天津大学

数据挖掘实验报告

天池学习赛——挖掘幸福感

学　院　　智能与计算学部

专　业　　人工智能

年　级　　2019级

学　号　　3019244132

姓　名　　游奕桁

日　期　　2021年10月24 日

一、任务描述

在社会科学领域，幸福感的研究占有重要的位置。这个涉及了哲学、心理学、社会学、经济学等多方学科的话题复杂而有趣；同时与大家生活息息相关，每个人对幸福感都有自己的衡量标准。如果能发现影响幸福感的共性，生活中是不是将多一些乐趣；如果能找到影响幸福感的政策因素，便能优化资源配置来提升国民的幸福感。目前社会科学研究注重变量的可解释性和未来政策的落地，主要采用了线性回归和逻辑回归的方法，在收入、健康、职业、社交关系、休闲方式等经济人口因素；以及政府公共服务、宏观经济环境、税负等宏观因素上有了一系列的推测和发现。

赛题尝试了**幸福感预测**这一经典课题，希望在现有社会科学研究外有其他维度的算法尝试，结合多学科各自优势，挖掘潜在的影响因素，发现更多可解释、可理解的相关关系。

赛题使用公开数据的问卷调查结果，选取其中多组变量，包括**个体变量**（性别、年龄、地域、职业、健康、婚姻与政治面貌等等）、**家庭变量**（父母、配偶、子女、家庭资本等等）、**社会态度**（公平、信用、公共服务等等），来预测其对**幸福感**的评价。

二、数据

- 赛题提供的数据分为**完整版**和**精简版**两类。本次实验选取精简版进行数据挖掘。
- **数据来源**：赛题使用的数据来自中国人民大学中国调查与数据中心主持之《中国综合社会调查（CGSS）》项目。赛题感谢此机构及其人员提供数据协助。中国综合社会调查为**多阶分层抽样**的**截面面访**调查。
- 数据预处理：
  - 使用panda库进行数据导入

```
train_data = pd.read_csv("happiness_train_abbr.csv", index_col='id')
```

  - 查看前五条数据，每条数据有41个属性

```
    happiness  survey_type  province  ...  status_3_before  view inc_ability
id                                    ...
1           4            1        12  ...                2     4           3
2           4            2        18  ...                1     4           2
3           4            2        29  ...                1     4           2
4           5            2        10  ...                1     3           2
5           4            1         7  ...                2     3          -8

[5 rows x 41 columns]
```

  - 查看数据的统计性描述，发现存在异常数据，多个属性的最小值存在负数

```
data description:
         happiness  survey_type  ...         view  inc_ability
count  8000.000000  8000.000000  ...  8000.00000  8000.000000
mean      3.850125     1.405500  ...     3.30350     1.094875
std       0.938228     0.491019  ...     1.98132     3.410180
min      -8.000000     1.000000  ...    -8.00000    -8.000000
25%       4.000000     1.000000  ...     3.00000     2.000000
50%       4.000000     1.000000  ...     4.00000     2.000000
75%       4.000000     2.000000  ...     4.00000     3.000000
max       5.000000     2.000000  ...     5.00000     4.000000

[8 rows x 40 columns]
```

- 检查存在异常的数据

```
print(train_data.info())    # work_status等存在异常
print(pd.value_counts(train_data.work_status))
print(train_data.work_status.unique())
```

```
 27  work_status      2951 non-null   float64
 28  work_yr          2951 non-null   float64
 29  work_type        2951 non-null   float64
 30  work_manage      2951 non-null   float64
```

```
None
 3.0    1767
 2.0     471
 5.0     285
 1.0     136
 4.0      88
 8.0      83
 6.0      45
 9.0      32
-8.0      25
 7.0      19
Name: work_status, dtype: int64
[ 3. nan  2.  5.  6.  1.  4.  8. -8.  9.  7.]
```

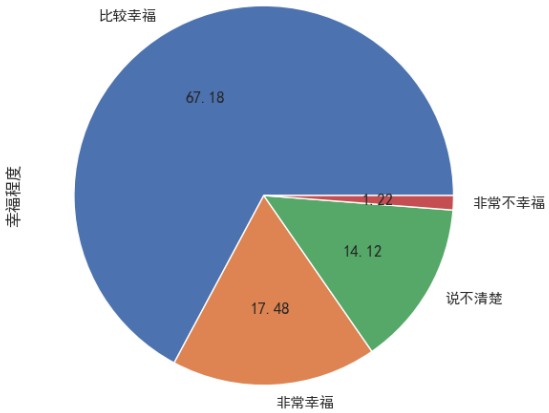
- 进行简单的预处理，去除存在大量nan值的属性，发现所有异常负值都为-8，去除存在负值的数据，数据从8000条减少为6620条

```
train_data.drop(['work_status', 'work_yr', 'work_type', 'work_manage'], axis=1,
inplace=True)
train_data = train_data.replace(-8, np.nan).dropna(how='any')
```

