Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

ECE 521 Assignment 1

Work Breakdown

Group Member Name	Contribution Percentage
Fan Guo	0%
Jeffrey Kirman	50%
Connor Smith	50%

Part 1: Euclidean distance

As stated in the assignment for $N_1 \times D$ input tensor $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)^T} & \dots & \boldsymbol{x}^{(N_1)^T} \end{bmatrix}^T$ and $N_2 \times D$ input tensor $\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}^{(1)^T} & \dots & \boldsymbol{z}^{(N_2)^T} \end{bmatrix}^T$ the Euclidean distance is

$$D_{euc}(\boldsymbol{X}, \boldsymbol{Z}) = \begin{bmatrix} ||\boldsymbol{x}^{(1)} - \boldsymbol{z}^{(1)}||_2^2 & \cdots & ||\boldsymbol{x}^{(1)} - \boldsymbol{z}^{(N_2)}||_2^2 \\ \vdots & \ddots & \vdots \\ ||\boldsymbol{x}^{(N_1)} - \boldsymbol{z}^{(1)}||_2^2 & \cdots & ||\boldsymbol{x}^{(N_1)} - \boldsymbol{z}^{(N_2)}||_2^2 \end{bmatrix}.$$

The **euclidean_distance.py** (*Appendix A*) function evaluates this using vectorization. It first converts the input matrices into 3D tensors of shape $N_1 \times 1 \times D$ and $1 \times N_2 \times D$ for input tensors $\boldsymbol{X}$ and $\boldsymbol{Z}$, respectively. These new tensors are subtracted from each other which broadcasts both vectors into the shape of a $N_1 \times N_2 \times D$ tensor before evaluation. This resultant tensor is then piecewise squared and all the elements on the D length axis are summed together to result in the Euclidian distance matrix.

Part 2: Regression

Question 1: Choosing nearest neighbours

For a given input features vector $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(1)^T} & \dots & \boldsymbol{x}^{(N_1)^T} \end{bmatrix}^T$ and targets vector $\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}^{(1)^T} & \dots & \boldsymbol{y}^{(N_1)^T} \end{bmatrix}^T$, let $\mathcal{N}_{x^*}^k \subseteq \{\boldsymbol{x^1}, \dots, \boldsymbol{x^N}\}$ denote the k nearest neighbors as measured using the above **euclidean_distance** function above and selected using the indices identified by the **tf.nn.top_k** function on the distance matrix. Then, the prediction function $\widehat{\boldsymbol{y}}(\boldsymbol{x}^*)$ is defined as

$$\widehat{\boldsymbol{y}}(\boldsymbol{x}^*) = \boldsymbol{Y}^T \boldsymbol{r}^*$$

Where $\boldsymbol{r}^*$ is the responsibility vector is constructed using the definition

$$\boldsymbol{r}^* = [r_1, r_2, \dots, r_N], r_n = \begin{cases} \frac{1}{k}, & x^{(n)} \in \mathcal{N}_{x^*}^k \\ 0, & \text{otherwise} \end{cases}$$

The full code for calculating this responsibility vector is available in Appendix A – **choosing_nearest_neighbours.py**.

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Question 2: Prediction

For the *data1D* dataset generated as instructed, the following mean squared error loss was calculated from the prediction function as follows:

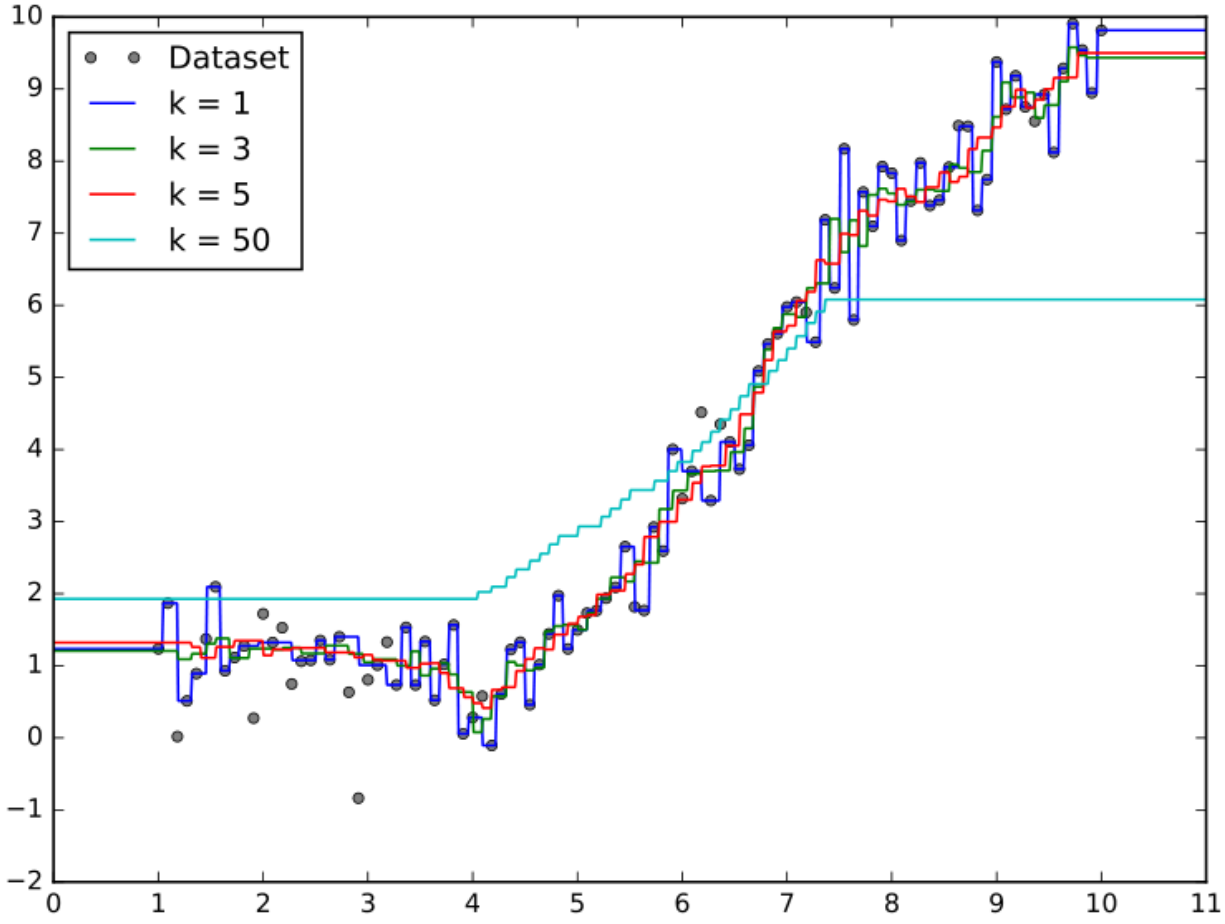
$$\mathcal{L} = \frac{1}{2N}\sum_{n=1}^{N}\left\|\hat{\boldsymbol{y}}\left(\boldsymbol{x}^{(n)}\right) - \boldsymbol{y}^{(n)}\right\|_2^2$$

