Project 1 of 9880

Shirong Zhao

March 25, 2018

1 Introduction

Question 4, 5 and 9 are solved using Tensorflow. Please check the attached file.

2 Codes and Outcomes

2.1 Q4

In [1]:
```
# import desired packages
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [2]:
```
# import desired packages
import tensorflow as tf
```

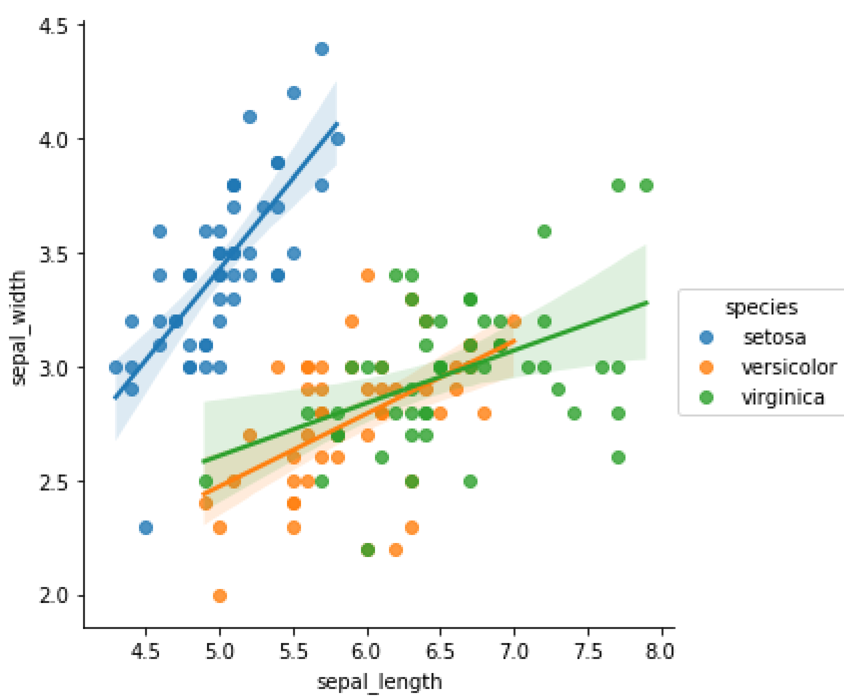
In [3]:
```
# Define the encoder function
def one_hot_encode(labels):
    n_labels=len(labels)
    n_unique_labels=len(np.unique(labels))
    one_hot_encode=np.zeros((n_labels, n_unique_labels))
    one_hot_encode[np.arange(n_labels), labels]=1
    return one_hot_encode
```

In [4]:
```
# Define the read dataset function
def read_dataset():
    iris = sns.load_dataset("iris")
    # plot the data
    g = sns.lmplot(x="sepal_length", y="sepal_width", hue="species",
                   truncate=True, size=5, data=iris)
    # split fertures and targets
    # get the data of virginica and versicolor
    X=iris.iloc[50:150,0:4].values
    y=iris.iloc[50:150,4]
    # Encode the dependent variable
    # Encode labels with value between 0 and n_classes-1.
    encoder=LabelEncoder()
    encoder.fit(y)
    encoder.classes_
    y=encoder.transform(y)
    Y=one_hot_encode(y)
    print(X.shape)
    return(X,Y)
```

In [5]:
```
# Prepare the data
def prepare_dataset():
    # prepare the data for splitting
    X, Y=read_dataset()
    # X1 and Y1 means virginica, X2 and Y2 means versicolor
    X1=X[0:50,]
    Y1=Y[0:50,]
    X2=X[50:100,]
    Y2=Y[50:100,]
    # shuffle the data for virginica
    # randome_state: the seed of the pseudo random number generator to use whe
n shuffling the data
    X1, Y1=shuffle(X1,Y1, random_state=1)
    # convert the dataset into train and test part
    train_x1, test_x1, train_y1, test_y1=train_test_split(X1,Y1,test_size=0.20
)
    # shuffle the data for versicolor
    # randome_state: the seed of the pseudo random number generator to use whe
n shuffling the data
    X2, Y2=shuffle(X2,Y2, random_state=2)
    # convert the dataset into train and test part
    train_x2, test_x2, train_y2, test_y2=train_test_split(X2,Y2,test_size=0.20
)
    # combine the data into train data and test data
    train_x=np.concatenate((train_x1, train_x2), axis=0)
    train_y=np.concatenate((train_y1, train_y2), axis=0)
    test_x=np.concatenate((test_x1, test_x2), axis=0)
    test_y=np.concatenate((test_y1, test_y2), axis=0)
    return(train_x, train_y, test_x, test_y)
```

In [6]:
```
# get the prepared data
train_x, train_y, test_x, test_y=prepare_dataset()
```

