Data Science Toolbox Assessment 2
Boosting algorithm based on tree model

Linlan Wang	Yewei Yuan	Tianhang Guo
fa19847	bf19915	ci19815

December 2019

Equity

- ☐ Linlan Wang: GBDT & Futher Research
- ☐ Tianhang Guo: Adaboost & Baseline
- ☐ Yewei Yuan: XGBoost & Data Processing

Contents

1 Introduction

In the previous assessment, we found that for the kdd_99 dataset, the classification algorithms are better than the clustering algorithms, and the tree-based classification algorithms are better than the linear algorithms, where random forest performs the best. In order to explore this problem more deeply, in this assessment, we made the following changes:

(1) We focused on tree algorithms. Random forest is a kind of Bagging-type Ensemble Learning tree algorithm. Through searching references, we found that the algorithm of Boosting Tree has an excellent performance in various experiments and competitions. The Boosting-type Ensemble Learning tree algorithms have value to be explored;

(2) We extended the problem of binary classification. No longer considered "dos", "probe", "u2r" and "r2l" as "abnormal" classes, but separated them and did classification in five categories together with "normal";

(3) Because the data set is very good, no significant difference in the results brought by optimizing algorithms was found in the experiment, so we analyzed the important features selected by Tree-Ensemble algorithms in detail.

The purpose of this assessment is to expand the content in courses, understand the mathematical principles of Tree-Ensemble algorithms, explore the changes of mind from binary classifications to multiple classifications, and finally after a series of analysis, can intuitively determine the type of a new attack through a single or few features .

2 Data Processing

(1) Read the data, check for missing values and remove if there are some

(2) Use kddnames.txt to **classify various attacks** into 5 categories: "normal", "dos", "probe", "u2r", "r2l"

(3) Changing nominal variables into numerical variables
This time we didn't use binary encoding, but directly number the nominal variables. The reason we binarize and use one-hot/dummy encoding is because we want some meaningful distance relationship between the different values.
Most of parametric models (generalized linear models, neural network, SVM, etc.) or methods using distance metrics (KNN, kernels, etc.) will require careful work to achieve good results. But some methods require almost no efforts to normalize features or handle both continuous and discrete features, like tree-based methods: Cart, Random Forest, bagging or boosting.

(4) Establish **random forest model** to look for important features

(5) Loop through the random forest five times and select the first few vectors with the sum of importance ratio of 95%. After experiments, 19 vectors were selected.

The reason for repeating the experiment five times is that due to the random property of the random forest,

the results produced each time are different. We want to retain 95% of the information, each experiment has about 20 significant features that can meet the requirements, so after repeating the experiment five times, we choose the intersection that retain 95% of the information to get the final feature sequence with 19 features.

(6) Split the data set with 70% of the training set and 30% of the test set

3 Boosting

Let's first talk about Boosting algorithm. Boosting is an ensemble modeling technique of converting a set of weak learners into strong learners. In general, it is relatively easy to find weak learner, and then repeatedly learn to get a series of weak classifiers and combine these weak classifiers to get a strong classifier. There are many boosting algorithms. The main variation between them is their method of weighting training data points and hypotheses. The different types of boosting algorithms are:

- AdaBoost
- Gradient Boosting
- XGBoost

These three algorithms have gained huge popularity, especially XGBoost, which has been responsible for winning many data science competitions.

3.1 Baseline

A decision tree makes predictions based on a series of questions. The outcome of each question determines which branch of the tree to follow. They can be constructed manually (when the amount of data is small) or by algorithms, and are naturally visualized as a tree.

The decision tree is typically read from top (root) to bottom (leaves). A question is asked at each node (split point) and the response to that question determines which branch is followed next. The prediction is given by the label of a leaf.

The reason why we use decision tree as our baseline is because of these several reasons, including:

- The tree output is easy to read and interpret
- They are able to handle non-linear numeric and categorical predictors and outcomes
- Decision trees can be used as a baseline benchmark for other predictive techniques
- They can be used as a building block for sophisticated machine learning algorithms such as random forests and gradient-boosted trees[2]

The [Figure 1] is the process of Decision Tree.

