**21/08/2020**

Examinationof cyber attack classification in the network traffic databaseusing the NSL-KDD dataset.

**By Jasmine Featherston**

**Assignment 5 : SIT719**

**INTRODUCTION**

Machine learning is a type of artificial intelligence that allows software applications to learn from a large amount of data and become more accurate in predicting outcomes, without human intervention. Steps performed during a machine learning task include training a model using training data, building a model and then performing the task. The end performance then provides feedback to the algorithm to improve its predictions. The feedback will occur many times in order to optimise the algorithm .

Machine learning is divided into three types; supervised learning, unsupervised learning and reinforcement. This project will be exploring a supervised learning model for intrusion detection classification. Supervised learning involves algorithms that automate decision making process by generalising from known examples. The user provides the algorithm with pairs of inputs and desired outputs. The algorithm then iterates over the data to produce the desired output ( Chio and Freeman 2018). The model is then fed new data and uses its trained model to predict the outcomes.

Depending on the characteristics of the target variable, the machine learning algorithm can be a classification (discrete target variable) or a regression (continuous target variable) task. This project takes the NSL -KDD dataset as the research object and investigates classification of systems intrusions by type. Classification is a predictive modelling problem that involves assigning a class label to an example.

The dataset is comprised of 4 different types of intrusion; DOS (denial of service), Probe (attempt at gathering information about a network of computers), U2R (user to root attack) and r2l (remote to local). There is a fifth category called benign for normal software requests. Five different classification algorithms will be used to assess the data in our classification task. These include; include Random Forest, KNN , MPLClassifier, Decision tree classifier with AdaBoost and Linear SVC using OneVsOne.

The project will use an open source library called Scikit-learn which is built on libraries such as Numpy, Pandas, Scipy and Matplotlib. To initiate a supervised learning model the initial NSL-KDD dataset will be split into two groups; the training set and the testing set. Initially, the training data, including features and target will be fed into the algorithm. The algorithm will then use this data to help form some relationships between the variables. This will enable predictions about future data. Following this, the test data from our independent variables are fed into the now trained algorithm. The model will hence make some predictions about the target (attack categories) using this new test data.

**DATA EXPLORATION**

The NSL-KDD data set consist for 42 fields (columns). The training data set has a length of 125,973 rows. The testing dataset has a length of 22,544 rows. There are 42 column headers in the data set which include; duration, protocol\_type, service, flag, src\_bytes, dst\_bytes, land, wrong\_fragment, urgent, hot, num\_failed\_logins, logged\_in, num\_compromised, root\_shell, su\_attempted, num\_root, num\_file\_creations, num\_shells, num\_access\_files, is\_host\_login, is\_guest\_login, count, srv\_count, serror\_rate, srv\_serror\_rate, same\_srv\_rate, diff\_srv\_rate, srv\_diff\_host\_rate, dst\_host\_count, dst\_host\_srv\_count, dst\_host\_same\_srv\_rate, dst\_host\_diff\_srv\_rate, dst\_host\_same\_src\_port\_rate, dst\_host\_srv\_diff\_host\_rate, dst\_host\_serror\_rate, dst\_host\_srv\_serror\_rate, dst\_host\_rerror\_rate, dst\_host\_srv\_rerror\_rate, attack\_type, attack\_category.

Label, protocol and flag fields are character types. Those fields cannot be directly used as inputs for an algorithm, so they have been changed to numerical data during data cleaning. Secondly, some columns have been dropped as they were not necessary for classification purposes. After pre-processing which involved changing categorical data to numerical data there were 118 columns in our training and test sets.

Our target variable, attack category, consists of 5 classes. These include; benign, dos, probe, r2l, u2r. Our features will include all other columns of data. The number of samples in each of our five classes include:

|  |  |
| --- | --- |
| **Attack class** | **Number of samples** |
| Benign | 67343 |
| Dos | 45927 |
| Probe | 11656 |
| R2l | 995 |
| u2r | 52 |

A deficiency of the NSL-KDD data set is its uneven distribution. It is heavily skewed towards attacks for Dos and probe. However, there is little data included in the classes r2l and u2r. Algorithms may have difficulty correctly predicting these two later categories due to their low numbers within the training dataset.