```
(100, 4)
```



In [7]:
```
# Define the important parameters and variables to work with the tensors
learning_rate=0.01
training_epochs=2000
cost_history=np.empty(shape=[1],dtype=float)
```

In [8]:
```
# placeholders and variables, input has 4 features and output has 2 classes
x=tf.placeholder(tf.float32, shape=[None,4])
y_=tf.placeholder(tf.float32, shape=[None,2])
# weight and bias
w=tf.Variable(tf.zeros([4,2]))
b=tf.Variable(tf.zeros([2]))
```

In [9]:
```
# model
# sigmoid function for two classes classfication
y=tf.nn.sigmoid(tf.matmul(x,w)+b)
# Define the cost function and optimizer
cost_function=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits
=y, labels=y_))
training_step=tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_f
unction)
```

In [10]:
```
# Initialize all the variables
init=tf.global_variables_initializer()
saver=tf.train.Saver()
```

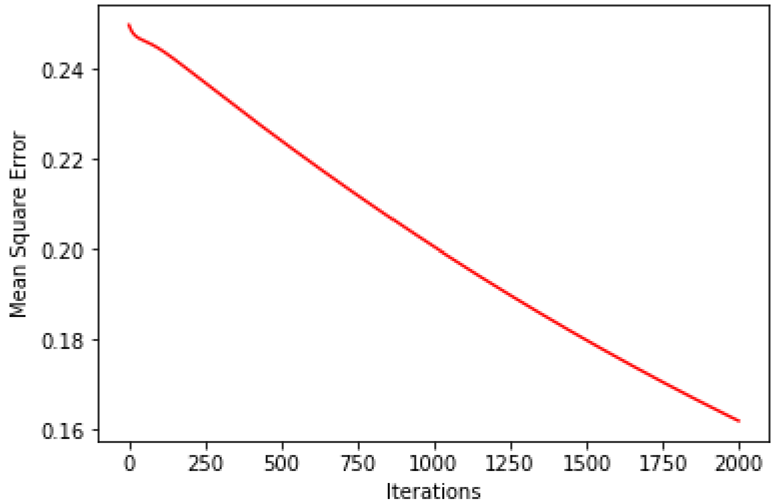
In [11]:
```
# define session as sess
sess=tf.Session()
sess.run(init)
```

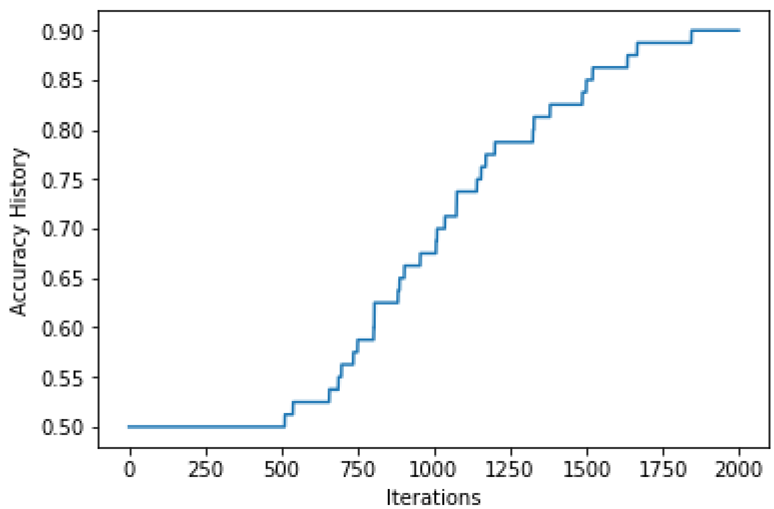
In [12]:
```
# Calculate the cost and accuracy for each epoch
mse_history=[]
accuracy_history=[]

for epoch in range(training_epochs):
    _,cost=sess.run([training_step,cost_function],feed_dict={x: train_x, y_: t
rain_y})
    cost_history=np.append(cost_history,cost)
    correct_prediction=tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    accuracy0=tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    pred_y=sess.run(y, feed_dict={x: test_x})
    mse=tf.reduce_mean(tf.square(pred_y-test_y))
    mse_=sess.run(mse)
    mse_history.append(mse_)
    accuracy=sess.run(accuracy0, feed_dict={x: train_x, y_: train_y})
    accuracy_history.append(accuracy)
    if epoch%100==0:
        print('epoch : ', epoch, '-cost: ', cost, "-MSE:", mse_,"-Train Accura
cy: ", accuracy)
```

```
epoch :  0 -cost:  0.6931472 -MSE: 0.24980074057679974 -Train Accuracy:  0.5
epoch :  100 -cost:  0.6854251 -MSE: 0.24450831095455722 -Train Accuracy:  0.
5
epoch :  200 -cost:  0.6810236 -MSE: 0.23957523281613477 -Train Accuracy:  0.
5
epoch :  300 -cost:  0.6766866 -MSE: 0.23432835838861896 -Train Accuracy:  0.
5
epoch :  400 -cost:  0.6723978 -MSE: 0.2291596711744604 -Train Accuracy:  0.5
epoch :  500 -cost:  0.66816056 -MSE: 0.22410207485671352 -Train Accuracy:
0.5
epoch :  600 -cost:  0.663978 -MSE: 0.21915905243495332 -Train Accuracy:  0.5
25
epoch :  700 -cost:  0.6598526 -MSE: 0.2143315898791661 -Train Accuracy:  0.5
625
epoch :  800 -cost:  0.65578663 -MSE: 0.20962024911508287 -Train Accuracy:
0.5875
epoch :  900 -cost:  0.6517819 -MSE: 0.2050251513670406 -Train Accuracy:  0.6
5
epoch :  1000 -cost:  0.64784 -MSE: 0.20054615454671376 -Train Accuracy:  0.6
75
epoch :  1100 -cost:  0.64396226 -MSE: 0.19618273195928332 -Train Accuracy:
0.7375
epoch :  1200 -cost:  0.6401496 -MSE: 0.19193412446262226 -Train Accuracy:
0.7875
epoch :  1300 -cost:  0.6364028 -MSE: 0.18779928921722724 -Train Accuracy:
0.7875
epoch :  1400 -cost:  0.6327223 -MSE: 0.18377694528956232 -Train Accuracy:
0.825
epoch :  1500 -cost:  0.62910855 -MSE: 0.1798656054321288 -Train Accuracy:
0.85
epoch :  1600 -cost:  0.62556154 -MSE: 0.1760636199098707 -Train Accuracy:
0.8625
epoch :  1700 -cost:  0.6220813 -MSE: 0.17236915828009058 -Train Accuracy:
0.8875
epoch :  1800 -cost:  0.6186675 -MSE: 0.1687802380356849 -Train Accuracy:  0.
8875
epoch :  1900 -cost:  0.6153199 -MSE: 0.16529475946616665 -Train Accuracy:
0.9
```

In [13]:
```
# plot mse and accuracy graph
plt.plot(mse_history,'r')
plt.xlabel('Iterations')
plt.ylabel('Mean Square Error')
plt.show()
plt.plot(accuracy_history)
plt.xlabel('Iterations')
plt.ylabel('Accuracy History')
plt.show()
```





In [14]:
```
# get the weights
sess.run(w)
```

Out[14]:
```
array([[ 0.37726048, -0.3772605 ],
       [ 0.26987386, -0.2698739 ],
       [-0.55869323,  0.55869323],
       [-0.4653222 ,  0.4653223 ]], dtype=float32)
```

In [15]:
```
# get the bias
sess.run(b)
```

Out[15]:
```
array([ 0.18476646, -0.18476644], dtype=float32)
```

In [16]:
```
# Print the final prediction accuracy
correct_prediction=tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy0=tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
accuracy=sess.run(accuracy0, feed_dict={x: train_x, y_: train_y})
print("Test Accuracy: ", accuracy)
```

```
Test Accuracy:  0.9
```

In [17]:
```
# Print the final mean square error
pred_y=sess.run(y, feed_dict={x: test_x})
mse=tf.reduce_mean(tf.square(pred_y-test_y))
print("MSE: %.4f" % sess.run(mse))
```

```
MSE: 0.1619
```

2.2 Q5

In [1]:
```
# import desired packages
import tensorflow as tf
import numpy as np
import collections
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
%matplotlib inline
learn = tf.contrib.learn
tf.logging.set_verbosity(tf.logging.ERROR)
```

In [2]:
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

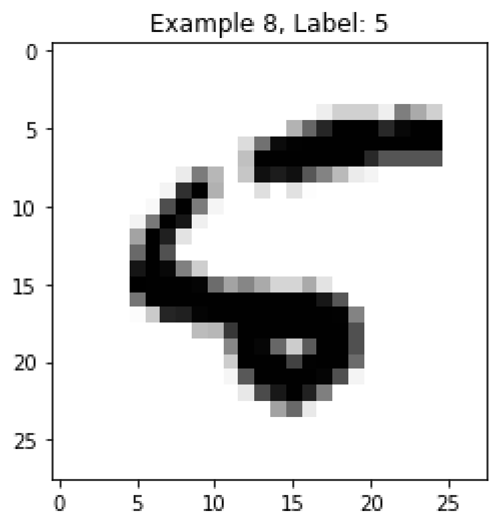
In [3]:
```
# get the train data and test data
train_data=mnist.train.images # Return np.array
train_labels=mnist.train.labels
test_data=mnist.test.images # Return np.array
test_labels=mnist.test.labels
```

In [4]:
```
# check the dimension of the data
print(train_data.shape)
print(train_labels.shape)
print(test_data.shape)
print(test_labels.shape)
```

