Project-personal

JiaXi Jiang

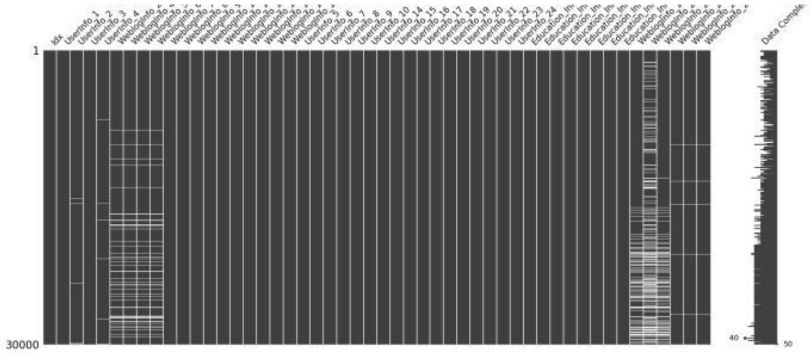
1. Introduction:

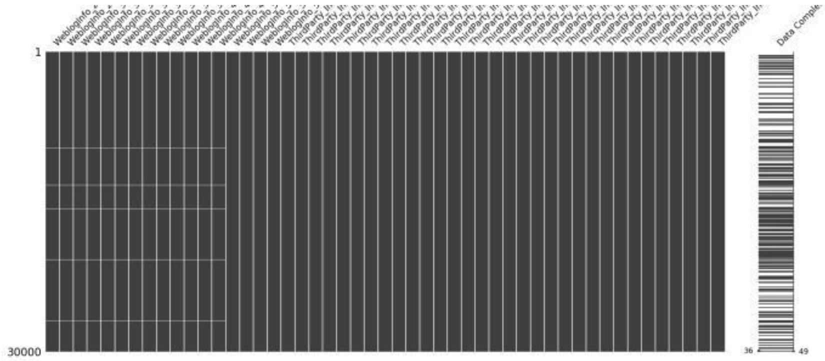
I did some work related to data reprocess, and did the model building

2. 3.4.5:The details of my work:

(1)

I detect the missing values use the 'missingno' packadge to see what the missing value look like below, there may be some patterns and we maybe use some way to predict the missing value, but, for convenient I choose to change the object missing value with mode of its column, and change the int or float value with mean





Code:

```
train_1 = train.iloc[:,0:50]
train_2 = train.iloc[:,51:100]
train_3 = train.iloc[:,101:150]
train_4 = train.iloc[:,150:204]

msno.matrix(train_1,labels= True)
plt.show()
msno.matrix(train_2,labels = True)
```

```
plt.show()
msno.matrix(train_3,labels = True)
plt.show()
msno.matrix(train_4,labels = True)
plt.show()


obj_cols =
list(combined_train_test.select_dtypes(include=['object']).columns)

for col in obj_cols:
     combined_train_test[col] = combined_train_test[col].fillna(value =
combined_train_test[col].mode()[0])

num_cols = list(combined_train_test.select_dtypes(include =
['int64']).columns)
num_cols.append(list(combined_train_test.select_dtypes(include =
['float64']).columns))

for col in num_cols:
    combined_train_test[col] = combined_train_test[col].fillna(value =
combined_train_test[col].mean())
```

(2)

delete something like ‘Telecom operators’ like the AT&T, cause this value has nothing to do with the result action. For example.

Code:

```
combined_train_test = combined_train_test.drop('UserInfo_9',axis=1)
```

