EARIN
Introduction to Artificial Intelligence

Exercise 4: Regression and Classification

Julia Czmut, 300168
Laura Ploch, 300176

April 2022

1. Introduction

The aim of the exercise was to write a program that predicts whether a packet is a DDoS attack packet or a normal packet. The dataset used for the exercise was NSL-KDD, a set useful for comparing different types of Intrusion Detection System methods. It consisted of almost 126 thousand records in the train set and over 22 thousand records in the test set with 42 attributes describing each record. The creators of the dataset stated that there are no redundant records in the train set, no duplicate records in the test set and the data are therefore balanced to provide good results.

The output of the prediction is an answer whether the packet is a DDoS attack packet, or a normal packet, therefore the problem to be solved is a classification problem, and methods considered will be the ones most suitable for classification problems.

2. Methodology

Preparation of the dataset

The dataset for our models was created using two input files: “KDDTrain+.txt” and “KDDTest+.txt”. The files were constructed in a way such that each line represented one packet. The attributes in each line were separated with a comma. To help us store the data, we have introduced two classes: Packets and Packet.

The Packets class contains a list of all packets within our input data. Elements of this list are objects of the Packet class, which hold a list of their attributes.

We have decided not to remove any columns from the input files, as the purpose of this dataset creation was intrusion detection.

Most of the columns contain numerical data of types int or float, examples of those attributes would be describing: number of source bytes, number of destination bytes, error rates, booleans specifying whether the host or guest was logged in, and many more.

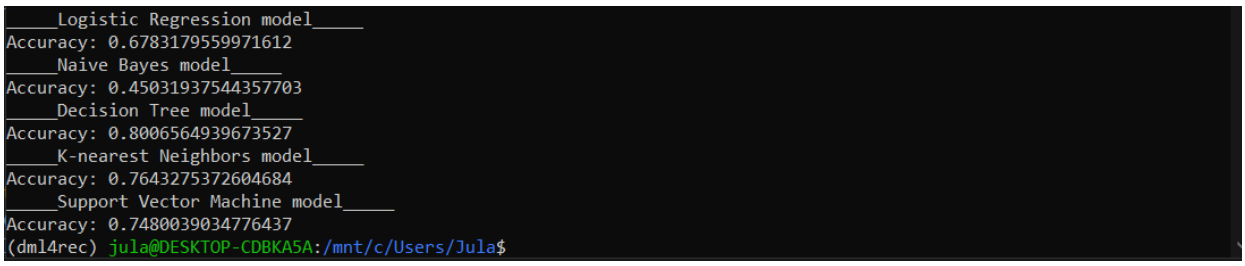
However, there were four attributes which had string values. These were: “class”, “protocol_type”, “service” and “flag”. Each of this attribute had some options for its value. We cannot feed the models string values, so encoding was needed in this case. The “class” attribute was the simplest case – it had only two options: “normal” and “anomaly” and it is the label which our models predict. Because of the binary nature of this attribute and the fact that our goal was to detect attack packets, we used label encoding and encoded “normal” to 0, and “anomaly” to 1. The remaining string attributes had more than two options for their values, so we had to use one-hot encoding. The module we used was from the sklearn library. Each option of those three attributes became a separate feature in our dataset and the ones which were corresponding to this particular packet’s attributes, were set to 1 and the rest was set to 0. This increased the overall number of features in our dataset from 42 to 123.

Having read and encoded the data, we had to split the attributes into features and labels. Features would be our input data X, so all attributes which the model would use to learn to later make a prediction. In our case, that would be all attributes except from “class”. A label (Y) is what we want our model to predict, in our case it is the “class” attribute. At that point, our dataset became a list [features, labels], in which the features were lists of packet attributes, and labels was a list of encoded “class” attribute for all packets.

The preprocessing was the same for the training set and the test set. They were both instances of the Packets class and after splitting to X and Y, the training set had x_train and y_train components, while the test set had x_test and y_test components. They were later used in the models training – when we needed train features and labels, we used x_train and y_train, and when we needed data to test how our models were trained and check how accurate they were, we used x_test and y_test (true labels).

