机器学习实战

2 K-近邻算法

2.1 kNN.py

```
'''
Created on Sep 16, 2010
kNN: k Nearest Neighbors

Input:      inX: vector to compare to existing dataset (1xN)
            dataSet: size m data set of known vectors (NxM)
            labels: data set labels (1xM vector)
            k: number of neighbors to use for comparison (should be an odd number)

Output:     the most popular class label

@author: pbharrin
'''
from numpy import *
import operator
from os import listdir

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

def createDataSet():
    group = array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
    labels = ['A','A','B','B']
    return group, labels

def file2matrix(filename):
    fr = open(filename)
    numberOfLines = len(fr.readlines())         #get the number of lines in the file
    returnMat = zeros((numberOfLines,3))        #prepare matrix to return
    classLabelVector = []                       #prepare labels return
    fr = open(filename)
```

```
    index = 0
    for line in fr.readlines():
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:3]
        classLabelVector.append(int(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector

def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - tile(minVals, (m,1))
    normDataSet = normDataSet/tile(ranges, (m,1))   #element wise divide
    return normDataSet, ranges, minVals

def datingClassTest():
    hoRatio = 0.50      #hold out 10%
    datingDataMat,datingLabels = file2matrix('datingTestSet2.txt')      #load data setfrom
file
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult =
classify0(normMat[i,:],normMat[numTestVecs:m,:],datingLabels[numTestVecs:m],3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierResult,
datingLabels[i])
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print "the total error rate is: %f" % (errorCount/float(numTestVecs))
    print errorCount

def img2vector(filename):
    returnVect = zeros((1,1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect
```

```
def handwritingClassTest():
    hwLabels = []
    trainingFileList = listdir('trainingDigits')           #load the training set
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]     #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
    testFileList = listdir('testDigits')        #iterate through the test set
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]     #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print "the classifier came back with: %d, the real answer is: %d" % (classifierResult,
classNumStr)
        if (classifierResult != classNumStr): errorCount += 1.0
    print "\nthe total number of errors is: %d" % errorCount
    print "\nthe total error rate is: %f" % (errorCount/float(mTest))
```

2.2 createDist.py

```
    '''
    Created on Oct 6, 2010

    @author: Peter
    '''
    from numpy import *
    import matplotlib
    import matplotlib.pyplot as plt
    from matplotlib.patches import Rectangle



    n = 1000 #number of points to create
    xcord = zeros((n))
    ycord = zeros((n))
    markers =[]
    colors =[]
    fw = open('testSet.txt','w')
```

```
for i in range(n):
    [r0,r1] = random.standard_normal(2)
    myClass = random.uniform(0,1)
    if (myClass <= 0.16):
        fFlyer = random.uniform(22000, 60000)
        tats = 3 + 1.6*r1
        markers.append(20)
        colors.append(2.1)
        classLabel = 1 #'didntLike'
        print ("%d, %f, class1") % (fFlyer, tats)
    elif ((myClass > 0.16) and (myClass <= 0.33)):
        fFlyer = 6000*r0 + 70000
        tats = 10 + 3*r1 + 2*r0
        markers.append(20)
        colors.append(1.1)
        classLabel = 1 #'didntLike'
        print ("%d, %f, class1") % (fFlyer, tats)
    elif ((myClass > 0.33) and (myClass <= 0.66)):
        fFlyer = 5000*r0 + 10000
        tats = 3 + 2.8*r1
        markers.append(30)
        colors.append(1.1)
        classLabel = 2 #'smallDoses'
        print ("%d, %f, class2") % (fFlyer, tats)
    else:
        fFlyer = 10000*r0 + 35000
        tats = 10 + 2.0*r1
        markers.append(50)
        colors.append(0.1)
        classLabel = 3 #'largeDoses'
        print ("%d, %f, class3") % (fFlyer, tats)
    if (tats < 0): tats =0
    if (fFlyer < 0): fFlyer =0
    xcord[i] = fFlyer; ycord[i]=tats
    fw.write("%d\t%f\t%f\t%d\n" % (fFlyer, tats, random.uniform(0.0, 1.7), classLabel))

fw.close()
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xcord,ycord, c=colors, s=markers)
type1 = ax.scatter([-10], [-10], s=20, c='red')
type2 = ax.scatter([-10], [-15], s=30, c='green')
type3 = ax.scatter([-10], [-20], s=50, c='blue')
ax.legend([type1, type2, type3], ["Class 1", "Class 2", "Class 3"], loc=2)
```

```
    #ax.axis([-5000,100000,-2,25])
    plt.xlabel('Frequent Flyier Miles Earned Per Year')
    plt.ylabel('Percentage of Body Covered By Tatoos')
    plt.show()
```

2.3 createDist2.py

```
'''
Created on Oct 6, 2010

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle


n = 1000 #number of points to create
xcord1 = []; ycord1 = []
xcord2 = []; ycord2 = []
xcord3 = []; ycord3 = []
markers =[]
colors =[]
fw = open('testSet.txt','w')
for i in range(n):
    [r0,r1] = random.standard_normal(2)
    myClass = random.uniform(0,1)
    if (myClass <= 0.16):
        fFlyer = random.uniform(22000, 60000)
        tats = 3 + 1.6*r1
        markers.append(20)
        colors.append(2.1)
        classLabel = 1 #'didntLike'
        xcord1.append(fFlyer); ycord1.append(tats)
    elif ((myClass > 0.16) and (myClass <= 0.33)):
        fFlyer = 6000*r0 + 70000
        tats = 10 + 3*r1 + 2*r0
        markers.append(20)
        colors.append(1.1)
        classLabel = 1 #'didntLike'
        if (tats < 0): tats =0
        if (fFlyer < 0): fFlyer =0
        xcord1.append(fFlyer); ycord1.append(tats)
    elif ((myClass > 0.33) and (myClass <= 0.66)):
        fFlyer = 5000*r0 + 10000
```

```
            tats = 3 + 2.8*r1
            markers.append(30)
            colors.append(1.1)
            classLabel = 2 #'smallDoses'
            if (tats < 0): tats =0
            if (fFlyer < 0): fFlyer =0
            xcord2.append(fFlyer); ycord2.append(tats)
        else:
            fFlyer = 10000*r0 + 35000
            tats = 10 + 2.0*r1
            markers.append(50)
            colors.append(0.1)
            classLabel = 3 #'largeDoses'
            if (tats < 0): tats =0
            if (fFlyer < 0): fFlyer =0
            xcord3.append(fFlyer); ycord3.append(tats)

fw.close()
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.scatter(xcord,ycord, c=colors, s=markers)
type1 = ax.scatter(xcord1, ycord1, s=20, c='red')
type2 = ax.scatter(xcord2, ycord2, s=30, c='green')
type3 = ax.scatter(xcord3, ycord3, s=50, c='blue')
ax.legend([type1, type2, type3], ["Did Not Like", "Liked in Small Doses", "Liked in Large Doses"], loc=2)
ax.axis([-5000,100000,-2,25])
plt.xlabel('Frequent Flyier Miles Earned Per Year')
plt.ylabel('Percentage of Time Spent Playing Video Games')
plt.show()
```

2.4 createFirstPlot.py

```
'''
Created on Oct 27, 2010

@author: Peter
'''
from numpy import *
import kNN
import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
datingDataMat,datingLabels = kNN.file2matrix('datingTestSet.txt')
#ax.scatter(datingDataMat[:,1], datingDataMat[:,2])
```

```
ax.scatter(datingDataMat[:,1], datingDataMat[:,2], 15.0*array(datingLabels),
15.0*array(datingLabels))
ax.axis([-2,25,-0.2,2.0])
plt.xlabel('Percentage of Time Spent Playing Video Games')
plt.ylabel('Liters of Ice Cream Consumed Per Week')
plt.show()
```

3 决策树

3.1 treePlotter.py

```
'''
Created on Oct 14, 2010

@author: Peter Harrington
'''
import matplotlib.pyplot as plt

decisionNode = dict(boxstyle="sawtooth", fc="0.8")
leafNode = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")

def getNumLeafs(myTree):
    numLeafs = 0
    firstStr = myTree.keys()[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are dictonaires, if
not they are leaf nodes
            numLeafs += getNumLeafs(secondDict[key])
        else:   numLeafs +=1
    return numLeafs

def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = myTree.keys()[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are dictonaires, if
not they are leaf nodes
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth
    return maxDepth

def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(nodeTxt, xy=parentPt,  xycoords='axes fraction',
```

```
                    xytext=centerPt, textcoords='axes fraction',
                    va="center", ha="center", bbox=nodeType, arrowprops=arrow_args )

def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center", rotation=30)

def plotTree(myTree, parentPt, nodeTxt):#if the first key tells you what feat was split on
    numLeafs = getNumLeafs(myTree)    #this determines the x width of this tree
    depth = getTreeDepth(myTree)
    firstStr = myTree.keys()[0]          #the text label for this node should be this
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW, plotTree.yOff)
    plotMidText(cntrPt, parentPt, nodeTxt)
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are dictonaires, if
not they are leaf nodes
            plotTree(secondDict[key],cntrPt,str(key))              #recursion
        else:    #it's a leaf node print the leaf node
            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD
#if you do get a dictonary you know it's a tree, and the first element will be another dict

def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)       #no ticks
    #createPlot.ax1 = plt.subplot(111, frameon=False) #ticks for demo puropses
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0;
    plotTree(inTree, (0.5,1.0), '')
    plt.show()

#def createPlot():
#    fig = plt.figure(1, facecolor='white')
#    fig.clf()
#    createPlot.ax1 = plt.subplot(111, frameon=False) #ticks for demo puropses
```

```
#       plotNode('a decision node', (0.5, 0.1), (0.1, 0.5), decisionNode)
#       plotNode('a leaf node', (0.8, 0.1), (0.3, 0.8), leafNode)
#       plt.show()

def retrieveTree(i):
    listOfTrees =[{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}},
                  {'no surfacing': {0: 'no', 1: {'flippers': {0: {'head': {0: 'no', 1: 'yes'}}, 1:
'no'}}}}
                  ]
    return listOfTrees[i]

#createPlot(thisTree)
```

3.2 trees.py

```
'''
Created on Oct 12, 2010
Decision Tree Source Code for Machine Learning in Action Ch. 3
@author: Peter Harrington
'''
from math import log
import operator

def createDataSet():
    dataSet = [[1, 1, 'yes'],
               [1, 1, 'yes'],
               [1, 0, 'no'],
               [0, 1, 'no'],
               [0, 1, 'no']]
    labels = ['no surfacing','flippers']
    #change to discrete values
    return dataSet, labels

def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet: #the the number of unique elements and their occurance
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys(): labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2) #log base 2
    return shannonEnt
```

```
def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]         #chop out axis used for splitting
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet

def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1           #the last column is used for the labels
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):                #iterate over all the features
        featList = [example[i] for example in dataSet]#create a list of all the examples of this
feature
        uniqueVals = set(featList)              #get a set of unique values
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy         #calculate the info gain; ie reduction in
entropy
        if (infoGain > bestInfoGain):           #compare this to the best gain so far
            bestInfoGain = infoGain                 #if better than current best, set to best
            bestFeature = i
    return bestFeature                              #returns an integer

def majorityCnt(classList):
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]#stop splitting when all of the classes are equal
    if len(dataSet[0]) == 1: #stop splitting when there are no more features in dataSet
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
```

```
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]       #copy all of labels, so trees don't mess up existing labels
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat,
value),subLabels)
    return myTree

def classify(inputTree,featLabels,testVec):
    firstStr = inputTree.keys()[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else: classLabel = valueOfFeat
    return classLabel

def storeTree(inputTree,filename):
    import pickle
    fw = open(filename,'w')
    pickle.dump(inputTree,fw)
    fw.close()

def grabTree(filename):
    import pickle
    fr = open(filename)
    return pickle.load(fr)
```

4 基于概率论的分类方法：朴素贝叶斯

4.1 bayes.py

```
'''
Created on Oct 19, 2010

@author: Peter
'''
from numpy import *

def loadDataSet():
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                 ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
```

```
                    ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                    ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                    ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
                    ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0,1,0,1,0,1]        #1 is abusive, 0 not
    return postingList,classVec

def createVocabList(dataSet):
    vocabSet = set([])  #create empty set
    for document in dataSet:
        vocabSet = vocabSet | set(document) #union of the two sets
    return list(vocabSet)

def setOfWords2Vec(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else: print "the word: %s is not in my Vocabulary!" % word
    return returnVec

def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)
    p0Num = ones(numWords); p1Num = ones(numWords)      #change to ones()
    p0Denom = 2.0; p1Denom = 2.0                        #change to 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = log(p1Num/p1Denom)          #change to log()
    p0Vect = log(p0Num/p0Denom)          #change to log()
    return p0Vect,p1Vect,pAbusive

def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)    #element-wise mult
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
```

```
        return 0

def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec

def testingNB():
    listOPosts,listClasses = loadDataSet()
    myVocabList = createVocabList(listOPosts)
    trainMat=[]
    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
    p0V,p1V,pAb = trainNB0(array(trainMat),array(listClasses))
    testEntry = ['love', 'my', 'dalmation']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb)
    testEntry = ['stupid', 'garbage']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb)

def textParse(bigString):       #input is big string, #output is word list
    import re
    listOfTokens = re.split(r'\W*', bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]

def spamTest():
    docList=[]; classList = []; fullText =[]
    for i in range(1,26):
        wordList = textParse(open('email/spam/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)#create vocabulary
    trainingSet = range(50); testSet=[]                   #create test set
    for i in range(10):
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
```

```
            del(trainingSet[randIndex])
        trainMat=[]; trainClasses = []
        for docIndex in trainingSet:#train the classifier (get probs) trainNB0
            trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
            trainClasses.append(classList[docIndex])
        p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
        errorCount = 0
        for docIndex in testSet:                #classify the remaining items
            wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
            if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
                errorCount += 1
                print "classification error",docList[docIndex]
        print 'the error rate is: ',float(errorCount)/len(testSet)
        #return vocabList,fullText

def calcMostFreq(vocabList,fullText):
        import operator
        freqDict = {}
        for token in vocabList:
            freqDict[token]=fullText.count(token)
        sortedFreq = sorted(freqDict.iteritems(), key=operator.itemgetter(1), reverse=True)
        return sortedFreq[:30]

def localWords(feed1,feed0):
        import feedparser
        docList=[]; classList = []; fullText =[]
        minLen = min(len(feed1['entries']),len(feed0['entries']))
        for i in range(minLen):
            wordList = textParse(feed1['entries'][i]['summary'])
            docList.append(wordList)
            fullText.extend(wordList)
            classList.append(1) #NY is class 1
            wordList = textParse(feed0['entries'][i]['summary'])
            docList.append(wordList)
            fullText.extend(wordList)
            classList.append(0)
        vocabList = createVocabList(docList)#create vocabulary
        top30Words = calcMostFreq(vocabList,fullText)      #remove top 30 words
        for pairW in top30Words:
            if pairW[0] in vocabList: vocabList.remove(pairW[0])
        trainingSet = range(2*minLen); testSet=[]                   #create test set
        for i in range(20):
            randIndex = int(random.uniform(0,len(trainingSet)))
            testSet.append(trainingSet[randIndex])
```

```
            del(trainingSet[randIndex])
        trainMat=[]; trainClasses = []
        for docIndex in trainingSet:#train the classifier (get probs) trainNB0
            trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
            trainClasses.append(classList[docIndex])
        p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
        errorCount = 0
        for docIndex in testSet:                #classify the remaining items
            wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
            if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
                errorCount += 1
        print 'the error rate is: ',float(errorCount)/len(testSet)
        return vocabList,p0V,p1V

def getTopWords(ny,sf):
        import operator
        vocabList,p0V,p1V=localWords(ny,sf)
        topNY=[]; topSF=[]
        for i in range(len(p0V)):
            if p0V[i] > -6.0 : topSF.append((vocabList[i],p0V[i]))
            if p1V[i] > -6.0 : topNY.append((vocabList[i],p1V[i]))
        sortedSF = sorted(topSF, key=lambda pair: pair[1], reverse=True)
        print "SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**"
        for item in sortedSF:
            print item[0]
        sortedNY = sorted(topNY, key=lambda pair: pair[1], reverse=True)
        print
"NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**
"
        for item in sortedNY:
            print item[0]
```

