

数据分析及实践-实验四

PB20151804 袁雨

数据分析及实践-实验四

1 分类算法实践

1.1 算法主要流程

1.2 算法关键技术

1.3 算法实现

1.4 实验记录

2 预测算法实践

2.1 算法主要流程

2.2 实验记录

2.3 算法实现

附录

1 分类算法实践

1.1 算法主要流程

使用实验三中与REPEAT相关度较高的三个离散特征：ST001D01T、ST127Q01TA、ST127Q02TA，用热图观察其独立性，发现独立性较好：



故选择朴素贝叶斯算法。

朴素贝叶斯方法是一组基于应用贝叶斯定理的监督学习算法，在给定类变量值的情况下，每对特征之间条件独立的“朴素”假设。

Step 1 导入数据集并选择特征

Step 2 为所有可能的 y 计算 $P(Y = y)$

Step 3 分类计算 $P(X = x|Y = y)$

Step 4 计算所有 y 的 $P(X = x_1|Y = y)P(X = x_2|Y = y) \dots P(X = x_n|Y = y)P(Y = y)$ ，求最大值

Step 5 在数据集上划分训练集和测试集(4:1比例、交叉验证)

1.2 算法关键技术

贝叶斯定理陈述以下关系，给定类变量 y 和依赖特征向量 x_1 到 x_n ：

$$P(y | x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$

使用朴素的条件独立假设

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y)$$

对全部 i ，这个关系简化为

$$P(y | x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1, \dots, x_n)}$$

因为 $P(x_1, \dots, x_n)$ 给定输入是常数，我们可以使用以下分类规则：

$$\begin{aligned} P(y | x_1, \dots, x_n) &\propto P(y) \prod_{i=1}^n P(x_i | y) \\ &\Downarrow \\ \hat{y} &= \arg \max_y P(y) \prod_{i=1}^n P(x_i | y) \end{aligned}$$

我们可以使用最大后验 (MAP) 估计来估计 $P(y)$ 和 $P(x_i | y)$ ；前者是类 y 在训练集中的相对频率。

1.3 算法实现

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set_style("darkgrid")

# Importing the dataset
dataset = pd.read_csv('D:\\jupyterlab\\lab3\\pica2015.csv')
data = dataset[['ST001D01T', 'ST127Q01TA', 'ST127Q02TA', 'REPEAT']]
data.head(10)

# heatmap
corr = data.iloc[:, :-1].corr(method="pearson")
cmap = sns.diverging_palette(250, 345, 80, 60, center='dark', as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)

# Calculate P(Y=y) for all possible y
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior=[]
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior

# Calculate P(X=x|Y=y) categorically
def calculate_likelihood_categorical(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    p_x_given_y = len(df[df[feat_name]==feat_val])/len(df)
    return p_x_given_y
```

```

# Calculate  $P(X=x_1|Y=y)P(X=x_2|Y=y).....P(X=x_n|Y=y)P(Y=y)$  for all y and find the
maximum
def naive_bayes_categorical(df,X,Y):
    # get featyre names
    features= list(df.columns)[: -1]
    # calculate prior(df,Y)
    prior = calculate_prior(df,Y)
    Y_pred=[]
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels=sorted(list(df[Y].unique()))
        likelihood=[1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):

likelihood[j]*=calculate_likelihood_categoorical(df,features[i],x[i],Y,labels[j]
])

        # calculate posterior probability(numerator only)
        post_prob=[1]*len(labels)
        for j in range(len(labels)):
            post_prob[j]=likelihood[j]*prior[j]
        Y_pred.append(np.argmax(post_prob))
    return np.array(Y_pred)

# cross validate
group=int(len(data)/5)
train=data[: -1*group]
test=data[-1*group:]
X_test=test.iloc[:, :-1].values
Y_test=test.iloc[:, -1].values
Y_pred=naive_bayes_categorical(train,X=X_test,Y="REPEAT")
accurate=0
acc=0
for i in range(len(Y_test)):
    if Y_pred[i]==Y_test[i]:
        accurate+=1
acc=accurate/len(Y_test)
print(acc)

...

