

实验六：消费金融风控实验报告

note：数据处理和WOE/IV值计算部分代码见实验二部分.ipynb文件最后一部分；在code文件夹也有此文件

特征与label关系分析

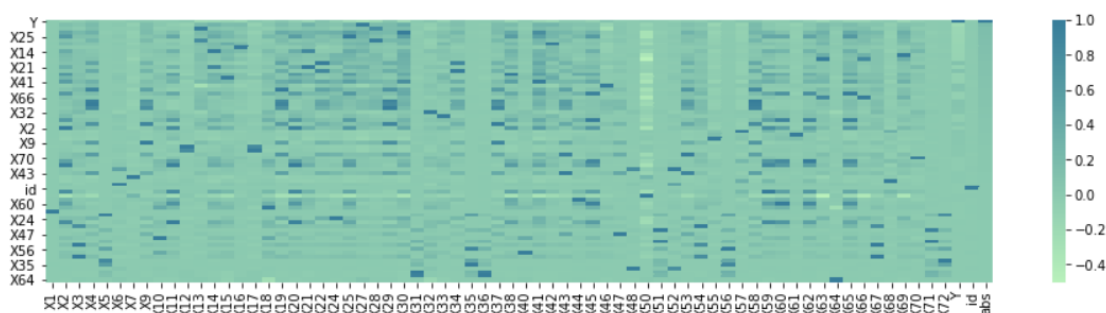
相关系数计算

通过相关系数观察特征与label的关系，取出相关系数绝对值前20的数据查看，可以发现相关性最高的是“最近14天机构联系计数”；总体来看，单个特征与label的相关性并不特别高。

```
In [55]: co=data.corr()  
co["abs"]=abs(co["Y"])  
new_co = co.sort_values(by=['abs'],ascending=False)  
new_co["Y"][0:20]
```

```
Out[55]: Y      1.000000  
最近14天机构联系计数    -0.244145  
手机账户余额          -0.164431  
上个月语音套餐总容量    -0.151542  
最近七天风险机构计数    -0.146208  
手机记录天数           0.145208  
联系人中黑名单人数计数    0.141244  
近30天历史搜索数量       -0.140307  
第三方风险分数          -0.128984  
平均充值金额           -0.119884  
网龄                   -0.114756  
最近3个月活跃天数       -0.114321  
近6个月平均期限         -0.113517  
过去12个月被叫次数均值   -0.111530  
近三个月平台借贷详情     -0.111350  
当前话费余额           -0.102974  
过去12个月话费均值       -0.096947  
过去12个月语音套餐容量均值  0.093385  
最近90天搜索数量         0.079281  
过去12个月平均主叫比例   -0.078752  
Name: Y, dtype: float64
```

