Assignment 1:
k Nearest Neighbours (k-NN)
Solution

February 6, 2018

1 Euclidean distance function

Pairwise distance:
One possible implementation

```
def distanceFunc(X, Y):

    # Inputs
    # X: is an N1xD matrix (N1 observations and D dimensions)
    # Y: is an N2xD matrix (N2 observations and D dimensions)
    #outputs
    # pair_dist: is the pairwise distance matrix (N1xN2)

    # expand dimensions of inputs to enable broadcasting
    X1 = tf.expand_dims(X, -1)
    Y1 = tf.expand_dims(tf.transpose(Y), 0)

    # calculating pairwise distance
    pair_dist = tf.reduce_sum((X1 - Y1)**2, 1)
    return pair_dist
```

2 Making Predictions for Regression

1. **Choosing the nearest neighbours:**
   One possible implementation is

```
def pickKNearest(pair_dist, k):
    # Inputs
    # pair_dist: is a N1xN2 matrix representing the pairwise distance
    # K: is a scalar int representing number of neighbors

    # finding the nearest K neighbors
    top_k, ind_k = tf.nn.top_k(-pair_dist, k, sorted=True)

    n_train = tf.shape(pair_dist)[1]  # Number of training observations

    R = tf.reduce_sum(tf.to_float(tf.equal(tf.expand_dims(ind_k,2),\
                               tf.reshape(tf.range(n_train), [1,1,-1]))),1)

    return R/tf.to_float(k)
```

2. **Prediction:**

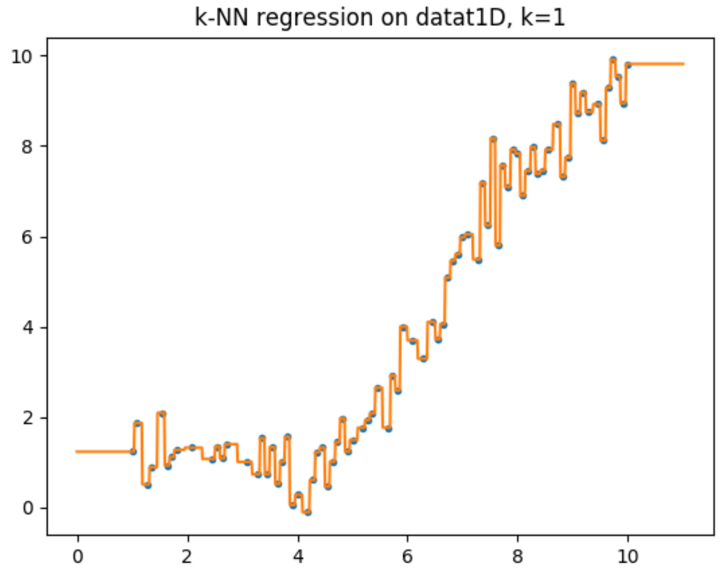
One possible implementation

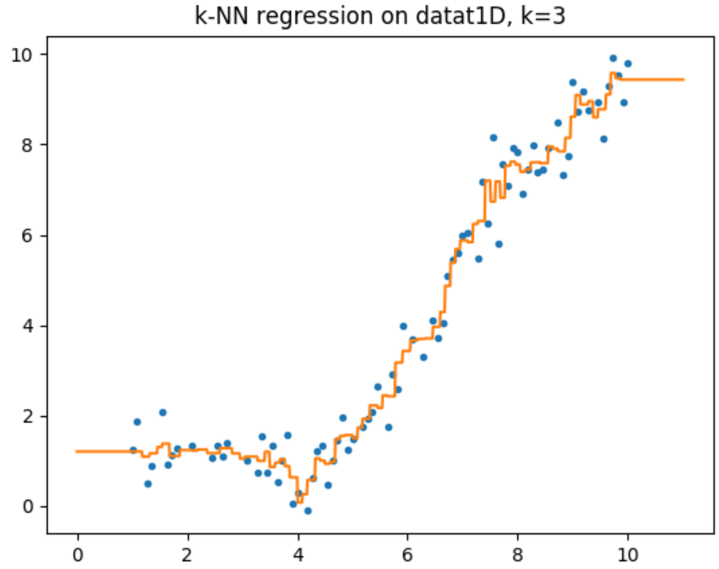
```
def predKNN(resp, train_target):
    # Predicting output
    return tf.matmul(resp, train_target)

## begin building graph
trainX = tf.placeholder(tf.float32, [None, 1], name='input_x')
trainY = tf.placeholder(tf.float32, [None, 1], name='input_y')

newX = tf.placeholder(tf.float32, [None, 1], name='new_x')
newY = tf.placeholder(tf.float32, [None, 1], name='new_y')

K = tf.placeholder("int32", name='K')

predY = predKNN(pickKNearest(distanceFunc(newX, trainX), K), trainY)
MSE = tf.reduce_mean(tf.reduce_sum((predY - newY)**2, 1))
sess = tf.InteractiveSession()
## end building graph

X = np.linspace(0.0,11.0,num=1000)[:,np.newaxis] # for plotting import matplotlib

# Find the nearest k neighbours:
kvec = [1,3,5,50]
mse_train_list = []
mse_valid_list = []
mse_test_list = []

trainData, trainTarget, validData, validTarget, testData, testTarget \
                                                                = generateData()
```

