Credit Card Fraud Detection

Project Proposal

1. Introduction

This project deals with the problem of building a machine learning model for fraud detection – identifying the fraudulent transactions from a set of credit card transactions, and hence the clients are financial institutions. The dataset used in this project is published by Kaggle[1], which contains credit card transactions from September 2013, made by European cardholders. This dataset presents transactions that occurred in two days, where there are 492 frauds out of 284,807 transactions. Therefore, the dataset is highly imbalanced as the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables, which are the result of a Principal Component Analysis (PCA) transformation. Unfortunately, due to confidentiality issues, the original features and the detailed background information about the data are not provided. Coded features V1 to V28 are the principal components obtained after applying PCA to the raw dataset. See Table 1 for more details.

Total Transactions	Legal	Fraudulent	Fraud Ratio	Number of Features
284807	284315	492	0.17%	28

Table 1: Summary of Dataset

Feature labeled as 'Class' is the response variable and it takes value 1 in case of fraud, and 0 otherwise. There are no missing values in the dataset.

2. Methodology

This project applies three different approaches. One is the baseline that works directly on the original data without any resampling. The other two are based on resampling techniques, i.e., undersampling and oversampling. A 75%-25% training-test split of the original dataset, with stratification, is used in all three approaches. Therefore, training and test datasets have the same proportion of fraudulent transactions as the original dataset and they are summarized in Table 2.

[1] https://www.kaggle.com/dalpozz/creditcardfraud

	Training	Test
Fraud	369 (0.17%)	123 (0.17%)
Non-fraud	213,236 (99.83%)	71,079 (99.83%)

Table 2: Summary of Training and Test Datasets

The three approaches are discussed further as follows.

1) Baseline: the baseline approach simply takes the original dataset as it comes and trains the models directly on the highly skewed data.

2) Undersampling: to make a dataset with 50-50 fraud and non-fraud transactions, this approach under-samples the majority class (the non-fraudulent transactions in this case) by only randomly selecting a fraction (i.e. the number of frauds) of the non-frauds and pair up these sampled non-frauds with all the frauds to form a balanced dataset. This is illustrated in Figure 1.

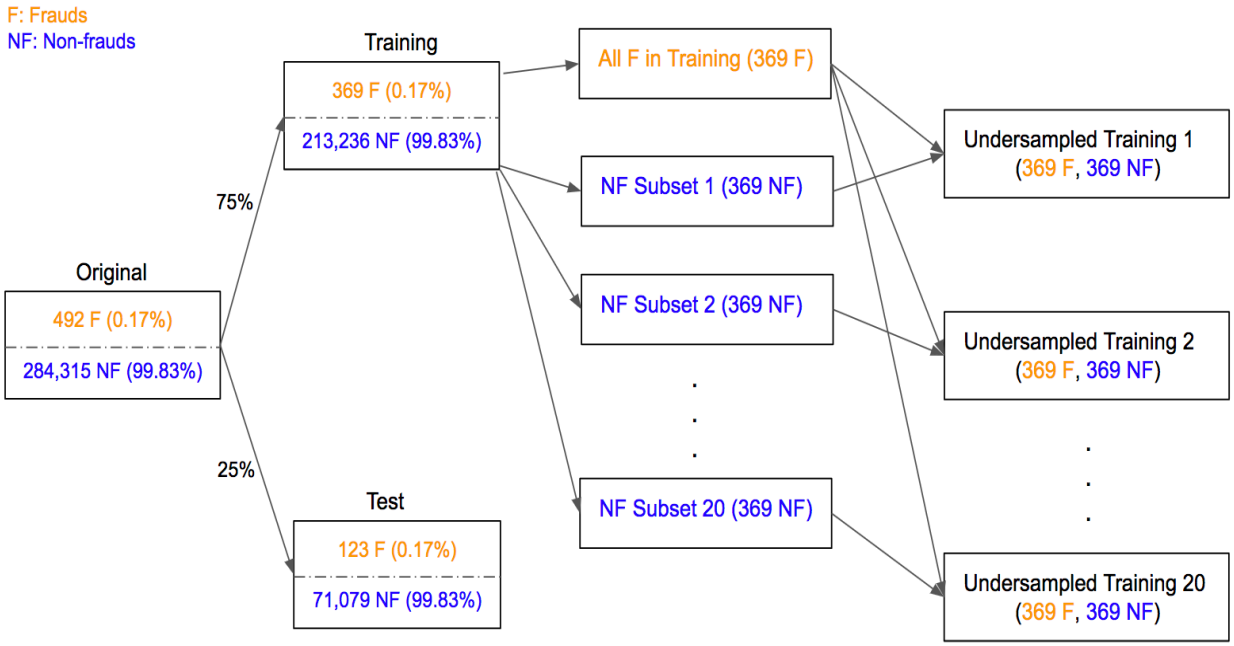

Figure 1: Illustration of Undersampling Approach

More specifically, 20 disjoint subsets of the non-frauds are sampled from the training dataset as indicated in Figure 1. Appending each of the 20 samples of non-frauds to all the frauds in the training data yields an undersampled and balanced training set. The undersampled training sets share the same fraudulent transactions and have different non-frauds subsets taken from the original dataset. Using each

undersampled training set to train a model will create 20 classifiers of the same type, and applying them on the test dataset will eventually produce 20 predicted outcomes (0 or 1) for every transaction in the test dataset. Performing a majority vote on those predictions will determine the final prediction of the transaction ("fraud", if more than 10 1's out of the 20 outcomes; "non-fraud", otherwise). This voting system makes this approach an ensemble-like learning algorithm since the algorithms that take place in the voting are all of the same type.