**EVALUATION OF MODELS**

Algorithms will be evaluated using standard metrics including accuracy, recall, precision, false negative rates and false positive rates.

|  |  |  |
| --- | --- | --- |
| Metric | Explanation | Formula |
| False Positive (FP) | False positives occur when the system recognises true events as erroneous ones. |  |
| False Negative (FN) | False negatives occur when the algorithm erroneously recognises something that the user does not intend it to find. |  |
| Recall (R) | The number of correct classifications out of the total number of true classifications. |  |
| Precision (P) | The number of correct classifications out of the total number classified as correct |  |
| Accuracy (A) | The number of correct classifications out of the total observations |  |

A high false positive rate occurs when the intrusion detection system has triggered an alarm for normal traffic. These can be costly to an organisation due to the increased time taken for a company to investigate these so-called attacks that were benign in the first instance. A company may lose its trust in the algorithm and discontinue its use. If false positives are high for an intrusion detection system, then the system is playing very safe. A high false negative rate is also an issue. These occur when a real attack has occurred, and the intrusion detection system has failed to detect it.

Recall measures actual positive classes. Out of all the true positive cases, how likely was the model to choose the correct class. It indicates how many of the positive classes the model is able to predict correctly.

The above metrics will be computed for each of the five algorithms and reported in Table 1. Using cross-validation, the time taken to fit the data (fit\_time) and the time taken to score the data (score\_time) will be calculated to assess the performance of each algorithm. The results will also be tabulated in Table 1.

**Learning curves**

Learning curves are widely used diagnostic tools in machine learning for algorithms that learn from dataset incrementally. A learning curve shows the validation and training score of an estimator for varying numbers of training samples. It is a tool to find out how much we benefit from adding more training data and whether the estimator suffers more from a variance error or a bias error . Learning curves help diagnose problems such as underfitting or overfitting a model as well as whether the training and validation datasets are suitably representative (Brownlee 2019).

The train learning curve is calculated from the training dataset which gives an idea of how well the model is learning. The validation learning curve is calculated from a holdout validation dataset that gives an idea of how well the model is generalising.

The curves are plotted with the mean cross-validation scores. Variability during cross-validation is shown with shaded areas that represent a standard deviation above and below the mean for all cross validations.

If the training score is high then the training data is well estimated by the model. If the model fits the data well it indicates a low bias with respect to that set of data. High bias can also be seen as high variability around the training curve.

If the variance is high then the model fits training data too well (overfitting) which leads to difficulty generalising the result to new data. If there is a large gap between the training curve and the validation curve then there is high variance. Generally, the smaller the gap, the lower the variance. A model with low variance has good generalisability to different training sets. . High variance can also be seen by more variability around the validation curve.

**DECISION\_TREE WITH ADABOOST**

Decision tree with Adaboost was one model used to classify the dataset. Decision tree is a tree shaped diagram used to determine a course of action. Each branch of the tree represents a possible decision or occurrence of reaction. A classification tree will determine a set of logical if-then conditions to classify problems. The advantages of a decision tree include that little effort is required for data preparation. Furthermore, non-linear data does not affect its performance. On the other hand, the algorithm is prone to overfitting (the algorithm captures too much noise in the data). The model can also suffer from high variance if there is small variation in the data set.

Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers (Brownlee 2020) . The basic concept is to set equal weights to each feature and train the data sample in each iteration such that it makes accurate predictions. If an observation is predicted incorrectly its weight is increased on the next iteration. If an observation is predicted correctly the weight is decreased upon the next iteration. A boosting algorithm usually only uses one variable to make a decision, thereby forming a decision stump rather than a tree. This makes the algorithm a weak learner and the resultant accuracy can be poor.

A boosting algorithm usually needs to be combined in an ensemble of algorithms to obtain good accuracy. Most often, it is combined with the base estimator Decision tree. Adaboost is one type of boosting algorithm.

