一、 实验题目

感知机算法实现

二、 实验内容

1. 算法原理

PLA 是典型的分类算法，每一次只能把数据分成两类（分多类可以采用多个 PLA）。对于输入 x，我们通过计算更新得到 w，使得 W*X 可以得到两类结果，一类大于 0，一类小于等于 0.

在几何上面看 PLA 会十分直观。

在一个 n 维空间里面，有 m 个点，现在我们要做的，就是找一条线，把这 m 个点分类。

所以我们可以知道，这条线是 n 维空间里面的。

这条线的法线，实际上就是权重 W。所以转成数学问题就是找到合适的 W，使得这 m 个点会被分成两类。

更新权重的方法还是从几何上面来看。

当 W*X 得到的是错误分类的时候，我们就要更新 w 了，也就是，我们要把这条分类的线转动一下。

假如点是应该在线的左边的，但是现在错误分到了右边，那么这条线要做的就是像右转一下。为了实现向右转一下，w 就应该加上一个指向右边的向量。假如 x 被错分，那么这个 x 的分类（1 或-1）乘上这个 x 就刚好是指向右边。于是就用 w+y*x 取代之前的权重 w。

用数学的形式来看；

输入空间是 X 属于R^n，输出空间就是{-1，+1}。

从输入空间映射到输出空间公式为：

f(x) = sign(w.x + b)

其中：

$$sign(x)\begin{cases}+1，\ x>0\\ -1, x\le 0\end{cases}$$

在输入样本 x 的最前面加上一个 1，那么就可以把偏置 b 也放进权重里面。

f(x) = sign(w.x)

找到一条可以成功分类的线，也就是所有数据到这条线上的距离的绝对值总和最小。

定义点到 n 维中的这条线距离：$\frac{1}{||w||}|w.x_i|$.

当一个点被误分，也就是+1 分成-1，或者-1 分成+1，都有一个结果，就是分类和正确类相乘小于 0，那么就有$-y_i(w.x_i)>0$.于是，为了去掉绝对值，我们课引进 yi.

新的距离为：$-\frac{1}{||w||}y_i(w.x_i)$.

总距离为：$-\frac{1}{||w||}\sum_{i=0}^{m}y_i(w.x_i)$.最小化这个数，因为前面的是常量，所以去掉。

最小化$L=-\sum_{i=0}^{m}y_i(w.x_i)$,用梯度下降法，因为要改变 的量是 w，所以对 w 取偏导。

有：$-\sum_{i=0}^{m} y_i x_i$。

所谓梯度，是一个向量，指向的是标量场增长最快的方向，长度是最大变化率。

那么，梯度下降更新 w，就有$w = w + y_i + x_i$,(学习率直接取了 1，对于这个可分数据集，学习率取什么都一样)。

2. 伪代码

1) Init-PLA

```
create data
initialize weights
start loop
    for data in dataset:
        if data i  can be classified by weights
            continue
        else
            update weights
            break from dataset
        if all data can be classified to the right class:
            end loop
        else
            continue to search data from beginnning
end loop
classify test dataset with weights above
calculate accuracy, recall, precision and F1
```

2) Pocket-PLA

```
create data
initialize weights
start loop
    for data in dataset:
        if data i  can be classified by weights
            continue
        else
            update weights
            break
    end update with data
    if all data can be classified to the right class:
        put weights into pocket
        end loop
    else
        if accuracy in train dataset with new weights is bigger than
           accuracy in train dataset with weights in pocket:
           put weights into pocket
end loop
classify test dataset with weights above
calculate accuracy, recall, precision and F1
```

3) 优化（如果有）

不算是优化，只能算是一种尝试。这次更新了权重后，再选的数据不是从头开始选，而是从这个误分数据的下一个数据开始。

```
create data
initialize weights
start loop
    for data in dataset:
        if data i  can be classified by weights
            continue
        else
            update weights
            continue to search data from next data
    end update with data
    if all data can be classified to the right class:
        put weights into pocket
        end loop
    else
        if accuracy in train dataset with new weights is bigger than
           accuracy in train dataset with weights in pocket:
           put weights into pocket
end loop
classify test dataset with weights above
calculate accuracy, recall, precision and F1
```

