Network Intrusion Detection and Classification

Xiaoyang Wang
s1839441
`s1839441@ed.ac.uk`

Yanting Zeng
s1829742
`s1829742@ed.ac.uk`

Kunbo Xu
s1825465
`s1825465@ed.ac.uk`

Qiuming Fu
s1829277
`s1829277@ed.ac.uk`

1 Introduction

In this project, we build several predictive models to distinguish types of connections in computer network and compare the performance of these multiclass classifiers. Basically, there are legitimate normal connections and illegitimate attacks or intrusions belonging to four main categories. We use the dataset of 1999 KDD intrusion detection contest [1], which was gathered by MIT Lincoln Labs. Nine weeks of raw TCP dump data were acquired from a simulated U.S. Air Force local-area network(LAN), where multiple attacks were performed. Then the raw training data were processed into connection data. The test data were generated in a similar way for two weeks.

This report is organized in the following order. In the second part, we make exploratory data analysis which visualizes the distribution of main classes and the distribution of labels in each intrusion class. We also show patterns of features in different classes. In the third part, we pre-process data by building sampling methods, performing standardization, and testing outlier detection techniques. In the fourth part, we fit several models combined with dimension reduction and feature selection methods. For each general model type, we pick one representative model with hyperparameters selected by grid search. The performance evaluation metric is shown in the fifth part. We compare the results on validation set generated from different models and use the best model to fit test data.

2 Exploratory data analysis

The original datasets contain a training set and a test set. Since test data and training data are not from the same distribution, there are labels only appearing in the test set but unknown in the training set. Thus, according to rules in [5] [8], we convert 24 original training attack types and additional 14 test attack types into four main classes: "probe", "DOS", "U2R", and "R2L". Now there are five classes containing one normal class and four intrusion classes.

Data visualization exploration is divided into two parts. In the first part, we plot the distribution of main classes. In terms of the attack labels from each intrusion class, we use mosaic plot to show their proportions. Figure 1 shows that the distribution of classes is imbalanced: “DoS” accounts for a large proportion while proportions of “U2R” and “R2L” are relatively small. This imbalanced pattern can be also found in Figure 2: four intrusion classes are dominated by one or two attack types.

In the second part, we explore the relationship between features and classes. There are 41 feature variables containing 34 continuous variables and 7 categorical variables. Figure 3 illustrates a main characteristic of features: many features are dominated by its one or two attributes. In fact, the percentage of single attribute exceeds 90% in half of features. We also find one useless feature $num\_outbound\_cmds$ as values of this feature in all classes are equal.

For categorical variable, Figure 4 shows that for different classes, the patterns are quite different. For the feature $Protocol\_type$, "DoS" has a much larger number on $icmp$ while udp is more common in