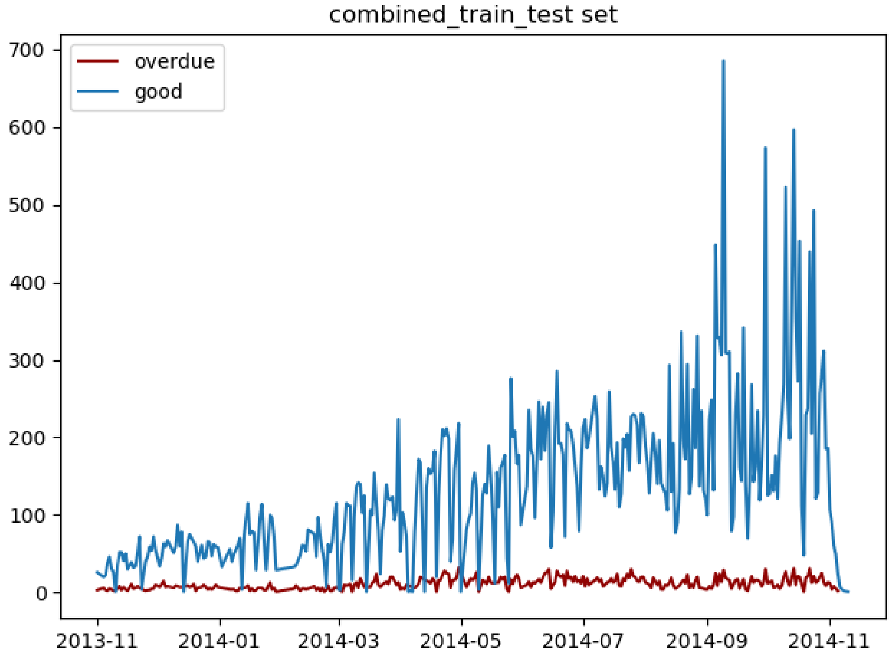
(3)

plot the overdue with date and group by the overdue, 0 or 1, see whether there are some pattern between the date and overdue(which is our target, our Y)

We can see that there did have some relationship between the time and overdue rate. So I

Decide to cut the date into different groups to change the object into numbers.

For 0~len(date) for I in date change 1~10 to 1, 11~20 to 2, ……29991~30000 to 3000

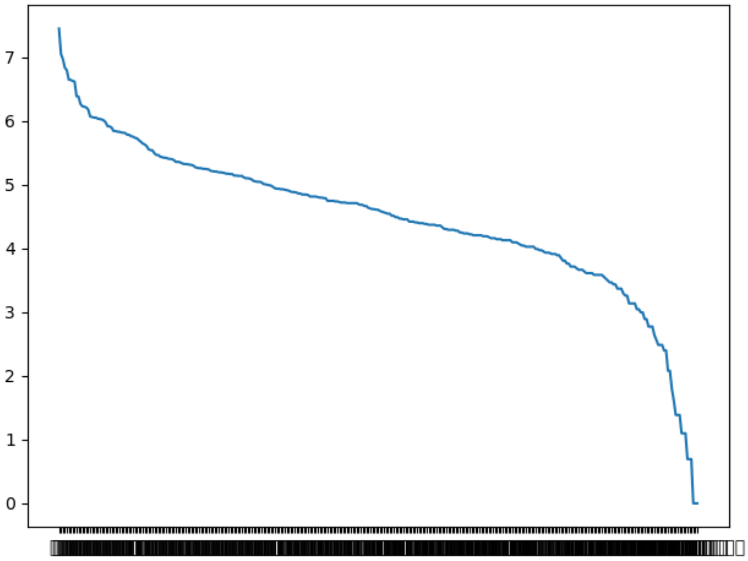


Code:

```
    list_total_1 = list_total[list_total['target'].isin([1])]
list_total_0 = list_total[list_total['target'].isin([0])]
list_total_1 = list_total_1[['ListingInfo']]
list_total_0 = list_total_0[['ListingInfo']]

#change the date in 'object' type to 'date'
list_total_1 =
pd.to_datetime(list_total_1['ListingInfo'],format='%Y %m %d')
list_total_0 =
pd.to_datetime(list_total_0['ListingInfo'],format='%Y %m %d')

#count with the date
list_total_1 = list_total_1.value_counts()
list_total_0 = list_total_0.value_counts()

# remane the label of date and taget,which have been counted and grouped
list_total_1 = list_total_1.rename_axis('date').reset_index(name='counts')
list_total_0 = list_total_0.rename_axis('date').reset_index(name='counts')

#rank the date from samll to big
list_total_1 = list_total_1.sort_values(by = 'date')
list_total_0 = list_total_0.sort_values(by = 'date')

#plot the results
plt.figure()
plt.plot(list_total_1['date'],list_total_1['counts'],color = '#900302')
plt.plot(list_total_0['date'],list_total_0['counts'])
plt.title('combined_train_test set')
plt.legend(('overdue','good'))
plt.show()
```

(4)

See whether the numbers of cities, I mean the repeat times of cities have the relationship with the overdue, I change it into log data, see the pattern, and cut it into 3 groups.

