimination with logistic classifier

The 10 features selected in this method were:

* radiotap.datarate
* wlan.fc.retry
* wlan.fc.pwrmgt
* wlan.fc.protected
* wlan\_mgt.fixed.capabilities.preamble
* wlan\_mgt.fixed.capabilities.short\_slot\_time
* wlan\_mgt.fixed.timestamp
* wlan\_mgt.fixed.auth\_seq
* wlan\_mgt.rsn.akms.type
* wlan.wep.key

The feature selection methods adopted have the advantage of simplicity and speed of computation (in both the feature selection and learning phase). A possible next step would be to increase the number of features and complexity of method and assess the impact this has during learning.

Please see the relevant appendix for relevant snippets of python code and a brief explanation of the different feature selection algorithms discussed above.

**Algorithm Comparison**

After selecting features, we moved into testing which algorithms might work better.

We reviewed a couple pieces of previous literature. In Kolias, they used the Weka framework to run a series of supervised classifiers over the dataset, J48 was top performer, but had a lot longer build time compared to Random Forest and OneR. In Aminatou, they use an ANN, C4.5, and an SVM, the SVM was the best performer, but takes alot longer to build. In Parker they chose a Radial Basis Function Classifier (RBFC) for the DEMISe Model and C4.5 for feature extraction/Logistic Regression for classification for DETEReD.

We decided to narrow it down to Supervised Learning as since the AWID dataset is all labelled and balanced, it seemed to be a better use case for supervised learning algorithms. We further narrowed it down to classification algorithms as we wanted the accuracy of it predicting a certain class.

I read through SciKit-Learn’s documentation and used their stratified dummy classifier to create a baseline to test the classifiers against. The stratified dummy classifier works by creating a random prediction by respecting the class distribution [1]. This allows us to see how much better the model is against just a random guess.

At first, we started with Logistic Regression, Decision Tree, Naïve Bayes, and SVM. these were the ones that had been tried in the earlier papers, so they sounded like a good base. Logistic Regression is actually a linear model for classification, instead of regression as it sounds. Decision Tree’s create predictive models by breaking down a dataset and using the data features as nodes to build the resulting tree with, this makes them good for categorisation tasks like the AWID data set. Naïve Bayes is a supervised classifier that works by applying Bayes theorem while assuming conditional independence among the predictors. Support vector Machines are supervised learning algorithm that can be used for classification. They work by “*If n is the number of input features, the SVM plots each feature value as a coordinate point in n-dimensional space. Subsequently, a classification process is executed by finding the hyperplane that distinguishes two classes*.”[2]

As we iterated, we added others to see how they would do and if there was an improvement with different feature sets/ pre-processing. Random forest we added as it is an ensemble classifier that improves on a basic Decision Tree, they work by running several decision trees at the same time during training and the output the average of the prediction, this also allows them to correct some of the error of overfitting present in regular Decision Tree’s. K Nearest Neighbour we added as they have a good reputation for classification issues, and it seemed worth it to test it out initially in SciKit-Learn.

Given the problem was classifying impersonation attacks in a balanced dataset we decided to go with these six algorithms and choose the best ones from among them. As we only had less than two months part-time, a deep learning model was considered to probably enough time that we would neglect the other models, so it was left for future improvements if possible.

We ran through the following algorithms with Kfold cross-validation to see which would have the best accuracy above the SciKit-Learn’s dummy classifier. We chose normal K-fold as the dataset was already balanced so stratified was not needed. As you can see from the below table, Decision Tree, Random Forest, and KNN had the highest accuracy above the baseline. I was expecting Random forest to be better rather than the same, but that will come out with tuning of hyperparameters and model evaluation.

.

|  |  |  |
| --- | --- | --- |
| Classifier | K-fold Accuracy | Percentage above Baseline |
| Dummy Classifier | 0.9582149933672758 | n/a |
| Logistic Regression | 0.9887371837807203 | %3.086987210972905 |
| K Nearest neighbour | 0.9998660415271265 | %4.1656628418178725 |
| Decision Tree | 0.9998969550208667 | %4.168625721309562 |
| Naive Bayesian | 0.9744757586686589 | %1.6686680152617404 |
| Support Vector Machine\_ classification | 0.9921170590962956 | %3.4171437148657287 |
| Random Forest | 0.9998969550208667 | %4.168625721309562 |

**Model Tuning**

**Deliverables from previous stages of the project.**

Preprocessing stage delivered two output datasets:

* 1. Robust Scaler preprocessed
  2. Standard Scaler preprocessed

Algorithmic feature selection stage output:

* 1. Decision Tree (10 features)
  2. Principal Component Analysis (5 features)
  3. Recursive Feature Elimination Logistic Classifier (5 features)
  4. Recursive Feature Elimination Logistic Classifier (10 features)

The algorithms selected for refining are:

* 1. Logistic Regression Classifier (LR)
  2. Decision Tree Classifier (DT)
  3. Random Forest Classifier (RF)
  4. Support Vector Classifier (SVC)
  5. K-Nearest Neighbour (KNN)
  6. Naive Bayes Classifier (NBC)

**Refining algorithms**

Considering the number of possible combinations of outputs from preprocessing, feature and algorithm selection stages the following approach was chosen for refining the algorithms:

* Build a skeleton for tuning all algorithms on one combination of preprocessed data and selected features.
* Use the skeleton to refine algorithms for all other combinations.