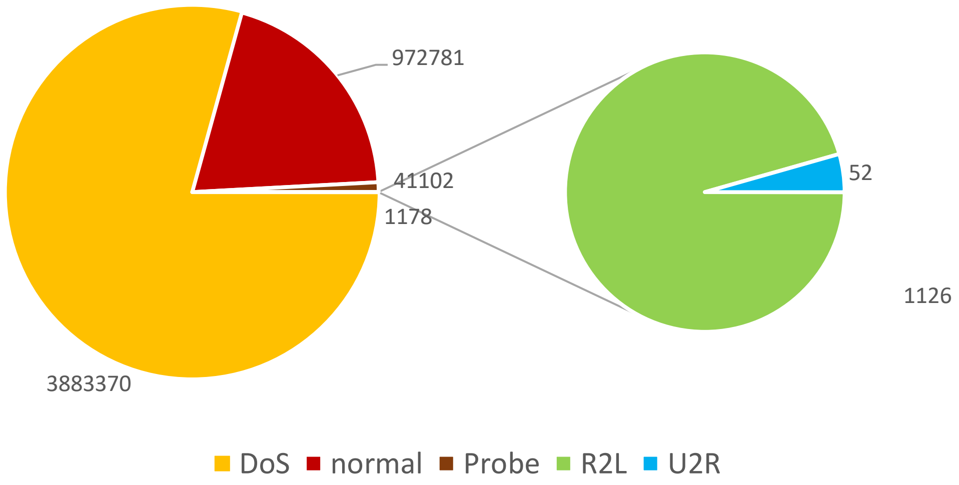

Figure 1: Pie of pie plot for proportions of five classes

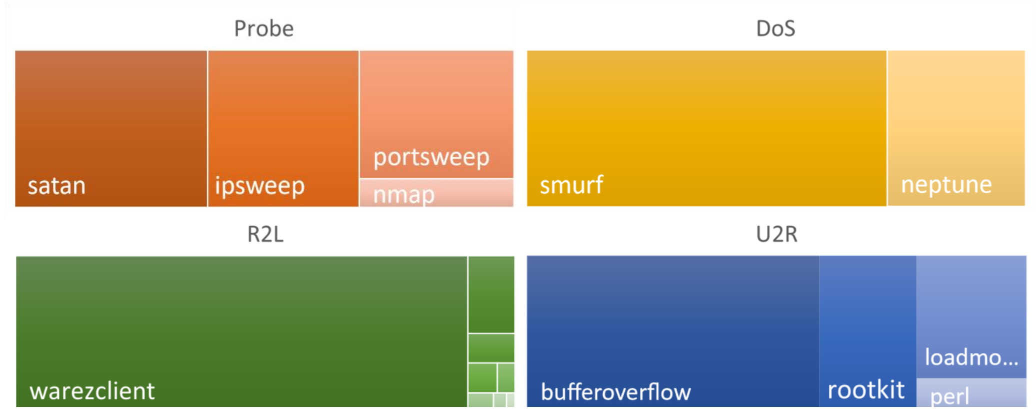

Figure 2: Mosaic plot for attack labels in main intrusion classes

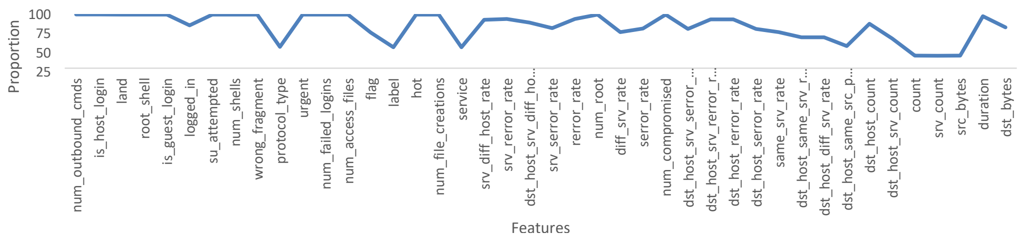

Figure 3: Line plot of the largest proportion of attribute in each feature (features are listed in descending order of the number of attributes)

normal. $Service$ variable involves 70 attributes and the patterns among five classes are more diverse. There are four variables that have only two attributes(one and zero). Little information can be derived from $land$ and is_host_login.

