Homework 2

15331191 廖颖泓

1. 模型的性能度量

(1) M_1 的 Confusion Matrix 如下所示

	PREDICTED CLASS		
ACTUAL CLASS		Class = +	Class = -
	Class = +	2	2
	Class = -	2	4

查准率 (precision) = 2 / (2 + 2) = 0.5

查全率 (recall) = 2 / (2 + 2) = 0.5

假正例率 (FPR) = 2 / (2 + 4) = 0.33

F-measure = 2 * 0.5 * 0.5 / (0.5 + 0.5) = 0.5

准确率(accuracy) = (2 + 4) / 10 = 0.6

(2) M_2 的 Confusion Matrix 如下所示

	PREDICTED CLASS		
ACTUAL CLASS		Class = +	Class = -
	Class = +	1	3
	Class = -	1	5

查准率 (precision) = 1 / (1 + 1) = 0.5

查全率 (recall) = 1 / (1 + 3) = 0.25

假正例率 (FPR) = 1 / (1 + 5) = 0.17

F-measure = 2 * 0.5 * 0.25 / (0.5 + 0.25) = 0.33

准确率(accuracy) = (1 + 5) / 10 = 0.6

M_1 和 M_2 的准确率、查准率相等，M_1 的查全率和 F-measure 大于 M_2，假正例率小于 M_2，所以 M_1 的在这个测试集上表现更好。

(3) M_1 的 Confusion Matrix 如下所示

	PREDICTED CLASS		
ACTUAL CLASS		Class = +	Class = -
	Class = +	4	0
	Class = -	4	2

查准率 (precision) = 4 / (4 + 4) = 0.5

查全率 (recall) = 4 / (4 + 0) = 1

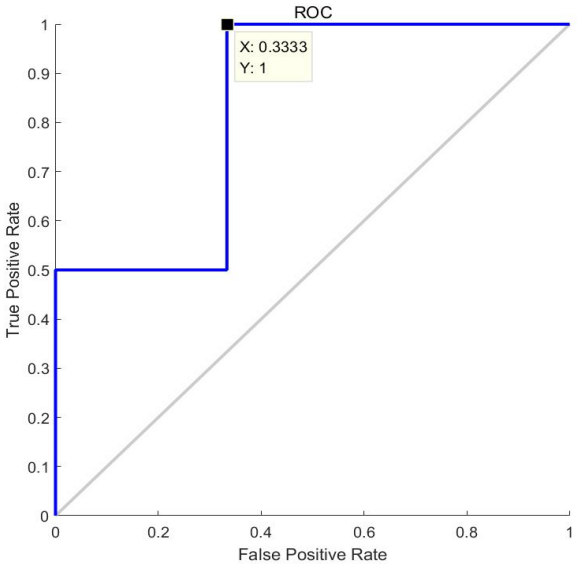
假正例率 (FPR) = 4 / (4 + 2) = 0.67

F-measure = 2 * 0.5 * 1 / (0.5 + 1) = 0.67

准确率(accuracy) = (4 + 2) / 10 = 0.6

阈值为 0.5 和阈值为 0.2 的准确率、查准率相等，阈值为 0.2 的查全率和 F-measure 大于阈值为 0.5，假正例率大于阈值为 0.5。ROC 曲线上的点(FPR, TPR)越接近左上角点(0, 1)，这个点对应的阈值产生的分类效果越好。结合(4)中的 ROC 曲线，阈值为 0.2 对应的点(1,0.76)比阈值为 0.5 对应的点(0.33,0.5)离左上角点(0,1)更远一点，所以相对来说阈值为 0.5 的 M_1 模型分类效果更好。