改变了权重初始化方法，用随机初始化的方案，进行多次计算，选出一个最优权重（因为是可分的数据集，所以认为更优的权重就是 recall 更高的权重）

```
create data
cross validation:
    9/10 of train data for train,1/10 for validation
strat loop to find best weights
    initialize weights randomly
    start loop
        for data in dataset:
            if data i  can be classified by weights
                continue
            else
                update weights
                break
        end update with data
        if all data can be classified to the right class:
            put weights into pocket
            end loop
        else
            if accuracy in train dataset with new weights is bigger than
               accuracy in train dataset with weights in pocket:
               put weights into pocket
    end loop
    if recall with this weights is better than previous weights(in validation set)
        bestrecall = new recall
        weights = new weights
end find weights
classify test dataset with weights above
calculate accuracy, recall, precision and F1
```

数据归一化

```
# 归一化
def autoNorm(dataSet):
    # 数据中最小值
    minVals = dataSet.min(1)
    # 数据中最大值
    maxVals = dataSet.max(1)
    ranges = maxVals - minVals
    m,n = dataSet.shape
    minVals = tile(minVals, (n,1)).T
    ranges = tile(ranges, (n,1)).T
    normDataSet = zeros(shape(dataSet))
    # 归一化数据 = 数据 -最小数 / (最大数 - 最小数)
    normDataSet = dataSet - minVals
    normDataSet = normDataSet/ranges
    return normDataSet
```

减少属性：

```
create data
if the sum of a column < 100
    delete this column
start loop
    for data in dataset:
        if data i  can be classified by weights
            continue
        else
            update weights
            break
    end update with data
    if all data can be classified to the right class:
        put weights into pocket
        end loop
    else
        if accuracy in train dataset with new weights is bigger than
           accuracy in train dataset with weights in pocket:
           put weights into pocket
end loop
classify test dataset with weights above
calculate accuracy, recall, precision and F1
```

固定 8000 属性

```
create data
calculate the sum of a column
sort it
new matrix with 8001 columns = sum[0:8000] and the first column(all is 1)
start loop
    for data in dataset:
        if data i  can be classified by weights
            continue
        else
            update weights
            break
    end update with data
    if all data can be classified to the right class:
        put weights into pocket
        end loop
    else
        if accuracy in train dataset with new weights is bigger than
           accuracy in train dataset with weights in pocket:
           put weights into pocket
end loop
classify test dataset with weights above
calculate accuracy, recall, precision and F1
```

3. 关键代码截图（带注释）

1) Init-PLA

```
# 重新从头选数据
for num in range(10000):
    isCompleted = True
     # 用误分数据更新权重后，下一次选择数据是从头开始的数据
    for i in range(m):
        # 如果第i个数据可以被分好，则计算下一个数据
        if (sign(dot(dataMat[i], weights.T)) == classLabels[i]):
            continue
        # 如果数据被分错了，就更新权重
        # W(t+1) <- W(t) + y(t) * x(i)
        else:
            isCompleted = False
            # 更新时除以数据大小，使得新权重和旧权重相差没有这么大
            #weights = weights + classLabels[i] * dataMat[i] / dataMat[i].sum()
            weights = weights + classLabels[i] * dataMat[i]
            break
    # 假如所有数据都分好了，就不用继续更新权重了
    if isCompleted:
        break
```

2) Pocket-PLA

```
    # 重新从头选数据
    for num in range(10000):
        isCompleted = True
         # 用误分数据更新权重后，下一次选择数据是从头开始的数据
        for i in range(m):
            # 如果第i个数据可以被分好，则计算下一个数据
            if (sign(dot(dataMat[i], weights.T)) == classLabels[i]):
                continue
            # 如果数据被分错了，就更新权重
            # W(t+1) <- W(t) + y(t) * x(i)
            else:
                isCompleted = False
                # 更新时除以数据大小，使得新权重和旧权重相差没有这么大
                #weights = weights + classLabels[i] * dataMat[i] / dataMat[i].sum()
                weights = weights + classLabels[i] * dataMat[i]
                # 用现在的权重分类训练数据，得到正确率
                myLable1 = classifyAll(dataMat, weights)
                myLable2 = classifyAll(dataMat, W_pocket)
                # 如果用新权重正确率大于用pocket里的权重，则更新pocket中的权重
                if(Accuracy(classLabels,myLable1) > Accuracy(classLabels,myLable2)):
                    W_pocket = weights
                break
        # 假如所有数据都分好了，就不用继续更新权重了
        if isCompleted:
            break

    return W
```