The hyperparameters of the Adaboost algorithm were optimised using Gridsearch. Gridsearch is the process of performing hyper parameter tuning in order to determine optimal values for a given model (Krishni 2019) . Parameters included in the Gridsearch function include a a list of parameters and the range of values for each parameter. The parameter ‘cv’ represents the cross-validation process that is performed in order to determine the hyperparameter value set which provides the best accuracy levels.

Training Adaboost alone on our dataset resulted in an accuracy of 62.97%. Optimisation of the algorithm using gridsearch improved the accuracy slightly to 65.90%. When Decision tree and Adaboost were used in an ensemble the accuracy rose to 75.26%.

**Decision\_tree with Adaboost (Name of model: clf16)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attack class | Accuracy | Precision | Recall | F1 | FPR% | FNR% |
| Benign | 76.37% | 65.12% | 97.22% | 78.00% | 39.39% | 2.78% |
| Dos | 91.71% | 96.31% | 78.55% | 86.53% | 1.54% | 21.45% |
| Probe | 94.44% | 83.63% | 60.09% | 69.93% | 1.29% | 39.91% |
| R2l | 88.87% | 94.44% | 2.64% | 5.14% | 0.02% | 97.36% |
| U2r | 99.12% | 66.67% | 2% | 3.88% | 0.01% | 98.00% |

Accuracy = 75.26%

Weighted precision = 81.04%

Weighted recall = 75.26%

Weighted F1 = 71.04%

The model had high false positives in the ‘benign’ category compared to its competitors. The false positive rate indicates that 39.39% of benign attacks were classified as true threats. This is not an ideal situation for any intrusion detection scenario due to the extra time and money needed to investigate these threats. The model is playing very safe so as not to miss any threats.

**Learning curve:**

The learning curve displays a high training sore indicating that the model may be overfitting the data to begin with. However, with increasing sample size the variance is reduced, and the model fits the training data very well with low bias. There is also a small gap between the learning and training curve indicating a low variance and good generalisation to other input data.

A screenshot of a map

Description automatically generated

Further work on this algorithm could be completed by further tuning of the hyperparameters.

**RANDOM FOREST CLASSIFIER**

The Random Forest classifier is an extension of the decision tree classifier. Several decision trees are collectively used to make predictions. The algorithm is usually trained with the ‘bagging’ method which uses a combination of learning models to increase the overall result.

Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature. The random forest classifier does not overfit data due to its many trees and has a high predictive accuracy (Gluck 2017). The advantages of Random Forest Classifier include their robustness to outliers, their scalability and ability to naturally model non-linear decision boundaries.

The accuracy for random forest was initially 75.24%. The model was very efficient in terms of its speed in fitting and scoring the data.

To improve the accuracy feature engineering was performed. Initially, the ten most important features that best predicted the outcome were selected. However, the accuracy of the model decreased to 75.13%. Following this, the twelve least effective features for predicting classifications were determined. Ten of these features were selected and ‘dropped’ from our list of features. The hyperparameter ‘n\_estimators’ was also changed to 100. The model was named ‘Sorted1’. The accuracy of the algorithm improved to 76.17%, which was the highest accuracy obtained for the Random Forest Classifier.

A validation curve was modelled to determine the optimal value for n\_estimators. ‘N\_estimators’ represents the number of decision trees in the forest that can be refined. Additional decision trees typically improve performance because predictions are made based on a larger number of votes. However, a large number of trees is computationally expensive. The graph demonstrated that both the training curve and validation curve were maximised at a value of n\_estimators ~18-20. However, there is considerable variability about the cross-validation curve indicating that perhaps a value of n\_estimators = 100 would be preferrable.

Contradictory to this curve, higher accuracy was found when n\_estimators was set to 100 rather than 20.

A close up of a map

Description automatically generated

‘Max\_features’ determines the number of features to resample prior to determining the best split. The default value is ‘auto’ which places no restriction on number. The value was changed to ‘log2’ during hyperparameter tuning; however, this reduced accuracy to 74.13%. The default value of ‘auto’ is preferred.

