神经网络&深度学习 TENSORFLOW Google

11 分类问题

西安科技大学 牟琦
muqi@xust.edu.cn


tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

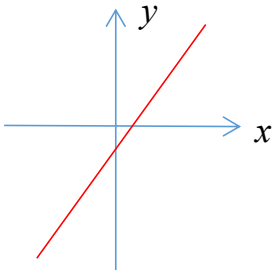
11.1 逻辑回归

线性回归

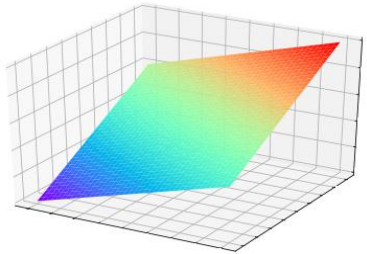
将自变量和因变量之间的关系，用**线性模型**来表示

根据已知的样本数据，对**未来**的、或者**未知**的数据进行**估计**

$$y = wx + b$$



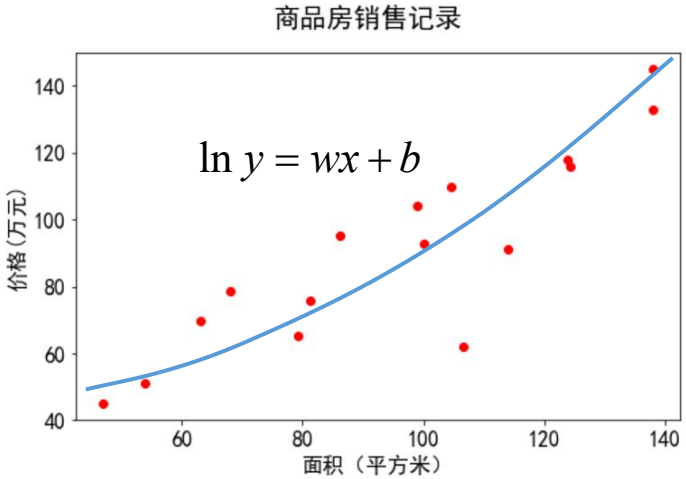
$$y = w_1x_1 + w_2x_2 + b$$



$$y = w_1x_1 + \ldots + w_mx_m + b$$

超平面 (Hyperplane)

广义线性回归



对数线性回归 (log-linear regression)

$$\ln y = wx + b \qquad y = e^{wx+b}$$

$$Y = wx + b$$

$$g(y) = wx + b \qquad y = h(wx + b)$$

$$y = g^{-1}(wx + b)$$

广义线性模型 (generalized linear model)

$g(\cdot)$: 联系函数 (link function)

任何单调可微函数

高维模型 $Y = g^{-1}(W^T X)$

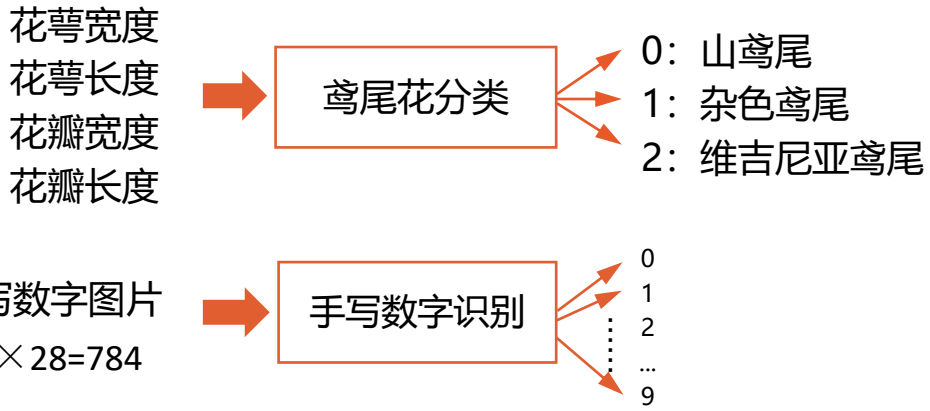
$$W = (w_0, w_1, \ldots, w_m)^T$$

$$X = (x^0, x^1, \ldots, x^m)^T$$

$$x^0 = 1$$

分类问题：垃圾邮件识别、图片分类、疾病判断

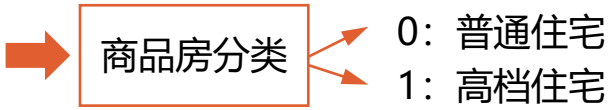
分类器：能够自动对输入的数据进行分类
输入：**特征**，输出：**离散**值



实现分类器

准备训练样本
训练分类器
对新样本分类

面　积
房间数
.....



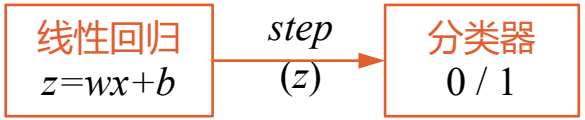
单位阶跃函数 (unit-step function)

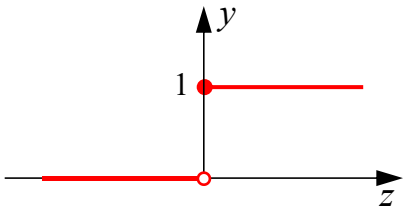
$$y=\begin{cases}0, & z<0;\\ 1, & z\geq 0;\end{cases}$$

$$z=wx+b$$

$$y=step(z)=\begin{cases}0, & z-1000000<0\\ 1, & z-1000000\geq 0\end{cases}$$

二分类问题：**1 / 0——正例和反例**

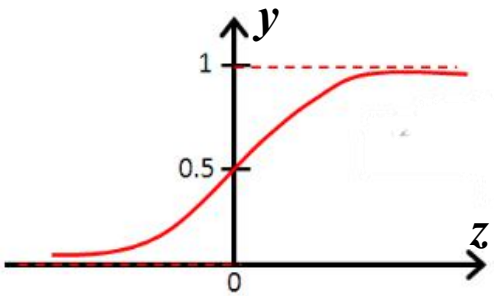




- 不光滑
- 不连续

对数几率函数 (logistic function)

$$y=\frac{1}{1+e^{-z}} \quad \Rightarrow \quad \ln\frac{y}{1-y}=z$$



- 单调上升，连续，光滑
- 任意阶可导

对数几率回归/逻辑回归 (logistic regression)

$$y=\frac{1}{1+e^{-(wx+b)}}$$

Sigmoid函数

$$y=g^{-1}(z)=\sigma(z)=\sigma(wx+b)$$

$$\sigma(z)=\frac{1}{1+e^{-z}}=\frac{1}{1+e^{-(wx+b)}}$$

多元模型

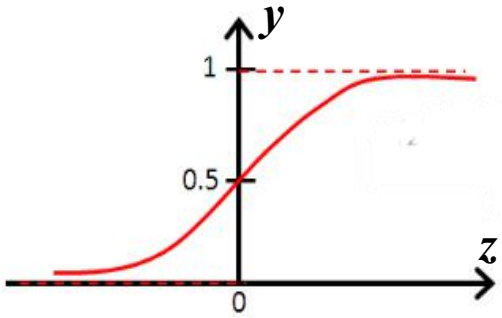
$$y=\frac{1}{1+e^{-(W^TX)}}$$

$$W=(w_0,w_1,...,w_m)^T$$
$$X=(x^0,x^1,...,x^m)^T$$
$$x^0=1$$

逻辑回归

$$y = g^{-1}(z) = \sigma(z) = \sigma(wx+b)$$

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(wx+b)}}$$



平方损失函数

$$Loss = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \sigma(wx_i+b))^2 = \sum_{i=1}^{n}(y_i - \frac{1}{1+e^{-(wx_i+b)}})^2$$

非凸函数

$$w^{(k+1)} = w^{(k)} - \eta\frac{\partial Loss}{\partial w}$$

$$b^{(k+1)} = b^{(k)} - \eta\frac{\partial Loss}{\partial b}$$

$$\frac{\partial Loss}{\partial w} = \sum_{i=1}^{n}(y_i - \sigma(wx_i+b))(-\sigma'(wx_i+b))x_i \quad (-\sigma'(wx_i+b) \to 0)$$

$$\frac{\partial Loss}{\partial b} = \sum_{i=1}^{n}(y_i - \sigma(wx_i+b))(-\sigma'(wx_i+b)) \quad (-\sigma'(wx_i+b) \to 0)$$

交叉熵损失函数

$$Loss = -\sum_{i=1}^{n} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)]$$

y_i	第i个样本的标记
$\hat{y}_i$	$\hat{y}_i = \sigma(wx_i + b)$

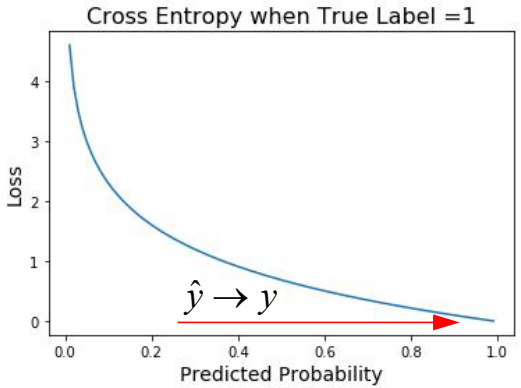
✗ $$Loss = -\sum_{i=1}^{n} [\hat{y}_i \ln y_i + (1 - \hat{y}_i) \ln(1 - y_i)]$$

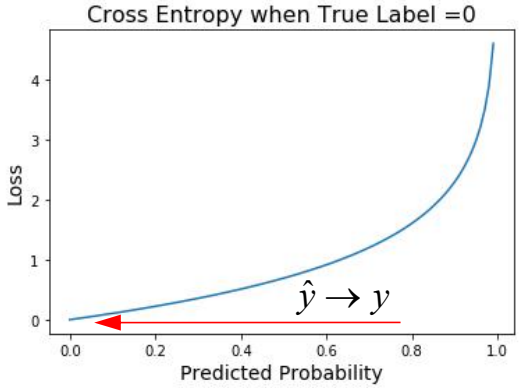
平均交叉熵损失函数

$$Loss = -\frac{1}{n}\sum_{i=1}^{n} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)]$$

交叉熵损失函数:概率分布之间的误差

$$Loss = -\sum_{i=1}^{n} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)]$$





- 无需对σ函数求导

$$\frac{\partial Loss}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} x_i (\hat{y}_i - y_i)$$

$$\frac{\partial Loss}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y)$$

- 凸函数

样本	标记	预测值	结果判断
样本1	0	0.1	正确
样本2	0	0.2	正确
样本3	1	0.8	正确
样本4	1	0.99	正确

准确率 (accuracy): $\frac{\text{正确分类的样本数}}{\text{总样本数}} = 100\%$

交叉熵损失: $Loss = -\frac{1}{n}\sum_{i=1}^{n}[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)]$

样本1：-(0×ln0.1 + 1×ln0.9)= -ln0.9=0.1053...

样本2：-(0×ln0.2 + 1×ln0.8)= -ln0.8=0.2231...

样本3：-(1×ln0.8 + 0×ln0.2)= -ln0.8=0.2231...

样本4：-(1×ln0.99 + 0×ln0.2)= -ln0.99=0.0100...

交叉熵损失：0.5616...　平均损失：0.1404...

模型A

样本	标记	预测值	结果判断
样本1	0	0.1	正确
样本2	0	0.2	正确
样本3	1	0.8	正确
样本4	1	0.49	错误

准确率：75%

交叉熵损失：

-(0×ln0.1+1×ln0.9)=-ln0.9=0.1053...

-(0×ln0.2+1×ln0.8)=-ln0.8=0.2231...

-(1×ln0.8+0×ln0.2)=-ln0.8=0.2231...

-(1×ln0.49+0×ln0.51)=-ln0.49=0.7133...

平均损失：**0.3162...**

模型B

样本	标记	预测值	结果判断
样本1	0	0.49	正确
样本2	0	0.45	正确
样本3	1	0.51	正确
样本4	1	0.1	错误

准确率：75%

交叉熵损失：

-(0×ln0.49+1×ln0.51)=-ln0.51=0.6733...

-(0×ln0.45+1×ln0.55)=-ln0.55=0.5978...

-(1×ln0.51+0×ln0.49)=-ln0.51=0.6733...

-(1×ln0.1+0×ln0.9)=-ln0.1=2.3025...