The calculated MSE values for the Training, Validation and Test datasets for values of $k \in \{1, 3, 5, 50\}$ are recorded in the table below. Code used to generate this data is available in *Appendix A* – **prediction.py**.

k	Training MSE Loss	Validation MSE Loss	Test MSE Loss
1	0.0000	0.2716	0.1394
3	0.1065	0.3244	0.1588
5	0.1211	0.3167	0.1851
50	1.2460	1.2287	0.7026

The following figures show the prediction lines overlaid with plots of the entire dataset *data1D.* Code used to generate these plots is available in *Appendix A* – **prediction2.py**. Plots of each individual lines can be seen in *Appendix B*.

Plot of the Prediction Function for $k = \{1, 3, 5, 50\}$



Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

As seen in the plot, the higher the k value, the smoother the line appears to be. This makes sense since each data point is equally weighted when choosing nearest neighbours and hence a change in nearest neighbours will have less of an effect on the prediction. The $k = 1$ line goes through all the points in the training dataset as expected, since the function only uses one neighbour which essentially maps any test point to a point that exists in the training dataset. The $k = 50$ line seems to be the worst of the four at predicting the correct trend. This is because the k chosen is too close to the total size of the training dataset, and takes too many data points into account that are not necessarily close to the test point in question.

The best choice of k using the plot would be $k = 5$ since it best follows the trend of points in the dataset while remaining smooth.

Part 3: Making Predictions for Classification

Question 1: Predicting Class Label

Using a slightly modified version of the code from Part 2 above, input features were sorted by Euclidean distance from all available training features and had the smallest k selected. From these k results, the indices identified by the **tf.nn.top_k** function are now matched with a training dataset, and the most common associated label is returned as the predicted label for the new point. The code to accomplish this task is available in *Appendix A* – **kNN_classification.py**.

Question 2: Face recognition using k-NN

Using the given face dataset with the label (name) prediction code above and varying k, the following accuracy results were obtained, defining accuracy as $\frac{\#\text{ of correct classifications}}{\#\text{ of labels in dataset}}$:

k	Training Accuracy	Validation Accuracy	Test Accuracy
1	100%	66.3%	71.0%
5	80.2%	60.9%	68.8%
10	72.4%	57.6%	66.7%
25	66.1%	59.8%	65.6%
50	58.8%	57.6%	58.1%
100	52.1%	47.8%	49.5%
200	42.7%	31.5%	39.8%

Using the value of $k = 1$ which maximizes validation accuracy, the calculated test accuracy was 71.0%.

For the $k = 10$ case, the images of an incorrect prediction with its 10 nearest neighbors is available in Appendix C. Code for this task is available in *Appendix A – Q3.py* and must be configured to use the **name_task** task.

The flaw of using kNN classification for facial recognition is it only considers the intensity values of each pixel in an image, which means it is highly sensitive to changes in lighting and face position. Furthermore, since it does not rely on the recognition of facial features, it can easily misclassify. (e.g. a photo rotated 180 degrees can have a completely different classification than its original image.) It is for this reason the accuracy can be quite low, and that increasing the amount of nearest neighbours results in a general trend of decreasing accuracy.

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

The misclassification in Appendix C makes sense upon inspection, as all the photos have similar intensity profiles (i.e. their heads are positioned in the same places and the lighting is similar).

Question 3: Gender Recognition using k-NN

Repeating the above process for gender classification, the following accuracy results were obtained:

k	Training Accuracy	Validation Accuracy	Test Accuracy
1	100%	91.3%	92.5%
5	90.6%	91.3%	90.3%
10	90.2%	89.1%	89.2%
25	83.3%	90.2%	88.2%
50	82.3%	89.1%	86.0%
100	78.2%	85.9%	86.0%
200	72.6%	78.3%	77.4%

Using the value of $k = 1$ which maximizes validation accuracy, the calculated test accuracy was 92.5%

For the k=10 case, the images of an incorrect prediction with its 10 nearest neighbors is available in Appendix D. Code for this task is available in *Appendix A – Q3.py* and must be configured to use the **gender_task** task (by changing the *NN_type* variable).