train5=data[group:]
test5=data[:group]
X_test5=test5.iloc[:, :-1].values
Y_test5=test5.iloc[:, -1].values
Y_pred5=naive_bayes_categorical(train5,X=X_test5,Y="REPEAT")
accurate5=0
acc5=0
for i in range(len(Y_test5)):
    if Y_pred5[i]==Y_test5[i]:
        accurate5+=1
acc5=accurate5/len(Y_test5)
print(acc5)

aver=(acc+acc2+acc3+acc4+acc5)/5.0
print(aver)

```

1.4 实验记录

(k折交叉验证, 4: 1比例, 共有5折)

| k | ACC |
|-----|--------------------|
| 1 | 0.9954870837223778 |
| 2 | 0.9967320261437909 |
| 3 | 0.9979769685652039 |
| 4 | 0.9982882041705571 |
| 5 | 0.9982882041705571 |
| 平均值 | 0.9973544973544973 |

2 预测算法实践

2.1 算法主要流程

Step 1 特征工程

1. 特征选择

使用scipy.stats.stats 的 pearsonr , 计算pearson相关系数知, 与MATH相关度较高的特征为 PV值、ST值、REPEAT。

| | NAME | PEARSONR | | | | | | |
|-----|----------|----------|-----|------------|----------|-----|------------|----------|
| 338 | PV15CIE | 0.866300 | 413 | PV6SSLI | 0.789436 | 86 | ST121Q01NA | 0.251537 |
| 345 | PV8SCIE | 0.820269 | 374 | PV7SCID | 0.788911 | 205 | ST129Q07TA | 0.248074 |
| 340 | PV3SCIE | 0.817146 | 387 | PV10SKCO | 0.788712 | 203 | ST129Q05TA | 0.246875 |
| 344 | PV7SCIE | 0.816209 | 358 | PV1SCED | 0.788633 | 72 | ST127Q01TA | 0.243871 |
| 346 | PV9SCIE | 0.815628 | 412 | PV5SSLI | 0.788530 | 134 | ST063Q05NB | 0.243597 |
| 341 | PV4SCIE | 0.815294 | 410 | PV3SSLI | 0.788475 | 204 | ST129Q06TA | 0.239214 |
| 347 | PV10SCIE | 0.814958 | 409 | PV2SSLI | 0.788339 | 201 | ST129Q03TA | 0.236040 |
| 339 | PV2SCIE | 0.814281 | 370 | PV3SCID | 0.788177 | 197 | ST113Q03TA | 0.234886 |
| 343 | PV6SCIE | 0.813352 | 402 | PV5SSPH | 0.788093 | 196 | ST113Q02TA | 0.224112 |
| 342 | PV5SCIE | 0.812225 | 404 | PV7SSPH | 0.787959 | 199 | ST129Q01TA | 0.223512 |
| 390 | PV3SKPE | 0.811548 | 415 | PV8SSLI | 0.787568 | 195 | ST113Q01TA | 0.220493 |
| 396 | PV9SKPE | 0.810392 | 414 | PV7SSLI | 0.787255 | 244 | ST065Class | 0.208486 |
| 395 | PV8SKPE | 0.807477 | 400 | PV3SSPH | 0.787249 | 215 | ST146Q03TA | 0.208486 |
| 394 | PV7SKPE | 0.807361 | 401 | PV4SSPH | 0.787244 | 198 | ST113Q04TA | 0.206736 |
| 391 | PV4SKPE | 0.806875 | 382 | PV5SKCO | 0.786998 | 149 | ST098Q06TA | 0.203509 |
| 393 | PV6SKPE | 0.806391 | 403 | PV6SSPH | 0.785637 | 55 | ST012Q06NA | 0.201173 |
| 392 | PV5SKPE | 0.804587 | 425 | PV8SSES | 0.785078 | 114 | ST062Q01TA | 0.199291 |
| 388 | PV1SKPE | 0.802373 | 426 | PV9SSES | 0.784763 | 216 | ST146Q04TA | 0.194018 |
| 389 | PV2SKPE | 0.799650 | 427 | PV10SSES | 0.784683 | 165 | ST104Q03NA | 0.193580 |
| 397 | PV10SKPE | 0.799589 | 419 | PV2SSES | 0.783213 | 164 | ST104Q02NA | 0.192248 |
| 378 | PV1SKCO | 0.797131 | 421 | PV4SSES | 0.783188 | 166 | ST104Q04NA | 0.189956 |
