University of Science and Technology of Hanoi

Bachelor's Thesis in Information and Communication Technology

Application of Machine Learning in Credit Card Fraud Detection

Authors:
DANG Anh Duc
BI9068

Supervisor:
DOAN Nhat Quang
ICT Lab
ICT Deparment
Vietnam France University

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in

Information and Communication Technology

May 18, 2021

Acknowledgement

I would like to express my sincere gratitude towards Dr. DOAN Nhat Quang for his detailed and constructive suggestion during the internship. He continuously guided me in the right direction throughout the span of this project. In addition, he also provided me with valuable knowledge, not only about the research topic, but also other important skills such as how to do a research or how to write a proper research report.

I would also like to thank my friends, NGUYEN Minh Thu, TRINH Mai Phuong, TRAN Thanh Long, NGUYEN Truong Giang. Without their help, this project would not have been possible.

Abstract

Credit card fraud is an [sth] problem in the financial world. The number of fraudulent transactions is expected to increase due to the recent trend of using non-cash payments. However, using machines to detect credit card fraud is not an easy task since the available datasets for this problem are highly imbalance i.e. the number of genuine cases greatly outnumber the fraudulent cases, which makes process of training a classification models harder and create inaccurate models.

In order to tackle this problem, our project suggests different techniques to resample the dataset, such as, undersampling, oversampling and hybrid strategy, which is a combination of both undersampling and oversampling. These techniques are implemented with different predictive models like Logistic Regression, Random Forest and XGBoost. Each combination between a resampling method and model is evaluated based on precision, recall, f1-score, precision-recall (PR) curve and receiver operating characteristics (ROC) curve.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Credit Card Fraud Detection

1.2 Aim of the project

In this project, our main objective is to explore different techniques to deal with an imbalanced dataset and evaluate them to see which method performs better than the others. More specifically, this project focuses on handling a credit card transaction dataset to build model to detect fraudulent transaction by using different sampling methods along with various models. After that we chose the most well-performed model based on a range of evaluation metrics.

1.3 Overview

This section provides an overall overview of the content entailed in each section. In section 2, we discuss relevant literature in the current field of research, focusing on the methods to build a credit card fraud detection model. Section 3 presents the methodology including the data processing steps, tools and libraries used, as well as the training of the model. In Seciton 4, we describe model's evaluation metrics - precision, recall, f1-score, PR curve, ROC curve and provide a detailed discussion on the results of our project. The final section 5 presents a brief conclusion of our project.

2 Literature Review

3 Methodology

3.1 Dataset Description

For this project, we used a dataset consists of transactions made by credit cards in two days in September 2013 by European cardholders which was collected by the Machine Learning Group of Université Libre de Bruxelles (ULB) and was published on Kaggle[†]. The dataset is contains a total of 284,807 transactions in total, in which only 492 are fraudulent. The dataset is considered to be highly skewed as the positive class (frauds) only accounts for 0.172% of the dataset. Figure 1 visualizes the class distribution of the dataset.