The intrusion detection is the classification problem therefore the classificatory models are used for predictions. For each algorithm the approach was the following:

* Default model: test model with default hyperparameters. The result accuracy and confusion matrix is used as a benchmark to evaluate the improvement after refining hyperparameters.
* Hyperparameter selection depending on the type of the algorythm based on the documentation [1], [2], [3], [4], [5], [6].
* Search for best estimator (a combination of hyperparameters) using randomized search cross-validation (RandimizedSearchCV) from sklearn library on the train set. The cross-validation approach leads to detecting the generalized model hyperparameters. RandimizedSearchCV was chosen instead of GridSearchCV because it is less computationally extensive and proved to provide relatively the same result [7]. Bayesian optimisation was not used due to the lack of time and resource considering the range of models to be tuned. For reproducibility of results set random\_state = 2019.
* Test refined model on the test set.
* Iteratively analyze the selected set of the hyperparameters and refine the arrays with the nearest values for each selected hyperparameter to achieve better accuracy with the next iteration. For models that are not improving run more extensive search of hyperparameters to achieve even slight improvement in accuracy.

Since different algorithms have different computational cost, initially LR/DT/RF have been run with default n\_iter=10 which means the RandimizedSearchCV is trying 10 combinations of hyperparameters randomly, while for KNN/NBC/SVC are computationally expensive and n\_iter=3 was chosen but when there was significant improvement from default model, the value of n\_iter was increased. In some cases n\_iter value was chosen to try all possible combinations which is technically grid search. This approach was used for only those models where there was a potential for improvement.

Results of refining each model were added to summary table which contains model accuracy, confusion matrix as well as calculated based on the confusion matrix attributes like specificity, sensitivity, precision, etc. which are required to evaluate the model performance and compare to others and choose the best model at next stage.

The best performing algorithm appeared to be Random Forest with RFE log 10 feature selection from Robust Scaler preprocessed dataset. Hyperparameters tuned are below. The tradeoff of improvement of accuracy is the computational cost/speed of the algorithm.

* ’n\_estimators’ determines the number of trees in the forest, the higher it is the better the data is learnt;
* 'max\_depth': shows the depth of each tree, the higher it is the more information is considered;
* ‘min\_samples\_leaf’: limits the minimum number of samples needed to be in the leaf node, the higher the parameter is, the more under fitted is the model;
* 'max\_features': determines the number of features to consider when splitting the node which improves the model accuracy.

The refined best model specifications and performance:

* accuracy: 0.9761442302903531
* parameters: {'bootstrap': True, 'class\_weight': None, 'criterion': 'gini', 'max\_depth': 50, 'max\_features': 'log2', 'max\_leaf\_nodes': None, 'min\_impurity\_decrease': 0.0, 'min\_impurity\_split': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'min\_weight\_fraction\_leaf': 0.0, 'n\_estimators': 10, 'n\_jobs': None, 'oob\_score': False, 'random\_state': None, 'verbose': 0, 'warm\_start': False}
* Confusion matrix: [ [19124 955]

[ 3 20076] ]

Model tuning report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **precision** | **recall** | **f1-score** | **support** |
| **0.0** | 1.00 | 0.95 | 0.98 | 20079 |
| **1.0** | 0.95 | 1.00 | 0.98 | 20079 |
|  |  |  |  |  |
| **accuracy** |  |  | 0.98 | 40158 |
| **macro avg** | 0.98 | 0.98 | 0.98 | 40158 |
| **weighted avg** | 0.98 | 0.98 | 0.98 | 40158 |

Scott Tasker

Aim: To evaluate the performance of the proposed intrusion classifiers.

**Background / Intro**

802.11 wireless networks, commonly known as Wi-Fi, have become the default for wireless local area networks. Securing these networks is critical to protect the networks users from malicious acts. Binary classification of incoming data packets (deciding if each data packet is ‘safe’ or ‘malicious’) is a challenging task, as each data packet received by the router has multiple parameters (157 fields), and a malicious signature may consist of only small differences across multiple fields. Machine learning algorithms are an ideal candidate for this application. This is because these algorithms can determine the probability that an unlabeled observation fits into a negative or positive set, based on labelled observations they have previously seen (training data). Labels are assigned based on these probabilities using a threshold for classification (default = 0.5).

To assess the performance of our intrusion detection model, we have utilised the reduced CLS portion of the Aegean Wifi Intrusion Dataset [1]. As this dataset is labelled, we can assess the models predicted classifications against the actual classifications for each observation in the test data. The lower the number of incorrectly classified observations, the higher the classifiers accuracy. For this application, the sensitivity of the classifier (e.g. the ability to correctly identify positive observations) will be given more weight when ranking algorithms than the overall accuracy, as the network must be protected, even at the expense that some ‘safe’ traffic will be labelled false positive and blocked. Another key metric for intrusion detection is the time required to make predictions. As to work in a real network, the algorithm would need to make predictions as traffic is received. A delay would not be a workable solution.

**Methods**

The performance of each model was assessed using the equations in figure 1, and receiver operator characteristic curves.



**Figure 1**. Calculation of binary classifier performance from confusion matrix. [2]

To enable high-throughput testing, a function (model\_evaluator.py) was built using the Python programming language, and the open source libraries: Numpy, Pandas, SciKit Learn and MatPlotLib. The function outputs a report for each binary classification model, and the results were compiled in an excel workbook.

Computation time was assessed using a short Python script to record the time required to train the model, and the time required to make predictions when run on an Apple MacBook Pro, 2.7GHz i5, 8GB DDR3 RAM. Results were recorded in an excel workbook.

**Results**

We evaluated 96 permutations of the intrusion detection classifier using the model\_evaulator function. Figure 2 is an example of the evaluation report returned by this function. The 10 best performing models are shown in Table 1. The full database is included in the appendix.

The 5 models with the highest sensitivity scores were taken forward for further analysis. Table 2 shows the computation time required to train each of these models, and the time required to predict unlabeled data. Table 3 compares these models, with the published benchmarks.



