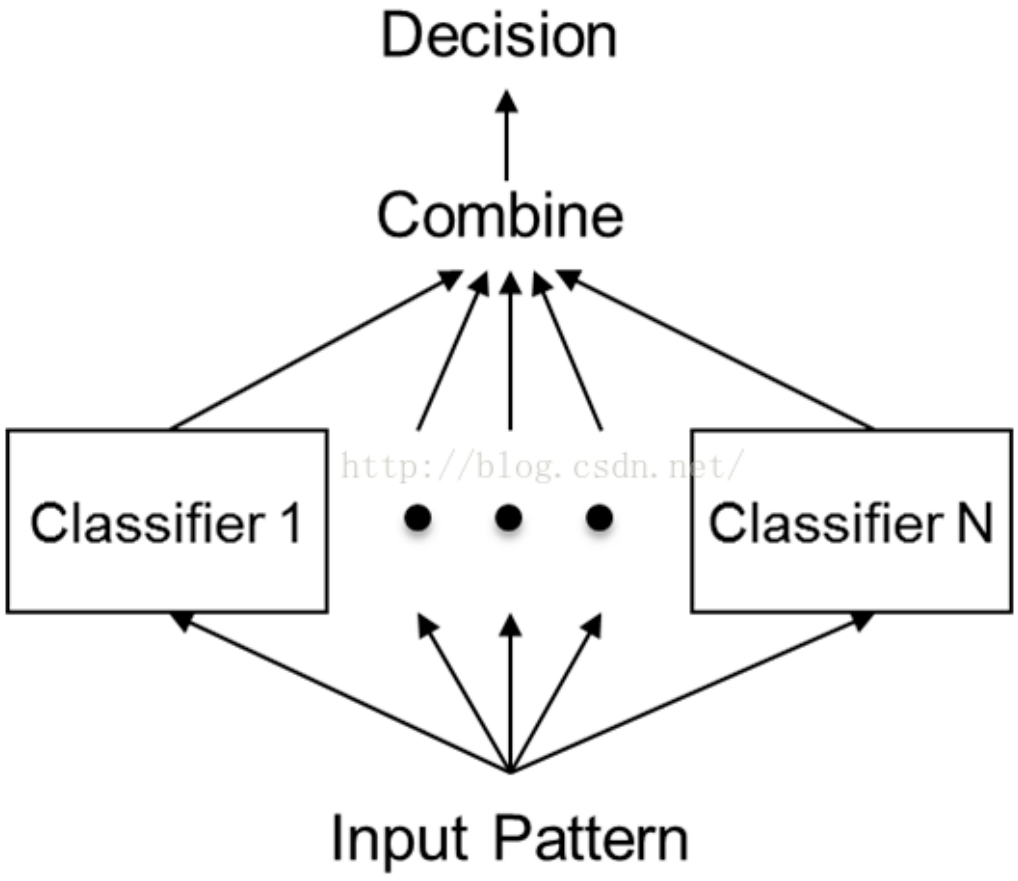
一.随机森林算法实现

1.概念

(1) 集成学习

集成学习是使用一系列学习器进行学习，并使用某种规则把各个学习结果进行整合从而获得比单个学习器更好的学习效果的一种机器学习方法。集成学习的方法都是指的同质个体学习器。而同质个体学习器使用最多的模型是 CART 决策树和神经网络。同质个体学习器按照个体学习器之间是否存在依赖关系可以分为两类，第一个是个体学习器之间存在强依赖关系，一系列个体学习器基本都需要串行生成，代表算法是 boosting 系列算法，第二个是个体学习器之间不存在强依赖关系，一系列个体学习器可以并行生成，代表算法是 bagging 和随机森林（Random Forest）系列算法。如下图所示



(2) 随机森林

是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。随机森林在决策树的训练过程中引入了随机属性选择，随机森林基学习器的多样性不仅来自样本扰动，还来自属性扰动，这就使的最终形成的集成学习模型的泛化性能较强。

2.原理

假设有 k 个特征，每次随机选取 sqrt(k)个特征来训练模型，当随机选取 10 次之后，就有 10 个模型生成，然后用 10 个模型分别对测试集进行测试，如果是回归问题，就对 10 个模型的预测值取个平均值，如果是分类问题，就对 10 个模型的预测值进行投票，选取票数最多的那个作为预测值。