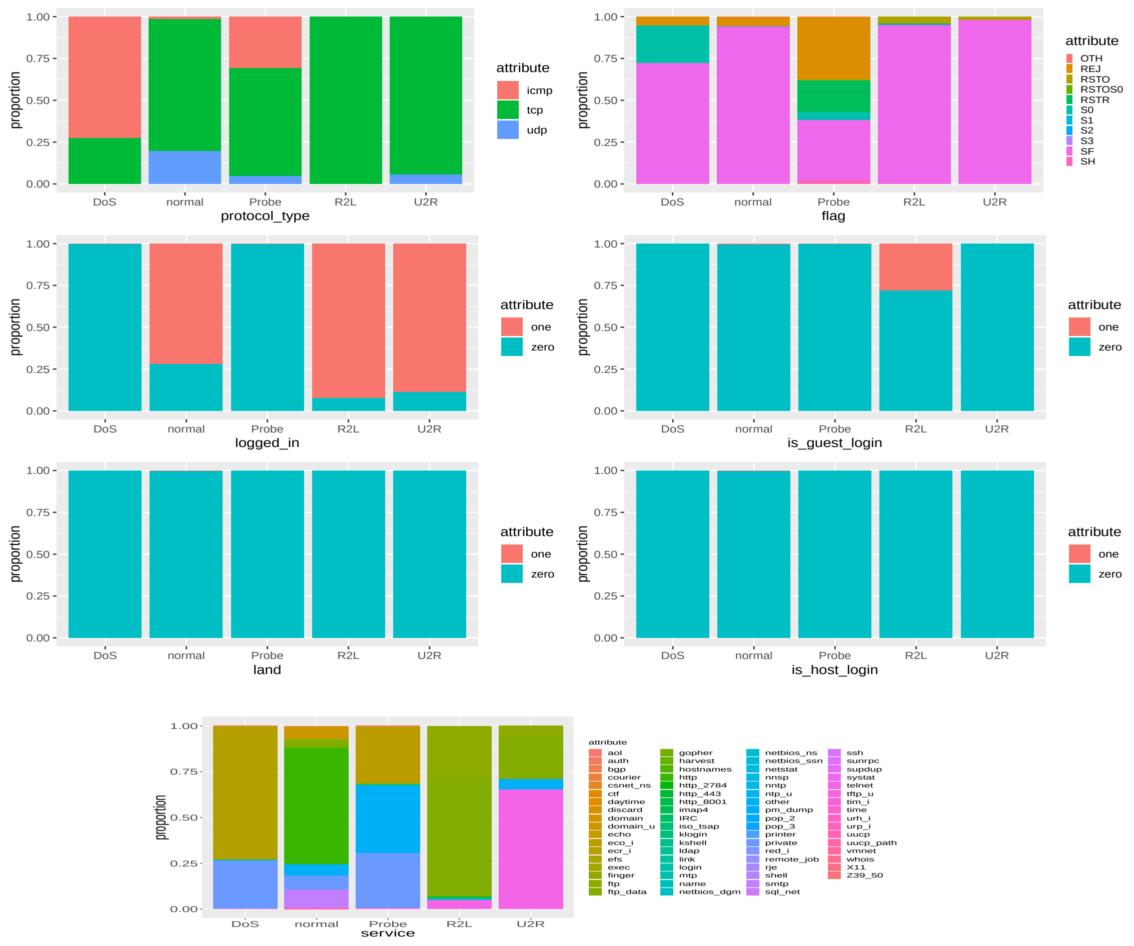

Figure 4: Patterns of categorical variables for different labels

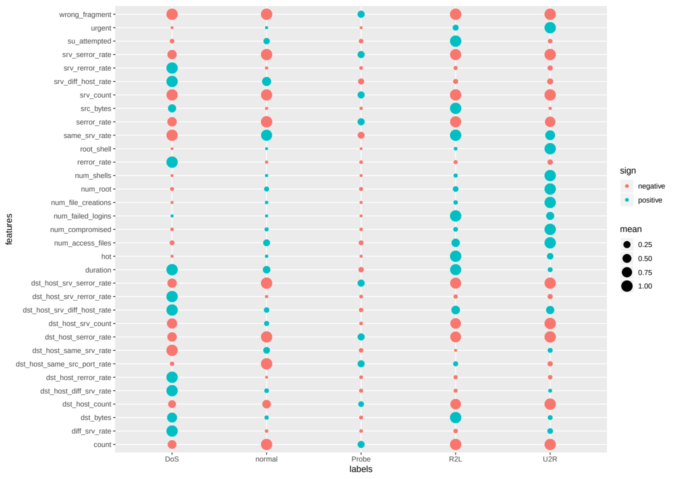

Figure 5: Patterns of features for different classes

For the relationship between numerical variables and labels, to make data less sensitive to the scale, we normalize features by defining adjusted mean μ_{ij} for each feature in each class.

$$\mu_{ij} = \frac{x_{ij}}{|max(x_{1j}, x_{2j}, x_{3j}, x_{4j}, x_{5j})|}, (i = 1, 2, ..., 5; j = 1, 2, ..., n_{feature})$$

where x_{ij} represents the mean for jth feature in ith class. In Figure 5, we use the size of circle to represent absoluted value of μ_{ij} and color to represent sign of μ_{ij}. So we can combine size and color to read this figure. For example, Probe attack category normally takes small absolute values on continuous features which is quite different from other attack categories.

3 Data preparation

We split the original training set into a new training set and a validation set with 80/20 ratio. Since models implemented in scikit-learn [6] package cannot operate on categorical inputs directly, we transform categorical features into numerical features. Here we use one-hot encoding as our categorical features are not ordinal. Then we deal with the existing imbalance, standardize features, and perform outlier detection.