3) 优化（如果有）

4. 创新点&优化（如果有）

No.1

从误分数据的下一个数据开始更新权重

```
    # 从误分数据下一个数据开始选数据
    for iteration in range(10):
        # 随机初始化
        weights = numpy.random.randn(1,n)
        W_pocket = weights
        # 用误分数据更新权重后，下一次选择数据是从误分数据往后的数据
        for num in range(100):
            isCompleted = True
            for i in range(m):
                # 如果第i个数据可以被分好，则计算下一个数据
                if (sign(dot(dataMat[i], weights.T)) == classLabels[i]):
                    continue
                # 如果数据被分错了，就更新权重
                # W(t+1) <- W(t) + y(t) * x(i)
                else:
                    isCompleted = False
                    weights = weights + dataMat[i] * classLabels[i]
                    myLable1 = classifyAll(dataMat, weights)
                    myLable2 = classifyAll(dataMat, W_pocket)
                    if(Accuracy(classLabels,myLable1) > Accuracy(classLabels,myLable2)):
                        W_pocket = weights
                        break
            if isCompleted:
                break
        # 如果这次完成分类用的更新次数更少，则选这次的权重
        if(num < bestNum):
            W = W_pocket
```

No.2

减少属性

```
# 删除列总和小于100的属性
colSum = myMat.sum(axis=0)
delNum = []
for i in range(len(colSum)):
    if(colSum[i] < 100):
        delNum.append(i)
# 形成新的矩阵
newMat = array( [[col[i] for i in range(0, 10001) if (i not in delNum or i == 0)] for col in myMat] )
```

留下 8000 属性

```
# 只保留总和最大的8000列和第一列（我们添加的常数列）
bigNum = colSum.argsort()
bigNum = bigNum[0:8000]
# 形成新的矩阵
newMat = array( [[col[i] for i in range(0, 10001) if (i in bigNum or i == 0)] for col in myMat] )
```

No.3

随机初始化权重，用 recall 找最佳权重

```
bestRecall = 0
for j in range(10):
    # 随机初始化
    weights = numpy.random.randn(1,n)
    print "随机初始化: ",j
    for i in range(2):
        # 交叉验证
        for iteration in range(10):
            trainMat,trainLabel,valiMat,valiLabel = trainData(iteration)
            W = pocket(trainMat, trainLabel,i,weights)
            myLablels = classifyAll(valiMat, W)
            acc=Accuracy(valiLabel,myLablels)
            rec=Recall(valiLabel,myLablels)
            pre=Precision(valiLabel,myLablels)
            f1=F1(valiLabel,myLablels)
            # 找recall最大的权重
            if(rec > bestRecall):
                W_final = W
                bestRecall = rec
    # 计算最终的四个指标
    myLablels = classifyAll(testDataSet, W_final)
    print ("    口袋算法: ")
    acc=Accuracy(testLabels,myLablels)
    rec=Recall(testLabels,myLablels)
    pre=Precision(testLabels,myLablels)
    f1=F1(testLabels,myLablels)
    print "Accuracy = " , acc
    print "Recall = " ,rec
    print "Precision = " , pre
    print "F1 = " , f1
```

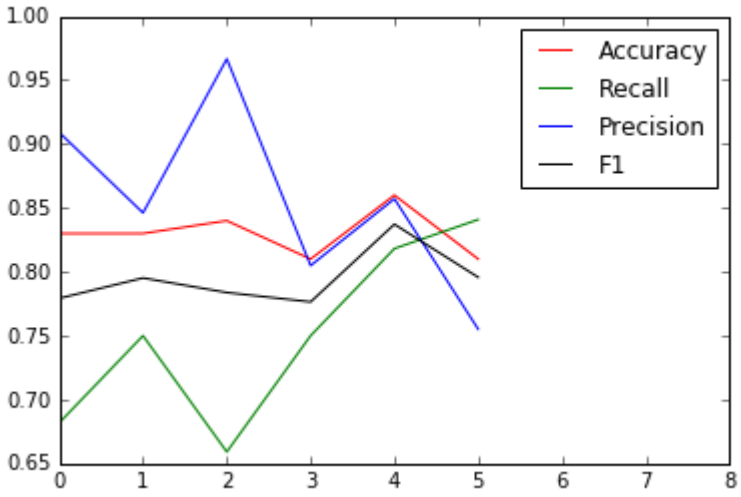
三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

