ECE521 Lab Report 1

February 2, 2018

- He Zhang 1000347546
- Xian Zhang 1000564766
- Yi Liu 999736889

This lab is for studying the KNN. Part1 is the practise of tensor flow matrix manuipulation. Part2 build KNN for simple regression use. Part3 uses KNN and faces dataset for classification purpose.

1 Part 1

```
In [2]: import tensorflow as tf

        def euclidean_dist(x, z):
          n2 = tf.shape(z)[0]
          x_ = tf.expand_dims(x,len(x.shape))
          # x_ = tf.tile(x_, [1,1, n2]) #optional, will be broadcasted
          z_ = tf.expand_dims(tf.transpose(z),0)
          res = tf.square(x_- z_)
          res = tf.reduce_sum(res, 1)
          return res
```

2 Part 2

2.1 2.1

Write a vectorized Tensorflow Python function that takes a pairwise distance matrix and returns the responsibilities of the training examples to a new test data point.

```
In [220]: def getResponsibility(distances, test_point, k):
              k_neighbors, k_indices = tf.nn.top_k(distances, k) # size is n*k
              k_indices = k_indices[test_point]
              k_indices = tf.expand_dims(k_indices,-1)
              size = tf.shape(distances)[1]
              updates = tf.ones((tf.shape(k_indices)[0],), dtype=tf.int32)
              return tf.scatter_nd(k_indices, updates, [size])

          print(tf.Session().run(getResponsibility(tf.constant([[1,2,3],[1,2,3]]), 0, 1)))
```

2.2 2.2

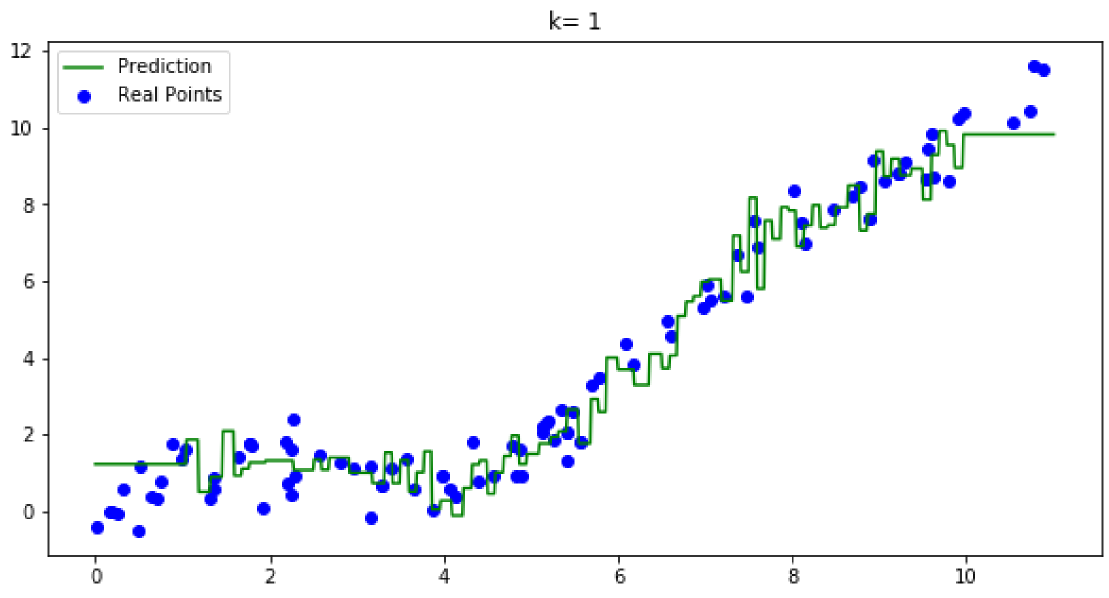
```
def getMSETensor(predictions, label):
    N = tf.cast(tf.shape(label)[0],tf.float32)
    mse = 1/(2*N) * tf.reduce_sum(tf.square((predictions-label)))
    return mse

class KNNBuilder:
  def __init__(self, k):
    self.k = k
  def build(self):
    tf.reset_default_graph()
    trainData = tf.placeholder(tf.float32, shape=(None, 1), name="trainData")
    trainTarget = tf.placeholder(tf.float32, shape=(None, 1), name="trainTarget")

    X = tf.placeholder(tf.float32, shape=(None,1), name="X")
    distances = -euclidean_dist(X,trainData)
    k_neighbors, k_indices = tf.nn.top_k(distances, k=self.k, name="k_neighbors") # size

    k_indices = tf.expand_dims(k_indices, 2)
    predictions = tf.gather_nd(trainTarget,k_indices)
    predictions = tf.reduce_mean(predictions, 1)

    X_labels = tf.placeholder(tf.float32, shape=(None,1), name="label")
    mse = getMSETensor(predictions, X_labels)

    return predictions, mse
```