(4) 调用Matlab中的函数plotroc(targets,outputs)绘制ROC曲线，得到下图

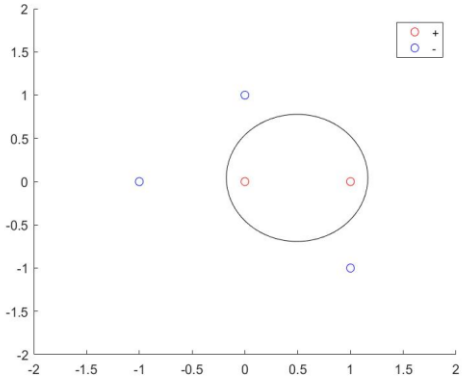


ROC曲线上的点(FPR, TPR)越接近左上角点(0, 1)，这个点对应的阈值产生的分类效果越好。结合该图，该曲线上的点(0.33, 1)最接近点(0, 1)，此时调用Matlab中的函数roc(targets,outputs)求出对应的阈值为0.44。

0	0	1
0	0	0.7300
0	0.2500	0.6900
0	0.5000	0.6700
0.1667	0.5000	0.5500
0.3333	0.5000	0.4700
0.3333	0.7500	0.4500
0.3333	1	0.4400
0.5000	1	0.3500
0.6667	1	0.1500
0.8333	1	0.0800
1	1	0

2. 神经网络

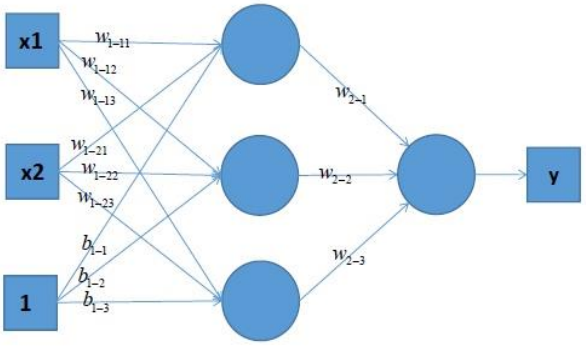
(1) 训练样本点分布如下图所示



根据样本点的分布，无法得到一个能够将训练集正实例点和负实例点完全正确分开的分离超

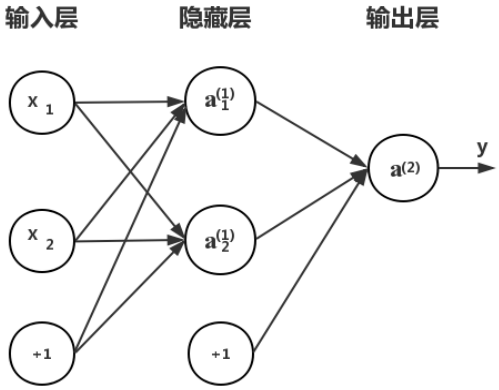
平面，所以此训练样本集不是线性可分。

(2) 如下图所示，如果将 Sigmoid 函数换成线性函数 y = wx+b，神经网络每一层的结果都是一个线性变换的结果，无论经过多少层的变换，始终得到一个线性变换的结果，说明神经网络退化成一个线性分类器。结合(1)，线性分类器无法处理数据线性不可分的情况，这就导致神经网络无法处理线性不可分的数据。



$$y = w_{2\text{-}1}(w_{1\text{-}11}x_1 + w_{1\text{-}21}x_2 + b_{1\text{-}1}) + w_{2\text{-}2}(w_{1\text{-}12}x_1 + w_{1\text{-}22}x_2 + b_{1\text{-}2}) + w_{2\text{-}3}(w_{1\text{-}13}x_1 + w_{1\text{-}23}x_2 + b_{1\text{-}3})$$

(3) 结合下图，使用前馈算法计算



$x = \begin{bmatrix} 0 & 1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 \end{bmatrix}$,

$y = [1 \quad 1 \quad 0 \quad 0 \quad 0]$,

$W_1 = \begin{bmatrix} W_{1-11} & W_{1-12} \\ W_{1-21} & W_{1-22} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$,