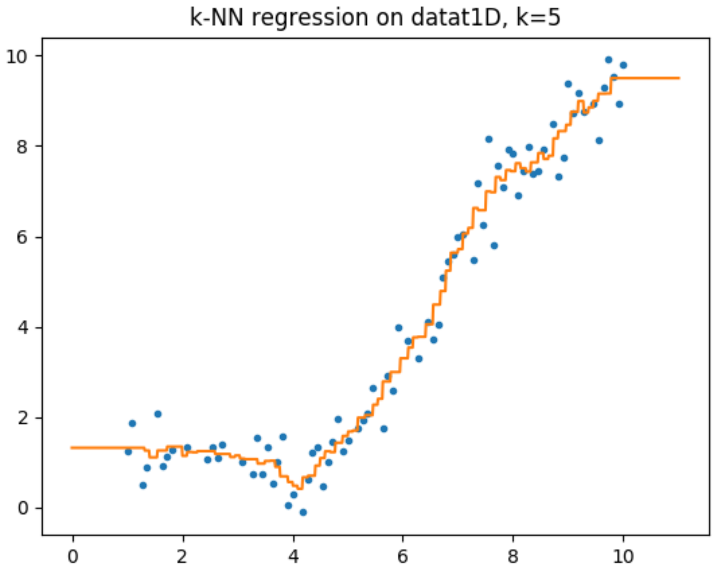
```
for kc in kvec:
    mse_train = sess.run(MSE, feed_dict={trainX:trainData, trainY:trainTarget,
                                         newX:trainData, newY:trainTarget, K:kc})
    mse_valid = sess.run(MSE, feed_dict={trainX:trainData, trainY:trainTarget,
                                         newX:validData, newY:validTarget, K:kc})
    mse_test = sess.run(MSE, feed_dict={trainX:trainData, trainY:trainTarget,
                                        newX:testData, newY:testTarget, K:kc})
    mse_valid_list.append(mse_valid)
    mse_test_list.append(mse_test)
   print("K=%d\t training MSE: %f, validation MSE: %f, test MSE: %f"%\
          (kc, mse_train, mse_valid, mse_test))
    yp = sess.run(predY, feed_dict={trainX:trainData, trainY:trainTarget,
                                    newX:X, K:kc})
    plt.figure(kc+1)
    plt.plot(trainData,trainTarget,'.')
    plt.plot(X,yp,'-')
    plt.title("k-NN regression on datat1D, k=%d"%kc)
    plt.show()

k_best = kvec[np.argmin(mse_valid_list)]
print("best K using validation set is K=%d"%(k_best))
```

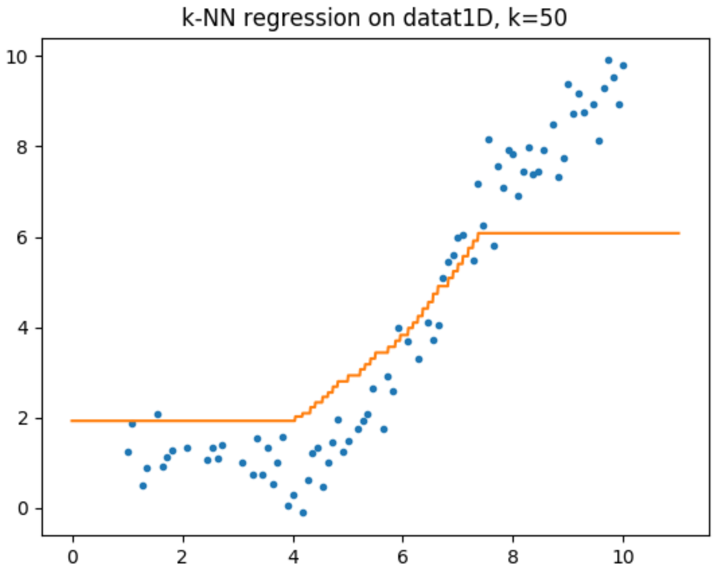
Table 1: The MSE for training, validation and testing datasets under different values of k

k	MSE		
	Train	Valid	Test
1	0	0.54	0.62
3	0.21	0.65	0.29
5	0.237	0.62	0.36
50	2.49	2.46	1.41