平均损失：**1.0617...**

神经网络&深度学习
TENSORFLOW
Google

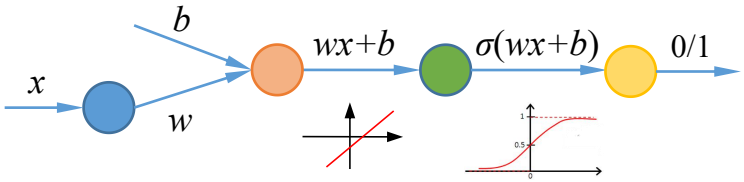
11.2 实例：实现一元逻辑回归

中国大学MOOC

逻辑回归

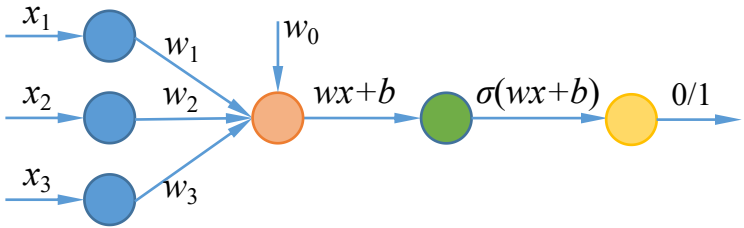
一元逻辑回归

$$y=\sigma(wx+b)=\frac{1}{1+e^{-(wx+b)}}$$



多元逻辑回归

$$y=\sigma(W^T X)=\frac{1}{1+e^{-W^T X}}$$



准确率 $=\frac{\text{正确分类的样本数}}{\text{总样本数}}$

交叉熵损失: $Loss=-\frac{1}{n}\sum_{i=1}^{n}[y_i\ln\hat{y}_i+(1-y_i)\ln(1-\hat{y}_i)]$

sigmoid()函数

$$y=\frac{1}{1+e^{-(wx+b)}}$$

```
In [1]: import tensorflow as tf
        print("TensorFlow version:",tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import numpy as np

In [3]: x=np.array([1.,2.,3.,4.])

In [4]: w = tf.Variable(1.)
        b = tf.Variable(1.)

In [5]: 1/(1+tf.exp(-(w*x+b)))

Out[5]: <tf.Tensor: id=25, shape=(4,), dtype=float32, numpy=array
        ([0.880797  , 0.95257413, 0.98201376, 0.9933072 ], dtype=
        float32)>
```

交叉熵损失函数

$$Loss = -\sum_{i=1}^{n}[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)]$$

```
In [6]: y=np.array([0,0,1,1])
        pred=np.array([0.1, 0.2, 0.8, 0.49])

In [7]: 1-y
Out[7]: array([1, 1, 0, 0])

In [8]: 1-pred
Out[8]: array([0.9 , 0.8 , 0.2 , 0.51])

In [9]: -tf.reduce_sum(y*tf.math.log(pred)+(1-y)*tf.math.log(1-pred))
Out[9]: <tf.Tensor: id=37, shape=(), dtype=float64, numpy=1.2649975061637104>

In [10]: -tf.reduce_mean(y*tf.math.log(pred)+(1-y)*tf.math.log(1-pred))
Out[10]: <tf.Tensor: id=49, shape=(), dtype=float64, numpy=0.3162493765409276>
```

准确率

$$= \frac{\text{正确分类的样本数}}{\text{总样本数}}$$

```
In [11]: y=np.array([0,0,1,1])
         pred=np.array([0.1, 0.2, 0.8, 0.49])

In [12]: tf.round(pred)
Out[12]: <tf.Tensor: id=51, shape=(4,), dtype=float64, numpy=array([0., 0., 1., 0.])>

In [13]: tf.equal(tf.round(pred), y)
Out[13]: <tf.Tensor: id=55, shape=(4,), dtype=bool, numpy=array([ True,  True,  True, False])>

In [14]: tf.cast(tf.equal(tf.round(pred), y),tf.int8)
Out[14]: <tf.Tensor: id=60, shape=(4,), dtype=int8, numpy=array([1, 1, 1, 0], dtype=int8)>

In [15]: tf.reduce_mean(tf.cast(tf.equal(tf.round(pred), y),tf.float32))
Out[15]: <tf.Tensor: id=67, shape=(), dtype=float32, numpy=0.75>

In [16]: tf.round(0.5)
Out[16]: <tf.Tensor: id=69, shape=(), dtype=float32, numpy=0.0>

In [17]: tf.round(0.500001)
Out[17]: <tf.Tensor: id=71, shape=(), dtype=float32, numpy=1.0>
```

where (condition, a, b)

```
In [18]: pred=np.array([0.1, 0.2, 0.8, 0.49])

In [19]: tf.where(pred<0.5, 0, 1)
Out[19]: <tf.Tensor: id=75, shape=(4,), dtype=int32, numpy=array([0, 0, 1, 0])>

In [20]: pred<0.5
Out[20]: array([ True,  True, False,  True])

In [21]: tf.where(pred<0.4, 0, 1)
Out[21]: <tf.Tensor: id=79, shape=(4,), dtype=int32, numpy=array([0, 0, 1, 1])>
```

```
In [22]: pred=np.array([0.1, 0.2, 0.8, 0.49])

         a=np.array([1,2,3,4])
         b=np.array([10,20,30,40])

In [23]: tf.where(pred<0.5, a, b)
```

参数a,b是数组

```
Out[23]: <tf.Tensor: id=83, shape=(4,), dtype=int32, numpy=array([ 1,  2, 30,  4])>

In [24]: tf.where(pred>=0.5)
```

参数a,b缺省

```
Out[24]: <tf.Tensor: id=85, shape=(1, 1), dtype=int64, numpy=array([[2]], dtype=int64)>

In [25]: y=np.array([0,0,1,1])
         pred=np.array([0.1, 0.2, 0.8, 0.49])

In [26]: tf.reduce_mean(tf.cast(tf.equal(tf.where(pred<0.5, 0, 1), y),tf.float32))
```

计算准确率

```
Out[26]: <tf.Tensor: id=94, shape=(), dtype=float32, numpy=0.75>
```

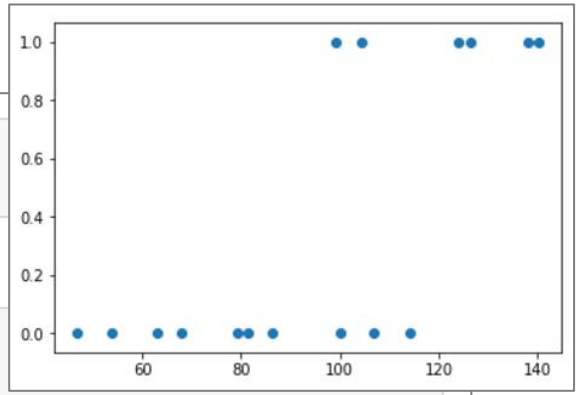
房屋销售记录

序号	面积（平方米）	类型	序号	面积（平方米）	类型
1	137.97	1	9	106.69	0
2	104.50	1	10	140.05	1
3	100.00	0	11	53.75	0
4	126.32	1	12	46.91	0
5	79.20	0	13	68.00	0
6	99.00	1	14	63.02	0
7	124.00	1	15	81.26	0
8	114.00	0	16	86.21	0

0：普通住宅
1：高档住宅

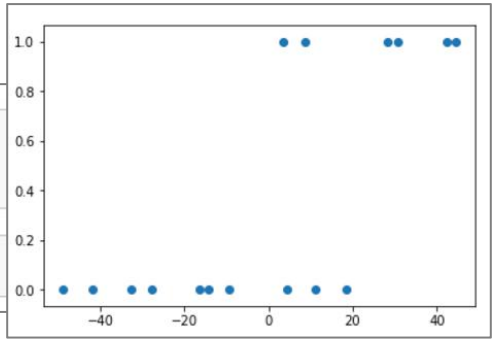
加载数据

```
In [1]: import tensorflow as tf
        print("TensorFlow version:",tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import numpy as np
        import matplotlib.pyplot as plt

In [3]: x = np.array([137.97, 104.50, 100.00, 126.32, 79.20, 99.00, 124.00, 114.00,
                      106.69, 140.05, 53.75,  46.91,  68.00, 63.02, 81.26,  86.21])
        y = np.array([1, 1, 0, 1, 0, 1, 1, 0, 0,  1, 0, 0,  0,  0, 0,  0])

In [4]: plt.scatter(x,y)
```



数据处理

```
In [5]: x_train=x-np.mean(x)
        y_train=y

In [6]: plt.scatter(x_train,y_train)
```



设置超参数

```
In [7]: learn_rate = 0.005
        iter=5

        display_step=1
```

设置模型变量初始值

```
In [8]: np.random.seed(612)
        w = tf.Variable(np.random.randn())
        b = tf.Variable(np.random.randn())
```

训练模型

```
i: 0,Train Loss: 0.852807, Accuracy: 0.625000
i: 1,Train Loss: 0.400259, Accuracy: 0.875000
i: 2,Train Loss: 0.341504, Accuracy: 0.812500
i: 3,Train Loss: 0.322571, Accuracy: 0.812500
i: 4,Train Loss: 0.313972, Accuracy: 0.812500
i: 5,Train Loss: 0.309411, Accuracy: 0.812500
```

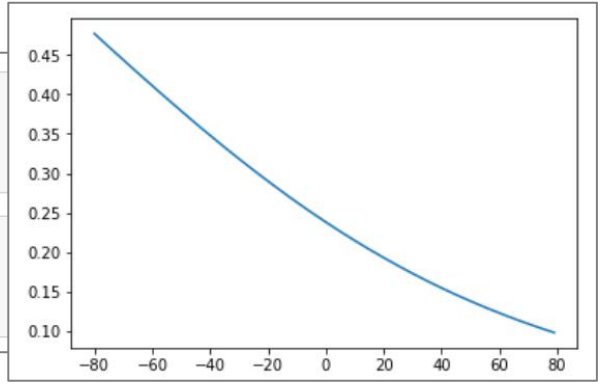
10/16
14/16
13/16

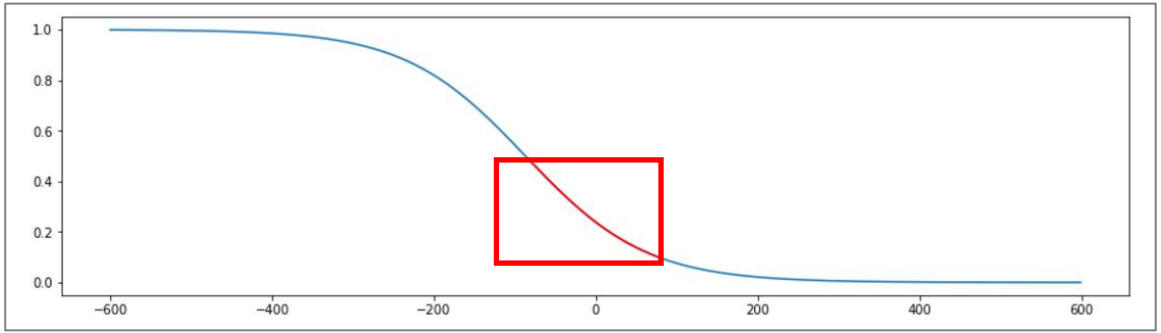
In [9]:

```
cross_train=[]
acc_train=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:
        pred_train =1/(1+tf.exp(-(w*x_train+b)))
        Loss_train = -tf.reduce_mean(y_train*tf.math.log(pred_train)+(1-y_train)*tf.math.log(1-pred_train))
        Accuracy_train = tf.reduce_mean(tf.cast(tf.equal(tf.where(pred_train<0.5, 0, 1), y_train),tf.float32))

    cross_train.append(Loss_train)
    acc_train.append(Accuracy_train)

    dL_dw,dL_db = tape.gradient(Loss_train,[w,b])

    w.assign_sub(learn_rate*dL_dw)
    b.assign_sub(learn_rate*dL_db)

    if i % display_step == 0:
        print("i: %i,Train Loss: %f, Accuracy: %f" %(i, Loss_train,Accuracy_train))
```

使用初始权值时的sigmoid函数

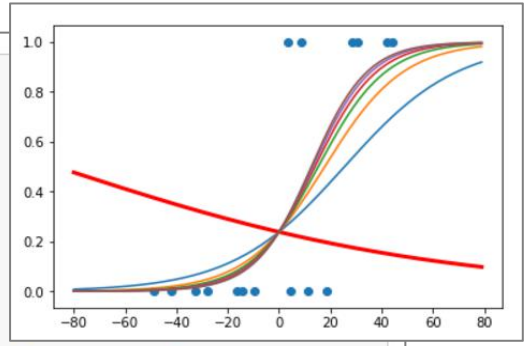
```
In [8]: np.random.seed(612)
        w = tf.Variable(np.random.randn())
        b = tf.Variable(np.random.randn())

In [9]: x_=range(-80,80)
        y_=1/(1+tf.exp(-(w*x_+b)))
        plt.plot(x_,y_)
```





11.2 实例：实现一元逻辑回归

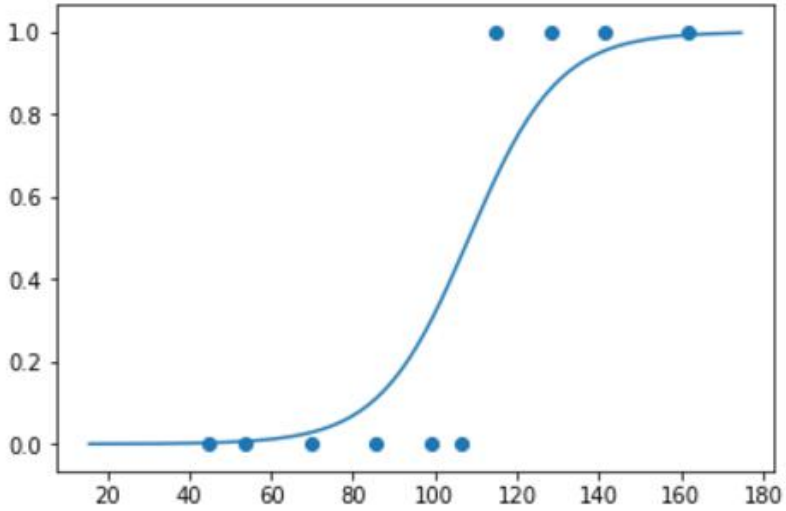
In [10]:
```
plt.scatter(x_train, y_train)
plt.plot(x_, y_, color="red", linewidth=3)

cross_train=[]
acc_train=[]

for i in range(0,iter+1):

    with tf.GradientTape() as tape:
        pred_train =1/(1+tf.exp(-(w*x_train+b)))
        Loss_train = -tf.reduce_mean(y_train*tf.math.log(pred_train)+(1-y_train)*tf.math.log(1-pred_train))
        Accuracy_train = tf.reduce_mean(tf.cast(tf.equal(tf.where(pred_train<0.5, 0, 1), y_train),tf.float32))

    cross_train.append(Loss_train)
    acc_train.append(Accuracy_train)

    dL_dw, dL_db = tape.gradient(Loss_train, [w, b])

    w.assign_sub(learn_rate*dL_dw)
    b.assign_sub(learn_rate*dL_db)

    if i % display_step == 0:
        print("i: %i,Train Loss: %f, Accuracy: %f" %(i, Loss_train,Accuracy_train))
        y_=1/(1+tf.exp(-(w*x_+b)))
        plt.plot(x_, y_)
```



```
In [11]: x_test=[128.15, 45.00, 141.43, 106.27, 99.00, 53.84, 85.36, 70.00,162.00,114.60]

In [12]: pred_test=1/(1+tf.exp(-(w*(x_test-np.mean(x))+b)))

In [13]: y_test=tf.where(pred_test<0.5,0,1)

In [14]: for i in range(len(x_test)):
             print(x_test[i],"\t",pred_test[i].numpy(),"\t",y_test[i].numpy(),"\t")
```

```
128.15    0.8610252       1
45.0      0.0029561974    0
141.43    0.9545566       1
106.27    0.45318928      0
99.0      0.2981362       0
53.84     0.00663888      0
85.36     0.108105935     0
70.0      0.028681064     0
162.0     0.9928677       1
114.6     0.6406205       1
```