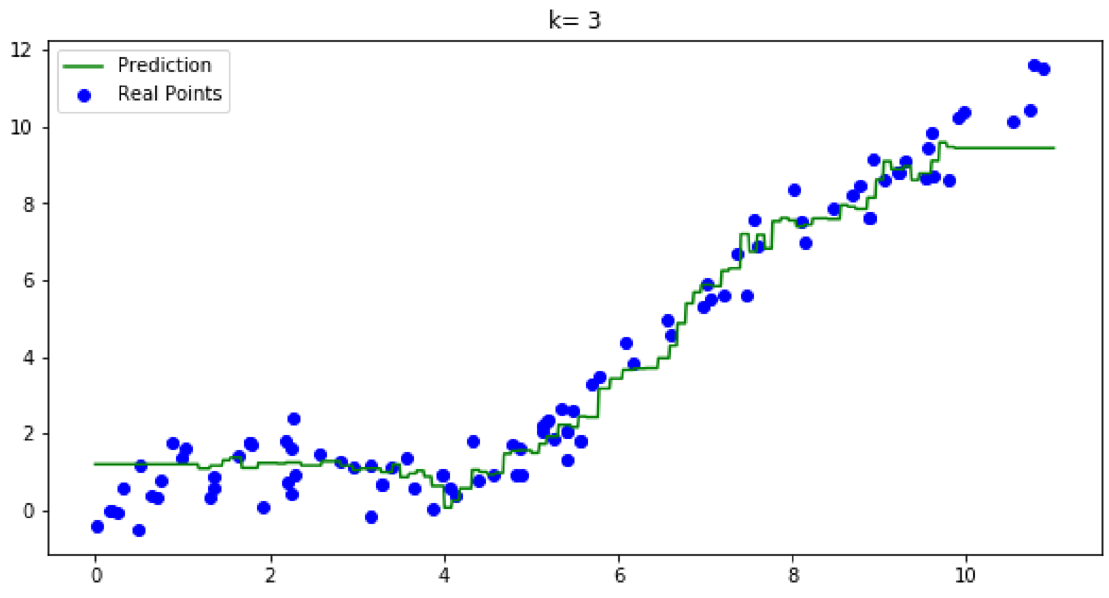
```
Validation Results:
```

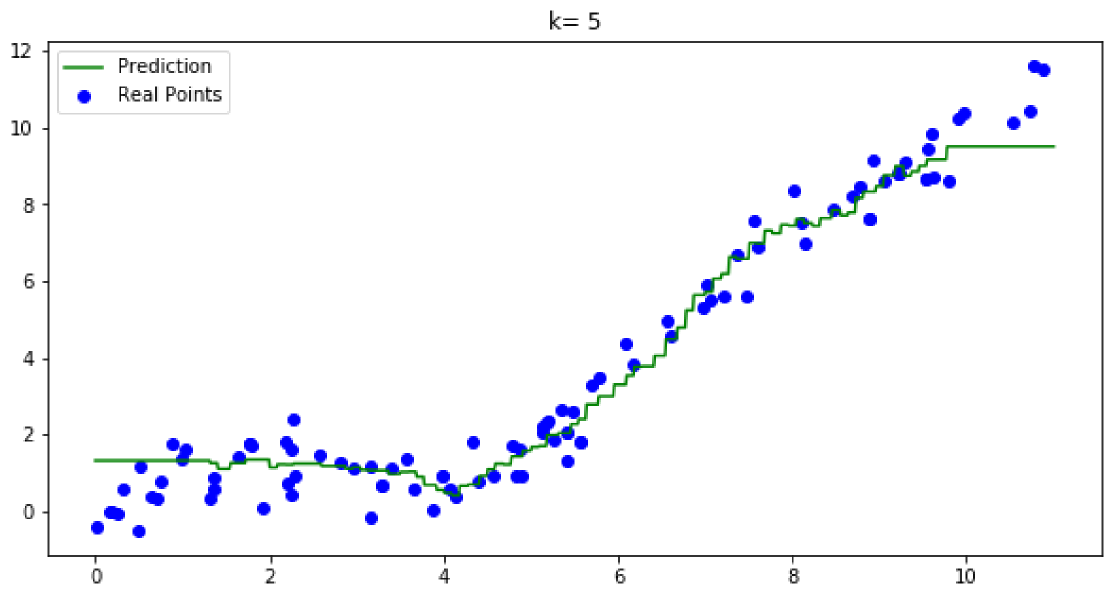
	training loss	validation loss	testing loss
k=1	0.0	0.272	0.311
k=3	0.105	0.326	0.145
k=5	0.119	0.310	0.178
