```
%matplotlib inline
import numpy as np # linear algebra
import seaborn as sns
sns.set(style='whitegrid')
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import tensorflow as tf
allData = pd.read_csv('/srv/p2pRisk/spider-wdzj/datas/total.csv')
totalIndexNum = 17
dropIndex = ['wdzjPlatId', 'platName', 'isBroken']
featureNum = totalIndexNum - len(dropIndex)
X = allData.drop(labels=dropIndex, axis=1).values
y = allData.isBroken.values
'''
trainset: 80%
testset: 20%
set seed for numpy and tensorflow
set for reproducible results
'''
seed = 5
np.random.seed(seed)
tf.set_random_seed(seed)
# set replace=False, Avoid double sampling
train_index = np.random.choice(len(X), round(len(X) * 0.8), replace=False)
# diff set
test_index = np.array(list(set(range(len(X))) - set(train_index)))
train_X = X[train_index]
train_y = y[train_index]
test_X = X[test_index]
test_y = y[test_index]
# Define the normalized function
def min_max_normalized(data):
    col_max = np.max(data, axis=0)
    col_min = np.min(data, axis=0)
    return np.divide(data - col_min, col_max - col_min)

# Normalized processing, must be placed after the data set segmentation,
# otherwise the test set will be affected by the training set
train_X = min_max_normalized(train_X)
test_X = min_max_normalized(test_X)

# Normalized processing, must be placed after the data set segmentation,
# otherwise the test set will be affected by the training set
train_X = min_max_normalized(train_X)
test_X = min_max_normalized(test_X)
# Define placeholders
data = tf.placeholder(dtype=tf.float32, shape=[None, featureNum])
target = tf.placeholder(dtype=tf.float32, shape=[None, 1])
# Declare the model you need to learn
mod = tf.matmul(data, A) + b
# Declare loss function
# Use the sigmoid cross-entropy loss function,
```

```
# first doing a sigmoid on the model result and then using the cross-
entropy loss function
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=mod,
labels=target))
# Define the learning rate,  batch_size etc.
learning_rate = 0.001
batch_size = 30
iter_num = 6000

# Define the optimizer
opt = tf.train.GradientDescentOptimizer(learning_rate)
# opt = tf.train.MomentumOptimizer(learnin )
# Define the goal
goal = opt.minimize(loss)
# Define the accuracy
# The default threshold is 0.5, rounded off directly
prediction = tf.round(tf.sigmoid(mod))
# Bool into float32 type
correct = tf.cast(tf.equal(prediction, target), dtype=tf.float32)
# Average
accuracy = tf.reduce_mean(correct)
# End of the definition of the model framework
# Start training model
# Define the variable that stores the result
loss_trace = []
train_acc = []
test_acc = []
# training model
for epoch in range(iter_num):
    # Generate random batch index
    batch_index = np.random.choice(len(train_X), size=batch_size)
    batch_train_X = train_X[batch_index]
    batch_train_y = np.matrix(train_y[batch_index]).T
    sess.run(goal, feed_dict={data: batch_train_X, target: batch_train_y})
    temp_loss = sess.run(loss, feed_dict={data: batch_train_X, target:
batch_train_y})
    # convert into a matrix, and the shape of the placeholder to
correspond
    temp_train_acc = sess.run(accuracy, feed_dict={data: train_X, target:
np.matrix(train_y).T})
    temp_test_acc = sess.run(accuracy, feed_dict={data: test_X, target:
np.matrix(test_y).T})
    # recode the result
    loss_trace.append(temp_loss)
    train_acc.append(temp_train_acc)
    test_acc.append(temp_test_acc)
    # output
    if (epoch + 1) % 300 == 0:
        print('epoch: {:4d} loss: {:5f} train_acc: {:5f} test_acc:
{:5f}'.format(epoch + 1, temp_loss,

temp_train_acc, temp_test_acc))

# Visualization of the results
```

```
# loss function
plt.plot(loss_trace)
plt.title('Cross Entropy Loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()

# accuracy
plt.plot(train_acc, 'b-', label='train accuracy')
plt.plot(test_acc, 'k-', label='test accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('Train and Test Accuracy')
plt.legend(loc='best')
plt.show()

latestData = pd.read_csv('/srv/p2pRisk/spider-wdzj/datas/latest.csv')
latestData.shape

latestTestX = latestData.drop(labels=dropIndex, axis=1).values
latestTestY = latestData.isBroken.values
acc = sess.run(accuracy, feed_dict={data: latestTestX, target:
np.matrix(latestTestY).T})
```