**Lab 5 实验报告**

PB20000296 郑滕飞

**目的/原理：**

**1、实验目的**

用不同机器学习算法对指定数据集进行四分类预测，并对比各种预处理方式与机器学习算法的效果。

**2、实验原理**

本次实验涉及的预处理方式包括对空数据与异常数据的处理、对单个特征基于一定评判标准(基尼系数、信息熵、方差)的筛选、主成分分析。而具体对比的学习器有普通逻辑斯蒂回归、决策树、神经网络、XGBoost与SVM五种。除了神经网络之外，其余学习器都是在之前实验中实现过的(其中决策树与XGBoost共同形成回归树，由于与本次多分类任务不同，采用调包实现)，而调用的自己实验的代码包括Logistic.py[内含逻辑斯蒂回归二分类器]与SVM.py[内含SMO算法实现的SVM]。

**3、文件目录**

main.ipynb 主要文件，包含对比测试与最终结果输出

main\_drop.ipynb 另一种处理方式(未被选取)，也包含对比测试与最终结果

test.py 包含测试用的部分函数

test\_pre.ipynb 对比各种预处理方式

test\_choice.ipynb 对比五种学习器的参数选择

Logistic.py 自己实现的逻辑斯蒂回归[来自以往实验]

SVM.py 自己实现的支持向量机[来自以往实验]

test\_label.csv 最终预测的标签

report.pdf 实验报告

**预处理[test\_pre.ipynb]：**

**1、空值与异常处理**

读取文件后，为了保证能正确存入numpy的向量中，需要去除空值。考虑到越界值的存在，这里定义了直接去掉空值所在行的读取方式readfile\_drop与用本列中位数填充空值的读取方式readfile。

此外，直接观察数据可发现，数据中有明显的异常值，因此构造了函数drop\_err，将绝对值在均值绝对值三百倍以上的点删除。

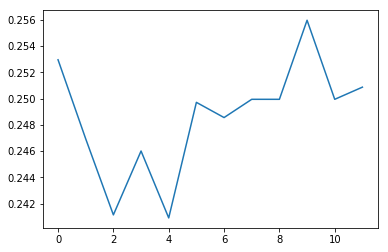
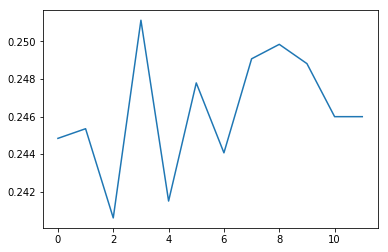
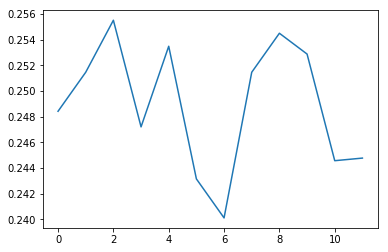
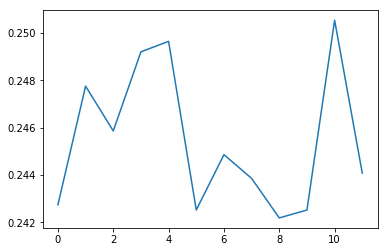
记读取并填充空值的数据集为X，删除空值的为Xd，填充空值后删除异常为Xr，删除空值与异常的为Xrd，也将它们的y对应命名，之后的对比将在这些之中进行。

**2、学习器选择**

为了对比预处理的效果，需要构造一个统一的学习器。由于数据中有异常数据与空值，直接将所有数据的所有特征进行学习的学习器应无法达到测试效果，于是回归与神经网络等模型可以排除。但是，决策树与基于决策树的XGBoost又以关心标签的顺序关系为主，无法刻画具体大小幅度造成的影响，因此，最终选择利用sklearn中默认参数的SVC作为用于比较的标准学习器[这里为了计算速度，调包实现了，最后比较结果时利用的是自己实现的]。以下计算的所有准确率都是三次随机7:3划分数据-测试的平均准确率。

