Data Science Toolbox Assessed Coursework 1

Kate Staniewicz, Samantha Wise, Junfan Huang

January 2, 2022

1 Introduction

This is a three person project working on creating model submissions, with the goal of predicting normal vs non-normal traffic.

As stated in the assignment brief, modelling normal versus non-normal traffic is an open-ended problem, however the group decided to interpret this as a classification problem. Our approach assessed and compared various machine learning algorithm for intrusion prediction systems. In a competition setting, our goal would be to find the best suitable machine learning algorithm which can predict the type of network attack with highest accuracy and then develop a system which uses this algorithm to detect network intrusion. The algorithms which we have compared are Random Forest Classification (by Samantha), combination of K-means and K-Nearest Neighbours (by Junfan) and Logistic Regression (by Kate).

This paper is outlined as follows: in section 1 we briefly go over the steps we took throughout the assignment. In section 2 we give a rough explanation of why each of the models were chosen and how they work. In section 3, we compare the performance of our models on the KD99 dataset. In section 4, we give a conclusion of our remarks and future directions of this work.

GitHub repo can be found:
`https://github.com/samanthawise/dtsassignment2`

1.1 Log

1. (06/11/2018) Group met for an hour to discuss our initial approach to the problem and mutually decided to do further research on one machine learning model:

    - Kate: Nearest Neighbours
    - Junfan: K-means
    - Samantha: Random Forest

2. (07/11/2018) We explored the internet to find similar examples and codes we could adapt and we decided on the following examples:

- K-means: `https://github.com/jadianes/kdd-cup-99-spark`
- Random forest: `https://github.com/ghuecas/kdd99ml/blob/master/detectAttack.py`
- Naive Bayes: `https://github.com/chadlimedamine/kdd-cup-99-Analysis-machine-learning-`

We spent the rest of our debugging the scripts.

3. (09/11/2018) We finalized our model decisions to be:

   - Junfan using a combination of Nearest Neighbours and K-means (Matlab and Python).
   - Kate using Logistic Regression (R).
   - Samantha using Random Forest (R).

   We decided to individually work on building up our models following the algorithms we chose.

4. We met with the lecturer and reach a decision our prediction task: leaving our two types of attacks for testing and using the rest of the attack classes for training. This involved randomly selecting a number of *normal.* data to include in both training and testing sets.

   We found that this approach was not successful as the training data size was too large for our models to compute, so we had to brainstorm how to change our approach.

5. (14/11/2018) We decided to approach the training and testing as follows:

   - Randomly splitting the data into 50% training and testing.
   - Splitting the data into 70% for training where 20% of it is *normal.* data and the rest is *smurf.* data, the testing data contained 30% of the total data size with 20% of it being *normal.* data and 80% being the rest of the attack types.
   - Same as above, except training contained all the different attacks and testing contained the *smurf.* type of attack only.

6. (19/11/2018) Group met to discuss how to compare our models. Began writing up the report.

2 Models explanation

2.1 Logistic Regression

For our first model we have chosen regression, because it is a widely used method by scientists. However, classical regression requires continuous variables, which would be difficult to implement, since the data we have contains categorical and

numerical variables. Therefore, we have chosen the logistic regression, which is a generalized linear model that allows mixed variables. The response variable in the logistic regression model is binary. For this model, the response 1 and 0 corresponds to, respectively, the normal and the abnormal traffic. The logistic regression estimates the parameters of the model. This means that it assigns the significance of each variable to our prediction.

2.2 Random Forest

Random Forest algorithm is a classification algorithm based on ensemble learning. An ensemble-learning model works by aggregating multiple Machine Learning models to reduce error and improve performance. Each of the models (the decision trees), when used on their own, is weak. However, when used together in an ensemble (random forest), the models are strong, and therefore produce more accurate results.

