Homework 3

Machine Learning Classification Model

*Task: Credit score classification (Kaggle LINK: [URL (https://www.kaggle.com/datasets/parisrohan/credit-score-classification?select=train.csv)](https://www.kaggle.com/datasets/parisrohan/credit-score-classification?select=train.csv))**

In [1]:

```
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest,SelectPercentile,f_classif
from sklearn.feature_selection import f_regression,mutual_info_regression,mutua
from sklearn.feature_selection import SelectFromModel,RFE,VarianceThreshold
```

In [2]:

```
data = pd.read_csv("train.csv") # 读取数据
df = data.copy()
```

```
C:\Users\19436\AppData\Local\Temp\ipykernel_28200\1035956267.py:
1: DtypeWarning: Columns (26) have mixed types. Specify dtype opt
ion on import or set low_memory=False.
  data = pd.read_csv("train.csv") # 读取数据
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   ID                        100000 non-null  object 
 1   Customer_ID               100000 non-null  object 
 2   Month                     100000 non-null  object 
 3   Name                      90015 non-null   object 
 4   Age                       100000 non-null  object 
 5   SSN                       100000 non-null  object 
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  object 
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64  
 10  Num_Credit_Card           100000 non-null  int64  
 11  Interest_Rate             100000 non-null  int64  
 12  Num_of_Loan               100000 non-null  object 
 13  Type_of_Loan              88592 non-null   object 
 14  Delay_from_due_date       100000 non-null  int64  
 15  Num_of_Delayed_Payment    92998 non-null   object 
 16  Changed_Credit_Limit      100000 non-null  object 
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  object 
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object 
 22  Payment_of_Min_Amount     100000 non-null  object 
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object 
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           98800 non-null   object 
 27  Credit_Score              100000 non-null  object 
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

Task 1

- 任务1：完成数据预处理（缺失处理、异常值处理等），及对分析有帮助的可视化分析，并尝试选择合适的变量，阐述选择的过程及原因

In [4]:

```
'''
*******************************************
PREPROCESSING
1. 数据类型处理
*******************************************
'''

def regex_decimal(x):
    '''Extract numbers in text'''
    regex = r'[-+]?\d+(?:\.\d+)?'
    matches = re.search(regex, x)
    if matches != None:
        return float(matches[0])
    else:
        return 0.0

def regex_int(x):
    '''Extract numbers in text'''
    regex = r'[-+]?\d+(?:\.\d+)?'
    matches = re.search(regex, x)
    if matches != None:
        return int(matches[0])
    else:
        return 0

def Num_of_Delayed_Payment(x):
    if isinstance(x, str):
        x = regex_int(x)
    return x

def Changed_Credit_Limit(x):
    if isinstance(x, str):
        x = regex_decimal(x)
    return x

def Amount_invested_monthly(x):
    if isinstance(x, str):
        x = regex_decimal(x)
    return x

def Monthly_Balance(x):
    if isinstance(x, str):
        x = regex_decimal(x)
    return x


# Annual_Income转为数值型
df['Annual_Income'] = df['Annual_Income'].apply(regex_decimal)

# Num_of_Loan
df['Num_of_Loan'] = df['Num_of_Loan'].apply(regex_int)

# Num_of_Delayed_Payment
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].apply(Num_of_Delaye

# Changed_Credit_Limit
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].apply(Changed_Credit_Li
# Outstanding_Debt
df['Outstanding_Debt'] = df['Outstanding_Debt'].apply(regex_decimal)

# Amount_invested_monthly
df['Amount_invested_monthly'] = df['Amount_invested_monthly'].apply(Amount_inve

# 将Monthly Balance中的数据转换为数值型，无法转换的处理为 NaN
df['Monthly_Balance'] = pd.to_numeric(df['Monthly_Balance'], errors="coerce")
```

In [5]:

