MNIST 手写字识别程序入门

一、部分基础

```
import numpy as np
test = np.array([[1,2,3],
                 [2,3,4],
                 [5,4,3],
                 [8,7,2]])
np.argmax(test,1)
```

```
array([2, 2, 0, 0])
```

1.1 线性回归基础

1.1.1 最小二乘法曲线拟合问题

1.1.2 模型假设

理想的线性模型

$$y = 0.99x + 9.31; x \in [1:1:10]$$

假设的线性模型

$$f(x;a,b) = ax + b$$

1.1.3 最小二乘法确定参数

$$f(x;a,b) = ax + b$$

均方误差函数

$$S = \sum_{i=1}^{n}(y_i - (ax_i + b))^2$$

对误差求极限，找误差函数的最小值点

$$\frac{\partial S}{\partial a} = -2(\sum_{i=1}^{n} x_i y_i - b\sum_{i=1}^{n} nx_i - a\sum_{i=1}^{n} x_i^2)$$
$$\frac{\partial S}{\partial b} = -2(\sum_{i=1}^{n} y_i - nb - a\sum_{i=1}^{n} x_i)$$

令偏导数为零，求得凸函数的极小值点处的a,b 值

$$a = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum {x_i}^2 - (\sum x_i)^2}$$
$$b = \frac{\sum {x_i}^2 \sum y_i - \sum x_i \sum x_i y_i}{n\sum {x_i}^2 - (\sum x_i)^2}$$

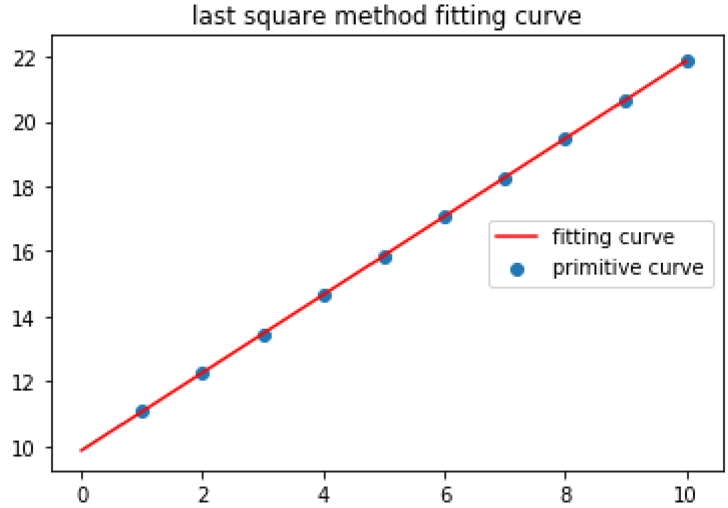
下面用两种代码的实现方式实现上述曲线拟合过程

- 用户自定义函数代码实现

- 调用numpy的最小二乘法的线性拟合问题lstsq函数实现

```
# 用户自定义函数代码实现
import numpy as np
import matplotlib.pyplot as plt

def calcAB(x,y):
    n = len(x)
    sumX,sumY,sumXY,sumXX =0,0,0,0
    for i in range(0,n):
        sumX  += x[i]
        sumY  += y[i]
        sumXX += x[i]*x[i]
        sumXY += x[i]*y[i]
    a = (n*sumXY -sumX*sumY)/(n*sumXX -sumX*sumX)
    b = (sumXX*sumY - sumX*sumXY)/(n*sumXX-sumX*sumX)
    return a,b,

# xi = [1,2,3,4,5,6,7,8,9,10]
# yi = [10,11.5,12,13,14.5,15.5,16.8,17.3,18,18.7]
xi = [1,2,3,4,5,6,7,8,9,10]
# yi = [1 for i in range(10)]
yi = [0] * 10
print(yi)
for num in xi:
  yi[num -1] = num*1.201030944 + 9.8678999766

a,b=calcAB(xi,yi)
print("y = 1.201030944*x + 9.8678999766", '\n', "f(x;a,b) = (a)%3.5fx + (b)%3.5f
x = np.linspace(0,10)
y = a * x + b
plt.plot(x,y,'red', label='fitting curve')
plt.scatter(xi,yi, label='primitive curve')
plt.legend(loc='right')
plt.title('last square method fitting curve')
plt.show()
```

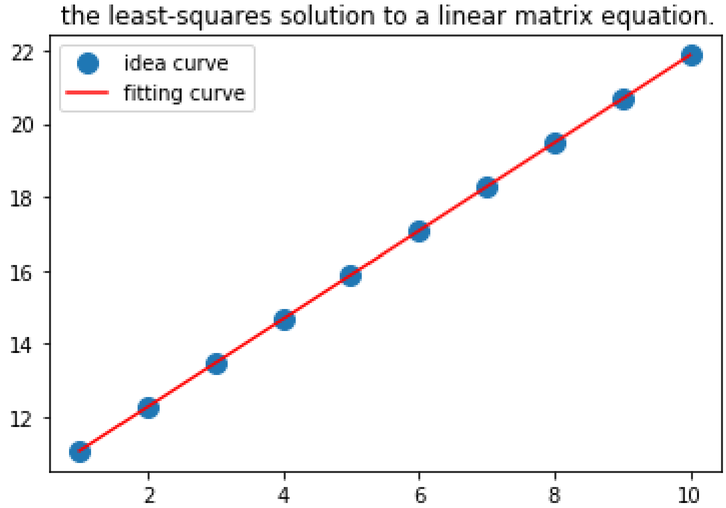
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
y = 1.201030944*x + 9.8678999766
 f(x;a,b) = (a)1.20103x + (b)9.86790
```




```
# 最小二乘法的线性拟合问题lstsq函数实现
import numpy as np
import matplotlib.pyplot as plt

x = [1,2,3,4,5,6,7,8,9,10]
# y = [10,11.5,12,13,14.5,15.5,16.8,17.3,18,18.7]
y = [0] * 10
for num in x:
  y[num -1] = num*1.201030944 + 9.8678999766
```

```
A = np.vstack([x,np.ones(len(x))]).T

a,b = np.linalg.lstsq(A,y, rcond=-1)[0]
print("1.理想曲线:","y = 1.201030944*x + 9.8678999766")
print('2.拟合曲线:', "f(x;a,b) = %10.5fx + %10.5f" %(a,b))
x = np.array(x)
y = np.array(y)

plt.plot(x,y,'o',label='idea curve',markersize=10)
plt.plot(x,a*x+b,'r',label='fitting curve')
plt.legend(loc='upper left')
plt.title(' the least-squares solution to a linear matrix equation.')
plt.show()
```

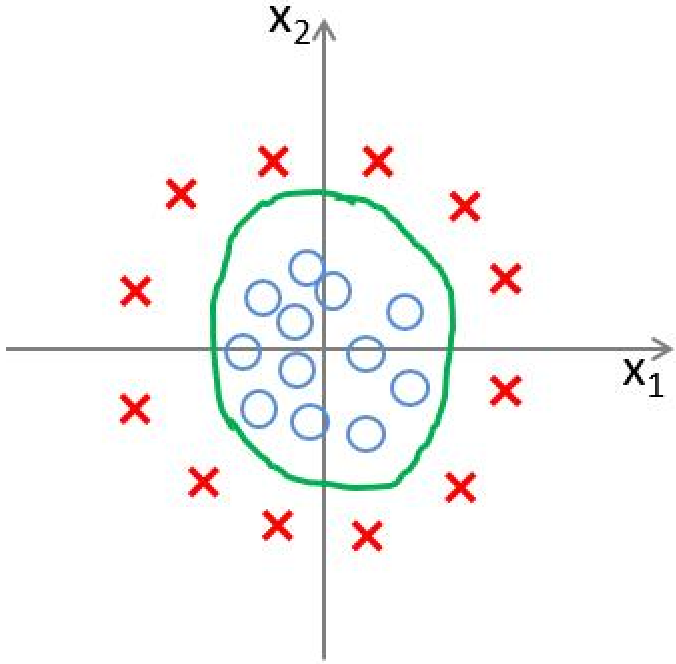
```
1.理想曲线: y = 1.201030944*x + 9.8678999766
2.拟合曲线: f(x;a,b) =    1.20103x +    9.86790
```

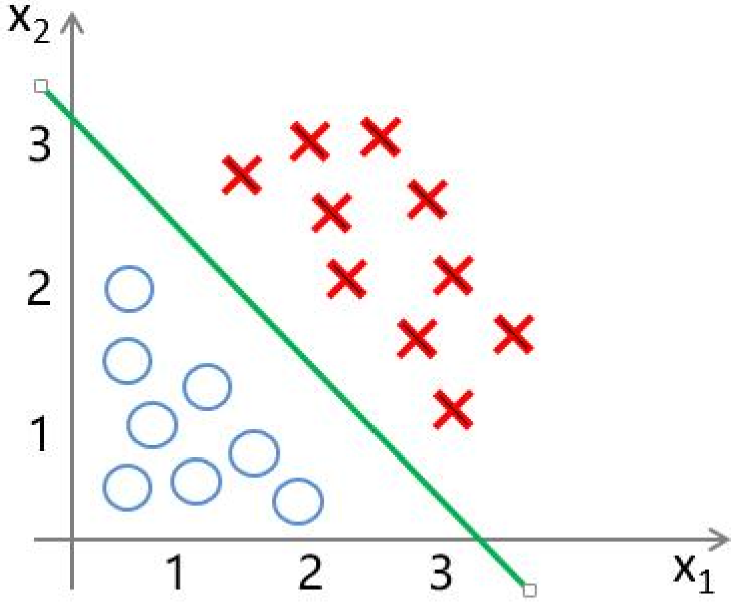



```
# 矩阵的转置
import numpy as np

x = np.array([[1, 2, 3, 4, 5],[1, 2, 3, 5, 4]])
print(x,x.T, x.transpose())
```

```
[[1 2 3 4 5]
 [1 2 3 5 4]] [[1 1]
 [2 2]
 [3 3]
 [4 5]
 [5 4]] [[1 1]
 [2 2]
 [3 3]
 [4 5]
 [5 4]]
```

1.2 逻辑回归问题





1.2.1 线性可分逻辑回归模型

- 本文只针对--->线性可分且特征数为n种的情况下，目标找到n维的超平面

$$z(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \ldots + \theta_n x_n^{(i)}$$

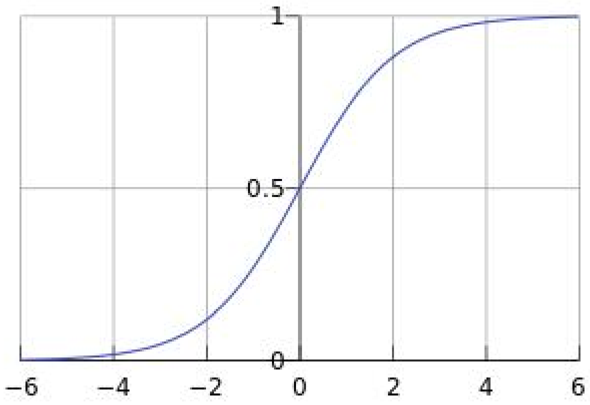
1.2.2 线性输出转逻辑输出

逻辑回归作为分类算法，和前面的线性回归模型的最大不同之处在于它的输出是 **0/1** 。那么如何将输出值转换成0/1呢？在此介绍一下sigmoid 函数。

- sigmoid 函数定义

$$sigmoid function; g(z) = \frac{1}{1+e^{-z}}$$

- 函数图像如下



函数的意义如下公式所示

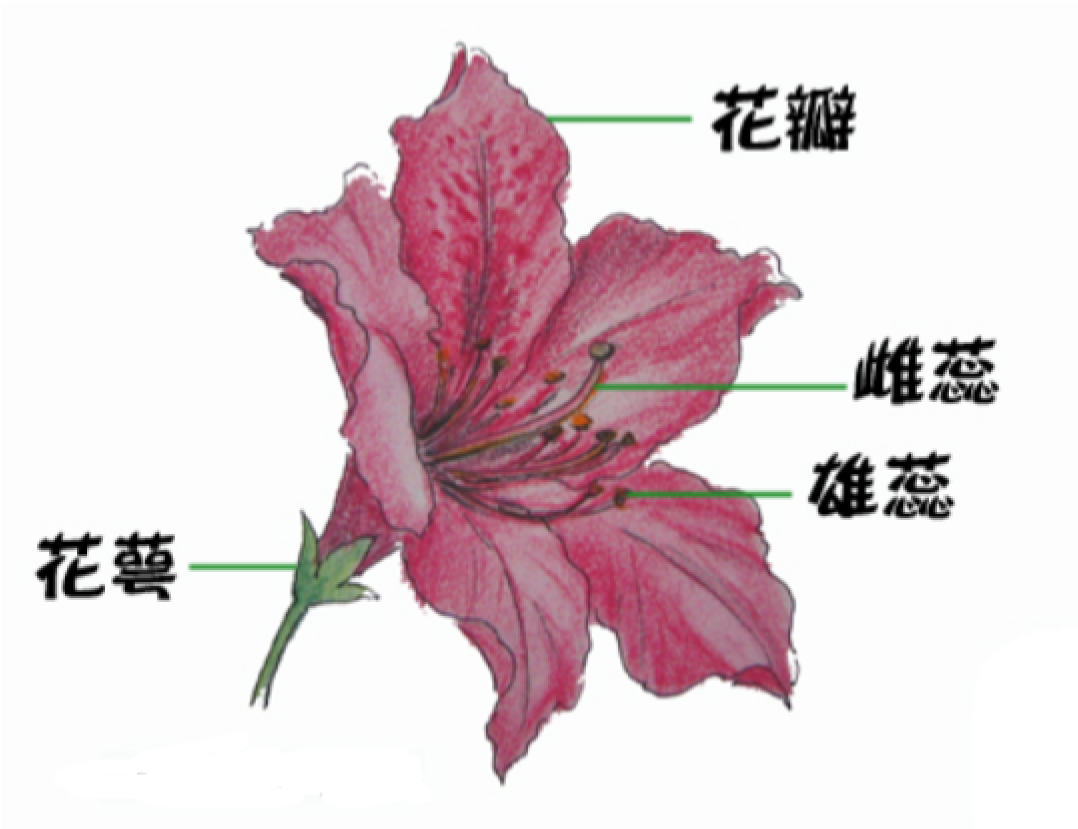
$$y = \begin{cases} 1, & \text{if } g(z) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

其中 y 表示分类结果。sigmoid 函数实际表达的是将样本分为“1”类的概率，这将在本文的最后一部分进行详细解释。

1.3 多分类问题

1.3.1 Iris莺尾花数据分类(3种结果分类)

接下来我们以莺尾花数据集进行多分类(根据4个特征进行3种花的分类问题)



1.3.2 莺尾花数据集介绍

下面截取iris数据集原始iris.csv文件中的部分内容

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
......
```

Iris 数据集中对原始iris.csv文件中的数据解释如下

	sepal_len	sepal_width	petal_len	petal_width	class
0	4.9	3.0	1.4	0.2	Iris-Setosa
1	4.7	3.2	1.3	0.2	Iris-Setosa
2	4.6	3.1	1.5	0.2	Iris-Setosa
3	5.0	3.6	1.4	0.2	Iris-Setosa
...	...	...	...	...	...

每个数据都是以都好分隔,换行符结束。

下面将用代码的方式，了解iris我们能获取到的数据有哪些。

```
import numpy as np
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()

print(iris.target)
print(type(iris.target),len(iris.target),type(iris))

print(len(iris.data), iris.data.shape, iris.data[range(1,150,1),3])
print("\n")
print(iris.feature_names,iris.target_names,iris.filename)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

1.3.3 对Iris数据可视化操作

```
<class 'numpy.ndarray'> 150 <class 'sklearn.utils.Bunch'>
```

```
import pandas as pd

# 读入数据
df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/iris/

'''
数据时以逗号为分隔符的，
但是这个数据没有列的名字，
所以先给每个列取个名字，
直接使用数据说明中的描述
'''

df.columns = ['sepal_len', 'sepal_width', 'petal_len', 'petal_width', 'class']

# 查看前5条数据
df.head()
```

	sepal_len	sepal_width	petal_len	petal_width	class
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

```
'''
最后类别一列，感觉前面的'Iris-'有点多余
即把class这一列的数据按'-'进行切分
取切分后的第二个数据，为了好看一点点
'''

df['class'] = df['class'].apply(lambda x: x.split('-')[1])

# 查看数据信息
df.describe()
```

	sepal_len	sepal_width	petal_len	petal_width
count	149.000000	149.000000	149.000000	149.000000
mean	5.848322	3.051007	3.774497	1.205369
std	0.828594	0.433499	1.759651	0.761292
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

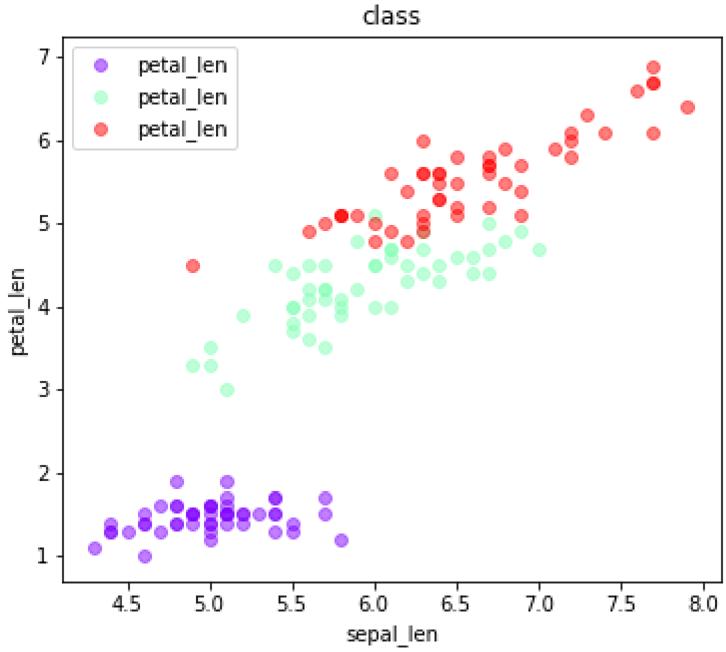
使用describe()可以很方便的查看数据的大致信息，可以看到数据是没有缺失值的，总共有145条，每一列的最大值、最小值、平均值都可以查看。

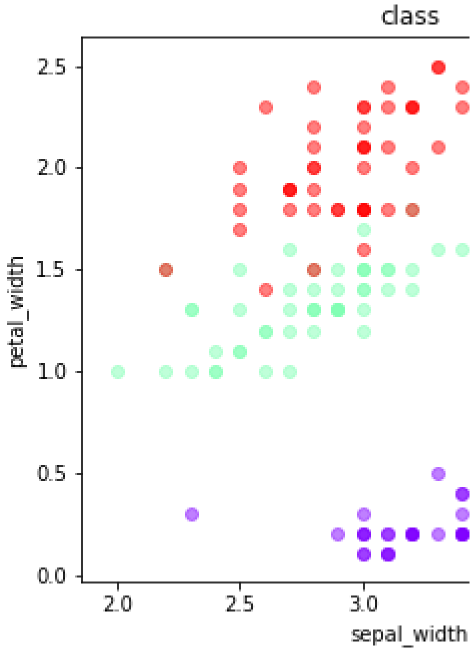
1.3.4 Iris 数据可视化

为了比较直观的查看数据的分布，用matplotlib进行了简单的可视化展示，查看数据的分布，画个图。

```
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline

def scatter_plot_by_category(feat, x, y):
    alpha = 0.5
    gs = df.groupby(feat)
    cs = cm.rainbow(np.linspace(0, 1, len(gs)))
    for g, c in zip(gs, cs):
        plt.scatter(g[1][x], g[1][y], color=c, alpha=alpha)

plt.figure(figsize=(20,5))

plt.subplot(131)
scatter_plot_by_category('class', 'sepal_len', 'petal_len')
plt.xlabel('sepal_len')
plt.ylabel('petal_len')
plt.legend(loc='upper left')
plt.title('class')

# 为了节省篇幅，省了第二、三个图的代码
plt.subplot(132)
scatter_plot_by_category('class', 'sepal_width', 'petal_width')
plt.xlabel('sepal_width')
plt.ylabel('petal_width')
plt.title('class')

plt.subplot(133)
scatter_plot_by_category('class', 'sepal_len', 'sepal_width')
plt.xlabel('sepal_len')
plt.ylabel('sepal_width')
plt.title('class')
```

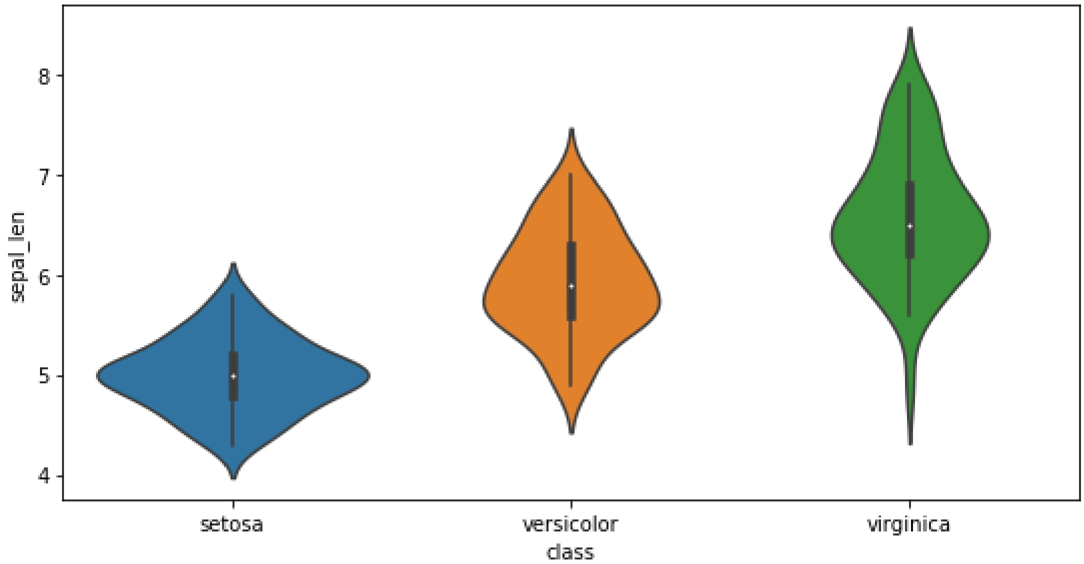
```
Text(0.5, 1.0, 'class')
```

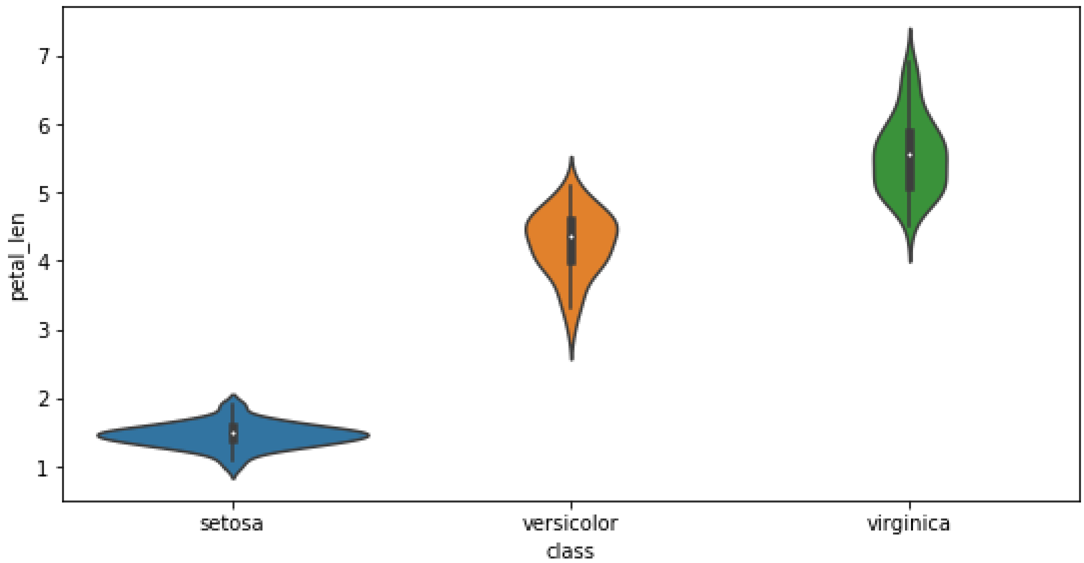





```
import seaborn as sb
```

```
plt.figure(figsize=(20, 10))
for column_index, column in enumerate(df.columns):
    if column == 'class':
        continue
    plt.subplot(2, 2, column_index + 1)
    sb.violinplot(x='class', y=column, data=df)
```





sepal_width

petal_width

1.3.5 Iris 模型训练

以下提供两种模型训练方法，一种是线性回归的方法，另一种是决策树的方法进行模型的训练

线性回归方法进行Iris模型训练

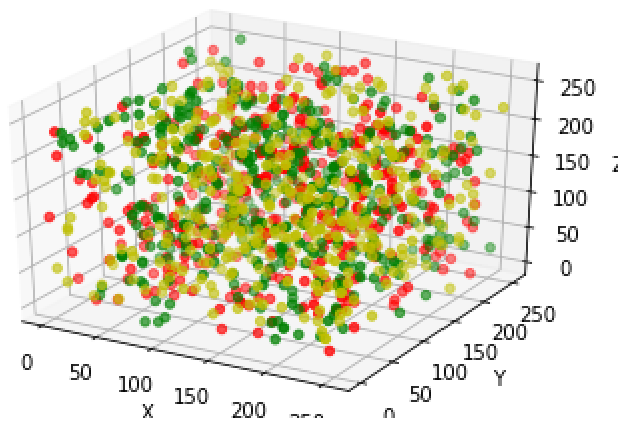
```
'''
    区分花(多分类):softmax回归
'''
from sklearn import datasets


import
import numpy as np
import pandas as pd
import tensorflow as tf
# 数据预处理
```

```
# import some data to play with
iris = datasets.load_iris()

data=pd.read_csv('./data/iris.data',names=['e_cd','e_kd','b_cd','b_kd','cat'])
# 独热编号
data['c1']=np.array(data['cat']=='Iris-setosa').astype(np.float32)
data['c2']=np.array(data['cat']=='Iris-versicolor').astype(np.float32)
data['c3']=np.array(data['cat']=='Iris-virginica').astype(np.float32)
del data['cat']

# 合并行
target=np.stack([data.c1.values,data.c2.values,data.c3.values]).T
shuju=np.stack([data.e_cd.values,data.e_kd.values,data.b_cd.values,data.b_kd.val
print(target.shape,shuju.shape)

# 定义网络

x=tf.placeholder("float",shape=[None,4])
y=tf.placeholder("float",shape=[None,3])

weight=tf.Variable(tf.truncated_normal([4,3]))
bias=tf.Variable(tf.truncated_normal([3]))
combine_input=tf.matmul(x,weight)+bias

pred=tf.nn.softmax(combine_input)

# 损失函数
loss=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=comb
# 正确率
correct_pred=tf.equal(tf.argmax(pred,1),tf.argmax(y,1))
accuracy=tf.reduce_mean(tf.cast(correct_pred,tf.float32))

# 梯度下降
train_step=tf.train.AdadeltaOptimizer(0.05).minimize(loss)

sess=tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(10000):
    index=np.random.permutation(len(target))
    shuju=shuju[index]
    target=target[index]
    sess.run(train_step,feed_dict={x:shuju,y:target})
    if i%1000==0:
        print(sess.run((loss,accuracy),feed_dict={x:shuju,y:target}))
```

使用决策树进行模型的计算

因为鸢尾花数据集很简单，特征已经全部提取好了，而且也很纯，所以就直接放到机器学习算法里面训练了。这里使用的是决策树分类算法。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

data = np.random.randint(0, 255, size=[40, 40, 40])

x, y, z = data[0], data[1], data[2]
ax = plt.subplot(111, projection='3d')  # 创建一个三维的绘图工程
#  将数据点分成三部分画，在颜色上有区分度
ax.scatter(x[:10], y[:10], z[:10], c='y')  # 绘制数据点
ax.scatter(x[10:20], y[10:20], z[10:20], c='r')
ax.scatter(x[30:40], y[30:40], z[30:40], c='g')

ax.set_zlabel('Z')  # 坐标轴
ax.set_ylabel('Y')
ax.set_xlabel('X')
plt.show()
```




```
# 首先对数据进行切分，即分出数据集和测试集
from sklearn.model_selection import train_test_split

all_inputs = df[['sepal_len', 'sepal_width',
                 'petal_len', 'petal_width']].values
all_classes = df['class'].values

(X_train,
 X_test,
 X_train,
 Y_test) = train_test_split(all_inputs, all_classes, train_size=0.8, random_stat


# 使用决策树算法进行训练
from sklearn.tree import DecisionTreeClassifier

# 定义一个决策树对象
decision_tree_classifier = DecisionTreeClassifier()

# 训练模型
model = decision_tree_classifier.fit(training_inputs, training_classes)

# 所得模型的准确性
print(decision_tree_classifier.score(testing_inputs, testing_classes))

# 使用训练的模型进行预测，为了偷懒，
# 直接把测试集里面的数据拿出来了三条
print(X_test[0:3])
print(Y_test[0:3])
model.predict(X_test[0:3])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:2
  FutureWarning)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last
<ipython-input-20-00a5a8d022b2> in <module>()
     18
     19 # 训练模型
---> 20 model = decision_tree_classifier.fit(training_inputs, training_cla
     21
     22 # 所得模型的准确性

NameError: name 'training_inputs' is not defined
```

SEARCH STACK OVERFLOW

```
print(__doc__)
```

```
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

# To getter a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```

二、MNIST手写字识别

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

tensorflow 入门

```
import tensorflow as tf
import numpy as np

# 使用 NumPy 生成假数据(phony data), 总共 100 个点.
x_data = np.float32(np.random.rand(2, 100)) # 随机输入
y_data = np.dot([0.100, 0.200], x_data) + 0.300

# 构造一个线性模型
#
b = tf.Variable(tf.zeros([1]))
W = tf.Variable(tf.random_uniform([1, 2], -1.0, 1.0))
y = tf.matmul(W, x_data) + b

# 最小化方差
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# 初始化变量
init = tf.initialize_all_variables()

# 启动图 (graph)
sess = tf.Session()
sess.run(init)

# 拟合平面
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# 得到最佳拟合结果 W: [[0.100  0.200]], b: [0.300]
```

初始化变量

将输入输入x_data 用随机输入的方式输入2行100列的二维数据。

y_data 则是通过线性模型进行矩阵的运算，具体的运算公式如下

$$y = w.*x + b; ==> y\_data = [0.100, 0.200].*x\_data + 0.300$$

y_data 的输出结果就是训练后要对比的结果，一般就是训练数据参考的label

注意

> 最终的到的y_data 结果是一个一维100列的数据，注意这个b=0.300代表的是参与到式子里是所有元素都为0.300的一个一行100列的数组。

```
import tensorflow as tf
import numpy as np

# 使用 NumPy 生成假数据(phony data), 总共 100 个点.
x_data = np.float32(np.random.rand(2, 100)) # 随机输入
y_data = np.dot([0.100, 0.200], x_data) + 0.300
```

```
import tensorflow as tf
x = tf.constant([[1., 1.], [2., 2.]])
y = tf.reduce_mean(x)  # 1.5
with tf.Session as sess:
  print(sess.run(tf.reduce_mean(x)))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last
<ipython-input-8-aaac89929b03> in <module>()
      2 x = tf.constant([[1., 1.], [2., 2.]])
      3 y = tf.reduce_mean(x)  # 1.5
----> 4 with tf.Session as sess:
      5   print(sess.run(tf.reduce_mean(x)))

AttributeError: __enter__
```

SEARCH STACK OVERFLOW

构建一个线性模型

$$y = w. * x + b$$

- 简单的逻辑回归模型

输入x是2行100列的数组;

```
# 构造一个线性模型
#
b = tf.Variable(tf.zeros([1]), name="biases") # 定义初值为0的一维向量
W = tf.Variable(tf.random_uniform([1, 2], -1.0, 1.0), name="weights")  # 均匀分布
y = tf.matmul(W, x_data) + b #
```

双击（或按回车键）即可修改

```
import numpy as np
x_data = np.random.rand(100).astype(np.float32)
print(type(x_data))
```

```
# 最小化方差
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```