A random forest trains each decision tree with a different subset of training data. Each node of each decision tree is split using a randomly selected attribute from the data. This element of randomness ensures that the Machine Learning algorithm creates models that are not correlated with one another. As a result, potential errors are evenly spread throughout the model and are cancelled out by the majority voting decision strategy of the model. [1]

2.3 Combination of Nearest Neighbours and K-means

2.3.1 K Nearest Neighbors

Another useful model for classification is K nearest neighbors. In this project, we chose to focus on only 1 nearest neighbor. What is the 1 nearest neighbor algorithm? The goal of the algorithm 1-nearest-neighbor is to find the nearest neighbor and then give the same label with the nearest neighbor. This classifier is commonly based on the Euclidean distance between a test sample and the specified training samples. [2]

One of the benefits of the nearest neighbors algorithm is that training is not necessary. We just need to randomly choose the samples as the training set and calculate the distance between the test set. However, for our project problem, we have nearly 500,000 data. If we choose 50% data for training and 50% for testing, we will calculate at least $250,000^2$ times to find the nearest one. In that case, it is necessary to decrease the training size. But how to decrease the training size and keep the samples' features? K-means might be a good choice.

2.3.2 K-means

K-means clustering is to find the centers of the data automatically. In our project, it can be use to represent the whole training dataset. Specifically, we could use K-means to find k_1 points for normal traffic and k_2 points for

non-normal. After finding the centers for the dataset, we just need to calculate $250,000(k_1+k_2)$ times to find the nearest neighbor which made it more feasible.

3 Analysis and Evaluations

In this project, the aim is to build a network intrusion detector to classify normal and abnormal network connections. In order to compare our models performance, models' simulations was divided into 4 parts, namely one-side holdout test, smurf training and predicting, dimensionality reduction and visualization.

3.1 One-side holdout test

In order to evaluate the models performance, we first randomly divide the data into *training set* and *validation set*, which respectively occupies one half. We used training set for model construction and validation set for performance evaluation. Through this partition, we was able to get two recognition rates which called **accuracy** in the following report. One is inside accuracy (testing the training set), the other is outside accuracy (testing validation set). It is worth noting that the accuracy in section 4 means outside accuracy. Although this method is easy to achieve, it is highly affected by data partitioning. [3] An improved method is two-side holdout test, which does an role reversal by using validation set for training and training set for validation. We just only focus on one-side holdout test in our project. Because of its simplicity, it also called basic method in the section of performance evaluation.

3.2 Smurf training and predicting

Analysing the KD99 dataset, it is easy to see that the *smurf.* attack is the most significant attack, taking up nearly half of our dataset. In that case, it would be very interesting to figure out if our classifiers are able to predict the *smurf* traffic successfully by training the dataset without *smurf* events or do a role reversal to predict other attacks training with *smurf* only. In this simulation, we divided the data into two parts. One dataset contains the *normal.* and *smurf* connections, the other dataset contains the *normal* and other attack except *smurf* connections.