k-NN regression on datat1D, k=1
10
8
6
4
2
0
0
2
4
6
8
10


k-NN regression on datat1D, k=3
10
8
6
4
2
0
0
2
4
6
8
10





3 Making Predictions for Classification

1. **Predicting class label:**

   One possible implementation

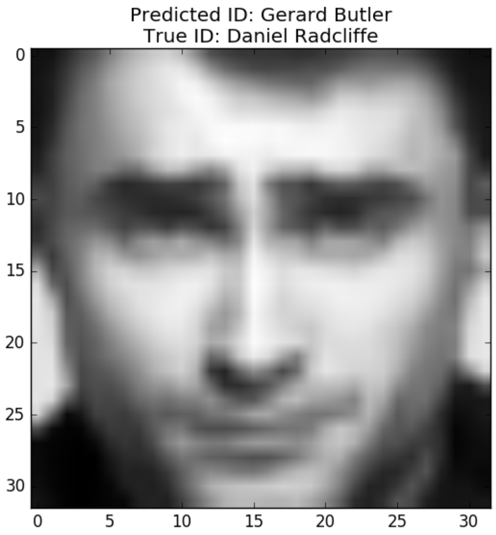
   ```
   def predict(pair_dist, train_target, k):
   ```

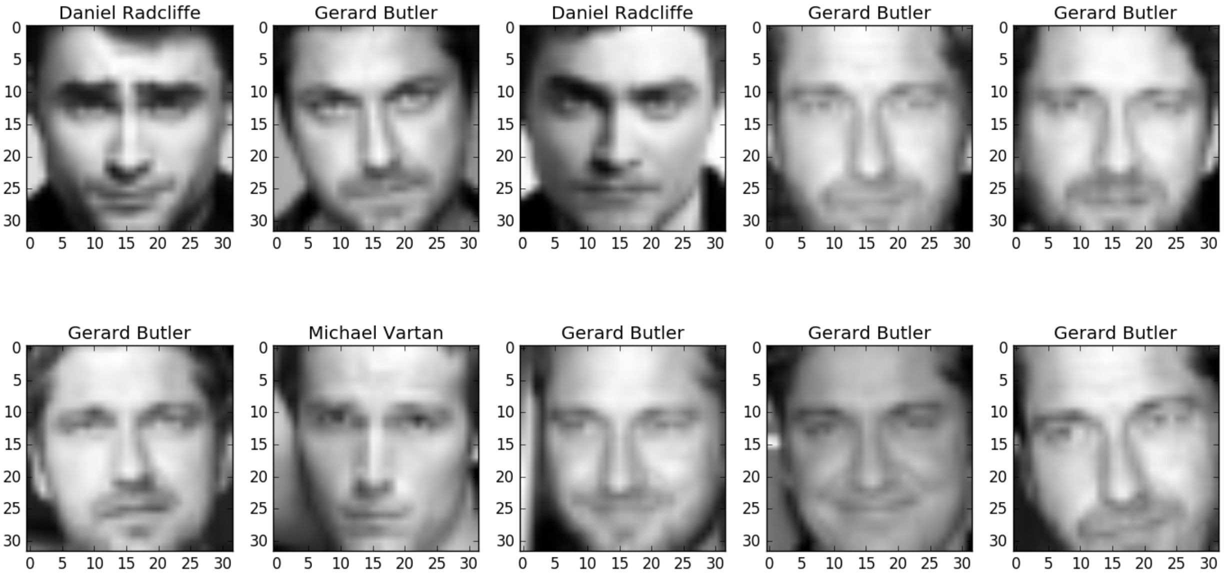
```
# Inputs
# pair_dist: is a N1xN2 matrix representing the pairwise distance
# K: is a scalar int representing number of neighbors

# finding the nearest K neighbors
top_k, ind_k = tf.nn.top_k(-pair_dist, k, sorted=True)

# get the class of the nearest neighbors
y_pred = tf.gather(train_target, ind_k, validate_indices = True)

# count the repeatition number of each  class
out, idx, count = tf.unique_with_counts(tf.squeeze(y_pred, 0))

# pick the maximum repeated class as the final class
y_final = tf.gather(out, tf.argmax(count))

return tf.to_int32(y_final), ind_k
```

2. **Face recognition using k-NN:**
   Results are provided in Table 2, the best validation accuracy achieved was 66.3% using $k = 1$. The test accuracy for $k = 1$ is 70.97%

Table 2: Validation accuracy for different values of k for face recognition problem

k	Validation Accuracy (%)
1	66.3
5	60.87
10	57.61
25	59.78
50	57.61
100	47.83
200	31.52





3. **Gender recognition using k-NN:**
Results are provided in Table 3, the best validation accuracy achieved was 91.3% using $k = 1$ and $k = 5$. The test accuracy for $k = 1$ is 92.47% and for $k = 5$ is 90.32%.

Table 3: Validation accuracy for different values of k for gender recognition problem

k	Validation Accuracy (%)
1	91.3
5	91.3
10	89.13
25	90.22
50	89.13
100	85.87
200	78.26