```
'''
*******************************************
PREPROCESSING
2. 处理缺失值
*******************************************
'''
def same_user_fill(df, col):
    # 用该用户其它月份该列的值填充缺失值
    nan_index = df[df[col].isnull()].index
    for each_index in nan_index:
        Customer_ID = df.loc[each_index, 'Customer_ID']
        # 该用户子表
        temp_df = df[df['Customer_ID'] == Customer_ID]
        notnan = temp_df[temp_df[col].notnull()]
        if len(notnan) > 0:
            if col == 'Monthly_Inhand_Salary' or col == 'Num_Credit_Inquiries':
                value = notnan.loc[notnan.index[0], col]
            elif col == 'Amount_invested_monthly' or col == 'Monthly_Balance':
                value = notnan[col].mean()
        else:
            if col == 'Monthly_Inhand_Salary':
                value = df.loc[each_index, 'Annual_Income'] / 12.0
            else:
                value = df[col].mean()
        df.loc[each_index, col] = value
    return df

# Monthly_Inhand_Salary
# 某个Customer的缺失值，填充为该Customer其它Monthly_Inhand_Salary的值
df = same_user_fill(df, 'Monthly_Inhand_Salary')

# Num_of_Delayed_Payment
# 用0填充
df['Num_of_Delayed_Payment'].fillna(0.0, inplace=True)

# Num_Credit_Inquiries
# 某个Customer的缺失值，填充为该Customer其它Num_Credit_Inquiries的值
df = same_user_fill(df, 'Num_Credit_Inquiries')

# Credit_History_Age
# Nan -> 0; a Year b month -> a*12+b;
df['Credit_History_Age'].fillna(0, inplace=True)

def Credit_History_Age(x):
    if isinstance(x, str):
        regex_year = r'(\d*) Years'
        regex_month = r'(\d*) Months'

        year_num = int(re.search(regex_year, x).groups()[0])
        month_num = int(re.search(regex_month, x).groups()[0])
        x = 12 * year_num + month_num
    return x

df['Credit_History_Age'] = df['Credit_History_Age'].apply(Credit_History_Age)

# Amount_invested_monthly
# 某个Customer的缺失值，填充为该Customer其它Amount_invested_monthly的平均值
```

```
df = same_user_fill(df, 'Amount_invested_monthly')

# Monthly_Balance
# 某个Customer的缺失值，填充为该Customer其它Monthly_Balance的平均值
df = same_user_fill(df, 'Monthly_Balance')
```

In [6]:

```
df.info() # loan
```

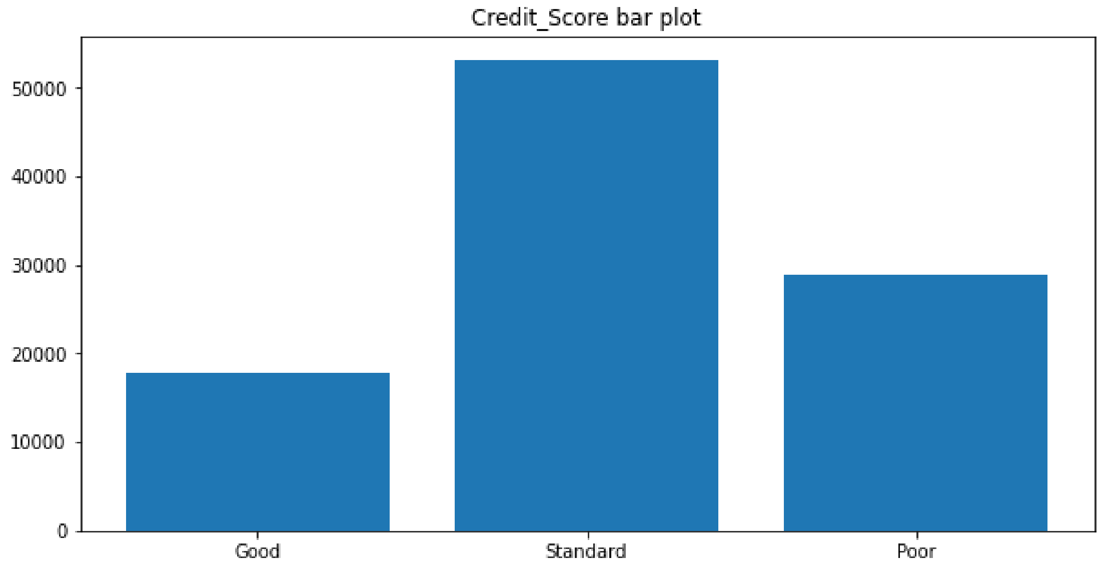
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  object
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  object
 7   Annual_Income             100000 non-null  float64
 8   Monthly_Inhand_Salary     100000 non-null  float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  int64
 13  Type_of_Loan              88592 non-null   object
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    100000 non-null  float64
 16  Changed_Credit_Limit      100000 non-null  float64
 17  Num_Credit_Inquiries      100000 non-null  float64
 18  Credit_Mix                100000 non-null  object
 19  Outstanding_Debt          100000 non-null  float64
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        100000 non-null  int64
 22  Payment_of_Min_Amount     100000 non-null  object
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   100000 non-null  float64
 25  Payment_Behaviour         100000 non-null  object
 26  Monthly_Balance           100000 non-null  float64
 27  Credit_Score              100000 non-null  object
dtypes: float64(10), int64(6), object(12)
memory usage: 21.4+ MB
```