3.3 Dimensionality reduction

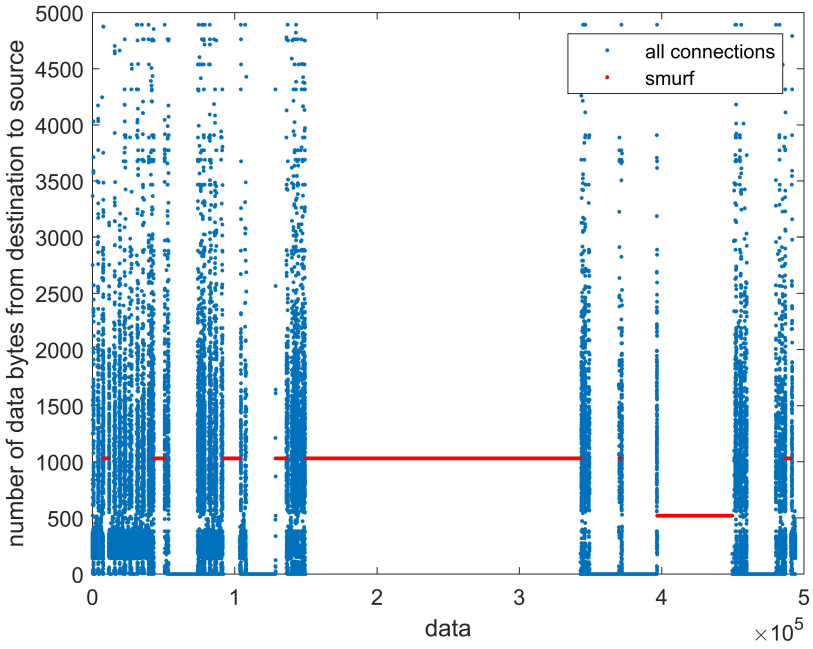

Figure 1: Comparison of the number of data bytes from destination to source

In this graph, it is obvious that in this dimension (dst_bytes) the *smurf's* behavior has a high identifiability indicating that dimensionality reduction is possible. So if it is doable, how to reduce our dimensions? Do we need to test all the combinations? Let's calculate how many times we would use, if we want to choose k dimensions from these 41 dimensions by testing all the possible combinations. Assuming that $k = 10$, which means $\binom{41}{10} = 1.1211e + 09$ times calculating which is obviously unfeasible. Is there any other way to reduce the dimensions? Drawing out all the data's distribution, we can find something interesting.

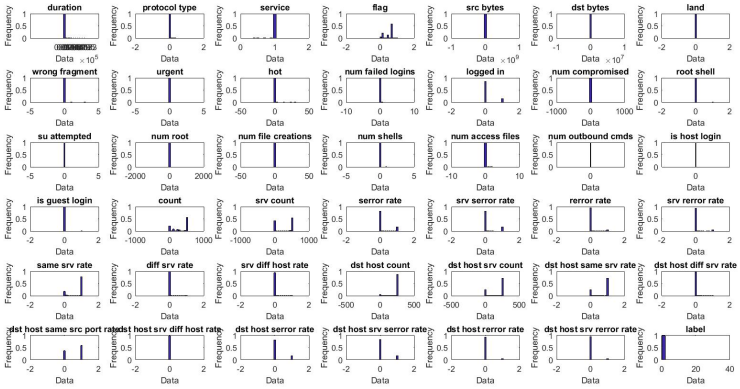

Figure 2: Data's distribution from different dimensions

From this graph, we could see that some dimensions concentrate on some certain values, others do not. These features, which have no significant difference, might be useless for training. Let's take the 1 nearest neighbor as an example. We have n dimensions and what we really care about is the distance. In Euclidean distance, the distance between the training and testing data would be:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

where $\mathbf{x}$ is the points from training set, $\mathbf{y}$ is the points from testing set.

For those dimensions which concentrate on some certain values, they usually have a low standard deviation. In that case, the data points tend to be close to the mean and also close to each other. For example, σ_i is the standard deviation of the ith dimension:

$$\sigma_i = \sqrt{\frac{\left(y_i^1 - \mu\right)^2 + \left(y_i^2 - \mu\right)^2 + \ldots + \left(y_i^N - \mu\right)^2}{N}}$$

where N is the size of the dataset and μ is the average of the this dimension.

Without loss of generality, let $i = 1$, in the distance function it will be $(x_1 - y_1)^2$. In standard deviation formula, the set of $\{y_1^j\}$ represents every point in the dataset which contains x_1, y_1, so the σ could change to this form:

$$\sigma_1 = \sqrt{\frac{(x_1 - \mu)^2 + (y_1 - \mu)^2 + \left(y_1^3 - \mu\right)^2 + \left(y_1^4 - \mu\right)^2 + \ldots + \left(y_i^N - \mu\right)^2}{N}}$$

$$(x_1 - \mu)^2 + (y_1 - \mu)^2 = N\sigma_1^2 - \left(\left(y_1^3 - \mu\right)^2 + \left(y_1^4 - \mu\right)^2 + \cdots + \left(y_1^N - \mu\right)^2\right)$$

Let

$$\hat{\sigma_1} = \sqrt{\frac{\left(y_1^3 - \mu\right)^2 + \left(y_1^4 - \mu\right)^2 + \ldots + \left(y_i^N - \mu\right)^2}{N-2}}$$

Then $\left(y_1^3 - \mu\right)^2 + \left(y_1^4 - \mu\right)^2 + \cdots + \left(y_1^N - \mu\right)^2 = (N-2)\,\hat{\sigma_1}^2$
So

$$(x_1 - \mu)^2 + (y_1 - \mu)^2 = N\sigma_1^2 - (N-2)\hat{\sigma_1}^2$$

From the mean inequality, it follows that,

$$2[(x_1 - \mu)^2 + (y_1 - \mu)^2] \geq (x_1 - y_1)^2$$

Now,

$$(x_1 - y_1)^2 \leq 2(N\sigma_1^2 - (N-2)\hat{\sigma_1}^2)$$

because N is a large number, we could assume $\hat{\sigma_1} \approx \sigma_1$.
We get:

$$(x_1 - y_1)^2 \leq 4\sigma_1^2$$

In that case, $(x_1 - y_1)^2$ is a really small number which could be ignored when σ_1 is small enough(i.e. $\sigma_1 \leq 0.01$).