**Figure 2**. Output from model\_evaluator function.



Table 1. Top performing algorithms (10/96). Algorithms ranked according by number of false negatives observed.



Table 2. Analysis of time required to build model, and time required to test model,



Table 3. Comparison of the top 5 performing algorithms from our project to published models [1][2][3]

**Conclusion**

* We identified 2 models, Robust Scaler-RFElog10-Decision Tree, and Robust Scaler FS-RFElog10-Random Forest giving the highest sensitivity (only 3 false negatives of 20079 positive observations).
* Tuning of the Robust Scaler-RFElog10 Decision Tree and Robust Scaler-RFElog10 Random Forest algorithms did not increase the sensitivity or the specificity of the algorithm.
* The Robust Scaler-RFElog10-Decision Tree required the lowest computation to use, as demonstrated in table 2.
* The top performing models had comparable performance to state-of-the-art models [2],[3].
* Future analysis would identify the 3 false-negative observations in the database, to see if the pre-processing, or feature selection could be adjusted to allow correct classification of these samples.

**Future work**

As discussed in the previous sections of this report, our approach focused on simplicity and speed of computation. The chosen model was also the one that minimized the number of false negatives and therefore minimized the risks of an IoT system being infected.

There are several possible avenues for future development of this work.

**Expand number of features / try different feature selection methods**

Other methods in the literature on this subject built models using more features than in our best model (which used 10 features). Rezvy et. al built a highly accurate model for a similar purpose using 36 features derived from the same dataset [1]. In light of this, a future development may be to gradually increase the number of features used to train the model and evaluate the performance iteratively to find the optimum number of features. For this project, there was not enough time to carry out the necessary iterations to achieve this. Increasing the number of features may improve the accuracy of the model but at the cost of model speed and simplicity.

Another feature selection method worth exploring would be the use of an auto-encoder (a neural network that compresses a dataset) as this is also very popular in the literature [2]. The trade-off here would be the loss of interpretability from combining / compressing features.

**More complex learning algorithms**

One potential future development would be to use more complex learning algorithms. The project mostly focused on simpler algorithms such as random forests and KNN. It would be interesting to explore artificial neural networks as a learning approach using this dataset and evaluate the results to compare against the current model. This approach is popular in the literature.

In general, most avenues for future development would be to increase the complexity and assess the impact this has on model accuracy versus speed.

**Input from domain experts**

The view taken in this project was that it is desirable to look to minimise the number of false negatives and therefore the risk of the system being infected. A next step would be to look for input from domain experts on the trade-off between sensitivity and specificity in the context of this issue. This could allow us to refine our choice of algorithm and give us direction for which avenues to explore further. Further work and deeper reading of the literature could also be informative on this subject.

Bibliography:

**References- Feature Selection**

[1] Kolias et. al, “Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset”, IEEE Communication Surveys & Tutorials, Vol. 18, No. 1, First Quarter 2016

[2] Parker et. al, “DEMISe: Interpretable Deep Extraction and Mutual Information Selection Techniques for IoT Intrusion Detection” in ACM International Conference on Availability, Reliability and Security (ARES), 26-29 Aug. 2019, U.K

[3] Aminanto et. al, “Wi-Fi Intrusion Detection Using Weighted-Feature Selection for Neural Networks Classifier” in 2017 International Workshop on Big Data and Information Security (IWBIS)

References-Algorithm Comparison

[1]“3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.1 documentation,” *Scikit-learn.org*, 2013. [Online]. Available: https://scikit-learn.org/stable/modules/model\_evaluation.html#dummy-estimators. [Accessed: 17-Jan-2020].

[2] Aminanto et. al, “Wi-Fi Intrusion Detection Using Weighted-Feature Selection for Neural Networks Classifier” in 2017 International Workshop on Big Data and Information Security (IWBIS)

[3] Parker et. al, “DEMISe: Interpretable Deep Extraction and Mutual Information Selection Techniques for IoT Intrusion Detection” in ACM International Conference on Availability, Reliability and Security (ARES), 26-29 Aug. 2019, U.K

[4] Kolias et. al, “Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset”, IEEE Communication Surveys & Tutorials, Vol. 18, No. 1, First Quarter 2016

[5][Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html), Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[6]M. Sidana, “Types of classification algorithms in Machine Learning,” *Medium*, 28-Feb-2017. [Online]. Available: https://medium.com/@Mandysidana/machine-learning-types-of-classification-9497bd4f2e14. [Accessed: 18-Jan-2020].

‌

**References- Model Tuning**

Literature:

[1] Python Scikit Learn documentation. “LogisticRegression”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html>

[2] Python Scikit Learn documentation. “DecisionTreeClassifier”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

[3] Python Scikit Learn documentation. “RandomForestClassifier”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

[4] Python Scikit Learn documentation. “SVC”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

[5] Python Scikit Learn documentation. “KNeighborsClassifier”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

[6] Python Scikit Learn documentation. “GaussianNB”. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

[7] Python Scikit Learn documentation. “Comparing randomized search and grid search for hyperparameter estimation”. Available: <https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html>

**References-Model Evaluation**

[1] C. Kolias, G. Kambourakis, A. Stavrou and S. Gritzalis, "Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 184-208, 2016. Available: 10.1109/comst.2015.2402161 [Accessed 13 January 2020].

[2] M. Aminanto, R. Choi, H. Tanuwidjaja, P. Yoo and K. Kim, "Deep Abstraction and Weighted Feature Selection for Wi-Fi Impersonation Detection", *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 621-636, 2018. Available: 10.1109/tifs.2017.2762828 [Accessed 13 January 2020].

