hw1_q1_q2

March 18, 2018

1 HW1 Q1. Linear Regression

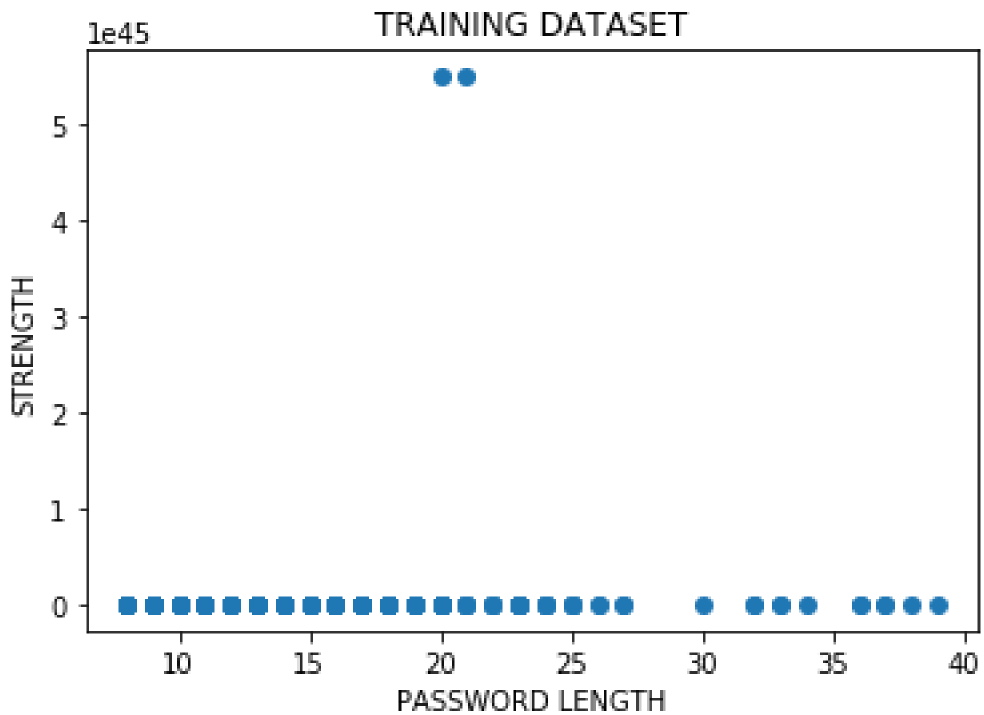
From the given password1.train file (the first column is the password and the second column is the strength), train your best linear regression model to predict the password strength from the password length.

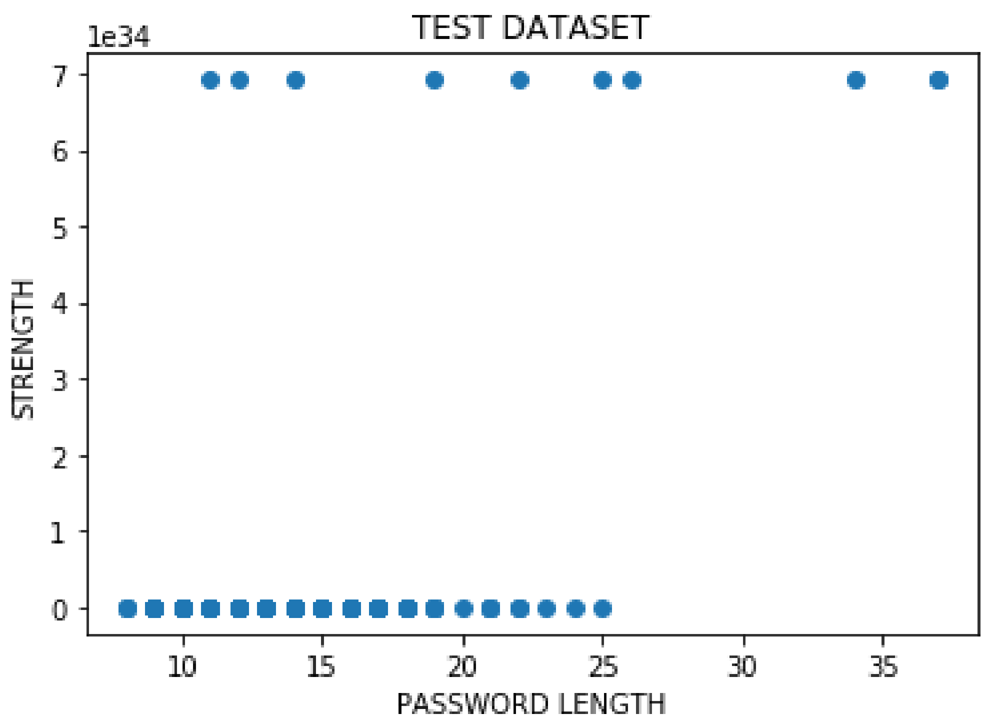
In [1]:
```
# import packages
import tensorflow as tf
import numpy as np
import os
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import matplotlib.pyplot as plt
```

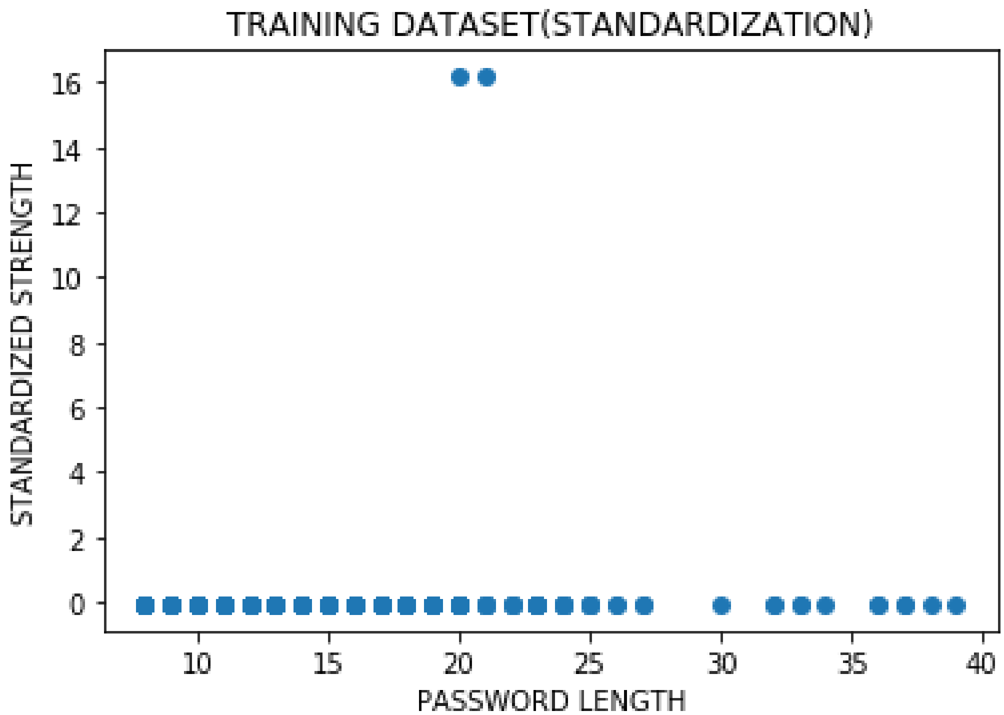
In [2]:
```
# load data
file_path = os.path.join('hw1data', 'password1.train')
load_data = pd.read_csv(file_path, sep='\t', header = None)
dataset = np.array(load_data)

new_dataset = []
for i in range(dataset.shape[0]):
    new_dataset.append([len(dataset[i][0]), dataset[i][-1]])
new_dataset = np.array(new_dataset)
```