**2、特征选取-信息熵**

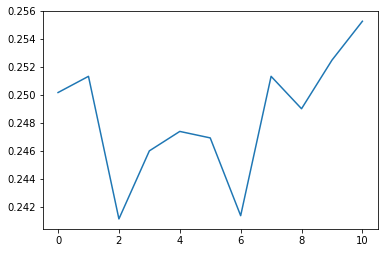
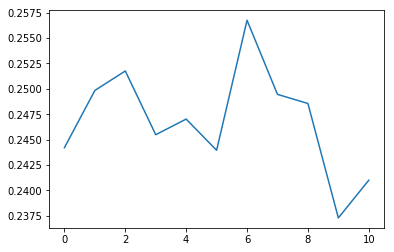
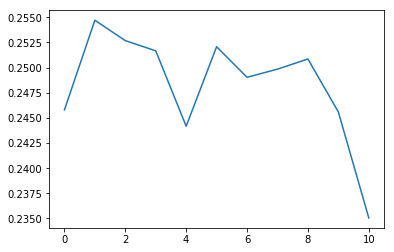
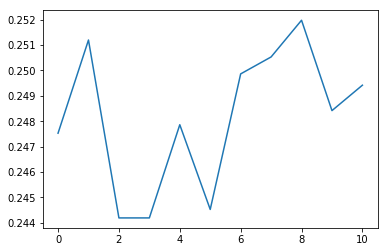
根据一些预先的测试，不同特征之间的差别很大，且由于异常数据无法完全剔除，利用方差进行预选取是完全无效的。因此，预处理的第一部分中，实现了基于比较信息熵的特征选取(即删掉信息熵相对最低的若干特征)，并在X、Xd、Xr与Xrd上测试，根据删除特征的数量为横轴，准确率为纵轴进行绘图，依次为[详细图片见ipynb文件中]：



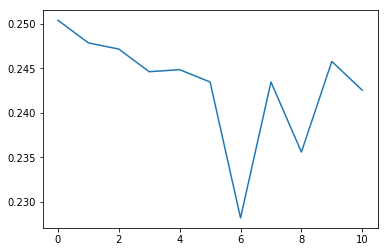
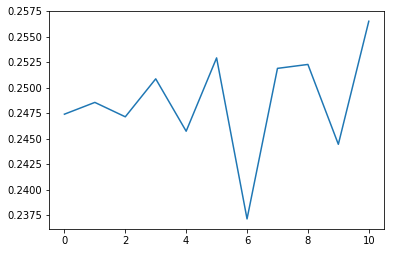
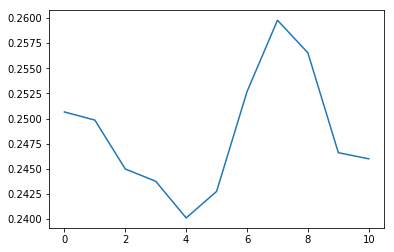
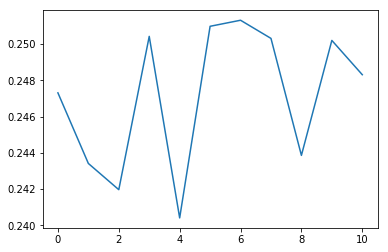
信息熵初筛一般筛选掉20-30个特征，对应横轴2-3的位置，而在这个段落内，可以发现单特征选择的表现几乎都不理想(只有Xd超过了0.25，且根据进一步测试可知较为偶然)，于是基本排除了单特征的选取方式。对基尼系数选取，结果是类似的。

**3、特征选取-RFE**

单特征既然选取失败，就对综合多个特征的选取方式，这里使用了代表性的递归特征删除法。递归特征删除法的原理是，先用学习器进行学习，再将学习器学习中占到较低重要性的特征删除，重复直到剩下的特征数量符合要求。这里首先尝试了与单特征较为接近的决策树为基学习器的选取：