[3] M. Aminanto and K. Kim, "Detecting Impersonation Attack in WiFi Networks Using Deep Learning Approach", *Information Security Applications*, pp. 136-147, 2017. Available: 10.1007/978-3-319-56549-1\_12 [Accessed 13 January 2020].

Future Work Section

**References**

[1] Rezvy et. al, “An efficient deep learning model for intrusion classification and prediction in 5G and IoT networks” in 53rd Annual Conference on Information Sciences and Systems (CISS), 2019.

[2] Kolias et. al, “Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset”, IEEE Communication Surveys & Tutorials, Vol. 18, No. 1, First Quarter 2016

Appendixes

**Appendix – Feature Selection – Alex Bates**

1. **Recursive feature elimination**

Recursive feature elimination is an algorithm that starts by building a model using all attributes to predict the target. It then removes features (removing the least useful features first) and re-builds a model on the remaining attributes. This is repeated until the specified number of attributes is reached.

Python code – recursive feature elimination using a logistic regression classifier with 10 features

# Initiate model

model = LogisticRegression()

# Try 10 features

rfe\_log = RFE(model, 10)

fit\_rfe\_log = rfe\_log.fit(X, Y)

# Create list with names of features

rfe\_log\_features = X.columns[fit\_rfe\_log.get\_support()]

for feature in rfe\_log\_features:

print(feature)

radiotap.datarate

wlan.fc.retry

wlan.fc.pwrmgt

wlan.fc.protected

wlan\_mgt.fixed.capabilities.preamble

wlan\_mgt.fixed.capabilities.short\_slot\_time

wlan\_mgt.fixed.timestamp

wlan\_mgt.fixed.auth\_seq

wlan\_mgt.rsn.akms.type

wlan.wep.key

1. **Feature importance and decision trees**

In this method, a random set of decision trees are fitted to subsets of the data with the results averaged across the trees. The top 10 features by feature importance are selected for the learning phase. Feature importance is calculated as the decrease in “node-impurity” weighted by the probability of reaching that node. Node impurity is a measure of the homogeneity of labels at a node. Nodes of a decision tree that greatly increase the homogeneity of labels imply that that feature has strong predictive power on the target – it is therefore a more important feature.

Python code – extra trees classifier with top 10 most important feature selected

# Initiate model and fit to data

model\_tree = ExtraTreesClassifier()

fit\_tree = model\_tree.fit(X,Y)

# Ensure reproducibility

np.random.seed(999)

# Get feature importances

feature\_importance = fit\_tree.feature\_importances\_

# Join feature importances to feature names and rank by importance

feature\_ranking\_data = {'Feature':X.columns.values, 'Importance': feature\_importance}

tree\_features\_imp = pd.DataFrame(data=feature\_ranking\_data).nlargest(10,'Importance')

# Get list object of the top 10 features

tree\_features = list(tree\_features\_imp['Feature'])

print(tree\_features\_imp)

22 wlan.fc.subtype 0.181461

23 wlan.fc.ds 0.120586

13 radiotap.datarate 0.113596

15 radiotap.channel.type.cck 0.070449

27 wlan.fc.protected 0.064409

28 wlan.duration 0.058927

16 radiotap.channel.type.ofdm 0.056004

2 frame.len 0.047525

11 radiotap.mactime 0.040032

3 frame.cap\_len 0.025655

1. **Principal Component Analysis**

Principal Component Analysis compresses a dataset statistically into its principal components. It captures as much of the variance in the dataset as possible at each principal component. These principal components can be used as features in the learning phase.

Python code – PCA with top 5 principal components selected

pca = PCA(n\_components=5)

fit\_pca = pca.fit(X)

# summarize components

print("Explained Variance: %s" % fit\_pca.explained\_variance\_ratio\_)

print("Total variance explained by first 5 components: %s" % sum(fit\_pca.explained\_variance\_ratio\_))

Explained Variance: [0.80629725 0.12093112 0.02414355 0.0183569 0.01252285]