In [15]:
```
plt.scatter(x_test,y_test)

x_=np.array(range(-80,80))
y_=1/(1+tf.exp(-(w*x_+b)))
plt.plot(x_+np.mean(x),y_)
plt.show()
```



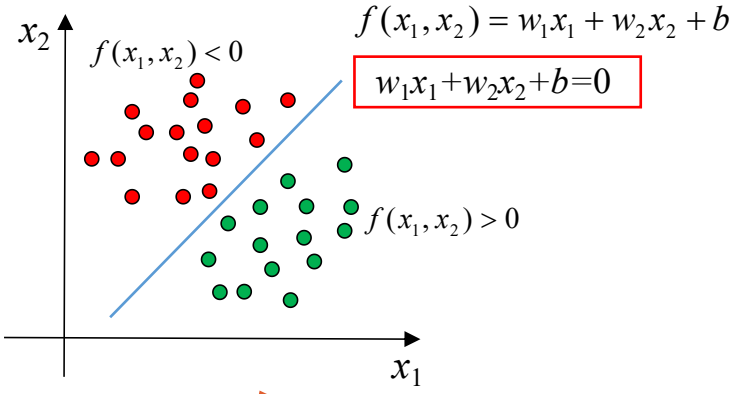
神经网络&深度学习
TENSORFLOW
Google

11.3 线性分类器

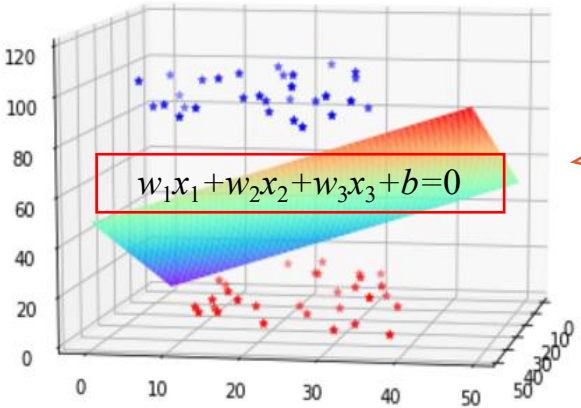
中国大学MOOC

线性分类器（Linear Classifier）

决策边界 m维空间：超平面 $W^T X = 0$

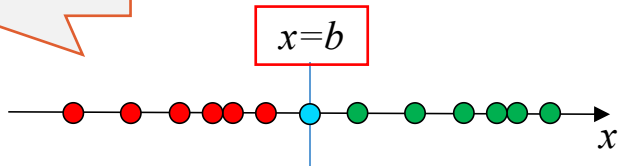


二维空间中的数据集

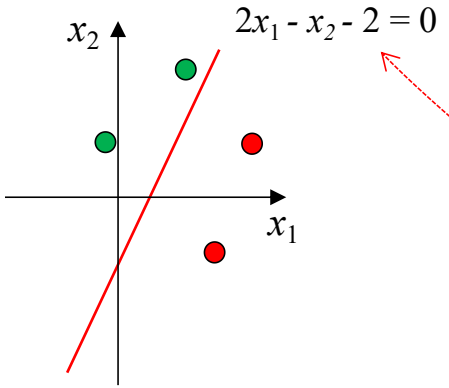


三维空间中的数据集

一维空间中的数据集



线性分类器实例



分类器

$y=step(z)=step(2x_1-x_2-2)$

$z >= 0,\ y=1$

$z < 0,\ y=0$

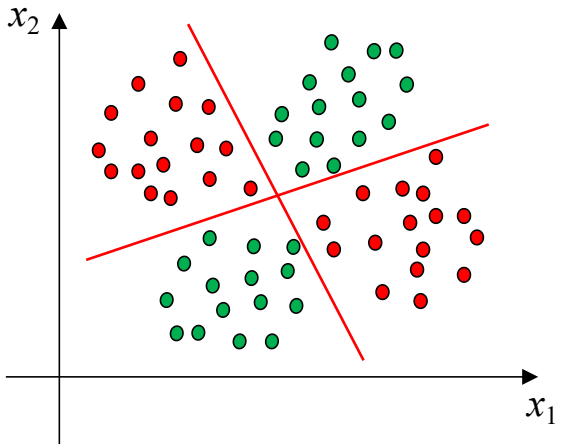
$y=\sigma(z)=\sigma(2x_1-x_2-2)$

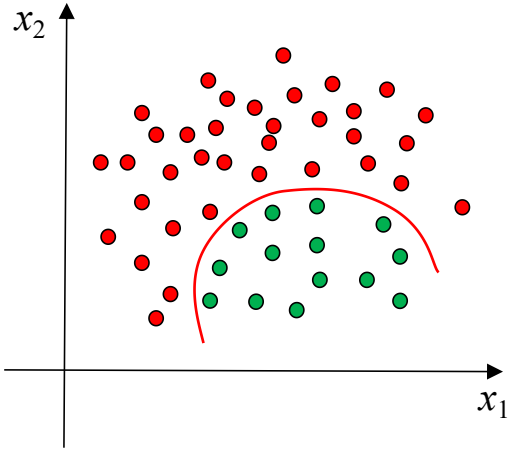
逻辑回归

$y=\sigma(z)=\sigma(w_1x_1+w_2x_2+b)$

(x_1, x_2)	$z = 2x_1 - x_2 - 2$	y
(1, 3)	-3	0
(3, 1)	3	1
(-1, 1)	-5	0
(2, -2)	4	1

线性不可分

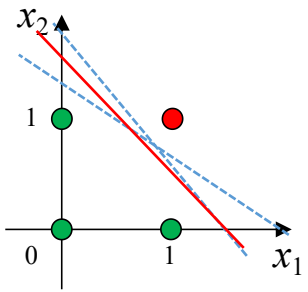




逻辑运算

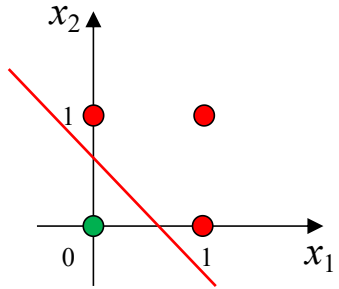
与运算

(x_1, x_2)	y
(0, 0)	0
(0, 1)	0
(1, 0)	0
(1, 1)	1



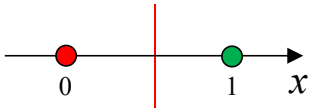
或运算

(x_1, x_2)	y
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	1



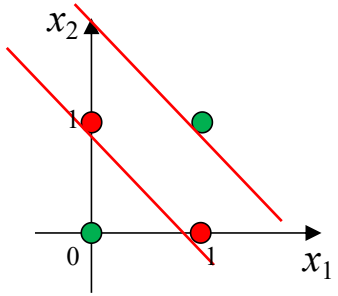
非运算

x	y
0	1
1	0



异或运算

(x_1, x_2)	y
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0



神经网络&深度学习
TENSORFLOW
Google

11.4 实例：实现多元逻辑回归

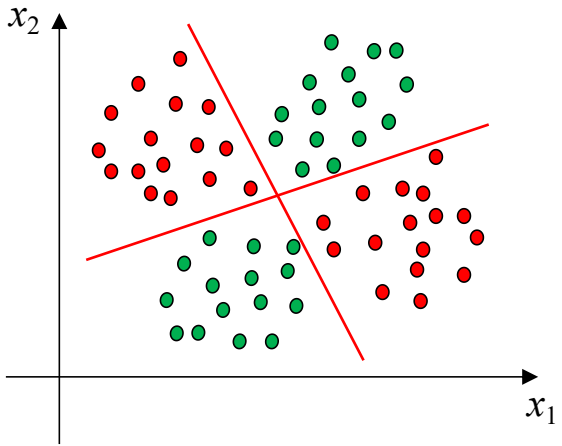
中国大学MOOC

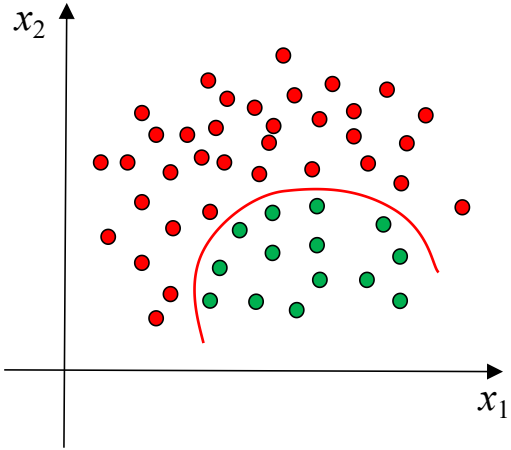
神经网络&深度学习
TENSORFLOW
Google

11.4.1 实现多元逻辑回归

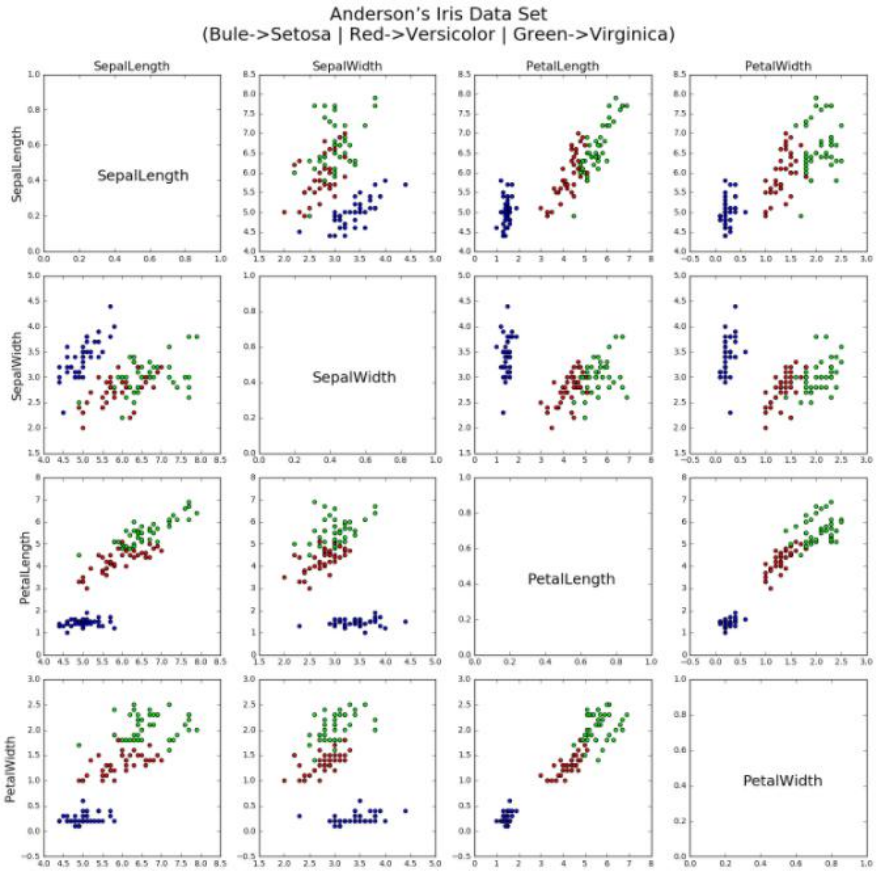
中国大学MOOC

线性不可分





- **Iris数据集**
  - 150个样本
  - 4个属性
    - 花萼长度（Sepal Length）
    - 花萼宽度（Sepal Width）
    - 花瓣长度（Petal Length）
    - 花瓣宽度（Petal Width）
  - 1个标签
    - 山鸢尾（Setosa）
    - 变色鸢尾（Versicolour）
    - 维吉尼亚鸢尾（Virginica）



加载数据

In [1]:
```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.0.0
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [3]:
```
TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1], TRAIN_URL)
```

In [4]:
```
df_iris = pd.read_csv(train_path, header=0)
```

处理数据

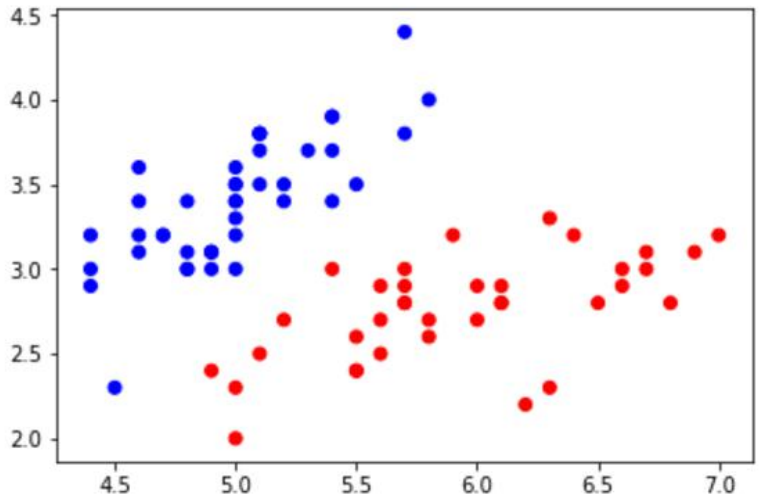
- 转化为NumPy数组
- 提取属性和标签
- 提取山鸢尾和变色鸢尾