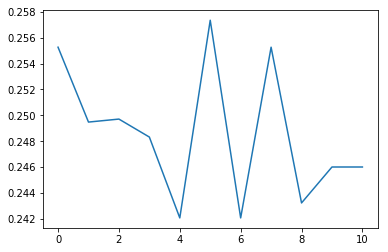
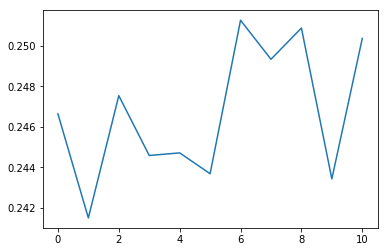
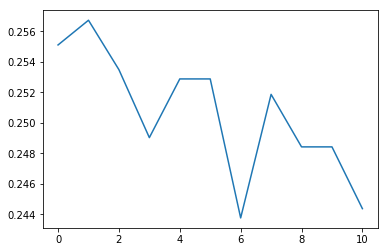
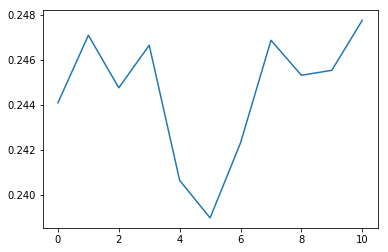
总体来看，它的效果要比信息熵进行特征选取要好一些，而且在删除特征不是过多时相对稳定。之后又实验了以逻辑斯蒂回归为基学习器的选取：



直接回归的结果并不如决策树进行特征选取，这是由于异常数据未经处理，容易影响回归结果。根据之后的进一步实验，先用决策树进行RFE，再用回归方法进行RFE可以在压缩时损失信息较少。

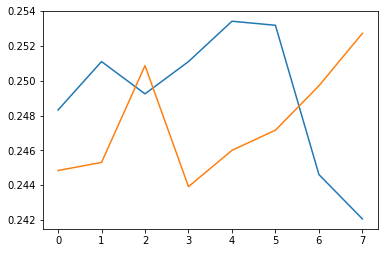
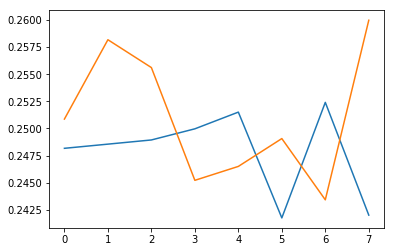
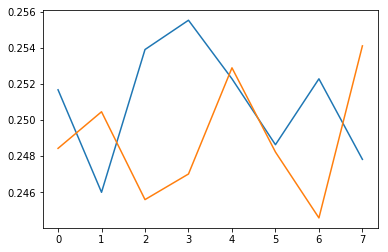
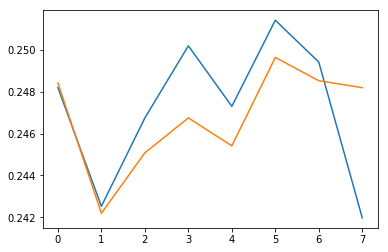
**3、主成分分析**