$W_2 = [W_{2-1} \quad W_{2-2}] = [0 \quad 0]$,

$b_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$，(考虑到矩阵加法，故增加列数来满足计算的需要)

$b_2 = [0 \quad 0 \quad 0 \quad 0 \quad 0]$

$z^{(1)} = W_1^T x + b_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

$a^{(1)} = g(z^{(1)}) = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$

$z^{(2)} = W_2^T a^{(1)} + b_2 = [0 \quad 0 \quad 0 \quad 0 \quad 0]$

$\tilde{y} = a^{(2)} = g(z^{(2)}) = [0.5 \quad 0.5 \quad 0.5 \quad 0.5 \quad 0.5] = [0 \quad 0 \quad 0 \quad 0 \quad 0]$

使用反向传播算法计算

$$\frac{\partial L}{\partial W_2}$$

$$= -\frac{\partial}{\partial W_2}\left((1-y)\log(1-\tilde{y}) + y log\tilde{y}\right)$$

$$= -(1-y)\frac{\partial}{\partial W_2}\log\left(1-g(W_2^T a^{(1)} + b_2)\right) - y\frac{\partial}{\partial W_2}\log\left(g(W_2^T a^{(1)} + b_2)\right)$$

$$= (1-y)\frac{1}{1-g(W_2^T a^{(1)}+ b_2)}g'\left(W_2^T a^{(1)} + b_2\right)a^{(1)T} - y\frac{1}{g(W_2^T a^{(1)}+ b_2)}g'(W_2^T a^{(1)} + b_2)a^{(1)T}$$

$$= \left(a^{(2)} - y\right)a^{(1)T}$$

$$\frac{\partial L}{\partial W_1}$$

$$= \frac{\partial L}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial z^{(1)}}\frac{\partial z^{(1)}}{\partial W_1}$$

$$= W_2^T g'(z^{(1)})(a^{(2)} - y)x^T$$

$$W_{2new}$$

$$= W_2 - \alpha\frac{\partial L}{\partial W_2}$$

$$= W_2 - \alpha\left(a^{(2)} - y\right)a^{(1)T}$$

$$= [0 \quad 0]$$

$$W_{1new}$$

$$= W_1 - \alpha\frac{\partial L}{\partial W_1}$$

$$= W_1 - \alpha W_2^T g'\left(z^{(1)}\right)(a^{(2)} - y)x$$

$$= \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

$$b_{2new}$$

$$= b_1 - \alpha(a^{(2)} - y)$$

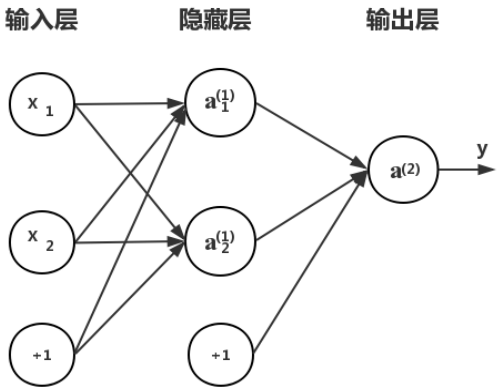
$$= [0 \quad 0 \quad 0 \quad 0 \quad 0]$$

$$b_{2new}$$

$$= [0 \quad 0 \quad 0 \quad 0 \quad 0]$$

因为链式法则对参数求偏导得到梯度再进行梯度下降，初始化所有参数为 0 会导致参数值不发生变化，每一层参数都是一样的。

(4) 神经网络架构图如图所示



为了让该网络可以正确地分类数据点，用训练样本集训练该网络，这个过程用 Python 在 Tensorflow 深度学习框架上实现，预测输出值大于 0.5 时为+，小于等于 0.5 时为-，经过 40000 次迭代，最终输出相应的参数 W_1，W_2，b_1 和 b_2。由于初始化参数为 0 不容易收敛，故初始化参数时使用随机数的方法初始化，如下图所示。

```
W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```

得到的结果为