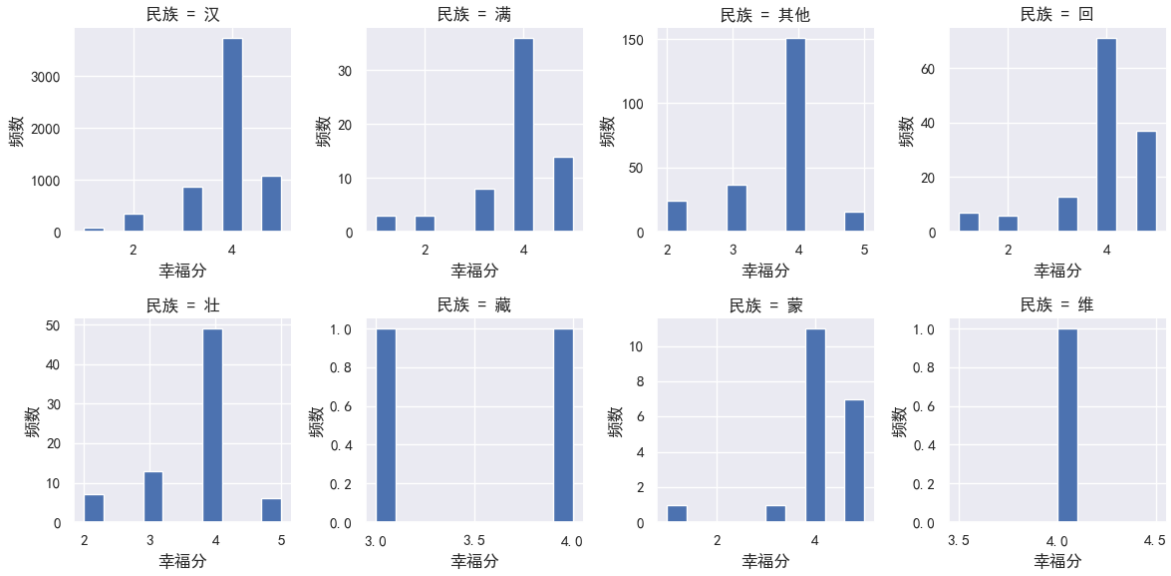
```
data description after preprocessing:
count    6620.000000
mean        3.878701
std         0.804842
min         1.000000
25%         4.000000
50%         4.000000
75%         4.000000
max         5.000000
Name: happiness, dtype: float64
```

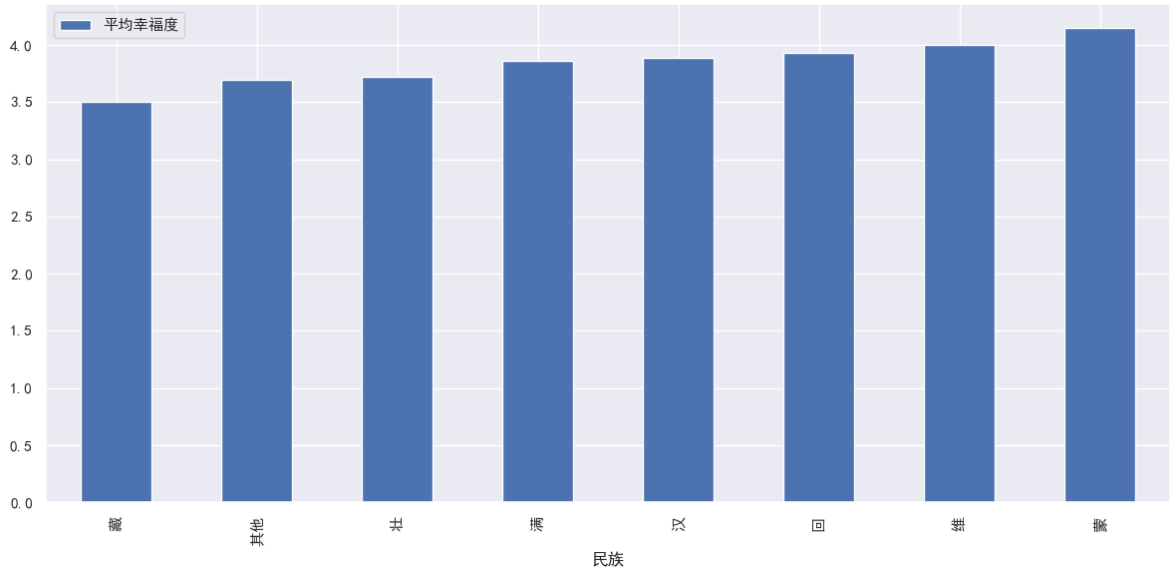
- 数据可视化
  - 幸福程度的5个分值作为分类标签，绘制饼图，获得幸福程度的分布

```
happiness_dict = {1: '非常不幸福', 2: '比较幸福', 3: '说不清楚', 4: '比较幸福', 5: '非常幸
福'}
happiness = vis_data.happiness.map(lambda x: happiness_dict[x])
h_cnt = happiness.value_counts()
h_cnt.name = '幸福程度'
h_cnt.plot.pie(figsize=(6, 6), autopct='%.2f')
```