虽然事先已有主成分分析会导致无效特征信息过多的猜测，但还是进行了测试：

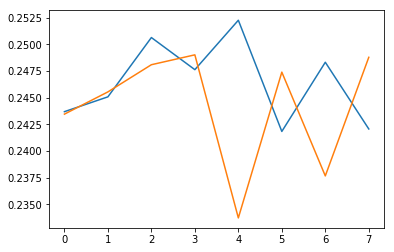
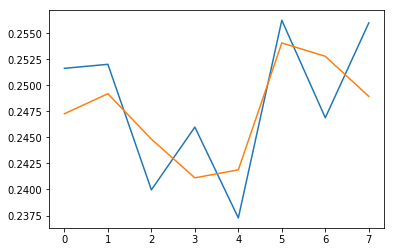
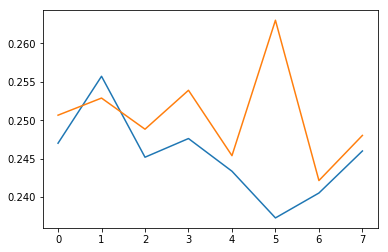
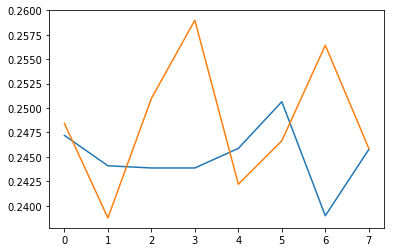


从结果来看，提取出的主成分越少，效果反而越差(X和Xd始终保持在0.25以下，而Xr和Xrd总体下降)。

后来还测试了在有RFE预处理时的PCA[蓝线为保留较多，黄线为保留较少，下同]：



和有回归预处理时的PCA：



总体来看，都没有对结果起到多少作用。因此，最终确定的选择方式是先通过决策树作RFE，将特征压缩到85个，再通过回归，进一步压缩到60个。

**参数选择[test\_choice.ipynb]：**

**1、读文件与预处理**

最终选择的预处理如上文。根据进一步的实验与分析，将某特征在均值300倍以上的数据认为异常可能会损失有效数据，因此最终通过丢弃比例的确定将数字变为了均值的500倍。

此外，在对比参数时，为了做到充分学习、对比，采用了填充空特征的方式，也就是说与上文Xd的生成基本相同。

**2、学习器的集成方式**

由于之前实现的LR与SVM都是二分类学习器，这里使用了ECOC的方式将其进行集成，以LR为例，代码如下：

def lr\_mul(X, y, p, X\_t, it, lr):

r = LogisticRegression(gamma = 1e-4)

y1 = y

for j in range(y1.size):

if y1[j] in p: y1[j] = 1

else: y1[j] = 0

r.fit(X, y1, max\_iter=it, lr=lr)

return r.predict(X\_t)

def test\_lr(X, y, X\_t, it, lr):

yp1 = lr\_mul(X, y, [0,1], X\_t, it, lr)

yp2 = lr\_mul(X, y, [0,2], X\_t, it, lr)

yp3 = lr\_mul(X, y, [0,3], X\_t, it, lr)

yp4 = lr\_mul(X, y, [0], X\_t, it, lr)

yp5 = lr\_mul(X, y, [1], X\_t, it, lr)

yp6 = lr\_mul(X, y, [2], X\_t, it, lr)

yp7 = lr\_mul(X, y, [3], X\_t, it, lr)

yp = np.zeros(yp1.size, dtype="int")

x = np.zeros(4, dtype="int")

for i in range(yp.size):

now = np.array([yp1[i],yp2[i],yp3[i],yp4[i],yp5[i],yp6[i],yp7[i]])

x[0] = np.sum(np.array([1,1,1,1,0,0,0]) == now)

x[1] = np.sum(np.array([1,0,0,0,1,0,0]) == now)

x[2] = np.sum(np.array([0,1,0,0,0,1,0]) == now)

x[3] = np.sum(np.array([0,0,1,0,0,0,1]) == now)

a = np.concatenate(np.argwhere(x == np.max(x)))

yp[i] = np.random.choice(a)

return yp

由于是二分类变为四分类，总共有7种可能的有效分割方式，于是训练7个相同参数的学习器，并根据每个学习器的结果进行集成，选择海明距离最小的作为最终结果。若有相同最小值，则在其中随机选择一个(这里比较了相同的元素个数，因此取最大值)。