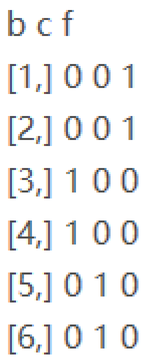


Code:

```
    #see the pattern of numbers of cities
plt.figure()
plt.plot(count_city_1['city'],np.log(count_city_1['counts']))
plt.show()
# from the plot we can see that we can get three categories
count_city_1['counts'] = count_city_1['counts'].apply(np.log)
print(count_city_1)

count_city_1['counts'] = count_city_1['counts'].astype(int)
print(count_city_1)

# from the plot we choose 3 and 4.5 as change point
for i in range(len(count_city_1['counts'])):
   if count_city_1['counts'][i] >= 4.5:
      count_city_1['counts'][i] = 3

   elif count_city_1['counts'][i] <3:
          count_city_1['counts'][i] = 1

     else:
          count_city_1['counts'][i] = 2
print(count_city_1)

#the 3 rank
col_3 = list(count_city_1[count_city_1['counts']==3]['city'])
col_2 = list(count_city_1[count_city_1['counts']==2]['city'])
col_1 = list(count_city_1[count_city_1['counts']==1]['city'])
print(col_3)
print(col_2)
print(col_1)
```

(5)

dummy variables, I did wrong in dealing with the dummy variables. Cause I just use the pd.get_dummies function to change the rest of objects into dummy variables, which I should not like ,what I should do is to use the pd.factorize, pd.qcut to treat different type of variables, maybe I will do that in the next Kaggle game. The essence of it is just change the object group into 1,0 columns.



```
 b c f
[1,] 0 0 1
[2,] 0 0 1
[3,] 1 0 0
[4,] 1 0 0
[5,] 0 1 0
[6,] 0 1 0
```

Code:

```
    def dummy_change(data):
  dummy_UserInfo_22 =
pd.get_dummies(data['UserInfo_22'],prefix='UserInfo_22')
  dummy_UserInfo_23
=pd.get_dummies(data['UserInfo_23'],prefix='UserInfo_23')
  dummy_UserInfo_24
=pd.get_dummies(data['UserInfo_24'],prefix='UserInfo_24')
  dummy_Education_Info2
=pd.get_dummies(data['Education_Info2'],prefix='Education_Info2')
  dummy_Education_Info3
=pd.get_dummies(data['Education_Info3'],prefix='Education_Info3')
  dummy_Education_Info4
=pd.get_dummies(data['Education_Info4'],prefix='Education_Info4')
```

```
  dummy_Education_Info6
=pd.get_dummies(data['Education_Info6'],prefix='Education_Info6')
  dummy_Education_Info7
=pd.get_dummies(data['Education_Info7'],prefix='Education_Info7')
  dummy_Education_Info8
=pd.get_dummies(data['Education_Info8'],prefix='Education_Info8')
  dummy_WeblogInfo_19
=pd.get_dummies(data['WeblogInfo_19'],prefix='WeblogInfo_19')
  dummy_WeblogInfo_20
=pd.get_dummies(data['WeblogInfo_20'],prefix='WeblogInfo_20')
  dummy_WeblogInfo_21
=pd.get_dummies(data['WeblogInfo_21'],prefix='WeblogInfo_21')


  data =
pd.concat([data,dummy_UserInfo_22,dummy_UserInfo_23,dummy_UserInfo_24,dummy
_Education_Info2
                 ,dummy_Education_Info3,dummy_Education_Info4,dummy_Educati
on_Info6,dummy_Education_Info7
                 ,dummy_Education_Info8,dummy_WeblogInfo_19,dummy_WeblogInf
o_20,dummy_WeblogInfo_21
                 ],axis = 1)
  return(data)

combined_train_test = dummy_change(combined_train_test)
print(combined_train_test)
print(combined_train_test.info())
#delete features that we dealt with before
delete_cols_1 = ['UserInfo_22','UserInfo_23', 'UserInfo_24',
'Education_Info2', 'Education_Info3',
'Education_Info4','Education_Info6','Education_Info7'
              ,'Education_Info8','WeblogInfo_19','WeblogInfo_20','WeblogInf
o_21']
for i in delete_cols_1:
    combined_train_test= combined_train_test.drop(i, axis=1)

print(combined_train_test)
```

(6)

made models: for this part, I almost copy from the internet, maybe changed some parameters or models. For the part before, they all made by my self.