| 372 | PV5SCID | 0.796963 | 360 | PV3SCED | 0.783105 | 79 | ST118Q04NA | 0.189852 |
| 411 | PV4SSLI | 0.796670 | 367 | PV10SCED | 0.782916 | 67 | ST021Q01TA | 0.188414 |
| 416 | PV5SSLI | 0.795606 | 422 | PV5SSES | 0.779539 | 41 | ST011Q07TA | 0.188206 |
| 398 | PV1SSPH | 0.795545 | 399 | PV2SSPH | 0.779230 | 202 | ST129Q04TA | 0.183443 |
| 417 | PV10SSLI | 0.795519 | 364 | PV7SCED | 0.778526 | 167 | ST104Q05NA | 0.182062 |
| 356 | PV9SCPE | 0.795126 | 359 | PV2SCED | 0.778427 | 105 | ST039Q03NA | 0.181339 |
| 353 | PV6SCPE | 0.794488 | 420 | PV3SSES | 0.778166 | 27 | ST006Q02TA | 0.181068 |
| 368 | PV1SCID | 0.794180 | 362 | PV5SCED | 0.777659 | 32 | ST008Q02TA | 0.176106 |
| 376 | PV9SCID | 0.794007 | 418 | PV1SSES | 0.777632 | 213 | ST146Q01TA | 0.171622 |
| 371 | PV4SCID | 0.793270 | 363 | PV6SCED | 0.776339 | 184 | ST093Q08NA | 0.164438 |
| 424 | PV7SSSES | 0.793042 | 423 | PV6SSES | 0.774320 | 115 | ST062Q02TA | 0.163900 |
| 406 | PV9SSPH | 0.792325 | 361 | PV4SCED | 0.773339 | 160 | ST103Q03NA | 0.162623 |
| 348 | PV1SCPE | 0.792228 | 328 | PV1READ | 0.771321 | 78 | ST118Q03NA | 0.160686 |
| 408 | PV1SSLI | 0.791714 | 365 | PV8SCED | 0.767409 | 42 | ST011Q08TA | 0.159860 |
| 355 | PV8SCPE | 0.791698 | 366 | PV9SCED | 0.764678 | 152 | ST098Q09TA | 0.156035 |
| 377 | PV10SCID | 0.791691 | 334 | PV7READ | 0.739880 | 43 | ST011Q09TA | 0.155926 |
| 351 | PV4SCPE | 0.791573 | 333 | PV6READ | 0.736715 | 234 | ST078Q02NA | 0.155879 |
| 357 | PV10SCPE | 0.791511 | 337 | PV10READ | 0.735570 | 194 | ST095Q15NA | 0.155259 |
| 379 | PV2SKCO | 0.791483 | 335 | PV8READ | 0.733312 | 163 | ST104Q01NA | 0.155121 |
| 384 | PV7SKCO | 0.791333 | 332 | PV5READ | 0.731972 | 138 | ST064Q02NA | 0.154033 |
| 407 | PV18SSPH | 0.791259 | 331 | PV4READ | 0.731085 | 221 | ST146Q09NA | 0.152801 |
| 352 | PV5SCPE | 0.791172 | 329 | PV2READ | 0.730023 | 108 | ST039Q06NA | 0.152568 |
| 380 | PV3SKCO | 0.791097 | 330 | PV3READ | 0.727893 | 240 | ST078Q08NA | 0.151197 |
| 349 | PV2SCPE | 0.790703 | 336 | PV9READ | 0.726312 | 120 | ST071Q04NA | 0.151071 |
| 381 | PV4SKCO | 0.790590 | 21 | ST001D01T | 0.531211 | 121 | ST071Q05NA | 0.150908 |
| 369 | PV2SCID | 0.790573 | 429 | REPEAT | 0.514225 | 190 | ST095Q04NA | 0.150883 |
| 383 | PV6SKCO | 0.790275 | 128 | ST063Q02NB | 0.321099 | 222 | ST076Q01NA | 0.149970 |
| 350 | PV3SCPE | 0.790233 | 130 | ST063Q03NB | 0.319982 | 191 | ST095Q07NA | 0.147748 |
| 373 | PV6SCID | 0.790178 | 73 | ST127Q02TA | 0.304912 | 106 | ST039Q04NA | 0.147739 |
| 405 | PV8SSPH | 0.790023 | 127 | ST063Q02NA | 0.296638 | 193 | ST095Q13NA | 0.147345 |
| 386 | PV9SKCO | 0.789841 | 200 | ST129Q02TA | 0.279715 | 192 | ST095Q08NA | 0.147112 |
| 385 | PV8SKCO | 0.789763 | 125 | ST063Q01NA | 0.273157 | 170 | ST107Q03NA | 0.146746 |