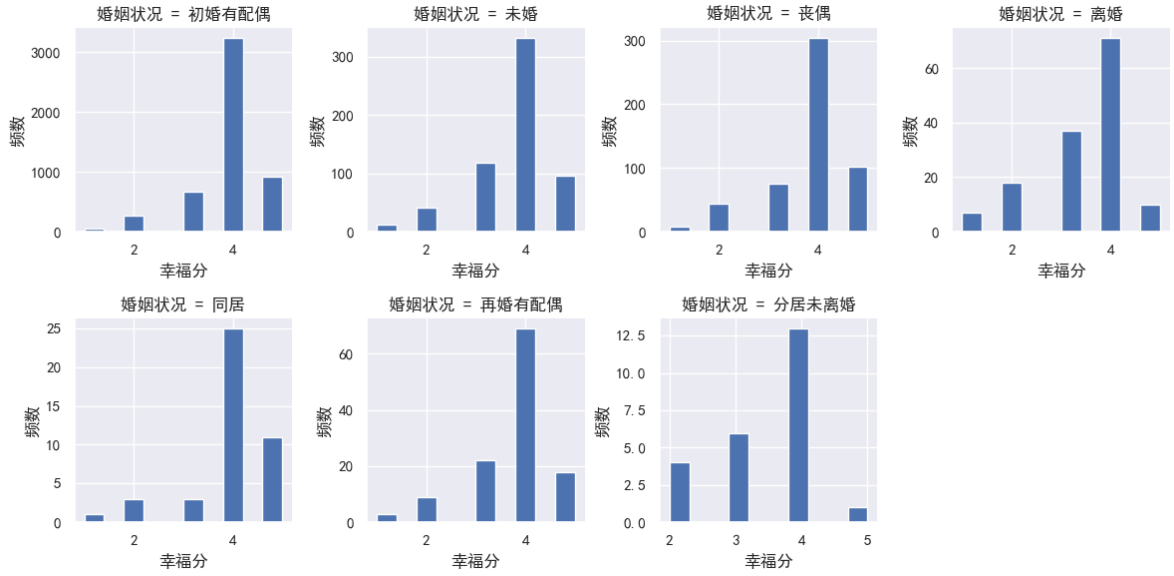


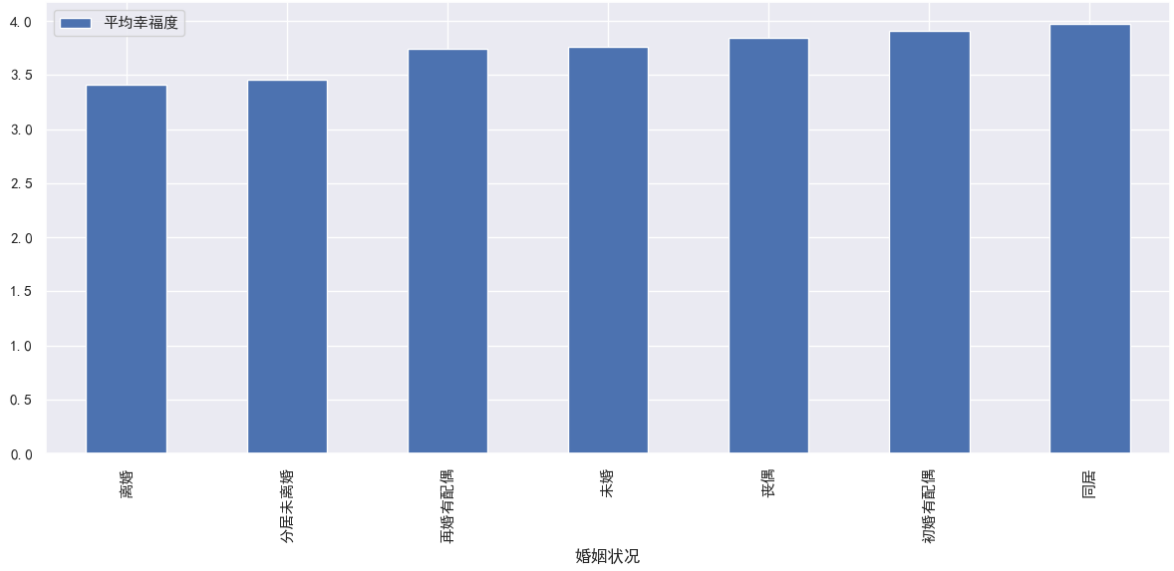
- 可视化幸福程度在各属性上的分布，离散属性使用直方图表示，连续属性使用盒图表示，我对所有属性都进行了可视化，报告中仅展示部分属性上的分布图
  - 民族属性上的幸福程度直方图