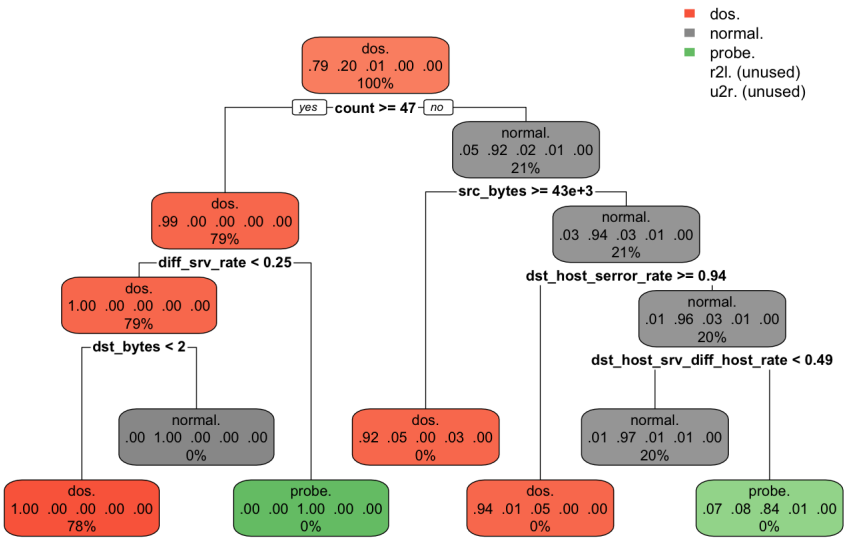

Figure 1: The process of Decision Tree

3.2 AdaBoost(Adaptive Boosting)

AdaBoost is a typical Boosting algorithm and belongs to the Boosting family. Due to different loss functions, there are different types of Boosting algorithms. AdaBoost is a Boosting algorithm whose loss function is exponential loss. AdaBoost works on improving the areas where the base learner fails. The base learner is a machine learning algorithm which is a weak learner and upon which the boosting method is applied to turn it into a strong learner.[1]

The strong learner obtained by AdaBoost is a linear combination of a series of weak learners. This is the additive model:

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

where $G_m(x)$ is the base leaner and α_m is the coefficient.

The loss function used by AdaBoost is exponential, which has the form:

$$L(y, f(x) = e^{-yf(x)}$$

3.2.1 Algorithm[3]

Input: training data set $T = \{(x_1, y_1), (x_2, y_2), (x_N, y_N)\}$, where $x_i \in X \subseteq \mathbb{R}^n$, number of iterations M.
1. Initialize the weight distribution of training samples: $D_1 = (w_{1,1}, w_{1,2}, ..., w_{1,i})$, $w_{1,i} = \frac{1}{N}$, $i = 1, 2, ..., N$.
2. For $m = 1, 2, ..., M$:
(a)Use the training data set with weight distribution D_m to learn and get the base classifier $G_m(x)$.
(b) Calculate the classification error rate of $G_m(x)$ on the training data set:

$$e_m = \frac{\sum_{i=1}^{N} w_i^{(m)} \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} e_i^{(m)}}$$

(c) Calculate the weight of $G_m(x)$ in the strong classifier:

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m}$$

(d) Update the weight distribution of the training data set (here, z_m is the normalization factor, in order to make the probability distribution of the sample sum to 1):

$$w_i^{(m+1)} = \frac{w_i^{(m)} e^{-y_i \alpha_m G_m(x_i)}}{z^{(m)}},\ i = 1, 2, ..., N$$
$$z_m = \sum_{i=1}^{N} w_i^{(m)} e^{-y_i \alpha_m G_m(x_i)}$$