| 354 | PV7SCPE | 0.789702 | 126 | ST063Q01NB | 0.268750 | 236 | ST078Q04NA | 0.146026 |
| 375 | PV8SSCI | 0.789465 | 206 | ST129Q08TA | 0.255327 | 237 | ST078Q05NA | 0.145362 |

以线性模型为例，分别使用PV值、ST值、PV+ST+REPAET、所有特征，计算MSE：

| 特征组合 | MSE (5折交叉验证平均值) |
|--------------|--------------------|
| PV值 | 1545.420742753812 |
| ST值 | 3310.4784834860925 |
| PV+ST+REPAET | 1375.8153971736388 |
| 所有特征 | 1384.067319636236 |

可见选择PV+ST+REPEAT对应的MSE最小，故选择使用它进行预测。

2. 预处理

(1) 缺失值处理

使用均值填充缺失值。

(2) 标准化

①使用 sklearn.preprocessing 类中的 StandardScaler，标准化样本x：

$$z = (x - u) / s$$

②使用MinMaxScaler将特征缩放到一个范围：

$$X_{std} = (X - X.min(axis = 0)) / (X.max(axis = 0) - X.min(axis = 0))$$

$$X_{scaled} = X_{std} * (max - min) + min$$

(3) 归一化

使用 sklearn.preprocessing 模块的 Normalizer，归一化为l2范数。

(4) 数据变换

对数据取对数。

以线性模型为例，分别使用 StandardScaler、MinMaxScaler、Normalizer、取对数，计算MSE：

| 预处理模型 | MSE (5折交叉验证平均值) |
|----------------|--------------------|
| 无预处理 | 1375.8153971736388 |
| StandardScaler | 1376.6503810356621 |
| MinMaxScaler | 1375.787439689372 |
| Normalizer | 1374.1119323622445 |
| 取对数 | 1626.280473738982 |

综上可见无预处理、使用StandardScaler、MinMaxScaler、Normalizer的MSE差不多，则需要根据具体模型再具体比较。

Step 2 构建模型

该问题为回归问题，选择了以下模型：

1. 线性模型

- 普通最小二乘线性回归

$$\min_w \|Xw - y\|_2^2$$

- 岭回归

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2$$

- 贝叶斯岭回归

$$p(w|\lambda) = \mathcal{N}(w|0, \lambda^{-1} \mathbf{I}_p)$$

- 广义线性回归

$$\min_w \frac{1}{2n_{\text{samples}}} \sum_i d(y_i, \hat{y}_i) + \frac{\alpha}{2} \|w\|_2^2,$$

2. SVM

支持向量机 (SVM) 是一组用于分类、回归和异常值检测的监督学习方法。

SVR模型中的自由参数是 C 和 epsilon。该实现基于 libsvm。拟合时间复杂度超过样本数量的二次方，这使得很难扩展到具有超过 10000 个样本的数据集。对于大型数据集，可以考虑使用 LinearSVR 或 SGDRegressor，可能在 Nystroem 转换器之后。

LinearSVR与参数 kernel='linear' 的 SVR 类似，但根据 liblinear 而不是 libsvm 实现，因此它在选择惩罚和损失函数方面具有更大的灵活性，并且应该更好地扩展到大量样本。此类支持密集和稀疏输入。

3. 最近邻回归

在数据标签是连续变量而不是离散变量的情况下，可以使用基于邻居的回归。分配给查询点的标签是根据其最近邻居的标签的平均值计算的。

支持向量分类产生的模型（如上所述）仅依赖于训练数据的一个子集，因为构建模型的成本函数并不关心超出边界的训练点。类似地，支持向量回归产生的模型只依赖于训练数据的一个子集，因为成本函数忽略了预测接近目标的样本。

4. 决策树

决策树用于拟合带有噪声观察的正弦曲线。所以，它学习了近似正弦曲线的局部线性回归。

5. 随机森林

随机森林是一种元估计器，它在数据集的各种子样本上拟合许多分类决策树，并使用平均来提高预测准确性和控制过拟合。如果 `bootstrap=True`（默认），则使用 `max_samples` 参数控制子样本大小，否则使用整个数据集来构建每棵树。

6. MLP

多层感知器 (MLP) 是一种监督学习算法，它通过在数据集上训练来学习函数 $f(\cdot) : R^m \rightarrow R^o$ ，其中 m 是输入的维数和输出的维数。给定一组特征 $X = x_1, x_2, \dots, x_m$ 和一个目标 y ，它可以学习用于分类或回归的非线性函数逼近器。它与逻辑回归的不同之处在于，在输入层和输出层之间，可以有一个或多个非线性层，称为隐藏层。

- 单层神经网络
- 多层神经网络

2.2 实验记录

1. 线性模型

- 普通最小二乘线性回归

已知使用Normalizer预处理的效果最好，结果如下：

(k折交叉验证，4: 1比例，共有5折)

| k | MSE |
|-----|--------------------|
| 1 | 1347.78393735 |
| 2 | 1376.44951398 |
| 3 | 1386.0070966 |
| 4 | 1390.81222927 |
| 5 | 1369.50688461 |
| 平均值 | 1374.1119323622445 |

