American Express Report

Jianxiao Yang

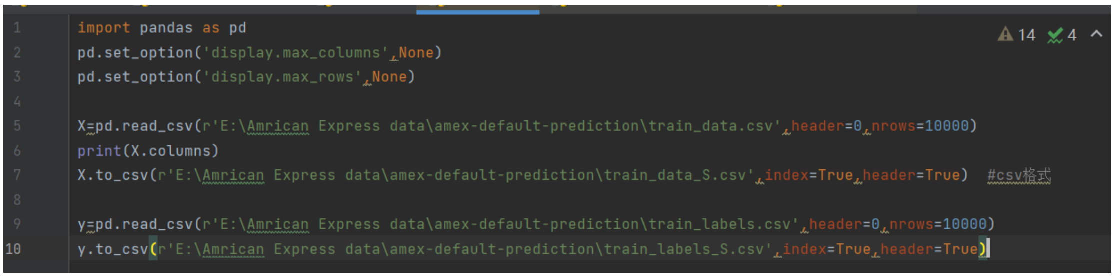
1. introduction

   'American Express' provides over 15G training data and each individual sample has 191 features. The project want participants to predict whether an individual will commit an payment default, which is defined as not paying his due in 120 days. The project is a typical data processing one and data need preprocessing, training, fitting, evaluating, optimizing.

   PS. The codes below are all written by myself, but the most of the idea comes from 1st winner's code, his github link: https://github.com/jxzly/Kaggle-American-Express-Default-Prediction-1st-solution.

2. Data preprocessing

   I've read 1st solution and he offers many delicate way to clean the data. I try to utilize his idea and writing my code but the datasets are too big to run in my computer. So I use pandas read(nrows=) function to take out only 10000 lines to testify if the code is working. I store it as '%X_S.csv' as below.

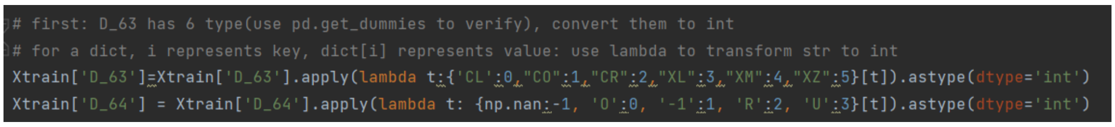


```
import pandas as pd
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)

X=pd.read_csv(r'E:\Amrican Express data\amex-default-prediction\train_data.csv',header=0,nrows=10000)
print(X.columns)
X.to_csv(r'E:\Amrican Express data\amex-default-prediction\train_data_S.csv',index=True,header=True)  #csv格式

y=pd.read_csv(r'E:\Amrican Express data\amex-default-prediction\train_labels.csv',header=0,nrows=10000)
y.to_csv(r'E:\Amrican Express data\amex-default-prediction\train_labels_S.csv',index=True,header=True)
```

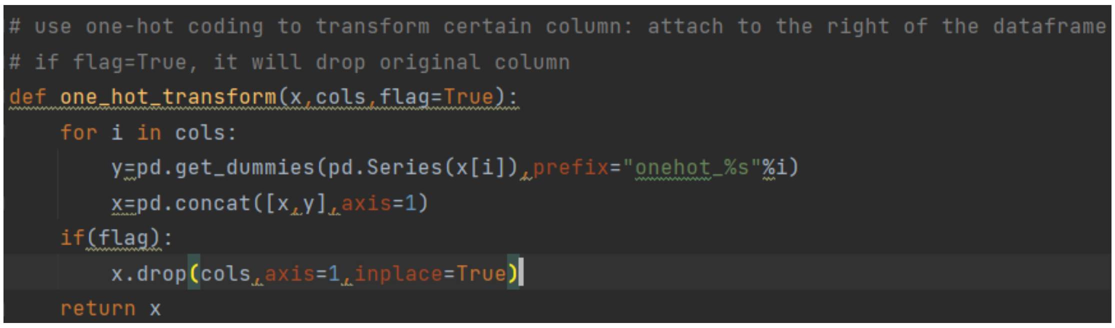
   By observing the data, the author found that there were several columns use String to represent classifications. It is hard to quantify String, so these columns should be turned into Integer. There are at least 2 ways to do it. The first one is to use pandas apply function to turn it into Interger, as below.



```
# first: D_63 has 6 type(use pd.get_dummies to verify), convert them to int
# for a dict, i represents key, dict[i] represents value: use lambda to transform str to int
Xtrain['D_63']=Xtrain['D_63'].apply(lambda t:{'CL':0,"CO":1,"CR":2,"XL":3,"XM":4,"XZ":5}[t]).astype(dtype='int')
Xtrain['D_64'] = Xtrain['D_64'].apply(lambda t: {np.nan:-1, 'O':0, '-1':1, 'R':2, 'U':3}[t]).astype(dtype='int')
```