```
In [5]: iris=np.array(df_iris)

In [6]: iris.shape
Out[6]: (120, 5)

In [7]: train_x=iris[:,0:2]
        train_y=iris[:,4]

In [8]: train_x.shape, train_y.shape
Out[8]: ((120, 2), (120,))

In [9]: x_train = train_x[train_y < 2]
        y_train = train_y[train_y < 2]

In [10]: x_train.shape, y_train.shape
Out[10]: ((78, 2), (78,))

In [11]: num=len(x_train)
```

处理数据

- 可视化样本

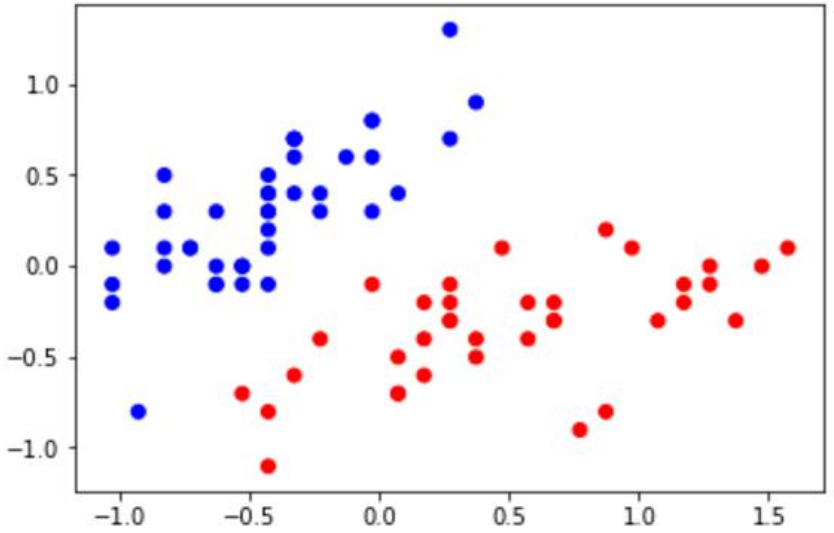
```
In [12]: cm_pt = mpl.colors.ListedColormap(["blue", "red"])
         plt.scatter(x_train[:,0], x_train[:,1], c=y_train, cmap=cm_pt)
         plt.show()
```



处理数据

- 属性中心化

```
In [13]: x_train=x_train-np.mean(x_train,axis=0)

In [14]: plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap=cm_pt)
         plt.show()
```



处理数据

- 生成多元模型的属性矩阵和标签列向量

```
In [15]: x0_train = np.ones(num).reshape(-1, 1)

In [16]: X =tf.cast(tf.concat((x0_train, x_train), axis = 1), tf.float32)
         Y= tf.cast(y_train.reshape(-1, 1), tf.float32)

In [17]: X.shape, Y.shape

Out[17]: (TensorShape([78, 3]), TensorShape([78, 1]))
```

设置超参数

```
In [18]: learn_rate=0.2
         iter=120

         display_step=30
```

设置模型参数初始值

```
In [19]: np.random.seed(612)
         W=tf.Variable(np.random.randn(3,1),dtype=tf.float32)
```

训练模型

```
i: 0, Acc:0.230769, Loss: 0.994269
i: 30, Acc:0.961538, Loss: 0.481892
i: 60, Acc:0.987179, Loss: 0.319128
i: 90, Acc:0.987179, Loss: 0.246626
i: 120, Acc:1.000000, Loss: 0.204982
```

In [20]:

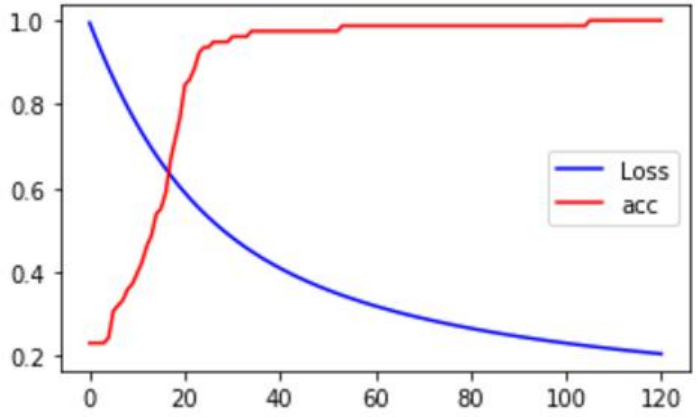
```
ce=[]
acc=[]

for i in range(0, iter+1):
    with tf.GradientTape() as tape:
        PRED =1/(1+tf.exp(-tf.matmul(X,W)))
        Loss =-tf.reduce_mean(Y*tf.math.log(PRED)+(1-Y)*tf.math.log(1-PRED))

    accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED.numpy()<0.5,0.,1.), Y),tf.float32))
    ce.append(Loss)
    acc.append(accuracy)

    dL_dW= tape.gradient(Loss,W)
    W.assign_sub(learn_rate*dL_dW)

    if i % display_step == 0:
        print("i: %i, Acc:%f, Loss: %f" % (i,accuracy,Loss))
```

可视化

- 绘制**损失**和**准确率**变化曲线

```
In [21]: plt.figure(figsize=(5,3))
         plt.plot(ce,color="blue",label="Loss")
         plt.plot(acc,color="red",label="acc")
         plt.legend()
         plt.show()
```

```
i: 0, Acc:0.230769, Loss: 0.994269
i: 30, Acc:0.961538, Loss: 0.481892
i: 60, Acc:0.987179, Loss: 0.319128
i: 90, Acc:0.987179, Loss: 0.246626
i: 120, Acc:1.000000, Loss: 0.204982
```



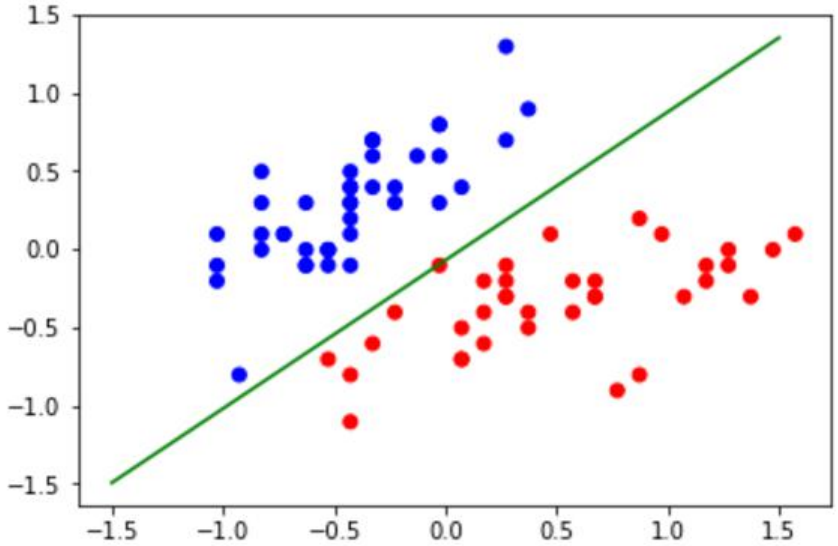
可视化

绘制决策边界

$$w_1x_1 + w_2x_2 + w_0 = 0$$

$$x2 = -\frac{w_1x_1 + w_0}{w_2}$$

```
In [22]: plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap=cm_pt)
         x_=[-1.5,1.5]
         y_=-(W[1]*x_+W[0])/W[2]
         plt.plot(x_,y_,color="g")
         plt.show()
```



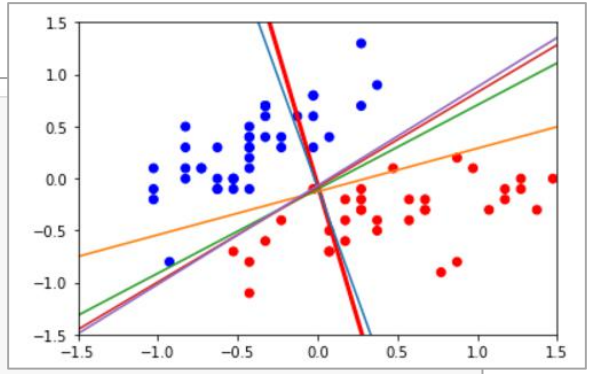
可视化

- 在训练过程中绘制**决策边界**

```
In [19]: np.random.seed(612)
         W=tf.Variable(np.random.randn(3,1),dtype=tf.float32)

In [20]: cm_pt = mpl.colors.ListedColormap(["blue", "red"])

In [21]: x_=[-1.5,1.5]
         y_=-(W[0]+W[1]*x_)/W[2]
```

训练模型

```
In [22]: plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap=cm_pt)
         plt.plot(x_,y_,color="red",linewidth=3)
         plt.xlim([-1.5,1.5])
         plt.ylim([-1.5,1.5])

         ce=[]
         acc=[]
         for i in range(0, iter+1):
             with tf.GradientTape() as tape:
                 PRED =1/(1+tf.exp(-tf.matmul(X,W)))
                 Loss =-tf.reduce_mean(Y*tf.math.log(PRED)+(1-Y)*tf.math.log(1-PRED))

             accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED.numpy()<0.5,0.,1.), Y),tf.float32))
             ce.append(Loss)
             acc.append(accuracy)

             dL_dW= tape.gradient(Loss,W)
             W.assign_sub(learn_rate*dL_dW)

             if i % display_step == 0:
                 print("i: %i, Acc:%f, Loss: %f" % (i,accuracy,Loss))
                 y_=-(W[0]+W[1]*x_)/W[2]
                 plt.plot(x_,y_)
```




```
i: 0, Acc:0.230769, Loss: 0.994269
i: 30, Acc:0.961538, Loss: 0.481892
i: 60, Acc:0.987179, Loss: 0.319128
i: 90, Acc:0.987179, Loss: 0.246626
i: 120, Acc:1.000000, Loss: 0.204982
```

使用测试集

加载数据集

In [1]:
```
import tensorflow as tf
print("TensorFlow version:",tf.__version__)
```

```
TensorFlow version: 2.0.0
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [3]:
```
TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1], TRAIN_URL)

TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"
test_path = tf.keras.utils.get_file(TEST_URL.split('/')[-1], TEST_URL)
```

数据处理

```
In [4]: df_iris_train = pd.read_csv(train_path, header=0)
        df_iris_test = pd.read_csv(test_path, header=0)

In [5]: iris_train=np.array(df_iris_train)
        iris_test=np.array(df_iris_test)

In [6]: iris_train.shape,iris_test.shape

Out[6]: ((120, 5), (30, 5))
```

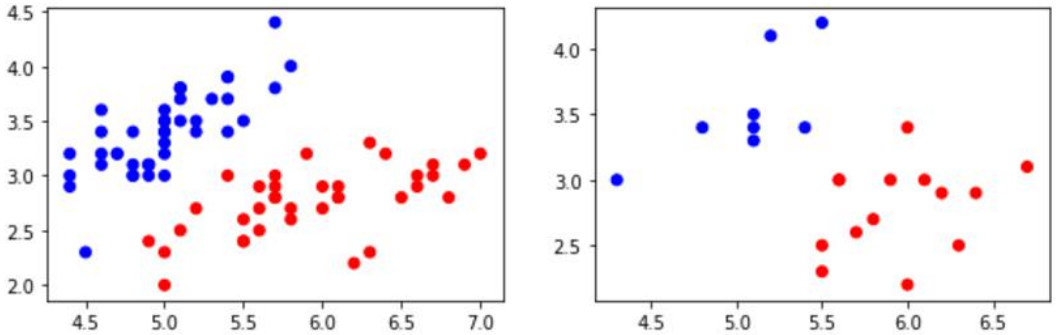
数据处理

```
In [7]: train_x=iris_train[:,0:2]
        train_y=iris_train[:,4]

        test_x=iris_test[:,0:2]
        test_y=iris_test[:,4]

In [8]: train_x.shape,train_y.shape
Out[8]: ((120, 2), (120,))

In [9]: test_x.shape,test_y.shape
Out[9]: ((30, 2), (30,))
```

数据处理

```
In [10]: x_train = train_x[train_y < 2]
         y_train = train_y[train_y < 2]

In [11]: x_train.shape, y_train.shape
Out[11]: ((78, 2), (78,))

In [12]: x_test = test_x[test_y < 2]
         y_test = test_y[test_y < 2]

In [13]: x_test.shape, y_test.shape
Out[13]: ((22, 2), (22,))

In [14]: num_train=len(x_train)
         num_test=len(x_test)

In [15]: num_train, num_test
Out[15]: (78, 22)
```