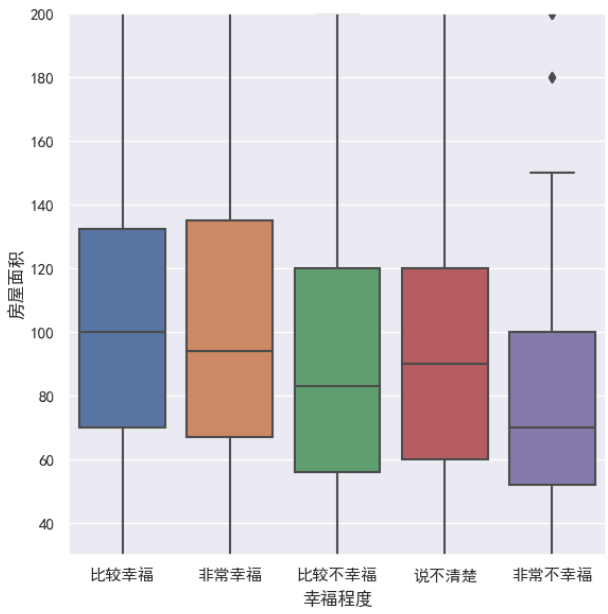


- 婚姻状况属性上的幸福程度直方图

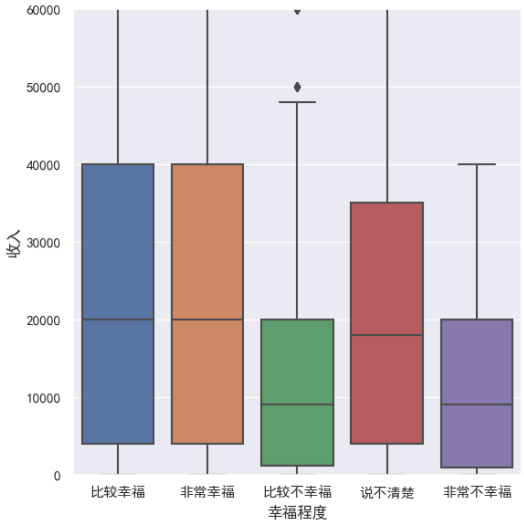




- 房屋面积属性上的幸福程度盒图



- 收入属性上的幸福程度盒图



三、算法

- 本次实验我采用了决策树、SVM、Logistic Regression、KNN、XGBoost五种分类算法进行训练和预测，对比四种模型的表现，其中决策树、SVM、Logistic Regression、XGBoost算法调用sklearn库中的算法实现，KNN较为基础，自己实现了算法。
- sklearn库中的决策树模型采用优化的CART决策树算法，选择gini指数作为分裂节点时的评价标准，ID3算法采用信息增益作为评价指标，经过测试，在这个数据集上，采用gini指数和信息增益作为评价指标得到的分类结果近似。

> 决策树算法的基本思想：
>
> 1）树以代表训练样本的单个结点开始。
>
> 2）如果样本都在同一个类．则该结点成为树叶，并用该类标记。
>
> 3）否则，算法选择最有分类能力的属性作为决策树的当前结点。
>
> 4）根据当前决策结点属性取值的不同，将训练样本分为若干子集，每个取值形成一个分枝，有几个取值形成几个分枝。匀针对上一步得到的一个子集，重复进行先前步骤，递归形成每个划分样本上的决策树。
>
> 5）递归划分步骤仅当下列条件之一成立时停止：
>
> ①给定结点的所有样本属于同一类。
>
> ②没有剩余属性可以用来进一步划分样本。在这种情况下，使用多数表决，将给定的结点转换成树叶。
>
> ③如果某一分枝没有满足该分支中已有分类的样本，则以样本的多数类创建一个树叶。

- SVM的基本思想是**求解能够正确划分训练数据集并且几何间隔最大的分离超平面**，其学习策略便是间隔最大化，最终化为一个凸二次规划问题的求解。本次实验采用高斯核的非线性支持向量机。
- 逻辑回归算法是**通过回归的思想来解决分类问题的算法**。其基本思想是将**样本所属正例的概率作为模型的输出，**根据此概率值对样本的类别进行预测。本次实验采用随机平均梯度下降作为损失函数的优化方法。
- KNN算法的核心思想是，如果一个样本在特征空间中的**K个最相邻的样本**中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

- XGBoost算法的基本思想与GBDT类似，不断地地进行特征分裂来生长一棵树，每一轮学习一棵树，其实就是去拟合**上一轮模型**的预测值与实际值之间的**残差**。当我们训练完成得到k棵树时，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，**每个叶子节点就对应一个分数**，最后只需**将每棵树对应的分数加起来**就是该样本的预测值。

- 分割训练集和测试集，除决策树和XGBoost外的三个模型选择采用降维后的数据进行训练，使用PCA算法进行降维，数据归一化处理。

```
train_X, test_X, train_y, test_y = train_test_split(train_data.iloc[:, 1:],
train_data['happiness'], test_size=0.2)
ss = StandardScaler()
pca = PCA(n_components=0.98, copy=True)
pca_trainX = pca.fit_transform(train_X)
pca_testX = pca.fit_transform(test_X)
pca_trainX = ss.fit_transform(pca_trainX)
pca_testX = ss.fit_transform(pca_testX)
```

  在保留98%的信息量的条件下，PCA算法可以将数据降至二维，极大降低了计算的复杂度

```
Before PCA, the shape of training data: (5296, 35)
After PCA, the shape of training data: (5296, 2)
```

- KNN实现代码：

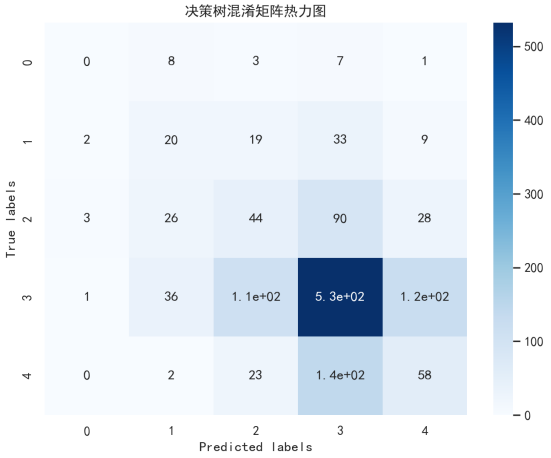
```
def KNN(train_x, train_y, test_x, k=3):
    train_y = train_y.values
    num_train = train_x.shape[0]
    num_test = test_x.shape[0]
    dist = np.zeros((num_test, num_train))
    for i in range(num_test):
        dist[i] = np.reshape(np.sqrt(np.sum(np.square(test_x[i] - train_x),
axis=1)), [1, num_train])
    predictedLabels = np.zeros((num_test, 1))
    for i in range(num_test):
        close_k = train_y[np.argsort(dist[i])[:k]].astype(np.int)
        predictedLabels[i] = np.argmax(np.bincount(close_k))
    return predictedLabels
```