3. Obtain the final classifier:

$$G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$$

3.2.2 Significant features

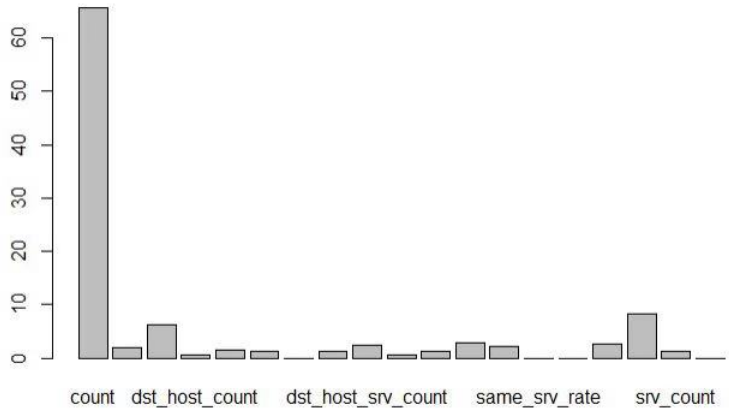

Figure 2: Significant features of Adaboost

3.3 Gradient Boosting

AdaBoost uses exponential loss function. The disadvantage of this loss function is that it is very sensitive to outliers. As a result, it usually performs poorly on datasets with more noise. Gradient Boosting has been improved in this regard, so that any loss function can be used (as long as the loss function is continuously differentiable), so that some more robust loss functions can be applied, making the model more resistant to noise.

The basic idea of Boosting is to make each iteration of the base learner pay more attention to the samples of the previous iteration of learning errors in the training process. The difference is the methods. AdaBoost uses a strategy of increasing the weight of the wrong samples in the previous iteration of learning. While in Gradient Boosting, the negative gradient is used as a measure of the errors of the base learner in the previous iteration. In the next iteration of learning, the errors made in the previous iteration are corrected by fitting negative gradients. The key question here is: Why can the previous iteration of errors be corrected by fitting a negative gradient? The answer given by the inventor of Gradient Boosting is: functional gradient descent algorithms.

3.3.1 Functional Gradient Descent Algorithms

A major step in machine learning is to minimize the loss function $L(\theta)$ through an optimization method, and then find the corresponding parameter θ. Gradient descent is a classic optimization method, and its parameter update formula is given by

$$\theta = \theta - \alpha \cdot \frac{\partial}{\partial \theta} L(\theta) \tag{1}$$

where α is the learning rate.

Gradient Boosting uses the same additive model as AdaBoost. In the m^{th} iteration, the first $m-1$ base learners are fixed, that is,

$$f_m(x) = f_{m-1}(x) + \rho_m h_m(x) \tag{2}$$

Therefore, in iteration m, our goal is to minimize the loss function $L(f) = \sum_{i=1}^{N} L(y_i, f_m(x_i))$, and then obtain the corresponding base learner.

If $f(x)$ is used as a parameter, gradient descent can also be used:

$$f_m(x) = f_{m-1}(x) - \rho_m \cdot \frac{\partial}{\partial f_{m-1}(x)} L(y, f_{m-1}(x)) \tag{3}$$

By comparing equation (5) and (6), it can be found that if $h_m(x) \approx \frac{\partial L(y, f_{m-1}(x))}{\partial f_{m-1}(x)}$, that is, the negative gradient of the loss function in the previous iteration is fitted by the base learner $h_m(x)$, then $L(f)$ is minimized by the gradient descent method. Since $f(x)$ is actually a function, this method is considered as a functional gradient descent algorithm.

Negative gradients are also called pseudo residuals, and as the name suggests, they are a concept close to residuals. Intuitively, the larger the residual $r = y - f(x)$, indicating that the result of the previous iteration of learner $f(x)$ differs greater from the real value y, then the next iteration of learners can fit the negative gradient, and be able to correct the errors of the previous learner.

3.3.2 Algorithm[4]

Input: training set $\{x_i, y_i\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.
Algorithm:

1. Initialize model with a constant value: $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
(a) Calculate the pseudo residual: $\tilde{y_i} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$, $i = 1, 2, ..., N$.
(b) By minimizing the square error, we use the base learner $h_m(x)$ to fit $\tilde{y_i}$:
$$w_m = \arg\min_{w} \sum_{i=1}^{N} [\tilde{y_i} - h_m(x_i; w)]^2$$
(c) Use line search to determine the step size ρ_m to minimize L:
$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \rho h_m(x_i; w_m))$$
(d) Update the model: $f_m(x) = f_{m-1}(x) + \rho_m h_m(x; w_m)$
3. Output $f_M(x)$.