4.2 create2Normal.py

```
'''
Created on Oct 6, 2010

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt

n = 1000 #number of points to create
xcord0 = []
ycord0 = []
```

```
xcord1 = []
ycord1 = []
markers =[]
colors =[]
fw = open('testSet.txt','w')
for i in range(n):
    [r0,r1] = random.standard_normal(2)
    myClass = random.uniform(0,1)
    if (myClass <= 0.5):
        fFlyer = r0 + 9.0
        tats = 1.0*r1 + fFlyer - 9.0
        xcord0.append(fFlyer)
        ycord0.append(tats)
    else:
        fFlyer = r0 + 2.0
        tats = r1+fFlyer - 2.0
        xcord1.append(fFlyer)
        ycord1.append(tats)
    #fw.write("%f\t%f\t%d\n" % (fFlyer, tats, classLabel))

fw.close()
fig = plt.figure()
ax = fig.add_subplot(111)
#ax.scatter(xcord,ycord, c=colors, s=markers)
ax.scatter(xcord0,ycord0, marker='^', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
plt.plot([0,1], label='going up')
plt.show()
```

4.3 monoDemo.py

```
'''
Created on Oct 6, 2010
Shows montonocity of a function and the log of that function
@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt

t = arange(0.0, 0.5, 0.01)
s = sin(2*pi*t)
logS = log(s)

fig = plt.figure()
ax = fig.add_subplot(211)
```

```
ax.plot(t,s)
ax.set_ylabel('f(x)')
ax.set_xlabel('x')

ax = fig.add_subplot(212)
ax.plot(t,logS)
ax.set_ylabel('ln(f(x))')
ax.set_xlabel('x')
plt.show()
```

5 Logistic 回归

5.1 logRegres.py

```
'''
Created on Oct 27, 2010
Logistic Regression Working Module
@author: Peter
'''
from numpy import *

def loadDataSet():
    dataMat = []; labelMat = []
    fr = open('testSet.txt')
    for line in fr.readlines():
        lineArr = line.strip().split()
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])
        labelMat.append(int(lineArr[2]))
    return dataMat,labelMat

def sigmoid(inX):
    return 1.0/(1+exp(-inX))

def gradAscent(dataMatIn, classLabels):
    dataMatrix = mat(dataMatIn)                 #convert to NumPy matrix
    labelMat = mat(classLabels).transpose() #convert to NumPy matrix
    m,n = shape(dataMatrix)
    alpha = 0.001
    maxCycles = 500
    weights = ones((n,1))
    for k in range(maxCycles):                  #heavy on matrix operations
        h = sigmoid(dataMatrix*weights)     #matrix mult
        error = (labelMat - h)              #vector subtraction
        weights = weights + alpha * dataMatrix.transpose()* error #matrix mult
    return weights

def plotBestFit(weights):
```

```
    import matplotlib.pyplot as plt
    dataMat,labelMat=loadDataSet()
    dataArr = array(dataMat)
    n = shape(dataArr)[0]
    xcord1 = []; ycord1 = []
    xcord2 = []; ycord2 = []
    for i in range(n):
        if int(labelMat[i])== 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')
    ax.scatter(xcord2, ycord2, s=30, c='green')
    x = arange(-3.0, 3.0, 0.1)
    y = (-weights[0]-weights[1]*x)/weights[2]
    ax.plot(x, y)
    plt.xlabel('X1'); plt.ylabel('X2');
    plt.show()

def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.01
    weights = ones(n)    #initialize to all ones
    for i in range(m):
        h = sigmoid(sum(dataMatrix[i]*weights))
        error = classLabels[i] - h
        weights = weights + alpha * error * dataMatrix[i]
    return weights

def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = shape(dataMatrix)
    weights = ones(n)    #initialize to all ones
    for j in range(numIter):
        dataIndex = range(m)
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.0001    #apha decreases with iteration, does not
            randIndex = int(random.uniform(0,len(dataIndex)))#go to 0 because of the constant
            h = sigmoid(sum(dataMatrix[randIndex]*weights))
            error = classLabels[randIndex] - h
            weights = weights + alpha * error * dataMatrix[randIndex]
            del(dataIndex[randIndex])
    return weights
```

```
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5: return 1.0
    else: return 0.0

def colicTest():
    frTrain = open('horseColicTraining.txt'); frTest = open('horseColicTest.txt')
    trainingSet = []; trainingLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(21):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    trainWeights = stocGradAscent1(array(trainingSet), trainingLabels, 1000)
    errorCount = 0; numTestVec = 0.0
    for line in frTest.readlines():
        numTestVec += 1.0
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(21):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(array(lineArr), trainWeights))!= int(currLine[21]):
            errorCount += 1
    errorRate = (float(errorCount)/numTestVec)
    print "the error rate of this test is: %f" % errorRate
    return errorRate

def multiTest():
    numTests = 10; errorSum=0.0
    for k in range(numTests):
        errorSum += colicTest()
    print "after %d iterations the average error rate is: %f" % (numTests,
errorSum/float(numTests))
```

5.2 plot2D.py

```
'''
Created on Oct 6, 2010

@author: Peter
'''
from numpy import *
import matplotlib
```

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import logRegres

dataMat,labelMat=logRegres.loadDataSet()
dataArr = array(dataMat)
weights = logRegres.stocGradAscent0(dataArr,labelMat)

n = shape(dataArr)[0] #number of points to create
xcord1 = []; ycord1 = []
xcord2 = []; ycord2 = []

markers =[]
colors =[]
for i in range(n):
	if int(labelMat[i])== 1:
		xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])
	else:
		xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])

fig = plt.figure()
ax = fig.add_subplot(111)
#ax.scatter(xcord,ycord, c=colors, s=markers)
type1 = ax.scatter(xcord1, ycord1, s=30, c='red', marker='s')
type2 = ax.scatter(xcord2, ycord2, s=30, c='green')
x = arange(-3.0, 3.0, 0.1)
#weights = [-2.9, 0.72, 1.29]
#weights = [-5, 1.09, 1.42]
weights = [13.03822793,	1.32877317,	-1.96702074]
weights = [4.12,	0.48,	-0.6168]
y = (-weights[0]-weights[1]*x)/weights[2]
type3 = ax.plot(x, y)
#ax.legend([type1, type2, type3], ["Did Not Like", "Liked in Small Doses", "Liked in Large Doses"], loc=2)
#ax.axis([-5000,100000,-2,25])
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```

5.3 plotGD.py

```
'''
Created on Oct 28, 2010

@author: Peter
'''
```

```
import matplotlib
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

leafNode = dict(boxstyle="round4", fc="0.8")
arrow_args = dict(arrowstyle="<-")

matplotlib.rcParams['xtick.direction'] = 'out'
matplotlib.rcParams['ytick.direction'] = 'out'

delta = 0.025
x = np.arange(-2.0, 2.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = -((X-1)**2)
Z2 = -(Y**2)
#Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
#Z2 = mlab.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
# difference of Gaussians
Z = 1.0 * (Z2 + Z1)+5.0

# Create a simple contour plot with labels using default colors.    The
# inline argument to clabel will control whether the labels are draw
# over the line segments of the contour, removing the lines beneath
# the label
plt.figure()
CS = plt.contour(X, Y, Z)
plt.annotate('', xy=(0.05, 0.05),    xycoords='axes fraction',
                    xytext=(0.2,0.2), textcoords='axes fraction',
                    va="center", ha="center", bbox=leafNode, arrowprops=arrow_args )
plt.text(-1.9, -1.8, 'P0')
plt.annotate('', xy=(0.2,0.2),    xycoords='axes fraction',
                    xytext=(0.35,0.3), textcoords='axes fraction',
                    va="center", ha="center", bbox=leafNode, arrowprops=arrow_args )
plt.text(-1.35, -1.23, 'P1')
plt.annotate('', xy=(0.35,0.3),    xycoords='axes fraction',
                    xytext=(0.45,0.35), textcoords='axes fraction',
                    va="center", ha="center", bbox=leafNode, arrowprops=arrow_args )
plt.text(-0.7, -0.8, 'P2')
plt.text(-0.3, -0.6, 'P3')
plt.clabel(CS, inline=1, fontsize=10)
plt.title('Gradient Ascent')
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

5.4 plotSDerror.py

```
'''
Created on Oct 6, 2010

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import logRegres

def stocGradAscent0(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.5
    weights = ones(n)    #initialize to all ones
    weightsHistory=zeros((500*m,n))
    for j in range(500):
        for i in range(m):
            h = logRegres.sigmoid(sum(dataMatrix[i]*weights))
            error = classLabels[i] - h
            weights = weights + alpha * error * dataMatrix[i]
            weightsHistory[j*m + i,:] = weights
    return weightsHistory

def stocGradAscent1(dataMatrix, classLabels):
    m,n = shape(dataMatrix)
    alpha = 0.4
    weights = ones(n)    #initialize to all ones
    weightsHistory=zeros((40*m,n))
    for j in range(40):
        dataIndex = range(m)
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01
            randIndex = int(random.uniform(0,len(dataIndex)))
            h = logRegres.sigmoid(sum(dataMatrix[randIndex]*weights))
            error = classLabels[randIndex] - h
            #print error
            weights = weights + alpha * error * dataMatrix[randIndex]
            weightsHistory[j*m + i,:] = weights
            del(dataIndex[randIndex])
```

```
        print weights
        return weightsHistory


dataMat,labelMat=logRegres.loadDataSet()
dataArr = array(dataMat)
myHist = stocGradAscent1(dataArr,labelMat)



n = shape(dataArr)[0] #number of points to create
xcord1 = []; ycord1 = []
xcord2 = []; ycord2 = []

markers =[]
colors =[]



fig = plt.figure()
ax = fig.add_subplot(311)
type1 = ax.plot(myHist[:,0])
plt.ylabel('X0')
ax = fig.add_subplot(312)
type1 = ax.plot(myHist[:,1])
plt.ylabel('X1')
ax = fig.add_subplot(313)
type1 = ax.plot(myHist[:,2])
plt.xlabel('iteration')
plt.ylabel('X2')
plt.show()
```

5.5 sigmoidPlot.py

```
'''
Created on Oct 6, 2010

@author: Peter
'''
import sys
from pylab import *

t = arange(-60.0, 60.3, 0.1)
s = 1/(1 + exp(-t))
ax = subplot(211)
ax.plot(t,s)
ax.axis([-5,5,0,1])
plt.xlabel('x')
```

```
plt.ylabel('Sigmoid(x)')
ax = subplot(212)
ax.plot(t,s)
ax.axis([-60,60,0,1])
plt.xlabel('x')
plt.ylabel('Sigmoid(x)')
show()
```

6 支持向量机

6.1 svmMLiA.py

```
'''
Created on Nov 4, 2010
Chapter 5 source file for Machine Learing in Action
@author: Peter
'''
from numpy import *
from time import sleep

def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat,labelMat

def selectJrand(i,m):
    j=i #we want to select any J not equal to i
    while (j==i):
        j = int(random.uniform(0,m))
    return j

def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    dataMatrix = mat(dataMatIn); labelMat = mat(classLabels).transpose()
    b = 0; m,n = shape(dataMatrix)
    alphas = mat(zeros((m,1)))
    iter = 0
```

```
    while (iter < maxIter):
        alphaPairsChanged = 0
        for i in range(m):
            fXi = float(multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[i,:].T)) + b
            Ei = fXi - float(labelMat[i])#if checks if an example violates KKT conditions
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or ((labelMat[i]*Ei > toler) and
(alphas[i] > 0)):
                j = selectJrand(i,m)
                fXj = float(multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[j,:].T)) + b
                Ej = fXj - float(labelMat[j])
                alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy();
                if (labelMat[i] != labelMat[j]):
                    L = max(0, alphas[j] - alphas[i])
                    H = min(C, C + alphas[j] - alphas[i])
                else:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
                if L==H: print "L==H"; continue
                eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T - dataMatrix[i,:]*dataMatrix[i,:].T
- dataMatrix[j,:]*dataMatrix[j,:].T
                if eta >= 0: print "eta>=0"; continue
                alphas[j] -= labelMat[j]*(Ei - Ej)/eta
                alphas[j] = clipAlpha(alphas[j],H,L)
                if (abs(alphas[j] - alphaJold) < 0.00001): print "j not moving enough";
continue
                alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])#update i by the
same amount as j
                                                                        #the
update is in the oppostie direction
                b1 = b - Ei-
labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T -
labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[i,:]*dataMatrix[j,:].T
                b2 = b - Ej-
labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[j,:].T -
labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[j,:]*dataMatrix[j,:].T
                if (0 < alphas[i]) and (C > alphas[i]): b = b1
                elif (0 < alphas[j]) and (C > alphas[j]): b = b2
                else: b = (b1 + b2)/2.0
                alphaPairsChanged += 1
                print "iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged)
        if (alphaPairsChanged == 0): iter += 1
        else: iter = 0
        print "iteration number: %d" % iter
    return b,alphas
```

```
def kernelTrans(X, A, kTup): #calc the kernel or transform data to a higher dimensional space
    m,n = shape(X)
    K = mat(zeros((m,1)))
    if kTup[0]=='lin': K = X * A.T      #linear kernel
    elif kTup[0]=='rbf':
        for j in range(m):
            deltaRow = X[j,:] - A
            K[j] = deltaRow*deltaRow.T
        K = exp(K/(-1*kTup[1]**2)) #divide in NumPy is element-wise not matrix like Matlab
    else: raise NameError('Houston We Have a Problem -- \
    That Kernel is not recognized')
    return K

class optStruct:
    def __init__(self,dataMatIn, classLabels, C, toler, kTup):   # Initialize the structure with the parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2))) #first column is valid flag
        self.K = mat(zeros((self.m,self.m)))
        for i in range(self.m):
            self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

def calcEk(oS, k):
    fXk = float(multiply(oS.alphas,oS.labelMat).T*oS.K[:,k] + oS.b)
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJ(i, oS, Ei):                 #this is the second choice -heurstic, and calcs Ej
    maxK = -1; maxDeltaE = 0; Ej = 0
    oS.eCache[i] = [1,Ei]    #set valid #choose the alpha that gives the maximum delta E
    validEcacheList = nonzero(oS.eCache[:,0].A)[0]
    if (len(validEcacheList)) > 1:
        for k in validEcacheList:     #loop through valid Ecache values and find the one that maximizes delta E
            if k == i: continue #don't calc for i, waste of time
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
```