The results are similar to that of name recognition, however, the accuracy results are considerably better for two reasons. First, there is only two categories for classification now, meaning there is more of a chance to get the classification right. Second, some features associated with photos of men that are different than that of women are now taken into consideration (i.e. wearing makeup can make the photo brighter and darker in certain areas of the photo).

The misclassification in Appendix D makes sense upon inspection, as all the photos have similar intensity profiles (i.e. their heads are positioned in the same places and the lighting is similar).

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Appendix A – Python code

choosing_nearest_neighbours.py

```
from euclidean_distance import euclidean_distance
import numpy as np
import tensorflow as tf

# Returns the responsibility vector r* for the new_point
def get_responsibility_matrix_indices(training_vectors, new_points, k):
    new_points = tf.reshape(new_points, [1,-1])
    pairwise_distances = euclidean_distance(training_vectors, new_points)
    values, indices = tf.nn.top_k(tf.transpose(-pairwise_distances), k,
sorted=True, name="responsibility_indices") # k nearest points
    return indices


def get_responsibility_matrix(training_vectors, new_points, k):
    indices = get_responsibility_matrix_indices(training_vectors, new_points, k)
    responsibility_value = tf.cast(1/k, dtype=tf.float64)
    off_value = tf.constant(0, dtype=tf.float64)
    r_depth = training_vectors.shape[0]
    r = tf.one_hot(indices=tf.transpose(indices), depth=r_depth,
on_value=responsibility_value, off_value=off_value, dtype=tf.float64)
    r = tf.reduce_sum(r, axis=0)
    return r # yT . r = k-NN prediction function y^


if __name__ == '__main__':
    sess = tf.InteractiveSession()

    t = tf.constant([[1,2,3],[4,5,6], [7,1,3], [6,0,1], [7,8,9], [3,6,8]])
    n = tf.constant([[3,4,5], [7,2,4], [3,4,7], [6,0,3]])
    k = tf.constant(3, dtype=tf.int32)

    print(sess.run(get_responsibility_matrix_indices(t,n,k)))

    print(sess.run(get_responsibility_matrix(t,n,k)))
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

euclidian_distance.py

```
import tensorflow as tf

def euclidean_distance(X, Z):
    D = X.shape[-1]
    X_int = tf.reshape(X, [-1, 1, D])
    Z_int = tf.reshape(Z, [1, -1, D])

    distance_pairs = X_int - Z_int
    eucl_dist = tf.reduce_sum(tf.square(distance_pairs), -1, name="euclidean_distances")
    return eucl_dist

if __name__ == '__main__':
    session = tf.InteractiveSession()
    X = tf.constant([[1,2,3], [4,5,6]])
    Z = tf.constant([[7,8,9], [1,2,3]])

    expected_result = tf.constant([[108, 0], [27, 27]])

    tf.assert_equal(euclidean_distance(X,Z), expected_result)
    print(session.run(euclidean_distance(X,Z)))
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

kNN_classification.py

```
import tensorflow as tf
from euclidean_distance import euclidean_distance

def kNN_classification(test_point, in_features, targets, k):
    distances = euclidean_distance(test_point, in_features)
    (val, ind) = tf.nn.top_k(-distances,k) # find closest neighbours in training set

    candidates = tf.gather(tf.constant(targets),ind) # Find the classifications for these
neighbours
    length = tf.shape(candidates)[0]
    class_list = []
    count_list = []

    # Count the frequency of nearest neighbours and put them into matrices
    # (reduced class list and count list)
    for i in range(0,length.eval()):
        (temp_class, __, temp_count) = tf.unique_with_counts(candidates[i])
        padding = tf.concat([tf.constant([0]), tf.constant([k]) -\ tf.shape(temp_class)],0)
        class_list.append(tf.pad(temp_class,[padding]))
        count_list.append(tf.pad(temp_count,[padding]))

    red_class_list = tf.stack(class_list)
    red_count_list = tf.stack(count_list)

    # Create an iterator for each test_point
    iterator = tf.cast(tf.linspace(0., length.eval() - 1., length.eval()),\ tf.int64)
    iterator = tf.reshape(iterator,[length,1])

    # Combine the iterator with the indices of the highest counts in the
    # reduced count list (red_count_list)
    count_loc = tf.concat([iterator, tf.reshape(tf.argmax(red_count_list,1),[length,1])],1)

    outputs = tf.gather_nd(red_class_list, count_loc)

    return (outputs, ind)

# Takes in 2 vectors and returns the % of occurrences they are the same elementwise
def classification_performance(results, targets):
    error = tf.count_nonzero(results - targets) / tf.cast(tf.shape(targets), tf.int64)
    return tf.cast(tf.constant(1.), tf.float64) - error
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

prediction.py

```
import tensorflow as tf
import numpy as np
from choosing_nearest_neighbours import *


def get_dataset():
  np.random.seed(521)
  Data = np.linspace(1.0, 10.0, num=100) [:, np.newaxis]
  Target = np.sin(Data) + 0.1*np.power(Data, 2) + 0.5 * np.random.randn(100, 1)
  randIdx = np.arange(100)
  np.random.shuffle(randIdx)

  return Data, Target, randIdx

def prediction():
  k_list = [1,3,5,50]
  Data, Target, randIdx = get_dataset()

  trainData = Data[randIdx[:80]]
  trainTarget = Target[randIdx[:80]]

  validData = Data[randIdx[80:90]]
  validTarget = Target[randIdx[80:90]]

  testData = Data[randIdx[90:100]]
  testTarget = Target[randIdx[90:100]]

  in_out_pairs = [(trainData, trainTarget, "train"), (validData, validTarget, "valid"),
(testData,\ testTarget, "test")]