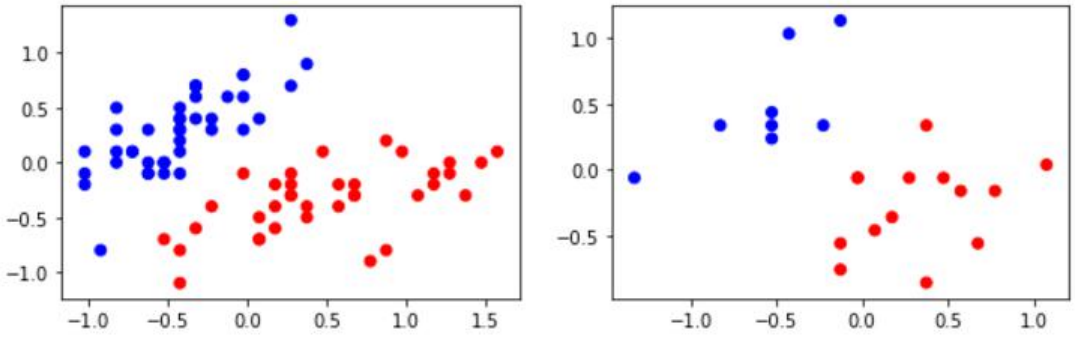
In [16]:
```
plt.figure(figsize=(10,3))
cm_pt = mpl.colors.ListedColormap(["blue", "red"])

plt.subplot(121)
plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap=cm_pt)

plt.subplot(122)
plt.scatter(x_test[:,0],x_test[:,1],c=y_test,cmap=cm_pt)

plt.show()
```



数据处理——中心化

```
In [17]: print(np.mean(x_train,axis=0))
         print(np.mean(x_test,axis=0))

         [5.42692308 3.1025641 ]
         [5.62727273 3.06363636]

In [18]: x_train=x_train-np.mean(x_train,axis=0)
         x_test=x_test-np.mean(x_test,axis=0)
```

In [19]:
```
plt.figure(figsize=(10,3))

plt.subplot(121)
plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap=cm_pt)

plt.subplot(122)
plt.scatter(x_test[:,0],x_test[:,1],c=y_test,cmap=cm_pt)

plt.show()
```



```
In [20]: x0_train = np.ones(num_train).reshape(-1, 1)
         X_train = tf.cast(tf.concat((x0_train, x_train), axis = 1), dtype=tf.float32)
         Y_train = tf.cast(y_train.reshape(-1, 1), dtype=tf.float32)

In [21]: X_train.shape, Y_train.shape

Out[21]: (TensorShape([78, 3]), TensorShape([78, 1]))

In [22]: x0_test = np.ones(num_test).reshape(-1, 1)
         X_test =tf.cast(tf.concat((x0_test, x_test), axis = 1), dtype=tf.float32)
         Y_test = tf.cast(y_test.reshape(-1, 1), dtype=tf.float32)

In [23]: X_test.shape, Y_test.shape

Out[23]: (TensorShape([22, 3]), TensorShape([22, 1]))
```

设置超参数、设置模型参数初识值

In [24]:
```
learn_rate=0.2
iter=120

display_step=30
```

In [25]:
```
np.random.seed(612)
W=tf.Variable(np.random.randn(3,1),dtype=tf.float32)
```

11.4.1 实现多元逻辑回归

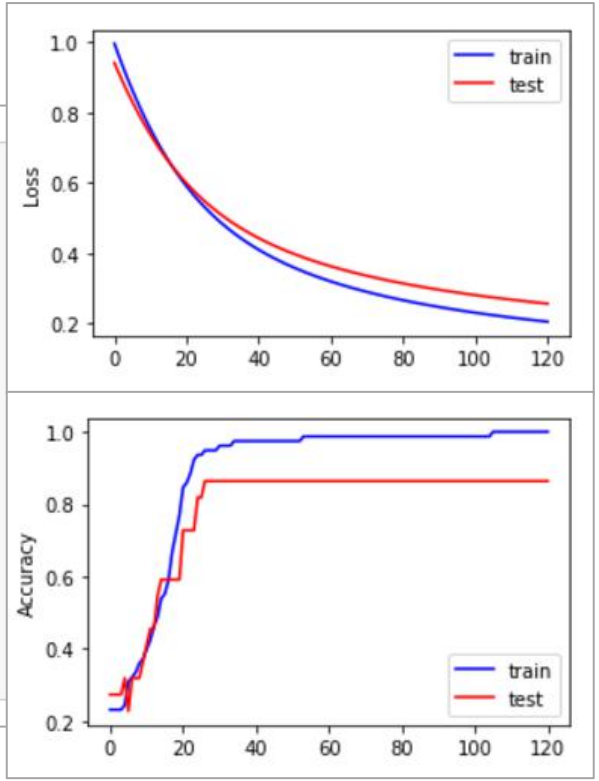

```
i: 0,  TrainAcc:0.230769,  TrainLoss: 0.994269 ,TestAcc:0.272727,  TestLoss: 0.939684
i: 30,  TrainAcc:0.961538,  TrainLoss: 0.481892 ,TestAcc:0.863636,  TestLoss: 0.505456
i: 60,  TrainAcc:0.987179,  TrainLoss: 0.319128 ,TestAcc:0.863636,  TestLoss: 0.362112
i: 90,  TrainAcc:0.987179,  TrainLoss: 0.246626 ,TestAcc:0.863636,  TestLoss: 0.295611
i: 120,  TrainAcc:1.000000,  TrainLoss: 0.204982 ,TestAcc:0.863636,  TestLoss: 0.256212
```

In [26]:

```
ce_train=[]
ce_test=[]
acc_train=[]
acc_test=[]

for i in range(0, iter+1):
    with tf.GradientTape() as tape:
        PRED_train =1/(1+tf.exp(-tf.matmul(X_train,W)))
        Loss_train =-tf.reduce_mean(Y_train*tf.math.log(PRED_train)+(1-Y_train)*tf.math.log(1-PRED_train))
        PRED_test =1/(1+tf.exp(-tf.matmul(X_test,W)))
        Loss_test =-tf.reduce_mean(Y_test*tf.math.log(PRED_test)+(1-Y_test)*tf.math.log(1-PRED_test))

    accuracy_train = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED_train.numpy()<0.5,0.,1.), Y_train),tf.float32))
    accuracy_test = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED_test.numpy()<0.5,0.,1.), Y_test),tf.float32))

    ce_train.append(Loss_train)
    ce_test.append(Loss_test)
    acc_train.append(accuracy_train)
    acc_test.append(accuracy_test)

    dL_dW= tape.gradient(Loss_train,W)
    W.assign_sub(learn_rate*dL_dW)

    if i % display_step == 0:
        print("i: %i,  TrainAcc:%f,  TrainLoss: %f ,TestAcc:%f,  TestLoss: %f" % (i,accuracy_train,Loss_train,accuracy_test,Loss_test))
```

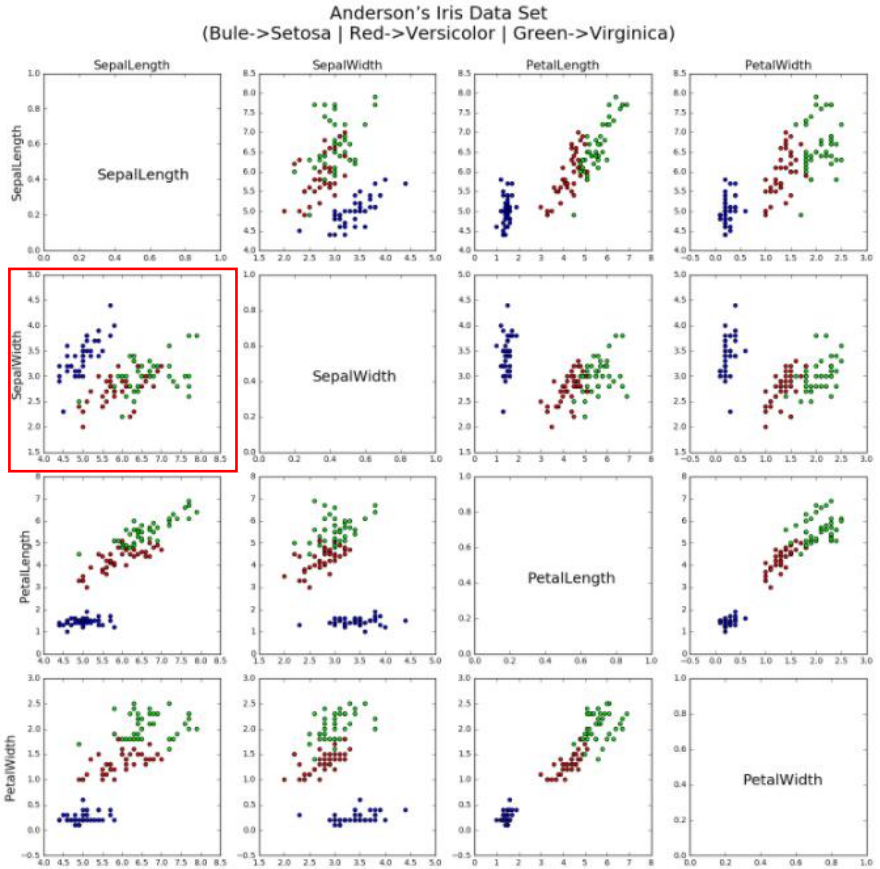
可视化

```
In [27]: plt.figure(figsize=(10,3))

         plt.subplot(121)
         plt.plot(ce_train,color="blue",label="train")
         plt.plot(ce_test,color="red",label="test")
         plt.ylabel("Loss")
         plt.legend()

         plt.subplot(122)
         plt.plot(acc_train,color="blue",label="train")
         plt.plot(acc_test,color="red",label="test")
         plt.ylabel("Accuracy")

         plt.legend()
         plt.show()
```



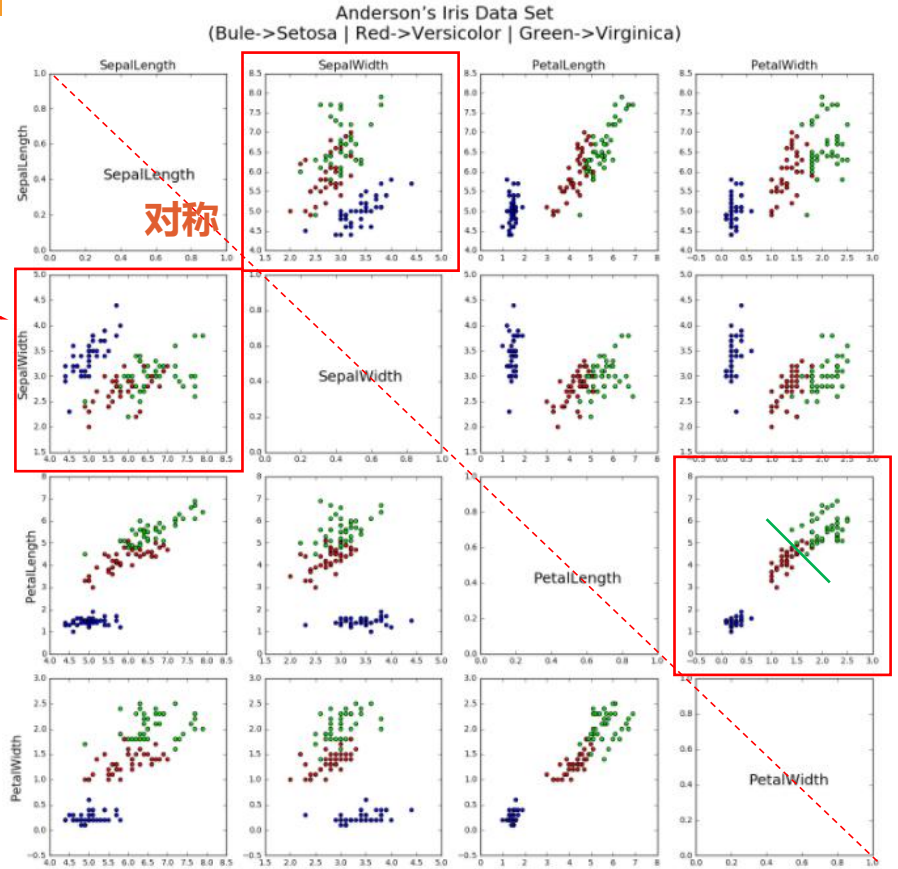
11.4 .1 实现多元逻辑回归

例程中的散点图



11.4.1 实现多元逻辑回归

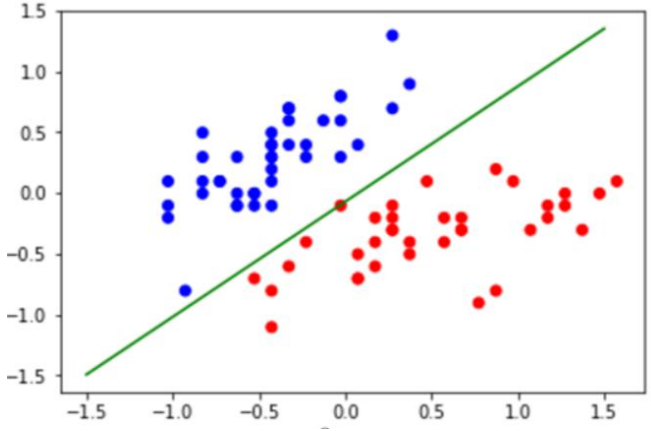
例程中的散点图

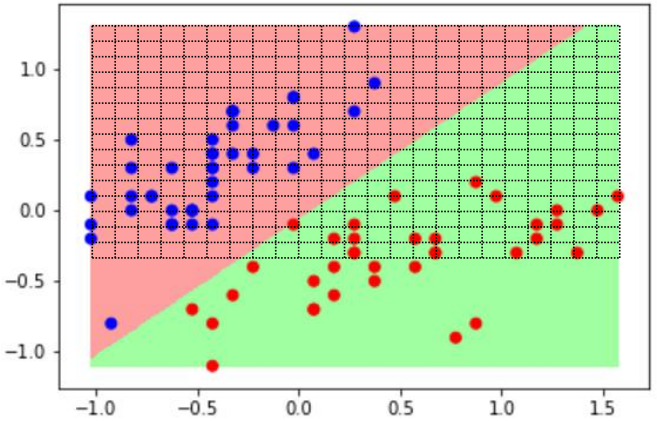



tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

11.4.2 绘制分类图

- 线性分类器
- 决策边界

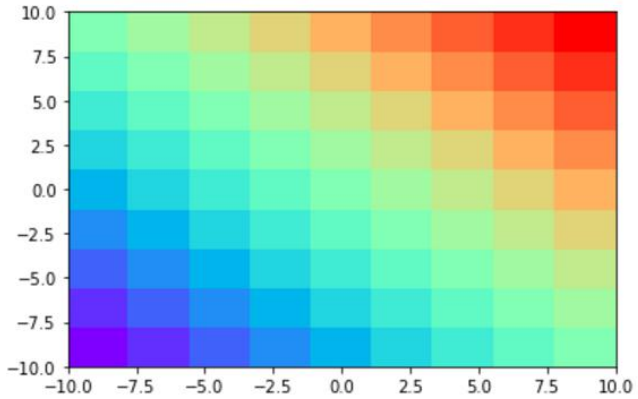

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

11.4.2 绘制分类图

生成网格坐标矩阵：np.meshgrid()