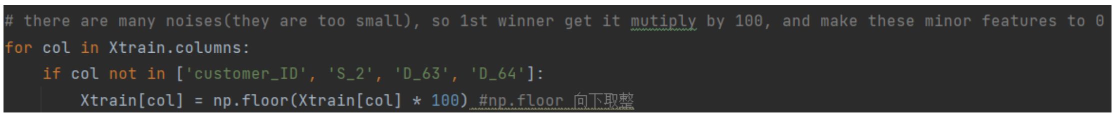
   And the other one is to use one-hot transform, namely, turn the column into a matrix and use independent vector to represent every classification.

   As below, pandas get dummies(column) can return a one-hot matrix and I concatenate with the original data at the end of the column. If necessary, original 'String' column can be dropped.



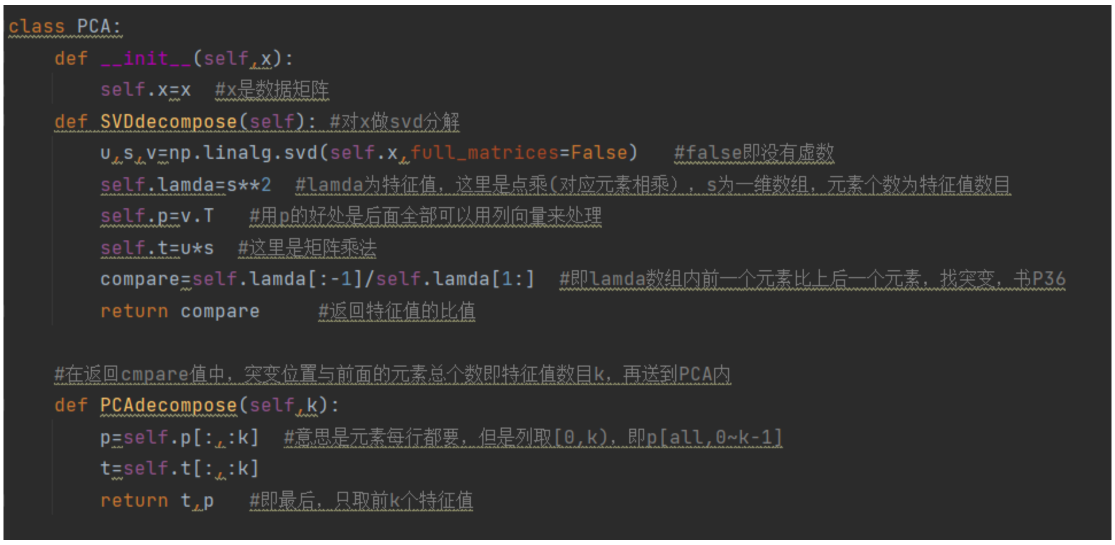
```
# use one-hot coding to transform certain column: attach to the right of the dataframe
# if flag=True, it will drop original column
def one_hot_transform(x,cols,flag=True):
    for i in cols:
        y=pd.get_dummies(pd.Series(x[i]),prefix="onehot_%s"%i)
        x=pd.concat([x,y],axis=1)
    if(flag):
        x.drop(cols,axis=1,inplace=True)
    return x
```