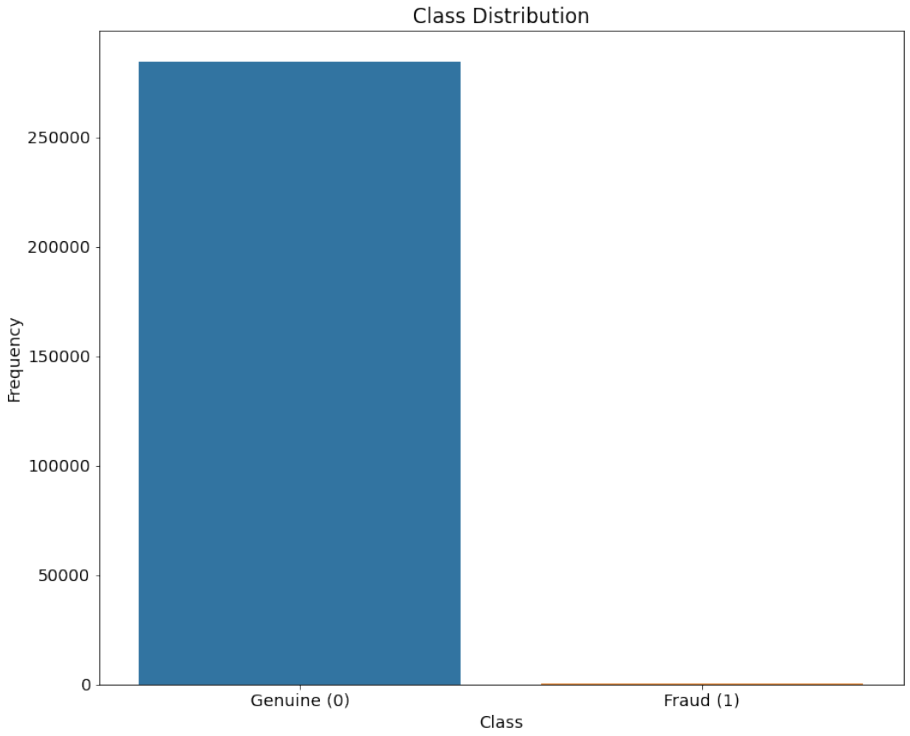

Figure 1: Dataset Class Distribution

The dataset only contains numerical values as a result of Principal Com-

[†]https://www.kaggle.com/mlg-ulb/creditcardfraud

ponents Analysis (PCA) transformation. Due to confidentiality issue, most of original attributes was not revealed. There are total 30 features, 28 of which was generated by PCA. The only features that was not transformed are '*Time*' and '*Amount*' Feature 'Class' is the target attribute and it takes value 1 in case of fraud and 0 otherwise. Table 1 gives a detailed description about the dataset's attributes.

Attribute	**Type**	**Description**
Time	Integer	Time elapsed between each transaction and the first transaction
V1	Double	First PCA component
V2	Double	Second PCA component
...	...	...
V28	Double	Last PCA component
Amount	Double	Transaction amount
Class	Integer	Target class (0 = Genuine and 1 = Fraud)

Table 1: Dataset Attribute Description

3.2 Data Pre-processing

3.2.1 Data Standardzation

It is a common requirement for machine learning techniques models that the data is normalized before training. A dataset which was not normalized before training might lead to undesirable outcomes as variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias. Although for Logistic Regression and Tree based models, they are not as sensitive to the magnitude of the variables as other models such as K-Nearest Neighbors or Support Vector Machine (SVM) [1], we still decided to apply standardization for the data in order to

retrieve the best possible result. Standardization can be achieved as follows:

$$z = \frac{Value - Mean}{StandardDeviation}$$

where:

$$Mean(\mu) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$StandardDeviation(\sigma) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

For this project, we standardize our data using the StandardScaler module in the scikit-learn library for convenience.

3.2.2 Data Splitting

After having standardized our data, we split the dataset into training set (70%) and test set (30%). For consistency whenever the program is executed, we assigned a constant *random_state* (14) when splitting. The training set will then be resampled and trained by different techniques and models as well as to tune each model's hyperparameter. The test set will be used to evaluate the performance of the models only.

3.3 Data Resampling

As mentioned in section 3.1, the dataset is highly unbalanced as the number of legitimate transactions outnumbers the number of fraudulent transactions. If we train our model directly on this dataset, the outcome will likely to be biased towards the genuine transactions. To tackle this problem, we used some commonly known resampling techniques such as Random Oversampling, Random Undersampling, Synthetic Minority Oversampling Technique (SMOTE) and Tomek Links Removal Undersampling.

In underampling methods, the majority class is reduced to a certain percent compared to the original data to make the number of instances between two classes balanced. Figure 2 depicts the main idea of undersampling. Undersampling is easy to implement and applying undersampling would greatly save training time and storage space when a huge dataset is used. In contrast to undersampling, in oversampling, we will work with the minority class. Instances of the minority class will be duplicated or new data points of that