(原始算法结果)

1) Init-PLA

```
初始化为0
  从头选数据
    初始算法:
Accuracy =  0.83
Recall =  0.681818181818
Precision =  0.909090909091
F1 =  0.779220779221
  从下一个数据开始
    初始算法:
Accuracy =  0.83
Recall =  0.75
Precision =  0.846153846154
F1 =  0.795180722892
初始化为1
  从头选数据
    初始算法:
Accuracy =  0.84
Recall =  0.659090909091
Precision =  0.966666666667
F1 =  0.783783783784
  从下一个数据开始
    初始算法:
Accuracy =  0.81
Recall =  0.75
Precision =  0.80487804878
F1 =  0.776470588235
随机初始化
  从头选数据
    初始算法:
Accuracy =  0.86
Recall =  0.818181818182
Precision =  0.857142857143
F1 =  0.837209302326
  从下一个数据开始
    初始算法:
Accuracy =  0.81
Recall =  0.840909090909
Precision =  0.755102040816
F1 =  0.795698924731
```



折线图中 0-2 为从头选数据（初始化分别为 0，1，随机），3-5 为选下一个数据（初始化分别为 0，1，随机）

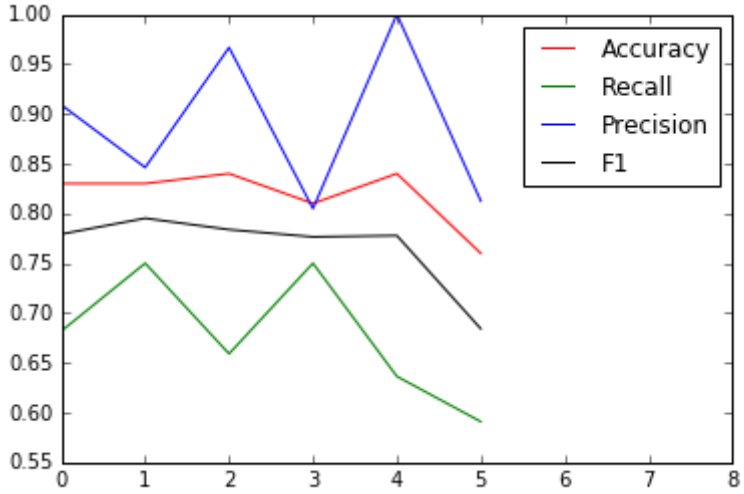
可以看到，precision 几乎一直是高于其他指标的。

最高 Accuracy 为 0.86，Recall 是 0.841，Precision 为 0.967，F1 为 0.837

采用初始化为 0，从头选择数据的方案，Accuracy 为 0.83，Recall 是 0.6818，Precision 为 0.9091，F1 为 0.7792。

2) Pocket-PLA

```
初始化为0
  从头选数据
    口袋算法:
Accuracy =  0.83
Recall =  0.681818181818
Precision =  0.909090909091
F1 =  0.779220779221
  从下一个数据开始
    口袋算法:
Accuracy =  0.83
Recall =  0.75
Precision =  0.846153846154
F1 =  0.795180722892
初始化为1
  从头选数据
    口袋算法:
Accuracy =  0.84
Recall =  0.659090909091
Precision =  0.966666666667
F1 =  0.783783783784
  从下一个数据开始
    口袋算法:
Accuracy =  0.81
Recall =  0.75
Precision =  0.80487804878
F1 =  0.776470588235
随机初始化
  从头选数据
    口袋算法:
Accuracy =  0.84
Recall =  0.636363636364
Precision =  1.0
F1 =  0.777777777778
  从下一个数据开始
    口袋算法:
Accuracy =  0.76
Recall =  0.590909090909
Precision =  0.8125
F1 =  0.684210526316
```



口袋算法的结果大致相同，不过就是随机初始化导致一些区别。

很神奇的是，Precision 在随机初始化时候可能会有 1.0.