3.1 Subset sampling

We notice that there exists severe imbalance of the distribution within and among classes in both the training set and the test test. To avoid biased classification, we attempt to over-sample minority labels using SMOTE technique. Alternatively, we also build an algorithm which upweights samples in the least attack types in the training set. In specific, we select a fixed number of instances from each label. Intuitively, we consider 500, 1000, and 2000 as the sampling size per label. Figure 6 illustrates an example sampling process within "Probe" class when we set the sampling size per label as 2000. For attack type $nmap$ with samples less than 2000, we take all samples from it; however, for every remaining attack type, we sample 2000 samples randomly. Then we calculate the sample weight for each attack type in "Probe" class, and thus the size of "Probe" data used for training is equal to N. We find that our customized sampling method performs better than SMOTE over-sampling method. A possible explanation is that we still exploit existing data points instead of synthesizing fake data points which don't reflect the reality. After sampling from each label, we convert the labels of those instances to their corresponding main classes.

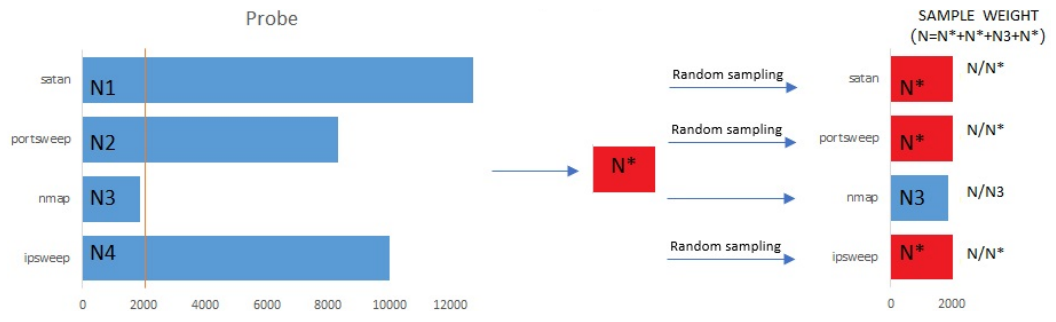

Figure 6: Illusion for subset sampling (taking sampling Probe class as an example)

In addition, we modify the loss function of each classifier in the following part by assigning more weights to classes with small N to achieve between-class balance. In the end, the total weight for each training instance is the product of class weight and sample weight.

3.2 Feature Standardization

Afterwards, we perform standardization on feature vectors by adjusting the mean and variance of each feature to 0 and 1 respectively. The reason for doing so is to eliminate the scale differences between features. Otherwise, features with large numerical units may have dominant influence on the resulting predictions. It is worth noting that tree based models do not need this operation because tree simply cuts the input space into multiple regions without considering their exact values.

3.3 Outlier detection

We attempt to remove outliers by deleting instances that have feature values beyond three standard deviations within each label, but have found that a large number of instances would be affected in this way. The situation is especially worse for minority labels. After all, many attacks themselves are considered rare and abnormal compared to normal connections.

Alternatively, we use a density-based *Local Outlier Factor* (LOF) method [2], which has been claimed to work well in previous intrusion detection experiments [4]. The basic idea is to compare the local density of an object to densities of its neighbors. Objects with significantly lower density values than neighbors are considered to be outliers. Since there are lots of minority labels, we only perform LOF method on labels with sample size larger than the number of neighbors, which is 20 by default. At the same time, we set the amount of contamination to 0.1 by default, indicating that there is supposed to be 10% outliers in the dataset. However, in later experiments, we find that LOF do not contribute a lot and even make the model performance worse. So we eventually abandoned these two attempts.

4 Learning models

For this multiclass classification task, we pick one representative model from each of the general model types. Specifically, we consider Softmax Regression as our linear model, soft margin kernel SVM as our non-linear model [3] and AdaBoost with ID3 Decision Tree [7] base estimators as our ensemble model [10]. Besides, we have trained them with different combinations of dimension reduction or feature selection techniques except for AdaBoost.