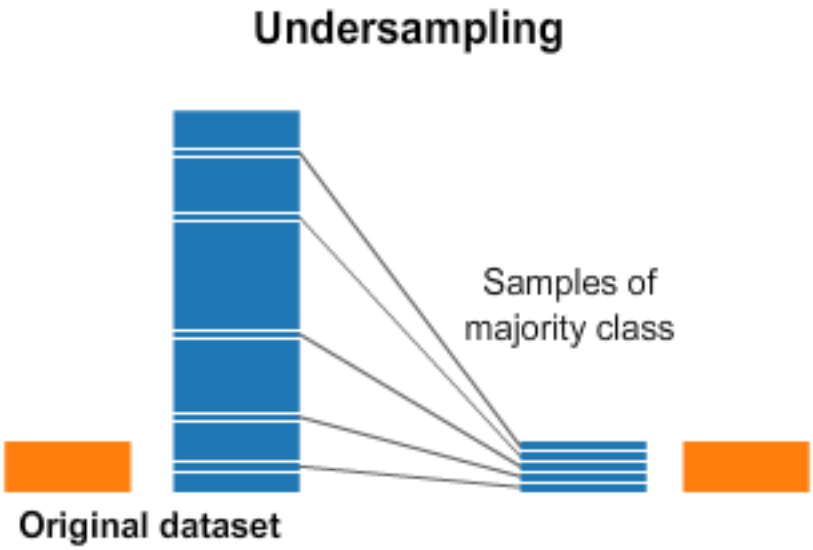

Figure 2: Undersampling Method

class will be generated to balance the ratio two classes, which is shown in Figure 3.

3.3.1 Random Oversampling

In Random Oversampling, random samples of the minority class is replicated to make the dataset balanced. Howerver, there is a high possiblity that the models would overfit the data since many instances of the same sample in the minority class is duplicated.

3.3.2 Random Undersampling

In Random Undersampling, random samples of the majority class removed from that dataset in order to balance the number of data between two classes. Unlike Random Oversampling, this method is less prone to overfitting; however during the process of elimination, useful information may be lost, resulting in inaccurate predictions by the models.

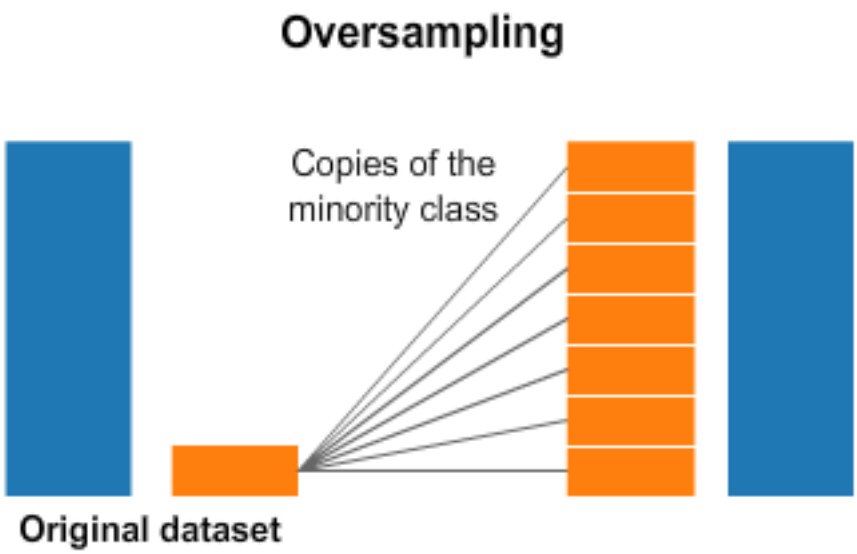

Figure 3: Oversampling Method

3.3.3 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is a popular oversampling techniques that was developed by Chawla, Bowyer, Hall and Kegelmeyer [2]. Instead of duplicating existing data like in Ranom Oversampling, SMOTE creates new data samples by interpolating between nearest minority samples. Number of chosen nearest neighbors would depend on the amount of oversampling required. By creating new data, this techniques tackled the problem of overfitting proposed by Random Oversampling. Figure 4 describes how synthetic examples are created

3.3.4 Tomek Links Removal Undersampling

Given two samples A and B of two different classes, if there isn't a sample C such that the distance between C and A or between C and B is less than the distance A and B, the pair (A,B) is a Tomek link [**NEED REF**]. Removing these links by eliminating the majority samples associated to these links, we will be able to perform undersampling for the dataset and also make the boundaries between two classes clearer.