In [7]:

```
'''
*******************************************
Feature Engineering
1. Drop useless column
2. Visualization
*******************************************
'''

# 1. Drop useless column
df.drop(['ID', 'Customer_ID', 'Month', 'Name',
         'Type_of_Loan', 'SSN', 'Occupation', 'Age'], axis=1, inplace=True)
```

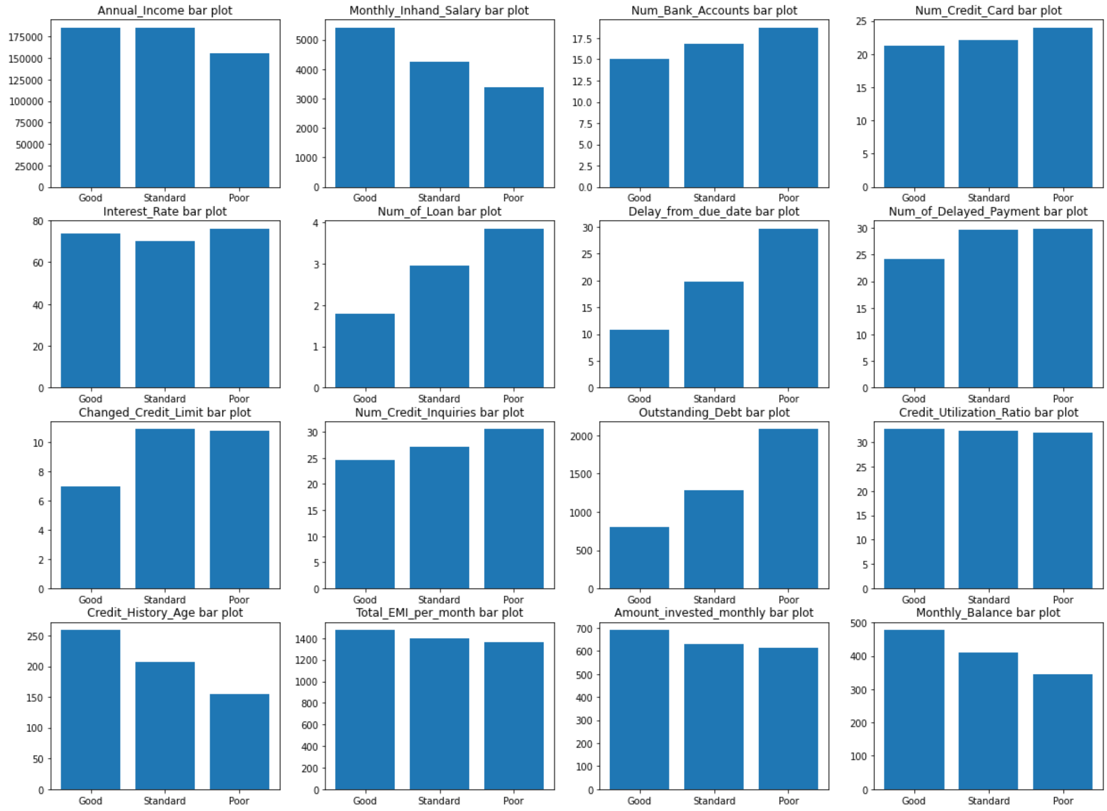
In [29]:

```
# 2. Visualization
# 2.1 绘制各个类别上的数据量图
def credit_score_count_plot(ax, col, df):
    column_cls_list = df['Credit_Score'].unique()
    cls_list_num = [len(df[df['Credit_Score'] == each_cls]) for each_cls
                    in column_cls_list]
    ax.set_title('{} bar plot'.format(col))
    ax.bar(column_cls_list, cls_list_num)
    return ax

f, axs = plt.subplots(1,1, figsize=(10,5))
axs = credit_score_count_plot(axs, 'Credit_Score', df)
plt.show()
```