4.1 Feature selection

We have applied embedded, wrapper, and filter methods respectively for feature selection. An intrinsic way of feature selection is to perform Softmax regression combined with L1 regularization, which is an embedded method. L1 encourages sparsity, which forces weight parameters of less important features to 0 and thus can be regarded as feature selection occurring at the model training stage. After training in the first model, we pick out features with non-zero weights and feed them into another model to do further training. We expect a better performance by using such wrapper selection method.

Alternatively, we use mutual information to select features, which quantifies the degree of dependence between each feature distribution and the class distribution [9]. We keep features with highest mutual information scores and remove others with lower scores. Since we only consider the statistical characteristics of the inputs without using learning algorithm, mutual information is a filter method.

4.2 Dimension reduction

An alternative to select and exclude features is to transform them into a lower dimension. To reduce dimensionality, we consider both principal component analysis (PCA) and kernel principal component analysis (KPCA) using linear projections and nonlinear projections respectively.

4.3 Model fitting

We train our models on the input feature vectors generated from above dimension reduction or feature selection procedure using Softmax regression as linear model and SVM as non-linear model respectively. Here we use Softmax regression for multinomial classification and soft margin SVM with radial basis function (rbf) kernel. Moreover, when fitting the SVM model, we choose one-vs-rest classifier which is suitable for such multiclass classification problem.

We have also fit multi-class AdaBoosted decision tree, which is an ensembling learning algorithm. The algorithm developed by Zhu et al in 2009 has been a successful extended AdaBoost algorithm for solving the multi-class classification problem [10]. For the aforementioned reason in the third part, we train AdaBoost with tree base estimators on raw input vectors without feature standardization. Moreover, we do not need feature selection or dimensionality reduction techniques for AdaBoost, because ID3 tree calculates feature importance inherently based on the criterion of entropy and keep the most importance features close to the root of the tree.

4.4 Hyperparameters selection

When fitting each model, we perform hold-out validation within the training set using Stratified ShuffleSplit validator. We have also considered Stratified K-Folds cross-validator but finally do not apply it due to a low speed of training and execution. Then we perform grid search on hyperparameters including number of components in PCA, C and gamma in SVM, number of features with top MI scores, etc., which returns the best combination of hyperparameters for each model. When choosing the number of components in linear PCA, we also refer to figures to find reasonable searching ranges. Figure 7 shows the explained ratio of principal components against the number of principal components for linear PCA when sample size per label is 2000. Principal components are listed in descending order of individual explained variance. The red line indicates the cumulative explained variance and an inflection point appears at around 10. The purple dash line highlights that 90% of variance has been explained by 72 components. Consequently, a reasonable initialized searching range of PCA components number could be (10,75).

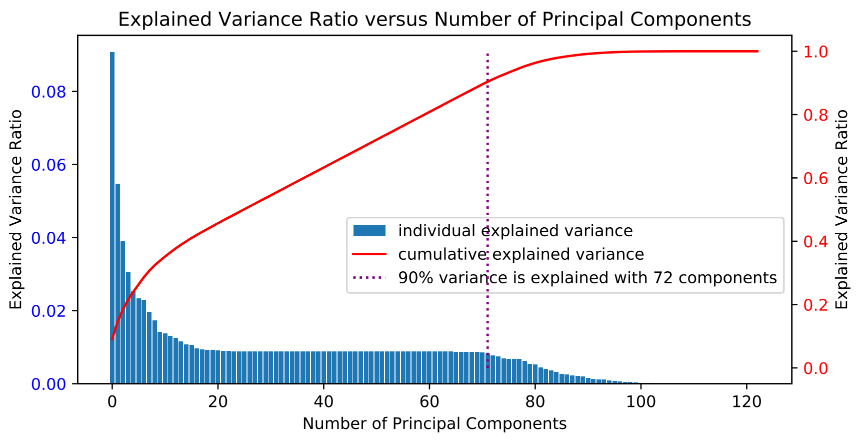

Figure 7: Explained Variance Ratio against the number of principal components when sample size per label = 2000

5 Performance evaluation

When evaluating the model performance, we value recall higher than precision because the proportion of some intrusion classes is extremely small and the cost of misclassifying an intrusion is higher than that of misclassifying a normal one. In addition, we evaluate PR-AUC which is appropriate for imbalanced datasets. Further, we evaluate macro-average performance in the validation set to avoid that the classes with the largest proportions would dominate the overall performance.