**3、核心参数的选取**

为了方便对比与减少时间复杂度，对于五种学习器，通过一些实验与调查找到了各自的两个本实验中的核心参数，并对它们进行比较。

对逻辑斯蒂回归，由于无需考虑稀疏性，采用L2范数。此外，简单对比后选取了1e-4的正则化项，而其核心参数在于迭代次数与学习率。

对决策树，最重要的是最高层数与每个分支最小的数据量。

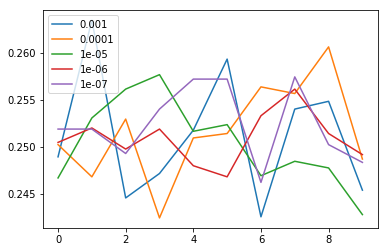
对神经网络，最重要的是神经元层数与迭代次数。

对XGBoost，最重要的是基学习器个数与作为基学习器的决策树的最高层数。

对SVM，效率起见，这里SMO算法中优化的目标是在可优化中随机选择的，最重要的参数是迭代次数与权重C。

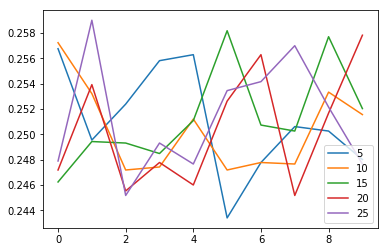
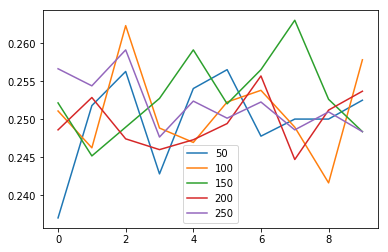
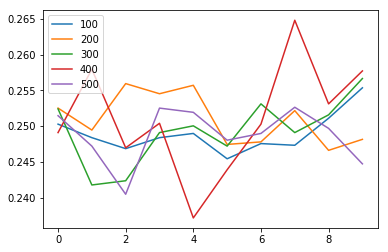
**4、参数选取结果**

逻辑斯蒂回归的结果如下，图例为学习率，横轴与迭代次数相关：



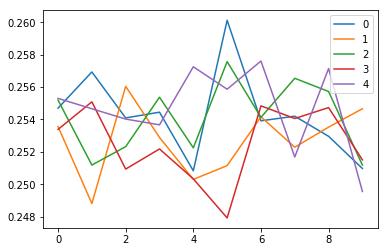
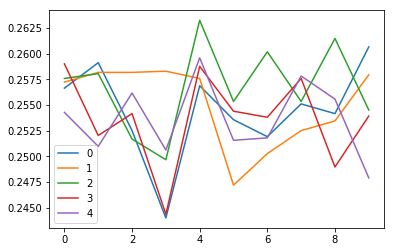
这里学习率1e-7时保证了稳定良好的结果，迭代次数取为50，对应横坐标4。

类似逻辑斯蒂回归，决策树、神经网络、XGBoost参数结果如下：



决策树每个分支最小200时相对稳定，取最高层数16；神经网络迭代150次时稳定较好，取层数80；XGB最大深度15时相对较好，取基学习器个数60。

最后是自己实现的SVM：



这里两张图对应颜色的线对应横坐标的参数是完全相同的，只是第一张图每个位置三次随机测试，第二张图10次，这是因为在下一部分中初步确定SVM最优后进行了进一步的参数调整。根据第一张图初步调整的结果，选择了1.5作为C的取值(即绿线)，并进行50次迭代。第二张图进行进一步判别后，仍取值1.5，但迭代60次。

**对比与输出[main.ipynb/main\_drop.ipynb]：**

**1、test.py构建**

此py文件包含对每个学习器进行测试的函数，输入训练集的X、y与测试集的X，返回预测的y。

之前展示的代码就是对自己集成的学习器的测试，而对调包的则类似(这里的参数已经在上一部分确定)：

def test\_xgb(X, y, X\_t):

c = XGBClassifier(n\_estimators=60, max\_depth=15)

c.fit(X, y)

return c.predict(X\_t)