3.程序源码

(1) 数据集的加载:

```
#数据集和训练集的加载
def loadDataSet(trainFileName,testFileName):
    trainDataSet = pd.read_csv(trainFileName,header=None,sep=",")
    testDataSet = pd.read_csv(testFileName,header=None,sep=",")
    return trainDataSet,testDataSet
```

(2) 随机选取特征，训练多个模型

```
def generateMultiModel(dataSet):
    #模型数目
    modelNums = 10

    # 特征总数
    totalFeature = len(dataSet.columns) - 1

    # 每次选取的特征的数目,四舍五入做个处理
    featNums = round(m.sqrt(totalFeature))

    #用矩阵存储每次选取的特征的位置
    randomFeatMat = np.zeros((modelNums,featNums),dtype=int)

    #模型列表
    modelList = []

    for i in range(modelNums):
        ranLsit = rd.sample(range(0,6),featNums)
        for j in range(featNums):
            randomFeatMat[i,j] = ranLsit[j]
```

```
    tempList = []
    #依次循环训练出10个模型
    for i in range(modelNums):
        tempList = []
        for j in range(featNums):
            dataSet[randomFeatMat[i,j]]
            tempList.append(dataSet[randomFeatMat[i,j]])
        selectedDataset = np.mat(tempList).T
        labelSet = np.mat(dataSet[dataSet.columns[-1]]).T
        models = decisionTreeModel(selectedDataset,labelSet)
        modelList.append(models)

    #返回每次选择的随机特征子集和该特征子集训练出来的模型集合
    return randomFeatMat,modelList
```

(3) 用训练生成的模型对测试集进行测试，并且返回测试结果

```
def testPerformance(modelList,randomFeatMat,testDataSet):

    dataSetBak = testDataSet.copy()

    #去除label标签之后的数据
    dataSetMat = np.mat(testDataSet[testDataSet.columns[:-1]])

    #预测数值列表
    predValueList = []

    # 取得每一个预测样本的实际数值列表
    actualLableSet = np.mat(testDataSet[testDataSet.columns[-1]]).T

    #测试样本总数
    sampleLen = len(testDataSet)
    tempList = []
    sums = 0
```

```
sums = 0
#循环所有的模型，对每一个样本进行预测，循环完之后对预测结果取一个均值
for rows in range(sampleLen):

    sums = 0
    for i in range(len(modelList)):
        tempList = []
        for ele in range(randomFeatMat.shape[1]):
            colums = randomFeatMat[i,ele]
            tempList.append(dataSetMat[rows,colums])
        tempSet = np.mat(tempList)
        result = modelList[i].predict(tempSet)
        sums += result
    tmpValue = sums / len(modelList)
    # print(tmpValue)
    predValueList.append(tmpValue)
#测试集合预测集组成的矩阵为
finalMatrix = np.hstack((np.mat(predValueList),actualLableSet))
return finalMatrix
```

(4) 求解随机森林的均方误差(因为是回归问题)

```
#均方误差评价回归性能
def evaluateIndex(dataSet):
    sums = 0
    row,column = dataSet.shape
    for i in range(row):
        sums += (dataSet[i,0] - dataSet[i,1])**2
    return sums / row
```

(5) 训练单个模型(决策树)

```
"""
   1.用单模型对数据集建模，然后测试，
   2.和多模型进行比较
"""
def singleMode(trainDataSet):
    #数据集
    dataSet = np.mat(trainDataSet[trainDataSet.columns[:-1]])
    #标签集
    labelSet = np.mat(trainDataSet[trainDataSet.columns[-1]]).T

    #用决策树建模
    models = decisionTreeModel(dataSet,labelSet)
    return models
```

(6) 测试单决策树的性能

```
#单模型在测试集上的测试
def testSingleModel(testDataSet,model):
    #预测标记值
    predictValue = []
    #真实标记值
    actualValue = np.mat(testDataSet[testDataSet.columns[-1]]).T

    dataSet = np.mat(testDataSet[testDataSet.columns[:-1]])
    for row in range(dataSet.shape[0]):
        value = model.predict(dataSet[row])
        predictValue.append(value)
    a = np.mat(predictValue)
    resutlMatrix = np.hstack((a,actualValue))
    return resutlMatrix
```