As for the trivial features, my idea is to use PCA to denoise them, but the author use an different and simple method: multiply all features by 100, and escalate those who less than 1. He thought any feature that is below 0.01 is trivial. This method is simple, but efficient.



```python
# there are many noises(they are too small), so 1st winner get it mutiply by 100, and make these minor features to 0
for col in Xtrain.columns:
    if col not in ['customer_ID', 'S_2', 'D_63', 'D_64']:
        Xtrain[col] = np.floor(Xtrain[col] * 100) #np.floor 向下取整
```

But I don't think it is good enough. I may use PCA instead: Using SVD to extract the eigenvalues and then sort them. Define the point where an/a_{n+1} sharply increased as the turning point and make the point after a_{n+1} as trivial. PCA code as below.



```python
class PCA:
    def __init__(self,x):
        self.x=x  #x是数据矩阵
    def SVDdecompose(self): #对x做svd分解
        u,s,v=np.linalg.svd(self.x,full_matrices=False)   #false即没有虚数
        self.lamda=s**2  #lamda为特征值，这里是点乘(对应元素相乘），s为一维数组，元素个数为特征值数目
        self.p=v.T   #用p的好处是后面全部可以用列向量来处理
        self.t=u*s  #这里是矩阵乘法
        compare=self.lamda[:-1]/self.lamda[1:]  #即lamda数组内前一个元素比上后一个元素，找突变，书P36
        return compare      #返回特征值的比值

    #在返回cmpare值中，突变位置与前面的元素总个数即特征值数目k，再送到PCA内
    def PCAdecompose(self,k):
        p=self.p[:,:k]  #意思是元素每行都要，但是列取[0,k)，即p[all,0~k-1]
        t=self.t[:,:k]
        return t,p   #即最后，只取前k个特征值
```

Also, in the datasets, there are many missing values. Normally, there were three ways to fill the blank: copying the value before it, or after it, or using the mean of the column. But there were some columns that most of the values are missing. It is like a unique feature for certain individual and not available for most. I am trying to find a way out so far.

3. Modeling
   As for choosing model, I found 1st winner using heavy ensemble with LGB and NN which I have no idea what that is so far. But I do write some simple models.

   Firstly, it is the most simple one: KNN. I recently learn the model in EC503 class. It is a good classifier with high accuracy if selecting proper K, my KNN coding is in the 'KNN 算法（自己写的）.py' , which I will upload in my github. But I found the datasets are too big and for a O(kn) model, thought it doesn't need training, but the testing part is too big.

   Secondly, it is the common used one for zero-one problem: Logistic Regression. Sklearn provides the package and all I need to is to build a grid and choosing the best C. There are many related types of models, like Linear Regression and polynomial regression. I can use them and set a proper threshold t. If the probability is below t, than y==0(namely, t is a threshold whether individual will commit a default.) But again, huge amounts of data becomes the biggest problem for time-consuming models, like polynomial regression. I don’t know if it can work or not but my next step is to testify them.

In addition, there also are many neural networks model, I am now studying from the beginning: BP network.

4. Optimizing
   In my opinion, when I am going to train a model, I should split an verify dataset from the training set. In sklearn, it provides train_test_split which can easily do that, and I can use K-fold, to training the model with different verify dataset. It can be great if I build up a grid and make the essential parameters self-improved. When the Optimizing is finished, my model will be ready to go.

5. Future plans
   This week, I scan the datasets and try many techniques to clean the data-preprocessing, like PCA, oneHot. Next week, I will finish the preprocessing part and start testing the simple model: KNN, Logistic Regression, Linear/ polynomial Regression with small datasets.(For instance, 10000 rows from the original one.)

   Then, I will start digging into more sophisticated models, like NN,SVM, writing a 3-layer BP model maybe. If it works well, I will look for 1st best solution with heavy ensemble with LGB and NN. Starting digging why he chooses that model and works perfect.

6. Model’s prospect if well trained
   If my model works well, it can cover most of the prediction situation by retrain the model. The core is constant. I don’t want to waste many time digging its future prospect balabala. It is a competition from a real large company and it is used for real situation. That is the thing. If I can do it, I may step into the data industry and it has lots fruits.