第四次作业

姓名：向建宇

学号：1801210687

1. 用SVM分析病人的住院情况。

数据集情况：共有三类I (2)S (24)A (64)，其中由于I的数量太少，便将I和A归为一类，其实跟用SVM处理多分类也一样，先选一类，其它归为另一类。

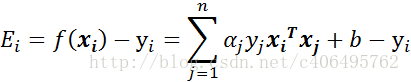
再看数据集：他的属性全是离散特征，这对于SVM求解是不可用的，我看了文档上面给的属性值有一定的范围约束，再结合实际生活，取了随机值，如我将数据集转化为数值：

|  |
| --- |
| 36.550960,34.842814,104.242082,1.256598,15,1  36.205445,37.958771,102.263019,1.910395,10,-1  39.926192,33.870581,98.986865,1.841590,10,1  36.595038,33.759269,90.431251,1.571887,15,1  36.184409,35.877928,98.141203,1.723031,10,1 |

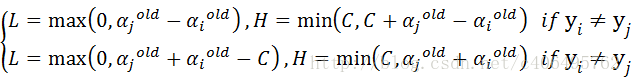
这里只展示前五个数，其中我删去了5.6.7列，因为不知道它的取值范围。

其中最主要的就是SMO算法了，通过前面的数学建模，可以为下面的算法做出铺垫

* 步骤1：计算误差：

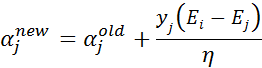
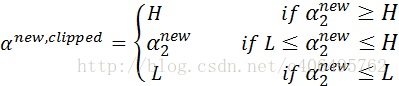


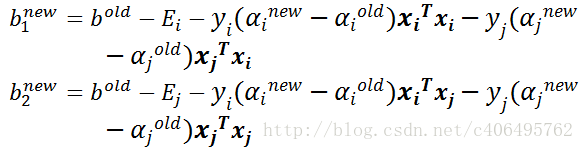
* 步骤2：计算上下界L和H：



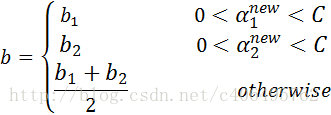
* 步骤3：计算η：



* 步骤4：更新αj：   
  
* 步骤5：根据取值范围修剪αj：   
  
* 步骤6：更新αi：   
  
* 步骤7：更新b1和b2：



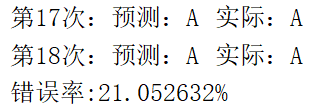
* 步骤8：根据b1和b2更新b：



对应代码：

|  |
| --- |
| **def** innerL(i, oS):  *"""  优化的SMO算法  Parameters：  i - 标号为i的数据的索引值  oS - 数据结构  Returns:  1 - 有任意一对alpha值发生变化  0 - 没有任意一对alpha值发生变化或变化太小  """  #步骤1：计算误差Ei* Ei = calcEk(oS, i)  *#优化alpha,设定一定的容错率。* **if** ((oS.labelMat[i] \* Ei < -oS.tol) **and** (oS.alphas[i] < oS.C)) **or** ((oS.labelMat[i] \* Ei > oS.tol) **and** (oS.alphas[i] > 0)):  *#使用内循环启发方式2选择alpha\_j,并计算Ej* j,Ej = selectJ(i, oS, Ei)  *#保存更新前的aplpha值，使用深拷贝* alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();  *#步骤2：计算上下界L和H* **if** (oS.labelMat[i] != oS.labelMat[j]):  L = max(0, oS.alphas[j] - oS.alphas[i])  H = min(oS.C, oS.C + oS.alphas[j] - oS.alphas[i])  **else**:  L = max(0, oS.alphas[j] + oS.alphas[i] - oS.C)  H = min(oS.C, oS.alphas[j] + oS.alphas[i])  **if** L == H:  *#print("L==H")* **return** 0  *#步骤3：计算eta* eta = 2.0 \* oS.X[i,:] \* oS.X[j,:].T - oS.X[i,:] \* oS.X[i,:].T - oS.X[j,:] \* oS.X[j,:].T  **if** eta >= 0:  print(**"eta>=0"**)  **return** 0  *#步骤4：更新alpha\_j* oS.alphas[j] -= oS.labelMat[j] \* (Ei - Ej)/eta  *#步骤5：修剪alpha\_j* oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)  *#更新Ej至误差缓存* updateEk(oS, j)  **if** (abs(oS.alphas[j] - alphaJold) < 0.00001):  *#print("alpha\_j变化太小")* **return** 0  *#步骤6：更新alpha\_i* oS.alphas[i] += oS.labelMat[j]\*oS.labelMat[i]\*(alphaJold - oS.alphas[j])  *#更新Ei至误差缓存* updateEk(oS, i)  *#步骤7：更新b\_1和b\_2* b1 = oS.b - Ei- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.X[i,:]\*oS.X[i,:].T - oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.X[i,:]\*oS.X[j,:].T  b2 = oS.b - Ej- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.X[i,:]\*oS.X[j,:].T - oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.X[j,:]\*oS.X[j,:].T  *#步骤8：根据b\_1和b\_2更新b* **if** (0 < oS.alphas[i]) **and** (oS.C > oS.alphas[i]): oS.b = b1  **elif** (0 < oS.alphas[j]) **and** (oS.C > oS.alphas[j]): oS.b = b2  **else**: oS.b = (b1 + b2)/2.0  **return** 1  **else**:  **return** 0 |