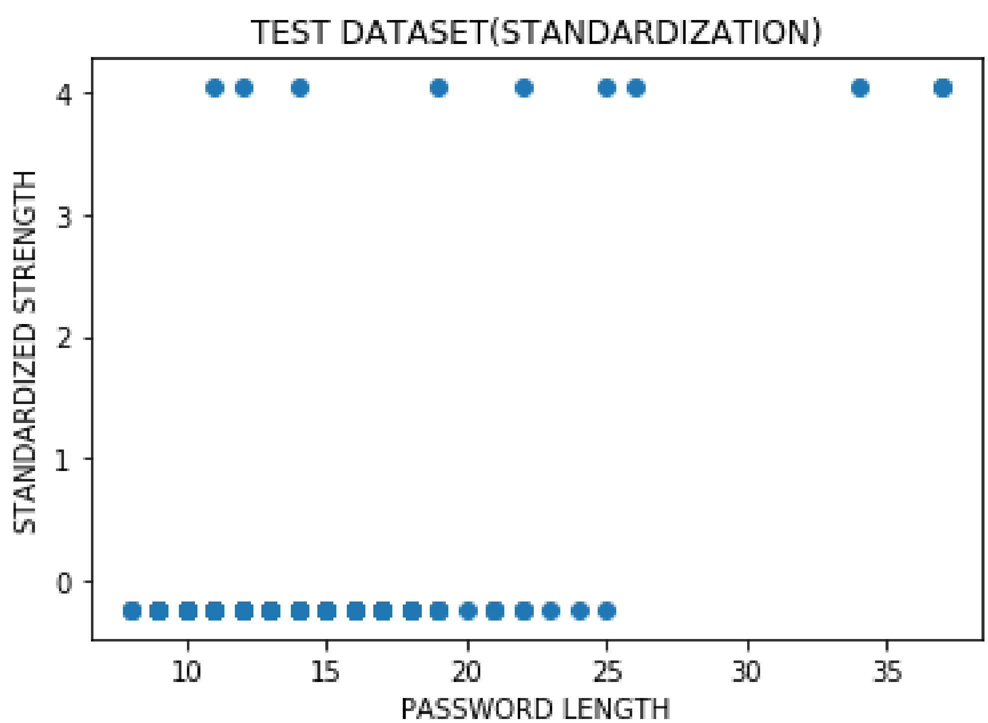
In [3]:
```
amount_train = int(0.75*dataset.shape[0])

sc = StandardScaler() # for data standardization

x_train = new_dataset[:amount_train,0]
y_train = new_dataset[:amount_train,-1]

x_train = x_train.reshape(x_train.shape[0], 1)
y_train = y_train.reshape(y_train.shape[0], 1)

x_test = new_dataset[amount_train+1:,0]
y_test = new_dataset[amount_train+1:,-1]
```

```python
x_test = x_test.reshape(x_test.shape[0], 1)
y_test = y_test.reshape(y_test.shape[0], 1)

plt.scatter(x_train[:], y_train[:])
plt.title('TRAINING DATASET')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('STRENGTH')
plt.show()

plt.scatter(x_test[:], y_test[:])
plt.title('TEST DATASET')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('STRENGTH')
plt.show()

y_train = sc.fit_transform(y_train)
y_test = sc.fit_transform(y_test)

plt.scatter(x_train[:], y_train[:])
plt.title('TRAINING DATASET(STANDARDIZATION)')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('STANDARDIZED STRENGTH')
plt.show()

plt.scatter(x_test[:], y_test[:])
plt.title('TEST DATASET(STANDARDIZATION)')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('STANDARDIZED STRENGTH')
plt.show()
```