调参：

| fit_intercept | MSE (5折交叉验证平均值) |
|---------------|--------------------|
| True | 1374.1119323622445 |
| False | 1375.8162814352863 |

选择fit_intercept=True。

- 岭回归

| 预处理模型 | MSE (5折交叉验证平均值) |
|----------------|--------------------|
| 无预处理 | 1375.8150089985106 |
| StandardScaler | 1375.7859749105896 |
| MinMaxScaler | 1375.7347883367029 |
| Normalizer | 6144.146151929144 |

可见无预处理、使用StandardScaler、MinMaxScaler、Normalizer的MSE差不多，选择MinMaxScaler进行接下来的调参。

调参：

| alpha | MSE |
|-------|--------------------|
| 0.1 | 1375.7347883367029 |
| 0.5 | 1375.4788470839153 |
| 1 | 1375.2916734349888 |
| 1.5 | 1375.2321866759394 |
| 2 | 1375.2840892655872 |
| 2.5 | 1375.4341381352042 |

取alpha=1.5左右较合适。

继续调整其他的参数，发现效果并不能显著提升。

- 贝叶斯岭回归

| 预处理模型 | MSE (5折交叉验证平均值) |
|----------------|--------------------|
| 无预处理 | 1377.562009070572 |
| StandardScaler | 1375.5841667712743 |
| MinMaxScaler | 1375.6732167739344 |
| Normalizer | 1377.0111009354296 |

使用StandardScaler预处理的效果最好。

调参:

| tol | MSE |
|------|--------------------|
| 1e-4 | 1375.5841667712743 |
| 1e-3 | 1375.5841671136398 |
| 1 | 1375.5855877353235 |
| 10 | 1375.5855877353235 |
| 100 | 1375.5843498522825 |

MSE的变化不大。

继续调整其他的参数，发现效果并不能显著提升。

- 广义线性回归

| 预处理模型 | MSE (5折交叉验证平均值) |
|----------------|--------------------|
| 无预处理 | 6942.665423163434 |
| StandardScaler | 1401.3295823884023 |
| MinMaxScaler | 1417.8453777678137 |
| Normalizer | 6017.14331933596 |

可见StandardScaler的效果较好。

| power | MSE (5折交叉验证平均值) |
|-------|--------------------|
| 0 | 1401.3295823884023 |
| 1 | 1402.8912927329318 |
| 2 | 1628.6738779872653 |
| 3 | 5475.4570682135945 |

可见power=0的效果最好。

继续调整其他的参数，发现效果并不能显著提升。

○ LinearSVR

| 预处理 | MSE(5折交叉验证平均值) |
|----------------|--------------------|
| 无 | 5072.063403794792 |
| StandardScaler | 1388.704925497831 |
| MinMaxScaler | 1477.6583950272986 |
| Normalizer | 6943.69740744765 |

可见StandardScaler的效果最好。

| max_iter | MSE(5折交叉验证平均值) |
|----------|--------------------|
| 1000 | 1388.704925497831 |
| 2000 | 1388.536142216777 |
| 5000 | 1388.68544473371 |
| 10000 | 1388.8103069696797 |

max_iter值的影响不大，后续调参发现存在未收敛问题，取较max_iter=10000。

| tol | MSE(5折交叉验证平均值) |
|------|--------------------|
| 1e-4 | 1388.68544473371 |
| 5e-5 | 1388.8701838042848 |
| 1e-5 | 1388.5102881564526 |
| 1e-6 | 1388.42888925356 |

tol值的影响不大，取tol=1e-6。

| C | MSE(5折交叉验证平均值) |
|---|--------------------|
| 1 | 1388.42888925356 |
| 5 | 1388.6125174162837 |

C值的影响不大，取C=1。

| loss | MSE(5折交叉验证平均值) |
|-----------------------------|------------------|
| epsilon_insensitive | 1388.42888925356 |
| squared_epsilon_insensitive | 1376.35622874785 |

可见loss为squared_epsilon_insensitive的效果更好，取loss='squared_epsilon_insensitive'。但此时取的max_iter=10000仍提示未收敛，故设置

