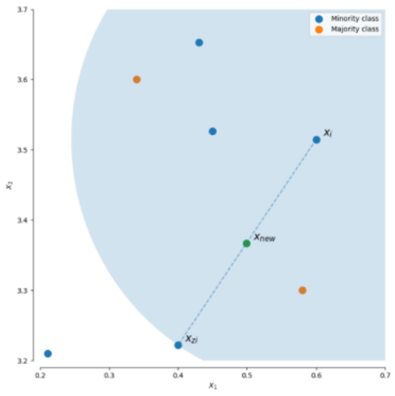
Q4 Fraud Transaction Detection

Author: Jiyao Wang 20797324

Date: 2021.12.13

For this task, firstly, I did a data distribution exploration. I found out that there was an imbalance in label. So I opted SMOTE over-sample algorithm to augment that part of data. Following is a example of SMOTE:



The SMOTE algorithm is called a synthetic minority oversampling technique. It is a technology that creates artificial data by introducing non-replicable minority classes that will not cause information loss based on the similarity of feature spaces between existing minority classes. Introducing new examples is an effective way to change learner biases and create more general biases, mainly in minority groups. The K-NN algorithm is used to infer and create new minority examples from the current imbalances in the minority categories. K-NN neighbors are randomly selected based on the number of oversampling required. Adding synthetically generated minority class examples can create more balance in class distribution.

```
Number of positive samples = 7506
Number of negative samples = 1289169

Number of positive samples = 50000
Number of negative samples = 1289169
```

Then I did a data prepocessing. To solve those datetime features, I transferred them into more features. For example, for feature 'trans_date_trans_time', I transferred it to 'dayname', 'minute', 'second'. After that, I dropped those useless features depends on my priori knowledge. Next, to make data suitable for most of machine learning models' input requirments, I transferred those discrete features which has lower 100 size of different values, and dropped those above 100.

```
ts_objs = np.array([pd.Timestamp(item) for item in np.array(data.trans_date_trans_time)])
data['Ts_obj'] = ts_objs
data['DayName'] = data['Ts_obj'].apply(lambda d: d.day_name())
data['Minute'] = data['Ts_obj'].apply(lambda d: d.minute)
data['Second'] = data['Ts_obj'].apply(lambda d: d.second)
```

After it, I realize there are too many features for one record. So I did a PCA to reduce dimension to 100. PCA, also known as principal component analysis, is defined as an orthogonal linear transformation that transforms the data into a new coordinate system so that the projection of the largest variance data of a scalar falls on the first coordinate (called the first A principal component), the second largest variance is on the second coordinate, and so on.

```
data  =  pd.get_dummies(data,  prefix=discrete_keys,    prefix_sep='_',  columns=discrete_keys)
pca  =  PCA(n_components=100)
data  =  pca.fit_transform(data)
```

To better evaluate and adjust my model, I split train data into train and validation dataset by ratio of 0.1. And to make model perform better, I rescale each dataset according to its distribution of each feature.

```
x_train,x_test,y_train,y_test  =  train_test_split(data,  y,  test_size  =  0.1,random_state=1)
```

For prediction model, XGBoost was choosed. It is an algorithm or engineering implementation based on GBDT. The basic idea of XGBoost is the same as GBDT, but some optimizations have been made, such as the second derivative to make the loss function more accurate; the regular term avoids tree overfitting; Block storage can be calculated in parallel. XGBoost is highly efficient, flexible and portable, and has been widely used in data mining, recommendation systems and other fields.