**2、main与main\_drop**

这里的特征选择与上一部分中过程相同。根据之前预处理的结果，将空值直接删去是比填充总体效果更好的。但是，考虑到最终的test\_feature中也有空值需要填充，无法确定究竟那种更好，因此两种情况下都做了对比，相当于之前的Xr与Xrd。

**3、学习器对比**

此处对比与画图的代码如下：

def test(X0, y0, n):

acc = np.zeros([5, n])

dif = np.zeros([10, n])

for j in range(n):

X, X\_t, y, y\_t = partition(X0, y0)

res = np.zeros([5, y\_t.size], dtype="int")

res[1] = tt.test\_tree(X, y, X\_t)

res[2] = tt.test\_mlp(X, y, X\_t)

res[3] = tt.test\_xgb(X, y, X\_t)

res[4] = tt.test\_svm(X, y, X\_t)

res[0] = tt.test\_lr(X, y, X\_t)

for i in range(5):

acc[i][j] = score(res[i], y\_t)

count = 0

for i in range(4):

for k in range(i+1, 5):

dif[count][j] = 1 - score(res[i], res[k])

count += 1

return acc, dif

def draw\_result(X0, y0, n):

acc, dif = test(X0, y0, n)

plt.plot(acc[0], label="lr")

plt.plot(acc[1], label="tree")

plt.plot(acc[2], label="mlp")

plt.plot(acc[3], label="xgb")

plt.plot(acc[4], label="svm")

plt.legend()

print("average acc:")

print(np.average(acc, axis=1))

dif\_a = np.average(dif, axis=1)

print("lr's dif with other 4:")

print(dif\_a[0:4])

print("tree's dif with other 3:")

print(dif\_a[4:7])

print("mlp's dif with other 2:")

print(dif\_a[7:9])

print("xgb's dif with svm:")