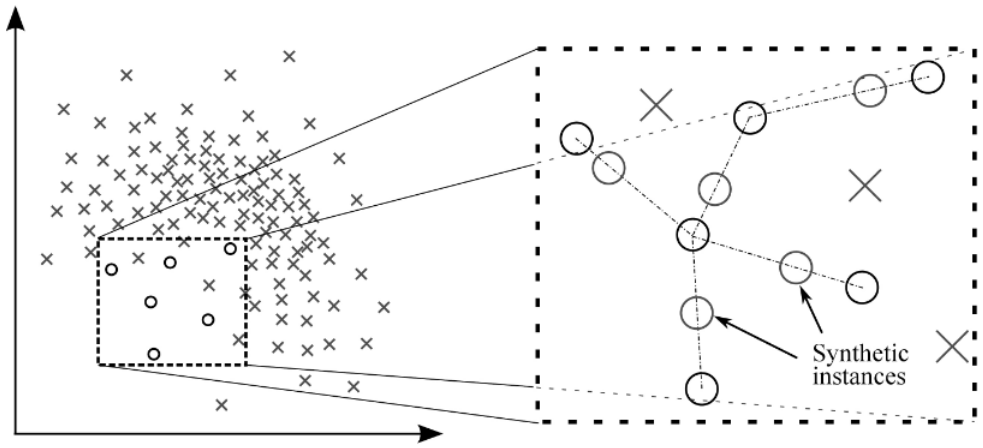

Figure 4: Synthesizing data using SMOTE [‡]

3.3.5 Combination of SMOTE and Tomek Links Removal

While creating new synthetic data points, minority clusters can get mixed up with the majority samples in the data spaces. In order to mitigate this situation, both SMOTE and Tomek Links Removal can be used to balance the dataset. Tomek Links Removal will be applied after synthesizing new minority data to remove samples that have invaded the majority class space.

3.4 Hyperparameter Tuning Using Cross Validation

Hyperparameter of a model is the external configuration of a model that can not be estimated by looking at the data. Since the parameters can not be changed during train, they must be set manually beforehand. In order to for a model to perform at its best, its parameters have to be tuned find the best possible values. In this project, we used K-fold Cross Validation to tune the parameters for our models. More specifically, we used 10-fold Cross Validation by implementing *GridSearchCV* provided by *scikt-learn*. In 10-fold Cross Validation, we divided the training set into 10 folds, and each experiments will take 1 different fold as a validation set and the rest will be used for training. Since each model has different parameters, the