结果如图：



1. 用adaboost分析学生成绩

根据数据的特征属性高达30项，肯定不能用简单的单层树，所以我的思想是，结合多颗决策树，组成随机森林。

首先根据论文要求，将G3的10分作为及格线，及大于等于十分判为类别1，小于判为0.

处理结果：

|  |
| --- |
| [['"GP"' '"F"' '18' ... '"0"' '"11"' 1]  ['"GP"' '"F"' '17' ... '"9"' '"11"' 1]  ['"GP"' '"F"' '15' ... '"12"' '"13"' 0]  ...  ['"MS"' '"F"' '18' ... '"11"' '"12"' 1]  ['"MS"' '"M"' '17' ... '"10"' '"10"' 1]  ['"MS"' '"M"' '18' ... '"10"' '"11"' 1]] |

随机森林大致过程如下：

1）从样本集中有放回随机采样选出n个样本；

2）从所有特征中随机选择k个特征，对选出的样本利用这些特征建立决策树（一般是CART，也可是别的或混合）；

3）重复以上两步m次，即生成m棵决策树，形成随机森林；

4）对于新数据，经过每棵树决策，最后投票确认分到哪一类。

一．处理数据：

切分数据集，实现交叉验证。

第一步，将训练集划分为大小相同的K份；

第二步，我们选择其中的K-1分训练模型，将用余下的那一份计算模型的预测值，

这一份通常被称为交叉验证集；第三步，我们对所有考虑使用的参数建立模型并做出预测，然后使用不同的K值重复这一过程。

|  |
| --- |
| **def** splitDataSet(dataSet,n\_folds):  fold\_size=len(dataSet)/n\_folds  data\_split=[]  begin=0  end=fold\_size  **for** i **in** range(n\_folds):  data\_split.append(dataSet[begin:end,:])  begin=end  end+=fold\_size  **return** data\_split   *#构建n个子集* **def** get\_subsamples(dataSet,n):  subDataSet=[]  **for** i **in** range(n):  index=[]  **for** k **in** range(len(dataSet)):  index.append(np.random.randint(len(dataSet)))  subDataSet.append(dataSet[index,:])  **return** subDataSet  *#划分数据集* **def** binSplitDataSet(dataSet,feature,value):  mat0=dataSet[np.nonzero(dataSet[:,feature]>value)[0],:]  mat1=dataSet[np.nonzero(dataSet[:,feature]<value)[0],:]  **return** mat0,mat1 |