1. Choose the number of features that we have selected before

```
       n_samples:toal number of samples
```

n_informative: Number of multi-information features

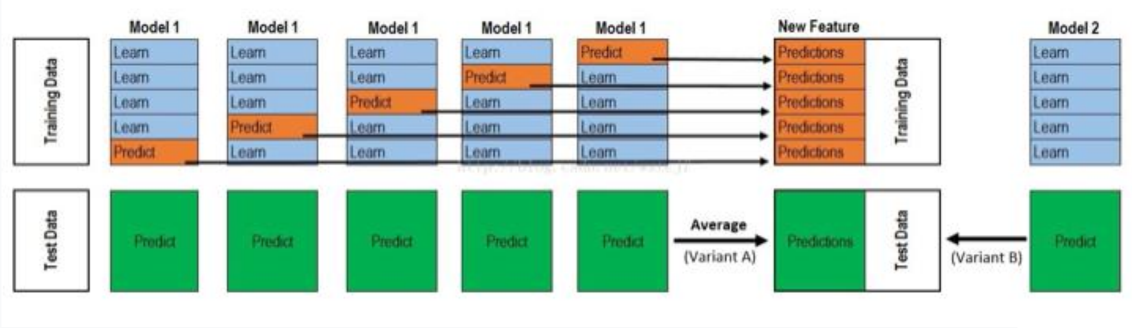
n_redundant: Redundant information, random linear combination of informative features

n_repeated: Repeat information, randomly extract n_informative and n_redundant features

n_features = n_informative + n_redundant+ n_repeated

2. Get the stacking model

What is the stacking model mean? For example:



what is the stacking? Let us use a example to explain that.

The first half is to use a basic model for 5-fold cross-validation, such as: using XGBoost as the basic model Model1, 5-fold cross-validation is to take out four folds as training data (training set), the other one as validation data (validation set). Note: In stacking, this part of data will use the entire training data. For example: suppose our entire training data contains 10000 rows of data, and test data contains 2500 rows of data, then each cross-validation is actually to divide the training data from the training set and the verification set. In each cross-validation, the training data will be 8000 rows, validation data is 2000 rows.

Each cross-validation consists of two processes: 1. Train the model based on training data; 2. Predict validation data based on the model generated by training data training. After the entire first cross-validation is completed, we will get the predicted value of the current validation data, which will be a 2000-row, 1-column data, denoted as a1. note! After this part of the operation is completed, we also need to make predictions on the original test data of the data set. This process will generate 2500 predicted values. This part of the predicted values will be used as part of the next layer of model test data, denoted as b1. Because we are performing 5-fold cross-validation, the above-mentioned process will be carried out five times, and finally will generate 2000 rows and 5 columns of data a1, a2, a3, a4, a5 predicted for validation data data. The forecast will be 2500 rows and 5 columns of data b1, b2, b3, b4, b5.

After completing the entire steps for Model1, we can find that a1, a2, a3, a4, and a5 are actually the predicted values of the original training data. Putting them together, a matrix of 10,000 rows and one column will be formed, denoted as A1 . For the data of b1, b2, b3, b4, and b5, we add the average of each part to obtain a matrix with 2500 rows and 1 column, which is denoted as B1.

The above is the complete process of a model in stacking. The same layer in stacking usually contains multiple models. Suppose there are Model2: LR, Model3: RF, Model4: GBDT, and Model5: SVM. For these four models, we can repeat the above Steps, after the end of the whole process, we can get the new A2, A3, A4, A5, B2, B3, B4, B5 matrix

After this, we merged A1, A2, A3, A4, A5 side by side to get a 10000 rows and 5 columns matrix as training data, B1, B2, B3, B4, B5 side by side merged and got a 2500 rows and 5 columns matrix as test data. Let the next layer of models be further trained based on them.