3. Methods

In order to select the two methods to be performed and analyzed, we tested a couple of popular classification algorithms to train the models, namely: Logistic Regression, Naïve Bayes, Decision Tree, K-Nearest Neighbors and Support Vector Machine. Decision Tree and K-Nearest Neighbors and Support Vector Machine models yielded the best accuracies of the prediction, which were 80%, 76% and 74% respectively. While Logistic Regression model produced accuracy of 68% and Naïve Bayes model resulted in accuracy of 45%. Therefore, K-neighbors and Decision Tree methods were selected for further comparison.



Decision tree method

Decision tree is a learning algorithm that breaks the data points into two categories at a time. The root node represents the entire dataset, which further gets divided into two and more sets according to the condition given at the particular decision branch.

The model hyperparameters were tuned using **HalvingGridSearchCV** class from model_selection package from sklearn library. The function performs search over specified parameter values with successive halving, in order to quickly and efficiently determine the optimal values for the model to improve the model's performance. Its strategy is to start evaluating all the candidates with a small amount of resources and iteratively select the best candidates using more and more resources. The parameters put to the testing were: criterion (gini/entropy), split parameter (best, random) and maximum depth (2, 4, 6, 8, 10).

However, testing the parameters found by the searching class we observed that they produced a model yielding worse accuracy than the model trained with default parameters. Moreover, the Precision-Recall metric showed that with a slight recall increase came decreased precision, so we made the decision to use default parameters to keep a higher accuracy.

K-Nearest Neighbors method

K-Nearest Neighbors is a pattern recognition algorithm that finds k members in the training set that are closest related to the evaluated data point from the test set.

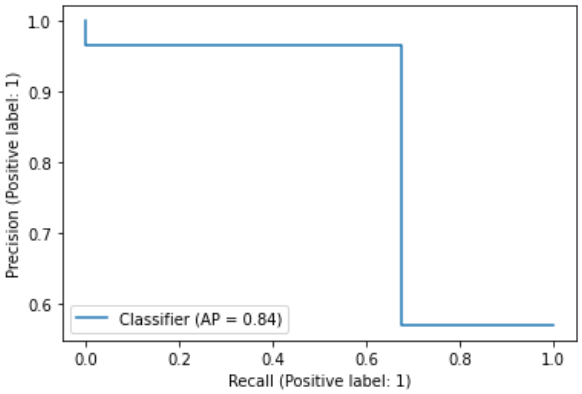
The KNN model hyperparameters were tuned with **RandomizedSearchCV**, another class from sklearn.model_selection package, which performs randomized search on a fixed number of parameter settings, sampled from the specified distributions. The parameters tested were the number of neighbors and the p parameter (1 or 2), which is used in calculating the distance between the evaluated data point and the training data in Minkowski distance formula $d = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$. The class provided us with the number of closest neighbors equal to 12 and p parameter equal to 2.

4. Comparison of the methods

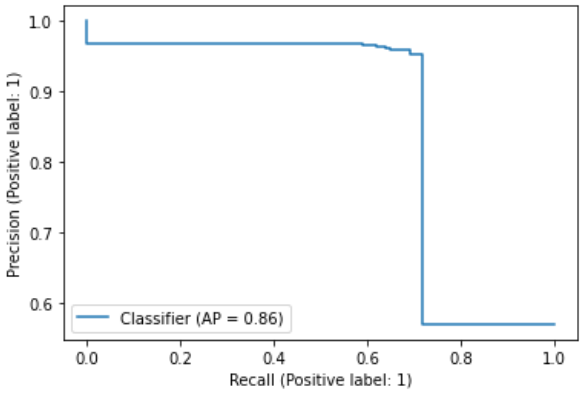
Precision-Recall metric

It is a measure of success of prediction when the classes are very imbalanced. In the graphs below, high area under the curve represents both high precision and high recall, so a situation when there is a low false positive rate and low false negative rate. High scores for both those measures mean that the classifier returns accurate results (high precision) and the majority of all positive results (high recall).

Precision is the ability of the classifier not to label a sample which is negative as a positive, and recall is the ability of the classifier to find all positive samples.



Precision-Recall for **Decision Tree** model



Precision-Recall for **K-Nearest Neighbors** model

Generally, our models have high precision and a slightly lower recall, so they could have found a bigger percentage of all positive cases, but most of the predicted labels are correct.

The Decision Tree classifier yielded lower recall than the K-Nearest Neighbors, however its high precision is very stable. The K-Nearest Neighbors classifier has some instances when recall increases, precision drops slightly, so it means that at that point the model may have got a bit less correct predictions, but it got more of all correct cases overall.