  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for k in k_list:
      print("k=%d" % k)
      for X, Y, name in in_out_pairs:
        r_matrix = get_responsibility_matrix(trainData, X, k)
        y_preds = tf.reduce_sum(tf.transpose(trainTarget) * r_matrix, axis=-1)
        y_preds = tf.reshape(y_preds, [-1,1])
        error = tf.reduce_sum(tf.square(Y-y_preds)) / (2*X.shape[0])
        print("    set=%s error=%lf" % (name,error.eval()))
      print("\n")



if __name__ == '__main__':
    prediction()
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

prediction2.py

```
import tensorflow as tf
import numpy as np
from choosing_nearest_neighbours import *
import matplotlib.pyplot as plt


def get_dataset():
    np.random.seed(521)
    Data = np.linspace(1.0, 10.0, num=100) [:, np.newaxis]
    Target = np.sin(Data) + 0.1*np.power(Data, 2) + 0.5 * np.random.randn(100, 1)
    randIdx = np.arange(100)
    np.random.shuffle(randIdx)
    return Data, Target, randIdx

def kNN_regression(test_points, in_features, targets, k):
    r_star = get_responsibility_matrix(in_features, test_points, k)
    targets = tf.constant(targets, tf.float64)
    return tf.matmul(targets, r_star, True, True)

def calculate_mse(prediction, targets):
    size = targets.shape[0]
    return tf.reduce_sum(tf.square(targets-prediction)) / (2*size)

def prepare_data():

    Data, Target, randIdx = get_dataset()

    trainData = Data[randIdx[:80]]
    trainTarget = Target[randIdx[:80]]

    validData = Data[randIdx[80:90]]
    validTarget = Target[randIdx[80:90]]

    testData = Data[randIdx[90:100]]
    testTarget = Target[randIdx[90:100]]

    return [(trainData, trainTarget, "Training"), (validData, validTarget, "Validation"),
(testData, testTarget, "Test")]

def regression():

    k_list = [1,3,5,50]
    in_out_pairs = prepare_data()
    y_hat = []
    test_points = np.linspace(0.0,11.0,num = 1000)[:,np.newaxis]

    for k in k_list:
        prediction = kNN_regression(test_points, in_out_pairs[0][0], in_out_pairs[0][1], k)
        y_hat.append((prediction, k))

    return y_hat

def plot_combined(y_hat_list):

    # Create a figure of size 8x6 inches, 80 dots per inch
    plt.figure(figsize=(8, 6), dpi=80)

    # Create a new subplot from a grid of 1x1
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
    plt.subplot(1, 1, 1)

    # Prepare data
    (data, targets, __) = get_dataset();
    x = np.linspace(0.0,11.0,num = 1000)[:,np.newaxis]

    # Plot data points
    plt.plot(data, targets, 'o', color='#7f7f7f', markersize=4., label="Dataset")

    # Plot regression lines
    y = []
    for i in range(0,len(y_hat_list)):
        y.append(tf.transpose(y_hat_list[i][0]).eval())
        plt.plot(x, y[i], linewidth=1.0, linestyle="-", label="k = " +
str(y_hat_list[i][1]))

    # Set limits and ticks
    plt.xlim(0.0, 11.0)
    plt.xticks(np.linspace(0, 11, 12, endpoint=True))
    plt.ylim(-2.0, 10.0)
    plt.yticks(np.linspace(-2, 10, 13, endpoint=True))

    # Add legend
    plt.legend(loc='upper left')

    # Save figure to file
    plt.savefig("combined.pdf", format="pdf")

    # Show result on screen
    plt.show()

def plot_individual(y_hat):

    # Create a figure of size 8x6 inches, 80 dots per inch
    plt.figure(figsize=(8, 6), dpi=80)

    # Create a new subplot from a grid of 1x1
    plt.subplot(1, 1, 1)

    # Prepare data
    (data, targets, __) = get_dataset();
    x = np.linspace(0.0,11.0,num = 1000)[:,np.newaxis]

    # Plot data points
    plt.plot(data, targets, 'o', color='#7f7f7f', markersize=4., label="Dataset")

    # Plot regression line
    y = tf.transpose(y_hat[0]).eval()
    k = y_hat[1]
    plt.plot(x, y, color="red", linewidth=1.0, linestyle="-", label="k = " + str(k))

    # Set limits and ticks
    plt.xlim(0.0, 11.0)
    plt.xticks(np.linspace(0, 11, 12, endpoint=True))
    plt.ylim(-2.0, 10.0)
    plt.yticks(np.linspace(-2, 10, 13, endpoint=True))

    # Add legend
    plt.legend(loc='upper left')
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
    # Save figure to file
    plt.savefig("k" + str(k) + "-plot.pdf", format="pdf")

    # Show result on screen
    plt.show()

if __name__ == '__main__':
    sess = tf.InteractiveSession()
    init = tf.global_variables_initializer()
    sess.run(init)

    y_hat_list = regression()
    plot_combined(y_hat_list)
    for y_hat in y_hat_list:
        plot_individual(y_hat)
```