3.3.3 GBDT(Gradient Boosting Decision Tree)

In the framework of Gradient Boosting, the most commonly used base learner is a decision tree (usually CART). The combination of these two gives the well-known Gradient Boosting Decision Tree (GBDT) algorithm.

3.3.4 GBDT Algorithm[5]

1.Initialization: $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$.

2. For $m = 1$ to M:

(a) Calculate the pseudo residual: $\tilde{y}_i = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$, $i = 1, 2, ..., N$.

(b)$\{R_{jm}\}_1^J = \arg\min_{\{R_{jm}\}_1^J} \sum_{i=1}^{N} [\tilde{y}_i - h_m(xi; \{R_{jm}, b_{jm}\}_1^J]^2$

(c) $\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$

(d) $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} I(x \in R_{jm})$

3. Output $f_M(x)$.

3.3.5 Significant features

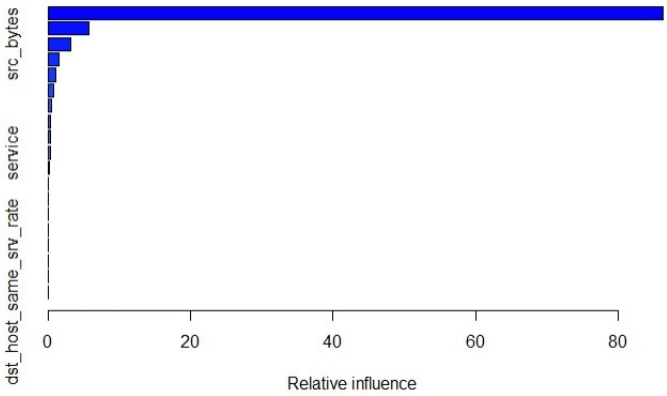

Figure 3: Significant features of GBDT

3.4 XGBoost(Extreme Gradient Boosting)

XGBoost is often used in some competitions, its effect is significant. It is a tool for massively parallel boosted tree. It is currently the fastest and best open source boosted tree toolkit. The algorithm applied by XGBoost is an improvement of GBDT (gradient boosting decision tree), which can be used for both classification and regression problems.

Generally speaking, the improvement in principle of XGBoost is mainly on the loss function. Firstly, adding a regularization term to the original loss function gives a new objective function, which is similar to pruning each tree and limiting the score on the leaf nodes to prevent overfitting. Secondly, CGBoost performs a second-order Taylor expansion on the objective function and optimize it.

3.4.1 Algorithm

1. Initialization $f_0(x)$.

2. For $m = 1$ to M:

(a) Calculate the first derivative of loss function in every training sample $g_i = \frac{\partial L(y_i, \hat{y_i}^{m-1})}{\partial \hat{y}^{(m-1)}}$ and the second

derivative $h_i = \frac{\partial^2 L(y_i, \hat{y_i}^{m-1})}{(\partial \hat{y}^{(m-1)})^2}$, $i = 1, 2, ..., N$.
(b) Generate a decision tree $f_m(x)$ recursively using a tree splitting algorithm.
(c) Add the new tree to the model, $\hat{y_i}^{(m)} = \hat{y_i}^{(m+1)} + f_m(x)$.

3.4.2 Differences between XGBoost and GBDT

1) Traditional GBDT uses CART as the base classifier, and XGBoost also supports linear classifiers. At this time, XGBoost is equivalent to logistic regression (classification problem) or linear regression (regression problem) with L1 and L2 regularization terms.

2) Traditional GBDT only uses first-order derivative for optimization, and XGBoost performs second-order Taylor expansion of the loss function, and uses first- and second-order derivatives at the same time.

3) XGBoost adds a regularization term (regularizer) to the loss function to control the complexity of the model. The regularization term reduces the variation of the model, makes the learned model simpler, and prevents overfitting. This is also a feature of XGBoost that is superior to traditional GBDT.

4) XGBoost borrows from the practice of random forest and supports column sampling, which not only reduces overfitting, but also reduces calculations. This is also a feature of XGBoost that is different from traditional GBDT.