结果如下：

(k折交叉验证，4: 1比例，共有5折)

| k | MSE |
|-----|-------------------|
| 1 | 349.06739279 |
| 2 | 1376.93477516 |
| 3 | 1388.04007864 |
| 4 | 1394.01146644 |
| 5 | 1370.99413503 |
| 平均值 | 1375.809569611593 |

2. 最近邻回归

调参：

| n_neighbors | MSE (5折交叉验证平均值) |
|-------------|--------------------|
| 2 | 2582.2085493122536 |
| 5 | 2061.2447783030802 |
| 10 | 1904.0411491159161 |
| 20 | 1830.1744632589136 |
| 50 | 1788.7863235564341 |
| 100 | 1788.2441994219357 |

继续调整参数，发现效果没有显著提升。

3. 决策树

使用Normalizer预处理，调整参数max_depth：

| max_depth | MSE (5折交叉验证平均值) |
|-----------|--------------------|
| None | 3235.651685031461 |
| 2 | 2567.318976675281 |
| 5 | 1765.6335884839923 |
| 7 | 1773.708740632666 |
| 10 | 2115.76917128711 |

选择max_depth=5，调整预处理方式：

| 预处理 | MSE (5折交叉验证平均值) |
|----------------|--------------------|
| 无预处理 | 1733.9744556914302 |
| StandardScaler | 1733.9866763971775 |
| MinMaxScaler | 1733.9866763971775 |
| Normalizer | 1765.6335884839923 |

可见StandardScaler和MinMaxScaler的效果较好。

4. 随机森林

鉴于跑一次5折交叉验证的时间较长，故选择手动划分4: 1的训练集与测试集进行调参。

| random_state | MSE (on test) |
|--------------|---------------|
| 1 | 1524.909 |
| 10 | 1529.847 |

可见变化不大。

| n_estimators | MSE (on train) | MSE (on test) |
|--------------|-----------------|---------------|
| 50 | 227.271 | 1541.161 |
| 100 | 217.069 | 1524.909 |
| 200 | 209.354 | 1515.271 |

该模型在训练集上的MSE很小，在测试集上的MSE变化不大，说明容易过拟合。

5. 神经网络

鉴于跑一次5折交叉验证的时间较长，故选择手动划分4：1的训练集与测试集进行调参。

o 单层神经网络

| hidden_layer_sizes | MSE (on test) |
|--------------------|---------------|
| 1000 | 1931.248 |
| 500 | 1846.179 |
| 200 | 1812.474 |
| 100 | 1800.622 |
| 50 | 1816.359 |

取hidden_layer_sizes=100。

| batch_size | MSE (on test) |
|------------|---------------|
| 200 | 1800.622 |
| 400 | 1795.711 |
| 500 | 1795.621 |
| 600 | 1816.038 |
| 1000 | 1807.391 |

取batch_size=500。

| alpha | MSE (on test) |
|-------|---------------|
| 2e-3 | 1814.487 |
| 5e-4 | 1799.993 |
| 2e-4 | 1791.754 |
| 1e-4 | 1791.019 |
| 2e-5 | 1795.711 |
| 2e-6 | 1836.466 |

取alpha=1e-4。

继续调参，发现效果没有显著提升。

- 多层神经网络

| hidden_layer_sizes | MSE (on test) |
|--------------------|---------------|
| (4,4,) | 6829.898 |
| (8,8,) | 1803.341 |
| (16,16) | 1835.912 |
| (64,64,) | 2473.297 |
| (32,32,) | 1793.429 |
| (8,8,8,) | 2076.437 |
| (16,16,16,) | 1840.166 |
| (32,32,32,) | 2172.594 |
| (8,8,8,8,) | 1790.203 |

取hidden_layer_sizes=(8,8,8,8,)

| alpha | MSE (on test) |
|-------|---------------|
| 2e-3 | 3041.097 |
| 2e-4 | 1790.203 |
| 3e-4 | 1787.517 |
| 4e-4 | 2117.977 |
| 2e-5 | 6847.103 |

取alpha =3e-4。

继续调整参数，发现效果没有显著提升。

综上所述，可见LinearRegression、Ridge、BayesianRidge、linearSVR的效果均较好，最终选择LinearRegression进行预测。

2.3 算法实现

代码实现的关键片段

