ECE521 Assignment 1

Fu Yan (1000819439)
Junjian Luo (1000394931)
Shuangfan Gao (1000523604)

January 31, 2018

1 Introduction

In this assignment, we explored a simple learning algorithm k^{th}-*nearest-neighbors* (kNN) and implemented it using TensorFlow in Part 1. We then applied our implementation in data regression, face recognition, and gender classification in Part 2 and 3.

2 Design and Discussion

The key idea of the kNN algorithm is that for each new data sample, we will take number k of its closest neighbors and make a prediction of its label based on its selected neighbors'.

2.1 Euclidean distance function

First we need to compute a pairwise Euclidean distance matrix between two given vector groups represented as two matrices X and Z. The pairwise distance matrix D can be computed as the following.

$$D_{ij} = ||x_i - z_j||_2^2$$

Our implementation of the distance function D_{euc} is shown in Figure 1.

```
# calculate euclidean distance square
def D_eucl_sqr(X, Z):
    if X.shape[1] != Z.shape[1]:
        print("D_eucl_sqr: X and Z must have the same number of column")
        return tf.constant([])
    return tf.reduce_sum(tf.square(X[:, None] - Z[None, :]), 2)
```

Figure 1: Python implementation of the Euclidean distance function in Tensorflow

2.2 Making Predictions for Regression

For regression application, we first compute the pairwise distance matrix using D_{euc} for training data set and the testing data set. Then we use the Tensorflow function $tf.nn.top\_k$ to select the closest entries from training set and return the responsibility matrix R. Finally, we compute prediction matrix by taking average of the selected entries' label in training set. The implementation is shown in Figure 2.

```python
# find k nearest neighbors, return responsility matrix
def K_NN(X, Z, k):
    if X.shape[0] < k:
        print("k exceeds the maximum allowed number by matrix!")
        return None;
    distance = D_eucl_sqr(Z, X)
    reciprocal = tf.reciprocal(distance)
    values, indices = tf.nn.top_k(reciprocal, k)
    kthValue = tf.reduce_min(values)
    # calculate condition mask
    cond = tf.greater_equal(tf.transpose(reciprocal), kthValue)
    # assign responsibility with 1 or 0
    R = tf.where(cond, tf.ones(cond.shape, tf.float64), tf.zeros(cond.shape, tf.float64))
    # normalize responsibilty to sum up to 1
    R = tf.divide(R, tf.reduce_sum(R, 0))
    return R


R = K_NN(training_data, testing_data, k)
predictions = tf.matmul(tf.transpose(R), training_target)
```

Figure 2: Python implementation of *kNN* function in Tensorflow

The result for different k values are shown in Figure 3 and 4. The blue dots are the training data, the green dots are the target values and the red dots are the predicted ones.