3.4.3 Significant features

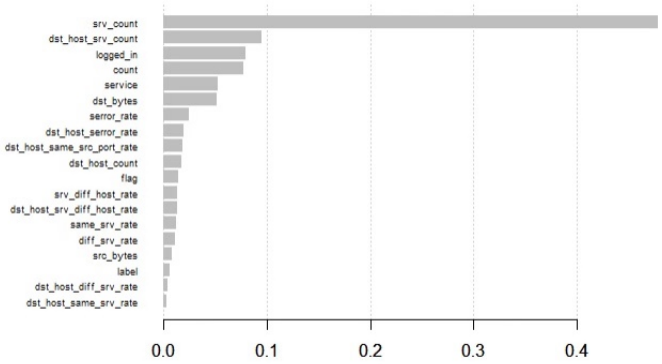

Figure 4: Significant features of XGboost

4 Comparison standard

We mainly use the confusion matrix in R as the criterion for evaluating the quality of the model. Here are some parameters of the confusion matrix in R

★ TP (True Positive): predict the positive class as the number of positive classes

★ FN (False Negative): predict the positive class as the number of negative classes

★ FP (False Positive): predict the negative class as the number of positive classes

★ TN (True Negative): predicts the negative class as the number of negative classes

★ CI (confidence interval)

★ No-information rate: The largest proportion of the observed classes

★ TPR (Sensitivity or true positive rate): $TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$

★ TNR (Specificity or true negative rate): $SPC = \frac{TN}{N} = \frac{TN}{TN+FP}$

★ PPV (Positive prediction value): $PPV = \frac{TP}{TP+FP}$

★ NPV (Negative predictive value): $NPV = \frac{TN}{TN+FN}$

★ Prevalence: $\frac{TP+FN}{TP+TN+FP+FN}$

★ Detection rate: $\frac{TP}{TP+TN+FP+FN}$

★ Detection prevalence: $\frac{TP+FP}{TP+TN+FP+FN}$

★ Accuracy = 100% – Error = $\frac{TP+TN}{TP+TN+FP+FN}$

We found that the accuracy of XGBoost is better than Adaboost and Adaboost is better than GBDT. Through comparison we find that the boosting tree models are better than the baseline, original decision tree. [Figure 5] is the confusion matrix of XGBoost, others will not be shown here.

```
          Reference
Prediction      1      2      3      4      5
         1 117434      5      4      0      0
         2      2  29176     17     24     11
         3      2      2   1212      0      0
         4      0      1      0    314      2
         5      0      0      0      0      3

Overall Statistics

               Accuracy : 0.9995
                 95% CI : (0.9994, 0.9996)
    No Information Rate : 0.7924
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9986

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4  Class: 5
Sensitivity            1.0000   0.9997 0.982968 0.928994 1.875e-01
Specificity            0.9997   0.9995 0.999973 0.999980 1.000e+00
Pos Pred Value         0.9999   0.9982 0.996711 0.990536 1.000e+00
Neg Pred Value         0.9999   0.9999 0.999857 0.999838 9.999e-01
Prevalence             0.7924   0.1969 0.008319 0.002281 1.080e-04
Detection Rate         0.7924   0.1969 0.008178 0.002119 2.024e-05
Detection Prevalence   0.7924   0.1972 0.008205 0.002139 2.024e-05
Balanced Accuracy      0.9998   0.9996 0.991471 0.964487 5.938e-01
```

Figure 5: Confusion matrix of XGBoost

To be honest, these accuracy are not way too different, but in terms of algorithm, GBDT is better than Adaboost, and more powerful than Adaboost. It can handle not only classification problems but also regression problems. However, in our experiments, GBDT performed poorly.

Just like AdaBoost, Gradient Boosting repeatedly selects a model that performs generally and adjusts each time based on the performance of the previous model. The difference is that AdaBoost locates the deficiencies of the model by increasing the weight of the mis-separated data points, while Gradient Boosting locates the deficiencies of the model by calculating the gradient. Therefore, we conclude that in this dataset, it is more effective to increase the weight of mis-separated data points than to increase the gradient.[6]

5 Further research

5.1 Time complexity

The time complexity is discussed to make it possible to run on large-scale data. We found that AdaBoost and XGBoost take 10 times as long as Decision Tree, and GBDT is 50 times as long as Decision Tree. One reason is that the algorithm time of Boosting itself is complicated, on the other hand, lies in different Boosting algorithms, the internal calculation time is also different. Combining time complexity and accuracy, we tested whether XGBoost can run on the complete set. The answer is yes, and it will not crash the system. This was not possible with any algorithm in the previous experiment.