[‡]https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family/

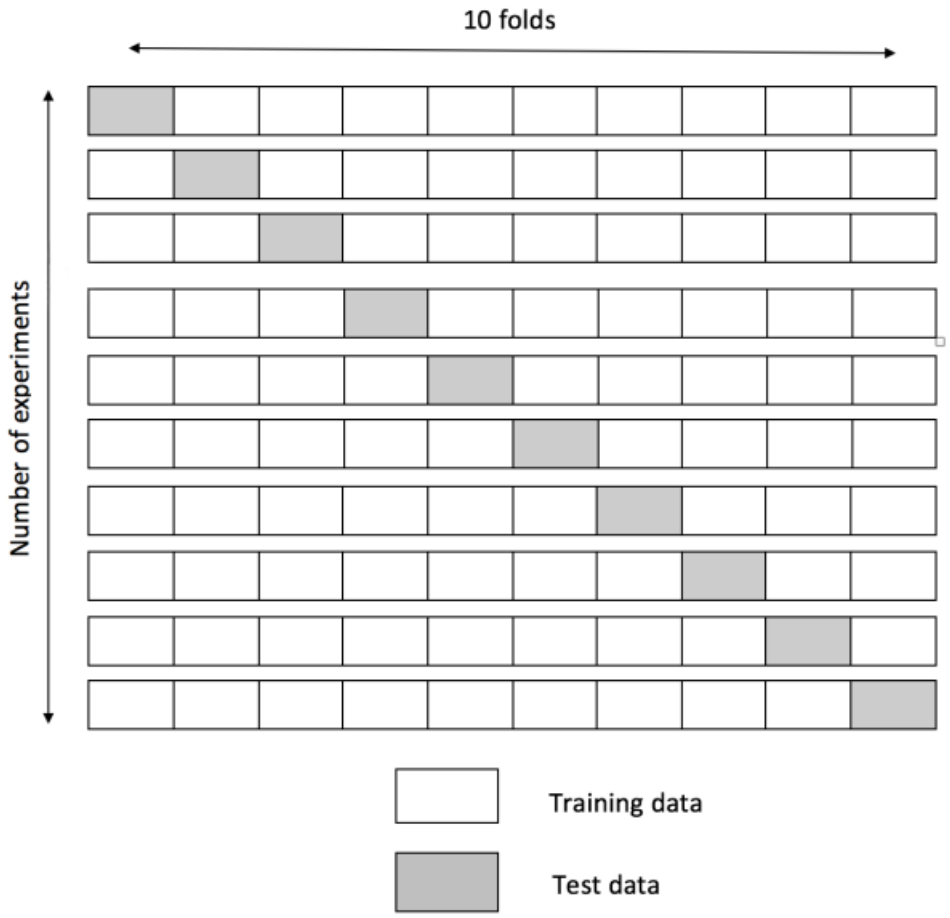

Figure 5: 10-Fold Cross Validation

hyperparameters tuning for each model was different.

3.4.1 Hyperparameters Tuning for Logistic Regression

In Logistic Regression models, the regularization parameter (C) is an important hyperparameter to keep an eye one. If C is too large, the model tends to overfit the data and vice versa. Given a list of possible of values for 'C', the *GridSearchCV* module will perform cross validation on the resampled

data for all values of 'C' and return the best value for that model.

3.4.2 Hyperparameters Tuning for Random Forest

3.4.3 Hyperparameters Tuning for XGBoost

4 Results

4.1 Evaluation Metrics

In this project, we use different metrics to evaluate the performance of a model: precision, recall, F1-score, PR curve, ROC curve [3].

4.1.1 Precision, Recall and F1-score

Precision can be defined as the number of correct positive prediction over the total of positive prediction. *Recall* is the number of correct positive prediction over the total of positive ground truth. Given a confusion matrix as in Figure 6, Precision and Recall score are computed as follows:

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

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

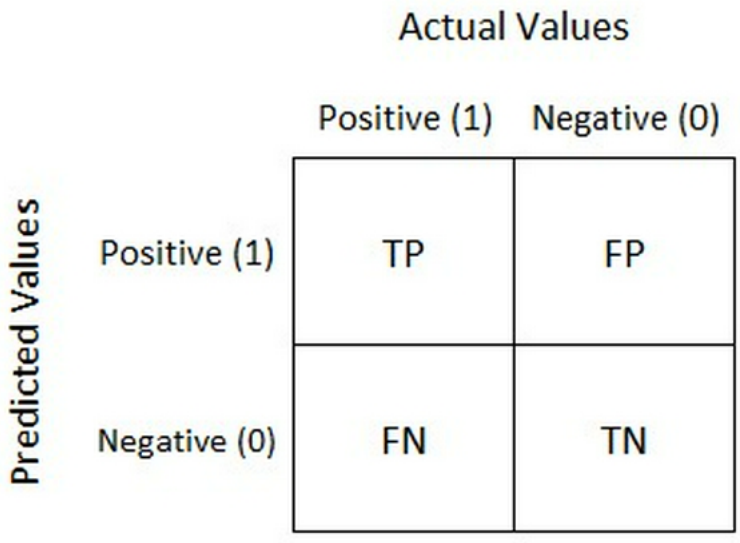

Figure 6: Confusion Matrix Example

When we use both precision and recall, it is a good idea to look into *F1-score* as well since it is a function of both precision and recall. F1-score is a good metric when we look for a balance between Precision and Recall since the number of True Negative (TN) does not contribute in the calculation of F1-score, which is very suitable for skewed data that has a lot of negative sample like in this project. The formula of F1-score is as follows:

$$Precision = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

4.1.2 Precision-Recall Curve

When working with an imbalanced data, we would want to keep track of both the precision and recall of the model to make sure the model does not overfit the majority class and produce unreliable predictions. Precision-Recall (PR) curve is a useful measure of a model prediction as it shows the trade-off between precision and recall for different threshold. A model with low precision and high recall will result in a lot of positive predictions but many of them would be wrong compared to the ground truth. A model with high precision and low recall would produce few positive prediction which might leave out a lot of true positive samples but the predicted ones usually match their ground truth. Figure 7 shows and ideal PR curve with both precision and recall being 1.