By testing different thresholds, we get different results. After series of simulations, we chose 0.9 as our threshold and reduced into 10 dimensions namely duration, *duration, src_bytes, dst_bytes, num_compromised, num_root, count, srv_count, dst_host_count, dst_host_srv_count* (including label) through which could also get a good result(as shown in the performance section).

3.4 Visualization

For the purpose of obtaining a deeper understanding of the data, we have visualized the dataset by using *tsne* toolbox. [4] Figure 3 shows the distribution of whole dataset, from which it is easy to recognize the *smurf* events (green). In this graph, a red point represents a normal connection surrounding by the three big *smurf* cycles.

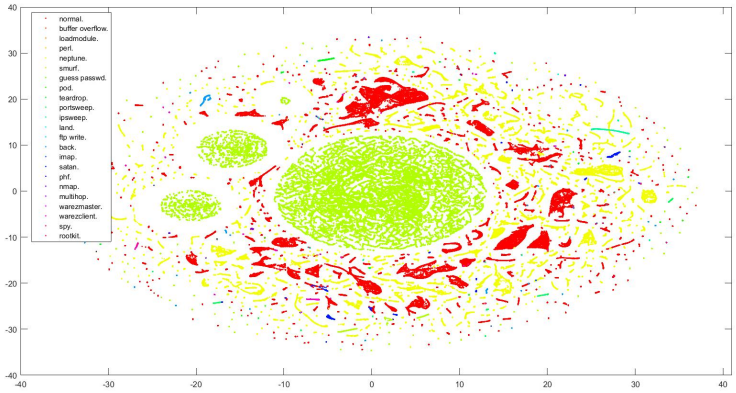

Figure 3: Visualization with 23 labels

In this project, we care only about the classification of the normal and abnormal connections, so the next two graphs show the real data visualization and one of our predictions.

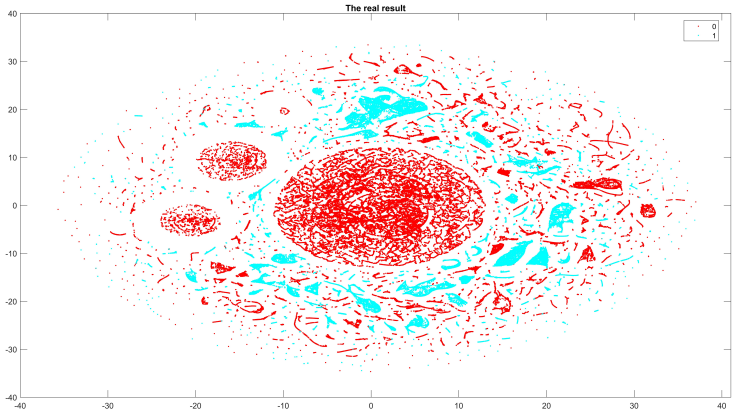

Figure 4: The visualization of the original data with 2 labels

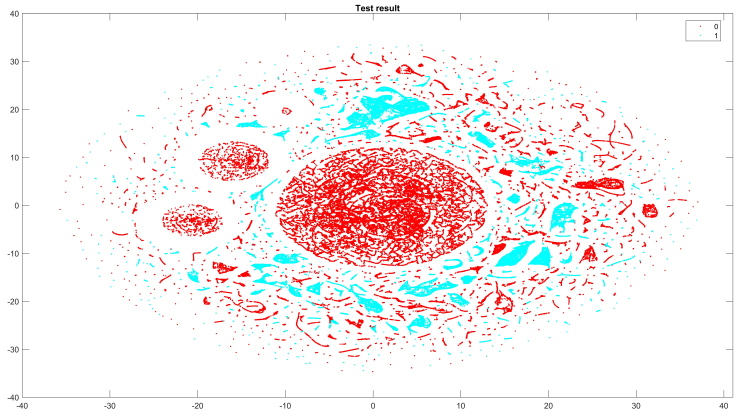

Figure 5: The visualization of predicted data with 2 labels

We are able to find a few differences between the real one and predicted one, which are showing where the wrong prediction is being made.