5.2 Feature analysis

First, we take XGBoost as an example, we analyze its main feature 'srv_count' while in other algorithms, the selected main feature is 'count'. By analyzing the histogram and boxplot of the 'srv_count' distribution, we find that 'srv_count' can clearly distinguish dos and other types, which also explains why XGBoost works best. Because the imbalance of the data set with most dos, algorithms that can distinguish dos and other features significantly is surely to achieve higher accuracy.

The [Figure 6] is the boxplot of the distribution of srv_count and the [Figure 7] is the histogram of distribution of srv_count.

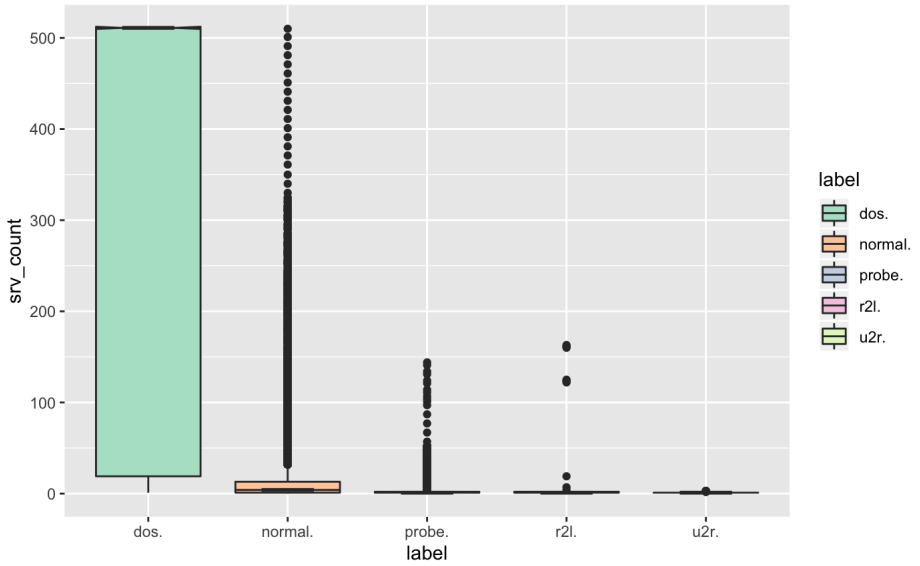

Figure 6: Boxplot of srv_count

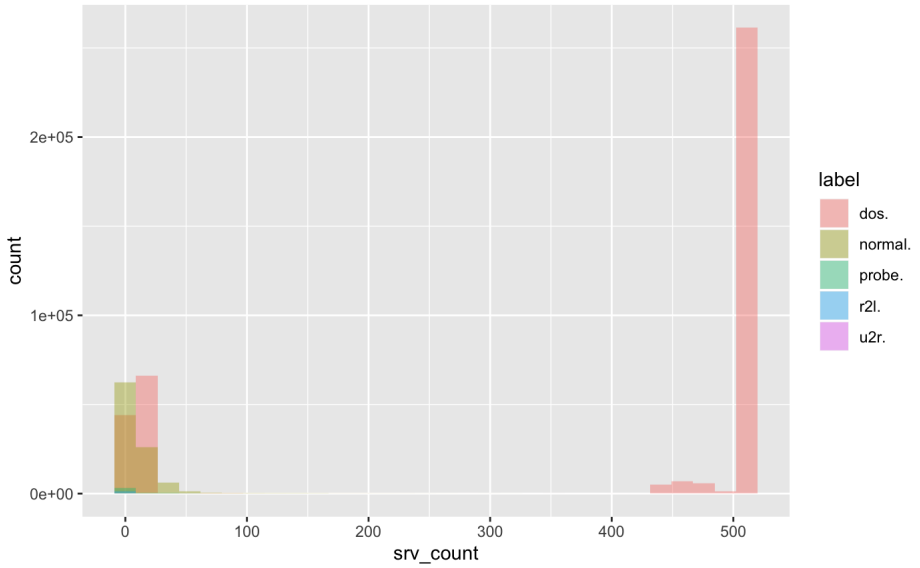

Figure 7: Histogram of srv_count

Secondly, we combined two different main features selected by several algorithms, 'srv_count' and 'count', see [Figure 8] and found that these two features were positively correlated, meanwhile the five types were clearly clustered. In the dataset 'count' indicates the number of connections with the same target host as the current connection in the past two seconds; 'srv_count' indicates the number of connections with the same service as the current connection in the past two seconds. As long as we can combine these two features, we can give an intuitive judgment of what kind of data it is without a computer when given a piece of data.

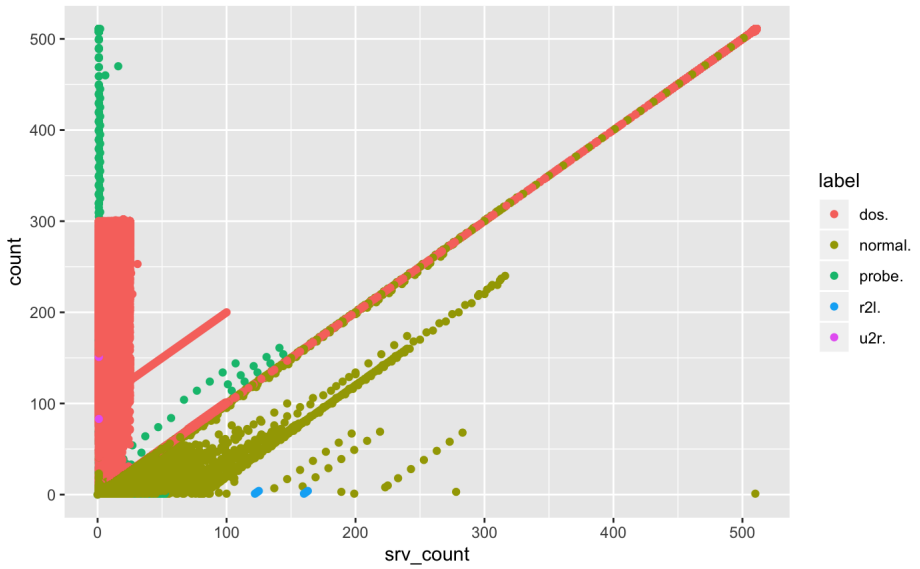

Figure 8: Combine srv_count and count

6 Conclusion