```
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:    #in this case (first time around) we don't have any valid eCache values
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEk(oS, k):#after any alpha has changed update the new value in the cache
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]

def innerL(i, oS):
    Ei = calcEk(oS, i)
    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)):
        j,Ej = selectJ(i, oS, Ei) #this has been changed from selectJrand
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        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
        eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j] #changed for kernel
        if eta >= 0: print "eta>=0"; return 0
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j) #added this for the Ecache
        if (abs(oS.alphas[j] - alphaJold) < 0.00001): print "j not moving enough"; return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])#update i by
the same amount as j
        updateEk(oS, i) #added this for the Ecache                    #the update is in the
oppostie direction
        b1 = oS.b - Ei- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,i] -
oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[i,j]
        b2 = oS.b - Ej- oS.labelMat[i]*(oS.alphas[i]-alphaIold)*oS.K[i,j]-
oS.labelMat[j]*(oS.alphas[j]-alphaJold)*oS.K[j,j]
        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
```

```
def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)):        #full Platt SMO
    oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler, kTup)
    iter = 0
    entireSet = True; alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        alphaPairsChanged = 0
        if entireSet:      #go over all
            for i in range(oS.m):
                alphaPairsChanged += innerL(i,oS)
                print "fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged)
            iter += 1
        else:#go over non-bound (railed) alphas
            nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
            for i in nonBoundIs:
                alphaPairsChanged += innerL(i,oS)
                print "non-bound, iter: %d i:%d, pairs changed %d" %
(iter,i,alphaPairsChanged)
            iter += 1
        if entireSet: entireSet = False #toggle entire set loop
        elif (alphaPairsChanged == 0): entireSet = True
        print "iteration number: %d" % iter
    return oS.b,oS.alphas

def calcWs(alphas,dataArr,classLabels):
    X = mat(dataArr); labelMat = mat(classLabels).transpose()
    m,n = shape(X)
    w = zeros((n,1))
    for i in range(m):
        w += multiply(alphas[i]*labelMat[i],X[i,:].T)
    return w

def testRbf(k1=1.3):
    dataArr,labelArr = loadDataSet('testSetRBF.txt')
    b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', k1)) #C=200 important
    datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
    svInd=nonzero(alphas.A>0)[0]
    sVs=datMat[svInd] #get matrix of only support vectors
    labelSV = labelMat[svInd];
    print "there are %d Support Vectors" % shape(sVs)[0]
    m,n = shape(datMat)
    errorCount = 0
    for i in range(m):
        kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
```

```
            predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
            if sign(predict)!=sign(labelArr[i]): errorCount += 1
        print "the training error rate is: %f" % (float(errorCount)/m)
        dataArr,labelArr = loadDataSet('testSetRBF2.txt')
        errorCount = 0
        datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
        m,n = shape(datMat)
        for i in range(m):
            kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', k1))
            predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
            if sign(predict)!=sign(labelArr[i]): errorCount += 1
        print "the test error rate is: %f" % (float(errorCount)/m)

def img2vector(filename):
        returnVect = zeros((1,1024))
        fr = open(filename)
        for i in range(32):
            lineStr = fr.readline()
            for j in range(32):
                returnVect[0,32*i+j] = int(lineStr[j])
        return returnVect

def loadImages(dirName):
        from os import listdir
        hwLabels = []
        trainingFileList = listdir(dirName)                    #load the training set
        m = len(trainingFileList)
        trainingMat = zeros((m,1024))
        for i in range(m):
            fileNameStr = trainingFileList[i]
            fileStr = fileNameStr.split('.')[0]         #take off .txt
            classNumStr = int(fileStr.split('_')[0])
            if classNumStr == 9: hwLabels.append(-1)
            else: hwLabels.append(1)
            trainingMat[i,:] = img2vector('%s/%s' % (dirName, fileNameStr))
        return trainingMat, hwLabels

def testDigits(kTup=('rbf', 10)):
        dataArr,labelArr = loadImages('trainingDigits')
        b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, kTup)
        datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
        svInd=nonzero(alphas.A>0)[0]
        sVs=datMat[svInd]
        labelSV = labelMat[svInd];
```

```
        print "there are %d Support Vectors" % shape(sVs)[0]
        m,n = shape(datMat)
        errorCount = 0
        for i in range(m):
            kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
            predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
            if sign(predict)!=sign(labelArr[i]): errorCount += 1
        print "the training error rate is: %f" % (float(errorCount)/m)
        dataArr,labelArr = loadImages('testDigits')
        errorCount = 0
        datMat=mat(dataArr); labelMat = mat(labelArr).transpose()
        m,n = shape(datMat)
        for i in range(m):
            kernelEval = kernelTrans(sVs,datMat[i,:],kTup)
            predict=kernelEval.T * multiply(labelSV,alphas[svInd]) + b
            if sign(predict)!=sign(labelArr[i]): errorCount += 1
        print "the test error rate is: %f" % (float(errorCount)/m)


'''#######********************************
Non-Kernel VErsions below
'''#######********************************

class optStructK:
    def __init__(self,dataMatIn, classLabels, C, toler):  # Initialize the structure with the parameters
        self.X = dataMatIn
        self.labelMat = classLabels
        self.C = C
        self.tol = toler
        self.m = shape(dataMatIn)[0]
        self.alphas = mat(zeros((self.m,1)))
        self.b = 0
        self.eCache = mat(zeros((self.m,2))) #first column is valid flag

def calcEkK(oS, k):
    fXk = float(multiply(oS.alphas,oS.labelMat).T*(oS.X*oS.X[k,:].T)) + oS.b
    Ek = fXk - float(oS.labelMat[k])
    return Ek

def selectJK(i, oS, Ei):         #this is the second choice -heurstic, and calcs Ej
    maxK = -1; maxDeltaE = 0; Ej = 0
    oS.eCache[i] = [1,Ei]  #set valid #choose the alpha that gives the maximum delta E
    validEcacheList = nonzero(oS.eCache[:,0].A)[0]
```

```
    if (len(validEcacheList)) > 1:
        for k in validEcacheList:    #loop through valid Ecache values and find the one that maximizes delta E
            if k == i: continue #don't calc for i, waste of time
            Ek = calcEk(oS, k)
            deltaE = abs(Ei - Ek)
            if (deltaE > maxDeltaE):
                maxK = k; maxDeltaE = deltaE; Ej = Ek
        return maxK, Ej
    else:    #in this case (first time around) we don't have any valid eCache values
        j = selectJrand(i, oS.m)
        Ej = calcEk(oS, j)
    return j, Ej

def updateEkK(oS, k):#after any alpha has changed update the new value in the cache
    Ek = calcEk(oS, k)
    oS.eCache[k] = [1,Ek]

def innerLK(i, oS):
    Ei = calcEk(oS, i)
    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)):
        j,Ej = selectJ(i, oS, Ei) #this has been changed from selectJrand
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        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
        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
        oS.alphas[j] -= oS.labelMat[j]*(Ei - Ej)/eta
        oS.alphas[j] = clipAlpha(oS.alphas[j],H,L)
        updateEk(oS, j) #added this for the Ecache
        if (abs(oS.alphas[j] - alphaJold) < 0.00001): print "j not moving enough"; return 0
        oS.alphas[i] += oS.labelMat[j]*oS.labelMat[i]*(alphaJold - oS.alphas[j])#update i by the same amount as j
        updateEk(oS, i) #added this for the Ecache                    #the update is in the oppostie direction
        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
            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

def smoPK(dataMatIn, classLabels, C, toler, maxIter):       #full Platt SMO
        oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler)
        iter = 0
        entireSet = True; alphaPairsChanged = 0
        while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
            alphaPairsChanged = 0
            if entireSet:     #go over all
                for i in range(oS.m):
                    alphaPairsChanged += innerL(i,oS)
                    print "fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged)
                iter += 1
            else:#go over non-bound (railed) alphas
                nonBoundIs = nonzero((oS.alphas.A > 0) * (oS.alphas.A < C))[0]
                for i in nonBoundIs:
                    alphaPairsChanged += innerL(i,oS)
                    print "non-bound, iter: %d i:%d, pairs changed %d" %
(iter,i,alphaPairsChanged)
                iter += 1
            if entireSet: entireSet = False #toggle entire set loop
            elif (alphaPairsChanged == 0): entireSet = True
            print "iteration number: %d" % iter
        return oS.b,oS.alphas
```

6.2 notLinSeperable.py

```
    '''
    Created on Oct 6, 2010

    @author: Peter
    '''
    from numpy import *
    import matplotlib
    import matplotlib.pyplot as plt

    xcord0 = []; ycord0 = []; xcord1 = []; ycord1 = []
    markers =[]
    colors =[]
    fr = open('testSet.txt')#this file was generated by 2normalGen.py
    for line in fr.readlines():
```

```
        lineSplit = line.strip().split('\t')
        xPt = float(lineSplit[0])
        yPt = float(lineSplit[1])
        label = int(lineSplit[2])
        if (label == 0):
            xcord0.append(xPt)
            ycord0.append(yPt)
        else:
            xcord1.append(xPt)
            ycord1.append(yPt)

fr.close()
fig = plt.figure()
ax = fig.add_subplot(221)
xcord0 = []; ycord0 = []; xcord1 = []; ycord1 = []
for i in range(300):
        [x,y] = random.uniform(0,1,2)
        if ((x > 0.5) and (y < 0.5)) or ((x < 0.5) and (y > 0.5)):
            xcord0.append(x); ycord0.append(y)
        else:
            xcord1.append(x); ycord1.append(y)
ax.scatter(xcord0,ycord0, marker='s', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
plt.title('A')
ax = fig.add_subplot(222)
xcord0 = random.standard_normal(150); ycord0 = random.standard_normal(150)
xcord1 = random.standard_normal(150)+2.0; ycord1 = random.standard_normal(150)+2.0
ax.scatter(xcord0,ycord0, marker='s', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
plt.title('B')
ax = fig.add_subplot(223)
xcord0 = []; ycord0 = []; xcord1 = []; ycord1 = []
for i in range(300):
        [x,y] = random.uniform(0,1,2)
        if (x > 0.5):
            xcord0.append(x*cos(2.0*pi*y)); ycord0.append(x*sin(2.0*pi*y))
        else:
            xcord1.append(x*cos(2.0*pi*y)); ycord1.append(x*sin(2.0*pi*y))
ax.scatter(xcord0,ycord0, marker='s', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
plt.title('C')
ax = fig.add_subplot(224)
xcord1 = zeros(150); ycord1 = zeros(150)
xcord0 = random.uniform(-3,3,350); ycord0 = random.uniform(-3,3,350);
```

```
    xcord1[0:50] = 0.3*random.standard_normal(50)+2.0; ycord1[0:50] = 0.3*random.standard_normal(50)+2.0

    xcord1[50:100] = 0.3*random.standard_normal(50)-2.0; ycord1[50:100] = 0.3*random.standard_normal(50)-3.0

    xcord1[100:150] = 0.3*random.standard_normal(50)+1.0; ycord1[100:150] = 0.3*random.standard_normal(50)
    ax.scatter(xcord0,ycord0, marker='s', s=90)
    ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
    plt.title('D')
    plt.show()
```

6.3 plotRBF.py

```
    '''
    Created on Nov 26, 2010

    @author: Peter
    '''
    from numpy import *
    import matplotlib
    import matplotlib.pyplot as plt

    xcord0 = []; ycord0 = []; xcord1 = []; ycord1 = []
    fw = open('testSetRBF2.txt', 'w')#generate data

    fig = plt.figure()
    ax = fig.add_subplot(111)
    xcord0 = []; ycord0 = []; xcord1 = []; ycord1 = []
    for i in range(100):
        [x,y] = random.uniform(0,1,2)
        xpt=x*cos(2.0*pi*y); ypt = x*sin(2.0*pi*y)
        if (x > 0.5):
            xcord0.append(xpt); ycord0.append(ypt)
            label = -1.0
        else:
            xcord1.append(xpt); ycord1.append(ypt)
            label = 1.0
        fw.write('%f\t%f\t%f\n' % (xpt, ypt, label))
    ax.scatter(xcord0,ycord0, marker='s', s=90)
    ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
    plt.title('Non-linearly Separable Data for Kernel Method')
    plt.show()
    fw.close()
```

6.4 plotSupportVectors.py

```
'''
Created on Nov 22, 2010

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

xcord0 = []
ycord0 = []
xcord1 = []
ycord1 = []
markers =[]
colors =[]
fr = open('testSet.txt')#this file was generated by 2normalGen.py
for line in fr.readlines():
    lineSplit = line.strip().split('\t')
    xPt = float(lineSplit[0])
    yPt = float(lineSplit[1])
    label = int(lineSplit[2])
    if (label == -1):
        xcord0.append(xPt)
        ycord0.append(yPt)
    else:
        xcord1.append(xPt)
        ycord1.append(yPt)

fr.close()
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xcord0,ycord0, marker='s', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
plt.title('Support Vectors Circled')
circle = Circle((4.6581910000000004, 3.507396), 0.5, facecolor='none', edgecolor=(0,0.8,0.8), linewidth=3, alpha=0.5)
ax.add_patch(circle)
circle = Circle((3.4570959999999999, -0.082215999999999997), 0.5, facecolor='none', edgecolor=(0,0.8,0.8), linewidth=3, alpha=0.5)
ax.add_patch(circle)
circle = Circle((6.0805730000000002, 0.41888599999999998), 0.5, facecolor='none', edgecolor=(0,0.8,0.8), linewidth=3, alpha=0.5)
```

```
ax.add_patch(circle)
#plt.plot([2.3,8.5], [-6,6]) #seperating hyperplane
b = -3.75567; w0=0.8065; w1=-0.2761
x = arange(-2.0, 12.0, 0.1)
y = (-w0*x - b)/w1
ax.plot(x,y)
ax.axis([-2,12,-8,6])
plt.show()
```

7 利用 AdaBoost 元算法提高分类性能

7.1 adaboost.py

```
'''
Created on Nov 28, 2010
Adaboost is short for Adaptive Boosting
@author: Peter
'''
from numpy import *

def loadSimpData():
    datMat = matrix([[ 1. ,  2.1],
        [ 2. ,  1.1],
        [ 1.3,  1. ],
        [ 1. ,  1. ],
        [ 2. ,  1. ]])
    classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
    return datMat,classLabels

def loadDataSet(fileName):      #general function to parse tab -delimited floats
    numFeat = len(open(fileName).readline().split('\t')) #get number of fields
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat-1):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat

def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):#just classify the data
    retArray = ones((shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
```