采用初始化为 0，从头选择数据的方案，结果就是前面用原始算法的结果 Accuracy 为 0.83，Recall 是 0.6818，Precision 为 0.9091，F1 为 0.7792。

为什么原始算法和口袋算法结果一直（随机初始化除外）？

之所以会产生这样结果的原因是这是一个可分数据集，最佳的权重，也就是原始算法的权重，也就是能够把数据全部分对的权重。

为什么这是个可分数据集？

因为这里一共只有 100 个数据，但是每个数据都有 10000 个属性。根据多年前学的线性代数，我们知道，一个 m*n 的矩阵，当 n 大于 m 的时候，是一定能找到一组大小为 n 的权重，完全把数据分开。

所以，除非数据集的样本数多于属性，否则就一定是线性可分的。口袋算法这种在线性不可分数据集里面有提升作用的方法，在这里并没有提升效果。

甚至，我们用了交叉验证的方法，其实用处并不大。首先，这就是一个线性可分数据集了，所以没办法用准确率作为验证好不好的指标，因为准确率为 100%。其次，这个数据集真的很小。

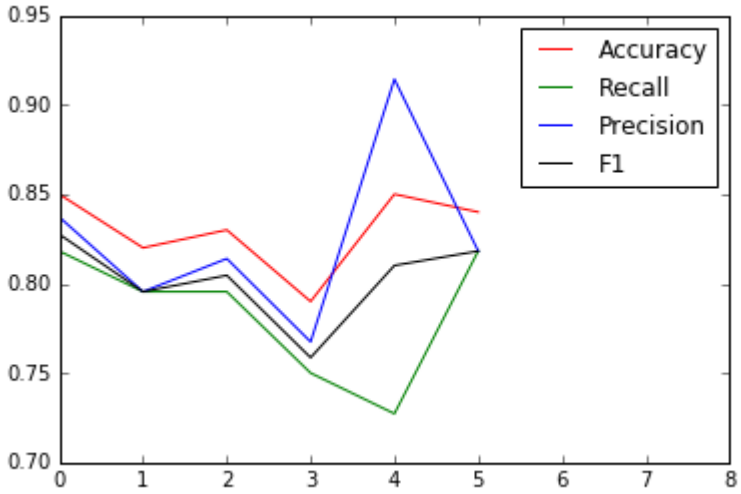
2. 评测指标展示即分析

	Accuracy	Precision	Recall	F-1
Init	0.83	0.909090909091	0.681818181818	0.779220779221
pocket	0.83	0.909090909091	0.681818181818	0.779220779221

|----------如有优化，请重复 1，2，分析优化后的算法结果----------|

1.初始算法 + 归一化数据

初始化为0
从头选数据
初始算法:
Accuracy = 0.85
Recall = 0.818181818182
Precision = 0.837209302326
F1 = 0.827586206897
从下一个数据开始
初始算法:
Accuracy = 0.82
Recall = 0.795454545455
Precision = 0.795454545455
F1 = 0.795454545455
初始化为1
从头选数据
初始算法:
Accuracy = 0.83
Recall = 0.795454545455
Precision = 0.813953488372
F1 = 0.804597701149
从下一个数据开始
初始算法:
Accuracy = 0.79
Recall = 0.75
Precision = 0.767441860465
F1 = 0.758620689655
随机初始化
从头选数据
初始算法:
Accuracy = 0.85
Recall = 0.727272727273
Precision = 0.914285714286
F1 = 0.810126582278
从下一个数据开始
初始算法:
Accuracy = 0.84
Recall = 0.818181818182
Precision = 0.818181818182
F1 = 0.818181818182