```
Hypothesis:  [[  9.99655724e-01]
 [  9.99839783e-01]
 [  3.40792729e-04]
 [  3.24051398e-05]
 [  2.51797028e-04]]
Correct (Y):  [[ 1.]
 [ 1.]
 [ 0.]
 [ 0.]
 [ 0.]]
Accuracy:  1.0

W1:  [[ 6.72061634 -2.00270867]
 [-8.53717041 -9.57501602]]
B1:  [ 3.91956186 -3.50800133]

W2:  [[ 16.95936012]
 [-17.38990593]]
B2:  [-8.14971828]
```

可见准确率为 1，可以正确分类训练样本集。得到的参数具体如下(答案可能因为矩阵转置有所差异)：

$$W_1 = \begin{bmatrix} W_{1-11} & W_{1-12} \\ W_{1-21} & W_{1-22} \end{bmatrix} = \begin{bmatrix} 6.71 & -2 \\ -8.53 & -9.58 \end{bmatrix},$$

$$W_2 = [W_{2-1} \quad W_{2-2}] = [16.96 \quad -17.39],$$

$$b_1 = \begin{bmatrix} 3.92 & 3.92 & 3.92 & 3.92 & 3.92 \\ -3.51 & -3.51 & -3.51 & -3.51 & -3.51 \end{bmatrix},$$

$$b_2 = [-8.15 \quad -8.15 \quad -8.15 \quad -8.15 \quad -8.15]$$

具体代码见附录

3. 决策树

(1)

P(+) = 4/10 = 0.4

P(-) = 6/10 = 0.6

Entropy(Class) = – 0.4log0.4 – 0.6log0.6 = 0.97

属性 A：

P(T) = 7/10 = 0.7

P(F) = 3/10 = 0.3

P(+|T) = 4/7

P(-|T) = 3/7

P(+|F) = 0/3 = 0

P(-|F) = 3/3 = 1

GAIN(A) = Entropy(Class) – 0.7(–(4/7)log(4/7) –(3/7)log(3/7)) – 0.3(–0log0–1log1) = 0.97 – 0.69 – 0 = 0.28

属性 B：

P(T) = 4/10 = 0.4

P(F) = 6/10 = 0.6

P(T|+) = 3/4 = 0.75

P(F|+) = 1/4 = 0.25

P(T|-) = 1/6

P(F|-) = 5/6

GAIN(B) = Entropy(Class) – 0.4(–0.75log0.75–0.25log0.25) – 0.6(–(1/6)log(1/6)–(5/6)log(5/6)) = 0.97 –0.32 – 0.39 = 0.26

因为 GAIN(A) > GAIN(B)，所以应该选择属性 A 进行划分。

(2)

P(+) = 4/10 = 0.4

P(-) = 6/10 = 0.6

Gini(Class) = 1 – 0.4 * 0.4 - 0.6 * 0.6 = 0.48

属性 A：

P(T) = 7/10 = 0.7

P(F) = 3/10 = 0.3

P(+|T) = 4/7

P(-|T) = 3/7

P(+|F) = 0/3 = 0

P(-|F) = 3/3 = 1

$GINI(T) = 1 – (4/7)^2 – (3/7)^2 = 0.49$

$GINI(F) = 1 – (0)^2 – (1)^2 = 0$

GINI(A) = 0.49 * 0.7 + 0 * 0.3 = 0.34

GAIN(A) = 0.48 – 0.34 = 0.14

属性 B：

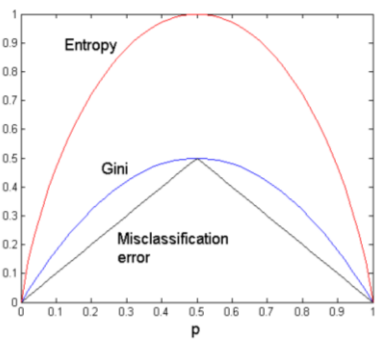
P(T) = 4/10 = 0.4

P(F) = 6/10 = 0.6

P(+|T) = 3/4 = 0.75

P(-|T) = 1/4 = 0.25

P(+|F) = 1/6
P(-|F) = 5/6
GINI(T) = 1 – $(0.75)^2$ – $(0.25)^2$ = 0.375
GINI(F) = 1 – $(1/6)^2$ – $(5/6)^2$ = 0.28
GINI(B) = 0.375 * 0.4 + 0.28 * 0.6 = 0.318