In [67]:

```
# 2.2 各个目标类别在各个特征值上的violin plot
def credit_score_violin(ax, feature, df):
    column_cls_list = df['Credit_Score'].unique()
    cls_list_num = [df[df['Credit_Score'] == each_cls][feature].mean()
                    for each_cls in column_cls_list]
    ax.set_title('{} bar plot'.format(feature))
    ax.bar(column_cls_list, cls_list_num)
    return ax

df_temp = df.drop(['Payment_of_Min_Amount', 'Credit_Mix', 'Credit_Score', 'Payn
f, axs = plt.subplots(4,4, figsize=(20,15))
for i in range(4):
    for j in range(4):
        axs[i][j] = credit_score_violin(axs[i][j],
                                        df_temp.columns[i*4+j],
                                        df)
```



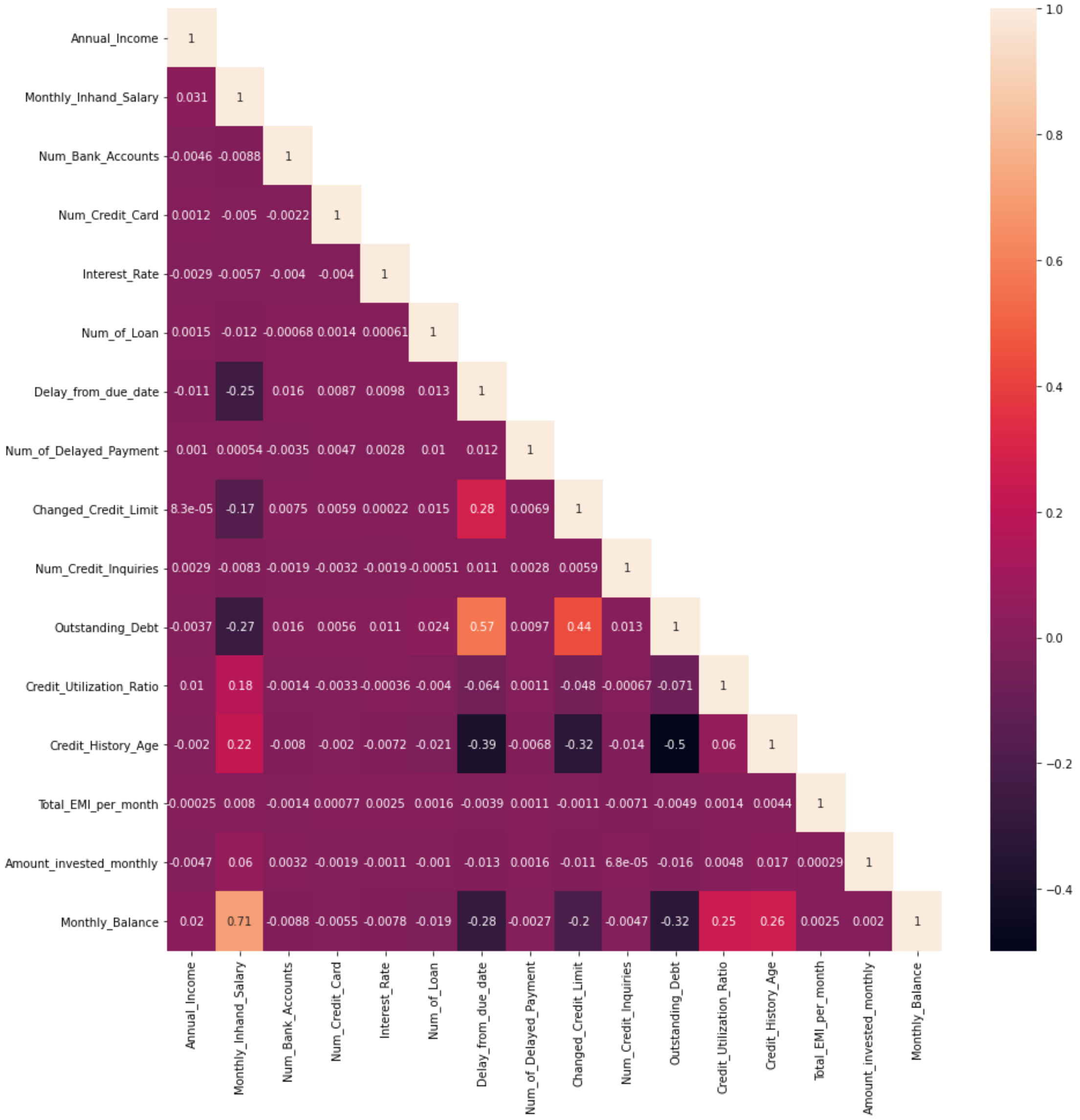
In [30]:

```
# 2.3 Correlation Matrix
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(),annot=True, mask=np.triu(df.corr(),k=1))
```

```
C:\Users\19436\AppData\Local\Temp\ipykernel_28200\3179775809.py:
3: FutureWarning: The default value of numeric_only in DataFrame.
corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(df.corr(),annot=True, mask=np.triu(df.corr(),k=1))
```

Out[30]:

```
<AxesSubplot:>
```



	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Num_Credit_Inquiries	Outstanding_Debt	Credit_Utilization_Ratio	Credit_History_Age	Total_EMI_per_month	Amount_invested_monthly	Monthly_Balance
