Credit card fraud detection

Mislav Spajić
mislav.spajic@racunarstvo.hr

December 2020

1 Problem description

Usage of credit cards on the internet has been rapidly growing in the past two decades and today it is the *de facto* standard of the internet shopping. It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Many anonymized datasets of credit card transaction data are offered on the internet so people can try to tackle this issue by successfully classifying fraudulent and non-fraudulent transaction using given features. I will try to solve this issue on a dataset credited to the Machine Learning Group at the Free University of Brussels (Université Libre de Bruxelles), which was downloaded from kaggle.com.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where I have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, author of the dataset couldn't provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise. In this paper, I will try to predict 'Class' value using features mentioned before.

creditcard.csv (143.84 MB)

Detail Compact Column

# Time	# V1	# V2	# V3	# V4
0	-1.3598071336738	-0.07278117330984497	2.536346737969144	1.3781552242744 3
0	1.1918571113148 6	0.2661507120596 3	0.1664801133532 1	0.4481540784609 11
1	-1.3583540615982 3	-1.3401630747360 9	1.7732093426311 9	0.3797795930343 28
1	-0.9662717115720 87	-0.1852260080828 98	1.7929933957872	-0.8632912750364 53
2	-1.1582330934952 3	0.8777367548484 51	1.548717846511	0.4030339339551 21
2	-0.4259658844124 54	0.9605230448829 85	1.1411093423221 9	-0.1682520797603 02
4	1.2296576345079 3	0.1410035070493 26	0.0453707735899 449	1.2026127367359 4
7	-0.6442694423481 46	1.4179635454738 5	1.0743803763556	-0.4921990184950 15
7	-0.8942860822028 2	0.2861571962765 44	-0.1131922127298 71	-0.2715261300886 04
9	-0.3382617524257 5	1.1195933764156 6	1.0443665515731 6	-0.2221872767382 96

Figure 1: My dataset

2 Aim and hypothesis

All of the credit card transactions can be classified in two classes: fraudulent and non-fraudulent. They are classified in the same manner in the dataset I will be working with. Using other data I will try to predict exact class. Although features V1, V2, ... V28 don't look useful, they are principal components of the original features obtained with PCA and they come from real transactions. Our hypothesis is that there is link between these features and classes of the transactions, and that there is a way to successfully predict class of the particular transaction using given features.

3 Materials, methodology and research plan

Using supervised learning, I will try to successfully predict which transactions were fraudulent and which weren't. Firstly, I will split the given data in 70-30 % ratio. I will use the 70% as training data on which I will train different models.

Classification algorithms I will be using are logistic regression, k-nearest neighbours, decision tree and random forest as ensemble of decision trees. Also, another representative of ensemble learning will be fitted, XGBoost classifier. Rest of the data i.e. 30% I will be using as the test data on which I will evaluate the models. Because this is a problem of imbalanced classification I will need to use random undersampling or oversampling technique on the train-

ing data to create a sub-sample and change the composition of samples and avoid overfitting of our models. I want to get a sub-sample with a 50/50 ratio of fraud and non-fraud transactions. As random undersampling is based on removing of the data and in our case it will cause a huge information loss, I decided to go with 'SMOTE' oversampling. 'SMOTE' stands for Synthetic Minority Over-sampling Technique. Unlike random undersampling, 'SMOTE' creates new synthetic points in order to have an equal balance of the classes. I will use grid search with stratified k-fold cross-validation for hyperparameter tuning of some of our models. Cross-validation should be done before oversampling because otherwise I will be directly influencing the validation set before implementing cross-validation causing a data leakage problem. I need to use k-fold CV and then oversample all the training folds separately, by only looking at that folds data. Given the class imbalance ratio and importance of predicting a fraudulent transaction, I will be evaluating the model performances using recall, precision and f1 score. To ensure there is no data leakage I constructed a pipeline. Complete pipeline can be seen in the figure below.

```
# For Time and Amount we use MinMax scaler and for V features robust scaler
scaler = ColumnTransformer(
    transformers=[
        ('scaler1', MinMaxScaler(), ['Time','Amount']),
        ('scaler2', RobustScaler(), X_train.columns.tolist()[1:29])
   ])

# For handling class imblance - SMOTE oversampler
oversampler = SMOTE(random_state=seed,n_jobs=-1)

#Various classification models
LR = LogisticRegression(max_iter=1000,random_state=seed)
kNN = KNeighborsClassifier(n_jobs=-1)
DT = DecisionTreeClassifier()
XGB = XGBClassifier(use_label_encoder=False,eval_metric='aucpr',random_state=seed,n_jobs = -1)
RF = RandomForestClassifier(random_state=seed,n_jobs=-1)



#Define the pipeline
steps = list()
steps.append(('scaler', scaler))
steps.append(('oversampler', oversampler))
steps.append(('classifier', LR))
#LR is just a placeholder

#Initializing imblearn pipeline
pipeline = Pipeline(steps=steps)
```