填充网格：plt.pcolomesh()

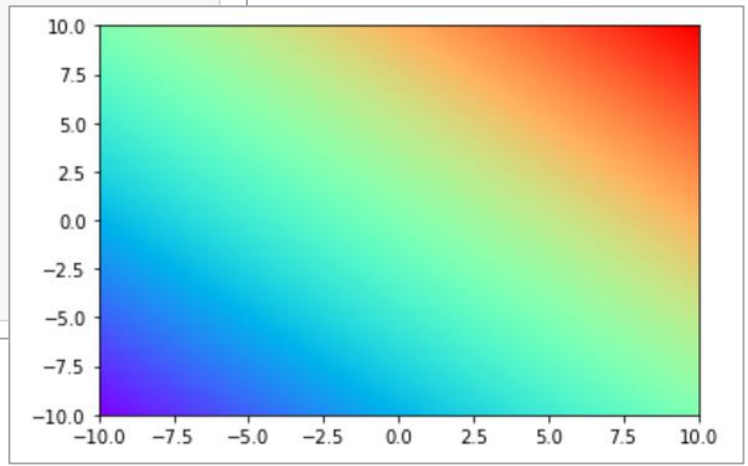



```
In [1]: import tensorflow as tf
        print("TensorFlow version:", tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import numpy as np
        import matplotlib as mpl
        import matplotlib.pyplot as plt

In [3]: n = 10

        x = np.linspace(-10, 10, n)
        y = np.linspace(-10, 10, n)

        X, Y = np.meshgrid(x, y)
        Z = X+Y

        plt.pcolormesh(X, Y, Z, cmap="rainbow")

        plt.show()
```

meshgrid()详见9.6小节

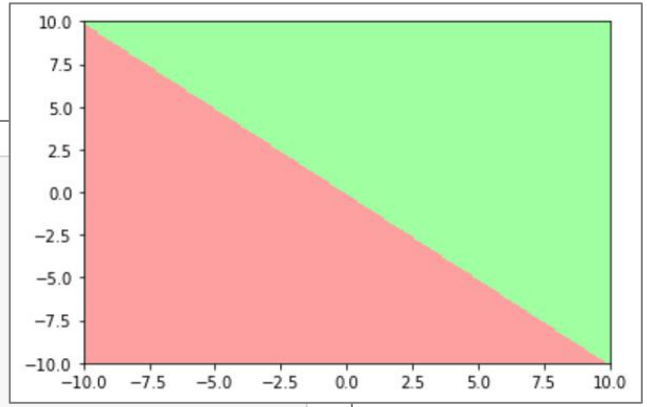
11.4.2 绘制分类图

```
In [4]: n = 200

        x = np.linspace(-10, 10, n)
        y = np.linspace(-10, 10, n)

        X, Y = np.meshgrid(x, y)
        Z = X+Y

        plt.pcolormesh(X, Y, Z, cmap="rainbow")

        plt.show()
```



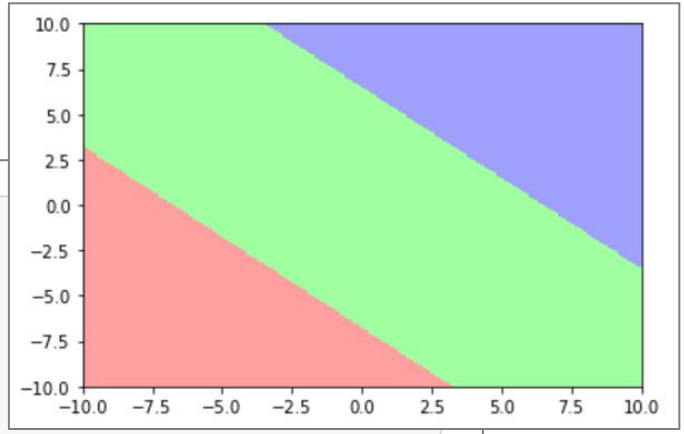
11.4.2 绘制分类图

```
In [5]: n = 200

        x = np.linspace(-10, 10, n)
        y = np.linspace(-10, 10, n)

        X, Y = np.meshgrid(x, y)
        Z = X+Y

        cm_bg = mpl.colors.ListedColormap(["#FFA0A0", "#A0FFA0"])
        plt.pcolormesh(X, Y, Z, cmap=cm_bg)

        plt.show()
```

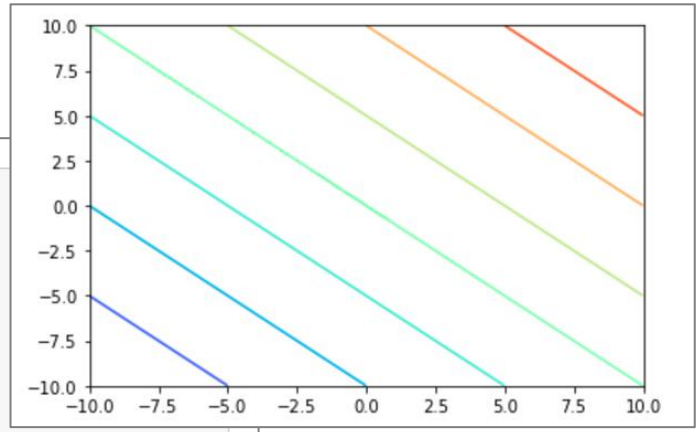


11.4.2 绘制分类图

```
In [6]: n = 200

        x = np.linspace(-10, 10, n)
        y = np.linspace(-10, 10, n)

        X, Y = np.meshgrid(x, y)
        Z = X+Y

        cm_bg = mpl.colors.ListedColormap(["#FFA0A0","#A0FFA0","#A0A0FF"])
        plt.pcolormesh(X, Y, Z, cmap=cm_bg)

        plt.show()
```



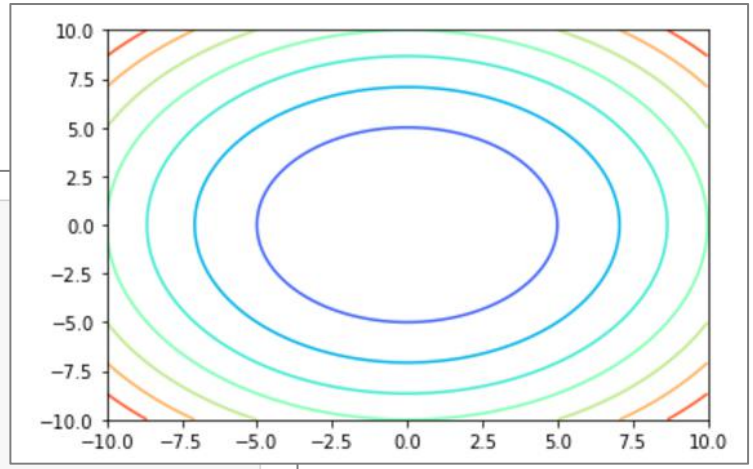
绘制轮廓线：plt.contour()



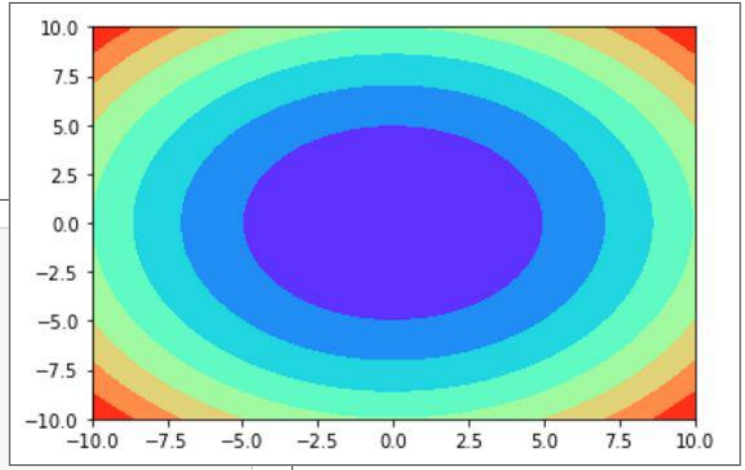

```
In [7]: n = 200

        x = np.linspace(-10, 10, n)
        y = np.linspace(-10, 10, n)

        X, Y = np.meshgrid(x, y)
        Z = X+Y

        plt.contour(X, Y, Z, cmap="rainbow")

        plt.show()
```

绘制轮廓线：plt.contour()

```
In [8]: n = 200

        x = np.linspace(-10, 10, n)
        y = np.linspace(-10, 10, n)

        X, Y = np.meshgrid(x, y)
        Z = X**2+Y**2

        plt.contour(X, Y, Z, cmap="rainbow")

        plt.show()
```

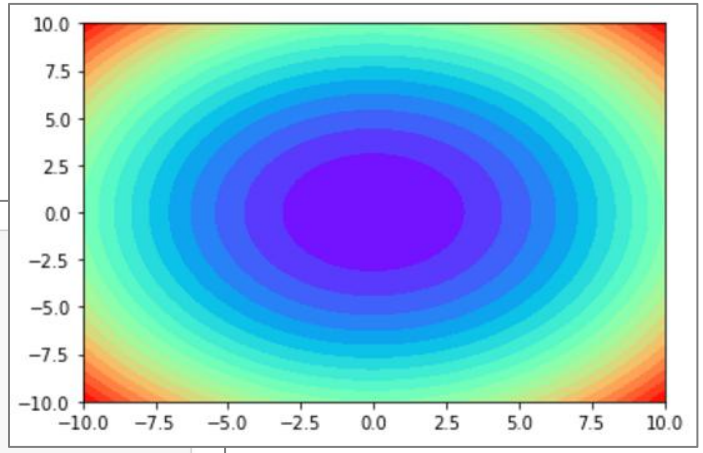


填充分区：plt.contourf()



In [9]:
```
n = 200

x = np.linspace(-10, 10, n)
y = np.linspace(-10, 10, n)

X, Y = np.meshgrid(x, y)
Z = X**2+Y**2

plt.contourf(X, Y, Z, cmap="rainbow")

plt.show()
```

填充分区：plt.contourf()

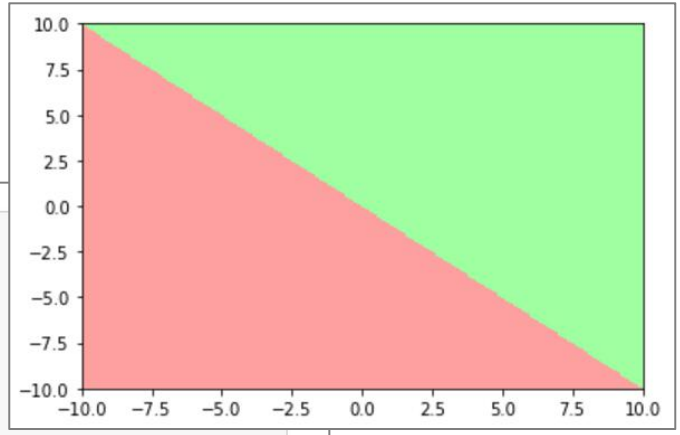



```
In [10]: n = 200

         x = np.linspace(-10, 10, n)
         y = np.linspace(-10, 10, n)

         X, Y = np.meshgrid(x, y)
         Z = X**2+Y**2

         plt.contourf(X, Y, Z, 20, cmap="rainbow")

         plt.show()
```

11.4.2 绘制分类图

生成网格坐标矩阵：np.meshgrid()

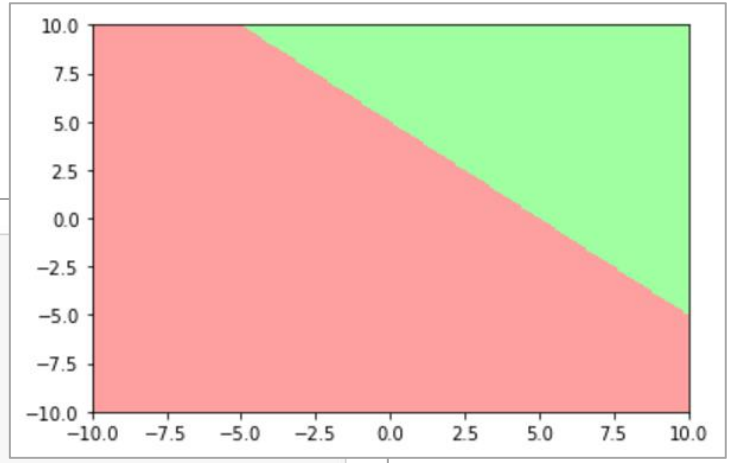
绘制分类图：pcolormesh()/plt.contourf()




```
In [11]: n = 200

         x = np.linspace(-10, 10, n)
         y = np.linspace(-10, 10, n)

         X, Y = np.meshgrid(x, y)
         Z = X+Y

         cm_bg = mpl.colors.ListedColormap(["#FFA0A0", "#A0FFA0"])
         Z=tf.where(Z<0, 0, 1)
         plt.pcolormesh(X, Y, Z, cmap=cm_bg)

         plt.show()
```