```
            retArray[dataMatrix[:,dimen] > threshVal] = -1.0
        return retArray


    def buildStump(dataArr,classLabels,D):
        dataMatrix = mat(dataArr); labelMat = mat(classLabels).T
        m,n = shape(dataMatrix)
        numSteps = 10.0; bestStump = {}; bestClasEst = mat(zeros((m,1)))
        minError = inf #init error sum, to +infinity
        for i in range(n):#loop over all dimensions
            rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
            stepSize = (rangeMax-rangeMin)/numSteps
            for j in range(-1,int(numSteps)+1):#loop over all range in current dimension
                for inequal in ['lt', 'gt']: #go over less than and greater than
                    threshVal = (rangeMin + float(j) * stepSize)
                    predictedVals = stumpClassify(dataMatrix,i,threshVal,inequal)#call
stump classify with i, j, lessThan
                    errArr = mat(ones((m,1)))
                    errArr[predictedVals == labelMat] = 0
                    weightedError = D.T*errArr  #calc total error multiplied by D
                    #print "split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error
is %.3f" % (i, threshVal, inequal, weightedError)
                    if weightedError < minError:
                        minError = weightedError
                        bestClasEst = predictedVals.copy()
                        bestStump['dim'] = i
                        bestStump['thresh'] = threshVal
                        bestStump['ineq'] = inequal
        return bestStump,minError,bestClasEst


    def adaBoostTrainDS(dataArr,classLabels,numIt=40):
        weakClassArr = []
        m = shape(dataArr)[0]
        D = mat(ones((m,1))/m)   #init D to all equal
        aggClassEst = mat(zeros((m,1)))
        for i in range(numIt):
            bestStump,error,classEst = buildStump(dataArr,classLabels,D)#build Stump
            #print "D:",D.T
            alpha = float(0.5*log((1.0-error)/max(error,1e-16)))#calc alpha, throw in
max(error,eps) to account for error=0
            bestStump['alpha'] = alpha
            weakClassArr.append(bestStump)                  #store Stump Params in
Array
```

```
            #print "classEst: ",classEst.T
            expon = multiply(-1*alpha*mat(classLabels).T,classEst) #exponent for D calc, getting messy
            D = multiply(D,exp(expon))                              #Calc New D for next iteration
            D = D/D.sum()
            #calc training error of all classifiers, if this is 0 quit for loop early (use break)
            aggClassEst += alpha*classEst
            #print "aggClassEst: ",aggClassEst.T
            aggErrors = multiply(sign(aggClassEst) != mat(classLabels).T,ones((m,1)))
            errorRate = aggErrors.sum()/m
            print "total error: ",errorRate
            if errorRate == 0.0: break
        return weakClassArr,aggClassEst

    def adaClassify(datToClass,classifierArr):
        dataMatrix = mat(datToClass)#do stuff similar to last aggClassEst in adaBoostTrainDS
        m = shape(dataMatrix)[0]
        aggClassEst = mat(zeros((m,1)))
        for i in range(len(classifierArr)):
            classEst = stumpClassify(dataMatrix,classifierArr[i]['dim'],\
                                     classifierArr[i]['thresh'],\
                                     classifierArr[i]['ineq'])#call stump classify
            aggClassEst += classifierArr[i]['alpha']*classEst
            print aggClassEst
        return sign(aggClassEst)

    def plotROC(predStrengths, classLabels):
        import matplotlib.pyplot as plt
        cur = (1.0,1.0) #cursor
        ySum = 0.0 #variable to calculate AUC
        numPosClas = sum(array(classLabels)==1.0)
        yStep = 1/float(numPosClas); xStep = 1/float(len(classLabels)-numPosClas)
        sortedIndicies = predStrengths.argsort()#get sorted index, it's reverse
        fig = plt.figure()
        fig.clf()
        ax = plt.subplot(111)
        #loop through all the values, drawing a line segment at each point
        for index in sortedIndicies.tolist()[0]:
            if classLabels[index] == 1.0:
                delX = 0; delY = yStep;
            else:
                delX = xStep; delY = 0;
                ySum += cur[1]
```

```
            #draw line from cur to (cur[0]-delX,cur[1]-delY)
            ax.plot([cur[0],cur[0]-delX],[cur[1],cur[1]-delY], c='b')
            cur = (cur[0]-delX,cur[1]-delY)
        ax.plot([0,1],[0,1],'b--')
        plt.xlabel('False positive rate'); plt.ylabel('True positive rate')
        plt.title('ROC curve for AdaBoost horse colic detection system')
        ax.axis([0,1,0,1])
        plt.show()
        print "the Area Under the Curve is: ",ySum*xStep
```

7.2 old_adaboost.py

```
    '''
    Created on Nov 28, 2010
    Adaboost is short for Adaptive Boosting
    @author: Peter
    '''
    from numpy import *

    def loadDataSet(fileName):
        dataMat = []; labelMat = []
        fr = open(fileName)
        for line in fr.readlines():
            lineArr = line.strip().split('\t')
            dataMat.append([float(lineArr[0]), float(lineArr[1])])
            labelMat.append(float(lineArr[2]))
        return dataMat,labelMat

    def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):#just classify the data
        retArray = ones((shape(dataMatrix)[0],1))
        if threshIneq == 'lt':
            retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
        else:
            retArray[dataMatrix[:,dimen] > threshVal] = -1.0
        return retArray


    def buildStump(dataArr,classLabels,D):
        dataMatrix = mat(dataArr); labelMat = mat(classLabels).T
        m,n = shape(dataMatrix)
        numSteps = 10.0; bestStump = {}; bestClasEst = mat(zeros((m,1)))
        minError = inf #init error sum, to +infinity
        for i in range(n):#loop over all dimensions
            rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
            stepSize = (rangeMax-rangeMin)/numSteps
            for j in range(-1,int(numSteps)+1):#loop over all range in current dimension
```

```
                for inequal in ['lt', 'gt']: #go over less than and greater than
                    threshVal = (rangeMin + float(j) * stepSize)
                    predictedVals = stumpClassify(dataMatrix,i,threshVal,inequal)#call stump classify with i, j, lessThan
                    errArr = mat(ones((m,1)))
                    errArr[predictedVals == labelMat] = 0
                    print "predictedVals",predictedVals.T,"errArr",errArr.T
                    weightedError = D.T*errArr  #calc total error multiplied by D
                    print "split: dim %d, thresh %.2f, thresh ineqal: %s, the weighted error is %.3f" % (i, threshVal, inequal, weightedError)
                    if weightedError < minError:
                        minError = weightedError
                        bestClasEst = predictedVals.copy()
                        bestStump['dim'] = i
                        bestStump['thresh'] = threshVal
                        bestStump['ineq'] = inequal
        return bestStump,minError,bestClasEst

    def adaBoostTrain(dataArr,classLabels,numIt=40):
        weakClassArr = []
        m = shape(dataArr)[0]
        D = mat(ones((m,1))/m)   #init D to all equal
        aggClassEst = mat(zeros((m,1)))
        for i in range(numIt):
            bestStump,error,classEst = buildStump(dataArr,classLabels,D)#build Stump
            #print "error",error
            alpha = float(0.5*log((1.0-error)/max(error,1e-16)))#calc alpha, throw in max(error,eps) to account for error=0
            bestStump['alpha'] = alpha
            print "alpha",alpha
            weakClassArr.append(bestStump)#store Stump Params in Array
            print "classEst",classEst
            expon = multiply(-1*alpha*mat(classLabels).T,classEst) #exponent for D calc, getting messy
            D = multiply(D,exp(expon)) #Calc New D, element-wise
            D = D/D.sum()
            print "D",D
            #calc training error of all classifiers, if this is 0 quit for loop early (use break)
            aggClassEst += alpha*classEst
            print "aggClassEst",aggClassEst
            aggErrors = multiply(sign(aggClassEst) != mat(classLabels).T,ones((m,1)))
            #print aggErrors
            errorRate = aggErrors.sum()/m
            print errorRate
```

```
            if errorRate == 0.0: break
        return weakClassArr
```

7.3 simpleDataPlot.py

```
'''
Created on Dec 13, 2010

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
datMat = matrix([[ 1. ,  2.1],
            [ 1.5,  1.6],
            [ 1.3,  1. ],
            [ 1. ,  1. ],
            [ 2. ,  1. ]])
classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]



xcord0 = []
ycord0 = []
xcord1 = []
ycord1 = []
markers =[]
colors =[]

for i in range(len(classLabels)):
    if classLabels[i]==1.0:
        xcord1.append(datMat[i,0]), ycord1.append(datMat[i,1])
    else:
        xcord0.append(datMat[i,0]), ycord0.append(datMat[i,1])
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xcord0,ycord0, marker='s', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50, c='red')
plt.title('decision stump test data')
plt.show()
```

8 预测数值型数据：回归

8.1 regression.py

```
'''
Created on Jan 8, 2011

@author: Peter
'''
```

```
from numpy import *

def loadDataSet(fileName):          #general function to parse tab -delimited floats
    numFeat = len(open(fileName).readline().split('\t')) - 1 #get number of fields
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat

def standRegres(xArr,yArr):
    xMat = mat(xArr); yMat = mat(yArr).T
    xTx = xMat.T*xMat
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T*yMat)
    return ws

def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat = mat(xArr); yMat = mat(yArr).T
    m = shape(xMat)[0]
    weights = mat(eye((m)))
    for j in range(m):                      #next 2 lines create weights matrix
        diffMat = testPoint - xMat[j,:]     #
        weights[j,j] = exp(diffMat*diffMat.T/(-2.0*k**2))
    xTx = xMat.T * (weights * xMat)
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T * (weights * yMat))
    return testPoint * ws

def lwlrTest(testArr,xArr,yArr,k=1.0):  #loops over all the data points and applies lwlr to each one
    m = shape(testArr)[0]
    yHat = zeros(m)
    for i in range(m):
        yHat[i] = lwlr(testArr[i],xArr,yArr,k)
```

```
    return yHat

def lwlrTestPlot(xArr,yArr,k=1.0):    #same thing as lwlrTest except it sorts X first
    yHat = zeros(shape(yArr))             #easier for plotting
    xCopy = mat(xArr)
    xCopy.sort(0)
    for i in range(shape(xArr)[0]):
        yHat[i] = lwlr(xCopy[i],xArr,yArr,k)
    return yHat,xCopy

def rssError(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    return ((yArr-yHatArr)**2).sum()

def ridgeRegres(xMat,yMat,lam=0.2):
    xTx = xMat.T*xMat
    denom = xTx + eye(shape(xMat)[1])*lam
    if linalg.det(denom) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = denom.I * (xMat.T*yMat)
    return ws

def ridgeTest(xArr,yArr):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean          #to eliminate X0 take mean off of Y
    #regularize X's
    xMeans = mean(xMat,0)      #calc mean then subtract it off
    xVar = var(xMat,0)           #calc variance of Xi then divide by it
    xMat = (xMat - xMeans)/xVar
    numTestPts = 30
    wMat = zeros((numTestPts,shape(xMat)[1]))
    for i in range(numTestPts):
        ws = ridgeRegres(xMat,yMat,exp(i-10))
        wMat[i,:]=ws.T
    return wMat

def regularize(xMat):#regularize by columns
    inMat = xMat.copy()
    inMeans = mean(inMat,0)      #calc mean then subtract it off
    inVar = var(inMat,0)           #calc variance of Xi then divide by it
    inMat = (inMat - inMeans)/inVar
    return inMat
```

```
def stageWise(xArr,yArr,eps=0.01,numIt=100):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean     #can also regularize ys but will get smaller coef
    xMat = regularize(xMat)
    m,n=shape(xMat)
    #returnMat = zeros((numIt,n)) #testing code remove
    ws = zeros((n,1)); wsTest = ws.copy(); wsMax = ws.copy()
    for i in range(numIt):
        print ws.T
        lowestError = inf;
        for j in range(n):
            for sign in [-1,1]:
                wsTest = ws.copy()
                wsTest[j] += eps*sign
                yTest = xMat*wsTest
                rssE = rssError(yMat.A,yTest.A)
                if rssE < lowestError:
                    lowestError = rssE
                    wsMax = wsTest
        ws = wsMax.copy()
        #returnMat[i,:]=ws.T
    #return returnMat

#def scrapePage(inFile,outFile,yr,numPce,origPrc):
#    from BeautifulSoup import BeautifulSoup
#    fr = open(inFile); fw=open(outFile,'a') #a is append mode writing
#    soup = BeautifulSoup(fr.read())
#    i=1
#    currentRow = soup.findAll('table', r="%d" % i)
#    while(len(currentRow)!=0):
#        title = currentRow[0].findAll('a')[1].text
#        lwrTitle = title.lower()
#        if (lwrTitle.find('new') > -1) or (lwrTitle.find('nisb') > -1):
#            newFlag = 1.0
#        else:
#            newFlag = 0.0
#        soldUnicde = currentRow[0].findAll('td')[3].findAll('span')
#        if len(soldUnicde)==0:
#            print "item #%d did not sell" % i
#        else:
#            soldPrice = currentRow[0].findAll('td')[4]
#            priceStr = soldPrice.text
#            priceStr = priceStr.replace('$',') #strips out $
```

```
#                   priceStr = priceStr.replace(',','') #strips out ,
#                   if len(soldPrice)>1:
#                       priceStr = priceStr.replace('Free shipping', '') #strips out Free Shipping
#                   print "%s\t%d\t%s" % (priceStr,newFlag,title)
#                   fw.write("%d\t%d\t%d\t%f\t%s\n" % (yr,numPce,newFlag,origPrc,priceStr))
#           i += 1
#           currentRow = soup.findAll('table', r="%d" % i)
#       fw.close()

    from time import sleep
    import json
    import urllib2
    def searchForSet(retX, retY, setNum, yr, numPce, origPrc):
        sleep(10)
        myAPIstr = 'AIzaSyD2cR2KFyx12hXu6PFU-wrWot3NXvko8vY'
        searchURL                                                                    =
'https://www.googleapis.com/shopping/search/v1/public/products?key=%s&country=US&q=lego
+%d&alt=json' % (myAPIstr, setNum)
        pg = urllib2.urlopen(searchURL)
        retDict = json.loads(pg.read())
        for i in range(len(retDict['items'])):
            try:
                currItem = retDict['items'][i]
                if currItem['product']['condition'] == 'new':
                    newFlag = 1
                else: newFlag = 0
                listOfInv = currItem['product']['inventories']
                for item in listOfInv:
                    sellingPrice = item['price']
                    if  sellingPrice > origPrc * 0.5:
                        print  "%d\t%d\t%d\t%f\t%f"  %  (yr,numPce,newFlag,origPrc,
sellingPrice)
                        retX.append([yr, numPce, newFlag, origPrc])
                        retY.append(sellingPrice)
            except: print 'problem with item %d' % i

    def setDataCollect(retX, retY):
        searchForSet(retX, retY, 8288, 2006, 800, 49.99)
        searchForSet(retX, retY, 10030, 2002, 3096, 269.99)
        searchForSet(retX, retY, 10179, 2007, 5195, 499.99)
        searchForSet(retX, retY, 10181, 2007, 3428, 199.99)
        searchForSet(retX, retY, 10189, 2008, 5922, 299.99)
        searchForSet(retX, retY, 10196, 2009, 3263, 249.99)
```