Total variance explained by first 5 components: 0.9822516621351488

**Appendix: Model Evaluation ( next page)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Preprocessing | FeatureSelection | Algorithms | Tuning | Accuracy | True Positive | True Negative | False Positive | False Negative | specificity = TN/(TN+FP) | recall (or sensitivity) = TP / (TP + FN) | precision = TP / (TP + FP) | F1score = (2\* True Positive)/((2\*True Positive)+False Positive+False Negative) | False Positives Rate = FP/(FP+TN) | MCC |
| RobustScaler | RFE, logistic classifier, 10 features | DT | default parameters | 0.97614423 | 20076 | 19124 | 955 | 3 | 0.999843154 | 0.99985059 | 0.954590842 | 0.976696667 | 0.04756213 | 0.976951366 |
| RobustScaler | RFE, logistic classifier, 10 features | DT | tuned parameters | 0.97614423 | 20076 | 19124 | 955 | 3 | 0.999843154 | 0.99985059 | 0.954590842 | 0.976696667 | 0.04756213 | 0.976951366 |
| RobustScaler | RFE, logistic classifier, 10 features | RF | default parameters | 0.97614423 | 20076 | 19124 | 955 | 3 | 0.999843154 | 0.99985059 | 0.954590842 | 0.976696667 | 0.04756213 | 0.976951366 |
| RobustScaler | RFE, logistic classifier, 10 features | RF | tuned parameters | 0.97614423 | 20076 | 19124 | 955 | 3 | 0.999843154 | 0.99985059 | 0.954590842 | 0.976696667 | 0.04756213 | 0.976951366 |
| RobustScaler | RFE, logistic classifier, 10 features | KNN | tuned parameters | 0.97552169 | 20076 | 19099 | 980 | 3 | 0.999842948 | 0.99985059 | 0.953457447 | 0.976103075 | 0.048807212 | 0.976371021 |
| RobustScaler | RFE, logistic classifier, 10 features | LR | tuned parameters | 0.97509836 | 20076 | 19082 | 997 | 3 | 0.999842808 | 0.99985059 | 0.952688274 | 0.975699844 | 0.049653867 | 0.975976977 |
| RobustScaler | RFE, logistic classifier, 10 features | LR | default parameters | 0.96269735 | 20076 | 18584 | 1495 | 3 | 0.999838597 | 0.99985059 | 0.930693987 | 0.964033613 | 0.074455899 | 0.964641156 |
| RobustScaler | RFE, logistic classifier, 10 features | SVC | default parameters | 0.96257284 | 20073 | 18582 | 1497 | 6 | 0.999677211 | 0.99970118 | 0.930598053 | 0.963912699 | 0.074555506 | 0.964507733 |
| StandardScaler | RFE, logistic classifier, 5 features | SVC | tuned parameters | 0.983564919 | 20067 | 19431 | 648 | 12 | 0.999382811 | 0.999402361 | 0.96871832 | 0.98382115 | 0.032272524 | 0.983921116 |
| StandardScaler | RFE, logistic classifier, 5 features | SVC | default parameters | 0.978509886 | 20067 | 19228 | 851 | 12 | 0.999376299 | 0.999402361 | 0.959317334 | 0.978949679 | 0.042382589 | 0.979128827 |
| StandardScaler | Decision tree classifier, 10 most important features | LR | default parameters | 0.927411724 | 20064 | 17179 | 2900 | 15 | 0.999127603 | 0.999252951 | 0.873715381 | 0.932277025 | 0.144429503 | 0.934260305 |
| RobustScaler | Decision tree classifier, 10 most important features | KNN | default parameters | 0.98819662 | 20053 | 19631 | 448 | 26 | 0.998677316 | 0.998705115 | 0.978147407 | 0.988319369 | 0.022311868 | 0.988343569 |
| StandardScaler | RFE, logistic classifier, 5 features | LR | tuned parameters | 0.965187509 | 20052 | 18708 | 1371 | 27 | 0.998558847 | 0.998655312 | 0.936003361 | 0.966314876 | 0.068280293 | 0.96672657 |
| StandardScaler | Decision tree classifier, 10 most important features | LR | tuned parameters | 0.929354051 | 20047 | 17274 | 2805 | 32 | 0.99815093 | 0.998406295 | 0.877253632 | 0.933917216 | 0.139698192 | 0.935628964 |
| StandardScaler | RFE, logistic classifier, 10 features | LR | tuned parameters | 0.98055182 | 20032 | 19345 | 734 | 47 | 0.99757632 | 0.997659246 | 0.964653761 | 0.980878933 | 0.036555605 | 0.980930375 |
| StandardScaler | RFE, logistic classifier, 10 features | LR | default parameters | 0.980003984 | 20032 | 19323 | 756 | 47 | 0.997573567 | 0.997659246 | 0.963632865 | 0.980350894 | 0.037651277 | 0.980408459 |
| StandardScaler | RFE, logistic classifier, 5 features | LR | default parameters | 0.960779919 | 20024 | 18559 | 1520 | 55 | 0.997045235 | 0.99726082 | 0.929446714 | 0.962160344 | 0.075700981 | 0.962540291 |
| RobustScaler | Decision tree classifier, 10 most important features | KNN | tuned parameters | 0.98142338 | 19911 | 19501 | 578 | 168 | 0.991458641 | 0.991633049 | 0.971789741 | 0.981611122 | 0.028786294 | 0.981415759 |
| StandardScaler | RFE, logistic classifier, 5 features | NBC | tuned parameters | 0.940659395 | 19698 | 18077 | 2002 | 381 | 0.979358544 | 0.981024951 | 0.907741935 | 0.942961775 | 0.099706161 | 0.941650897 |
| RobustScaler | Decision tree classifier, 10 most important features | NB | default parameters | 0.9558992 | 18613 | 19774 | 305 | 1466 | 0.930979284 | 0.926988396 | 0.983877788 | 0.95458625 | 0.01519 | 0.953849383 |
| RobustScaler | Decision tree classifier, 10 most important features | NB | tuned parameters | 0.9558992 | 18613 | 19774 | 305 | 1466 | 0.930979284 | 0.926988396 | 0.983877788 | 0.95458625 | 0.01519 | 0.953849383 |