TRAINING DATASET
1e45
STRENGTH
PASSWORD LENGTH


TEST DATASET
1e34
STRENGTH
PASSWORD LENGTH


TRAINING DATASET(STANDARDIZATION)
STANDARDIZED STRENGTH
PASSWORD LENGTH
16
14
12
10
8
6
4
2
0
10
15
20
25
30
35
40


TEST DATASET(STANDARDIZATION)
STANDARDIZED STRENGTH
PASSWORD LENGTH
4
3
2
1
0
10
15
20
25
30
35

1.0.1 (a) Print the last 10 values of {step, cost, W, b}

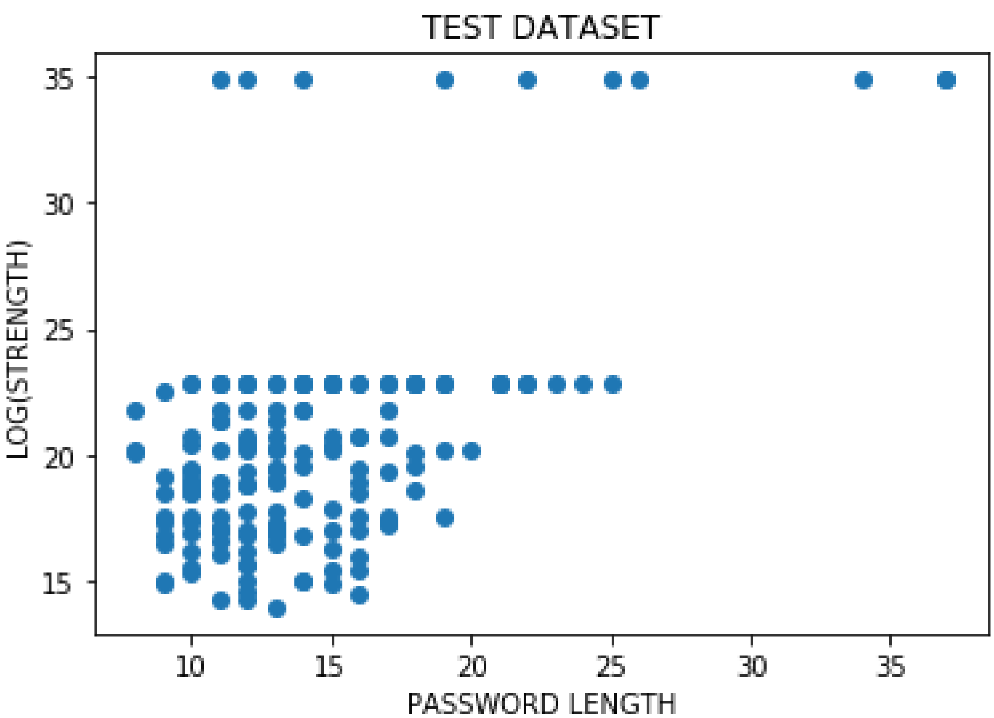
In [4]:
```
learning_rate = 10**(-3)
epochs = 50001

g1 = tf.Graph()
with g1.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 1])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([1]), name='weight')
    b = tf.Variable(tf.random_normal([1]), name='bias')

    hypothesis = tf.add(tf.multiply(X, W), b)

    cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(cost)

with tf.Session(graph=g1) as sess:
    print("step | cost | W | b")
    sess.run(tf.global_variables_initializer())

    for step in range(epochs):
        cost_val, W_val, b_val, _ = \
            sess.run([cost, W, b, train],
                     feed_dict={X: x_train, Y: y_train})
        if step > (epochs-11):
        #if step % 10000 == 0:
            print(step, cost_val, W_val, b_val)

    predictions = sess.run(hypothesis, feed_dict={X: x_test, Y: y_test})
    print("MSE: ", metrics.mean_squared_error(predictions, y_test))
```

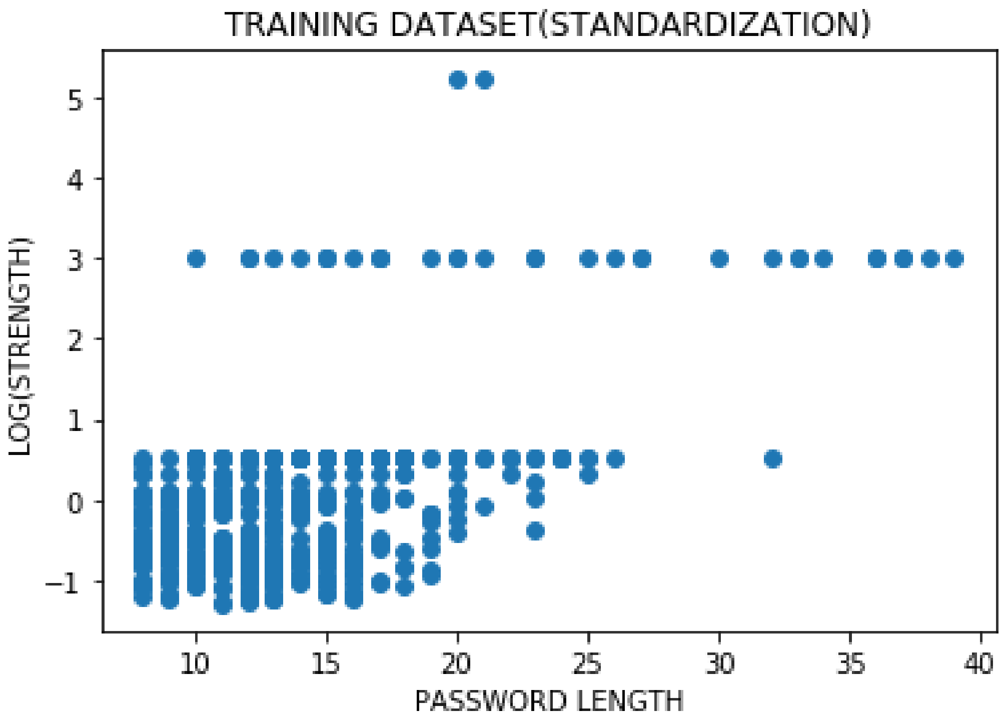
```
step | cost | W | b
49991 0.9935167 [0.01564561] [-0.21588336]
49992 0.9935167 [0.01564561] [-0.21588336]
49993 0.9935167 [0.01564561] [-0.21588336]
49994 0.9935167 [0.01564561] [-0.21588336]
49995 0.9935167 [0.01564561] [-0.21588336]
49996 0.9935167 [0.01564561] [-0.21588336]
49997 0.9935167 [0.01564561] [-0.21588336]
49998 0.9935167 [0.01564561] [-0.21588336]
49999 0.9935167 [0.01564561] [-0.21588336]
50000 0.9935167 [0.01564561] [-0.21588336]
MSE:  0.9328962448660587
```