(7) 求解单模型均方误差

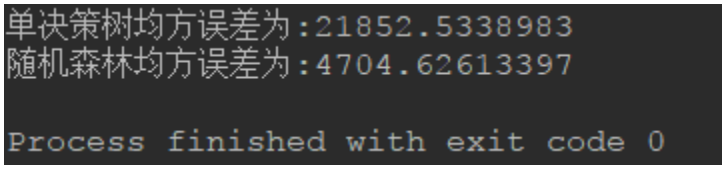
```
#均方误差评价回归性能
def evaluateIndex(dataSet):
    sums = 0
    row,column = dataSet.shape
    for i in range(row):
        sums += (dataSet[i,0] - dataSet[i,1])**2
    return sums / row
```

(8) 程序主函数如下

```
trainFilePath = "D://WeKaDataSet//cpuTrain.csv"
testFilePath = "D://WeKaDataSet//cpuTest.csv"

# 返回训练集和测试集
trainDataSet, testDataSet = loadDataSet(trainFilePath, testFilePath)

trainData = trainDataSet.copy()
testData = testDataSet.copy()

# 训练单模型
singleModel = singleMode(trainData)
# 对单模型性能进行测试
resultSingleMat = testSingleModel(testData, singleModel)

# 返回训练好的模型的集合(随机森林)
randomFeatMat, modelList = generateMultiModel(trainDataSet)
# 在测试样本上测试模型的好坏
resultMatrix = testPerformance(modelList, randomFeatMat, testDataSet)
# 对测试结果进行评价(多模型均方误差)
mse = evaluateIndex(resultMatrix)
# 对测试结果进行评价(单模型均方误差)
mseSingle = evaluateIndex(resultSingleMat)

print("单决策树均方误差为:" + str(mseSingle))
print("随机森林均方误差为:" + str(mse))
```

4.程序执行结果

程序最终执行结果如下图所示



5.总结

运用多模型对数据进行预测,性能比单模型提高不少,但是由于随机森林每次都是随机选取 sqrt(k)个特征来训练模型，导致最终程序的均方误差起伏较大。运用随机森林可以降低方差。

二. 基于线性回归的融合模型算法实现

1.原理

stacking 模型融合，首先从初始训练集训练出初级学习器，把初级学习器的输出当做次级训练集的输入特征，来训练次级学习器,初始样本的标记仍被当做样例标记。Stacking 算法描述如下图所示:

输入: 训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
初级学习算法 $\mathfrak{L}_1,\mathfrak{L}_2,\ldots,\mathfrak{L}_T$;
次级学习算法 $\mathfrak{L}$.
过程:
1: **for** $t=1,2,\ldots,T$ **do**
2: $h_t=\mathfrak{L}_t(D)$;
3: **end for**
4: $D'=\varnothing$;
5: **for** $i=1,2,\ldots,m$ **do**
6: **for** $t=1,2,\ldots,T$ **do**
7: $z_{it}=h_t(\boldsymbol{x}_i)$;
8: **end for**
9: $D'=D'\cup((z_{i1},z_{i2},\ldots,z_{iT}),y_i)$;
10: **end for**
11: $h'=\mathfrak{L}(D')$;
输出: $H(\boldsymbol{x})=h'(h_1(\boldsymbol{x}),h_2(\boldsymbol{x}),\ldots,h_T(\boldsymbol{x}))$

2.程序源码

(1) 数据集的加载

```
def loadDataSet(trainFilePath,testFilePath):
    trainDataSet = pd.read_csv(trainFilePath,header=None,sep=",")
    testDataSet = pd.read_csv(testFilePath, header=None, sep=",")
    return trainDataSet,testDataSet
```