k=50	1.248	1.229	0.707

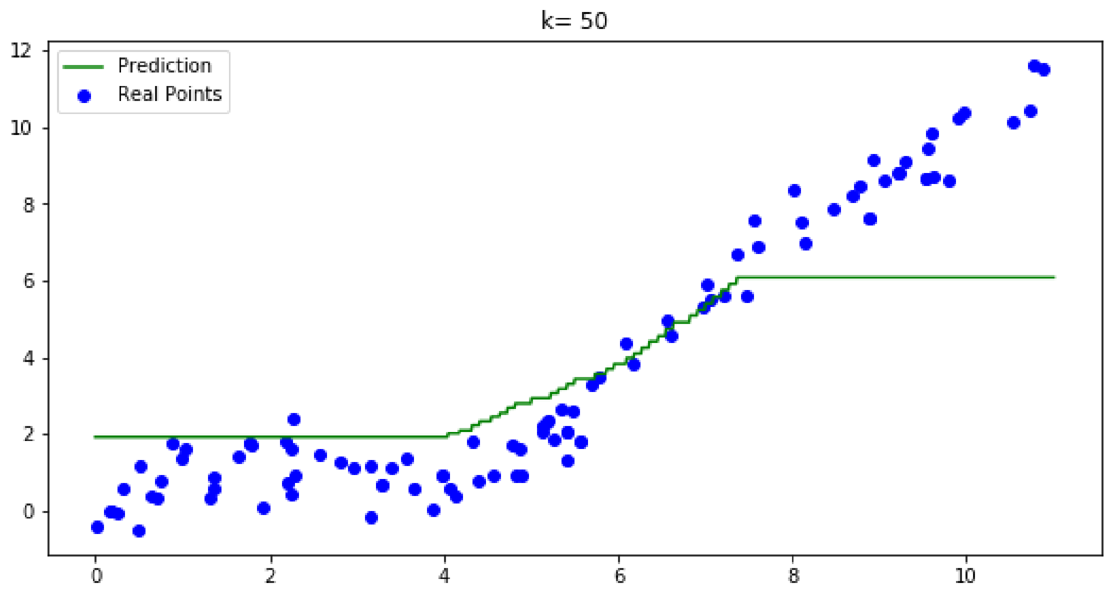
From the validation loss result, when k = 1 is the best. However, from the testing loss result, when k = 3 is the best.

```
Images:
```