Hyperparameter tuning can be advantageous in creating a model that is better at classification. In the case of random forest, however, it may not be necessary as the algorithms are already good at the task (Meinert 2019).

**Random Forest Table (Name of model: Sorted 1)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attack class | Accuracy | Precision | Recall | F1 | FPR% | FNR% |
| Benign | 77.22% | 65.96% | 97.42% | 78.66% | 38.05% | 2.58% |
| Dos | 92.04% | 96.35% | 79.53% | 87.14% | 1.54% | 20.46% |
| Probe | 94.74% | 85.50% | 61.585 | 71.59% | 1.25% | 38.425 |
| R2l | 89.20% | 97.30% | 5.59% | 10.58% | 0.02% | 94.40% |
| U2r | 99.11% | 60.00% | 1.50% | 2.93% | 0.0089% | 98.5% |

Overall accuracy = 76.17%.

Weighted average precision = 81.88%

Weighted average recall = 76.17%

Weighted average F1 = 70%

**Learning curve**

The learning curve demonstrates that for small amounts of data the model is overfitting data. In the early stages, the training curve is high, there is a large gap between the training score and the cross-validation score indicating a large variance. For sample sizes greater than 30,000, as is the case of our sample size, there is a smaller gap between the two curves indicating more optimal variance. The learning curve has a very high score indicating that the model fits the data well and there is low bias. There is a small degree of variability across the cross-validation curve indicating the model may suffer from a small amount of variance.

A close up of a map

Description automatically generated

To improve the random forest model, further tuning of different hyperparameters could be explored including; ‘max\_depth’, ‘min\_samples\_leaf’ and ‘max\_samples\_split’ .

**KNN**

K - Nearest Neighbours (KNN) is a supervised learning algorithm which has the basic assumption that similar items exist in close proximity. The KNN algorithm is often called a lazy learning model as it has no preconceived ideas about the input data, and it does not learn during the training phase. Instead the model records all of the training data points that are passed and uses this information to make the local generalisations around the test sample during classification (Chio and Freeman 2018)

Parameters that can be modified to optimise the algorithm include the chosen number of neighbours: K. Research has shown that when k is small, bias is low, but variance is high. Conversely, when k is high, there is a smoother decision boundary which means a lower variance but a higher bias.

The advantages of the algorithm include a quick training phase and it performs well with non-linear data. Disadvantages include its slow running time during the testing phase making it more costly to run. Secondly, its suitability is questionable for large dimensional data and it does not work well on rare events (Jaroli 2019).

The KNN model was very slow to run during the testing of this NSL-KDD dataset. In particular tuning the hyperparameters was incredibly slow. A validation curve was used to find the optimal value of neighbours (K) for the algorithm. Error rate was at its lowest when n\_neighbors = 7. This value was then used to train the algorithm which optimised accuracy to 76.31%.

A picture containing different, photo, boat, person

Description automatically generated

**KNN (neighbours = 7)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attack class | Accuracy | Precision | Recall | F1 | FPR% | FNR% |
| Benign | 78.15% | 66.97% | 97.24% | 79.32% | 36.29% | 2.76% |
| Dos | 91.41% | 96.16% | 77.75% | 85.98% | 1.59% | 22.25% |
| Probe | 94.62% | 79.04% | 68.01% | 73.11% | 2.17% | 31.99% |
| R2l | 89.14% | 96.61% | 6.64% | 12.43% | 0.03% | 93.36% |
| U2r | 99.13% | 75.00% | 3% | 5.77% | 0.01% | 97% |

Accuracy = 76.32%

Weighted precision = 81.61%

Weighted recall = 76.32%

Weighted F1 score = 72.62%

**Learning curve:**

The learning curve shows improvement in the learning score as the number of samples is increased. This indicates that the training data is fitted very well with the model and there is a low bias. Furthermore, the gap between the training curve and the learning curve, indicating low variability and good generalisation to new input data.