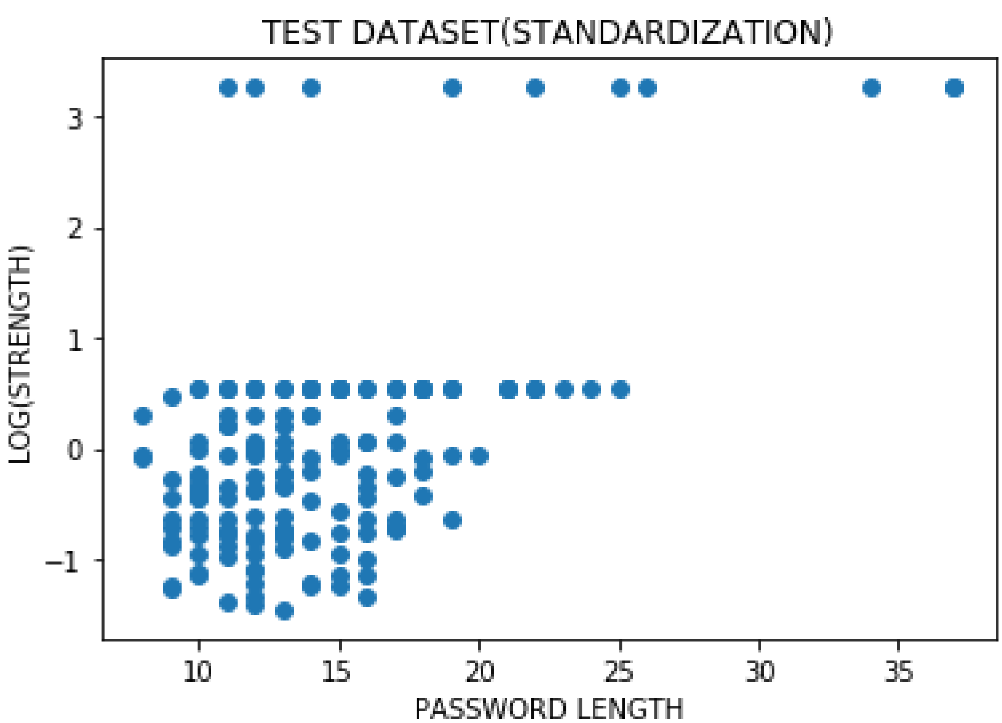
1.0.2 (b) Take a log10 for strengh values. And return the regression to predict log10(strength). Print the last 10 values of {step, cost, W, b}

In [5]:
```
x_train = new_dataset[:amount_train,0]
y_train = np.log10(new_dataset[:amount_train,-1])
plt.scatter(x_train, y_train)
plt.title('TRAINING DATASET')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('LOG(STRENGTH)')
plt.show()

x_test = new_dataset[amount_train+1:,0]
y_test = np.log10(new_dataset[amount_train+1:,-1])
plt.scatter(x_test, y_test)
plt.title('TEST DATASET')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('LOG(STRENGTH)')
plt.show()

x_train = x_train.reshape(x_train.shape[0], 1)
y_train = y_train.reshape(y_train.shape[0], 1)
y_train = sc.fit_transform(y_train)

x_test = x_test.reshape(x_test.shape[0], 1)
y_test = y_test.reshape(y_test.shape[0], 1)
y_test = sc.fit_transform(y_test)

plt.scatter(x_train, y_train)
plt.title('TRAINING DATASET(STANDARDIZATION)')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('LOG(STRENGTH)')
plt.show()

plt.scatter(x_test, y_test)
plt.title('TEST DATASET(STANDARDIZATION)')
plt.xlabel('PASSWORD LENGTH')
plt.ylabel('LOG(STRENGTH)')
plt.show()
```


TRAINING DATASET
LOG(STRENGTH)
PASSWORD LENGTH
45
40
35
30
25
20
15
10
15
20
25
30
35
40


TEST DATASET
LOG(STRENGTH)
PASSWORD LENGTH
35
30
25
20
15
10
15
20
25
30
35


TRAINING DATASET(STANDARDIZATION)
LOG(STRENGTH)
PASSWORD LENGTH
5
4
3
2
1
0
−1
10
15
20
25
30
35
40


TEST DATASET(STANDARDIZATION)
LOG(STRENGTH)
PASSWORD LENGTH
3
2
1
0
−1
10
15
20
25
30
35

```
In [6]: learning_rate = 10**(-4)
        epochs = 300001

        g2 = tf.Graph()
        with g2.as_default():
            X = tf.placeholder(tf.float32, shape = [None, 1])
            Y = tf.placeholder(tf.float32, shape = [None, 1])

            W = tf.Variable(tf.random_normal([1]), name='weight')
            b = tf.Variable(tf.random_normal([1]), name='bias')

            hypothesis = tf.add(tf.multiply(X, W), b)

            cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
            optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
            train = optimizer.minimize(cost)

            accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

        with tf.Session(graph=g2) as sess:
        #sess = tf.Session()
            print("step | cost | W | b")
            sess.run(tf.global_variables_initializer())

            for step in range(epochs):
                cost_val, W_val, b_val, _ = \
                    sess.run([cost, W, b, train],
                             feed_dict={X: x_train, Y: y_train})
                if step > (epochs-11):
                    print(step, cost_val, W_val, b_val)
            MSE_Len = sess.run(accuracy, feed_dict={X: x_test, Y: y_test})
            print("MSE: ", MSE_Len)
```

```
step | cost | W | b
299991 0.6859833 [0.10871579] [-1.4997728]
299992 0.6859833 [0.10871579] [-1.4997728]
299993 0.6859833 [0.10871579] [-1.4997728]
299994 0.6859833 [0.10871579] [-1.4997728]
299995 0.6859833 [0.10871579] [-1.4997728]
299996 0.6859833 [0.10871579] [-1.4997728]
299997 0.6859833 [0.10871579] [-1.4997728]
299998 0.6859833 [0.10871579] [-1.4997728]
299999 0.6859833 [0.10871579] [-1.4997728]
300000 0.6859833 [0.10871579] [-1.4997728]
MSE:  0.69554937
```

1.0.3 (c) Between (a) and (b), which is a better model or representation? Explain

As can be seen from the plots, data preprocessing, taking log10, made the gap between each Y values smaller, made the train easier.
When log10 is taken, it is possible that linear regression model is able to be applied even without Standardization.

2 Q2. Multivariable Linear Regression

From the given password1.train file, train your best linear regression model to predict the password strength from password length, number of digits, number of symbols, and the number of upper_class letter in a password.