```
(55000, 784)
(55000, 10)
(10000, 784)
(10000, 10)
```

In [5]:
```
# display some digits
def display(i):
    img=test_data[i]
    plt.title("Example %d, Label: %d" % (i, np.where(test_labels[i] == 1)[0]))
    plt.imshow(img.reshape((28,28)), cmap=plt.cm.gray_r)
```

In [6]:
```
display(8)
```



In [7]:
```
# Define the important parameters and variables to work with the tensors
learning_rate=0.01
training_epochs=2000
cost_history=np.empty(shape=[1],dtype=float)
```

In [8]:
```
# placeholders and variables, input has 784 features and output has 10 classes
x=tf.placeholder(tf.float32, shape=[None,784])
y_=tf.placeholder(tf.float32, shape=[None,10])
# weight and bias
w=tf.Variable(tf.zeros([784,10]))
b=tf.Variable(tf.zeros([10]))
```

In [9]:
```
# model
y=tf.matmul(x,w)+b
```

In [10]:
```
# loss function
cost_function=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_
, logits=y))
```

In [11]:
```
# optimiser
training_step=tf.train.GradientDescentOptimizer(0.01).minimize(cost_function)
```

In [12]:
```
# session parameters
sess=tf.InteractiveSession()
#initialising variables
init=tf.global_variables_initializer()
sess.run(init)
```

In [13]:
```
# Train
for _ in range(2000):
    batch_xs, batch_ys = mnist.train.next_batch(400)
    sess.run(training_step, feed_dict={x: batch_xs, y_: batch_ys})
```

In [14]:
```
# get the weights
sess.run(w)
```

Out[14]:
```
array([[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.]], dtype=float32)
```

In [15]:
```
# get the bias
sess.run(b)
```

Out[15]:
```
array([-0.06587118,  0.14213304, -0.033221  , -0.04591586,  0.04101602,
        0.11140792, -0.01584894,  0.0807186 , -0.18818985, -0.02622885],
      dtype=float32)
```

In [16]:
```
# Print the final prediction accuracy
correct_prediction=tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy0=tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
accuracy=sess.run(accuracy0, feed_dict={x: test_data, y_: test_labels})
print("Test Accuracy: ", accuracy)
```

```
Test Accuracy:  0.885
```

In [17]:
```
# Check whether we can separate digits 3 and 8
location=sess.run(tf.argmax(test_labels,1))
np_array=np.array(location)
index8=np.where(np_array==8)
index3=np.where(np_array==3)
# get the data for digits 8 and digits 3
test_data8=test_data[index8]
test_data3=test_data[index3]
test_labels8=test_labels[index8]
test_labels3=test_labels[index3]
```

In [18]:
```
f8=sess.run(tf.argmax(y,1), feed_dict={x: test_data8})
print(collections.Counter(f8))
f3=sess.run(tf.argmax(y,1), feed_dict={x: test_data3})
print(collections.Counter(f3))
```

```
Counter({8: 813, 3: 39, 5: 30, 9: 17, 6: 15, 7: 14, 2: 14, 4: 11, 1: 11, 0: 1
0})
Counter({3: 885, 5: 36, 8: 24, 2: 19, 7: 17, 9: 14, 6: 7, 0: 6, 1: 1, 4: 1})
```

In [19]:
```
# From the outcome, we can see that for each case, the mistakes are not so bi
g.
# Hence, our method could separate digits 3 and 8
```

2.3 Q9

Check Q4 and Q5, and also Tensorflow Installation on Palmetto.txt (Thanks to Yuanxun)

3 References

https://www.youtube.com/watch?v=RSKhj2BZQBg

https://www.youtube.com/watch?v=yX8KuPZCAMo

https://github.com/random-forests/tutorials/blob/master/ep7.ipynb

https://github.com/tensorflow/tensorflow/blob/r1.1/tensorflow/examples/tutorials/mnist/mnist_softmax.py

http://cs231n.github.io/neural-networks-case-study/

https://www.tensorflow.org/tutorials/layers

http://yann.lecun.com/exdb/mnist/