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Q3.py

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from kNN_classification import *

name_task = 0
gender_task = 1

def data_segmentation(data_path, target_path, task):
# task = 0 >> select the name ID targets for face recognition task
# task = 1 >> select the gender ID targets for gender recognition task
    data = np.load(data_path)/255
    data = np.reshape(data, [-1, 32*32])
    target = np.load(target_path)
    np.random.seed(45689)
    rnd_idx = np.arange(np.shape(data)[0])
    np.random.shuffle(rnd_idx)
    trBatch = int(0.8*len(rnd_idx))
    validBatch = int(0.1*len(rnd_idx))

    trainData, validData, testData = data[rnd_idx[1:trBatch],:], \
    data[rnd_idx[trBatch+1:trBatch + validBatch],:],\
    data[rnd_idx[trBatch + validBatch+1:-1],:]

    trainTarget, validTarget, testTarget = target[rnd_idx[1:trBatch], task], \
    target[rnd_idx[trBatch+1:trBatch + validBatch], task],\
    target[rnd_idx[trBatch + validBatch + 1:-1], task]

    return trainData, validData, testData, trainTarget, validTarget, testTarget

# Takes linearized picture data and puts it back into matrix form
def form_picture(data, index):
    pictures = np.reshape(data, [-1,32,32])
    return pictures[index]

def print_pictures(dataset, indeces, print_type):

    types = ["NN-Name-", "NN-Gender-", "OO-Name-", "OO-Gender-"]

    for i in range(0, len(indeces)):
        pic = np.reshape(dataset[indeces[i]], [-32,32])
        plt.imshow(pic, cmap="gray")

        # Save figure to file
        plt.savefig(types[print_type] + str(i) + ".pdf", format="pdf")

        # Show result on screen
        plt.show()

def perform_classification(NN_type):

    # List of nearest neighbours
    k = [1,5,10,25,50,100,200]

    # Load dataset
    (trainData, validData, testData, trainTarget, validTarget, testTarget) =
data_segmentation("data.npy", "target.npy", NN_type)
```

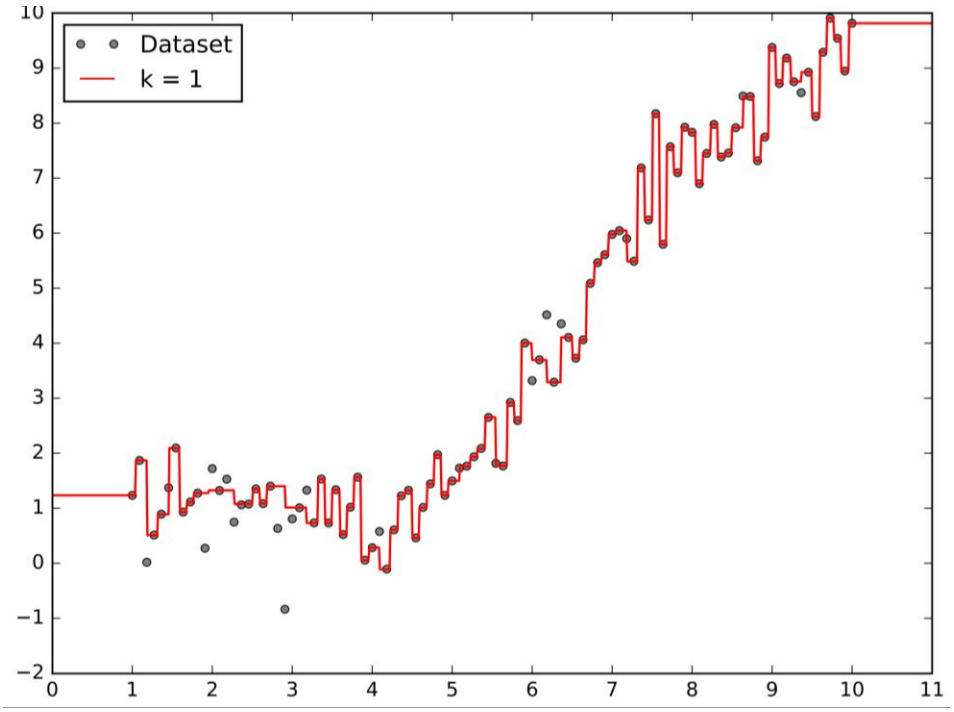
Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
    # Classification based on training data/targets
    classification_training = []
    performance_training = []
    for i in range(0,len(k)):
        classifications = kNN_classification(trainData, trainData, trainTarget, k[i])
        classification_training.append(classifications)

        performance = classification_performance(classifications[0], trainTarget)
        performance_training.append(performance)

    # Classification based on validation data/targets
    classification_validation = []
    performance_validation = []
    for i in range(0,len(k)):
        classifications = kNN_classification(validData, trainData, trainTarget, k[i])
        classification_validation.append(classifications)

        performance = classification_performance(classifications[0], validTarget)
        performance_validation.append(performance)

    # Classification based on test data/targets
    classification_test = []
    performance_test = []
    for i in range(0,len(k)):
        classifications = kNN_classification(testData, trainData, trainTarget, k[i])
        classification_test.append(classifications)

        performance = classification_performance(classifications[0], testTarget)
        performance_test.append(performance)

    return (classification_training, performance_training, classification_validation, \
            performance_validation, classification_test, performance_test)

if __name__ == '__main__':
    sess = tf.InteractiveSession()
    init = tf.global_variables_initializer()
    sess.run(init)

    # Put the task type here
    NN_type = name_task

    (trainData, validData, testData, trainTarget, validTarget, testTarget) =
data_segmentation("data.npy", "target.npy", NN_type)
    (classification_training, performance_training, classification_validation, \
            performance_validation, classification_test, performance_test) =
perform_classification(NN_type)

    # Picture at index = 0 is known to misclassify the name (0 instead of 3)
    if NN_type == name_task:
        print_pictures(validData, [0], NN_type+2)
        print_pictures(trainData, classification_validation[2][1][0].eval(), NN_type)
     # Picture at index = 1 is known to misclassify the gender (1 instead of 0)
    else:
        print_pictures(validData, [1], NN_type+2)
        print_pictures(trainData, classification_validation[2][1][1].eval(), NN_type)
```