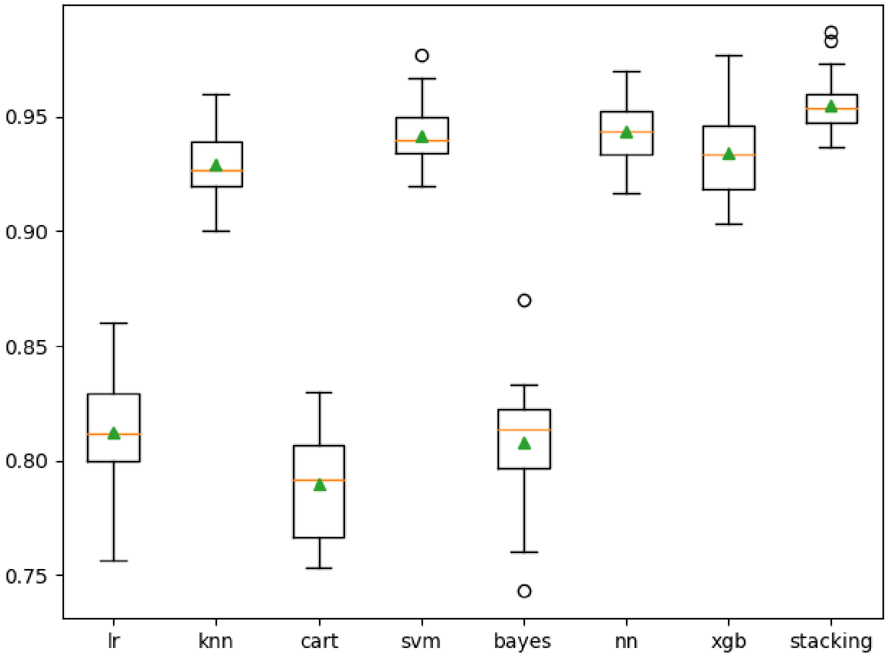
In the next layer of model training, the matrix of 10000 rows and 5 columns is used as the feature matrix of the model, and the True Labels of the most original training data is used as the output matrix of the model for model training. After the model training is completed, the processed 2500 rows and 5 columns of test data are used as the feature matrix, and the model is used to output 2500 rows and 1 column as the result.

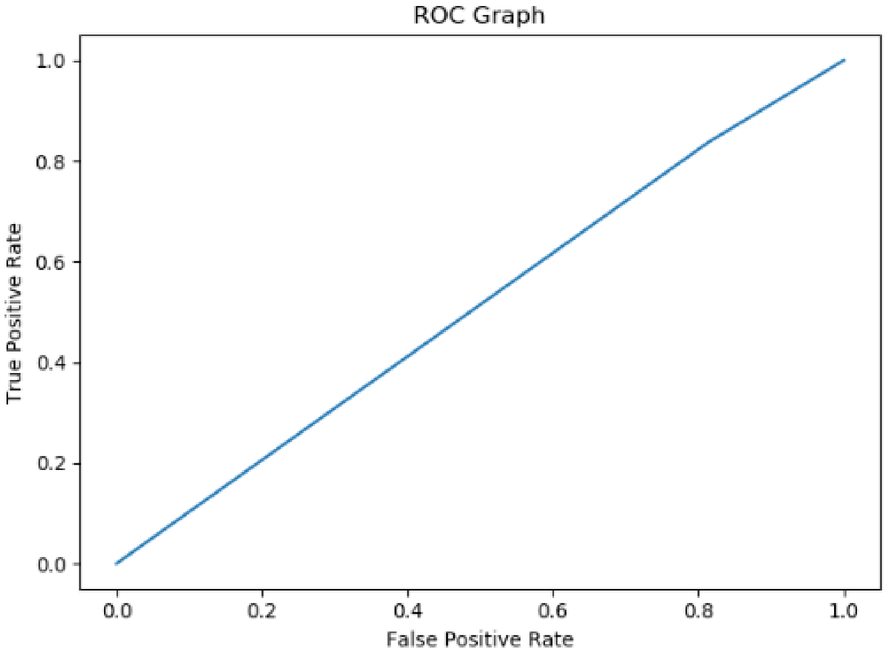
3. Get the individual models

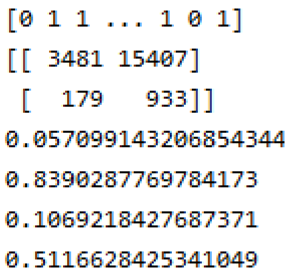
   Like knn, LogisticRegression, DecisionTree, svm, nn, xgb, also the stacking