生成网格坐标矩阵：np.meshgrid()

绘制分类图：pcolormesh()/plt.contourf()




```
In [12]: n = 200

         x = np.linspace(-10, 10, n)
         y = np.linspace(-10, 10, n)

         X, Y = np.meshgrid(x, y)
         Z = X+Y

         cm_bg = mpl.colors.ListedColormap(["#FFA0A0", "#A0FFA0"])
         Z=tf.where(Z<5, 0, 1)
         plt.pcolormesh(X, Y, Z, cmap=cm_bg)

         plt.show()
```

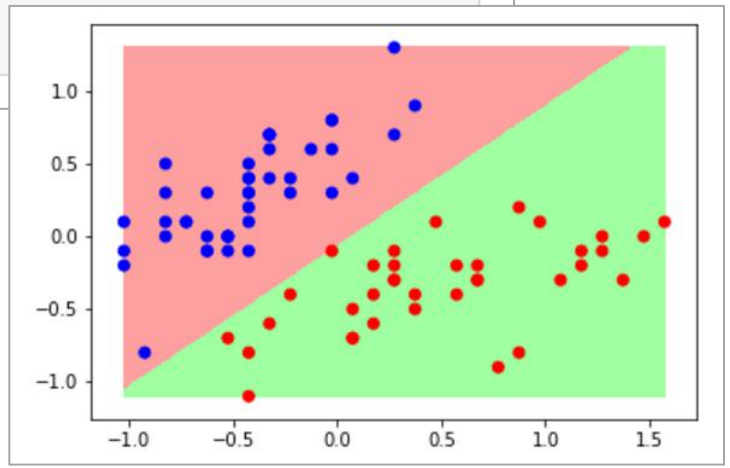
- 根据鸢尾花分类模型，绘制分类图

In [29]:
```
M=300
x1_min, x2_min = x_train.min(axis=0)
x1_max, x2_max = x_train.max(axis=0)
t1 = np.linspace(x1_min, x1_max, M)
t2 = np.linspace(x2_min, x2_max, M)
m1,m2 = np.meshgrid(t1, t2)
```

In [30]:
```
m0=np.ones(M*M)
X_mesh = tf.cast(np.stack((m0,m1.reshape(-1),m2.reshape(-1)), axis=1),dtype=tf.float32)
Y_mesh =tf.cast(1/(1+tf.exp(-tf.matmul(X_mesh,W))),dtype=tf.float32)
Y_mesh=tf.where(Y_mesh<0.5,0,1)
```

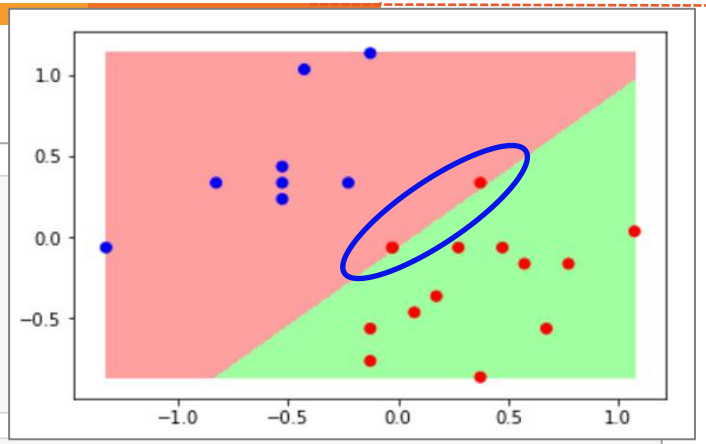
In [31]:
```
n=tf.reshape(Y_mesh,m1.shape)
```

```
In [32]: cm_pt = mpl.colors.ListedColormap(["blue", "red"])
         cm_bg = mpl.colors.ListedColormap(["#FFA0A0","#A0FFA0"])

         plt.pcolormesh(m1, m2, n, cmap=cm_bg)
         plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap=cm_pt)

         plt.show()
```



11.4.2 绘制分类图

0.863636=19/22



In [33]:
```
M=300
x1_min, x2_min = x_test.min(axis=0)
x1_max, x2_max = x_test.max(axis=0)
t1 = np.linspace(x1_min, x1_max, M)
t2 = np.linspace(x2_min, x2_max, M)
m1,m2 = np.meshgrid(t1, t2)
```

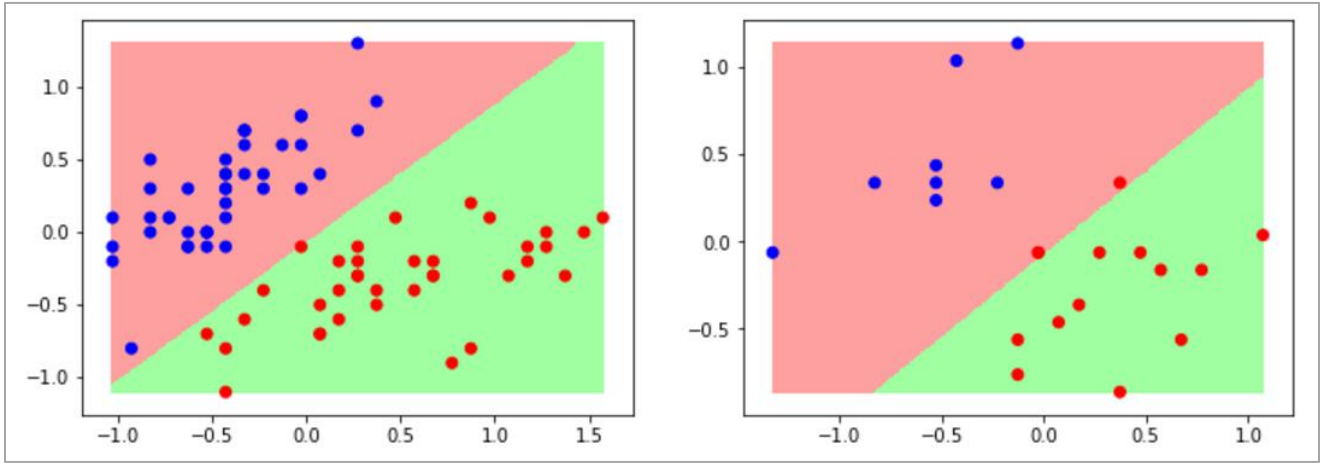
In [34]:
```
m0=np.ones(M*M)
X_mesh = tf.cast(np.stack((m0,m1.reshape(-1),m2.reshape(-1)), axis=1),dtype=tf.float32)
Y_mesh =tf.cast(1/(1+tf.exp(-tf.matmul(X_mesh,W))),dtype=tf.float32)
Y_mesh=tf.where(Y_mesh<0.5,0,1)
```

In [35]:
```
n=tf.reshape(Y_mesh,m1.shape)
```

In [36]:
```
plt.pcolormesh(m1, m2, n, cmap=cm_bg)
plt.scatter(x_test[:,0],x_test[:,1],c=y_test,cmap=cm_pt)
plt.show()
```

13	5.5	2.5	4	1.3	1
14	5.6	2.8	4.9	2	2
15	5.5	4.2	1.4	0.2	0
16	5.5	2.3	4	1.3	1
17	5.6	3	4.1	1.3	1
18	5.6	3	4.5	1.5	1
19	5.7	2.6	3.5	1	1
20	5.8	2.7	3.9	1.2	1
21	5.7	2.5	5	2	2
22	5.9	3	4.2	1.5	1

- 分别绘制**训练集**和**测试集**的分类图

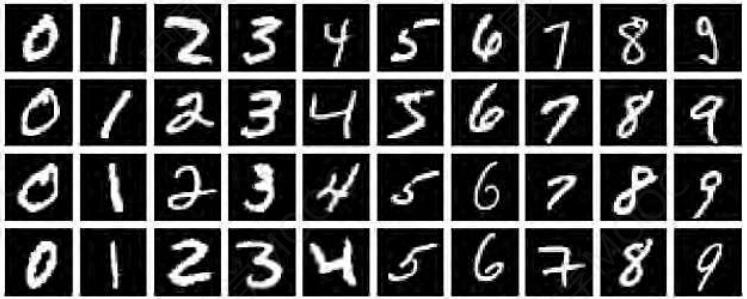



tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

11.5 多分类问题

逻辑回归：二分类问题

多分类问题：把输入样本划分为多个类别



自然顺序码

0——山鸢尾（Setosa）

1——变色鸢尾（Versicolour）

2——维吉尼亚鸢尾（Virginica）

独冷编码（one cold）

(0,1,1)——山鸢尾（Setosa）

(1,0,1)——变色鸢尾（Versicolour）

(1,1,0)——维吉尼亚鸢尾（Virginica）

独热编码（One-Hot Encoding）

- 使非偏序关系的数据，取值不具有偏序性
- 到原点等距

(1,0,0)——山鸢尾（Setosa）

(0,1,0)——变色鸢尾（Versicolour）

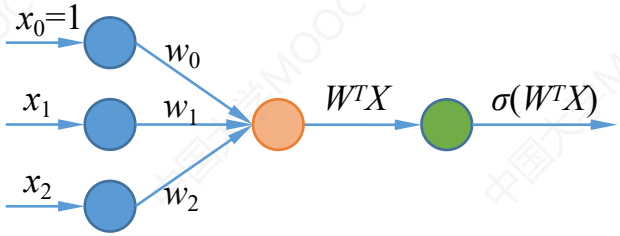
(0,0,1)——维吉尼亚鸢尾（Virginica）

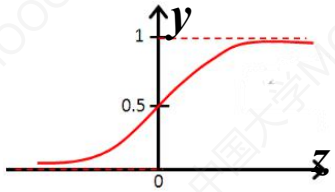
应用

离散的特征

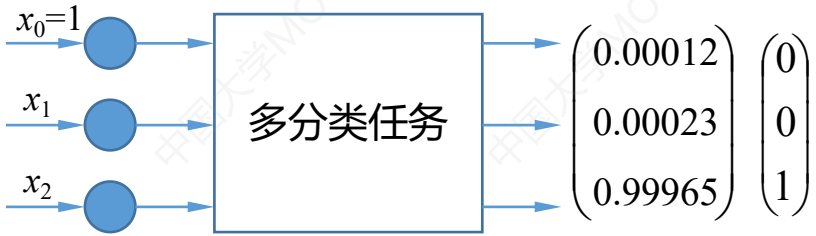
多分类问题中的类别标签

二分类





多分类

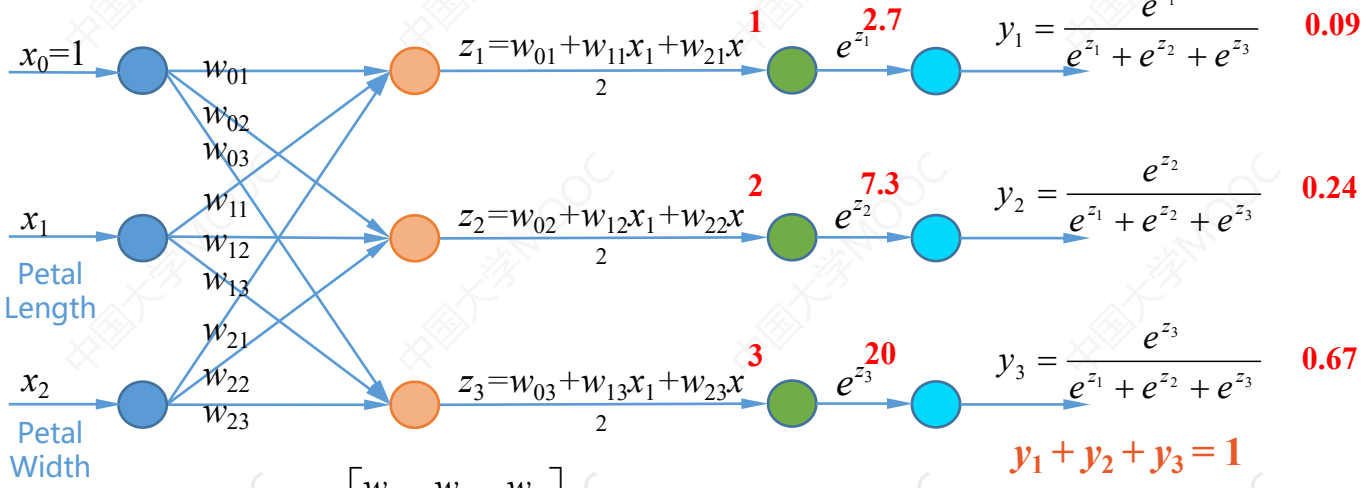


模型输出　标签

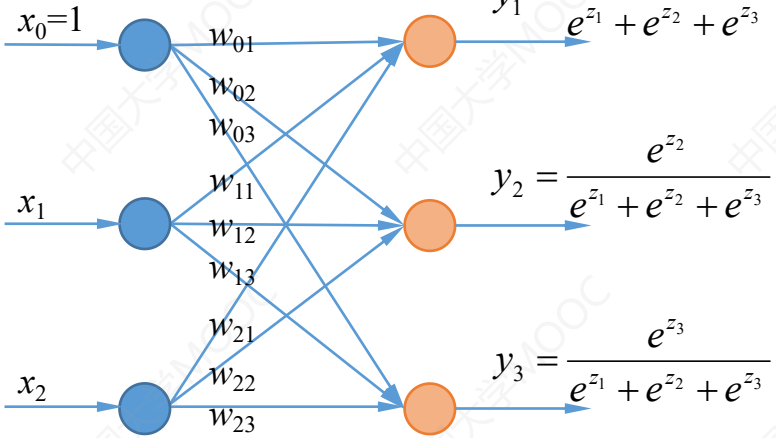
softmax()函数 $Y = soft\max(W^T X)$

广义线性回归, 实现多分类

例：使用属性**花瓣长度**和**花瓣宽度**，构造分类器，能够**识别3种类型**的鸢尾花

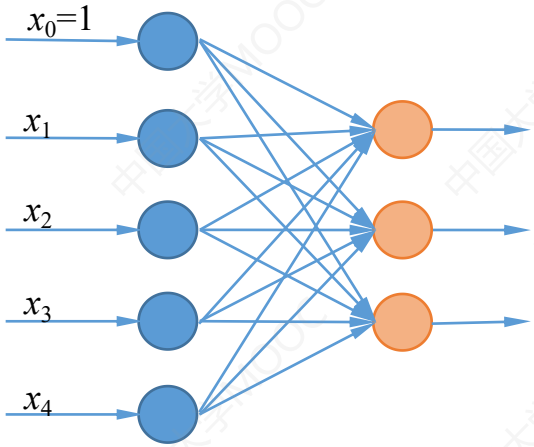


$$W=\begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$



$$W = \begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

Softmax函数是Logistic函数在**多分类**问题上的推广



$$y_1=\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$$y_2=\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$$y_3=\frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$$y_k=\frac{e^{z_k}}{\sum_{p=1}^{C}e^{z_p}}$$