四、实验结果

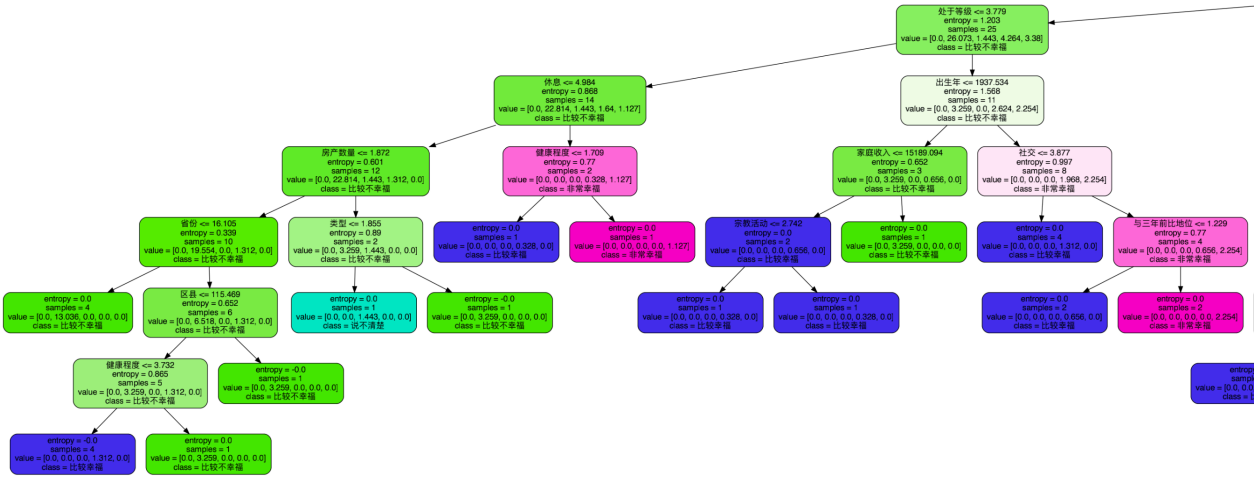
基本未进行过参数调整的模型训练结果：

- 决策树模型训练结果：

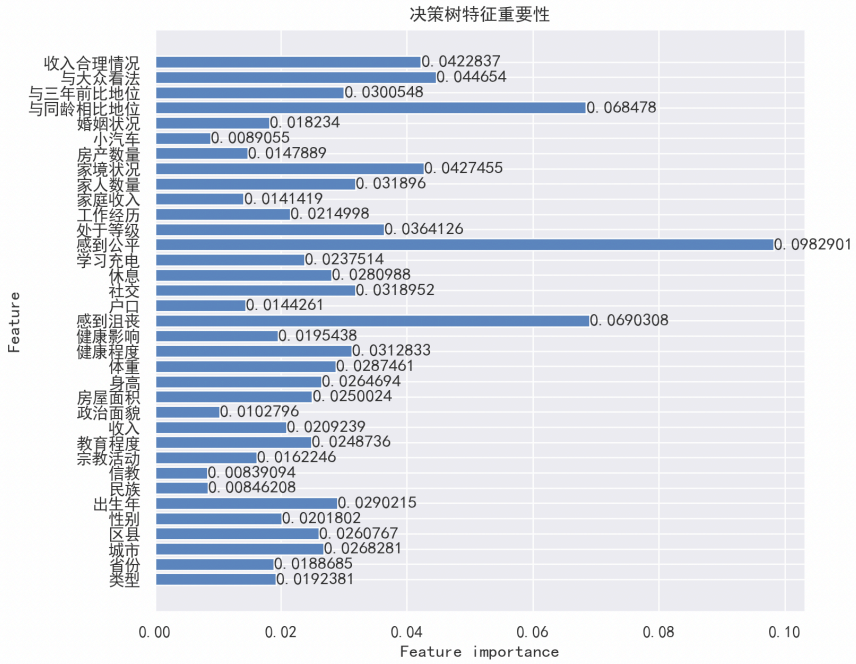
```
采用决策树进行分类
用时: 26965387.00 ns
accuracy = 0.49
              precision    recall  f1-score   support

         1.0       0.00      0.00      0.00        19
         2.0       0.22      0.24      0.23        83
         3.0       0.22      0.23      0.22       191
         4.0       0.66      0.66      0.66       806
         5.0       0.26      0.26      0.26       225

    accuracy                           0.49      1324
   macro avg       0.27      0.28      0.27      1324
weighted avg       0.49      0.49      0.49      1324

The confusion matrix result of Decision Tree:
 [[  0   8   3   7   1]
 [  2  20  19  33   9]
 [  3  26  44  90  28]
 [  1  36 112 532 125]
 [  0   2  23 142  58]]
```