```
def crossValidation(xArr,yArr,numVal=10):
    m = len(yArr)
    indexList = range(m)
    errorMat = zeros((numVal,30))#create error mat 30columns numVal rows
    for i in range(numVal):
        trainX=[]; trainY=[]
        testX = []; testY = []
        random.shuffle(indexList)
        for j in range(m):#create training set based on first 90% of values in indexList
            if j < m*0.9:
                trainX.append(xArr[indexList[j]])
                trainY.append(yArr[indexList[j]])
            else:
                testX.append(xArr[indexList[j]])
                testY.append(yArr[indexList[j]])
        wMat = ridgeTest(trainX,trainY)    #get 30 weight vectors from ridge
        for k in range(30):#loop over all of the ridge estimates
            matTestX = mat(testX); matTrainX=mat(trainX)
            meanTrain = mean(matTrainX,0)
            varTrain = var(matTrainX,0)
            matTestX = (matTestX-meanTrain)/varTrain #regularize test with training
params
            yEst = matTestX * mat(wMat[k,:]).T + mean(trainY)#test ridge results and
store
            errorMat[i,k]=rssError(yEst.T.A,array(testY))
            #print errorMat[i,k]
    meanErrors = mean(errorMat,0)#calc avg performance of the different ridge weight
vectors
    minMean = float(min(meanErrors))
    bestWeights = wMat[nonzero(meanErrors==minMean)]
    #can unregularize to get model
    #when we regularized we wrote Xreg = (x-meanX)/var(x)
    #we can now write in terms of x not Xreg:  x*w/var(x) - meanX/var(x) +meanY
    xMat = mat(xArr); yMat=mat(yArr).T
    meanX = mean(xMat,0); varX = var(xMat,0)
    unReg = bestWeights/varX
    print "the best model from Ridge Regression is:\n",unReg
    print "with constant term: ",-1*sum(multiply(meanX,unReg)) + mean(yMat)
```

8.2 Old_regression.py

```
"""
Created on Jan 8, 2011

@author: Peter
"""
```

```
from numpy import *

def loadDataSet(fileName):          #general function to parse tab -delimited floats
    numFeat = len(open(fileName).readline().split('\t')) - 1 #get number of fields
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr =[]
        curLine = line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curLine[i]))
        dataMat.append(lineArr)
        labelMat.append(float(curLine[-1]))
    return dataMat,labelMat

def standRegres(xArr,yArr):
    xMat = mat(xArr); yMat = mat(yArr).T
    xTx = xMat.T*xMat
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T*yMat)
    return ws

def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat = mat(xArr); yMat = mat(yArr).T
    m = shape(xMat)[0]
    weights = mat(eye((m)))
    for j in range(m):                      #next 2 lines create weights matrix
        diffMat = testPoint - xMat[j,:]     #
        weights[j,j] = exp(diffMat*diffMat.T/(-2.0*k**2))
    xTx = xMat.T * (weights * xMat)
    if linalg.det(xTx) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = xTx.I * (xMat.T * (weights * yMat))
    return testPoint * ws

def lwlrTest(testArr,xArr,yArr,k=1.0):  #loops over all the data points and applies lwlr to each one
    m = shape(testArr)[0]
    yHat = zeros(m)
    for i in range(m):
        yHat[i] = lwlr(testArr[i],xArr,yArr,k)
```

```
    return yHat

def lwlrTestPlot(xArr,yArr,k=1.0):    #same thing as lwlrTest except it sorts X first
    yHat = zeros(shape(yArr))             #easier for plotting
    xCopy = mat(xArr)
    xCopy.sort(0)
    for i in range(shape(xArr)[0]):
        yHat[i] = lwlr(xCopy[i],xArr,yArr,k)
    return yHat,xCopy

def rssError(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    return ((yArr-yHatArr)**2).sum()

def ridgeRegres(xMat,yMat,lam=0.2):
    xTx = xMat.T*xMat
    denom = xTx + eye(shape(xMat)[1])*lam
    if linalg.det(denom) == 0.0:
        print "This matrix is singular, cannot do inverse"
        return
    ws = denom.I * (xMat.T*yMat)
    return ws

def ridgeTest(xArr,yArr):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean          #to eliminate X0 take mean off of Y
    #regularize X's
    xMeans = mean(xMat,0)      #calc mean then subtract it off
    xVar = var(xMat,0)           #calc variance of Xi then divide by it
    xMat = (xMat - xMeans)/xVar
    numTestPts = 30
    wMat = zeros((numTestPts,shape(xMat)[1]))
    for i in range(numTestPts):
        ws = ridgeRegres(xMat,yMat,exp(i-10))
        wMat[i,:]=ws.T
    return wMat

def regularize(xMat):#regularize by columns
    inMat = xMat.copy()
    inMeans = mean(inMat,0)      #calc mean then subtract it off
    inVar = var(inMat,0)           #calc variance of Xi then divide by it
    inMat = (inMat - inMeans)/inVar
    return inMat
```

```
def stageWise(xArr,yArr,eps=0.01,numIt=100):
    xMat = mat(xArr); yMat=mat(yArr).T
    yMean = mean(yMat,0)
    yMat = yMat - yMean     #can also regularize ys but will get smaller coef
    xMat = regularize(xMat)
    m,n=shape(xMat)
    returnMat = zeros((numIt,n)) #testing code remove
    ws = zeros((n,1)); wsTest = ws.copy(); wsMax = ws.copy()
    for i in range(numIt):#could change this to while loop
        #print ws.T
        lowestError = inf;
        for j in range(n):
            for sign in [-1,1]:
                wsTest = ws.copy()
                wsTest[j] += eps*sign
                yTest = xMat*wsTest
                rssE = rssError(yMat.A,yTest.A)
                if rssE < lowestError:
                    lowestError = rssE
                    wsMax = wsTest
        ws = wsMax.copy()
        returnMat[i,:]=ws.T
    return returnMat

def scrapePage(inFile,outFile,yr,numPce,origPrc):
    from BeautifulSoup import BeautifulSoup
    fr = open(inFile); fw=open(outFile,'a') #a is append mode writing
    soup = BeautifulSoup(fr.read())
    i=1
    currentRow = soup.findAll('table', r="%d" % i)
    while(len(currentRow)!=0):
        currentRow = soup.findAll('table', r="%d" % i)
        title = currentRow[0].findAll('a')[1].text
        lwrTitle = title.lower()
        if (lwrTitle.find('new') > -1) or (lwrTitle.find('nisb') > -1):
            newFlag = 1.0
        else:
            newFlag = 0.0
        soldUnicde = currentRow[0].findAll('td')[3].findAll('span')
        if len(soldUnicde)==0:
            print "item #%d did not sell" % i
        else:
            soldPrice = currentRow[0].findAll('td')[4]
            priceStr = soldPrice.text
```

```
                priceStr = priceStr.replace('$','') #strips out $
                priceStr = priceStr.replace(',','') #strips out ,
                if len(soldPrice)>1:
                    priceStr = priceStr.replace('Free shipping', '') #strips out Free Shipping
                print "%s\t%d\t%s" % (priceStr,newFlag,title)
                fw.write("%d\t%d\t%d\t%f\t%s\n" % (yr,numPce,newFlag,origPrc,priceStr))
            i += 1
            currentRow = soup.findAll('table', r="%d" % i)
        fw.close()

    def setDataCollect():
        scrapePage('setHtml/lego8288.html','out.txt', 2006, 800, 49.99)
        scrapePage('setHtml/lego10030.html','out.txt', 2002, 3096, 269.99)
        scrapePage('setHtml/lego10179.html','out.txt', 2007, 5195, 499.99)
        scrapePage('setHtml/lego10181.html','out.txt', 2007, 3428, 199.99)
        scrapePage('setHtml/lego10189.html','out.txt', 2008, 5922, 299.99)
        scrapePage('setHtml/lego10196.html','out.txt', 2009, 3263, 249.99)

    def crossValidation(xArr,yArr,numVal=10):
        m = len(yArr)
        indexList = range(m)
        errorMat = zeros((numVal,30))#create error mat 30columns numVal rows
        for i in range(numVal):
            trainX=[]; trainY=[]
            testX = []; testY = []
            random.shuffle(indexList)
            for j in range(m):#create training set based on first 90% of values in indexList
                if j < m*0.9:
                    trainX.append(xArr[indexList[j]])
                    trainY.append(yArr[indexList[j]])
                else:
                    testX.append(xArr[indexList[j]])
                    testY.append(yArr[indexList[j]])
            wMat = ridgeTest(trainX,trainY)    #get 30 weight vectors from ridge
            for k in range(30):#loop over all of the ridge estimates
                matTestX = mat(testX); matTrainX=mat(trainX)
                meanTrain = mean(matTrainX,0)
                varTrain = var(matTrainX,0)
                matTestX = (matTestX-meanTrain)/varTrain #regularize test with training
params
                yEst = matTestX * mat(wMat[k,:]).T + mean(trainY)#test ridge results and
store
                errorMat[i,k]=rssError(yEst.T.A,array(testY))
                #print errorMat[i,k]
```

```
        meanErrors = mean(errorMat,0)#calc avg performance of the different ridge weight
vectors
        minMean = float(min(meanErrors))
        bestWeights = wMat[nonzero(meanErrors==minMean)]
        #can unregularize to get model
        #when we regularized we wrote Xreg = (x-meanX)/var(x)
        #we can now write in terms of x not Xreg:  x*w/var(x) - meanX/var(x) +meanY
        xMat = mat(xArr); yMat=mat(yArr).T
        meanX = mean(xMat,0); varX = var(xMat,0)
        unReg = bestWeights/varX
        print "the best model from Ridge Regression is:\n",unReg
        print "with constant term: ",-1*sum(multiply(meanX,unReg)) + mean(yMat)
```

9 树回归

9.1 regTrees.py

```
'''
Created on Feb 4, 2011
Tree-Based Regression Methods
@author: Peter Harrington
'''
from numpy import *

def loadDataSet(fileName):          #general function to parse tab -delimited floats
    dataMat = []                    #assume last column is target value
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = map(float,curLine) #map all elements to float()
        dataMat.append(fltLine)
    return dataMat

def binSplitDataSet(dataSet, feature, value):
    mat0 = dataSet[nonzero(dataSet[:,feature] > value)[0],:][0]
    mat1 = dataSet[nonzero(dataSet[:,feature] <= value)[0],:][0]
    return mat0,mat1

def regLeaf(dataSet):#returns the value used for each leaf
    return mean(dataSet[:,-1])

def regErr(dataSet):
    return var(dataSet[:,-1]) * shape(dataSet)[0]

def linearSolve(dataSet):     #helper function used in two places
    m,n = shape(dataSet)
    X = mat(ones((m,n))); Y = mat(ones((m,1)))#create a copy of data with 1 in 0th postion
```

```
    X[:,1:n] = dataSet[:,0:n-1]; Y = dataSet[:,-1]#and strip out Y
    xTx = X.T*X
    if linalg.det(xTx) == 0.0:
        raise NameError('This matrix is singular, cannot do inverse,\n\
        try increasing the second value of ops')
    ws = xTx.I * (X.T * Y)
    return ws,X,Y

def modelLeaf(dataSet):#create linear model and return coeficients
    ws,X,Y = linearSolve(dataSet)
    return ws

def modelErr(dataSet):
    ws,X,Y = linearSolve(dataSet)
    yHat = X * ws
    return sum(power(Y - yHat,2))

def chooseBestSplit(dataSet, leafType=regLeaf, errType=regErr, ops=(1,4)):
    tolS = ops[0]; tolN = ops[1]
    #if all the target variables are the same value: quit and return value
    if len(set(dataSet[:,-1].T.tolist()[0])) == 1: #exit cond 1
        return None, leafType(dataSet)
    m,n = shape(dataSet)
    #the choice of the best feature is driven by Reduction in RSS error from mean
    S = errType(dataSet)
    bestS = inf; bestIndex = 0; bestValue = 0
    for featIndex in range(n-1):
        for splitVal in set(dataSet[:,featIndex]):
            mat0, mat1 = binSplitDataSet(dataSet, featIndex, splitVal)
            if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN): continue
            newS = errType(mat0) + errType(mat1)
            if newS < bestS:
                bestIndex = featIndex
                bestValue = splitVal
                bestS = newS
    #if the decrease (S-bestS) is less than a threshold don't do the split
    if (S - bestS) < tolS:
        return None, leafType(dataSet) #exit cond 2
    mat0, mat1 = binSplitDataSet(dataSet, bestIndex, bestValue)
    if (shape(mat0)[0] < tolN) or (shape(mat1)[0] < tolN):    #exit cond 3
        return None, leafType(dataSet)
    return bestIndex,bestValue#returns the best feature to split on
                                      #and the value used for that split
```

```
def createTree(dataSet, leafType=regLeaf, errType=regErr, ops=(1,4)):#assume dataSet is NumPy
Mat so we can array filtering
    feat, val = chooseBestSplit(dataSet, leafType, errType, ops)#choose the best split
    if feat == None: return val #if the splitting hit a stop condition return val
    retTree = {}
    retTree['spInd'] = feat
    retTree['spVal'] = val
    lSet, rSet = binSplitDataSet(dataSet, feat, val)
    retTree['left'] = createTree(lSet, leafType, errType, ops)
    retTree['right'] = createTree(rSet, leafType, errType, ops)
    return retTree

def isTree(obj):
    return (type(obj).__name__=='dict')

def getMean(tree):
    if isTree(tree['right']): tree['right'] = getMean(tree['right'])
    if isTree(tree['left']): tree['left'] = getMean(tree['left'])
    return (tree['left']+tree['right'])/2.0

def prune(tree, testData):
    if shape(testData)[0] == 0: return getMean(tree) #if we have no test data collapse the tree
    if (isTree(tree['right']) or isTree(tree['left'])):#if the branches are not trees try to prune them
        lSet, rSet = binSplitDataSet(testData, tree['spInd'], tree['spVal'])
    if isTree(tree['left']): tree['left'] = prune(tree['left'], lSet)
    if isTree(tree['right']): tree['right'] =    prune(tree['right'], rSet)
    #if they are now both leafs, see if we can merge them
    if not isTree(tree['left']) and not isTree(tree['right']):
        lSet, rSet = binSplitDataSet(testData, tree['spInd'], tree['spVal'])
        errorNoMerge = sum(power(lSet[:,-1] - tree['left'],2)) +\
            sum(power(rSet[:,-1] - tree['right'],2))
        treeMean = (tree['left']+tree['right'])/2.0
        errorMerge = sum(power(testData[:,-1] - treeMean,2))
        if errorMerge < errorNoMerge:
            print "merging"
            return treeMean
        else: return tree
    else: return tree

def regTreeEval(model, inDat):
    return float(model)

def modelTreeEval(model, inDat):
    n = shape(inDat)[1]
```

```
    X = mat(ones((1,n+1)))
    X[:,1:n+1]=inDat
    return float(X*model)

def treeForeCast(tree, inData, modelEval=regTreeEval):
    if not isTree(tree): return modelEval(tree, inData)
    if inData[tree['spInd']] > tree['spVal']:
        if isTree(tree['left']): return treeForeCast(tree['left'], inData, modelEval)
        else: return modelEval(tree['left'], inData)
    else:
        if isTree(tree['right']): return treeForeCast(tree['right'], inData, modelEval)
        else: return modelEval(tree['right'], inData)

def createForeCast(tree, testData, modelEval=regTreeEval):
    m=len(testData)
    yHat = mat(zeros((m,1)))
    for i in range(m):
        yHat[i,0] = treeForeCast(tree, mat(testData[i]), modelEval)
    return yHat
```

9.2 treeExplore.py