Figure 2: Pipeline

4 Evaluation of results

All base models will be trained and evaluated with a 3-fold Stratified CV. Main metrics for evaluation are Recall, Precision and f1 score, as accuracy is not useful because of the imbalance. Loop for training and evaluating base models can be seen in figure below.

```
splits=3
skf = StratifiedKFold(n_splits=splits)
scores={}
for name, classifier in zip(names, classifiers):
    scores[name]={}
    scores[name]['Recall']=[]
    scores[name]['Precision']=[]
    scores[name]['f1']=[]
    for train_index, test_index in skf.split(X_train, y_train):

        X_train_fold, X_valid_fold = X_train.iloc[train_index,:], X_train.iloc[test_index,:]
        y_train_fold, y_valid_fold = y_train.iloc[train_index], y_train.iloc[test_index]
        steps[2]=('classifier', classifier))
        pipeline = Pipeline(steps=steps)
        pipeline.fit(X_train_fold, y_train_fold)

        scores[name]['Recall'].append(recall_score(y_valid_fold, pipeline.predict(X_valid_fold)))
        scores[name]['Precision'].append(precision_score(y_valid_fold, pipeline.predict(X_valid_fold)))
        scores[name]['f1'].append(f1_score(y_valid_fold, pipeline.predict(X_valid_fold)))
```

Figure 3: Base models training

Results of base models training are included in figure below.

```
------------------Model:LR------------------
0.91 recall with a standard deviation of 0.02
0.07 precision with a standard deviation of 0.01
0.12 f1 with a standard deviation of 0.01
------------------Model:kNN-----------------
0.86 recall with a standard deviation of 0.03
0.43 precision with a standard deviation of 0.02
0.58 f1 with a standard deviation of 0.02
------------------Model:DT------------------
0.77 recall with a standard deviation of 0.04
0.39 precision with a standard deviation of 0.01
0.52 f1 with a standard deviation of 0.01
------------------Model:RF------------------
0.82 recall with a standard deviation of 0.01
0.87 precision with a standard deviation of 0.03
0.84 f1 with a standard deviation of 0.02
------------------Model:XGB-----------------
0.84 recall with a standard deviation of 0.01
0.77 precision with a standard deviation of 0.04
0.80 f1 with a standard deviation of 0.02
```

Figure 4: Base models results

It can be can see that the best base models are RF and XGB classifier. The worst ones are Logistic Regression and Decision Tree. Since Logistic Regression has great recall and its training doesn't last long I will include it into hyperparameter optimization, Decision Tree will be excluded. Utilizing sklearn GridSearchCV I tried to find better parameters, parameter grid can be seen in the figure below.

```python
# Defining models that will go into loop for grid search and coresponding parameters
names = ['LR','kNN','RF','XGB']
classifiers = [LR,kNN, RF,XGB]

parameters = [
    {

        'classifier__solver' : ['sag','newton-cg','lbfgs'],
        'classifier__C' : [0.01, 0.1, 1, 10]
    },
    {

        'classifier__n_neighbors': [3, 4, 5, 6, 7],
        'classifier__p': [1, 2],
        'classifier__weights': ['uniform', 'distance']
    },
    {

        'classifier__n_estimators': [100, 200, 400],
        'classifier__max_depth': [5, 10, None],
        'classifier__min_samples_split': [2, 5, 10, None],

    },
    {

        'classifier__n_estimators': [100, 150, 200],
        'classifier__min_child_weight': [1, 5, 10],
        'classifier__subsample': [0.6, 0.8, 1.0],
        'classifier__max_depth': [3, 4, 5, 6]

    }

]
```