3) Oversampling: this approach is to generate new samples in the class (the fraudulent transactions in this case) which is under-represented and to make the two classes balanced. This is illustrated in Figure 2.

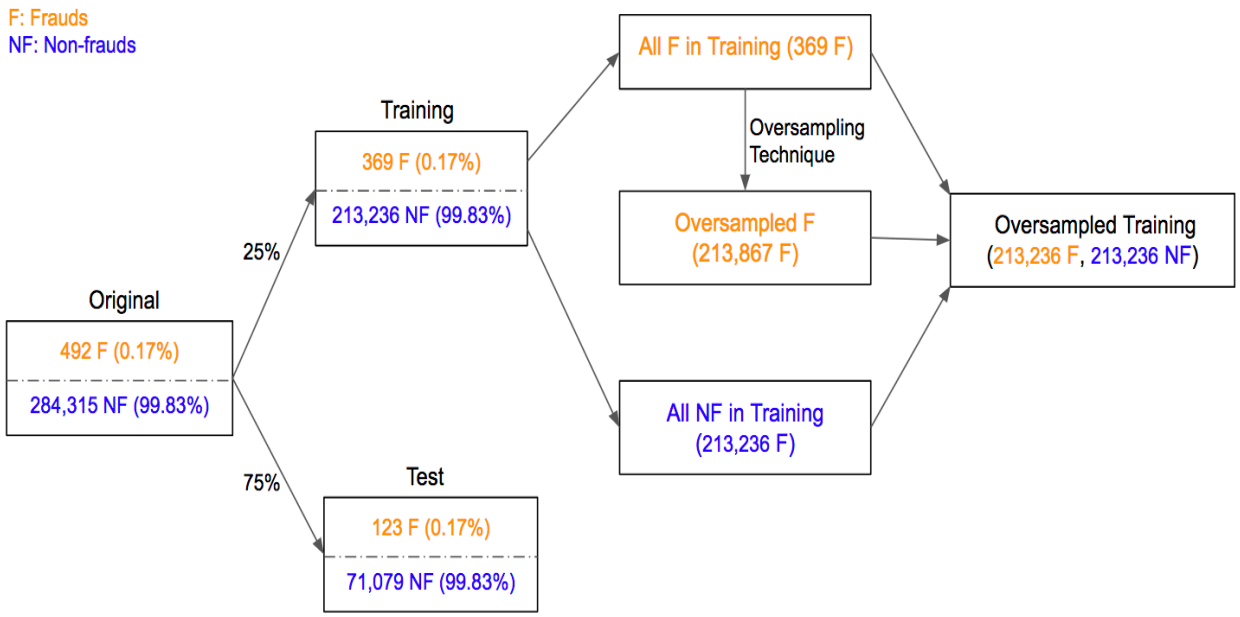

Figure 2: Illustration of Oversampling Approach

The imbalanced-learn package[2] provides a variety of resampling techniques for imbalanced datasets. Three different oversampling techniques are applied in this project:

1. Random Oversampling (RO): randomly sample, with replacement, the current available samples, so each fraud instance from the original dataset will be replicated multiple times to generate the new dataset.

[2] http://contrib.scikit-learn.org/imbalanced-learn/stable/index.html

2. Synthetic Minority Oversampling Technique (SMOTE): creates "synthetic" examples by considering a sample from the original dataset and its *k* nearest neighbors. Randomly select one of those *k* neighbors and take the vector between the selected neighbor and the current sample. Multiplying the vector by a random number between 0 and 1, and adding it to the current sample will create a new, synthetic sample. In other words, this created synthetic sample lies on the line segment joining the current sample and one of its *k* nearest neighbors.

3. Adaptive Synthetic (ADASYN): like SMOTE, ADASYN also creates synthetic data points for the minority class but it focuses on generating samples next to the original samples which are wrongly classified using a k-Nearest Neighbors classifier (these samples are called "difficult to learn"). ADASYN places more weight on minority class examples that are harder to learn and hence shifts the decision boundary towards them.

Logistic regression is trained for all three approaches and *L2* norm regularization is used. Five-fold cross validation is applied to tune the regularization parameter for logistic regression, where 0.01, 0.1, 1, 10, and 100 are the candidates considered. Random forest is trained for the baseline and undersampling approach, but it is not currently used for oversampling approach due to the long execution time it requires. Each integer in the closed interval [40, 50] is considered as the number of trees in every random forest and "sqrt" is assigned to *max_features*, i.e. 5 features are checked when looking for a split. As discussed above, for the undersampling approach, 20 logistic regressions and random forests are trained, respectively.