```
# Importing the libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import preprocessing
```

```

# 数据导入
initialdata = pd.read_csv("D:\\jupyterlab\\lab4\\pica2015.csv",na_values=' ')
df = pd.DataFrame(initialdata)

# 均值填充缺失值
pd.set_option('display.max_rows',None)
pd.set_option('display.max_columns',None)
for column in list(df.columns[df.isnull().sum() > 0]):
    mean_val = df[column].mean()
    df[column].fillna(mean_val, inplace=True)
print(df.isna().sum())

# 计算pearsonr系数
from scipy.stats.stats import pearsonr
pea=[] for i in range(430)
for i in range(19,430):
    temp=df.iloc[:, i]
    if temp.dtypes!='object':
        pea[i]=[df.iloc[0:0,i].name,pearsonr(temp,df['MATH'])[0]]
        print(pea[i])
for i in range(19,430):
    if pea[i][1]<0:
        pea[i][1]=-1*pea[i][1]
peadf=pd.DataFrame(pea,columns=['NAME', 'PEARSONR'])
# peadf.dropna(inplace=True)
# peadfabs=abs(peadf)
peadf.sort_values(by='PEARSONR', inplace=True,ascending = False) #
inplace=True, 表示直接替换原有df对象
print(peadf)

#获取行列信息
pd.get_option("max_info_columns")
pd.options.display.max_info_columns = 450
df.info()

# Use feature ST+PV+REPEAT
X_PV=df.iloc[:,328:428]
X_ST=df.iloc[:,21:245]
X_STPVRE=pd.concat([X_ST,X_PV,df['REPEAT']], axis=1)
Y=df.iloc[:,-1]

# 预处理
Normalizer = preprocessing.Normalizer().fit_transform(X_STPVRE)
MinMaxScaler=preprocessing.MinMaxScaler().fit_transform(X_STPVRE)
StandardScaler=preprocessing.StandardScaler().fit_transform(X_STPVRE)

#使用线性模型
regr = linear_model.LinearRegression()
#计算5折交叉验证MSE
score = cross_val_score(regr,Normalizer, Y,
cv=5,scoring='neg_mean_squared_error')
print(score)
print(np.mean(score))

# 预测
topred=pd.read_csv("D:\\桌面文件\\pica2015.csv",na_values=' ')
df_topred = pd.DataFrame(topred)
# 均值填充缺失值
for column in list(df_topred.columns[df_topred.isnull().sum() > 0]):

```

```
mean_val = df_topred[column].mean()
df_topred[column].fillna(mean_val, inplace=True)
regr = linear_model.LinearRegression()
regr.fit(Normalizer, Y)
X_topred=pd.concat([df_topred.iloc[:6426,21:245],df_topred.iloc[:6426,328:428],d
f_topred.iloc[:6426,-1]], axis=1)
X_topred_Normalizer=preprocessing.Normalizer().fit_transform(X_topred)
y_train_pred = regr.predict(Normalizer)
# print(np.mean(y_train_pred))
y_test_pred = pd.Series(regr.predict(X_topred_Normalizer))
print('MSE train: %.3f' % (mean_squared_error(Y, y_train_pred)))
y_test_pred.rename('MATH', inplace=True)
y_test_pred.to_csv("pred3.csv", index_label="index")
```

附录

NaiveBayes.ipynb

lab4.ipynb