Figure 5: Parameter grid

Best parameters can be seen in the figure below.

```
{'LR': {'Best_Params': {'classifier__C': 0.01, 'classifier__solver': 'sag'},
  'Best_Score': 0.130166793273526 3},
 'kNN': {'Best_Params': {'classifier__n_neighbors': 3,
   'classifier__p': 1,
   'classifier__weights': 'distance'},
  'Best_Score': 0.6846738833335836},
 'RF': {'Best_Params': {'classifier__max_depth': None,
   'classifier__min_samples_split': 2,
   'classifier__n_estimators': 400},
  'Best_Score': 0.851732624213402},
 'XGB': {'Best_Params': {'classifier__max_depth': 6,
   'classifier__min_child_weight': 1,
   'classifier__n_estimators': 200,
   'classifier__subsample': 0.8},
  'Best_Score': 0.814604811152206}}
```

Figure 6: Best parameters for each model

After hyperparameter tuning, my scores increased, Random Forest stayed the best algorithm.Final evaluation of this model was done on the test set and it can be seen in the figures below.

```
print("Recall on test set is:", recall_score(y_test, rf_final_pipeline.predict(X_test)))
print("Precision on test set is :", precision_score(y_test, rf_final_pipeline.predict(X_test)))
print("f1 score on test set is:", f1_score(y_test, rf_final_pipeline.predict(X_test)))
```

```
Recall on test set is: 0.7905405405405406
Precision on test set is : 0.8602941176470589
f1 score on test set is: 0.8239436619718311
```

Figure 7: Test scores

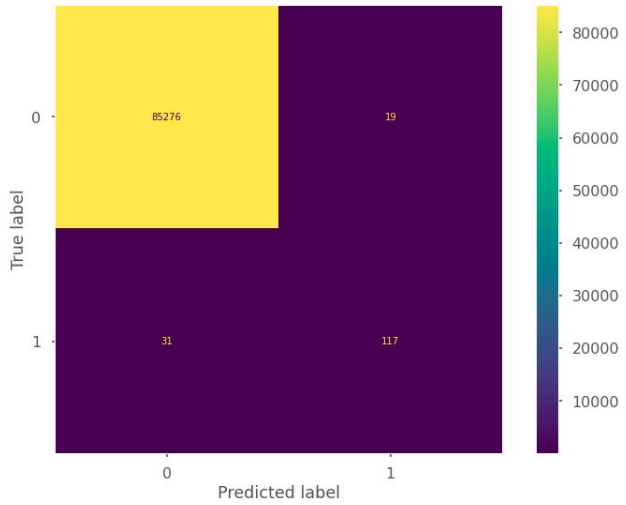

Figure 8: Confusion matrix

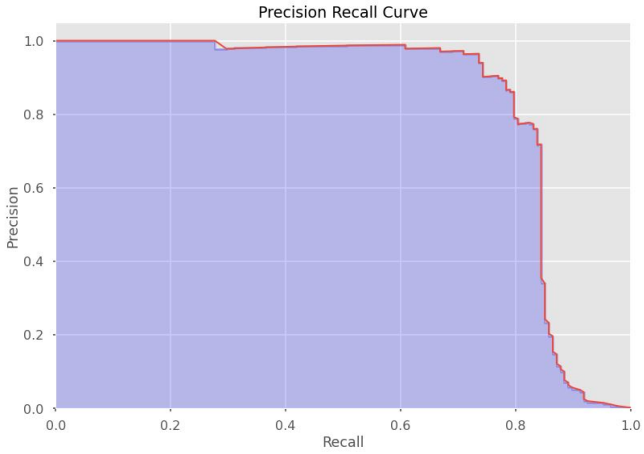

Figure 9: AUPRC

5 Conclusion

Final f1 score of 0.82 of the best model developed (Random Forest based) isn't the best but it isn't terrible either. It was interesting to see that Random Forest outperformed XGB classifier even with tuned hyperparameters.