5.1 Results on Validation Set

Fig.8 shows results of all the pipelines (model) being evaluated in this experiment. All the pipelines have been fully tuned in training session, which means the comparison is unbiased. Adaboost model hits the highest PR-AUC score of 0.83 on the validation set without. The lowest performance is found in pipeline *Linear PCA* as dimension reduction technique with *Logistic Regression* as classifier.

Table 1 shows detailed performance information of the best model (AdaBoost) on the validation set. For precision results, our model performs well in identifying "Dos" and "Probe" but have high false positive rate when classifying "R2L" and "U2R". One possible reason is that the number of "R2L" and "U2R" in our validation set is too small compared with other classes. Therefore, the precision of "R2L" is lower than that of "Probe" even they have similar number of false positive cases. For class recalls, AdaBoost has a great overall performance, which proves its power in this task since recall is more important than precision in this case. AdaBoost is able to capture more than 95% attacks in each type. 100% recall for "U2R" is reached by the model to flawlessly find all the 11 targets from over 700 thousands cases.

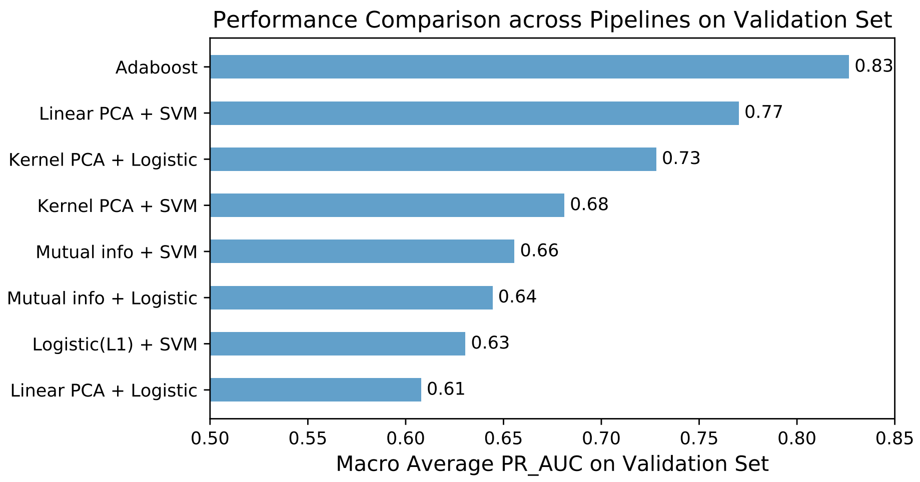

Figure 8: A visualization of pipeline performances. Pipelines are listed on the left side which are in the form of *data preprocessing method + classifier*. AdaBoost model reaches the highest average PR_AUC without feature selection or dimensionality reduction stage.

Table 1: The Confusion Matrix of Best Model (AdaBoost) on Validation Set

	DoS	**Probe**	**R2L**	**U2R**	**Normal**	**Recall%**
DoS	776413	197	7	0	58	99.97%
Probe	18	8169	8	4	21	99.38%
R2L	0	0	218	7	0	96.89%
U2R	0	0	0	11	0	100.0%
Normal	31	1419	1510	719	190877	98.11%
Precision%	99.99%	83.48%	12.51%	1.48%	99.96%	

5.2 Results on Test Set

The model performance on final test set is shown in Table 2. Compared to validation result, great increase can be observed in the precision of two difficult classes "R2L" and "U2R", changing from 12.51% to 87.46% and from 1.48% to 38.86% respectively. However, the recall of these two classes fall significantly: only 8.49% of "R2L" and 65.79% of "U2R" can be found. One explanation for change of the performance is that the test set and the validation set split from the original training set are from different probability distributions. To have further evaluation, we compare our results with the winning performance results in the KDD contest. Our best model has superior performance compared to that of contest winner in terms of the recall results, especially for the class "U2R".