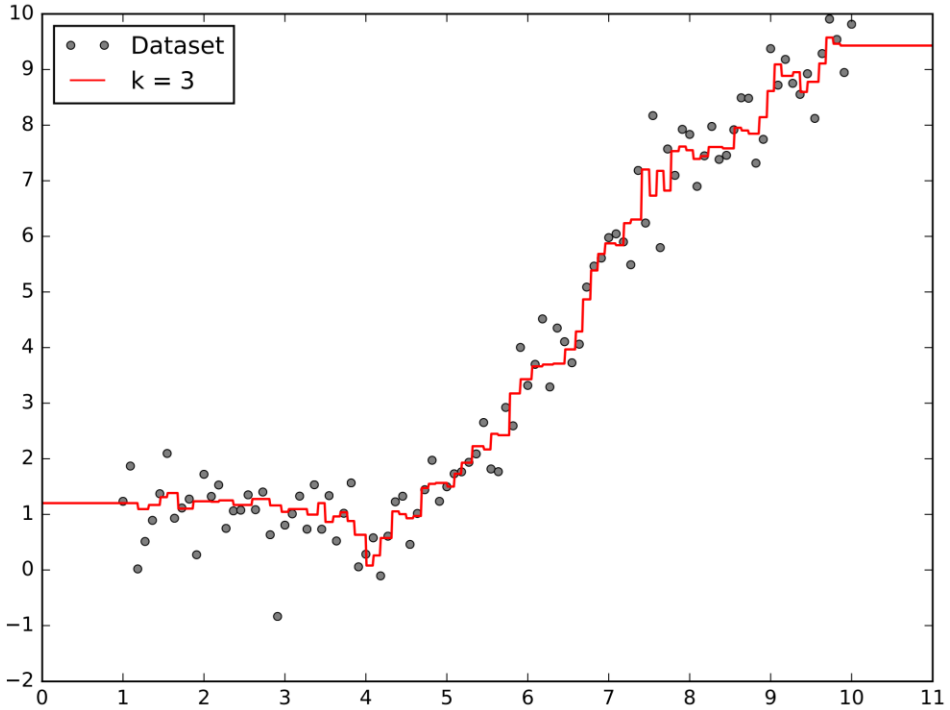
Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Appendix B – Individual plots of the prediction function

