def trainNB0(trainMatrix, trainCategory):

# 总email数

numTrainDocs = len(trainMatrix)

# 总变量数

numWords = len(trainMatrix[0])

# 侮辱性文件的出现概率

pAbusive = sum(trainCategory) / float(numTrainDocs)

# 构造单词出现次数列表

# p0Num 正常的统计

# p1Num 侮辱的统计

p0Num = np.ones(numWords)

p1Num = np.ones(numWords)

# 整个数据集单词出现总数，2.0根据样本/实际调查结果调整分母的值（2主要是避免分母为0，当然值可以调整）

# p0Denom 正常的统计

# p1Denom 侮辱的统计

p0Denom = 2.0

p1Denom = 2.0

for i in range(numTrainDocs):

if trainCategory[i] == 1:

# 累加辱骂词的频次

p1Num += trainMatrix[i]

# 对每篇文章的辱骂的频次 进行统计汇总

p1Denom += sum(trainMatrix[i])

else:

p0Num += trainMatrix[i]

p0Denom += sum(trainMatrix[i])

# 类别1，即spam的[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]列表

p1Vect = np.log(p1Num / p1Denom)

# 类别0，即ham的[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]列表

p0Vect = np.log(p0Num / p0Denom)

return p0Vect, p1Vect, pAbusive

def classifyNB(testk, p0Vec, p1Vec, pClass1):

# 计算公式 log(P(F1|C))+log(P(F2|C))+....+log(P(Fn|C))+log(P(C))

# 上面的计算公式，没有除以贝叶斯准则的公式的分母，也就是 P(w) （P(w) 指的是此文档在所有的文档中出现的概率）就进行概率大小的比较了，

# 因为 P(w) 针对的是包含侮辱和非侮辱的全部文档，所以 P(w) 是相同的。

# 使用 NumPy 数组来计算两个向量相乘的结果，这里的相乘是指对应元素相乘，即先将两个向量中的第一个元素相乘，然后将第2个元素相乘，以此类推。

# 我的理解是：这里的 testk \* p1Vec 的意思就是将每个词与其对应的概率相关联起来

p1 = sum(testk \* p1Vec) + np.log(pClass1) # P(w|c1) \* P(c1)

p0 = sum(testk \* p0Vec) + np.log(1.0 - pClass1) # P(w|c0) \* P(c0)

if p1 > p0:

return 1

else:

return 0

# Create a dictionary of words with its frequency

train\_dir = 'D:/PG\_SUSTech1/Fall\_Semester/Machine\_learning/Python/Lab6/ling-spam/train-mails'

dictionary = make\_Dictionary(train\_dir)

train\_labels = np.zeros(702)

train\_labels[351:701] = 1

train\_matrix = extract\_features(train\_dir)

# Test the unseen mails for Spam

test\_dir = 'D:/PG\_SUSTech1/Fall\_Semester/Machine\_learning/Python/Lab6/ling-spam/test-mails'

test\_matrix = extract\_features(test\_dir)

test\_labels = np.zeros(260)

test\_labels[130:260] = 1

# 训练数据

p0V, p1V, pAb = trainNB0(train\_matrix, train\_labels)

# 测试数据

result = []

for testk in test\_matrix:

# num\_1 += classifyNB(testk, p0V, p1V, pAb)

result = np.append(result,classifyNB(testk, p0V, p1V, pAb))

#print(result)

final\_matrix = confusion\_matrix(test\_labels,result)

TP = final\_matrix[0][0]

FP = final\_matrix[0][1]

FN = final\_matrix[1][0]

TN = final\_matrix[1][1]

p = TP / (TP + FP)

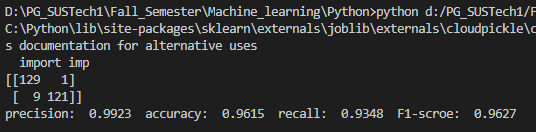
r = TP / (TP + FN)

F1 = 2 \* r \* p / (r + p)

acc = (TP + TN) / (TP + TN + FP + FN)

print(final\_matrix)

print("precision: ", "%.4f" % p, " accuracy: ", "%.4f" % acc, " recall: ", "%.4f" % r, " F1-scroe: ", "%.4f" % F1)



准确率accuracy为0.9615，260封email中有250封email预测正确

recall为0.9348

F1-score为0.9627

TP=129，FP=1，FN=9，TN=121.