```
    from numpy import *

    from Tkinter import *
    import regTrees

    import matplotlib
    matplotlib.use('TkAgg')
    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
    from matplotlib.figure import Figure

    def reDraw(tolS,tolN):
        reDraw.f.clf()              # clear the figure
        reDraw.a = reDraw.f.add_subplot(111)
        if chkBtnVar.get():
            if tolN < 2: tolN = 2
            myTree=regTrees.createTree(reDraw.rawDat, regTrees.modelLeaf,\
                                       regTrees.modelErr, (tolS,tolN))
            yHat = regTrees.createForeCast(myTree, reDraw.testDat, \
                                           regTrees.modelTreeEval)
        else:
            myTree=regTrees.createTree(reDraw.rawDat, ops=(tolS,tolN))
            yHat = regTrees.createForeCast(myTree, reDraw.testDat)
        reDraw.a.scatter(reDraw.rawDat[:,0], reDraw.rawDat[:,1], s=5) #use scatter for data set
        reDraw.a.plot(reDraw.testDat, yHat, linewidth=2.0) #use plot for yHat
```

```
        reDraw.canvas.show()

def getInputs():
        try: tolN = int(tolNentry.get())
        except:
                tolN = 10
                print "enter Integer for tolN"
                tolNentry.delete(0, END)
                tolNentry.insert(0,'10')
        try: tolS = float(tolSentry.get())
        except:
                tolS = 1.0
                print "enter Float for tolS"
                tolSentry.delete(0, END)
                tolSentry.insert(0,'1.0')
        return tolN,tolS

def drawNewTree():
        tolN,tolS = getInputs()#get values from Entry boxes
        reDraw(tolS,tolN)

root=Tk()

reDraw.f = Figure(figsize=(5,4), dpi=100) #create canvas
reDraw.canvas = FigureCanvasTkAgg(reDraw.f, master=root)
reDraw.canvas.show()
reDraw.canvas.get_tk_widget().grid(row=0, columnspan=3)

Label(root, text="tolN").grid(row=1, column=0)
tolNentry = Entry(root)
tolNentry.grid(row=1, column=1)
tolNentry.insert(0,'10')
Label(root, text="tolS").grid(row=2, column=0)
tolSentry = Entry(root)
tolSentry.grid(row=2, column=1)
tolSentry.insert(0,'1.0')
Button(root, text="ReDraw", command=drawNewTree).grid(row=1, column=2, rowspan=3)
chkBtnVar = IntVar()
chkBtn = Checkbutton(root, text="Model Tree", variable = chkBtnVar)
chkBtn.grid(row=3, column=0, columnspan=2)

reDraw.rawDat = mat(regTrees.loadDataSet('sine.txt'))
reDraw.testDat = arange(min(reDraw.rawDat[:,0]),max(reDraw.rawDat[:,0]),0.01)
reDraw(1.0, 10)
```

```
root.mainloop()
```

10 利用 K-均值聚类算法对未标注数据分组

10.1 kMeans.py

```
'''
Created on Feb 16, 2011
k Means Clustering for Ch10 of Machine Learning in Action
@author: Peter Harrington
'''
from numpy import *

def loadDataSet(fileName):          #general function to parse tab -delimited floats
    dataMat = []                    #assume last column is target value
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = map(float,curLine) #map all elements to float()
        dataMat.append(fltLine)
    return dataMat

def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2))) #la.norm(vecA-vecB)

def randCent(dataSet, k):
    n = shape(dataSet)[1]
    centroids = mat(zeros((k,n)))#create centroid mat
    for j in range(n):#create random cluster centers, within bounds of each dimension
        minJ = min(dataSet[:,j])
        rangeJ = float(max(dataSet[:,j]) - minJ)
        centroids[:,j] = mat(minJ + rangeJ * random.rand(k,1))
    return centroids

def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m,2)))#create mat to assign data points
                                      #to a centroid, also holds SE of each point
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):#for each data point assign it to the closest centroid
            minDist = inf; minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j,:],dataSet[i,:])
```

```
                    if distJI < minDist:
                        minDist = distJI; minIndex = j
                if clusterAssment[i,0] != minIndex: clusterChanged = True
                clusterAssment[i,:] = minIndex,minDist**2
            print centroids
            for cent in range(k):#recalculate centroids
                ptsInClust = dataSet[nonzero(clusterAssment[:,0].A==cent)[0]]#get all the point in this cluster
                centroids[cent,:] = mean(ptsInClust, axis=0) #assign centroid to mean
        return centroids, clusterAssment

    def biKmeans(dataSet, k, distMeas=distEclud):
        m = shape(dataSet)[0]
        clusterAssment = mat(zeros((m,2)))
        centroid0 = mean(dataSet, axis=0).tolist()[0]
        centList =[centroid0] #create a list with one centroid
        for j in range(m):#calc initial Error
            clusterAssment[j,1] = distMeas(mat(centroid0), dataSet[j,:])**2
        while (len(centList) < k):
            lowestSSE = inf
            for i in range(len(centList)):
                ptsInCurrCluster = dataSet[nonzero(clusterAssment[:,0].A==i)[0],:]#get the data points currently in cluster i
                centroidMat, splitClustAss = kMeans(ptsInCurrCluster, 2, distMeas)
                sseSplit = sum(splitClustAss[:,1])#compare the SSE to the currrent minimum
                sseNotSplit = sum(clusterAssment[nonzero(clusterAssment[:,0].A!=i)[0],1])
                print "sseSplit, and notSplit: ",sseSplit,sseNotSplit
                if (sseSplit + sseNotSplit) < lowestSSE:
                    bestCentToSplit = i
                    bestNewCents = centroidMat
                    bestClustAss = splitClustAss.copy()
                    lowestSSE = sseSplit + sseNotSplit
            bestClustAss[nonzero(bestClustAss[:,0].A == 1)[0],0] = len(centList) #change 1 to 3,4, or whatever
            bestClustAss[nonzero(bestClustAss[:,0].A == 0)[0],0] = bestCentToSplit
            print 'the bestCentToSplit is: ',bestCentToSplit
            print 'the len of bestClustAss is: ', len(bestClustAss)
            centList[bestCentToSplit] = bestNewCents[0,:].tolist()[0]#replace a centroid with two best centroids
            centList.append(bestNewCents[1,:].tolist()[0])
            clusterAssment[nonzero(clusterAssment[:,0].A == bestCentToSplit)[0],:]= bestClustAss#reassign new clusters, and SSE
        return mat(centList), clusterAssment
```

```
import urllib
import json
def geoGrab(stAddress, city):
    apiStem = 'http://where.yahooapis.com/geocode?'  #create a dict and constants for the goecoder
    params = {}
    params['flags'] = 'J'#JSON return type
    params['appid'] = 'aaa0VN6k'
    params['location'] = '%s %s' % (stAddress, city)
    url_params = urllib.urlencode(params)
    yahooApi = apiStem + url_params      #print url_params
    print yahooApi
    c=urllib.urlopen(yahooApi)
    return json.loads(c.read())

from time import sleep
def massPlaceFind(fileName):
    fw = open('places.txt', 'w')
    for line in open(fileName).readlines():
        line = line.strip()
        lineArr = line.split('\t')
        retDict = geoGrab(lineArr[1], lineArr[2])
        if retDict['ResultSet']['Error'] == 0:
            lat = float(retDict['ResultSet']['Results'][0]['latitude'])
            lng = float(retDict['ResultSet']['Results'][0]['longitude'])
            print "%s\t%f\t%f" % (lineArr[0], lat, lng)
            fw.write('%s\t%f\t%f\n' % (line, lat, lng))
        else: print "error fetching"
        sleep(1)
    fw.close()

def distSLC(vecA, vecB):#Spherical Law of Cosines
    a = sin(vecA[0,1]*pi/180) * sin(vecB[0,1]*pi/180)
    b = cos(vecA[0,1]*pi/180) * cos(vecB[0,1]*pi/180) * \
                        cos(pi * (vecB[0,0]-vecA[0,0]) /180)
    return arccos(a + b)*6371.0 #pi is imported with numpy

import matplotlib
import matplotlib.pyplot as plt
def clusterClubs(numClust=5):
    datList = []
    for line in open('places.txt').readlines():
        lineArr = line.split('\t')
        datList.append([float(lineArr[4]), float(lineArr[3])])
```

```
        datMat = mat(datList)
        myCentroids, clustAssing = biKmeans(datMat, numClust, distMeas=distSLC)
        fig = plt.figure()
        rect=[0.1,0.1,0.8,0.8]
        scatterMarkers=['s', 'o', '^', '8', 'p', \
                        'd', 'v', 'h', '>', '<']
        axprops = dict(xticks=[], yticks=[])
        ax0=fig.add_axes(rect, label='ax0', **axprops)
        imgP = plt.imread('Portland.png')
        ax0.imshow(imgP)
        ax1=fig.add_axes(rect, label='ax1', frameon=False)
        for i in range(numClust):
            ptsInCurrCluster = datMat[nonzero(clustAssing[:,0].A==i)[0],:]
            markerStyle = scatterMarkers[i % len(scatterMarkers)]
            ax1.scatter(ptsInCurrCluster[:,0].flatten().A[0], ptsInCurrCluster[:,1].flatten().A[0],
marker=markerStyle, s=90)
        ax1.scatter(myCentroids[:,0].flatten().A[0], myCentroids[:,1].flatten().A[0], marker='+',
s=300)
        plt.show()
```

11 使用 Apriori 算法进行关联分析

11.1 apriori.py

```
'''
Created on Mar 24, 2011
Ch 11 code
@author: Peter
'''
from numpy import *

def loadDataSet():
    return [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]

def createC1(dataSet):
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])

    C1.sort()
    return map(frozenset, C1)#use frozen set so we
                             #can use it as a key in a dict

def scanD(D, Ck, minSupport):
    ssCnt = {}
```

```
        for tid in D:
            for can in Ck:
                if can.issubset(tid):
                    if not ssCnt.has_key(can): ssCnt[can]=1
                    else: ssCnt[can] += 1
        numItems = float(len(D))
        retList = []
        supportData = {}
        for key in ssCnt:
            support = ssCnt[key]/numItems
            if support >= minSupport:
                retList.insert(0,key)
            supportData[key] = support
        return retList, supportData

    def aprioriGen(Lk, k): #creates Ck
        retList = []
        lenLk = len(Lk)
        for i in range(lenLk):
            for j in range(i+1, lenLk):
                L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]
                L1.sort(); L2.sort()
                if L1==L2: #if first k-2 elements are equal
                    retList.append(Lk[i] | Lk[j]) #set union
        return retList

    def apriori(dataSet, minSupport = 0.5):
        C1 = createC1(dataSet)
        D = map(set, dataSet)
        L1, supportData = scanD(D, C1, minSupport)
        L = [L1]
        k = 2
        while (len(L[k-2]) > 0):
            Ck = aprioriGen(L[k-2], k)
            Lk, supK = scanD(D, Ck, minSupport)#scan DB to get Lk
            supportData.update(supK)
            L.append(Lk)
            k += 1
        return L, supportData

    def generateRules(L, supportData, minConf=0.7):    #supportData is a dict coming from scanD
        bigRuleList = []
        for i in range(1, len(L)):#only get the sets with two or more items
            for freqSet in L[i]:
```

```
            H1 = [frozenset([item]) for item in freqSet]
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    prunedH = [] #create new list to return
    for conseq in H:
        conf = supportData[freqSet]/supportData[freqSet-conseq] #calc confidence
        if conf >= minConf:
            print freqSet-conseq,'-->',conseq,'conf:',conf
            brl.append((freqSet-conseq, conseq, conf))
            prunedH.append(conseq)
    return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    m = len(H[0])
    if (len(freqSet) > (m + 1)): #try further merging
        Hmp1 = aprioriGen(H, m+1)#create Hm+1 new candidates
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if (len(Hmp1) > 1):    #need at least two sets to merge
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)

def pntRules(ruleList, itemMeaning):
    for ruleTup in ruleList:
        for item in ruleTup[0]:
            print itemMeaning[item]
        print "                   -------->"
        for item in ruleTup[1]:
            print itemMeaning[item]
        print "confidence: %f" % ruleTup[2]
        print            #print a blank line


from time import sleep
from votesmart import votesmart
votesmart.apikey = 'a7fa40adec6f4a77178799fae4441030'
#votesmart.apikey = 'get your api key first'
def getActionIds():
    actionIdList = []; billTitleList = []
    fr = open('recent20bills.txt')
    for line in fr.readlines():
```

```
            billNum = int(line.split('\t')[0])
            try:
                billDetail = votesmart.votes.getBill(billNum) #api call
                for action in billDetail.actions:
                    if action.level == 'House' and \
                    (action.stage == 'Passage' or action.stage == 'Amendment Vote'):
                        actionId = int(action.actionId)
                        print 'bill: %d has actionId: %d' % (billNum, actionId)
                        actionIdList.append(actionId)
                        billTitleList.append(line.strip().split('\t')[1])
            except:
                print "problem getting bill %d" % billNum
            sleep(1)                                                    #delay to be polite
        return actionIdList, billTitleList

def getTransList(actionIdList, billTitleList): #this will return a list of lists containing ints
        itemMeaning = ['Republican', 'Democratic']#list of what each item stands for
        for billTitle in billTitleList:#fill up itemMeaning list
            itemMeaning.append('%s -- Nay' % billTitle)
            itemMeaning.append('%s -- Yea' % billTitle)
        transDict = {}#list of items in each transaction (politician)
        voteCount = 2
        for actionId in actionIdList:
            sleep(3)
            print 'getting votes for actionId: %d' % actionId
            try:
                voteList = votesmart.votes.getBillActionVotes(actionId)
                for vote in voteList:
                    if not transDict.has_key(vote.candidateName):
                        transDict[vote.candidateName] = []
                        if vote.officeParties == 'Democratic':
                            transDict[vote.candidateName].append(1)
                        elif vote.officeParties == 'Republican':
                            transDict[vote.candidateName].append(0)
                    if vote.action == 'Nay':
                        transDict[vote.candidateName].append(voteCount)
                    elif vote.action == 'Yea':
                        transDict[vote.candidateName].append(voteCount + 1)
            except:
                print "problem getting actionId: %d" % actionId
            voteCount += 2
        return transDict, itemMeaning
```

12 使用 FP-growth 算法来高效分析频繁项集

12.1 fpGrowth.py

```
'''
Created on Jun 14, 2011
FP-Growth FP means frequent pattern
the FP-Growth algorithm needs:
1. FP-tree (class treeNode)
2. header table (use dict)

This finds frequent itemsets similar to apriori but does not
find association rules.