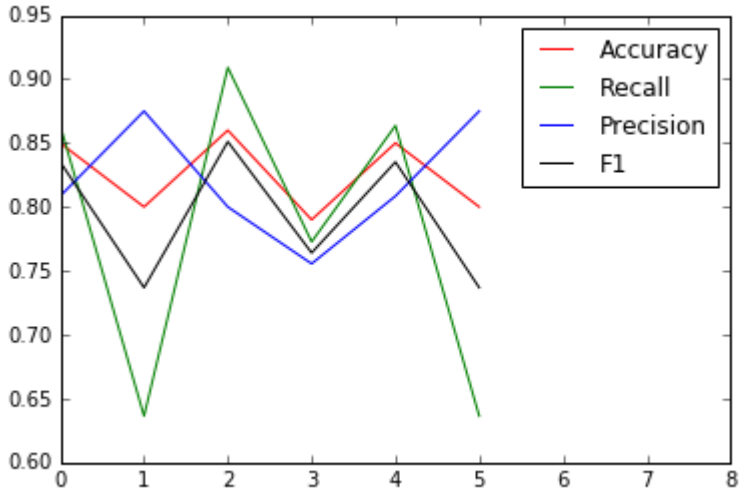
可以看到，假如归一化，效果有一点点的提升。但是提升的不多，准确率也不过85%。

2.口袋算法 + 归一化数据

不展示啦，因为是可分数据，所以口袋算法会有同样结果。

3.删除无用属性（固定只要 8000 个属性）

初始化为0
  从头选数据
    口袋算法:
Accuracy = 0.85
Recall = 0.863636363636
Precision = 0.808510638298
F1 = 0.835164835165
  从下一个数据开始
    口袋算法:
Accuracy = 0.8
Recall = 0.636363636364
Precision = 0.875
F1 = 0.736842105263
初始化为1
  从头选数据
    口袋算法:
Accuracy = 0.86
Recall = 0.909090909091
Precision = 0.8
F1 = 0.851063829787
  从下一个数据开始
    口袋算法:
Accuracy = 0.79
Recall = 0.772727272727
Precision = 0.755555555556
F1 = 0.76404494382
随机初始化
  从头选数据
    口袋算法:
Accuracy = 0.85
Recall = 0.863636363636
Precision = 0.808510638298
F1 = 0.835164835165
  从下一个数据开始
    口袋算法:
Accuracy = 0.8
Recall = 0.636363636364
Precision = 0.875
F1 = 0.736842105263



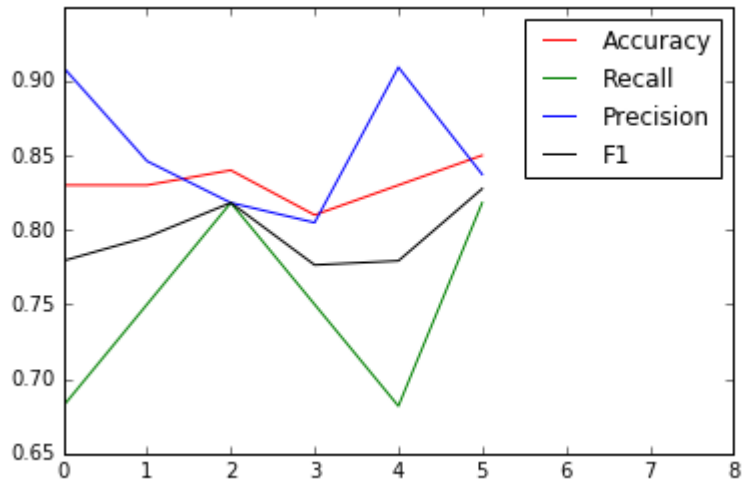
最正确的方案应该是用验证集得到最佳的大小，但是 Python 很慢，属性这么大，一个循环下来就是明天的事了。于是只能大致选了 4/5 的属性作为固定属性大小。

因为总归是可分的（8000 仍然大于 100），所以直接用口袋算法，反正原始算法也是一样的结果。

结果还过得去，正确率有 86%，Recall 有 91%，F1 也有 85%。

4.当某列总和较小时（这里认为是 100，因为总共 100 个样本），认为这是不重要属性，删去。

初始化为0
从头选数据
初始算法:
Accuracy = 0.83
Recall = 0.681818181818
Precision = 0.909090909091
F1 = 0.779220779221
从下一个数据开始
初始算法:
Accuracy = 0.83
Recall = 0.75
Precision = 0.846153846154
F1 = 0.795180722892
初始化为1
从头选数据
初始算法:
Accuracy = 0.84
Recall = 0.818181818182
Precision = 0.818181818182
F1 = 0.818181818182
从下一个数据开始
初始算法:
Accuracy = 0.81
Recall = 0.75
Precision = 0.80487804878
F1 = 0.776470588235
随机初始化
从头选数据
初始算法:
Accuracy = 0.83
Recall = 0.681818181818
Precision = 0.909090909091
F1 = 0.779220779221
从下一个数据开始
初始算法:
Accuracy = 0.85
Recall = 0.818181818182
Precision = 0.837209302326
F1 = 0.827586206897