k= 1
Prediction
Real Points
12
10
8
6
4
2
0
0
2
4
6
8
10


k= 3
Prediction
Real Points
12
10
8
6
4
2
0
0
2
4
6
8
10





From the graphs, when k = 5 it fits the data best and is not overfitting or underfitting as much. It is smoothier than the others and more likely to fit for more generalized data.

3 Part 3

3.1 3.1

```
class KNN_classifer_builder:
        def build(self):
            tf.reset_default_graph()
            trainData = tf.placeholder(tf.float32, shape=(None, 1024), name="trainData") #n2*1
            trainTarget = tf.placeholder(tf.int32, shape=(None, 1), name="trainTarget")
            X = tf.placeholder(tf.float32, shape=(None, 1024), name="X")

            distances = -euclidean_dist(X,trainData) #n1*n2
            k_values, k_indices = tf.nn.top_k(distances, k=self.k, name="k_neighbors") # size i

            k_indices = tf.expand_dims(k_indices, 2) #traintarget is 2d, so the indice should b
            predictions = tf.gather_nd(trainTarget,k_indices)
            predictions = tf.squeeze(predictions, 2)
            self.all_predictions = predictions
            return predictions
            #tf.unique_with_counts(predictions)
          def run(self, sess, test):
            feed_dict={tf.get_default_graph().get_tensor_by_name('trainData:0'):self.trainData,
                        tf.get_default_graph().get_tensor_by_name('trainTarget:0'):self
                        tf.get_default_graph().get_tensor_by_name('X:0'):test}

            res = []
            for i in range(test.shape[0]):
                get_counts_i = tf.unique_with_counts(self.all_predictions[i])
                uniques, indices, counts = get_counts_i
                max_i = tf.cast(tf.argmax(counts),tf.int32)
                tmp = uniques[max_i]
                res.append(tmp)
            return sess.run(res, feed_dict=feed_dict)
```

3.2 3.2

```
In [60]: def accuracy(predictions, labels):
            p = np.array(predictions)
            l = np.array(labels)
            return p[p == l].size / l.size
```

```
Validation Results:

k = 1: accuracy is 0.6630434782608695.
k = 5: accuracy is 0.6086956521739131.
k = 10: accuracy is 0.5]0869565217391.
k = 25: accuracy is 0.5978260869565217.
k = 50: accuracy is 0.5760869565217391.
k = 100: accuracy is 0.4782608695652174.
k = 200: accuracy is 0.31521739130434784.
```

From the validation results, the best k is when k = 1.

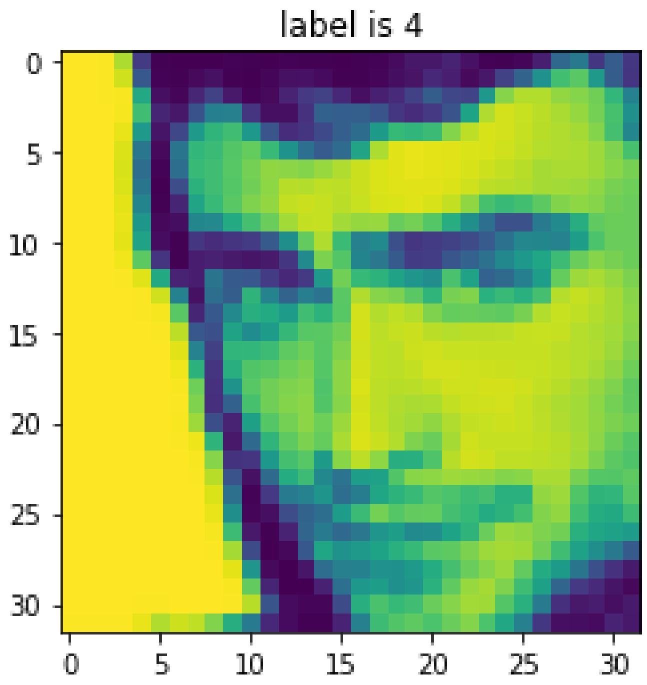
```
For test data:

k = 1: accuracy is 0.7096774193548387.
```