2.0.1 (a) Print the last 10 values of {step, cost, W, b}

```
In [7]: def data_preprocess(dataset):
            out = []
            for i in range(dataset.shape[0]):
                s = str(dataset[i][0])
                symbols = set(['`','~','!','@','#','$','%','^','&','*','(',')','_','-','+','=','
                l = len(s)
                dg = 0
                sb = 0
                up = 0
                for c in s:
                    if c.isdigit():
                        dg = dg+1
                    elif c in symbols:
                        sb = sb+1
                    elif c.isupper():
                        up = up+1
                out.append([l, dg, sb, up, dataset[i][1]])
            return out

        q2_dataset = np.array(data_preprocess(dataset))

        amount_train = int(0.75*dataset.shape[0])

        x_train = q2_dataset[:amount_train,:4]
        y_train = q2_dataset[:amount_train,-1]

        x_test = q2_dataset[amount_train+1:,:4]
        y_test = q2_dataset[amount_train+1:,-1]

        y_train = y_train.reshape(y_train.shape[0], 1)
        y_train_sc = sc.fit_transform(y_train)
```

```
        y_test = y_test.reshape(y_test.shape[0], 1)
        y_test_sc = sc.fit_transform(y_test)
```

```
In [8]: learning_rate = 10**(-3)
        epochs = 20001

        g3 = tf.Graph()
        with g3.as_default():
            X = tf.placeholder(tf.float32, shape = [None, 4])
            Y = tf.placeholder(tf.float32, shape = [None, 1])

            W = tf.Variable(tf.random_normal([4, 1]), name='weight')
            b = tf.Variable(tf.random_normal([1]), name='bias')

            hypothesis = tf.add(tf.matmul(X, W), b)

            cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

            optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
            train = optimizer.minimize(cost)

        with tf.Session(graph=g3) as sess:
            sess.run(tf.global_variables_initializer())

            for step in range(epochs):
                cost_val, W_val, b_val, _ = \
                    sess.run([cost, W, b, train],
                             feed_dict={X: x_train, Y: y_train_sc})
                if step > (epochs-11):
                #if step % 10000 == 0:
                    print('step: ', step,' cost : ', cost_val, '\nW: ', np.array2string(W_val).r

            predictions = sess.run(hypothesis, feed_dict={X: x_test, Y: y_test_sc})
            print("MSE: ", metrics.mean_squared_error(predictions, y_test_sc))
```

```
step:  19991  cost :  0.9802611
W:  [[ 0.01809262] [-0.04685337] [-0.06644833] [ 0.00592055]]
b:  [-0.05209522]
-------------
step:  19992  cost :  0.9802612
W:  [[ 0.01809247] [-0.04685352] [-0.06644855] [ 0.00592015]]
b:  [-0.05209194]
-------------
step:  19993  cost :  0.9802611
W:  [[ 0.01809232] [-0.04685367] [-0.06644878] [ 0.00591975]]
b:  [-0.05208866]
-------------
step:  19994  cost :  0.9802611
```

```
W:  [[ 0.01809216] [-0.04685382] [-0.066449  ] [ 0.00591935]]
b:  [-0.05208538]
-------------
step:  19995  cost :  0.9802611
W:  [[ 0.01809201] [-0.04685397] [-0.06644922] [ 0.00591894]]
b:  [-0.0520821]
-------------
step:  19996  cost :  0.9802611
W:  [[ 0.01809186] [-0.04685412] [-0.06644945] [ 0.00591854]]
b:  [-0.05207882]
-------------
step:  19997  cost :  0.9802611
W:  [[ 0.01809171] [-0.04685427] [-0.06644967] [ 0.00591814]]
b:  [-0.05207554]
-------------
step:  19998  cost :  0.98026097
W:  [[ 0.01809156] [-0.04685442] [-0.0664499 ] [ 0.00591774]]
b:  [-0.05207226]
-------------
step:  19999  cost :  0.98026097
W:  [[ 0.0180914 ] [-0.04685457] [-0.06645012] [ 0.00591733]]
b:  [-0.05206898]
-------------
step:  20000  cost :  0.9802609
W:  [[ 0.01809125] [-0.04685472] [-0.06645034] [ 0.00591693]]
b:  [-0.0520657]
-------------
MSE:  0.9728988358812096
```

2.0.2 (b) Take a log10 for strength values. And rerun the regression to predict log10(strength). Print the last 10 values of {step, cost, W, b}

In [9]:
```
y_train = np.log10(y_train)
y_train_sc = sc.fit_transform(y_train)

y_test = np.log10(y_test)
y_test_sc = sc.fit_transform(y_test)
```

In [10]:
```
learning_rate = 10**(-4)
epochs = 390001

g4 = tf.Graph()
with g4.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 4])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([4, 1]), name='weight')
```

```
            b = tf.Variable(tf.random_normal([1]), name='bias')

            hypothesis = tf.add(tf.matmul(X, W), b)

            cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

            optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
            train = optimizer.minimize(cost)

        with tf.Session(graph=g4) as sess:
            sess.run(tf.global_variables_initializer())

            for step in range(epochs):
                cost_val, W_val, b_val, _ = \
                    sess.run([cost, W, b, train],
                             feed_dict={X: x_train, Y: y_train_sc})
                if step > (epochs-11):
                #if step % 10000 == 0:
                    print('step: ', step,' cost : ', cost_val, '\nW: ', np.array2string(W_val).

            predictions = sess.run(hypothesis, feed_dict={X: x_test, Y: y_test_sc})
            MSE_All = metrics.mean_squared_error(predictions, y_test_sc)
            print("MSE: ", MSE_All)
```

```
step:  389991  cost :  0.6171816
W:  [[0.1196439 ] [0.01756956] [0.14259863] [0.20154847]]
b:  [-1.9918396]
-------------
step:  389992  cost :  0.6171816
W:  [[0.1196439 ] [0.01756956] [0.14259864] [0.20154849]]
b:  [-1.9918398]
-------------
step:  389993  cost :  0.6171816
W:  [[0.11964391] [0.01756957] [0.14259864] [0.2015485 ]]
b:  [-1.9918399]
-------------
step:  389994  cost :  0.6171816
W:  [[0.11964392] [0.01756957] [0.14259864] [0.2015485 ]]
b:  [-1.99184]
-------------
step:  389995  cost :  0.61718154
W:  [[0.11964393] [0.01756958] [0.14259864] [0.20154852]]
b:  [-1.9918401]
-------------
step:  389996  cost :  0.61718154
W:  [[0.11964393] [0.01756958] [0.14259864] [0.20154853]]
b:  [-1.9918402]
-------------
```

```
step:  389997  cost :  0.6171816
W:  [[0.11964393] [0.01756959] [0.14259866] [0.20154855]]
b:  [-1.9918404]
-------------
step:  389998  cost :  0.61718154
W:  [[0.11964393] [0.01756959] [0.14259866] [0.20154856]]
b:  [-1.9918405]
-------------
step:  389999  cost :  0.61718154
W:  [[0.11964394] [0.0175696 ] [0.14259867] [0.20154858]]
b:  [-1.9918406]
-------------
step:  390000  cost :  0.6171815
W:  [[0.11964395] [0.01756961] [0.14259869] [0.20154859]]
b:  [-1.9918407]
-------------
MSE:  0.6574987962194435
```

2.0.3 (c) Choose the best features among length, number of digits, number of symbols and number of upper-class letter. And print the last 10 values of {step, cost, W, b}

We saw the result of length model at Q1(b). So, I will build new models with the information of digits, symbols and capitals and compare all the results to Q2(b). And then, I am going to figure out which feature is the most related to the password strength.