Annual_Income	1															
Monthly_Inhand_Salary	0.031	1														
Num_Bank_Accounts	-0.0046	-0.0088	1													
Num_Credit_Card	0.0012	-0.005	-0.0022	1												
Interest_Rate	-0.0029	-0.0057	-0.004	-0.004	1											
Num_of_Loan	0.0015	-0.012	-0.00068	0.0014	0.00061	1										
Delay_from_due_date	-0.011	-0.25	0.016	0.0087	0.0098	0.013	1									
Num_of_Delayed_Payment	0.001	0.00054	-0.0035	0.0047	0.0028	0.01	0.012	1								
Changed_Credit_Limit	8.3e-05	-0.17	0.0075	0.0059	0.00022	0.015	0.28	0.0069	1							
Num_Credit_Inquiries	0.0029	-0.0083	-0.0019	-0.0032	-0.0019	-0.00051	0.011	0.0028	0.0059	1						
Outstanding_Debt	-0.0037	-0.27	0.016	0.0056	0.011	0.024	0.57	0.0097	0.44	0.013	1					
Credit_Utilization_Ratio	0.01	0.18	-0.0014	-0.0033	-0.00036	-0.004	-0.064	0.0011	-0.048	-0.00067	-0.071	1				
Credit_History_Age	-0.002	0.22	-0.008	-0.002	-0.0072	-0.021	-0.39	-0.0068	-0.32	-0.014	-0.5	0.06	1			
Total_EMI_per_month	-0.00025	0.008	-0.0014	0.00077	0.0025	0.0016	-0.0039	0.0011	-0.0011	-0.0071	-0.0049	0.0014	0.0044	1		
Amount_invested_monthly	-0.0047	0.06	0.0032	-0.0019	-0.0011	-0.001	-0.013	0.0016	-0.011	6.8e-05	-0.016	0.0048	0.017	0.00029	1	
Monthly_Balance	0.02	0.71	-0.0088	-0.0055	-0.0078	-0.019	-0.28	-0.0027	-0.2	-0.0047	-0.32	0.25	0.26	0.0025	0.002	1