| StandardScaler | Decision tree classifier, 10 most important features | NBC | tuned parameters | 0.944842871 | 18613 | 19330 | 749 | 1466 | 0.929505674 | 0.926988396 | 0.96131598 | 0.943840166 | 0.037302655 | 0.9411152 |
| RobustScaler | RFE, logistic classifier, 10 features | NB | tuned parameters | 0.91715225 | 18613 | 18218 | 1861 | 1466 | 0.925523268 | 0.926988396 | 0.90910423 | 0.917959214 | 0.092683899 | 0.910616797 |
| RobustScaler | RFE, logistic classifier, 5 features | RF | default parameters | 0.93821903 | 18609 | 19068 | 1011 | 1470 | 0.928425358 | 0.926789183 | 0.948470948 | 0.937504723 | 0.050351113 | 0.933640556 |
| RobustScaler | RFE, logistic classifier, 5 features | KNN | tuned parameters | 0.93821903 | 18609 | 19068 | 1011 | 1470 | 0.928425358 | 0.926789183 | 0.948470948 | 0.937504723 | 0.050351113 | 0.933640556 |
| RobustScaler | RFE, logistic classifier, 5 features | DT | default parameters | 0.93821903 | 18609 | 19068 | 1011 | 1470 | 0.928425358 | 0.926789183 | 0.948470948 | 0.937504723 | 0.050351113 | 0.933640556 |
| RobustScaler | RFE, logistic classifier, 5 features | DT | tuned parameters | 0.93821903 | 18609 | 19068 | 1011 | 1470 | 0.928425358 | 0.926789183 | 0.948470948 | 0.937504723 | 0.050351113 | 0.933640556 |
| RobustScaler | Decision tree classifier, 10 most important features | SVM | tuned parameters | 0.96331989 | 18606 | 20079 | 0 | 1473 | 0.931653675 | 0.926639773 | 1 | 0.961923226 | 0 | 0.962621303 |
| RobustScaler | Decision tree classifier, 10 most important features | RF | tuned parameters | 0.96331989 | 18606 | 20079 | 0 | 1473 | 0.931653675 | 0.926639773 | 1 | 0.961923226 | 0 | 0.962621303 |
| RobustScaler | Decision tree classifier, 10 most important features | LR | default parameters | 0.95634743 | 18606 | 19799 | 280 | 1473 | 0.930754043 | 0.926639773 | 0.985174203 | 0.955010907 | 0.013944918 | 0.954389107 |
| RobustScaler | Decision tree classifier, 10 most important features | SVM | default parameters | 0.95231336 | 18606 | 19637 | 442 | 1473 | 0.930222643 | 0.926639773 | 0.976795464 | 0.951056815 | 0.022013048 | 0.94969183 |
| RobustScaler | Decision tree classifier, 10 most important features | LR | tuned parameters | 0.95099358 | 18606 | 19584 | 495 | 1473 | 0.930047015 | 0.926639773 | 0.974085126 | 0.949770291 | 0.024652622 | 0.948165213 |
| StandardScaler | RFE, logistic classifier, 10 features | NBC | tuned parameters | 0.940759002 | 18606 | 19173 | 906 | 1473 | 0.928654461 | 0.926639773 | 0.953567036 | 0.939910586 | 0.045121769 | 0.936490423 |
| RobustScaler | RFE, logistic classifier, 5 features | SVC | default parameters | 0.9376961 | 18606 | 19050 | 1029 | 1473 | 0.928226867 | 0.926639773 | 0.947593583 | 0.936999547 | 0.051247572 | 0.933050951 |
| RobustScaler | RFE, logistic classifier, 5 features | LR | default parameters | 0.9376214 | 18606 | 19047 | 1032 | 1473 | 0.928216374 | 0.926639773 | 0.947448824 | 0.936928771 | 0.051396982 | 0.932967365 |
| RobustScaler | RFE, logistic classifier, 5 features | LR | tuned parameters | 0.9376214 | 18606 | 19047 | 1032 | 1473 | 0.928216374 | 0.926639773 | 0.947448824 | 0.936928771 | 0.051396982 | 0.932967365 |
| RobustScaler | RFE, logistic classifier, 10 features | SVC | tuned parameters | 0.85756263 | 15339 | 19099 | 980 | 4740 | 0.801166156 | 0.763932467 | 0.939947301 | 0.842848508 | 0.048807212 | 0.833945818 |
| RobustScaler | RFE, logistic classifier, 5 features | NB | default parameters | 0.8055182 | 15337 | 17011 | 3068 | 4742 | 0.782007079 | 0.76383286 | 0.833306167 | 0.797058518 | 0.152796454 | 0.753325122 |
| RobustScaler | RFE, logistic classifier, 5 features | NB | tuned parameters | 0.79710145 | 15337 | 16673 | 3406 | 4742 | 0.778566425 | 0.76383286 | 0.818278824 | 0.790119005 | 0.169629962 | 0.740652653 |
| RobustScaler | RFE, logistic classifier, 10 features | NB | default parameters | 0.81866627 | 15336 | 17540 | 2539 | 4743 | 0.787147153 | 0.763783057 | 0.857958042 | 0.808136165 | 0.12645052 | 0.77326187 |
| RobustScaler | RFE, logistic classifier, 5 features | SVC | tuned parameters | 0.82025997 | 13872 | 19068 | 1011 | 6207 | 0.754421365 | 0.690871059 | 0.932070147 | 0.79354728 | 0.050351113 | 0.783421334 |
| RobustScaler | RFE, logistic classifier, 5 features | RF | tuned parameters | 0.82025997 | 13872 | 19068 | 1011 | 6207 | 0.754421365 | 0.690871059 | 0.932070147 | 0.79354728 | 0.050351113 | 0.783421334 |
| StandardScaler | PCA, top 5 features | KNN | tuned parameters | 0.335997809 | 1473 | 12020 | 8059 | 18606 | 0.39247698 | 0.073360227 | 0.154532102 | 0.099490054 | 0.40136461 | -0.795236456 |
| StandardScaler | PCA, top 5 features | KNN | default parameters | 0.335524678 | 1472 | 12002 | 8077 | 18607 | 0.392106897 | 0.073310424 | 0.154152267 | 0.099365465 | 0.402261069 | -0.798015436 |