结果不如前面固定 8000 个属性好。可能是因为我的条件放的比较宽松，所以删去的属性比较少。

5.删去部分属性，加上数据归一化，但是结果仍然相差不多。在这里就不截图了。

6.随机初始化，交叉验证，用 recall 找到最佳权重（尽量把病人都找出来）

```
Accuracy =  0.85
Recall =  0.840909090909
Precision =  0.822222222222
F1 =  0.831460674157
```

可以看到，这并不是一个很好的结果。只能算是不错。因为四种指标都是比较相近的，而且也都比较高。

如果这是一个大数据集，或者进行更多次迭代，可能结果会更好。

Python 写起来简单，但是迭代时候太慢，所以一般取得验证次数都不会太多。

但是，像这样的随机初始化，如果迭代次数足够大，最终的结果就会稳定。不过这次真的不够时间去找到它的稳定值了。现在只能知道迭代次数大于 100 时候，四种指标会在 0.8 以上。

四、 回答问题

1. 请查询相关资料，记录这四种评测指标有什么意义，尤其是在信息检索中

评估(Evaluation)在机器学习中是一个必要的工作，因为不根据某种标准方法进行评估，很难知道算法的好坏,而其评价指标往往有如下几点:准确率(Accuracy),精确率(Precision),召回率(Recall)和 F1-Measure。

准确率(accuracy)：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比。

这是一个最基本的评估。只有+1 的我们预测为+1，或者-1 预测为-1，才算是正确。在信息检索中，意思就是我们要的检索到了，或者我们不要的没有检索到占全部样本的比例。

如果我们希望，我们的检索得到的东西越多越好，这就是“召回率”。“召回率“计算的是所有"正确被检索的 item(TP)"占所有"应该检索到的 item(TP+FN)"的比例。“召回率”也叫作“查全率”，在这里“查全率”会比较好理解，就是，要找的东西，找的越全越好。哪怕有些不是我们要找的，也被放进来了。反正，相关的内容，一定要找到越多越好。

如果我们希望，检索到的东西要尽量都是我们想要的，那就要用“精确率”了。“精确率”计算的是所有"正确被检索的 item(TP)"占所有"实际被检索到的(TP+FP)"的比例.。也就是检索到的东西一定要相关，哪怕检索到的比较少。

“召回率”和“精确率”不可兼得，因为，当我们希望尽可能多的相关内容被检索到（召回率高）的时候，那么肯定一些不相关的东西也会混进来（精确率低）。反之亦然。

召回率和精确率指标有的时候是矛盾的，那么有没有办法综合考虑他们呢？方法是有的，最常见的方法应该就是 F Measure 了。

F Measure 是 Precision 和 Recall 加权调和平均：但是，F1-measure 认为精确率和召回率的权重是一样的,但有些场景下,我们可能认为召回率或者精确

率会更加重要,那么,我们就可以通过调整参数使用 Fa-measure 可以帮助我们更好的结果.

2. 思考迭代数对于原始 PLA 算法有什么影响

对于原始 PLA 算法而言，数据必须是可分的，否则就不会停下了。因为每一次，都是找一个误分数据，进行权重更新。但是一个不可分数据集永远都会有误分数据。

为了让这个算法停下了，我们就需要引入迭代次数。当迭代到一定次数的时候，就认为这是一个不可分数据集，就不要继续去更新权重了。直接停下。

这个迭代的次数应该不能太小。不然，本来这是一个可分数据集，不过就是因为需要迭代次数太多，程序早早停下了，就没有找到最合适的权重。

但是迭代次数又不宜过大，否则，本来就是一个不可分数据集，但是还是在不停找把数据完全分开的权值。

3. 有什么其他的手段可以解决数据集非线性可分的问题？

核函数， 我们可以让空间从原本的线性空间变成一个更高维的空间，在这个高维的线性空间下，再用一个超平面进行划分。不过要小心 overfitting 问题。

PS：如果不喜欢我们提供的模板，可以自行重新设计，唯一的要求就是这个模板里面提到的内容大纲不能更改，另，本学期的实验不需要写实验感想。

命名格式为：学号_姓名拼音.pdf