A close up of a map

Description automatically generated

**ONEVSONE SVC LINEAR CLASSIFIER**

SVM (support Vector Machine) is a learning algorithm mostly used for classification. SVC (Support vector Classification) is a subset of SVM. Both algorithms distinguish between classes by drawing a decision boundary. How to draw or determine the decision boundary is the most critical part of SVM modelling (Chio and Freeman 2018). The models use a mechanism called kernels, which essentially calculate the distance between two observations. A kernel is just a function which takes two data points as inputs and returns a similarity score. This similarity can be interpreted as a metric of closeness. The nearer the data points are, the higher the similarity. The advantage of a kernel is that they can provide similarity scores from higher dimensions without the need for data transformation. In this model a linear kernel has been used. The objective of a linear SVC is to fit the data you provide, returning a “best-fit” hyperplane that divides, or categorises, your data (Harrison, 2020).

Algorithms such as SVM were designed for binary classification datasets and fit a binary classification on each. As the NSL-KDD dataset has a target output of 5 categories , a One-vs-One strategy also needs to be used. One-vs-One is a heuristic method for using binary classification algorithms for multi-class classification. The One-Vs-One approach splits the multi-class classification dataset into multiple binary classification datasets and fits a binary classification model on each (sklearn.multiclass.OneVsOneClassifier – scikit-learn 0.23.2 documentation, 2020). The dataset is split into one dataset for each class versus every other class. Each binary classification model may predict one class label and the model with the most predictions or votes is predicted by the one-vs-one strategy (Brownlee, 2020).

An advantage of the linear SVC model is it scales better to a large number of samples and it has more choice in terms of penalties and loss functions. As data points in a dataset are not always separated linearly the algorithm is able to fit data points in a higher dimensional space in order to make them linearly separated.

To improve the accuracy the error rate for a range of values for the hyperparameter ‘C’ was plotted. The C parameter tells the SVM algorithm to what extent it should avoid misclassifying each training example. For large values of C, the optimisation will choose a smaller margin hyperplane if it is able to fit all the training points correctly. Misclassification will be more easily tolerated if C values are small.

**A picture containing map, table, bunch, covered

Description automatically generated**

The graph above demonstrates that the error rate of the algorithm is least when C is equal to 12. The algorithm was then trained and tested using C=12. The resultant accuracy was 77.03%.

**Linear SVC One-Vs-One Table (Name of model: onevsone2)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attack class | Accuracy | Precision | Recall | F1 | FPR% | FNR% |
| Benign | 78.16% | 66.93% | 97.50% | 79.37% | 36.46% | 2.50% |
| Dos | 91.39% | 97.17% | 76.81% | 85.80% | 1.15% | 23.19% |
| Probe | 96.21% | 90.05% | 72.80% | 80.51% | 0.97% | 27.20% |
| R2l | 89.63% | 95.08% | 9.75% | 17.69% | 0.07% | 90.25% |
| U2r | 98.66% | 13.04% | 9.00% | 10.65% | 0.54% | 91% |

Accuracy = 77.03%

Weighted average precision = 82.39%

Weighted average recall = 77.03%

Weighted average f1-score = 74.02%

The false positive rate for the classification “Benign” was low indicating that threats were less likely to be categorised as “Benign”.

**Learning curve:**

The learning curve displays some very mild variance for a small number of training set sizes. The variance improves as the training size data increases.

A close up of a device

Description automatically generated

**MLP CLASSIFIER**

Artificial neural networks are widely used in machine learning classification tasks. They are inspired by the structure and function of biological neural networks. An ANN consists of an input layer, some hidden layers and an output layer (Chio and Freeman 2018). The predictive capability of neural networks comes from the hierarchical or multi-layered structure of the networks. The data structure can pick out features at different scales or resolutions and combine them into higher order features MLP Classifier is one type of ANN, it stands for multi-layer perceptron classifier. This is a multilayer feedforward artificial network model that maps sets of input data onto a set of appropriate outputs (Brownlee 2016).