GAIN(B) = 0.48 – 0.318 = 0.162
因为 GINI(A) > GINI(B)，GAIN(A) < GAIN(B)，所以应该选择属性 B 进行划分。
(3)
P(+) = 4/10 = 0.4
P(-) = 6/10 = 0.6
Classification Error(Class) = 1 – max(0.4,0.6) = 0.4
属性 A：
P(T) = 7/10 = 0.7
P(F) = 3/10 = 0.3
P(+|T) = 4/7
P(-|T) = 3/7
P(+|F) = 0/3 = 0
P(-|F) = 3/3 = 1
Classification Error (T) = 1 – max(4/7, 3/7) = 3/7
Classification Error (F) = 1 – max(0, 1) = 0
Classification Error(A) = 3/7 * 0.7 + 0 * 0.3 = 0.3
GAIN(A) = 0.4 – 0.3 = 0.1
属性 B：
P(T) = 4/10 = 0.4
P(F) = 6/10 = 0.6
P(+|T) = 3/4 = 0.75
P(-|T) = 1/4 = 0.25
P(+|F) = 1/6
P(-|F) = 5/6
Classification Error (T) = 1 – max(0.75, 0.25) = 0.25
Classification Error (F) = 1 – max(1/6, 5/6) = 1/6
Classification Error(B) = 0.25 * 0.4 + 1/6 * 0.6 = 0.2
GAIN(B) = 0.4 – 0.2 = 0.2
因为 Classification Error(A) > Classification Error(B)，GAIN(A) < GAIN(B)，所以应该选择属性 B 进行划分。
(4) 对于二分类问题，有下图表示熵、基尼系数和分类误差率之间的关系。



熵增益根据特征属性划分数据，使得原本“混乱”的数据的熵(混乱度)减少，按照不同特征划分数据熵减少的程度会不一样。选择过程中往往会选择熵减少程度最大的特征来划分数据，一般会优先选择有较多属性值的特征，因为属性值多的特征会有相对较大的信息增益。因为信息增益反映的给定一个条件以后不确定性减少的程度，分得越细的数据集确定性更高，也就是条件熵越小，信息增益越大。

基尼增益可以选择最适合数据分割的特征，在选择最优特征的同时，还决定最优二值切分点。利用这种构建的决策树往往是一棵二叉树，构建过程中对每一个变量进行遍历，从中选择切分点。选择一个切分点满足分类均变量。 然后在选出所有变量中最小分类误差最小的变量作为切分变量。往下遍历的每个结点得到的基尼系数都会下降。

分类误差增益可以解决根据基尼增益选择特征时虽然每次划分后基尼系数下降但是分类误差没有改变的情况。

附录

simpleNN.py

```
import tensorflow as tf
import numpy as np
learning_rate = 1
x_data = [[0, 0], [1, 0], [0, 1], [-1, 0], [1, -1]]
y_data = [[1], [1], [0], [0], [0]]

# placeholders for a tensor that will be always fed.
x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))

train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
```

```
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(40001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            print(step, cost_val)

    # Accuracy report
    h, c, a, weight1, bias1, weight2, bias2 = sess.run([hypothesis, predicted, accuracy, W1, b1, W2,
b2], feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
    print("\nW1: ", weight1, "\nB1: ", bias1)
    print("\nW2: ", weight2, "\nB2: ", bias2)
```