A good Precision-Recall curve is one which has a large area under the curve. In our models, they are both similar to each other, with K-Nearest Neighbors slightly winning because of higher recall.

It is important to remember that the Precision-Recall metric does not consider true negatives, so simply because of that it is not deciding firmly about the quality of the model, because it is not taking all factors under consideration. However, this metric can be used when specificity does not matter. Specificity is the proportion of true negatives that are correctly predicted by the model. In our case, it is more important that our model does not label an actual attack packet as a normal one, than label a normal packet correctly, so specificity does not matter as much as for example precision.

Precision P is defined as the number of true positives T_p over the number of true positives plus the number of false positives F_p.

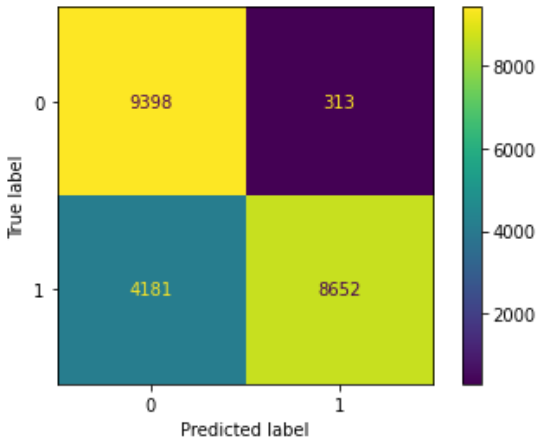
$$P = \frac{T_p}{T_p + F_p}$$

Recall R is defined as the number of true positives T_p over the number of true positives plus the number of false negatives F_n .

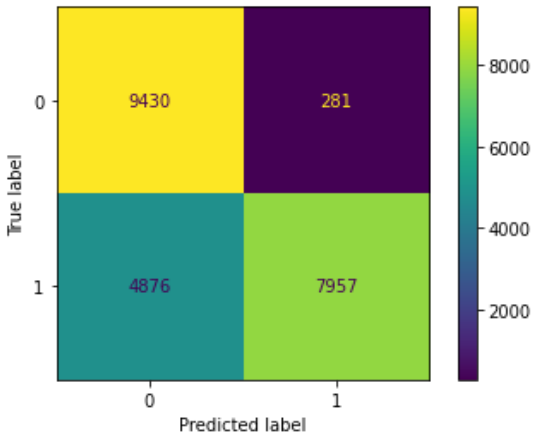
$$R = \frac{T_p}{T_p + F_n}$$

Confusion matrix metric

This metric is a table with 4 different combinations of predicted and actual (true) values.



Confusion matrix for Decision Tree model



Confusion matrix for K-Nearest Neighbors model

Looking at the matrices we can tell that the results of both methods were close, with Decision Tree leading with the number of true positive labels (records that were correctly labelled positive (1 = attack packet) and were indeed attack packets). However, the confusion matrix unravels an unwanted result, that is a high number of undetected attack packets. Although the number of packets that were neutral but were labelled as attack packets is very small in case of both methods.

Classification report metric

This metric generates a text report with main classification metrics. Precision and recall were discussed earlier. They are related to F1 score, which is defined as the harmonic mean of precision and recall. F1 metric tells us what percent of positive predictions were correct, so considering this metric, the Decision Tree model performed better, because F1 score is larger.

$$F1 = 2\frac{P \times R}{P + R}$$

```
print(classification_report(y_test, predictions_kneighbors, target_names=['normal', 'anomaly']))
```

```
              precision    recall  f1-score   support

      normal       0.66      0.97      0.79      9711
     anomaly       0.97      0.62      0.76     12833

    accuracy                           0.77     22544
   macro avg       0.81      0.80      0.77     22544
weighted avg       0.83      0.77      0.77     22544
```

Classification report for K-Nearest Neighbors

```
print(classification_report(y_test, predictions_decisiontree, target_names=['normal', 'anomaly']))
```

```
              precision    recall  f1-score   support

      normal       0.69      0.97      0.81      9711
     anomaly       0.97      0.67      0.79     12833

    accuracy                           0.80     22544
   macro avg       0.83      0.82      0.80     22544
weighted avg       0.85      0.80      0.80     22544
```

Classification report for Decision Tree