查看heat图，同样发现对于Y的相关度较低，但是特征之间存在部分较高的相关度。



WOE和IV值计算

代码：

```
def compute_WOE_IV(df,col,target):  
    total = df.groupby([col])[target].count()  
    total = pd.DataFrame({'total': total})  
  
    bad    = df.groupby([col])[target].sum()
```

```

bad    = pd.DataFrame({'bad': bad})

regroup = total.merge(bad,left_index=True,right_index=True,how='left')
regroup.reset_index(level=0,inplace=True)

N = sum(regroup['total'])
B = sum(regroup['bad'])

regroup['good'] = regroup['total'] - regroup['bad']
G = N - B

regroup['bad_pcnt'] = regroup['bad'].map(lambda x: x*1.0/B)
regroup['good_pcnt'] = regroup['good'].map(lambda x: x * 1.0 / G)

regroup["WOE"] = regroup.apply(lambda
x:np.log(x.good_pcnt*1.0/x.bad_pcnt),axis=1)

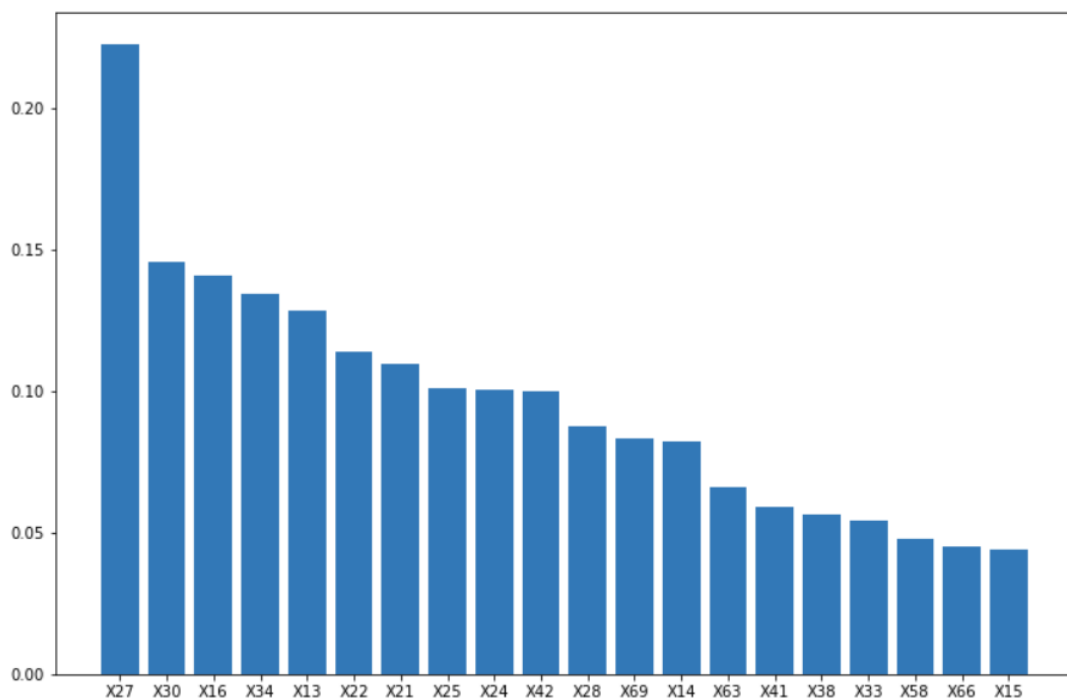
WOE_dict = regroup[[col,"WOE"]].set_index(col).to_dict(orient="index")
IV = regroup.apply(lambda x:(x.good_pcnt-
x.bad_pcnt)*np.log(x.good_pcnt*1.0/x.bad_pcnt),axis = 1)

IV = sum(IV)

return {"WOE":WOE_dict,"IV":IV}

```

对每一列进行等深分箱后，计算IV值，取前20个画图。注意不能使用等宽分箱，会出现WOE值计算得inf的情况，说明分箱效果差。



特征工程过程

处理缺失值

对nan值个数进行排序，删除nan值过多的列和行，剩余nan值用列平均值填充：

```

data = pd.read_csv("data/train_new.csv")

```

```
# 删除非nan值小于60个的行
data.dropna(thresh=60, inplace=True)

# 删除nan值过高的几个列
del data["x8"]
del data["x23"]
del data["x26"]
del data["x39"]
del data["x49"]

# 剩余部分nan用平均值填充
data=data.fillna(data.mean())
```

归一化

对所有feature都进行归一化的代码：

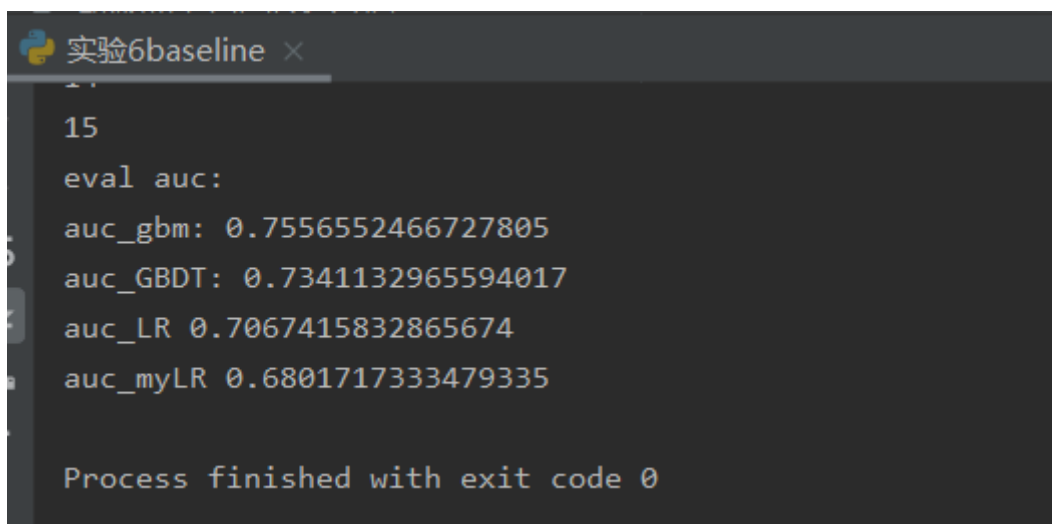
```
numeric_feats = data.dtypes[data.dtypes != "object"].index
data[numeric_feats] = data[numeric_feats].apply(lambda x: (x - x.min()) /
(x.max() - x.min()))
```

分割测试集和训练集

由于数据集足够大，测试之后决定了训练集与测试集的大小比为0.95：0.05

```
n=data.shape[0]
train=data[0:int(n*0.95)]
test=data[int(n*0.95):n]
train.to_csv("./train.csv",index=False)
test.to_csv("./test.csv",index=False)
```

不同模型结果



```
实验6baseline x
15
eval auc:
auc_gbm: 0.7556552466727805
auc_GBDT: 0.7341132965594017
auc_LR 0.7067415832865674
auc_myLR 0.6801717333479335

Process finished with exit code 0
```

查阅资料发现，auc值分布在0到1之间，

auc<0.5：效果比随机生成器还差，建议反预测

auc=0.5：效果和随机生成器一样

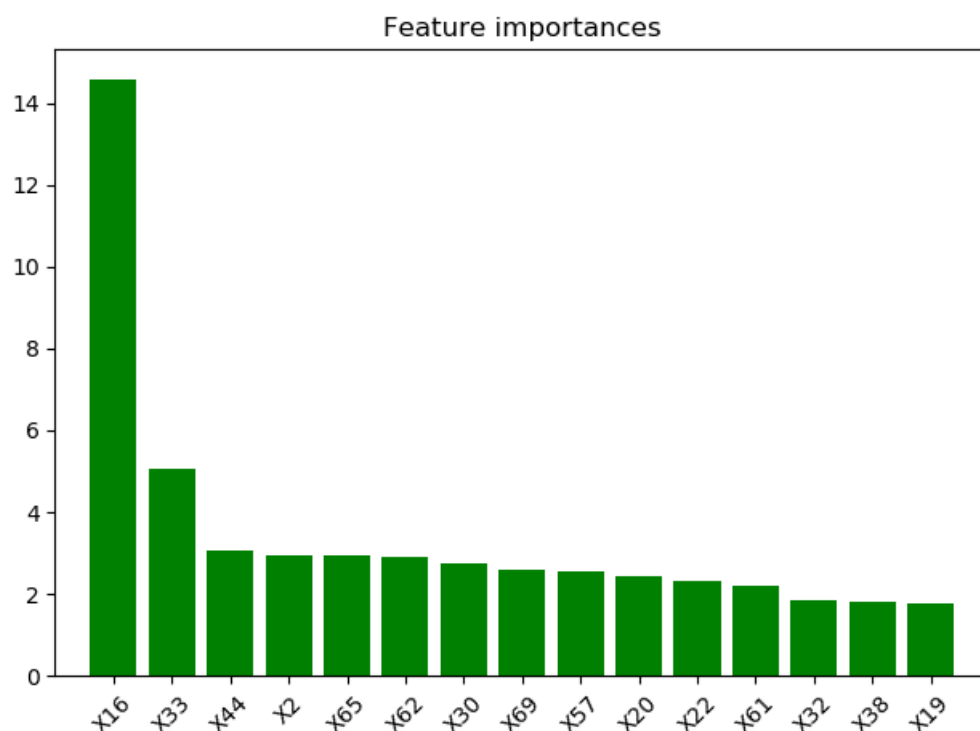
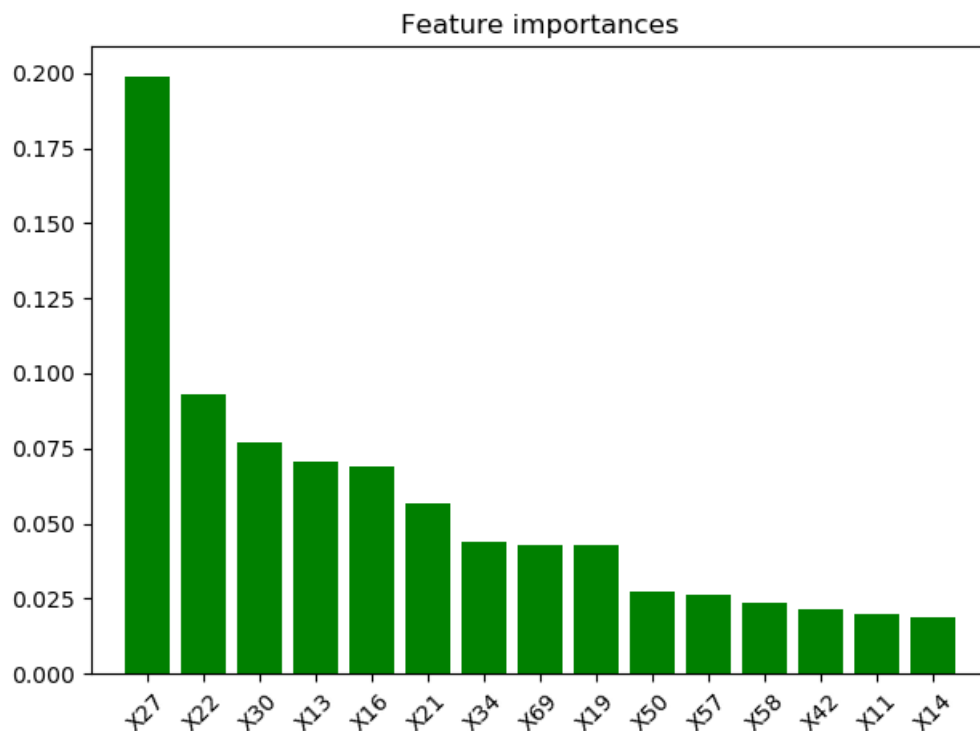
auc=[0.5,0.7]：效果较差

auc=[0.7,0.85]：效果一般

可见在这个问题中，LR算法效果一般，尤其是自己手写的LR算法，效果相对较差；而gbm和GBDT分类器能够达到更好一点的分类效果。

从训练完成时间看，gbm算法非常快，GBDT次之，两个LR算法都相当慢。

输出了GBDT和LR模型前几位的feature重要性柱状图：



可以看到两个分类器提取的重要特征还是非常不同的...orz，第一个和WOE值/IV值计算得出的结果较为吻合。

对归一化和正则化的讨论

实验结果

未归一化，无正则化

```
实验6baseline x
↑ eval auc: (0.7547895080231959, 0.7340984683611875, 0.7138317432942325)
↓
Process finished with exit code 0
```

未归一化，但是有正则化

可以看到LR分类器给了一个warning，迭代次数不够，没有收敛

```
E:\pycharmpro\py3\lib\site-packages\sklearn\linear_model\logistic.py:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
eval auc: (0.7547895080231959, 0.7340984683611875, 0.49670699936580937)
```

归一化，无正则化

```
实验6baseline x
eval auc: (0.7556552466727805, 0.7341132965594017, 0.710169178335318)

Process finished with exit code 0
```

归一化且正则化

```
实验6baseline x
eval auc: (0.7556552466727805, 0.7341132965594017, 0.7066366452684361)

Process finished with exit code 0
```

可以看出，数据是否归一化对GBDT算法几乎没有影响，但是对LR算法影响大；没有归一化也没有正则项的LR算法几乎没有提取特征进行分类的作用。

概率模型（树形模型）不需要归一化，因为它们不关心变量的值，而是关心变量的分布和变量之间的条件概率。而LR，SVM等分类器是需要归一化的。LR模型因为使用了梯度下降法，每次下降的梯度是固定的值（无量纲），当数据过大/过小的时候，梯度下降就很难到达局部最优，或者很容易超出局部最优，出现上面warning的不收敛的问题。特别是有正则化的时候，LR的W矩阵的大小将成为损失函数的一部分，W又与特征值的量纲有关。这时候数据的范围大小将影响LR的结果。

手写auc

```
def myAUC(self, prob, labels):
    f = list(zip(prob, labels))
    rank = [x2 for x1, x2 in sorted(f, key=lambda x: x[0])]
    rankList = [i+1 for i in range(len(rank)) if rank[i]==1]
    cnt_pos = 0
    cnt_neg = 0
    for i in range(len(labels)):
        if(labels[i]==1):
            cnt_pos+=1
        else:
            cnt_neg+=1
    return (sum(rankList) - (cnt_pos*(cnt_pos+1))/2)/(cnt_pos*cnt_neg)
```

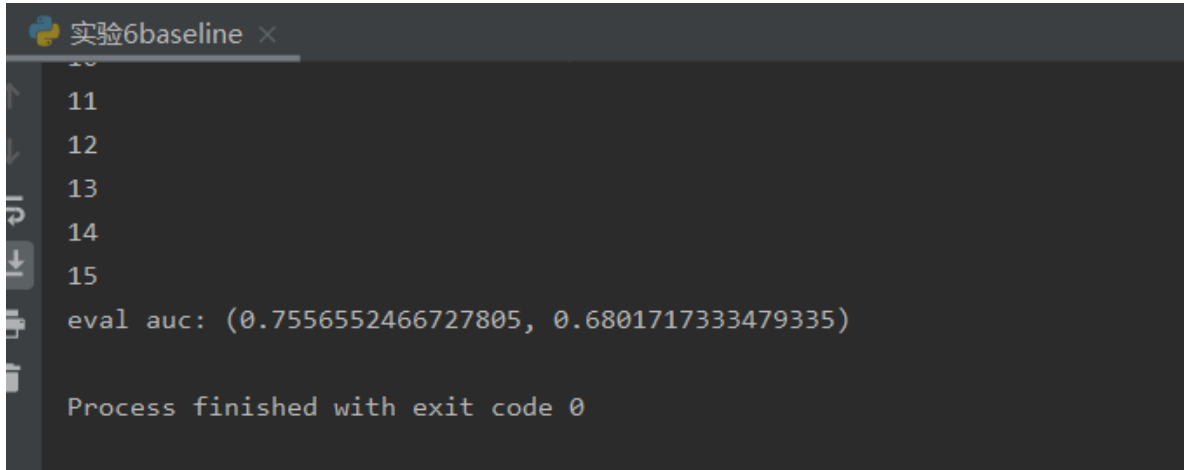
暂时没有看出和sklearn的auc的区别。

myLR

简单实现了LR:

- 1.没有设置early stop, 而是跑完n_epoch=15 (可变), 输出当前epoch查看训练进度
- 2.尝试了linear search, 但是速度过慢, 对结果的改善不明显, 最终没有使用
- 3.使用了l1正则项

最终跑出来的结果稍微差于sklearn的LR: (前面一个数据是gbm的auc, sklearn的LR的auc值在71左右)

A terminal window titled '实验6baseline' with a close button. The terminal shows a list of numbers from 11 to 15, followed by the evaluation result 'eval auc: (0.7556552466727805, 0.6801717333479335)' and the message 'Process finished with exit code 0'.

```
11
12
13
14
15
eval auc: (0.7556552466727805, 0.6801717333479335)

Process finished with exit code 0
```