- 部分决策树可视化：



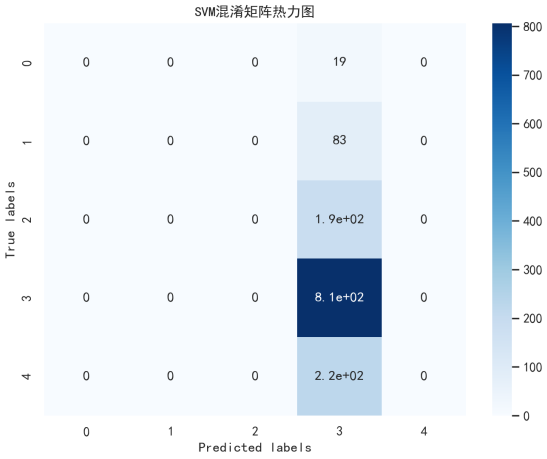
- 特征重要性可视化：



- 将决策树可视化后发现，最终生成的决策树非常庞大，但准确率仍然不高。将特征重要性可视化后发现，感到公平对人们的幸福感极为重要。

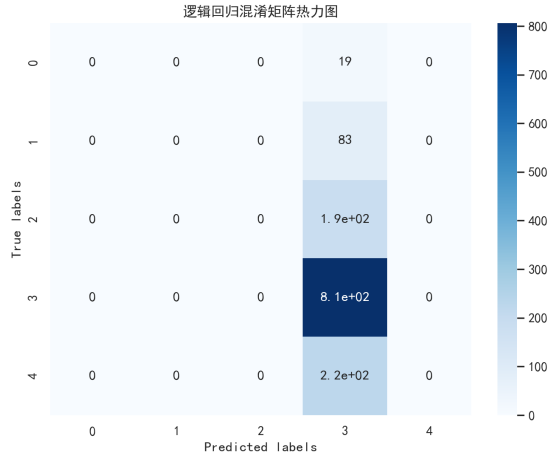
- SVM模型训练结果：

```
采用RBF kernel SVM进行分类
用时: 1195656292.00 ns
accuracy = 0.61
              precision    recall  f1-score   support

         1.0       1.00      0.00      0.00        19
         2.0       1.00      0.00      0.00        83
         3.0       1.00      0.00      0.00       191
         4.0       0.61      1.00      0.76       806
         5.0       1.00      0.00      0.00       225

    accuracy                           0.61      1324
   macro avg       0.92      0.20      0.15      1324
weighted avg       0.76      0.61      0.46      1324

The confusion matrix result of SVM:
 [[  0   0   0  19   0]
 [  0   0   0  83   0]
 [  0   0   0 191   0]
 [  0   0   0 806   0]
 [  0   0   0 225   0]]
```



- 逻辑回归模型训练结果：

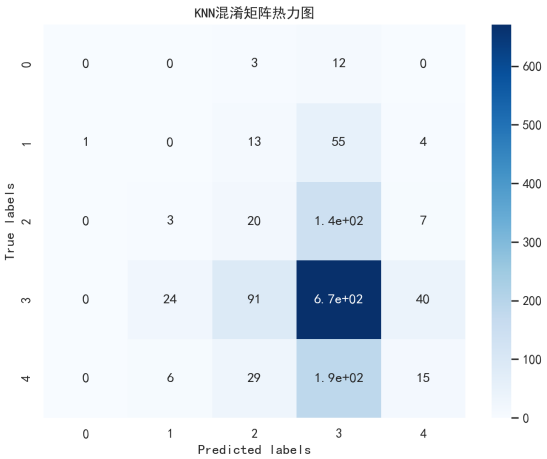
```
采用logisitic回归模型进行分类
用时: 908180358.00 ns
accuracy = 0.61
              precision    recall  f1-score   support

         1.0       1.00      0.00      0.00        19
         2.0       1.00      0.00      0.00        83
         3.0       1.00      0.00      0.00       191
         4.0       0.61      1.00      0.76       806
         5.0       1.00      0.00      0.00       225

    accuracy                           0.61      1324
   macro avg       0.92      0.20      0.15      1324
weighted avg       0.76      0.61      0.46      1324

The confusion matrix result of Logistic Regression:
 [[  0   0   0  19   0]
 [  0   0   0  83   0]
 [  0   0   0 191   0]
 [  0   0   0 806   0]
 [  0   0   0 225   0]]
```



  - 通过对比发现svm和逻辑回归算法无法很好地拟合这样的数据集，最终的结果将所有的样本都分到了同一类，建立的模型没有任何意义，还用了非常长的时间来收敛，调整了多种参数对结果都没有产生太大的影响，发现本数据集可能无法用函数来进行拟合，只能通过决策树或者KNN这样不使用决策函数的算法来进行分类。

- KNN模型训练结果：

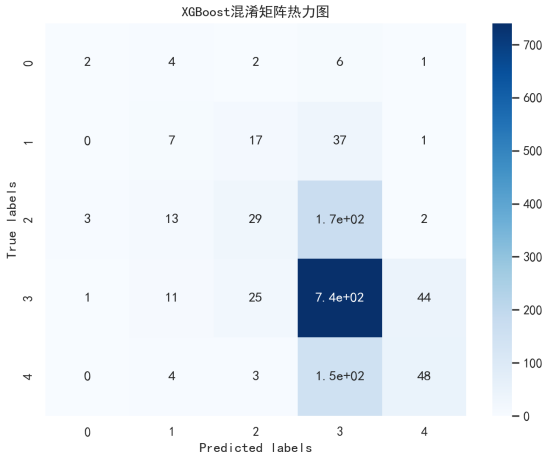
```
采用KNN进行分类
用时: 589864779.00 ns
accuracy = 0.53
              precision    recall  f1-score   support

         1.0       0.00      0.00      0.00        15
         2.0       0.00      0.00      0.00        73
         3.0       0.13      0.11      0.12       174
         4.0       0.63      0.81      0.71       826
         5.0       0.23      0.06      0.10       236

    accuracy                           0.53      1324
   macro avg       0.20      0.20      0.19      1324
weighted avg       0.45      0.53      0.48      1324

The confusion matrix result of KNN:
 [[  0   0   3  12   0]
 [  1   0  13  55   4]
 [  0   3  20 144   7]
 [  0  24  91 671  40]
 [  0   6  29 186  15]]
```