Plot of the Prediction Function for $k = 1$

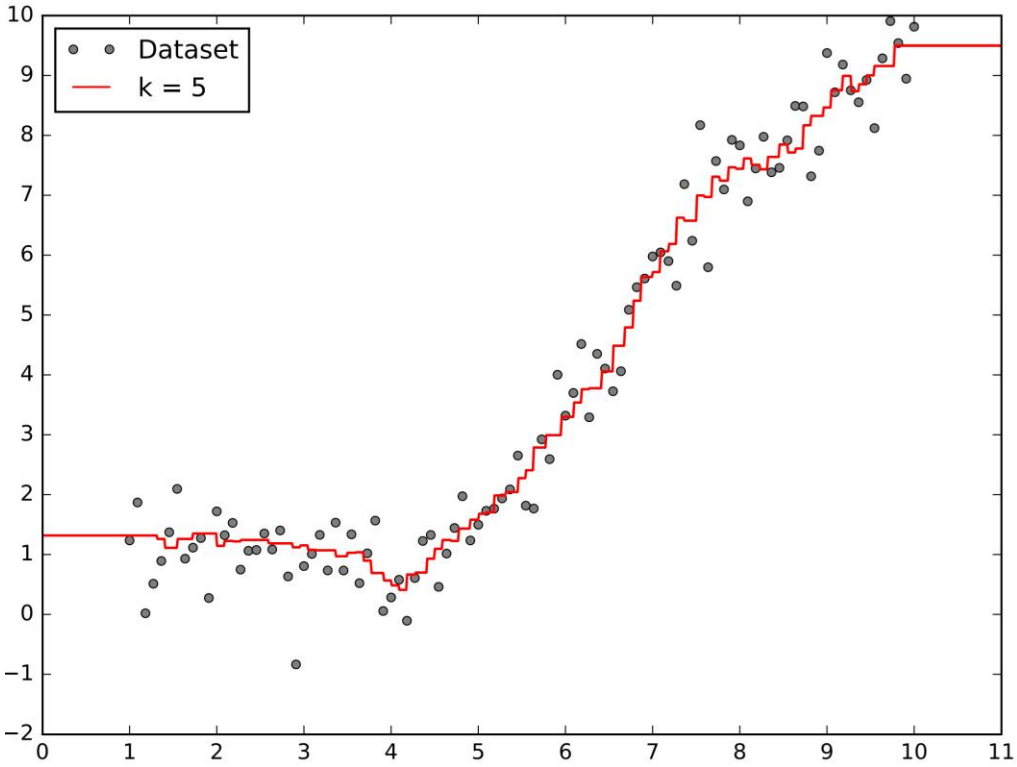


Plot of the Prediction Function for $k = 3$

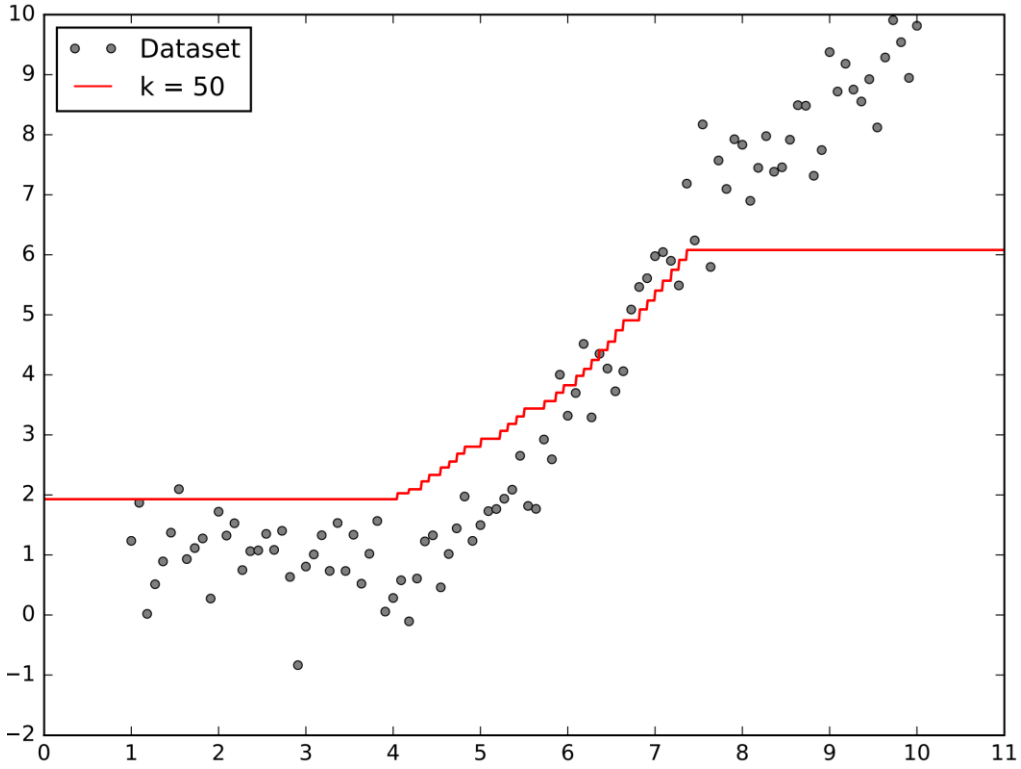


Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Plot of the Prediction Function for $k = 5$