(2) 训练多个模型(通过改变算法参数和数据) 并且对训练集进行交叉预测以产生次级训练集。

```
def generateMuitiModel(trainDataSet):
    #交叉验证预测数值
    predValue = []
    #存放训练好的模型
    modelList = []
    trainDataSetBak = trainDataSet.copy()
    #取出不包括label项的所有数据
    dataSet = np.mat(trainDataSet[trainDataSet.columns[:-1]])

    #取出label这一列的数据
    labelSet = np.mat(trainDataSet[trainDataSet.columns[-1]].values.tolist()).T
    #训练出9个模型
    for i in range(3):
        model,algri = decisionTree(i+3,dataSet,labelSet)
        modelList.append(model)
        trainPredValue = cross_val_predict(algri, dataSet, labelSet, cv=5)
        predValue.append(trainPredValue)

        model,algri = bayesianRidge(300+i,dataSet,labelSet)
        modelList.append(model)
        trainPredValue = cross_val_predict(algri, dataSet, labelSet, cv=5)
        predValue.append(trainPredValue)
```

```
    model,algri = svmAlgrith((i / 10)+0.1,dataSet,labelSet)
    modelList.append(model)
    trainPredValue = cross_val_predict(algri, dataSet, labelSet, cv=5)
    predValue.append(trainPredValue)
tmpMat = np.mat(predValue).T
#次级学习器的训练集
secondTrainData = np.hstack((dataSet,tmpMat,labelSet))
return modelList,secondTrainData
```

(3) 形成次级学习器的测试集

```
"""
1.通过训练好的初级学习器对测试集进行预测
2.将预测值当做新的特征，形成新的数据集(次级学习器的测试集)
3.新形成的数据集的标签还是原来测试集的标签
"""
def generateNewTestData(modelList,testDataSet):
    #产生除label的数据集
    dataSet = np.mat(testDataSet[testDataSet.columns[:-1]])
    #label集
    labelSet = np.mat(testDataSet[testDataSet.columns[-1]]).T
    #存放预测值的集合
    predictValue = []
    rows,columns = dataSet.shape
    for row in range(rows):
        featValue = dataSet[row]
        tempList = []
        for model in modelList:
            values = model.predict(featValue)
            tempList.append(values[0])
        predictValue.append(tempList)
    resutlMat = np.mat(predictValue)
    #原来特征+预测值+标签  形成新的数据集(是次级学习器的测试集)
    secondTestData = np.hstack((dataSet,resutlMat,labelSet))
    return secondTestData,np.mat(np.hstack((resutlMat,labelSet)))
```

(4) 训练次级学习器，并且用次级测试集测试，然后返回测试结果

```
#训练次级学习器，并且用次级测试集测试，然后返回测试结果
def generateSecondLearn(secondTrainData,secondTestData):
    #产生除label的数据集
    dataSet = secondTrainData[:,:-1]

    #label集
    labelSet = secondTrainData[:,-1]
    reg = linear_model.LinearRegression()
    modelLR = reg.fit(dataSet, labelSet)

    #次级测试集的真实标记
    labset = secondTestData[:,-1]
    # 次级测试集除label以外的数据
    testData = secondTestData[:,:-1]
    #次级测试集的预测标记
    predValue = modelLR.predict(testData)

    secondResMat = np.hstack((labset,predValue))
    return secondResMat
```

(5) 计算次级学习器(融合模型)均方误差

```python
#次级学习器均方误差评价融合之后模型的回归性能
def evaluateIndex(dataSet):
    sums = 0
    row,column = dataSet.shape
    for i in range(row):
        sums += (dataSet[i,0] - dataSet[i,1])**2
    return sums / row
```

(6) 计算 9 个单模型的均方误差

```python
#初级学习器均方误差评价单个模型的回归性能
def evalSingModeMse(dataSet):
    #9个模型的预测结果+测试集的真实标签
    mseList = []
    rows,columns = dataSet.shape
    label = dataSet[:,-1]
    for col in range(columns-1):
        tmpValue = evaluateIndex(np.hstack((dataSet[:,col],label)))
        mseList.append(tmpValue)
    return mseList
```

3.程序最终执行结果(总共 9 个单模型)

```
融合线性回归的均方误差为:
 17166.9145877


9个单模型的均方误差分别为:
 [[ 27667.48867629]
 [ 15044.34361358]
 [ 54508.42107947]
 [ 22360.22130944]
 [ 15044.34361358]
 [ 54512.51055497]
 [ 21879.75254782]
 [ 15044.34361358]
 [ 54500.99274863]]
```

4.总结

当生成次级训练集的时候，不能直接用训练初级学习器的集合直接生成次级训练集，必须对初级训练集进行交叉预测，以防止过拟合现象的出现。由于 stack 模型性能可能不是太好，所以有可能单个学习的性能要优于融合之后的学习器，但是若 BaseModel 属于弱学习器，则融合之后的学习器的性能要好于单个学习器的性能。