k：输出的索引

z_k：第k个输出接收到的所有输入的线性组合

C：类别总数

$$W=\begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

$$z_1=w_{01}+w_{11}x_1+w_{21}x_2+w_{31}x_3+w_{41}x_4$$

$$z_1=w_{02}+w_{12}x_1+w_{22}x_2+w_{32}x_3+w_{42}x_4$$

$$z_1=w_{03}+w_{13}x_1+w_{23}x_2+w_{33}x_3+w_{43}x_4$$

- **二元交叉熵损失函数** (BCE)

$$Loss = -\sum_{i=1}^{n}[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)]$$

- **多分类交叉熵损失函数** (CCE)

$$Loss = -\sum_{i=1}^{n}\sum_{p=1}^{C} y_{i,p} \ln(\hat{y}_{i,p})$$

$$y_k = \frac{e^{z_k}}{\sum_{p=1}^{C} e^{z_p}}$$

$$Loss = -\sum_{i=1}^{n}\sum_{p=1}^{C} y_{i,p} \ln(\hat{y}_{i,p})$$

i：样本的索引

n：样本的总数

$y_{i,p}$：第i个样本属于第p类的标记值，0/1

$\hat{y}_{i,p}$：第i个样本属于第p类的预测概率

C：类别总数

	样本	标记	预测值	结果判断
模型A	样本1	0 0 1	0.3 0.3 0.4	正确
	样本2	0 1 0	0.3 0.4 0.3	正确
	样本3	1 0 0	0.1 0.2 0.7	错误
模型B	样本1	0 0 1	0.1 0.2 0.7	正确
	样本2	0 1 0	0.1 0.7 0.2	正确
	样本3	1 0 0	0.3 0.4 0.3	错误

准确率：

模型A=模型B= 2/3 = 66.7%

交叉熵损失：

模型A

样本1：-(0×ln0.3 + 0×ln0.3+1×ln0.4)= -ln0.4=0.9162...

样本2：-(0×ln0.3 + 1×ln0.4+0×ln0.3)= -ln0.4=0.9612...

样本3：-(1×ln0.1 + 0×ln0.2+0×ln0.7)= -ln0.1=2.3025...

交叉熵损失和：**5.9677...**

模型B

样本1：-(0×ln0.1 + 0×ln0.2+1×ln0.7)= -ln0.7=0.3566...

样本2：-(0×ln0.1 + 1×ln0.7+0×ln0.2)= -ln0.7=0.3566...

样本3：-(1×ln0.3 + 0×ln0.4+0×ln0.3)= -ln0.3=1.2039...

交叉熵损失和：**1.9173...**

- **互斥的多分类问题**：每个样本只能够属于**一个类别**

  鸢尾花的识别

  手写数字识别

- **非互斥的多分类问题**：一个样本可以同时属于**多个类别**

  构建**多个**一对多的**逻辑回归**

  标签：

  彩色图片

  包括人物的图片

  包括汽车的图片

  户外图片

  室内图片

神经网络&深度学习
TENSORFLOW
Google

11.6 实例：实现多分类问题

中国大学MOOC

独热编码

一维数组/张量　　编码深度

```
one_hot ( indices, depth )
```

```
In [1]: import tensorflow as tf
        print("TensorFlow version:",tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import numpy as np

In [3]: a=[0,2,3,5]
        b=tf.one_hot(a,6)
        b

Out[3]: <tf.Tensor: id=4, shape=(4, 6), dtype=float32, numpy=
        array([[1., 0., 0., 0., 0., 0.],  0
               [0., 0., 1., 0., 0., 0.],  2
               [0., 0., 0., 1., 0., 0.],  3
               [0., 0., 0., 0., 0., 1.]],  5 dtype=float32)>
```

准确率

```
In [4]: pred=np.array([[0.1,0.2,0.7],
                       [0.1,0.7,0.2],
                       [0.3,0.4,0.3]])
        y=np.array([2,1,0])
        y_onehot=np.array([[0,0,1],
                           [0,1,0],
                           [1,0,0]])

In [5]: tf.argmax(pred,axis=1)
Out[5]: <tf.Tensor: id=7, shape=(3,), dtype=int64, numpy=array([2, 1, 1], dtype=int64)>

In [6]: tf.equal(tf.argmax(pred,axis=1),y)
Out[6]: <tf.Tensor: id=12, shape=(3,), dtype=bool, numpy=array([ True,  True, False])>

In [7]: tf.cast(tf.equal(tf.argmax(pred,axis=1),y),tf.float32)
Out[7]: <tf.Tensor: id=18, shape=(3,), dtype=float32, numpy=array([1., 1., 0.], dtype=float32)>

In [8]: tf.reduce_mean(tf.cast(tf.equal(tf.argmax(pred,axis=1),y),tf.float32))
Out[8]: <tf.Tensor: id=26, shape=(), dtype=float32, numpy=0.6666667>
```

预测值

标记

标记独热编码

预测值中的最大数索引

判读预测值是否与样本标记相同

将布尔值转化为数字

准确率

交叉熵损失函数

$$Loss = -\sum_{i=1}^{n}\sum_{p=1}^{C} y_{i,p}\ln(\hat{y}_{i,p})$$

```
In [9]: -y_onehot*tf.math.log(pred)

Out[9]: <tf.Tensor: id=30, shape=(3, 3), dtype=float64, numpy=
        array([[-0.        , -0.        ,  0.35667494],   样本1
               [-0.        ,  0.35667494, -0.        ],   样本2
               [ 1.2039728 , -0.        , -0.        ]])>样本3

In [10]: -tf.reduce_sum(y_onehot*tf.math.log(pred))        所有样本交叉熵之和

Out[10]: <tf.Tensor: id=37, shape=(), dtype=float64, numpy=1.917322692203401>

In [11]: -tf.reduce_sum(y_onehot*tf.math.log(pred))/len(pred)        平均交叉熵损失

Out[11]: <tf.Tensor: id=46, shape=(), dtype=float64, numpy=0.6391075640678003>
```

例：使用**花瓣长度**、**花瓣宽度**将**三种**鸢尾花区分开

- **加载数据**

```
In [1]: import tensorflow as tf
        print("TensorFlow version:", tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import pandas as pd
        import numpy as np
        import matplotlib as mpl
        import matplotlib.pyplot as plt

In [3]: TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
        train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1], TRAIN_URL)

In [4]: df_iris_train = pd.read_csv(train_path, header=0)
```

处理数据

```
In [5]: iris_train=np.array(df_iris_train)

In [6]: iris_train.shape
Out[6]: (120, 5)

In [7]: x_train=iris_train[:,2:4]
        y_train=iris_train[:,4]

In [8]: x_train.shape,y_train.shape
Out[8]: ((120, 2), (120,))

In [9]: num_train=len(x_train)
```

提取花瓣长度、花瓣宽度属性

处理数据

```
In [10]: x0_train = np.ones(num_train).reshape(-1,1)
         X_train=tf.cast(tf.concat([x0_train,x_train],axis=1),tf.float32)
         Y_train=tf.one_hot(tf.constant(y_train,dtype=tf.int32),3)

In [11]: X_train.shape,Y_train.shape

Out[11]: (TensorShape([120, 3]), TensorShape([120, 3]))
```

设置超参数、设置模型参数初始值

```
In [12]: learn_rate = 0.2
         iter = 500
         display_step =100

In [13]: np.random.seed(612)
         W = tf.Variable(np.random.randn(3,3),dtype=tf.float32)
```

训练模型

```
In [14]: acc=[]
         cce=[]

         for i in range(0,iter+1):
             with tf.GradientTape() as tape:

                 PRED_train=tf.nn.softmax(tf.matmul(X_train,W))
                 Loss_train=-tf.reduce_sum(Y_train*tf.math.log(PRED_train))/num_train

             accuracy=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_train.numpy(),axis=1),y_train),tf.float32))

             acc.append(accuracy)
             cce.append(Loss_train)

             dL_dW = tape.gradient(Loss_train,W)
             W.assign_sub(learn_rate*dL_dW)

             if i % display_step == 0:
                 print("i: %i, Acc: %f, Loss: %f" % (i, accuracy, Loss_train))
```

```
i: 0, Acc: 0.350000, Loss: 4.510763
i: 100, Acc: 0.808333, Loss: 0.503537
i: 200, Acc: 0.883333, Loss: 0.402912
i: 300, Acc: 0.891667, Loss: 0.352650
i: 400, Acc: 0.941667, Loss: 0.319778
i: 500, Acc: 0.941667, Loss: 0.295599
```

训练结果

```
In [15]: PRED_train.shape
Out[15]: TensorShape([120, 3])
```

属于每种类别的概率

```
In [16]: tf.reduce_sum(PRED_train,axis=1)
```

概率之和为1

```
Out[16]: <tf.Tensor: id=24068, shape=(120,), dtype=float32, numpy=
array([1.        , 1.        , 1.        , 0.9999999 , 1.        ,
       0.99999994, 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.0000001 , 1.        , 1.        , 1.        ,
       1.        , 0.99999994, 0.99999994, 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.9999999 , 1.        ,
       0.99999994, 0.99999994, 1.        , 1.        , 1.        ,
       1.        , 0.99999994, 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.0000001 ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.0000001 , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
```

训练结果

转换为自然顺序码

```
In [17]: tf.argmax(PRED_train.numpy(),axis=1)

Out[17]: <tf.Tensor: id=24071, shape=(120,), dtype=int64, numpy=
         array([2, 1, 2, 0, 0, 0, 0, 2, 1, 0, 1, 1, 0, 0, 2, 2, 2, 2, 2, 0, 2, 2,
                0, 1, 1, 0, 1, 2, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 0,
                0, 1, 0, 2, 0, 2, 0, 1, 1, 0, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 1, 2,
                0, 2, 2, 0, 0, 1, 0, 2, 2, 0, 1, 1, 1, 2, 0, 1, 1, 1, 2, 0, 1, 1,
                2, 0, 2, 1, 0, 0, 2, 0, 0, 2, 2, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
                2, 1, 0, 2, 0, 1, 1, 0, 0, 1], dtype=int64)>
```

绘制分类图

In [18]:
```
M=500
x1_min, x2_min = x_train.min(axis=0)
x1_max, x2_max = x_train.max(axis=0)
t1 = np.linspace(x1_min, x1_max, M)
t2 = np.linspace(x2_min, x2_max, M)
m1,m2 = np.meshgrid(t1, t2)
```

In [19]:
```
m0=np.ones(M*M)
X_ = tf.cast(np.stack((m0,m1.reshape(-1),m2.reshape(-1)), axis=1),tf.float32)
Y_ =tf.nn.softmax(tf.matmul(X_,W))
```

In [20]:
```
Y_=tf.argmax(Y_.numpy(),axis=1)
```

转换为自然顺序码，决定网格颜色

绘制分类图

```
In [21]: n=tf.reshape(Y_,m1.shape)
```

和m1形状相同

```
In [22]: n
Out[22]: <tf.Tensor: id=24081, shape=(500, 500), dtype=int64, numpy=
         array([[0, 0, 0, ..., 1, 1, 1],
                [0, 0, 0, ..., 1, 1, 1],
                [0, 0, 0, ..., 1, 1, 1],
                ...,
                [2, 2, 2, ..., 2, 2, 2],
                [2, 2, 2, ..., 2, 2, 2],
                [2, 2, 2, ..., 2, 2, 2]], dtype=int64)>
```

绘制分类图

```
In [25]: plt.figure(figsize=(8,6))

         cm_bg = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])

         plt.pcolormesh(m1, m2, n, cmap=cm_bg)
         plt.scatter(x_train[:,0],x_train[:,1],c=y_train,cmap="brg")

         plt.show()
```