Table 2: The Confusion Matrix of Best Model (AdaBoost) on Test Set

	DoS	**Probe**	**R2L**	**U2R**	**Normal**	**Recall%**
DoS	223604	351	2	0	5896	97.28%
Probe	19	3570	39	8	530	85.69%
R2L	4	10	1374	96	14705	8.49%
U2R	0	25	13	150	40	65.79%
Normal	94	415	143	132	59809	98.71%
Precision%	99.95%	81.67%	87.46%	38.86%	73.86%	

Table 3: Performance of the Contest Winning Entry [1]

	DoS	**Probe**	**R2L**	**U2R**	**Normal**	**Recall%**
DoS	223226	1328	7	0	5299	97.1%
Probe	184	3471	0	0	511	83.3%
R2L	0	294	1360	8	14527	8.4%
U2R	0	20	10	30	168	13.2%
Normal	78	243	6	4	60262	99.5%
Precision%	99.99%	64.8%	98.8%	71.4%	74.6%	

6 Conclusions

Using the dataset of 1999 KDD contest, we aim to train different intrusion detection classifiers and find the model with the best performance. Since the training set and the test set are from different distributions and the test set includes labels that never appear in the training set, we convert all the labels from two sets into five main classes. By data visualization, we find severe imbalance among and within classes. Thus we build sampling method and use subsets from the training set to avoid biased classification. At the training stage, we train Softmax regression model and soft margin SVM model respectively, combined with embedded, wrapper, and filter feature selection techniques or PCA dimension reduction techniques. However, we find that the ensembling model AdaBoosted decision tree without feature standardization and feature selection process returns the best macro-average PR_AUC on the validation set. It also beats other models in the training speed. Compared to the performance of the winning model in the KDD contest, our AdaBoost model achieves higher recall values for most of the classes in the test set. The improvement of the recall of a minority class "U2R" is the most significant. We may conclude that AdaBoost with tree-based estimators is suitable for such multi-class intrusion classification problems, especially for detecting minority attack types.

References

[1] Kdd cup 1999: http://kdd.ics.uci.edu/databases/kddcup 99/kddcup99.html.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000.

[3] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

[4] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *In Proceedings of SIAM Conference on Data Mining*, 2003.

[5] P. Natesan and P. P Balasubramanie. Multi stage filter using enhanced adaboost for network intrusion detection. *International Journal of Network security and its Applications*, 4:121–135, 05 2012.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.

[8] C. Thomas, V. Sharma, and N. Balakrishnan. Usefulness of darpa dataset for intrusion detection system evaluation. In *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, 2008.

[9] J. R. Vergara and P. A. Estévez. A review of feature selection methods based on mutual information. *Neural Computing and Applications*, 24(1):175–186, Jan 2014.

[10] J. Zhu, H. Zou, S. Rosset, and T. Hastie. Multi-class adaboost, 2009.

7 Individual Contributions

Xiaoyang Wang: Investigate and implement deviation based outlier detection in data preprocessing stage. Build grid search pipelines by combining PCA with logistic regression or support vector machine in order to tune the number of components and hyper-parameters of models at the same time. Implement tool function named *batch PCA* to solve problem on memory management. Enable kernel PCA to be applied on the whole training data. Illustrate and analyze results of models in the report.

Yanting Zeng: Implement data loader, one-hot encoding scheme and feature standardization. Implement LOF outlier detection method. Implement SMOTE oversampling method. Conceive and implement our customized sampling strategy. Implement grid search skeleton and modify the loss function for each model. Implement mutual information as filter feature selection method. Implement logistic regression with L1 regularization as embedded feature selection method and feed features with non-zero weights to our subsequent SVM as wrapper feature selection method. Implement variance visualization for linear PCA. Come up with and implement macro average PR-AUC evaluation metric. Write code to generate our confusion matrix.

Kunbo Xu:Responsible for the exploratory data analysis (making plots and analyze the results). Plotting the flow chart of the customized sampling. Running the grid search pipelines to tune the number of components and hyper-parameters of models("Logistic(L1) + SVM", "Mutual info + Logistic" and "Mutual info + SVM", "Linear PCA + SVM" and "Linear PCA + Logistic", "Adaboost").

Qiuming Fu: Researched on all the data preprocess, outlier detection, dimensionality reduction, feature selection, model fitting, and performance evaluation methods in the project. Wrote the interim report and contributed to large parts of the final report. Ran grid search pipelines and found hyperparameters of logistic regression and SVM models with kernel PCA.