At k = 1, the accuracy for testing data is 70.97%

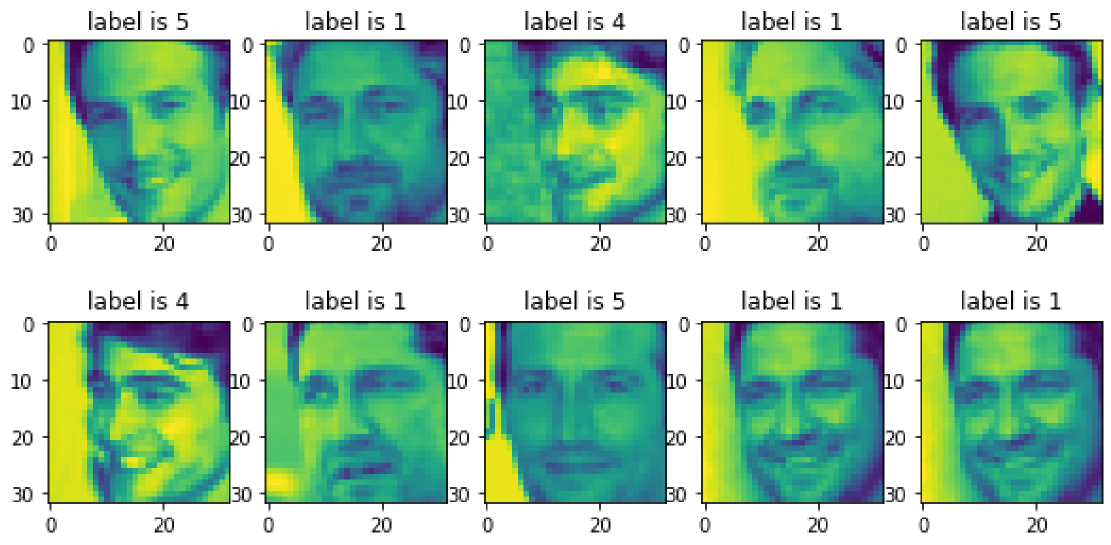
```
Images:
```

The 7th case from the testing data is predicted wrong, so we decide to use this one to study. plotting the picked wrong-predicted figure

```
In [146]: plt.imshow(testData[6].reshape(32,32))
          plt.title("label is {}".format(testTarget[6]))
```



Plotting the 10 nearest neighbours for the picked graph



All of these images share similar features. For example, a tilted smiling male face. However, the correct labeled neighbours only account for 2 out of the total 10 images. The facing direction might distract the predictor.

3.3 3.3

```
Now switch to the gender predictions:

Validation Tests:
for k in [1, 5, 10, 25, 50, 100, 200]:


k = 1: accuracy is 0.9130434782608695.
k = 5: accuracy is 0.9130434782608695.
k = 10: accuracy is 0.8913043478260869.
k = 25: accuracy is 0.9021739130434783.
k = 50: accuracy is 0.8913043478260869.
k = 100: accuracy is 0.8586956521739131.
k = 200: accuracy is 0.782608695652174.
```