GridSearchCV was then used to optimise a list of hyperparameters including; hidden layer sizes, activation, solver, alpha and learning rate. Following this, several values of max\_iter were considered to optimise the algorithm. Max\_iter specifies the number of layers and the number of nodes wanted in the Neural Network Classifier. Optimum ‘max\_iter’ was found to be 40. The optimal parameters were hence assigned to the algorithm and the resultant accuracy was 76.70%.

**MLP classifier Table (Name of model: clf9)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attack class | Accuracy | Precision | Recall | F1 | FPR% | FNR% |
| Benign | 78.46% | 67.40% | 96.85% | 79.48% | 35.46% | 3.15% |
| Dos | 92.49% | 95.15% | 82.02% | 88.10% | 2.14% | 17.98% |
| Probe | 94.13% | 80.92% | 59.35% | 68.48% | 1.68% | 40.65% |
| R2l | 89.48% | 95.11% | 8.31% | 15.29% | 0.06% | 91.69% |
| U2r | 99.14% | 1% | 2.50% | 4.88% | 0% | 97.5% |

Accuracy = 76.85%

Weighted precision = 81.71%

Weighted recall = 76.85%

Weighted F1 score = 73.23%

A disadvantage of the MLP classifier is the high false negative rate for benign classifications when compared with other algorithms (3.15%). The rate is significantly higher than others indicating more true attacks were classified as benign. This would leave a network more vulnerable to attacks, hacks and probing. On the other hand, false positives for the benign class were lower than other models . This would save time and money as investigations for false alarms would be less likely.

**Learning curve:**

The training curve has a high score initially showing that it may be overfitting the data in the early stages. As the sample size increases the two curves converge showing that the model fits the data well and bias is low. For sample sizes under 20,000 the data shows high variance. Sample sizes greater than 20,000 show less variance and will be better generalised to new input data.

A screenshot of a cell phone

Description automatically generated

**COMPARISON OF ALL FIVE CLASSIFICATION MODELS**

**Table 1: Evaluation metrics per classification model**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithms** | Accuracy% | Precision% | Recall% | Weighted F1 % | Average Fit time | Average score time |
| Decision Tree with AdaBoost | 75.26% | 81.04% | 75.26% | 71.04% | 27.53secs | 1.38secs |
| Random Forest | 76.17% | 81.88% | 76.17% | 72.33% | 4.85secs | 0.7secs |
| KNN | 76.32% | 81.61% | 76.32% | 72.62% | 3.54secs | 125.57secs |
| SVC linear | 77.03% | 82.39% | 77.03% | 74.02% | 7.65secs | 0.48secs |
| MLP Classifier | 76.85% | 81.71% | 76.85% | 73.23% | 19.60secs | 0.17secs |

SVC linear had the highest level of accuracy. MLP Classifier was second best. The least accurate algorithm was Decision Tree with AdaBoost.

Although KNN performed well in its detection effect it took a long time to score the data in our cross-validation time experiment. This will lead to a long detection delay in the practical application scenario which will affect the response time of attack classification.

Precision measures how many positive predictions are true. Our model shows that SVC linear is superior at predicting true positives. Recall measures actual positive classes. It indicates how many of the positive classes the model is able to predict correctly. SVC linear again came out as ahead.

The F1 score is the weighted average of precision and recall. This score is useful in cases of uneven class distribution because it takes into account both false positives and false negatives. Our dataset certainly displays an unevenness given that very little intrusions are in class r2l or u2r. SVC linear again performed slightly above the other algorithms with this metric.

The average time to fit each model differed significantly. Using cross-validation results and the ‘fit\_time’ an average score for each algorithm was recorded . The number of iterations (CV) was set to 2. Random Forrest and KNN were both fast to fit the training data.

The score\_time was also recorded. This result was very poor from KNN taking 125.57seconds to score the result. The algorithm as a whole was very slow. Random Forest performed very well on this metric. Overall, Random Forest was the fastest algorithm. This is helpful if there is a lot of incoming requests and the software needs to act quickly to thwart real attacks.