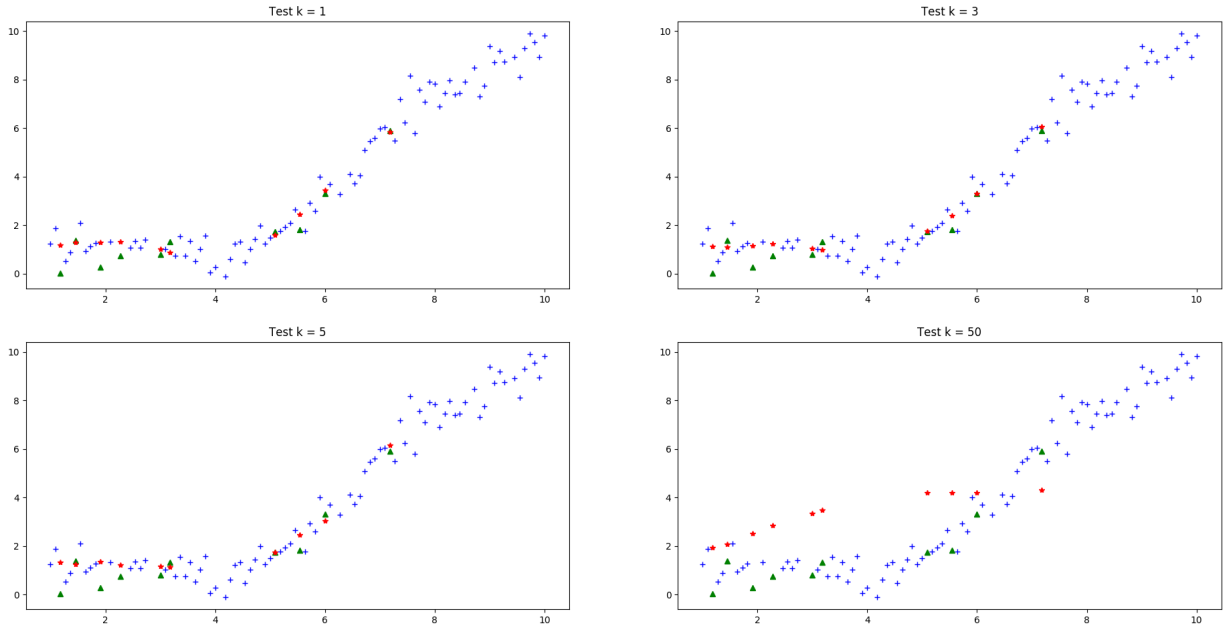


Figure 3: Regression predictions for testing data with *kNN*

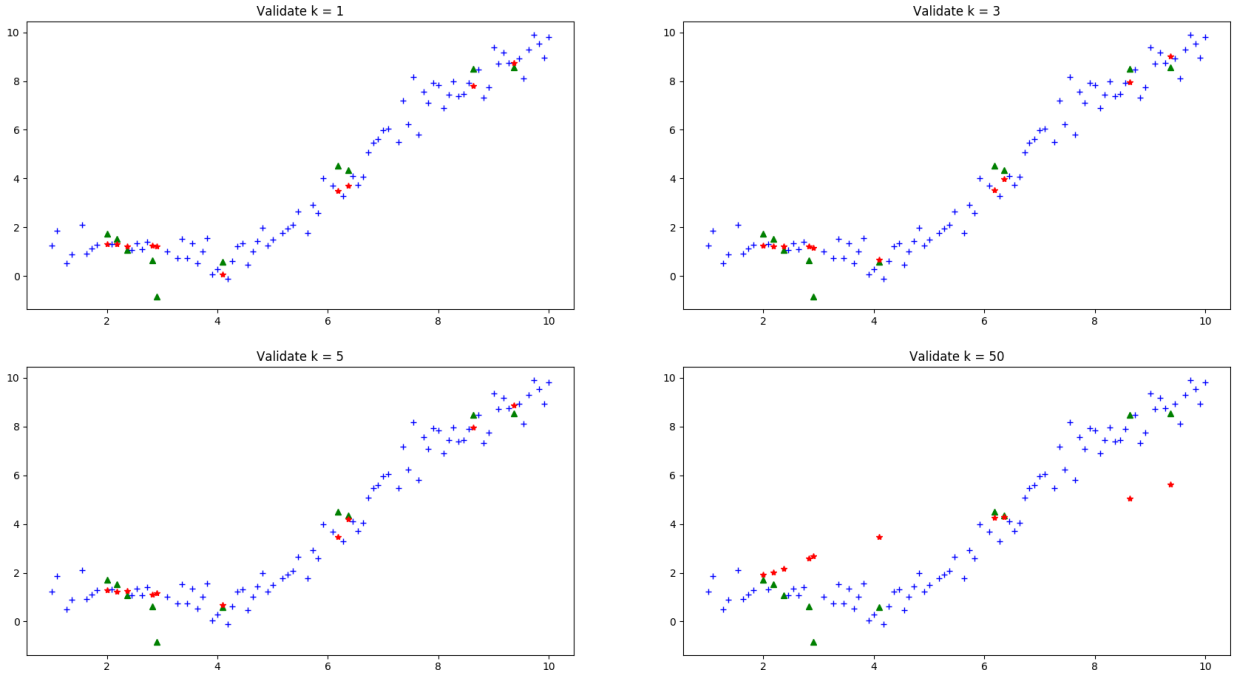


Figure 4: Regression predictions for validation data with *kNN*

The errors for different k values are shown in Table 1. Based on our results of validation data, we can conclude that the algorithm gives the best performance when $k = 5$ as it give the minimum validation errors.