4. Kfold, 10-fold

5. Got the mean absolute error (MAE). The scikit-learn library inverts the sign on this error to make it maximizing, from -infinity to 0 for the best score. The f1 score. The CSV. The AUC graph. To see the result of my model.







```
[0 1 1 ... 1 0 1]
[[ 3481 15407]
 [  179   933]]
0.057099143206854344
0.8390287769784173
0.1069218427687371
0.5116628425341049
```

Code:

```
 def get_dataset():
X, y = make_classification(n_samples=3000, n_features=50,
n_informative=15, n_redundant=5)
return X, y

 def get_stacking():
# define the base models
level0 = list()
level0.append(('lr', LogisticRegression()))
level0.append(('knn', KNeighborsClassifier()))
level0.append(('cart', DecisionTreeClassifier()))
level0.append(('svm', SVC()))
level0.append(('bayes', GaussianNB()))
level0.append(('nn',MLPClassifier(hidden_layer_sizes=(100,),
activation='relu',
                  solver='lbfgs', alpha=0.9, batch_size='auto',
                  learning_rate='constant')))
# define meta learner model
level1 = XGBClassifier(learning_rate= 0.1,
                   n_estimators= 1000,
                   max_depth= 10,
                   min_child_weight= 1,
                   subsample= 0.7,
                   colsample_bytree= 0.9,
                 )
# define the stacking ensemble
model = StackingClassifier(estimators=level0, final_estimator=level1,
cv=5)
model.fit(X, y)
return model
```

we choose the logisticregression, knn, decisiontree, svm, neuron network, bayes,xgb, also, we use the10-fold cross-validation, also we can plot the result with the box-plot.

```
# get a list of models to evaluate
def get_models():
   models = dict()
   models['lr'] = LogisticRegression()
   models['knn'] = KNeighborsClassifier()
   models['cart'] = DecisionTreeClassifier()
   models['svm'] = SVC()
   models['bayes'] = GaussianNB()
   models['nn']        =       MLPClassifier(hidden_layer_sizes=(100,),
activation='relu',
                solver='lbfgs', alpha=0.9, batch_size='auto',
                learning_rate='constant')
  models['xgb'] = XGBClassifier(learning_rate= 0.1,
                  n_estimators= 1000,
                  max_depth= 10,
                  min_child_weight= 1,
                  subsample= 0.7,
                  colsample_bytree= 0.9,
                 )
  models['stacking'] = get_stacking()
  return models
# evaluate a give model using cross-validation
def evaluate_model(model):
   cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
random_state=1)
   scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv,
n_jobs=-1, error_score='raise')
   return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
   scores = evaluate_model(model)
   results.append(scores)
   names.append(name)
   print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()

predict = models['stacking'].predict(test_data_X)
print(predict)
predict = pd.DataFrame(predict)
predict.to_csv('C:/project/data/predict.csv')
```

Conclude:

(5) as we can see from the result, our predict is very bad, it may because we just choose the
Feature, and no matter how I change the parameter, the result just remain bad.

I learned how to clean the data, how to deal with the missing value, the date value, the object value, how to made a model, and how to adjust the parameters. And learned the basic step of model making.

I think the most important failure is the way we pre-process the data, especially the way we treat the dummy variables. Maybe I will redo this project in the future. I have other class in the summer, so that I don't have so much time to prove it. So be it.

(6)

Calculate the percentage of the code:

((60-15)/60+130)*100 = 23.684

(7)

References:

Ensemble learning with Stacking and Blending

By **Hussain Mujtaba** May 30, 2020

Available from: https://www.mygreatlearning.com/blog/ensemble-learning/ [20 June 2020].