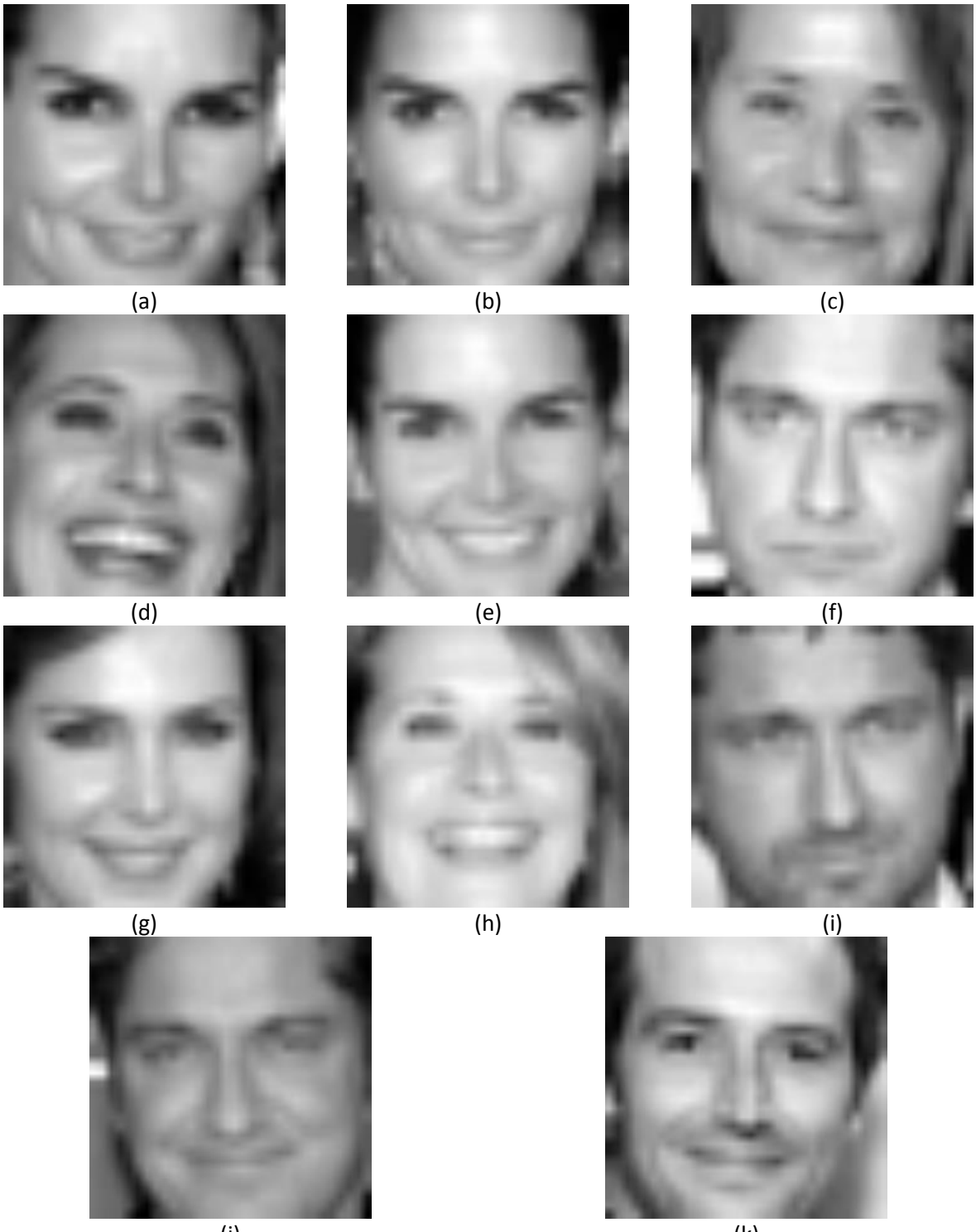


Plot of the Prediction Function for $k = 50$



Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Appendix C – Failed name classification



(a) (b) (c)

(d) (e) (f)

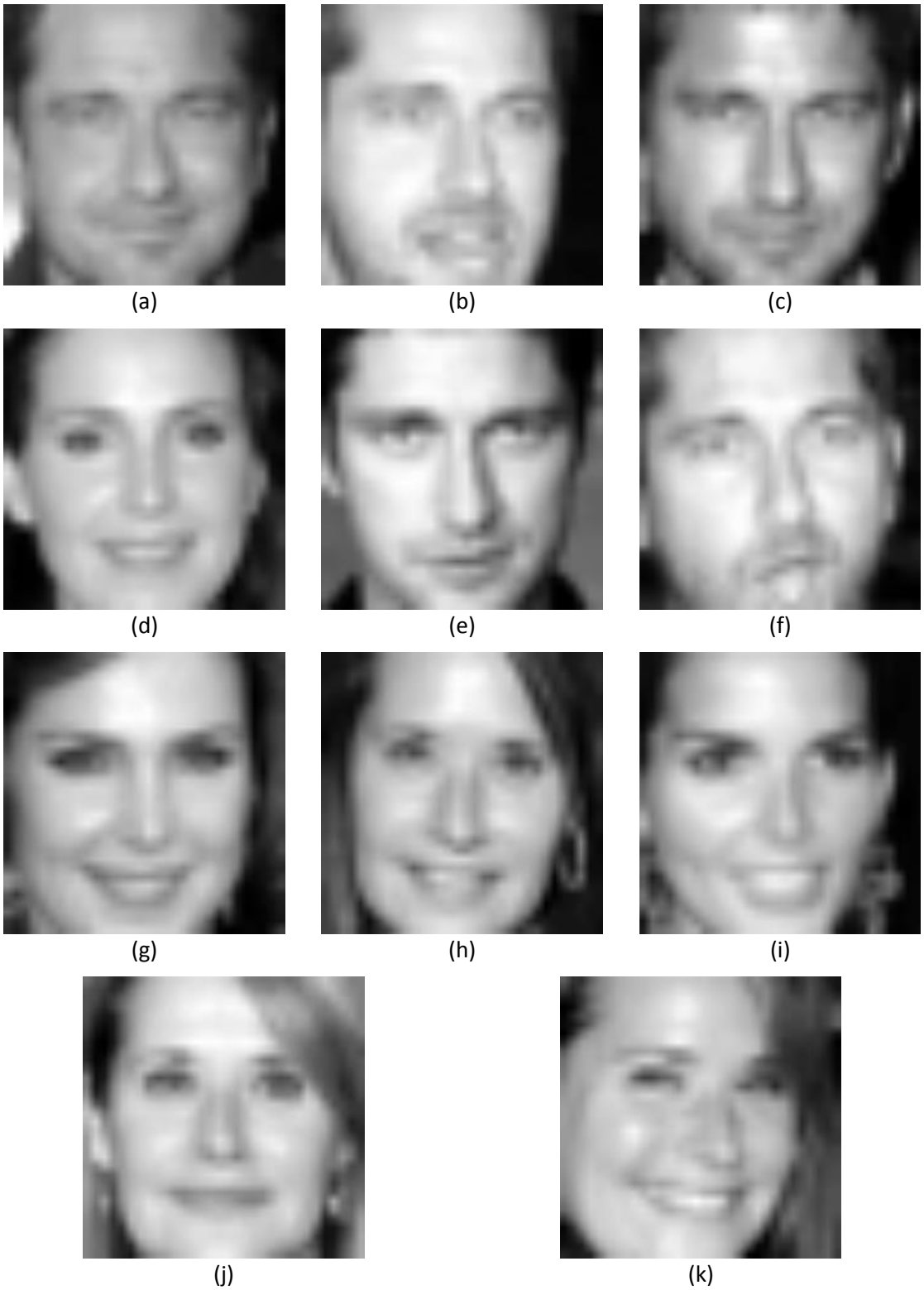
(g) (h) (i)

(j) (k)

Failed case of name classification where Angie Harmon (3) is misclassified as Lorraine Bracco (0): (a) original image, (b) – (k) closest to furthest nearest neighbour

Fan Guo (1000626539)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Appendix D – Failed gender classification



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k)

Failed case of gender classification where a male (0) is misclassified as a female (1): (a) original image, (b) – (k) closest to furthest nearest neighbour