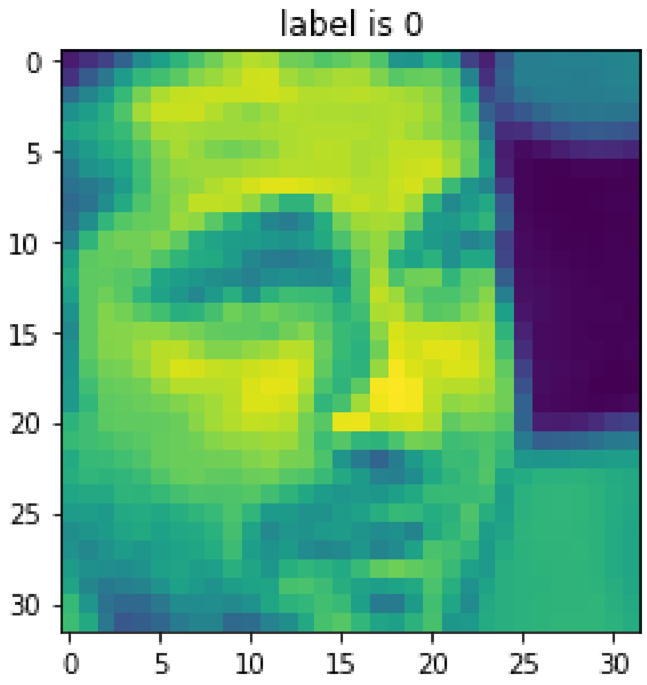
The best k is 1 and 5 and they are the same.

```
    Use Test Data:
k = 1: accuracy is 0.9247311827956989.
```

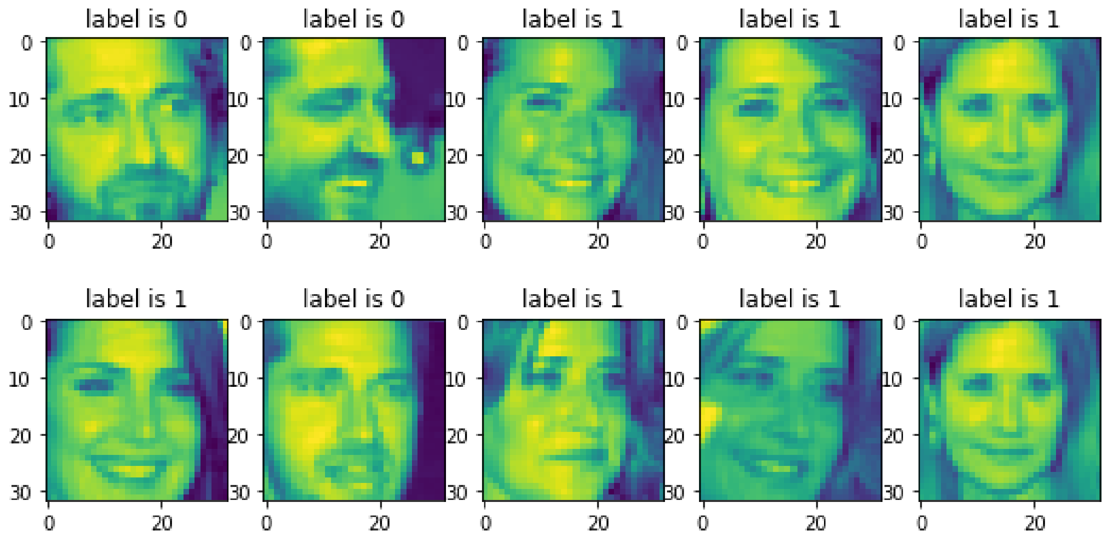
At k=1, the accuracy for testing data is 92.47%. The accuracy is pretty high. The classification for gender is a binary case which is simpler task comparing to the previous one.

```
plotting the picked wrong-predicted figure ( index = 1)
```

```
In [164]: plt.imshow(testData[1].reshape(32,32))
          plt.title("label is {}".format(testTarget[1]))
```



```
And plot the neighbours
```



The voting result is female which is wrong compared to true label. Some noise factors such as facing direction might cause distraction. Sometimes, the hair color is the same as the background color. So it's somewhat difficult to find the correct neighbours.