In this assessment, we have deeply explored the mathematical principles behind three similar algorithms and used them to train the dataset. According to the experience from last assessment, we improved the way of data preprocessing. Furthermore, we used the visual-analysis of the distribution of the original dataset by scatter plot. For this assessment our group researched in just one direction with analysis in time complexity and important features. Our **inference goal** is to extend the binary categorization ("normal" vs "abnormal") to multi-class categorization ("dos", "probe", "u2r", "r2l and "normal"). After comparing the results of three algorithms, XGBoost shows the **best result** with accuracy 99.95%. These tree boosting algorithms have gained huge popularity in real world applications.

References

[1] Boosting Algorithms: AdaBoost, Gradient Boosting and XGBoost,
`https://medium.com/hackernoon/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38`

[2] What is a Decision Tree?
`https://www.displayr.com/what-is-a-decision-tree/`

[3] Friedman, J. H., Hastie, T. and Tibshirani, R. (2008). *The Elements of Staistical Learning*. Springer.

[4] Friedman, J. H. (1999). Greedy Function Approximation: A Gradient Boosting Machine. Technical report, Dept. of Statistics, Stanford University

[5] Friedman, J. H. (1999). Stochastic Gradient Boosting. Technical report, Dept. of Statistics, Stanford University.

[6] Boosting with AdaBoost and Gradient Boosting,
`https://medium.com/diogo-menezes-borges/boosting-with-adaboost-and-gradient-boosting-9cbab2a1af81`