@author: Peter
'''
class treeNode:
    def __init__(self, nameValue, numOccur, parentNode):
        self.name = nameValue
        self.count = numOccur
        self.nodeLink = None
        self.parent = parentNode      #needs to be updated
        self.children = {}

    def inc(self, numOccur):
        self.count += numOccur

    def disp(self, ind=1):
        print '  '*ind, self.name, ' ', self.count
        for child in self.children.values():
            child.disp(ind+1)

def createTree(dataSet, minSup=1): #create FP-tree from dataset but don't mine
    headerTable = {}
    #go over dataSet twice
    for trans in dataSet:#first pass counts frequency of occurance
        for item in trans:
            headerTable[item] = headerTable.get(item, 0) + dataSet[trans]
    for k in headerTable.keys():  #remove items not meeting minSup
        if headerTable[k] < minSup:
            del(headerTable[k])
    freqItemSet = set(headerTable.keys())
    #print 'freqItemSet: ',freqItemSet
    if len(freqItemSet) == 0: return None, None  #if no items meet min support -->get out
    for k in headerTable:
        headerTable[k] = [headerTable[k], None] #reformat headerTable to use Node link
```

```
        #print 'headerTable: ',headerTable
        retTree = treeNode('Null Set', 1, None) #create tree
        for tranSet, count in dataSet.items():    #go through dataset 2nd time
            localID = {}
            for item in tranSet:    #put transaction items in order
                if item in freqItemSet:
                    localID[item] = headerTable[item][0]
            if len(localID) > 0:
                orderedItems = [v[0] for v in sorted(localID.items(), key=lambda p: p[1], reverse=True)]
                updateTree(orderedItems, retTree, headerTable, count)#populate tree with ordered freq itemset
        return retTree, headerTable #return tree and header table

    def updateTree(items, inTree, headerTable, count):
        if items[0] in inTree.children:#check if orderedItems[0] in retTree.children
            inTree.children[items[0]].inc(count) #increment count
        else:     #add items[0] to inTree.children
            inTree.children[items[0]] = treeNode(items[0], count, inTree)
            if headerTable[items[0]][1] == None: #update header table
                headerTable[items[0]][1] = inTree.children[items[0]]
            else:
                updateHeader(headerTable[items[0]][1], inTree.children[items[0]])
        if len(items) > 1:#call updateTree() with remaining ordered items
            updateTree(items[1::], inTree.children[items[0]], headerTable, count)

    def updateHeader(nodeToTest, targetNode):     #this version does not use recursion
        while (nodeToTest.nodeLink != None):      #Do not use recursion to traverse a linked list!
            nodeToTest = nodeToTest.nodeLink
        nodeToTest.nodeLink = targetNode

    def ascendTree(leafNode, prefixPath): #ascends from leaf node to root
        if leafNode.parent != None:
            prefixPath.append(leafNode.name)
            ascendTree(leafNode.parent, prefixPath)

    def findPrefixPath(basePat, treeNode): #treeNode comes from header table
        condPats = {}
        while treeNode != None:
            prefixPath = []
            ascendTree(treeNode, prefixPath)
            if len(prefixPath) > 1:
                condPats[frozenset(prefixPath[1:])] = treeNode.count
```

```
            treeNode = treeNode.nodeLink
        return condPats

    def mineTree(inTree, headerTable, minSup, preFix, freqItemList):
        bigL = [v[0] for v in sorted(headerTable.items(), key=lambda p: p[1])]#(sort header table)
        for basePat in bigL:    #start from bottom of header table
            newFreqSet = preFix.copy()
            newFreqSet.add(basePat)
            #print 'finalFrequent Item: ',newFreqSet        #append to set
            freqItemList.append(newFreqSet)
            condPattBases = findPrefixPath(basePat, headerTable[basePat][1])
            #print 'condPattBases :',basePat, condPattBases
            #2. construct cond FP-tree from cond. pattern base
            myCondTree, myHead = createTree(condPattBases, minSup)
            #print 'head from conditional tree: ', myHead
            if myHead != None: #3. mine cond. FP-tree
                #print 'conditional tree for: ',newFreqSet
                #myCondTree.disp(1)
                mineTree(myCondTree, myHead, minSup, newFreqSet, freqItemList)

    def loadSimpDat():
        simpDat = [['r', 'z', 'h', 'j', 'p'],
                   ['z', 'y', 'x', 'w', 'v', 'u', 't', 's'],
                   ['z'],
                   ['r', 'x', 'n', 'o', 's'],
                   ['y', 'r', 'x', 'z', 'q', 't', 'p'],
                   ['y', 'z', 'x', 'e', 'q', 's', 't', 'm']]
        return simpDat

    def createInitSet(dataSet):
        retDict = {}
        for trans in dataSet:
            retDict[frozenset(trans)] = 1
        return retDict

    import twitter
    from time import sleep
    import re

    def textParse(bigString):
        urlsRemoved = re.sub('(http:[/][/]|www.)([a-z]|[A-Z]|[0-9]|[/.]|[~])*', '', bigString)
        listOfTokens = re.split(r'\W*', urlsRemoved)
        return [tok.lower() for tok in listOfTokens if len(tok) > 2]
```

```
def getLotsOfTweets(searchStr):
    CONSUMER_KEY = ''
    CONSUMER_SECRET = ''
    ACCESS_TOKEN_KEY = ''
    ACCESS_TOKEN_SECRET = ''
    api = twitter.Api(consumer_key=CONSUMER_KEY,
consumer_secret=CONSUMER_SECRET,
                      access_token_key=ACCESS_TOKEN_KEY,
                      access_token_secret=ACCESS_TOKEN_SECRET)
    #you can get 1500 results 15 pages * 100 per page
    resultsPages = []
    for i in range(1,15):
        print "fetching page %d" % i
        searchResults = api.GetSearch(searchStr, per_page=100, page=i)
        resultsPages.append(searchResults)
        sleep(6)
    return resultsPages

def mineTweets(tweetArr, minSup=5):
    parsedList = []
    for i in range(14):
        for j in range(100):
            parsedList.append(textParse(tweetArr[i][j].text))
    initSet = createInitSet(parsedList)
    myFPtree, myHeaderTab = createTree(initSet, minSup)
    myFreqList = []
    mineTree(myFPtree, myHeaderTab, minSup, set([]), myFreqList)
    return myFreqList

#minSup = 3
#simpDat = loadSimpDat()
#initSet = createInitSet(simpDat)
#myFPtree, myHeaderTab = createTree(initSet, minSup)
#myFPtree.disp()
#myFreqList = []
#mineTree(myFPtree, myHeaderTab, minSup, set([]), myFreqList)
```

13 利用 PCA 来简化数据

13.1 pca.py

```
'''
Created on Jun 1, 2011

@author: Peter Harrington
'''
```

```
from numpy import *

def loadDataSet(fileName, delim='\t'):
    fr = open(fileName)
    stringArr = [line.strip().split(delim) for line in fr.readlines()]
    datArr = [map(float,line) for line in stringArr]
    return mat(datArr)

def pca(dataMat, topNfeat=9999999):
    meanVals = mean(dataMat, axis=0)
    meanRemoved = dataMat - meanVals #remove mean
    covMat = cov(meanRemoved, rowvar=0)
    eigVals,eigVects = linalg.eig(mat(covMat))
    eigValInd = argsort(eigVals)            #sort, sort goes smallest to largest
    eigValInd = eigValInd[:-(topNfeat+1):-1]  #cut off unwanted dimensions
    redEigVects = eigVects[:,eigValInd]       #reorganize eig vects largest to smallest
    lowDDataMat = meanRemoved * redEigVects#transform data into new dimensions
    reconMat = (lowDDataMat * redEigVects.T) + meanVals
    return lowDDataMat, reconMat

def replaceNanWithMean():
    datMat = loadDataSet('secom.data', ' ')
    numFeat = shape(datMat)[1]
    for i in range(numFeat):
        meanVal = mean(datMat[nonzero(~isnan(datMat[:,i].A))[0],i]) #values that are not
NaN (a number)
        datMat[nonzero(isnan(datMat[:,i].A))[0],i] = meanVal  #set NaN values to mean
    return datMat
```

13.2 createFig1.py

```
'''
Created on Jun 1, 2011

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt

n = 1000 #number of points to create
xcord0 = []
ycord0 = []
xcord1 = []
ycord1 = []
markers =[]
```

```
colors =[]
fw = open('testSet.txt','w')
for i in range(n):
    [r0,r1] = random.standard_normal(2)
    fFlyer = r0 + 9.0
    tats = 1.0*r1 + fFlyer + 0
    xcord0.append(fFlyer)
    ycord0.append(tats)
    fw.write("%f\t%f\n" % (fFlyer, tats))

fw.close()
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xcord0,ycord0, marker='^', s=90)
plt.xlabel('hours of direct sunlight')
plt.ylabel('liters of water')
plt.show()
```

13.2 createFig2.py

```
'''
Created on Jun 1, 2011

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
import pca

dataMat = pca.loadDataSet('testSet.txt')
lowDMat, reconMat = pca.pca(dataMat, 1)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(dataMat[:,0], dataMat[:,1], marker='^', s=90)
ax.scatter(reconMat[:,0], reconMat[:,1], marker='o', s=50, c='red')
plt.show()
```

13.3 createFig3

```
'''
Created on Jun 1, 2011

@author: Peter
'''
from numpy import *
import matplotlib
```

```
import matplotlib.pyplot as plt
import pca

n = 1000 #number of points to create
xcord0 = []; ycord0 = []
xcord1 = []; ycord1 = []
xcord2 = []; ycord2 = []
markers =[]
colors =[]
fw = open('testSet3.txt','w')
for i in range(n):
        groupNum = int(3*random.uniform())
        [r0,r1] = random.standard_normal(2)
        if groupNum == 0:
                x = r0 + 16.0
                y = 1.0*r1 + x
                xcord0.append(x)
                ycord0.append(y)
        elif groupNum == 1:
                x = r0 + 8.0
                y = 1.0*r1 + x
                xcord1.append(x)
                ycord1.append(y)
        elif groupNum == 2:
                x = r0 + 0.0
                y = 1.0*r1 + x
                xcord2.append(x)
                ycord2.append(y)
        fw.write("%f\t%f\t%d\n" % (x, y, groupNum))

fw.close()
fig = plt.figure()
ax = fig.add_subplot(211)
ax.scatter(xcord0,ycord0, marker='^', s=90)
ax.scatter(xcord1,ycord1, marker='o', s=50,    c='red')
ax.scatter(xcord2,ycord2, marker='v', s=50,    c='yellow')
ax = fig.add_subplot(212)
myDat = pca.loadDataSet('testSet3.txt')
lowDDat,reconDat = pca.pca(myDat[:,0:2],1)
label0Mat = lowDDat[nonzero(myDat[:,2]==0)[0],:2][0] #get the items with label 0
label1Mat = lowDDat[nonzero(myDat[:,2]==1)[0],:2][0] #get the items with label 1
label2Mat = lowDDat[nonzero(myDat[:,2]==2)[0],:2][0] #get the items with label 2
#ax.scatter(label0Mat[:,0],label0Mat[:,1], marker='^', s=90)
#ax.scatter(label1Mat[:,0],label1Mat[:,1], marker='o', s=50,    c='red')
```

```
#ax.scatter(label2Mat[:,0],label2Mat[:,1], marker='v', s=50,    c='yellow')
ax.scatter(label0Mat[:,0],zeros(shape(label0Mat)[0]), marker='^', s=90)
ax.scatter(label1Mat[:,0],zeros(shape(label1Mat)[0]), marker='o', s=50,    c='red')
ax.scatter(label2Mat[:,0],zeros(shape(label2Mat)[0]), marker='v', s=50,    c='yellow')
plt.show()
```

13.4 createFig4.py

```
'''
Created on Jun 14, 2011

@author: Peter
'''
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
import pca

dataMat = pca.replaceNanWithMean()

#below is a quick hack copied from pca.pca()
meanVals = mean(dataMat, axis=0)
meanRemoved = dataMat - meanVals #remove mean
covMat = cov(meanRemoved, rowvar=0)
eigVals,eigVects = linalg.eig(mat(covMat))
eigValInd = argsort(eigVals)                    #sort, sort goes smallest to largest
eigValInd = eigValInd[::-1]#reverse
sortedEigVals = eigVals[eigValInd]
total = sum(sortedEigVals)
varPercentage = sortedEigVals/total*100

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(range(1, 21), varPercentage[:20], marker='^')
plt.xlabel('Principal Component Number')
plt.ylabel('Percentage of Variance')
plt.show()
```

14 利用 SVD 简化数据

14.1 svdRec.py

```
'''
Created on Mar 8, 2011

@author: Peter
'''
from numpy import *
from numpy import linalg as la
```

```
def loadExData():
    return[[0, 0, 0, 2, 2],
           [0, 0, 0, 3, 3],
           [0, 0, 0, 1, 1],
           [1, 1, 1, 0, 0],
           [2, 2, 2, 0, 0],
           [5, 5, 5, 0, 0],
           [1, 1, 1, 0, 0]]

def loadExData2():
    return[[0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 5],
           [0, 0, 0, 3, 0, 4, 0, 0, 0, 0, 3],
           [0, 0, 0, 0, 4, 0, 0, 1, 0, 4, 0],
           [3, 3, 4, 0, 0, 0, 0, 2, 2, 0, 0],
           [5, 4, 5, 0, 0, 0, 0, 5, 5, 0, 0],
           [0, 0, 0, 0, 5, 0, 1, 0, 0, 5, 0],
           [4, 3, 4, 0, 0, 0, 0, 5, 5, 0, 1],
           [0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 4],
           [0, 0, 0, 2, 0, 2, 5, 0, 0, 1, 2],
           [0, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0],
           [1, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0]]

def ecludSim(inA,inB):
    return 1.0/(1.0 + la.norm(inA - inB))

def pearsSim(inA,inB):
    if len(inA) < 3 : return 1.0
    return 0.5+0.5*corrcoef(inA, inB, rowvar = 0)[0][1]

def cosSim(inA,inB):
    num = float(inA.T*inB)
    denom = la.norm(inA)*la.norm(inB)
    return 0.5+0.5*(num/denom)

def standEst(dataMat, user, simMeas, item):
    n = shape(dataMat)[1]
    simTotal = 0.0; ratSimTotal = 0.0
    for j in range(n):
        userRating = dataMat[user,j]
        if userRating == 0: continue
        overLap = nonzero(logical_and(dataMat[:,item].A>0, \
                                      dataMat[:,j].A>0))[0]
        if len(overLap) == 0: similarity = 0
```