A model with the number of digits (digit model)

```
In [11]: # with digits.
         x_train_dg = x_train[:,1]
         x_train_dg = x_train_dg.reshape(x_train_dg.shape[0], 1)

         x_test_dg = x_test[:,1]
         x_test_dg = x_test_dg.reshape(x_test_dg.shape[0], 1)
```

```
In [12]: learning_rate = 10**(-4)
         epochs = 140000

         g5 = tf.Graph()
         with g5.as_default():
             X = tf.placeholder(tf.float32, shape = [None, 1])
             Y = tf.placeholder(tf.float32, shape = [None, 1])

             W = tf.Variable(tf.random_normal([1]), name='weight')
             b = tf.Variable(tf.random_normal([1]), name='bias')

             hypothesis = tf.add(tf.multiply(X, W), b)
```

```
        cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
        optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
        train = optimizer.minimize(cost)

        accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

    with tf.Session(graph=g5) as sess:
        print("step | cost | W | b")
        sess.run(tf.global_variables_initializer())

        for step in range(epochs):
            cost_val, W_val, b_val, _ = \
                sess.run([cost, W, b, train],
                         feed_dict={X: x_train_dg, Y: y_train_sc})
            if step > (epochs-11):
            #if step % 10000 == 0:
                print(step, cost_val, W_val, b_val)

        MSE_Digit = sess.run(accuracy, feed_dict={X: x_test_dg, Y: y_test_sc})
        print("MSE: ", MSE_Digit)
```

```
step | cost | W | b
139990 0.99475586 [0.036235] [-0.09886631]
139991 0.994756 [0.036235] [-0.09886631]
139992 0.99475586 [0.036235] [-0.09886632]
139993 0.994756 [0.036235] [-0.09886633]
139994 0.994756 [0.03623501] [-0.09886634]
139995 0.994756 [0.03623501] [-0.09886634]
139996 0.994756 [0.03623501] [-0.09886635]
139997 0.994756 [0.03623501] [-0.09886636]
139998 0.994756 [0.03623501] [-0.09886637]
139999 0.994756 [0.03623502] [-0.09886637]
MSE:  0.9945654
```

A model with the number of symbols (symbol model)

```
In [13]: # with symbols
         x_train_sb = x_train[:,2]
         x_train_sb = x_train_sb.reshape(x_train_sb.shape[0], 1)

         x_test_sb = x_test[:,2]
         x_test_sb = x_test_sb.reshape(x_test_sb.shape[0], 1)
```

```
In [14]: learning_rate = 10**(-4)
         epochs = 120000

         g6 = tf.Graph()
```

```
with g6.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 1])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([1]), name='weight')
    b = tf.Variable(tf.random_normal([1]), name='bias')

    hypothesis = tf.add(tf.multiply(X, W), b)

    cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(cost)

    accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

with tf.Session(graph=g6) as sess:
    print("step | cost | W | b")
    sess.run(tf.global_variables_initializer())

    for step in range(epochs):
        cost_val, W_val, b_val, _ = \
            sess.run([cost, W, b, train],
                     feed_dict={X: x_train_sb, Y: y_train_sc})
        if step > (epochs-11):
        #if step % 10000 == 0:
            print(step, cost_val, W_val, b_val)
    MSE_Sym = sess.run(accuracy, feed_dict={X: x_test_sb, Y: y_test_sc})
    print("MSE: ", MSE_Sym)
```

```
step | cost | W | b
119990 0.9643215 [0.17278014] [-0.19552563]
119991 0.9643215 [0.17278014] [-0.19552563]
119992 0.9643215 [0.17278014] [-0.19552563]
119993 0.9643215 [0.17278014] [-0.19552563]
119994 0.9643215 [0.17278014] [-0.19552563]
119995 0.9643215 [0.17278014] [-0.19552563]
119996 0.9643215 [0.17278014] [-0.19552563]
119997 0.9643215 [0.17278014] [-0.19552563]
119998 0.9643215 [0.17278014] [-0.19552563]
119999 0.9643215 [0.17278014] [-0.19552563]
MSE:  0.95120066
```

A model with the number of capitals (capital model)

In [15]:
```
# with upper-class letters
x_train_up = x_train[:,3]
x_train_up = x_train_up.reshape(x_train_up.shape[0], 1)
```

```
x_test_up = x_test[:,3]
x_test_up = x_test_up.reshape(x_test_up.shape[0], 1)
```

In [16]:
```
learning_rate = 11**(-4)
epochs = 180000

g7 = tf.Graph()
with g7.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 1])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([1]), name='weight')
    b = tf.Variable(tf.random_normal([1]), name='bias')

    hypothesis = tf.add(tf.multiply(X, W), b)

    cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(cost)

    accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

with tf.Session(graph=g7) as sess:
    print("step | cost | W | b")
    sess.run(tf.global_variables_initializer())

    for step in range(epochs):
        cost_val, W_val, b_val, _ = \
            sess.run([cost, W, b, train],
                     feed_dict={X: x_train_up, Y: y_train_sc})
        if step > (epochs-11):
        #if step % 10000 == 0:
            print(step, cost_val, W_val, b_val)
    MSE_Cap = sess.run(accuracy, feed_dict={X: x_test_up, Y: y_test_sc})
    print("MSE: ", MSE_Cap)
```

```
step | cost | W | b
179990 0.9993657 [0.0257479] [-0.01687783]
179991 0.9993657 [0.0257479] [-0.01687783]
179992 0.9993657 [0.0257479] [-0.01687783]
179993 0.9993657 [0.0257479] [-0.01687783]
179994 0.9993657 [0.0257479] [-0.01687783]
179995 0.9993657 [0.0257479] [-0.01687783]
179996 0.9993657 [0.0257479] [-0.01687783]
179997 0.9993657 [0.0257479] [-0.01687783]
179998 0.9993657 [0.0257479] [-0.01687783]
179999 0.9993657 [0.0257479] [-0.01687783]
```

```
MSE:  1.0021528
```