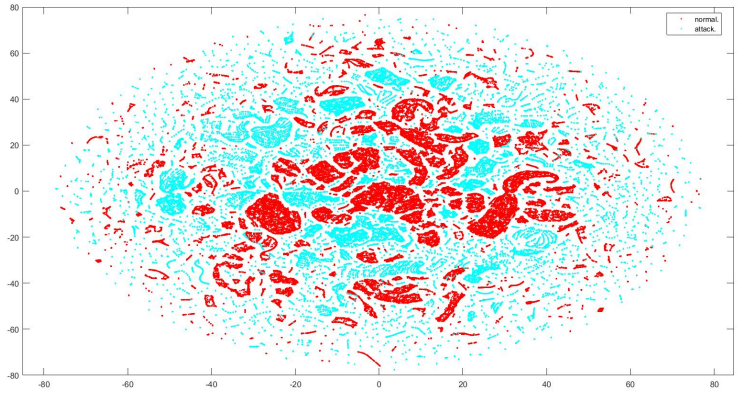

Figure 6: The visualization of the original data without *smurf*

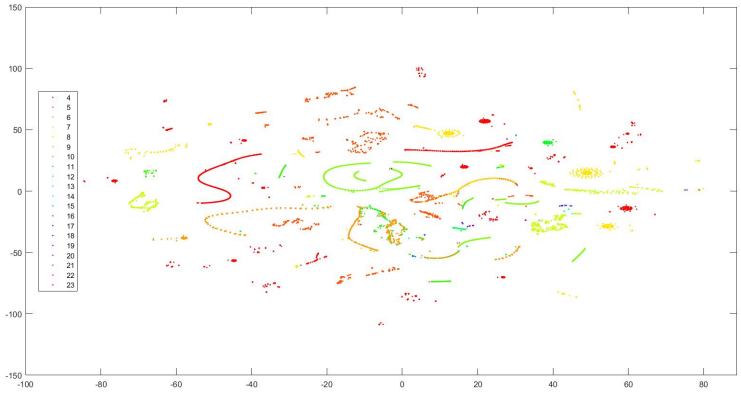

Figure 7: The visualization of the original data without *smurf, neptune* and *normal* connections

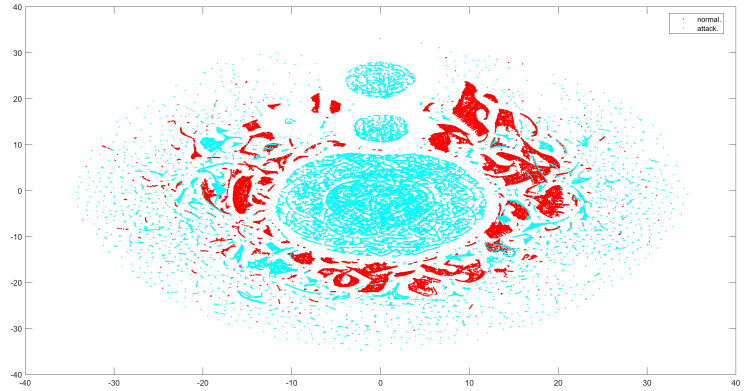

Figure 8: Dimensionality reduction

Figure 8 shows that after reducing the dimensions, the graph as a whole remains stable compared to the Figure 4.

4 Performance Table

Determining the best model is not a trivial task. Various models can be better in certain aspects, but fail to perform at some tasks. Depending of the desired features of the model we compare their performances. In principal, our aim was to train models to obtain the highest possible accuracy of predictions, while

avoiding the over-fitting problem. We created a performance table that compares our models based on accuracy, False Positive and False Negative type of errors, F1 scores and time for a model to run. We explain those in more details with the interpretation of the results below.