In order to evaluate a model based on the PR curve, we use the Area Under the Curve (AUC), which is also known as Average Precision (AP), of the curve. The AUC of the PR curve can be calculated using integral; however we can vary the threshold by a small amount each time so that the AUC can be calculated by summing up the areas as. The AUC can be treated as a weighted sum of the precision scores. The formula to compute the AP is as follows:

$$AP = \sum_{n} (R_n - R_{n-1}) \times P_n$$

where P_n and R_n represent the precision and recall at threshold n^{th}.

4.1.3 Receiver Operating Characteristic Curve

Similarly to PR curve, Receiver Operating Characteristic (ROC) [4] curve illustrates the diagnostic probability of a model with varied threshold. In

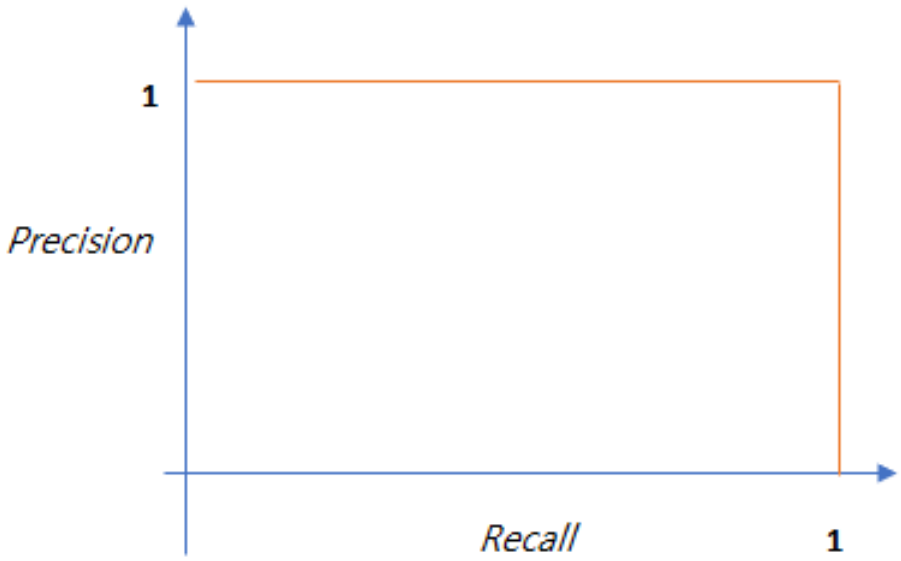

Figure 7: Ideal Precision-Recall Curve

order to understand this metric, we must first understand the concepts of True Positive Rate (TPR) and False Positive Rate (FPR). The TPR is also know as Sensitivity or Recall and the FPR is also known as the inverse Specificity which is calculated as the total number of true negatives over the sum of the number of true negatives and false positives. Consider Figure 6, the FNR is computed as:

$$FPR = 1 - \frac{TN}{TN + FP}$$

4.2 Performance

4.3 Discussion

Overall, the models we chose achieved our main criteria, real time detection and high accuracy. Both YOLOv4 and YOLOv5 perform similarly, with the average precision on unknown test data achieving 0.75. This result means the system should perform accurately and reliably.

4.3.1 Difficulties

5 Conclusion

6 References

[1] Zakaria Jaadi: "When and Why to Standardize Your Data?", 2019, https://builtin.com/data-science/when-and-why-standardize-your-data

[2] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer: "SMOTE: Synthetic Minority Over-sampling Technique", 2002, https://arxiv.org/pdf/1106.1813.pdf

[3] Hossin, M. and Sulaiman, M.N: "A Review on Evaluation Metric for Data Classification Evaluations", 2015, https://www.researchgate.net/publication/275224157

[4] Andrew P.Bradley: "The use of the area under the ROC curve in the evaluation of machine learning algorithms", 1997, https://doi.org/10.1016/S0031-3203(96)00142-2