A model with password length and the number of symbols

```
In [17]: x_train_LS = np.array([[row[0], row[2]] for row in x_train])
         x_test_LS = np.array([[row[0], row[2]] for row in x_test])
```

```
In [18]: learning_rate = 10**(-4)
         epochs = 280000

         g8 = tf.Graph()
         with g8.as_default():
             X = tf.placeholder(tf.float32, shape = [None, 2])
             Y = tf.placeholder(tf.float32, shape = [None, 1])

             W = tf.Variable(tf.random_normal([2, 1]), name='weight')
             b = tf.Variable(tf.random_normal([1]), name='bias')

             hypothesis = tf.add(tf.matmul(X, W), b)

             cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
             optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
             train = optimizer.minimize(cost)

             accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

         with tf.Session(graph=g8) as sess:
             sess.run(tf.global_variables_initializer())

             for step in range(epochs):
                 cost_val, W_val, b_val, _ = \
                     sess.run([cost, W, b, train],
                              feed_dict={X: x_train_LS, Y: y_train_sc})
                 if step > (epochs-11):
                 #if step % 10000 == 0:
                     print('step: ', step,' cost : ', cost_val, ' W: ', np.array2string(W_val).r
             MSE_LenSym = sess.run(accuracy, feed_dict={X: x_test_LS, Y: y_test_sc})
             print("MSE: ", MSE_LenSym)
```

```
step:  279990  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279991  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279992  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279993  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279994  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279995  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279996  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279997  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
```

```
step:  279998  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
step:  279999  cost :  0.65485585  W:  [[0.10795229] [0.16106606]]  b:  [-1.6714728]
MSE:  0.6684651
```

A model with password length and the number of digits

```
In [19]: x_train_LD = np.array([[row[0], row[1]] for row in x_train])
         x_test_LD = np.array([[row[0], row[1]] for row in x_test])
```

```
In [20]: learning_rate = 10**(-3)
         epochs = 40001

         g9 = tf.Graph()
         with g9.as_default():
             X = tf.placeholder(tf.float32, shape = [None, 2])
             Y = tf.placeholder(tf.float32, shape = [None, 1])

             W = tf.Variable(tf.random_normal([2, 1]), name='weight')
             b = tf.Variable(tf.random_normal([1]), name='bias')

             hypothesis = tf.add(tf.matmul(X, W), b)

             cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
             optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
             train = optimizer.minimize(cost)

             accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

         with tf.Session(graph=g9) as sess:
             sess.run(tf.global_variables_initializer())

             for step in range(epochs):
                 cost_val, W_val, b_val, _ = \
                     sess.run([cost, W, b, train],
                              feed_dict={X: x_train_LD, Y: y_train_sc})
                 if step > (epochs-11):
                 #if step % 10000 == 0:
                     print('step: ', step,' cost : ', cost_val, ' W: ', np.array2string(W_val).r
             MSE_LenDig = sess.run(accuracy, feed_dict={X: x_test_LD, Y: y_test_sc})
             print("MSE: ", MSE_LenDig)
```

```
step:  39991  cost :  0.6840477  W:  [[0.10841315] [0.02202509]]  b:  [-1.555967]
step:  39992  cost :  0.68404776  W:  [[0.10841316] [0.0220251 ]]  b:  [-1.5559671]
step:  39993  cost :  0.6840477  W:  [[0.10841317] [0.02202511]]  b:  [-1.5559672]
step:  39994  cost :  0.6840477  W:  [[0.10841317] [0.02202511]]  b:  [-1.5559673]
step:  39995  cost :  0.68404776  W:  [[0.10841318] [0.02202512]]  b:  [-1.5559675]
step:  39996  cost :  0.6840477  W:  [[0.10841319] [0.02202513]]  b:  [-1.5559676]
```

```
step:  39997  cost :  0.68404776  W:  [[0.10841319] [0.02202514]]  b:  [-1.5559677]
step:  39998  cost :  0.68404776  W:  [[0.1084132 ] [0.02202514]]  b:  [-1.5559678]
step:  39999  cost :  0.6840478  W:  [[0.1084132 ] [0.02202515]]  b:  [-1.5559679]
step:  40000  cost :  0.6840478  W:  [[0.10841321] [0.02202516]]  b:  [-1.555968]
MSE:  0.69667774
```

A model with password length and the number of caplitals

In [21]:
```python
x_train_LC = np.array([[row[0], row[3]] for row in x_train])
x_test_LC = np.array([[row[0], row[3]] for row in x_test])
```

In [22]:
```python
learning_rate = 10**(-3)
epochs = 50001

g10 = tf.Graph()
with g10.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 2])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([2, 1]), name='weight')
    b = tf.Variable(tf.random_normal([1]), name='bias')

    hypothesis = tf.add(tf.matmul(X, W), b)

    cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(cost)

    accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

with tf.Session(graph=g10) as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(epochs):
        cost_val, W_val, b_val, _ = \
            sess.run([cost, W, b, train],
                     feed_dict={X: x_train_LC, Y: y_train_sc})
        if step > (epochs-11):
        #if step % 10000 == 0:
            print('step: ', step,' cost : ', cost_val, ' W: ', np.array2string(W_val).r
    MSE_LenCap = sess.run(accuracy, feed_dict={X: x_test_LC, Y: y_test_sc})
    print("MSE: ", MSE_LenCap)
```

```
step:  49991  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49992  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49993  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49994  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
```

```
step:  49995  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49996  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49997  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49998  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  49999  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
step:  50000  cost :  0.6413826  W:  [[0.1223746 ] [0.22720578]]  b:  [-1.8373805]
MSE:  0.6793331
```

A model with digits and symbols

In [23]:
```
x_train_DS = np.array([[row[1], row[2]] for row in x_train])
x_test_DS = np.array([[row[1], row[2]] for row in x_test])
```

In [24]:
```
learning_rate = 10**(-3)
epochs = 30001

g11 = tf.Graph()
with g11.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 2])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([2, 1]), name='weight')
    b = tf.Variable(tf.random_normal([1]), name='bias')

    hypothesis = tf.add(tf.matmul(X, W), b)

    cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(cost)

    accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

with tf.Session(graph=g11) as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(epochs):
        cost_val, W_val, b_val, _ = \
            sess.run([cost, W, b, train],
                     feed_dict={X: x_train_DS, Y: y_train_sc})
        if step > (epochs-11):
        #if step % 10000 == 0:
            print('step: ', step,' cost : ', cost_val, ' W: ', np.array2string(W_val).r
    MSE_DigSym = sess.run(accuracy, feed_dict={X: x_test_DS, Y: y_test_sc})
    print("MSE: ", MSE_DigSym)
```

```
step:  29991  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29992  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
```

```
step:  29993  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29994  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29995  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29996  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29997  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29998  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  29999  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
step:  30000  cost :  0.9565019  W:  [[0.0444297 ] [0.17944928]]  b:  [-0.32429853]
MSE:  0.9430415
```