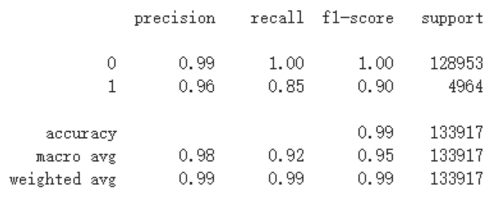
```
from  xgboost  import  XGBClassifier
#xgb  =  XGBClassifier(n_estimators=100,learning_rate=  0.3,max_depth=20,  gamma=0.1,  seed=1)
xbg  =  XGBClassifier(random_state=0)
param  =  {'max_depth':  [100,  80],
                        'n_estimators':  [50,  80,  100],
                  'learning_rate':  [0.1,0.3]
                        }
gsCv  =  GridSearchCV(xbg,  param,  cv=5)
gsCv.fit(x_train,  y_train)

best  =  gsCv.best_params_

xbg  =  XGBClassifier(**best,random_state=0)
xbg.fit(x_train,  y_train)


y_pred=xbg.predict(x_test)
```

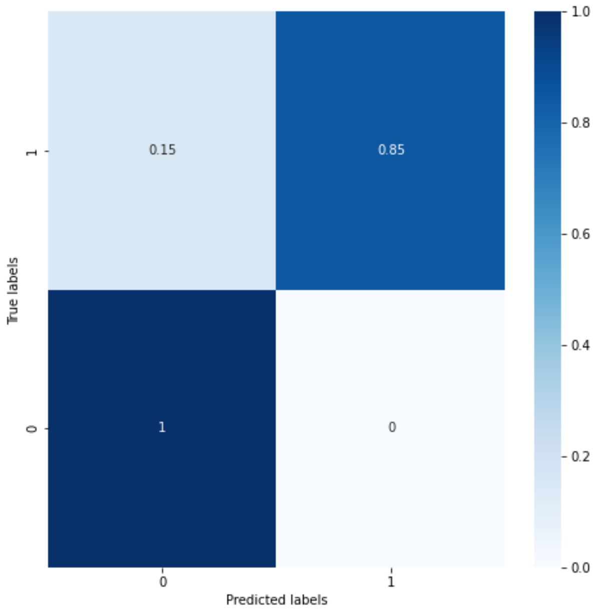
I implemented it by public library. I tested on valid dataset by Accuracy, Recall, F1 score and AUC curve. And I searched its best parameters by grid search. Finally, Its best performance is following:

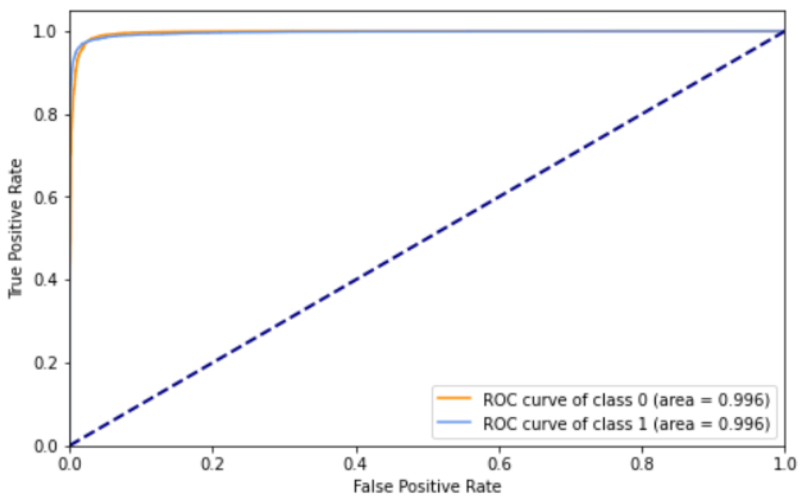


```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    128953
           1       0.96      0.85      0.90      4964

    accuracy                           0.99    133917
   macro avg       0.98      0.92      0.95    133917
weighted avg       0.99      0.99      0.99    133917
```





Reference:
[1] 机器学习之类别不平衡问题 (3) —— 采样方法: https://www.cnblogs.com/massquantity/p/9382710.html
[2] 不平衡数据集的处理: https://www.cnblogs.com/kamekin/p/9824294.html
[3] imblearn document: https://imbalanced-learn.org/stable/index.html
[4] https://zhuanlan.zhihu.com/p/162001079