Very interestingly, accuracy and weighted recall were exactly the same for each algorithm.

**Accuracy by class for top 3 algorithms:**

The accuracy by class has been plotted in a side-by-side column graph for the top three algorithms (SVC linear, MLPClassifier and Random Forest Algorithm). KNN has been eliminated due to its slow processing speed, and DecisionTree has been eliminated due to its below average accuracy score.

A screenshot of a cell phone

Description automatically generated

The results are very close. SVC\_linear and MLPClassifier are equal for benign. MLPClassifier is slightly ahead for dos. SVC\_linear is slightly ahead for probe.

**Recall scores: comparison by algorithm and class**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithms | Benign | DOS | Probe | R2l | U2r |
| Decision tree / AdaBoost | 97.09% | 78.30% | 59.35% | 1.44% | 1.50% |
| Random Forest | 97.42% | 79.53% | 61.58% | 5.59% | 1.50% |
| KNN | 97.24% | 77.75% | 68.01% | 6.64% | 3% |
| SVC linear | 97.50% | 76.81% | 72.80% | 9.75% | 9.00% |
| MLP Classifier | 96.85% | 82.02% | 59.35% | 8.31% | 2.50% |

Above is a table listing recall scores by class. The detection effect (recall) of R2l and U2r data is poor which is closely related to the imbalance of sample proportion in the NSL-KDD dataset. Most classifiers did well at recalling benign attacks, however, MLP had a weaker result. The MLP algorithm was more likely to incorrectly label a benign attack as a true attack.

**CONCLUSION**

Overall Random Forest and SVC linear were the preferred algorithms. Both had good accuracy and were timely in their function. SVC linear performed slightly better on accuracy, whereas Random Forest performed slightly better on timing. Both of these algorithms provide overall better results than their competitors.

Future directions for improving our algorithms could include investigating more ensemble learning systems. This would enable two or more algorithms to combine to learn off each-other weaknesses and improve the predictive probability.

**REFERENCES**

Brownlee, J 2016, *Crash Course On Multi-Layer Perceptron Neural Networks*, Machine Learning Mastery, retrieved 21 August 2020, <<https://machinelearningmastery.com/neural-networks-crash-course/>>

Brownlee, J 2019, *How to use Learning Curves to Diagnose Machine Learning*, Machine Learning Mastery, retrieved 21 August 2020,

<<https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>>

Brownlee, J 2020, *Boosting and AdaBoost for Machine learning, Machine Learning Mastery,* retrieved 21 August 2020, <<https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>>

Brownlee, J 2020, *How to Use One-vs-Rest and One-vs-One for multiclass Classification*, Machine Learning Mastery, retrieved 21 August 2020,

<<https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/>>

Chio, C & Freeman, D 2018. Anomaly Detection. In: C. Cho, ed., *Classifying and Clustering*, 1st ed. Sebastapol: O’Reilly Media, pp. 25-77.

Gluck, C 2017, *Running Random Forests? Inspect The Feature Importance’s with this code*, Towards data science, retrieved 21 August 2020, <<https://towardsdatascience.com/running-random-forests-inspect-the-feature-importances-with-this-code-2b00dd72b92e> >

[Accessed 22 August 2020]

Harrison, 2020. *Python Programming Tutorials*. [online] Pytonprogramming.net. Available at: <<https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/>>

Jaroli, H 2020, *K-Nearest Neighbors (KNN) with Python,* datascience+, retrieved 21 August 2020, <<https://datascienceplus.com/k-nearest-neighbors-knn-with-python/>>

Krishni, 2019, An introduction to Grid Search, Medium, Retrieved 21 August 2020,

<<https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998>>

Meinert, R 2019, *Optimizing Hyperparameters in Random Forest Classification,* Towards data science, retrieved 21 August, 2020, <<https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6>>

Scifit-learn.org.2020. *Sklearn.Multiclass.OneVsOneclassifier-Scikit-Learn 0.23.2 Documentation*.[online] Available at: < <https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html>>

[Accessed 22 August 2020]