```
            else: similarity = simMeas(dataMat[overLap,item], \
                                       dataMat[overLap,j])
            print 'the %d and %d similarity is: %f' % (item, j, similarity)
            simTotal += similarity
            ratSimTotal += similarity * userRating
        if simTotal == 0: return 0
        else: return ratSimTotal/simTotal

def svdEst(dataMat, user, simMeas, item):
        n = shape(dataMat)[1]
        simTotal = 0.0; ratSimTotal = 0.0
        U,Sigma,VT = la.svd(dataMat)
        Sig4 = mat(eye(4)*Sigma[:4]) #arrange Sig4 into a diagonal matrix
        xformedItems = dataMat.T * U[:,:4] * Sig4.I    #create transformed items
        for j in range(n):
            userRating = dataMat[user,j]
            if userRating == 0 or j==item: continue
            similarity = simMeas(xformedItems[item,:].T,\
                                 xformedItems[j,:].T)
            print 'the %d and %d similarity is: %f' % (item, j, similarity)
            simTotal += similarity
            ratSimTotal += similarity * userRating
        if simTotal == 0: return 0
        else: return ratSimTotal/simTotal

def recommend(dataMat, user, N=3, simMeas=cosSim, estMethod=standEst):
        unratedItems = nonzero(dataMat[user,:].A==0)[1]#find unrated items
        if len(unratedItems) == 0: return 'you rated everything'
        itemScores = []
        for item in unratedItems:
            estimatedScore = estMethod(dataMat, user, simMeas, item)
            itemScores.append((item, estimatedScore))
        return sorted(itemScores, key=lambda jj: jj[1], reverse=True)[:N]

def printMat(inMat, thresh=0.8):
        for i in range(32):
            for k in range(32):
                if float(inMat[i,k]) > thresh:
                    print 1,
                else: print 0,
            print ''

def imgCompress(numSV=3, thresh=0.8):
        myl = []
```

```
    for line in open('0_5.txt').readlines():
        newRow = []
        for i in range(32):
            newRow.append(int(line[i]))
        myl.append(newRow)
    myMat = mat(myl)
    print "****original matrix******"
    printMat(myMat, thresh)
    U,Sigma,VT = la.svd(myMat)
    SigRecon = mat(zeros((numSV, numSV)))
    for k in range(numSV):#construct diagonal matrix from vector
        SigRecon[k,k] = Sigma[k]
    reconMat = U[:,:numSV]*SigRecon*VT[:numSV,:]
    print "****reconstructed matrix using %d singular values******" % numSV
    printMat(reconMat, thresh)
```

15 大数据与 MapReduce

15.1 mrMean.py

```
'''
Created on Feb 28, 2011

@author: Peter
'''
from mrjob.job import MRJob

class MRmean(MRJob):
    def __init__(self, *args, **kwargs):
        super(MRmean, self).__init__(*args, **kwargs)
        self.inCount = 0
        self.inSum = 0
        self.inSqSum = 0

    def map(self, key, val): #needs exactly 2 arguments
        if False: yield
        inVal = float(val)
        self.inCount += 1
        self.inSum += inVal
        self.inSqSum += inVal*inVal

    def map_final(self):
        mn = self.inSum/self.inCount
        mnSq = self.inSqSum/self.inCount
        yield (1, [self.inCount, mn, mnSq])

    def reduce(self, key, packedValues):
```

```
        cumVal=0.0; cumSumSq=0.0; cumN=0.0
        for valArr in packedValues: #get values from streamed inputs
            nj = float(valArr[0])
            cumN += nj
            cumVal += nj*float(valArr[1])
            cumSumSq += nj*float(valArr[2])
        mean = cumVal/cumN
        var = (cumSumSq - 2*mean*cumVal + cumN*mean*mean)/cumN
        yield (mean, var) #emit mean and var

    def steps(self):
        return ([self.mr(mapper=self.map, mapper_final=self.map_final,\
                                reducer=self.reduce,)])

if __name__ == '__main__':
    MRmean.run()
```

15.2 mrMeanMapper.py

```
'''
Created on Feb 21, 2011
Machine Learning in Action Chapter 18
Map Reduce Job for Hadoop Streaming
mrMeanMapper.py
@author: Peter Harrington
'''
import sys
from numpy import mat, mean, power

def read_input(file):
    for line in file:
        yield line.rstrip()

input = read_input(sys.stdin)#creates a list of input lines
input = [float(line) for line in input] #overwrite with floats
numInputs = len(input)
input = mat(input)
sqInput = power(input,2)

#output size, mean, mean(square values)
print "%d\t%f\t%f" % (numInputs, mean(input), mean(sqInput)) #calc mean of columns
print >> sys.stderr, "report: still alive"
```

15.3 mrMeanReducer.py

```
'''
Created on Feb 21, 2011
```

```
@author: Peter
'''
import sys
from numpy import mat, mean, power

def read_input(file):
    for line in file:
        yield line.rstrip()

input = read_input(sys.stdin)#creates a list of input lines

#split input lines into separate items and store in list of lists
mapperOut = [line.split('\t') for line in input]

#accumulate total number of samples, overall sum and overall sum sq
cumVal=0.0
cumSumSq=0.0
cumN=0.0
for instance in mapperOut:
    nj = float(instance[0])
    cumN += nj
    cumVal += nj*float(instance[1])
    cumSumSq += nj*float(instance[2])

#calculate means
mean = cumVal/cumN
meanSq = cumSumSq/cumN

#output size, mean, mean(square values)
print "%d\t%f\t%f" % (cumN, mean, meanSq)
print >> sys.stderr, "report: still alive"
```

15.4 mrSVM.py

```
'''
Created on Feb 27, 2011
MapReduce version of Pegasos SVM
Using mrjob to automate job flow
@author: Peter
'''
from mrjob.job import MRJob

import pickle
from numpy import *

class MRsvm(MRJob):
```

```
    DEFAULT_INPUT_PROTOCOL = 'json_value'

    def __init__(self, *args, **kwargs):
        super(MRsvm, self).__init__(*args, **kwargs)
        self.data =
pickle.load(open('C:\Users\Peter\machinelearninginaction\Ch15\svmDat27'))
        self.w = 0
        self.eta = 0.69
        self.dataList = []
        self.k = self.options.batchsize
        self.numMappers = 1
        self.t = 1  #iteration number

    def configure_options(self):
        super(MRsvm, self).configure_options()
        self.add_passthrough_option(
            '--iterations', dest='iterations', default=2, type='int',
            help='T: number of iterations to run')
        self.add_passthrough_option(
            '--batchsize', dest='batchsize', default=100, type='int',
            help='k: number of data points in a batch')

    def map(self, mapperId, inVals): #needs exactly 2 arguments
        #input: nodeId, ('w', w-vector) OR nodeId, ('x', int)
        if False: yield
        if inVals[0]=='w':                      #accumulate W-vector
            self.w = inVals[1]
        elif inVals[0]=='x':
            self.dataList.append(inVals[1])#accumulate data points to calc
        elif inVals[0]=='t': self.t = inVals[1]
        else: self.eta=inVals #this is for debug, eta not used in map

    def map_fin(self):
        labels = self.data[:,-1]; X=self.data[:,0:-1]#reshape data into X and Y
        if self.w == 0: self.w = [0.001]*shape(X)[1] #init w on first iteration
        for index in self.dataList:
            p = mat(self.w)*X[index,:].T #calc p=w*dataSet[key].T
            if labels[index]*p < 1.0:
                yield (1, ['u', index])#make sure everything has the same key
        yield (1, ['w', self.w])          #so it ends up at the same reducer
        yield (1, ['t', self.t])

    def reduce(self, _, packedVals):
        for valArr in packedVals: #get values from streamed inputs
```

```
            if valArr[0]=='u':    self.dataList.append(valArr[1])
            elif valArr[0]=='w': self.w = valArr[1]
            elif valArr[0]=='t':    self.t = valArr[1]
        labels = self.data[:,-1]; X=self.data[:,0:-1]
        wMat = mat(self.w);      wDelta = mat(zeros(len(self.w)))
        for index in self.dataList:
            wDelta += float(labels[index])*X[index,:] #wDelta += label*dataSet
        eta = 1.0/(2.0*self.t)             #calc new: eta
        #calc new: w = (1.0 - 1/t)*w + (eta/k)*wDelta
        wMat = (1.0 - 1.0/self.t)*wMat + (eta/self.k)*wDelta
        for mapperNum in range(1,self.numMappers+1):
            yield (mapperNum, ['w', wMat.tolist()[0] ]) #emit w
            if self.t < self.options.iterations:
                yield (mapperNum, ['t', self.t+1])#increment T
                for j in range(self.k/self.numMappers):#emit random ints for mappers iid
                    yield (mapperNum, ['x', random.randint(shape(self.data)[0]) ])

    def steps(self):
        return ([self.mr(mapper=self.map, reducer=self.reduce,
                         mapper_final=self.map_fin)]*self.options.iterations)

if __name__ == '__main__':
    MRsvm.run()
```

15.5 mrSVMkickStart.py

```
'''
Created on Feb 27, 2011

@author: Peter
'''
from mrjob.protocol import JSONProtocol
from numpy import *

fw=open('kickStart2.txt', 'w')
for i in [1]:
    for j in range(100):
        fw.write('["x", %d]\n' % random.randint(200))
fw.close()
```

15.6 pegasos.py

```
'''
Created on Feb 24, 2011
Sequential Pegasos
the input T is k*T in Batch Pegasos
@author: Peter Harrington
'''
```

```
from numpy import *

def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():
        lineArr = line.strip().split('\t')
        #dataMat.append([float(lineArr[0]), float(lineArr[1]), float(lineArr[2])])
        dataMat.append([float(lineArr[0]), float(lineArr[1])])
        labelMat.append(float(lineArr[2]))
    return dataMat,labelMat

def seqPegasos(dataSet, labels, lam, T):
    m,n = shape(dataSet); w = zeros(n)
    for t in range(1, T+1):
        i = random.randint(m)
        eta = 1.0/(lam*t)
        p = predict(w, dataSet[i,:])
        if labels[i]*p < 1:
            w = (1.0 - 1/t)*w + eta*labels[i]*dataSet[i,:]
        else:
            w = (1.0 - 1/t)*w
        print w
    return w

def predict(w, x):
    return w*x.T

def batchPegasos(dataSet, labels, lam, T, k):
    m,n = shape(dataSet); w = zeros(n);
    dataIndex = range(m)
    for t in range(1, T+1):
        wDelta = mat(zeros(n)) #reset wDelta
        eta = 1.0/(lam*t)
        random.shuffle(dataIndex)
        for j in range(k):#go over training set
            i = dataIndex[j]
            p = predict(w, dataSet[i,:])            #mapper code
            if labels[i]*p < 1:                     #mapper code
                wDelta += labels[i]*dataSet[i,:].A #accumulate changes
        w = (1.0 - 1/t)*w + (eta/k)*wDelta          #apply changes at each T
    return w

datArr,labelList = loadDataSet('testSet.txt')
```

```
datMat = mat(datArr)
#finalWs = seqPegasos(datMat, labelList, 2, 5000)
finalWs = batchPegasos(datMat, labelList, 2, 50, 100)
print finalWs

import matplotlib
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
x1=[]; y1=[]; xm1=[]; ym1=[]
for i in range(len(labelList)):
    if labelList[i] == 1.0:
        x1.append(datMat[i,0]); y1.append(datMat[i,1])
    else:
        xm1.append(datMat[i,0]); ym1.append(datMat[i,1])
ax.scatter(x1, y1, marker='s', s=90)
ax.scatter(xm1, ym1, marker='o', s=50, c='red')
x = arange(-6.0, 8.0, 0.1)
y = (-finalWs[0,0]*x - 0)/finalWs[0,1]
#y2 = (0.43799*x)/0.12316
y2 = (0.498442*x)/0.092387 #2 iterations
ax.plot(x,y)
ax.plot(x,y2,'g-.')
ax.axis([-6,8,-4,5])
ax.legend(('50 Iterations', '2 Iterations') )
plt.show()
```

15.7 proximalSVM.py

```
'''
Created on Feb 25, 2011

@author: Peter
'''
import numpy

def map(key, value):
    # input key= class for one training example, e.g. "-1.0"
    classes = [float(item) for item in key.split(",")]      # e.g. [-1.0]
    D = numpy.diag(classes)

    # input value = feature vector for one training example, e.g. "3.0, 7.0, 2.0"
    featurematrix = [float(item) for item in value.split(",")]
    A = numpy.matrix(featurematrix)

    # create matrix E and vector e
```

```
        e = numpy.matrix(numpy.ones(len(A)).reshape(len(A),1))
        E = numpy.matrix(numpy.append(A,-e,axis=1))

        # create a tuple with the values to be used by reducer
        # and encode it with base64 to avoid potential trouble with '\t' and '\n' used
        # as default separators in Hadoop Streaming
        producedvalue = base64.b64encode(pickle.dumps( (E.T*E, E.T*D*e) )

        # note: a single constant key "producedkey" sends to only one reducer
        # somewhat "atypical" due to low degree of parallism on reducer side
        print "producedkey\t%s" % (producedvalue)

    def reduce(key, values, mu=0.1):
        sumETE = None
        sumETDe = None

        # key isn't used, so ignoring it with _ (underscore).
        for _, value in values:
            # unpickle values
            ETE, ETDe = pickle.loads(base64.b64decode(value))
            if sumETE == None:
                # create the I/mu with correct dimensions
                sumETE = numpy.matrix(numpy.eye(ETE.shape[1])/mu)
            sumETE += ETE

            if sumETDe == None:
                # create sumETDe with correct dimensions
                sumETDe = ETDe
            else:
                sumETDe += ETDe

            # note: omega = result[:-1] and gamma = result[-1]
            # but printing entire vector as output
            result = sumETE.I*sumETDe
            print "%s\t%s" % (key, str(result.tolist()))
```

15.8 py27dbg.py

```
    '''
    Created on Feb 27, 2011
    MapReduce version of Pegasos SVM
    Using mrjob to automate job flow
    @author: Peter
    '''
    from mrjob.job import MRJob
```

```
import pickle
from numpy import *

class MRsvm(MRJob):

    def map(self, mapperId, inVals): #needs exactly 2 arguments
        if False: yield
        yield (1, 22)

    def reduce(self, _, packedVals):
        yield "fuck ass"

    def steps(self):
        return ([self.mr(mapper=self.map, reducer=self.reduce)])

if __name__ == '__main__':
    MRsvm.run()
```

15.9 wc.py

```
from mrjob.job import MRJob
import json


class MRWordCountUtility(MRJob):

    def __init__(self, *args, **kwargs):
        super(MRWordCountUtility, self).__init__(*args, **kwargs)
        self.chars = 0
        self.words = 0
        self.lines = 0

    def mapper(self, _, line):
        if False:
            yield    # I'm a generator!

        self.chars += len(line) + 1    # +1 for newline
        self.words += sum(1 for word in line.split() if word.strip())
        self.lines += 1

    def mapper_final(self):
        yield('chars', self.chars)
        yield('words', self.words)
        yield('lines', self.lines)

    def reducer(self, key, values):
```

```
            yield(key, sum(values))


if __name__ == '__main__':
    MRWordCountUtility.run()
```