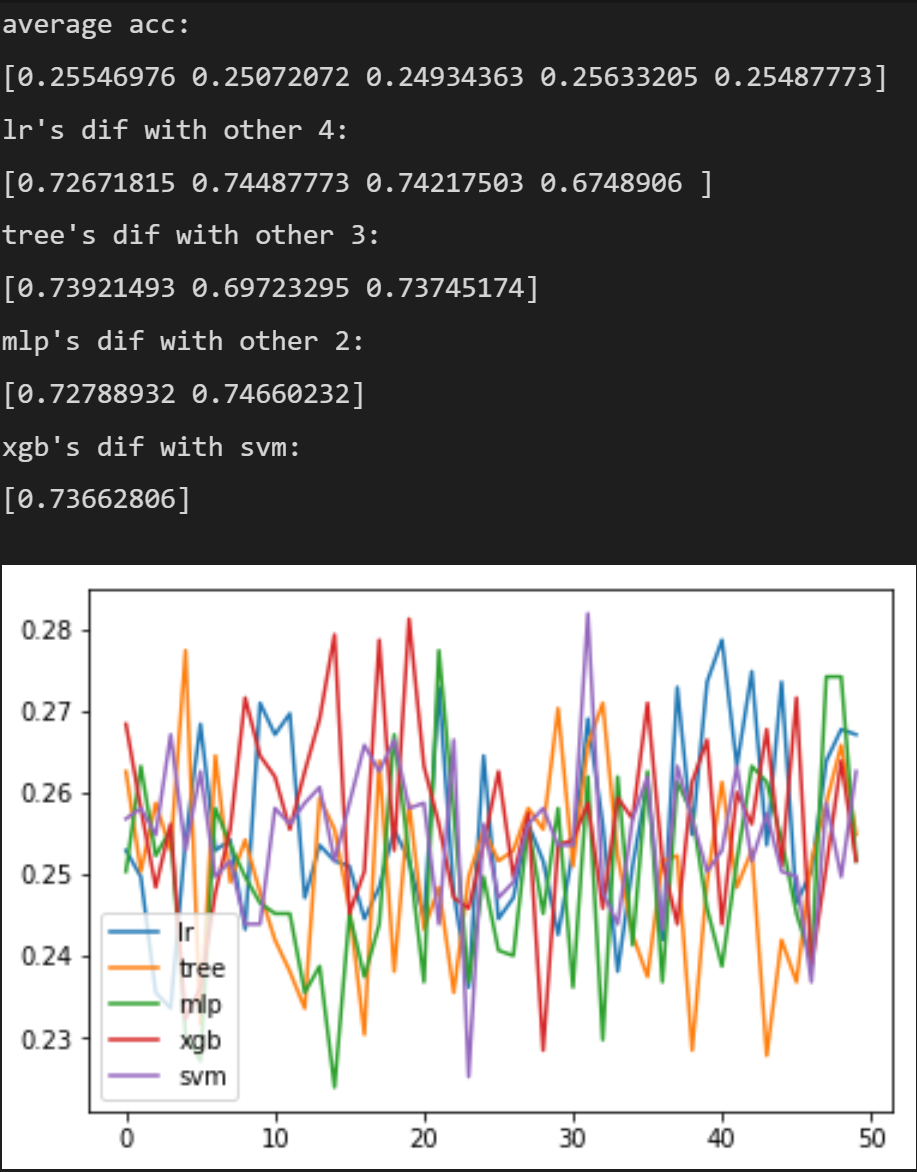
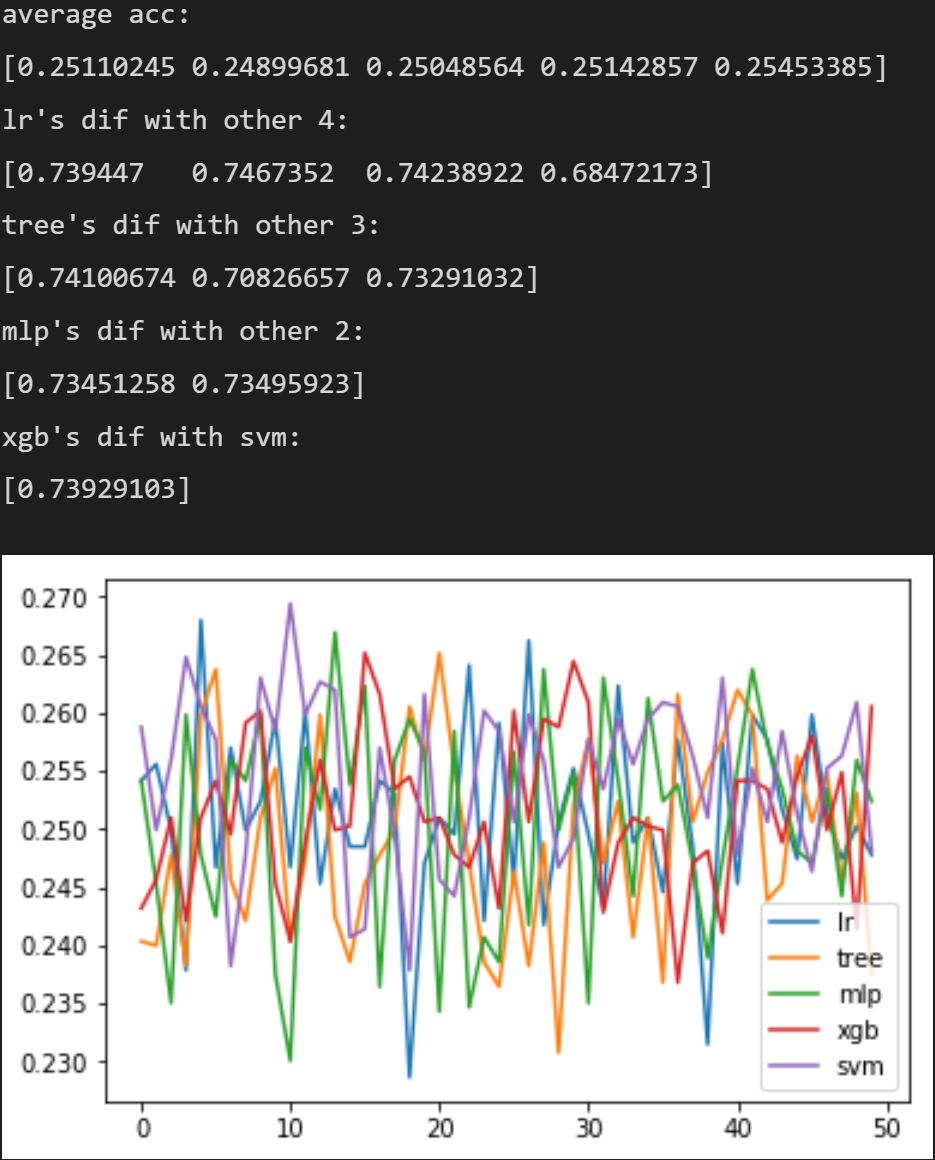
print(dif\_a[9:10])

draw\_result(X0, y0, 50)

这里X0与y0是之前得到的训练集与测试集。在确定的参数下，进行n次测试，并计算每次每个学习器的准确率与不同学习器预测的结果相差，最终将准确率画图并输出平均准确率与差距。

**3、各自结果**

左侧为main中的结果，右侧为main\_drop中的结果，排列顺序为回归、决策树、神经网络、XGBoost、SVM：



从准确率上来看，填充空值时SVM最优，且比其他显著好；删除空值时XGBoost、回归、SVM排前三，且都高于了填充时的最优情况。这样的结果是正常的，因为删去空值也删去了对应的y，对有效数据的预测会更准确。

从接近程度上，回归与SVM始终是最相似的，而回归与神经网络则一直能在相差最大 榜上排前二(在删除空值时，神经网络与SVM的相差是最大的)。

无论哪种情况，普通决策树和神经网络都是效果最差的。前者未经集成时本身较弱，而后者过于不可控，并不适合这种相对简单的样例。

**4、最终选择**

将SVM进行进一步调参后，我们在main与main\_drop中分别针对SVM与XGBoost进行最后测试：将各自的训练集随机划分100次，计算预测的平均准确率与最大准确率，用最大准确率的一次预测测试集，作为预测结果。两者表现出的平均准确率与最大准确率如下：

SVM:

max acc: 0.27437079049982277

average: 0.25501240694789096

XGBoost:

max acc: 0.2812097812097812

average: 0.25332689832689825

这个结果里，虽然XGBoost表现出的最大准确率更高，但平均准确率甚至还不如填充空值的SVM。于是，最终选择在填充空值的情况下取SVM对测试集的最大准确率时预测结果为最终结果，因为虽然最大值看似比起来更低，但它经过了更多数据的测试，也更加稳定。

最后将其输出到文件：

with open("test\_label.csv", "w") as fi:

print("label", file=fi)

for i in predict: print(i, file=fi)

作为最终结果。

**总结：**

说实话，最后做出来的结果也还是挺差，主要来源于，并没有得到任何真正显著高于25%正确率(大概需要至少30%)的学习器，因此也无法以更好的信息进行优化。整个实验中预处理与参数的选择虽然用了很多时间(ipynb文件只展现了部分)，但大部分还是和盲测无区别。

不过没想到的是，这种情况下，线性SVM竟然表现出了相对良好与稳定的性能，也算是有了一个基本的思考方向(但离真正有效的结果还是很远)。

[总之ML实验完结，是时候新年快乐了]