	Prediction errors			
Errors	$k = 1$	$k = 3$	$k = 5$	$k = 50$
Training Error	0.0	0.137286	0.134678	3.766430
Testing Error	0.171928	0.143676	0.190297	1.971929
Validation Error	0.350622	0.314605	0.303125	2.326455

Table 1: Square Errors of predictions

2.3 Making Predictions for Classification

In this section, we modified our *kNN* function from previous section and enabled it to perform classification tasks. Instead of taking average of the label values, we used Tensorflow function $tf.unique\_with\_counts$ to get the majority-voted label for each testing data. The function is implement as shown in Figure 5.

```python
# find k nearest neighbors, return responsility matrix
def K_NN(X, Z, k):
    if X.shape[0] < k:
        print("k exceeds the maximum allowed number by matrix!")
        return None;
    distance = D_eucl_sqr(Z, X)
    reciprocal = tf.reciprocal(distance)
    values, indices = tf.nn.top_k(reciprocal, k)
    return indices


kVals = K_NN(trainData, testData, k)
votes = tf.cast(tf.gather(trainTarget, kVals), tf.int64)
predictions = tf.map_fn(lambda v: tf.gather(tf.unique_with_counts(v)[0], \
                                            tf.argmax(tf.unique_with_counts(v)[2])), votes)
```

Figure 5: Python implementation of *kNN* function in Tensorflow

To measure our algorithm's performance, we use the the following equation to calculate our errors.

$$E_{in} = \frac{1}{N}\sum_{i=1}^{N} 1(y_i \neq \hat{y}_i)$$

Errors of gender classification is displayed in Table 2. Errors of gender classification is displayed in Table 3.

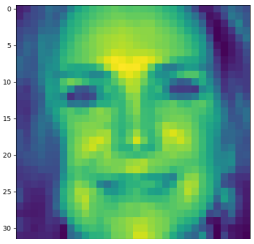
	Prediction errors						
Errors	$k=1$	$k=5$	$k=10$	$k=25$	$k=50$	$k=100$	$k=200$
Testing	0.290322	0.311827	0.333333	0.344086	0.419354	0.505376	0.602150
Validation	0.336956	0.391304	0.423913	0.402173	0.423913	0.521739	0.684782

Table 2: Errors of kNN face recognition

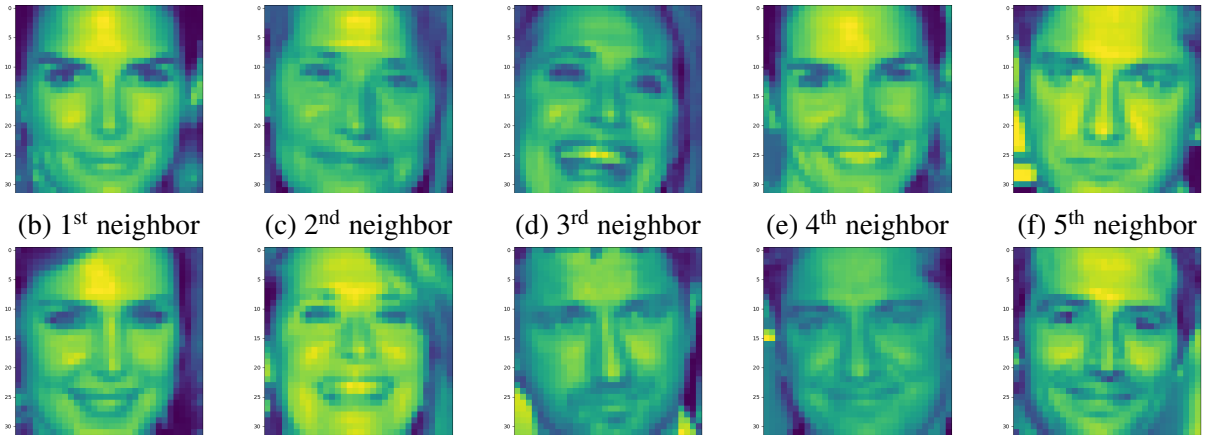
	Prediction errors						
Errors	$k=1$	$k=5$	$k=10$	$k=25$	$k=50$	$k=100$	$k=200$
Testing	0.075268	0.096774	0.107526	0.118279	0.139784	0.139784	0.225806
Validation	0.086956	0.086956	0.108695	0.097826	0.108695	0.141304	0.217391