第二步用数据集进行迭代，计算每一棵树的权重值

|  |
| --- |
| def treeForecast(tree,data,alpha="huigui"):  if alpha=="huigui":  if not isinstance(tree,dict):  return float(tree)  if data[tree['bestFeature']]>tree['bestVal']:  if type(tree['left'])=='float':  return tree['left']  else:  return treeForecast(tree['left'],data,alpha)  else:  if type(tree['right'])=='float':  return tree['right']  else:  return treeForecast(tree['right'],data,alpha)  else:  if not isinstance(tree,dict):  return int(tree)  if data[tree['bestFeature']]>tree['bestVal']:  if type(tree['left'])=='int':  return tree['left']  else:  return treeForecast(tree['left'],data,alpha)  else:  if type(tree['right'])=='int':  return tree['right']  else:  return treeForecast(tree['right'],data,alpha)  def gini(dataSet):  corr=0.0  for i in set(dataSet[:,-1]):  corr+=(len(np.nonzero(dataSet[:,-1]==i)[0])/len(dataSet))\*\*2  return 1-corr |

其中把分错的类的权重加大，正确的减少，并计算当前弱分类器所占的权重。

三．随机森林预测：

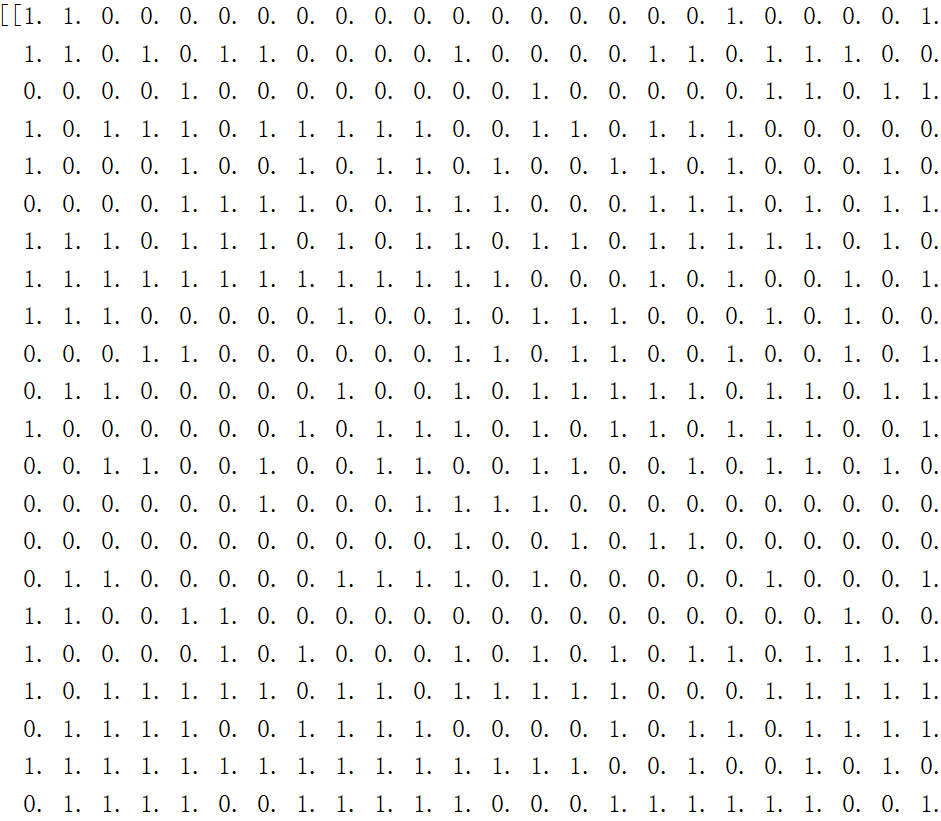
|  |
| --- |
| def predictTree(Trees,dataSet,alpha="huigui"):  m=len(dataSet)  yhat=np.mat(zeros((m,1)))  for tree in Trees:  yhat+=createForeCast(tree,dataSet,alpha)  if alpha=="huigui": yhat/=len(Trees)  else:  for i in range(len(yhat)):  if yhat[i,0]>len(Trees)/2:  yhat[i,0]=1  else:  yhat[i,0]=0  return yhat |

结果分析：

测试集真实类别为：



预测结果为：



统计错误率：