| StandardScaler | PCA, top 5 features | RF | default parameters | 0.262737188 | 1470 | 9081 | 10998 | 18609 | 0.327952329 | 0.073210817 | 0.117901829 | 0.090330906 | 0.547736441 | -1.331498602 |
| StandardScaler | PCA, top 5 features | LR | tuned parameters | 0.112580308 | 1469 | 3052 | 17027 | 18610 | 0.140891884 | 0.073161014 | 0.079422578 | 0.076163318 | 0.848000398 | -5.311306526 |
| StandardScaler | PCA, top 5 features | LR | default parameters | 0.112480701 | 1469 | 3048 | 17031 | 18610 | 0.140733216 | 0.073161014 | 0.079405405 | 0.076155421 | 0.848199612 | -5.319068958 |
| StandardScaler | RFE, logistic classifier, 5 features | DT | tuned parameters | 0.533019573 | 1468 | 19937 | 142 | 18611 | 0.517199336 | 0.073111211 | 0.911801242 | 0.135368159 | 0.007072065 | 0.234877717 |
| StandardScaler | RFE, logistic classifier, 5 features | DT | default parameters | 0.532745655 | 1468 | 19926 | 153 | 18611 | 0.517061525 | 0.073111211 | 0.905613819 | 0.135299539 | 0.007619901 | 0.232265742 |
| RobustScaler | RFE, logistic classifier, 10 features | KNN | default parameters | 0.5364062 | 1467 | 20074 | 5 | 18612 | 0.518895725 | 0.073061407 | 0.996603261 | 0.136142174 | 0.000249016 | 0.268986564 |
| StandardScaler | RFE, logistic classifier, 10 features | DT | tuned parameters | 0.533293491 | 1467 | 19949 | 130 | 18612 | 0.517336169 | 0.073061407 | 0.91859737 | 0.135357077 | 0.006474426 | 0.237645097 |
| StandardScaler | RFE, logistic classifier, 10 features | RF | default parameters | 0.533094278 | 1467 | 19941 | 138 | 18612 | 0.517236013 | 0.073061407 | 0.914018692 | 0.135307139 | 0.006872852 | 0.235728216 |
| StandardScaler | RFE, logistic classifier, 10 features | DT | default parameters | 0.533044474 | 1467 | 19939 | 140 | 18612 | 0.517210967 | 0.073061407 | 0.912881145 | 0.13529466 | 0.006972459 | 0.235250552 |
| StandardScaler | RFE, logistic classifier, 5 features | RF | tuned parameters | 0.532994671 | 1467 | 19937 | 142 | 18612 | 0.517185919 | 0.073061407 | 0.911746426 | 0.135282184 | 0.007072065 | 0.234773507 |
| StandardScaler | RFE, logistic classifier, 10 features | RF | tuned parameters | 0.531600179 | 1467 | 19881 | 198 | 18612 | 0.516483516 | 0.073061407 | 0.881081081 | 0.134933775 | 0.009861049 | 0.221659994 |
| StandardScaler | PCA, top 5 features | DT | tuned parameters | 0.510508491 | 1467 | 19034 | 1045 | 18612 | 0.505604845 | 0.073061407 | 0.583996815 | 0.129874729 | 0.052044425 | 0.06268211 |
| StandardScaler | RFE, logistic classifier, 5 features | RF | default parameters | 0.532895064 | 1464 | 19936 | 143 | 18615 | 0.517133148 | 0.072911998 | 0.911014312 | 0.135017984 | 0.007121869 | 0.234222113 |
| StandardScaler | RFE, logistic classifier, 5 features | KNN | default parameters | 0.52492654 | 1463 | 19617 | 462 | 18616 | 0.513090785 | 0.072862194 | 0.76 | 0.132975823 | 0.023009114 | 0.164799981 |
| StandardScaler | RFE, logistic classifier, 5 features | KNN | tuned parameters | 0.525673589 | 1461 | 19649 | 430 | 18618 | 0.513471137 | 0.072762588 | 0.772607086 | 0.132999545 | 0.021415409 | 0.170979282 |
| StandardScaler | RFE, logistic classifier, 10 features | KNN | default parameters | 0.535011704 | 1444 | 20041 | 38 | 18635 | 0.518176647 | 0.071915932 | 0.974358974 | 0.13394555 | 0.001892525 | 0.258233672 |
| StandardScaler | RFE, logistic classifier, 10 features | KNN | tuned parameters | 0.534762687 | 1444 | 20031 | 48 | 18635 | 0.518052035 | 0.071915932 | 0.967828418 | 0.133883455 | 0.002390557 | 0.255663904 |
| StandardScaler | RFE, logistic classifier, 10 features | SVC | default parameters | 0.521788934 | 1444 | 19510 | 569 | 18635 | 0.511469393 | 0.071915932 | 0.717337308 | 0.130726055 | 0.028338065 | 0.141644572 |
| StandardScaler | PCA, top 5 features | DT | default parameters | 0.47903282 | 1438 | 17799 | 2280 | 18641 | 0.488446762 | 0.071617112 | 0.386767079 | 0.12085557 | 0.113551472 | -0.109934204 |
| StandardScaler | Decision tree classifier, 10 most important features | RF | tuned parameters | 0.49818218 | 1395 | 18611 | 1468 | 18684 | 0.499021317 | 0.069475571 | 0.487251135 | 0.121611019 | 0.073111211 | -0.010387559 |
| StandardScaler | Decision tree classifier, 10 most important features | DT | tuned parameters | 0.51200259 | 1394 | 19167 | 912 | 18685 | 0.506366903 | 0.069425768 | 0.604509974 | 0.124547688 | 0.045420589 | 0.07420519 |
| StandardScaler | Decision tree classifier, 10 most important features | KNN | default parameters | 0.511380049 | 1351 | 19185 | 894 | 18728 | 0.506026956 | 0.067284227 | 0.601781737 | 0.121035657 | 0.04452413 | 0.07123891 |
| StandardScaler | Decision tree classifier, 10 most important features | KNN | tuned parameters | 0.511529459 | 1346 | 19196 | 883 | 18733 | 0.506103509 | 0.067035211 | 0.603858232 | 0.120674198 | 0.043976294 | 0.07239128 |