A model with symbols and capitals

```
In [25]: x_train_SC = np.array([[row[2], row[3]] for row in x_train])
         x_test_SC = np.array([[row[2], row[3]] for row in x_test])
```

```
In [26]: learning_rate = 10**(-3)
         epochs = 20001

         g12 = tf.Graph()
         with g12.as_default():
             X = tf.placeholder(tf.float32, shape = [None, 2])
             Y = tf.placeholder(tf.float32, shape = [None, 1])

             W = tf.Variable(tf.random_normal([2, 1]), name='weight')
             b = tf.Variable(tf.random_normal([1]), name='bias')

             hypothesis = tf.add(tf.matmul(X, W), b)

             cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
             optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
             train = optimizer.minimize(cost)

             accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

         with tf.Session(graph=g12) as sess:
             sess.run(tf.global_variables_initializer())

             for step in range(epochs):
                 cost_val, W_val, b_val, _ = \
                     sess.run([cost, W, b, train],
                               feed_dict={X: x_train_SC, Y: y_train_sc})
                 if step > (epochs-11):
                 #if step % 10000 == 0:
                     print('step: ', step,' cost : ', cost_val, ' W: ', np.array2string(W_val).r
             MSE_SymCap = sess.run(accuracy, feed_dict={X: x_test_SC, Y: y_test_sc})
             print("MSE: ", MSE_SymCap)
```

```
step:  19991  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19992  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19993  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19994  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19995  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19996  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19997  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19998  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  19999  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
step:  20000  cost :  0.96430206  W:  [[0.172259  ] [0.00455622]]  b:  [-0.19788046]
MSE:  0.95165807
```

A model without password length.

In [27]:
```
x_train_DSC = np.array([[row[1], row[2], row[3]] for row in x_train])
x_test_DSC = np.array([[row[1], row[2], row[3]] for row in x_test])
```

In [28]:
```
learning_rate = 10**(-3)
epochs = 20001

g13 = tf.Graph()
with g13.as_default():
    X = tf.placeholder(tf.float32, shape = [None, 3])
    Y = tf.placeholder(tf.float32, shape = [None, 1])

    W = tf.Variable(tf.random_normal([3, 1]), name='weight')
    b = tf.Variable(tf.random_normal([1]), name='bias')

    hypothesis = tf.add(tf.matmul(X, W), b)

    cost = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
    train = optimizer.minimize(cost)

    accuracy = tf.reduce_mean(tf.square(tf.subtract(hypothesis, Y)))

with tf.Session(graph=g13) as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(epochs):
        cost_val, W_val, b_val, _ = \
            sess.run([cost, W, b, train],
                     feed_dict={X: x_train_DSC, Y: y_train_sc})
        if step > (epochs-11):
        #if step % 1000 == 0:
            print('step: ', step,' cost : ', cost_val, ' W: ', np.array2string(W_val).r
    MSE_DigSymCap = sess.run(accuracy, feed_dict={X: x_test_DSC, Y: y_test_sc})
    print("MSE: ", MSE_DigSymCap)
```

```
step:  19991  cost :  0.95647913  W:  [[ 0.04463352] [ 0.17989875] [-0.00496396]]  b:  [-0.32205
step:  19992  cost :  0.95647913  W:  [[ 0.04463354] [ 0.17989878] [-0.00496395]]  b:  [-0.32205
step:  19993  cost :  0.95647913  W:  [[ 0.04463357] [ 0.17989883] [-0.00496393]]  b:  [-0.32205
step:  19994  cost :  0.95647913  W:  [[ 0.04463359] [ 0.17989886] [-0.00496392]]  b:  [-0.32205
step:  19995  cost :  0.95647913  W:  [[ 0.04463362] [ 0.17989889] [-0.0049639 ]]  b:  [-0.32205
step:  19996  cost :  0.9564792  W:  [[ 0.04463364] [ 0.17989893] [-0.00496389]]  b:  [-0.322050
step:  19997  cost :  0.9564792  W:  [[ 0.04463366] [ 0.17989896] [-0.00496387]]  b:  [-0.322051
step:  19998  cost :  0.95647925  W:  [[ 0.04463369] [ 0.17989899] [-0.00496386]]  b:  [-0.32205
step:  19999  cost :  0.9564792  W:  [[ 0.04463371] [ 0.17989902] [-0.00496384]]  b:  [-0.322051
step:  20000  cost :  0.95647925  W:  [[ 0.04463374] [ 0.17989907] [-0.00496383]]  b:  [-0.32205
MSE:  0.9425431
```

In [29]:
```
print('        < Mean Square Error >\n')
print('Four features : ', "%22.10f" %(MSE_All))
print('Digit, Symbol & Capital : ', "%0.10f" %(MSE_DigSymCap))
print('Length : ', "%29.10f" %(MSE_Len))
print('Length & Capital : ', "%19.10f" %(MSE_LenCap))
print('Length & Digit : ', "%21.10f" %(MSE_LenDig))
print('Length & Symbol : ', "%20.10f" %(MSE_LenSym))
print('Capital : ', "%28.10f" %(MSE_Cap))
print('Digit : ', "%30.10f" %(MSE_Digit))
print('Digit & Symbol : ', "%21.10f" %(MSE_DigSym))
print('Symbol : ', "%29.10f" %(MSE_Sym))
print('Symbol & Capital : ', "%19.10f" %(MSE_SymCap))
```

```
        < Mean Square Error >

Four features :             0.6574987962
Digit, Symbol & Capital :   0.9425430894
Length :                    0.6955493689
Length & Capital :          0.6793330908
Length & Digit :            0.6966777444
Length & Symbol :           0.6684650779
Capital :                   1.0021528006
Digit :                     0.9945654273
Digit & Symbol :            0.9430415034
Symbol :                    0.9512006640
Symbol & Capital :          0.9516580701
```

The model without only password length as a feature has lower MSE than four features model. MSE of the models with password length is close to four features model.
Therefore, the information of password length is the stongest feature.

In [ ]: