HW5-520

1.

(1) use “np.random.uniform(0, 100, 1000)” to get uniform distribution from 0~100, return 1-d array. Then use “np.array.sort()” to sort as ascending order.

(2) create error parameter with “np.random.normal(0, 2, 1000)”, and y = x1*3 + x2*4 + error.

(3) use “np.concatenate((x1.reshape(1000,1), x2.reshape(1000,1)), axis=1)” to combine x1, x2 as 1000*2-dimensional array.

(4) Normal Equation is “minW = np.dot(XT * X).I * XT * Y ”.

(5) use “linear_model.LinearRegression()” to fit x, y.

```
coefficient from Normal Equation:  [3.01899861 3.97951202]
coefficient from sklearn linear model:  [3.01129536 3.98667401]
jiang@jiang-Inspiron-7557:~/Jupyter$
```

2.

(1) use “make_classification(n_samples=1000, n_features=2, n_informative=2,

n_redundant=0, n_classes=2)”

(2) use “train_test_split(X, Y, test_size=0.2)”

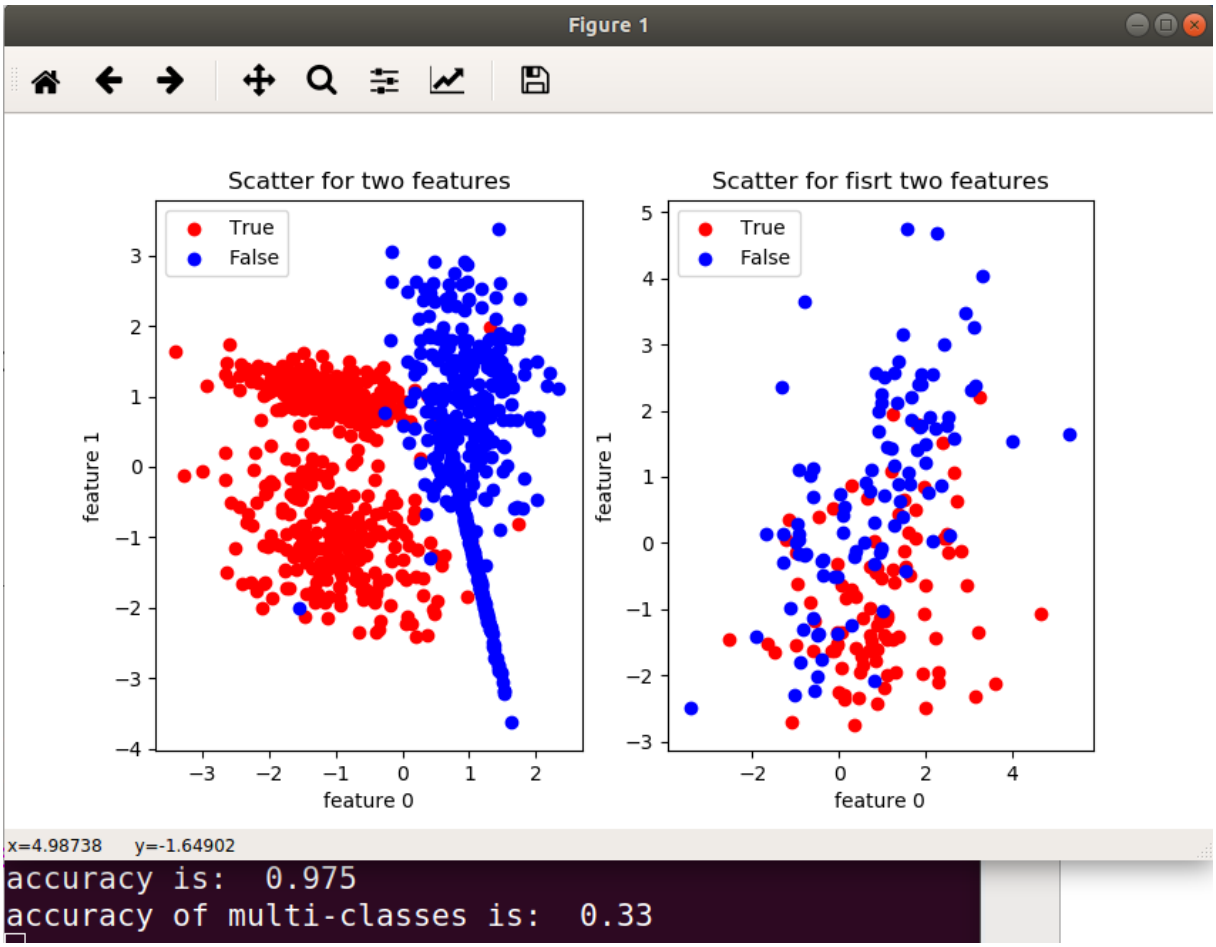
(3) use “LogisticRegression().fit(x_train, y_train)”

(4) write ‘for loop’ to count the same result between y_predict and y_test. Or just use “clf.score(x_test, y_test)”

(5) use “scatter()” function with the two features as x, y axis. But I can’t draw the division line, because of not understanding how “.contour()” works.

3.

As above, just change the arguments to “make_classification(n_samples=1000, n_features=5, n_informative=5, n_redundant=0, n_classes=10)”. The accuracy decreases significantly.



Appendix:

```
import numpy as np
from numpy.linalg import inv        # used to get inverse matrix
from sklearn import linear_model    # linear_model is regarded as a class
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

#1.1
x1 = np.random.uniform(0, 100, 1000)   # uniform distribution
x2 = np.random.uniform(0, 100, 1000)
x1.sort()
x2.sort()

#1.2
error = np.random.normal(0, 2, 1000)  # error belongs to gaussian distribution
y = x1*3 + x2*4 + error

#1.3
x = np.concatenate((x1.reshape(1000,1), x2.reshape(1000,1)), axis=1)

#1.4
#minW = np.dot(XT * X).I * XT * Y
xT = x.T
a = inv(np.dot(xT, x))
b = np.dot(a, xT)
minW = np.dot(b, y)
print("coefficient from Normal Equation: ", minW)

#1.5
regr = linear_model.LinearRegression()
regr.fit(x, y)
```

```
print("coefficient from sklearn linear model: ", regr.coef_)


#2.1
X, Y = make_classification(n_samples=1000, n_features=2, n_informative=2,
                           n_redundant=0, n_classes=2)   #binary output

#2.2
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

#2.3
clf = LogisticRegression()
clf.fit(x_train, y_train)

#2.4
y_predict = clf.predict(x_test)
count = 0
length = y_test.size
for y1, y2 in zip(y_predict, y_test):
    if y1==y2:
        count += 1
accuracy = count / length
print("accuracy is: ", accuracy)
print("accuracy is: ", clf.score(x_test, y_test))

#2.5
fig, ax = plt.subplots(1, 2)
```

```
ax[0].scatter(X[Y==0,0], X[Y==0,1], c='r', label='True')
ax[0].scatter(X[Y==1,0], X[Y==1,1], c='b', label='False')
ax[0].legend()
ax[0].set_title('Scatter for two features')
ax[0].set_xlabel('feature 0')
ax[0].set_ylabel('feature 1')


#3
X2, Y2 = make_classification(n_samples=1000, n_features=5, n_informative=5,
                             n_redundant=0, n_classes=10)
x_train2, x_test2, y_train2, y_test2 = train_test_split(X2, Y2, test_size=0.2)
clf2 = LogisticRegression()
clf2.fit(x_train2, y_train2)
print("accuracy of multi-classes is: ", clf2.score(x_test2, y_test2))

ax[1].scatter(X2[Y2==0,0], X2[Y2==0,1], c='r', label='True')
ax[1].scatter(X2[Y2==1,0], X2[Y2==1,1], c='b', label='False')
ax[1].legend()
ax[1].set_title('Scatter for fisrt two features')
ax[1].set_xlabel('feature 0')
ax[1].set_ylabel('feature 1')
plt.show()
```