| StandardScaler | Decision tree classifier, 10 most important features | SVC | default parameters | 0.503013098 | 1332 | 18868 | 1211 | 18747 | 0.501608401 | 0.066337965 | 0.523790798 | 0.117761471 | 0.060311769 | 0.018020107 |
| StandardScaler | Decision tree classifier, 10 most important features | SVC | tuned parameters | 0.504382688 | 1321 | 18934 | 1145 | 18758 | 0.502334713 | 0.065790129 | 0.53568532 | 0.117187847 | 0.057024752 | 0.026524353 |
| RobustScaler | PCA, top 5 features | LR | tuned parameters | 0.28783804 | 666 | 10893 | 9186 | 19413 | 0.359433775 | 0.033168983 | 0.067600487 | 0.044502355 | 0.457492903 | -1.116608497 |
| RobustScaler | PCA, top 5 features | LR | default parameters | 0.287887843 | 662 | 10899 | 9180 | 19417 | 0.359513128 | 0.032969769 | 0.067262751 | 0.044249858 | 0.457194083 | -1.116298501 |
| RobustScaler | Decision tree classifier, 10 most important features | DT | default parameters | 0.48585587 | 459 | 19052 | 1027 | 19620 | 0.492656185 | 0.022859704 | 0.308882907 | 0.042568978 | 0.051147966 | -0.109589614 |
| RobustScaler | Decision tree classifier, 10 most important features | DT | tuned parameters | 0.48585587 | 459 | 19052 | 1027 | 19620 | 0.492656185 | 0.022859704 | 0.308882907 | 0.042568978 | 0.051147966 | -0.109589614 |
| StandardScaler | Decision tree classifier, 10 most important features | DT | default parameters | 0.490412869 | 148 | 19546 | 533 | 19931 | 0.495123743 | 0.007370885 | 0.21732746 | 0.014258189 | 0.026545147 | -0.10695481 |
| RobustScaler | PCA, top 5 features | DT | default parameters | 0.35063997 | 140 | 13941 | 6138 | 19939 | 0.4114817 | 0.006972459 | 0.022300096 | 0.010623364 | 0.305692515 | -0.769436736 |
| RobustScaler | PCA, top 5 features | DT | tuned parameters | 0.37367399 | 127 | 14879 | 5200 | 19952 | 0.42717694 | 0.006325016 | 0.023840811 | 0.009997638 | 0.258977041 | -0.661943328 |
| StandardScaler | Decision tree classifier, 10 most important features | RF | default parameters | 0.495443 | 89 | 19807 | 272 | 19990 | 0.497700832 | 0.004432492 | 0.246537396 | 0.008708415 | 0.013546491 | -0.068904856 |
| StandardScaler | PCA, top 5 features | RF | tuned parameters | 0.456546641 | 78 | 18256 | 1823 | 20001 | 0.477193716 | 0.003884656 | 0.041031036 | 0.007097361 | 0.090791374 | -0.310648875 |
| RobustScaler | PCA, top 5 features | RF | default parameters | 0.49681259 | 12 | 19939 | 140 | 20067 | 0.49840024 | 0.000597639 | 0.078947368 | 0.001186298 | 0.006972459 | -0.073782921 |
| StandardScaler | Decision tree classifier, 10 most important features | NBC | default parameters | 0.499551771 | 7 | 20054 | 25 | 20072 | 0.499775707 | 0.000348623 | 0.21875 | 0.000696136 | 0.001245082 | -0.022483688 |
| RobustScaler | Decision tree classifier, 10 most important features | RF | default parameters | 0.5000249 | 1 | 20079 | 0 | 20078 | 0.500012451 | 4.98033E-05 | 1 | 9.96016E-05 | 0 | 0.007057144 |
| RobustScaler | PCA, top 5 features | RF | tuned parameters | 0.47920713 | 1 | 19243 | 836 | 20078 | 0.489382264 | 4.98033E-05 | 0.001194743 | 9.56206E-05 | 0.04163554 | -0.212530863 |
| RobustScaler | PCA, top 5 features | SVM | default parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| RobustScaler | PCA, top 5 features | SVM | tuned parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| RobustScaler | PCA, top 5 features | NB | default parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| RobustScaler | PCA, top 5 features | NB | tuned parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| RobustScaler | PCA, top 5 features | KNN | default parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| RobustScaler | PCA, top 5 features | KNN | tuned parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| StandardScaler | RFE, logistic classifier, 10 features | NBC | default parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| StandardScaler | RFE, logistic classifier, 5 features | NBC | default parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| RobustScaler | RFE, logistic classifier, 5 features | KNN | default parameters | 0.5 | 0 | 20079 | 0 | 20079 | 0.5 | 0 | #DIV/0! | 0 | 0 | 0 |
| StandardScaler | RFE, logistic classifier, 10 features | SVC | tuned parameters | 0.496563574 | 0 | 19941 | 138 | 20079 | 0.498275862 | 0 | 0 | 0 | 0.006872852 | -0.083476388 |
| StandardScaler | PCA, top 5 features | NBC | default parameters | 0.487673689 | 0 | 19584 | 495 | 20079 | 0.493759927 | 0 | 0 | 0 | 0.024652622 | -0.160980117 |
| StandardScaler | PCA, top 5 features | NBC | tuned parameters | 0.487673689 | 0 | 19584 | 495 | 20079 | 0.493759927 | 0 | 0 | 0 | 0.024652622 | -0.160980117 |
| StandardScaler | PCA, top 5 features | SVC | default parameters | 0.48070123 | 0 | 19304 | 775 | 20079 | 0.490160729 | 0 | 0 | 0 | 0.03859754 | -0.204349972 |
| StandardScaler | PCA, top 5 features | SVC | tuned parameters | 0.446137756 | 0 | 17916 | 2163 | 20079 | 0.471535728 | 0 | 0 | 0 | 0.107724488 | -0.3678394 |