- The accuracy is being computed by

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

  TP stands for True Positive, which in our case means that the abnormal data was predicted as abnormal. TN (True Negative) means that normal data was predicted as normal. The Random Tree model performs the best at the accuracy, when we train all types of attacks. All of the models perform fairly well even when we reduced the number of variables to 10. Only the Logistic regression model is able to correctly recognize the smurf traffic as abnormal. All models failed to detect abnormal traffic when we used the smurf data for training.

- The **False Positive** type of error occurs when normal data is being classified as abnormal. It is an undesirable kind of error because we do not want to interrupt the normal traffic.

$$\text{False Positive Rate} = \frac{FP}{FP + TN}$$

- The **False Negative** errors means that the abnormal data was predicted as normal. It is important to keep in mind that high rate of False Negative error means a potential danger, since the abnormal traffic is not being detected.

$$\text{False Negative Rate} = \frac{FN}{FN + TP}$$

- **F1 Score** is a measure to check the accuracy of the binary classification models.

$$\text{F1 score} = 2\,\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

  Precision is the ratio saying how much of the classified abnormal data were actually abnormal.

$$\text{Recall} = \frac{TP}{TP + FN}$$

  Recall is a ratio of how much data was correctly reported as abnormal and how much data was predicted correctly.

- We measured in seconds how much **time** does it take to train each model. It is important to note that each model was trained different computers. We believe the actual time is longer but the computer fails to report it, so we do not want to make any claims regarding time as the performance measure. Ideally, we could have computed the computation complexity for each model. That would work in favour for some model in term of efficiency.

Performance						
Model	**Variables**	**Accuracy**	**FP rate**	**FN rate**	**F1 score**	**Time**
Random Forest	Basic	0.9997	0.00024672	0.00006051	0.99971510	12.0371s
	Smurf Predict	0.0922	0.00021341	1.00000000	0.00000000	18.1229s
	Smurf Train	0.1957	0.00000000	0.00000000	0.00127557	13.5596s
	10 variables	0.9996	0.00053456	0.00013111	0.99973280	5.6064s
Regression	Basic	0.9958	0.01359131	0.00184498	0.9974134	1.0447s
	Smurf Predict	0.9987	0.00597546	0.00015182	0.9992161	1.2932s
	Smurf Train	0.2026	0.01372933	0.9874432	0.02472091	25.7937s
	10 variables	0.9835	0.00532549	0.01922108	0.9896439	3.5908s
K-means&KNN	Basic	0.9954	0.01585527	0.00176932	0.9954	15.648s
	Smurf Predict	0.1102	0.8897	0.9600	0.00008615	3.9055s
	Smurf Train	0.1125	0.8889	0.1373	0.0038	4.1104s
	10 variables	0.9752	0.02012863	0.02578245	0.9748	3.8500s

Table 1: 'Basic' model includes all 41 variables. For 'Smurf Predict' we used the normal data and the abnormal data expect the smurf one for training. 'Smurf Train' takes the smurf and normal data for training. In '10 variables' we take for parameters: *duration, src_bytes, dst_bytes, num_compromised, num_root, count, srv_count, dst_host_count, dst_host_srv_count.*

5 Discussion and Conclusions

In this project, we presented and compared the performance of three different models, namely: Logistic Regression, Random Forests and a combination of Nearest Neighbours & K-means, that aims to predict abnormal traffic. In terms of accuracy, the Random Forest model scored the highest, even when restricting the number of variables in the dataset. The accuracy drops rapidly when we tried to predict abnormal data based on data concerning normal and smurf attack data. Poor results are what we expected, since the smurf data constitutes a large part of the entire dataset. This suggests that it is not easy to predict a new type of the attack, unless it has similar properties to already known types of attacks.

6 Bibliography

References

[1] Bahnsen A. (2017). Machine Learning Algorithms: Introduction to Random Forests. http://www.dataversity.net/machine-learning-algorithms-introduction-random-forests/

[2] Peterson L. (2013). K-nearest neighbor. http://scholarpedia.org/article/K-nearest_neighbor

[3] Jang J. (2017). Performance Evaluation: Accuracy Estimate for Classification and Regression. http://u.camdemy.com/media/18687

[4] van der Maaten L. (2018). t-SNE. https://lvdmaaten.github.io/tsne/