Table 3: Errors of kNN gender classification

To understand the algorithm's performance and potential problems, we took a closer look at the case where $k = 10$. We displayed one of the falsely recognized picture and compare them with its neighbors as shown in Figure 6. We also perform the same analysis on incorrect one selected from our gender classification result as shown in Figure 7.



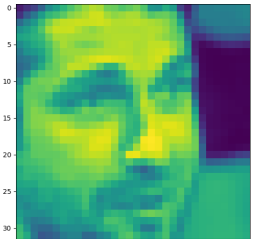
(a) Example of mistake in face recognition when $k = 10$



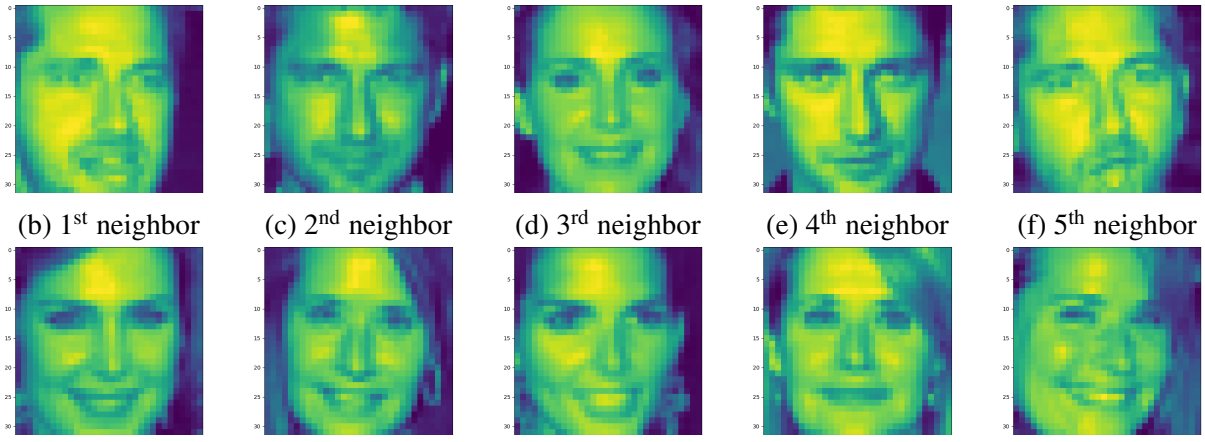
(b) 1st neighbor (c) 2nd neighbor (d) 3rd neighbor (e) 4th neighbor (f) 5th neighbor

(g) 6th neighbor (h) 7th neighbor (i) 8th neighbor (j) 9th neighbor (k) 10th neighbor

Figure 6: Incorrectly classified pictures in face recognition and its 10 neighbors



(a) Example of mistake in gender classification when $k = 10$



(b) 1st neighbor (c) 2nd neighbor (d) 3rd neighbor (e) 4th neighbor (f) 5th neighbor

(g) 6th neighbor (h) 7th neighbor (i) 8th neighbor (j) 9th neighbor (k) 10th neighbor

Figure 7: Incorrectly classified pictures in gender classification and its 10 neighbors

Clearly, the neighbors may contains some pictures of completely different persons with similar face direction. Thus we can say that the kNN algorithm is not idea for application for feature recognition. One possible improvement we can make in this case is to increase the training set. As we increase the number of training data, it is more likely to contain the pictures of the right person in the neighbors.

3 Conclusion

For regression problem, the kNN algorithm gives a adequate accuracy when k is around 5. It is simple in principle and good for many applications. However, its problem is that it requires much more time to train compare to other algorithms.