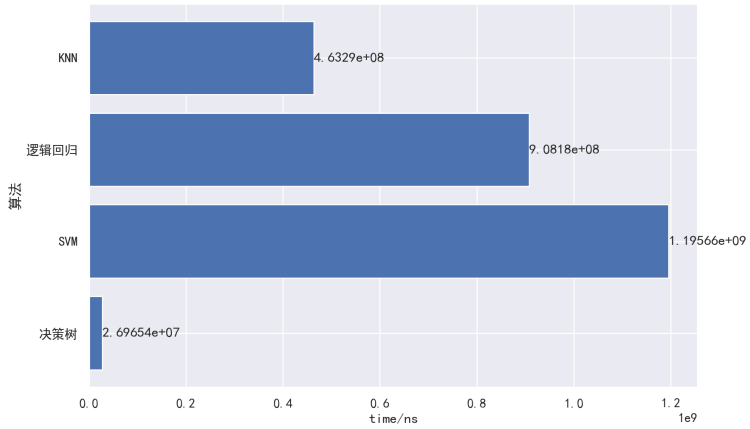
  - KNN算法简单，准确率一般，k值的选取对算法最终的准确性有一定的影响，经过测试发现k=5时准确率最高。

- XGBoost模型训练结果：

```
采用XGBoost进行分类
用时：1969471238.00 ns
accuracy = 0.62
              precision    recall  f1-score   support

           0       0.33      0.13      0.19        15
           1       0.18      0.11      0.14        62
           2       0.38      0.13      0.20       219
           3       0.67      0.90      0.77       821
           4       0.50      0.23      0.32       207

    accuracy                           0.62      1324
   macro avg       0.41      0.30      0.32      1324
weighted avg       0.57      0.62      0.57      1324

The confusion matrix result of Decision Tree:
 [[  2   4   2   6   1]
 [  0   7  17  37   1]
 [  3  13  29 172   2]
 [  1  11  25 740  44]
 [  0   4   3 152  48]]
```



- 各模型用时对比



- 发现XGBoost的拟合表现最好，准确率最高，对XGBoost进行参数调整
  - 使用GridSearchCV寻找参数的最佳取值

```
def Tuning(cv_params, other_params, x_train_array, y_train_):
    model2 = XGBClassifier(**other_params)
    optimized_GBM = GridSearchCV(estimator=model2, param_grid=cv_params,
scoring='accuracy', cv=5, n_jobs=-1)
    optimized_GBM.fit(x_train_array, y_train_)
    evalute_result = optimized_GBM.cv_results_['mean_test_score']
    # print('每轮迭代运行结果:{0}'.format(evalute_result))
    print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
    print('最佳模型得分:{0}'.format(optimized_GBM.best_score_))
    return optimized_GBM
```

  - 调参过程(每次选择出最佳取值后加入other_params，再计算下一个参数的最佳取值)

```
other_params = {
    'use_label_encoder': False,
    'booster': 'gbtree',
    'objective': 'multi:softmax',
```

```
    'num_class': 5,
    'learning_rate': 0.05,
    'max_depth': 5,
    'min_child_weight': 5,
    'colsample_bytree': 0.8,
    'subsample': 0.6,
    'reg_alpha': 0.5
}
cv_params = {
    # 'learning_rate': [0.01, 0.02, 0.05, 0.1, 0.15],
    # 'max_depth': [2, 3, 4, 5],
    # 'min_child_weight': [0, 2, 5, 10, 20],
    # 'subsample': [0.6, 0.7, 0.8, 0.85, 0.95],
    # 'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9],
    'reg_alpha': [0, 0.25, 0.5, 0.75, 1]
}
```

```
参数的最佳取值：{'learning_rate': 0.05}
最佳模型得分:0.6359484739964724
```

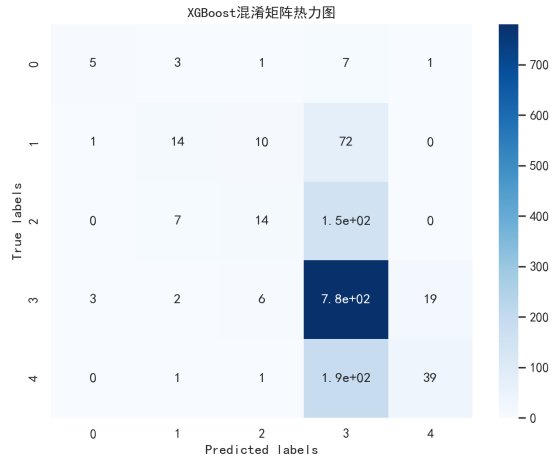
```
参数的最佳取值：{'max_depth': 5, 'min_child_weight': 5}
最佳模型得分:0.636327792328113
```

```
参数的最佳取值：{'colsample_bytree': 0.8, 'subsample': 0.6}
最佳模型得分:0.6391620788568781
```

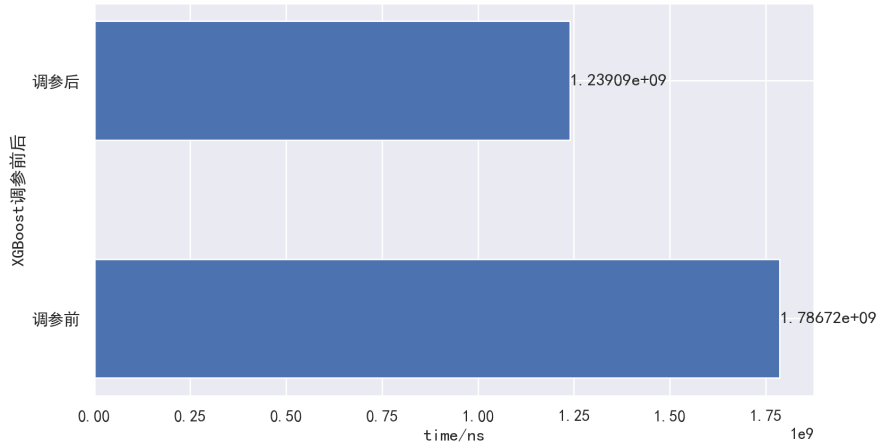
```
参数的最佳取值：{'reg_alpha': 0.5}
最佳模型得分:0.6365218165945089
```

- 调参后的结果
  - 模型预测准确度提高2%

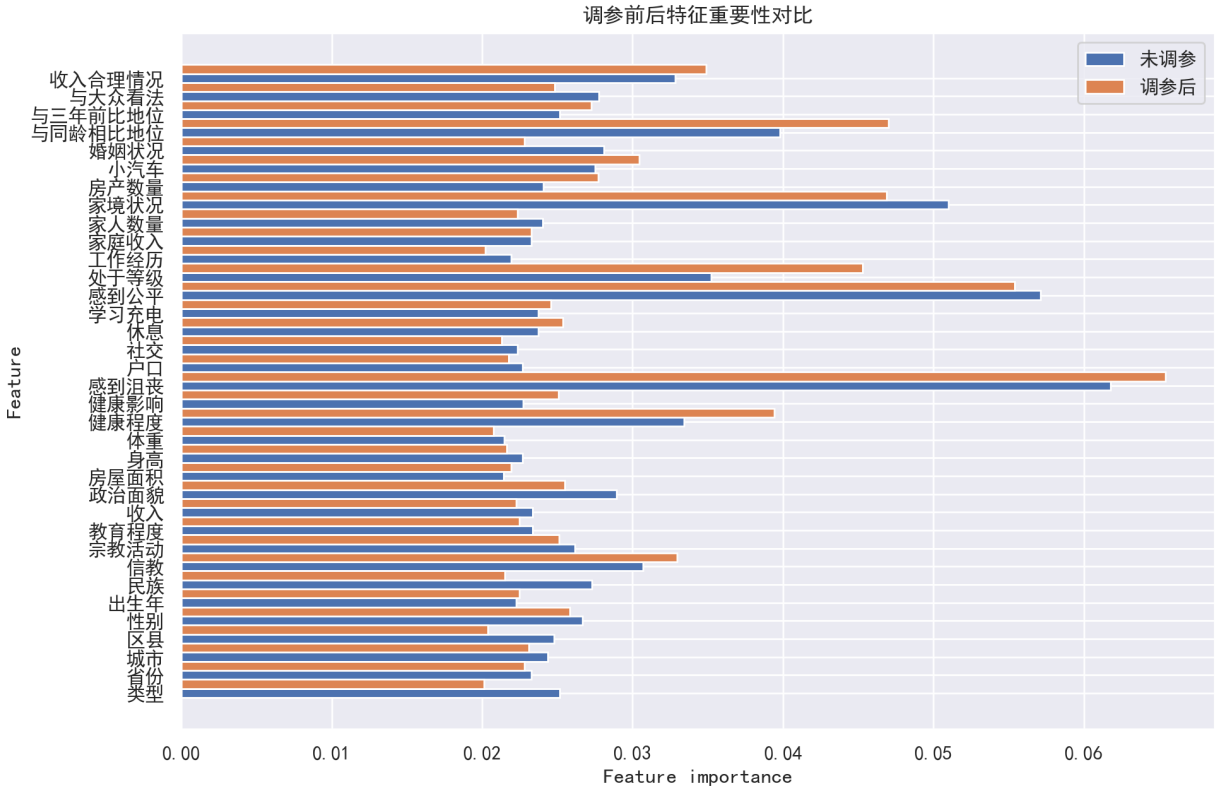
```
采用调参后的XGBoost进行分类
用时: 1385943022.00 ns
accuracy = 0.64
              precision    recall  f1-score   support

           0       0.56      0.29      0.38        17
           1       0.52      0.14      0.23        97
           2       0.44      0.08      0.14       171
           3       0.65      0.96      0.78       810
           4       0.66      0.17      0.27       229

    accuracy                           0.64      1324
   macro avg       0.56      0.33      0.36      1324
weighted avg       0.61      0.64      0.56      1324

The confusion matrix result of Decision Tree:
[[  5   3   1   7   1]
 [  1  14  10  72   0]
 [  0   7  14 150   0]
 [  3   2   6 780  19]
 [  0   1   1 188  39]]
```



  - 运行时间缩短



- 特征重要性变化，但是整体情况相似



五、实验体会

- 本次实验的目的不在于追求分类的高准确率，通过本次实验对比了几个常用的分类算法的异同和在同一任务上的表现情况，对课程所涉及到的几种分类算法有了更深刻的理解。
- 在本次实验中，我花费了较多的精力在进行数据的预处理和可视化上，掌握了使用matplotlib进行绘图的更多技巧，让实验的结果更具有可解释性，也可以让数据的情况更加明了。
- 了解了课堂上没有过多介绍的XGBoost模型，学会了如何调整参数让模型的拟合效果更好，运行时间更短，将模型简单优化。