Task2

【至少使用三个分类算法】（如决策树、支撑向量机、boosting方法等）,对Credit score建立分类模型。

可自行将数据划分为测试集与训练集， 可以使用交叉验证的方法，对模型结果根据常用分类指标（如accuracy,precision,recall,f1 score等）进行模型比较与评价。

In [91]:

```
def alter_label(x):
    if x == 'Poor':
        return 0
    elif x == 'Standard':
        return 1
    elif x == 'Good':
        return 2

y = df['Credit_Score'].apply(alter_label)
X = pd.get_dummies(df[['Annual_Income', 'Monthly_Inhand_Salary',
                       'Num_Bank_Accounts','Num_Credit_Card',
                       'Interest_Rate', 'Num_of_Loan',
                       'Delay_from_due_date', 'Num_of_Delayed_Payment',
                       'Changed_Credit_Limit','Num_Credit_Inquiries',
                       'Credit_Mix', 'Outstanding_Debt',
                       'Credit_Utilization_Ratio', 'Credit_History_Age',
                       'Payment_of_Min_Amount', 'Total_EMI_per_month',
                       'Amount_invested_monthly', 'Payment_Behaviour',
                       'Monthly_Balance']])

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    random_state=17,
                                                    test_size=.20)

print(len(X_train), len(X_test))
```

```
80000 20000
```

In [98]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score, precision_score, recall_score, make_score
```

In [105]:

```
'''
Decision Tree
'''
from sklearn.tree import DecisionTreeClassifier


clf = DecisionTreeClassifier(random_state=7)

'''
十折交叉验证
- 将数据集划分为10份，每次取其中一份为validation，9份为train
-
'''
kfold = KFold(n_splits=10)
result = cross_val_score(clf, X_train, y_train, cv=kfold,
                         scoring=make_scorer(f1_score, average='micro'))
print(result)
```

```
[0.701625 0.687625 0.700125 0.694375 0.708875 0.689625 0.689
0.693625
 0.69425  0.690375]
```

In [109]:

```
'''
Support Vector Machine
'''
from sklearn.svm import LinearSVC


clf = LinearSVC(penalty='l2', random_state=7, multi_class='ovr')

kfold = KFold(n_splits=10)
result = cross_val_score(clf, X_train, y_train, cv=kfold,
                         scoring=make_scorer(f1_score, average='micro'))
print(result)
```

Out[109]:

```
array([0.58425 , 0.2215  , 0.30525 , 0.3135  , 0.41825 , 0.5365
,
       0.493875, 0.47475 , 0.539375, 0.444   ])
```

In [110]:

```
'''
Boosting Method
'''
from sklearn.ensemble import GradientBoostingClassifier


clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
                                 max_depth=1, random_state=0)

kfold = KFold(n_splits=10)
result = cross_val_score(clf, X_train, y_train, cv=kfold,
                         scoring=make_scorer(f1_score, average='micro'))
print(result, result.mean())
```

```
[0.67525  0.677375 0.676125 0.680625 0.685875 0.662875 0.680125
0.676375
 0.677125 0.677125] 0.6768875
```

上述三个模型，经过10折交叉验证，并使用f1值判断模型的性能并进行性能比较

f1值具有综合考虑$Precion$和$Recall$的能力，能够更好地判断分类模型的性能，f1值的计算公式如下：

$$f1 = \frac{2 \times (precision \times recall)}{precision + recall}$$

对比三个分类模型，boosting方法得到的分类模型效果较好