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 input tensor and input tensor the Euclidean distance is

The **euclidean\_distance.py** (*Appendix A*) function evaluates this using vectorization. It first converts the input matrices into 3D tensors of shape and for input tensors and , respectively. These new tensors are subtracted from each other which broadcasts both vectors into the shape of a tensor before evaluation. This resultant tensor is then piecewise squared and all the elements on the 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 and targets vector , let denote thek 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 is defined as

Where is the responsibility vector is constructed using the definition

The full code for calculating this responsibility vector is available in Appendix A – **choosing\_nearest\_neighbours.py**.

## Question 2: Prediction

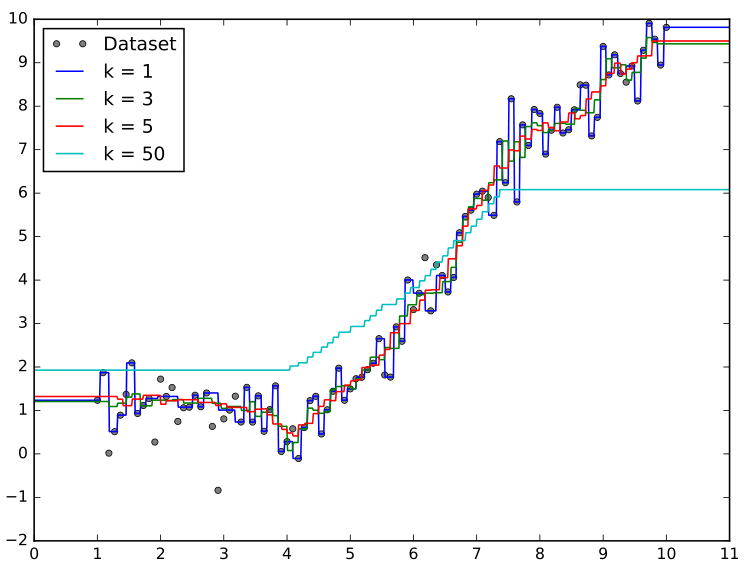
For the *data1D* dataset generated as instructed, the following mean squared error loss was calculated from the prediction function as follows:

The calculated MSE values for the Training, Validation and Test datasets for values of are recorded in the table below. Code used to generate this data is available in *Appendix A –* **prediction.py**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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



As seen in the plot, the higher the 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 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 line seems to be the worst of the four at predicting the correct trend. This is because the 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 using the plot would be 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 selected. From these 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 , the following accuracy results were obtained, defining accuracy as :

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 which maximizes validation accuracy, the calculated test accuracy was 71.0%.

For the 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.

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:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 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).

# 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**)))** |

## 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**)))** |

## 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 |

## 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**()** |

## prediction2.py

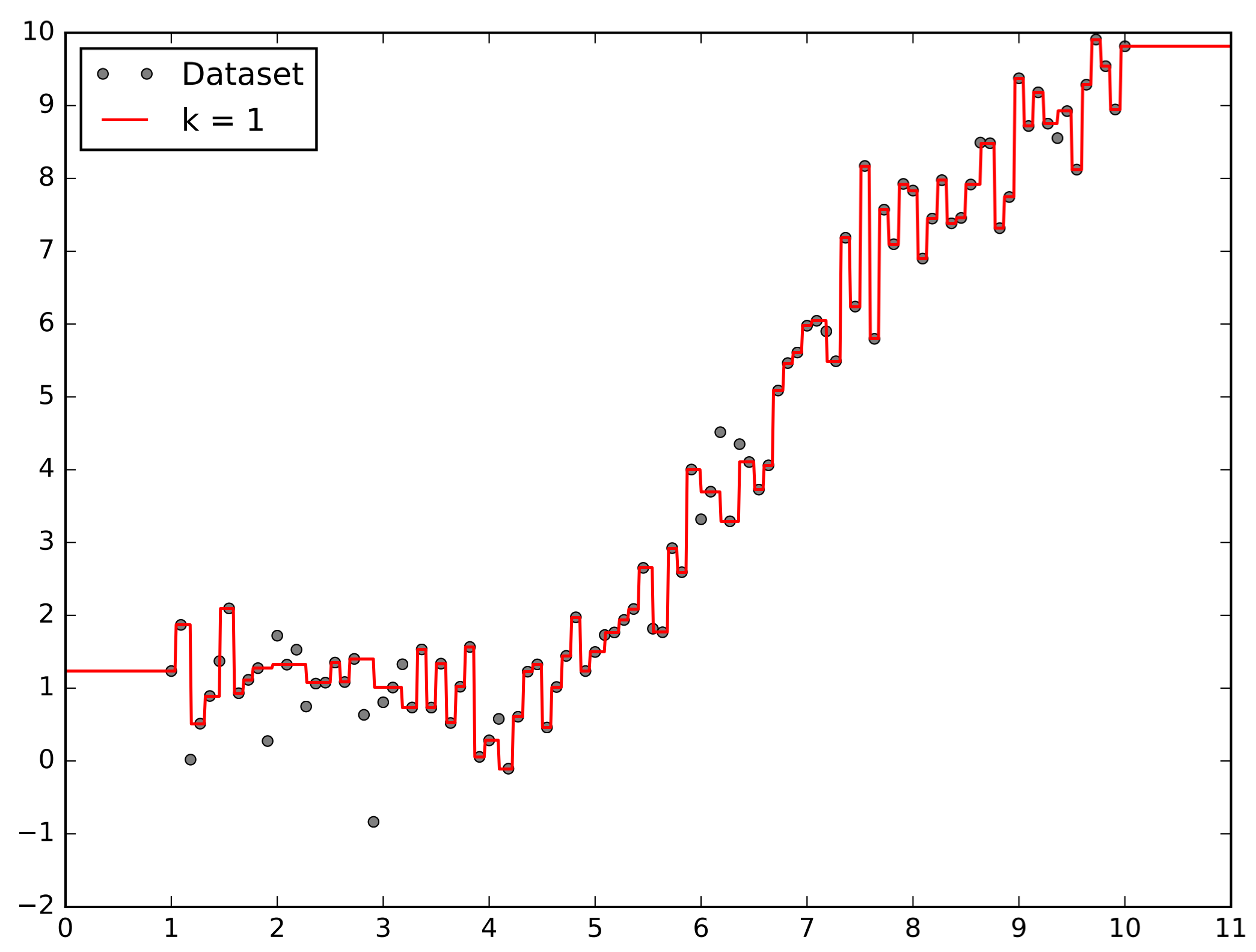
|  |
| --- |
| **import** tensorflow **as** tf  **import** numpy **as** np  **from** choosing\_nearest\_neighbours **import** **\***  **import** matplotlib**.**pyplot **as** plt  **def** get\_dataset**():**  np**.**random**.**seed**(**521**)**  Data **=** np**.**linspace**(**1.0**,** 10.0**,** num**=**100**)** **[:,** 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  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'**)**    # 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**)** |

## Q3.py

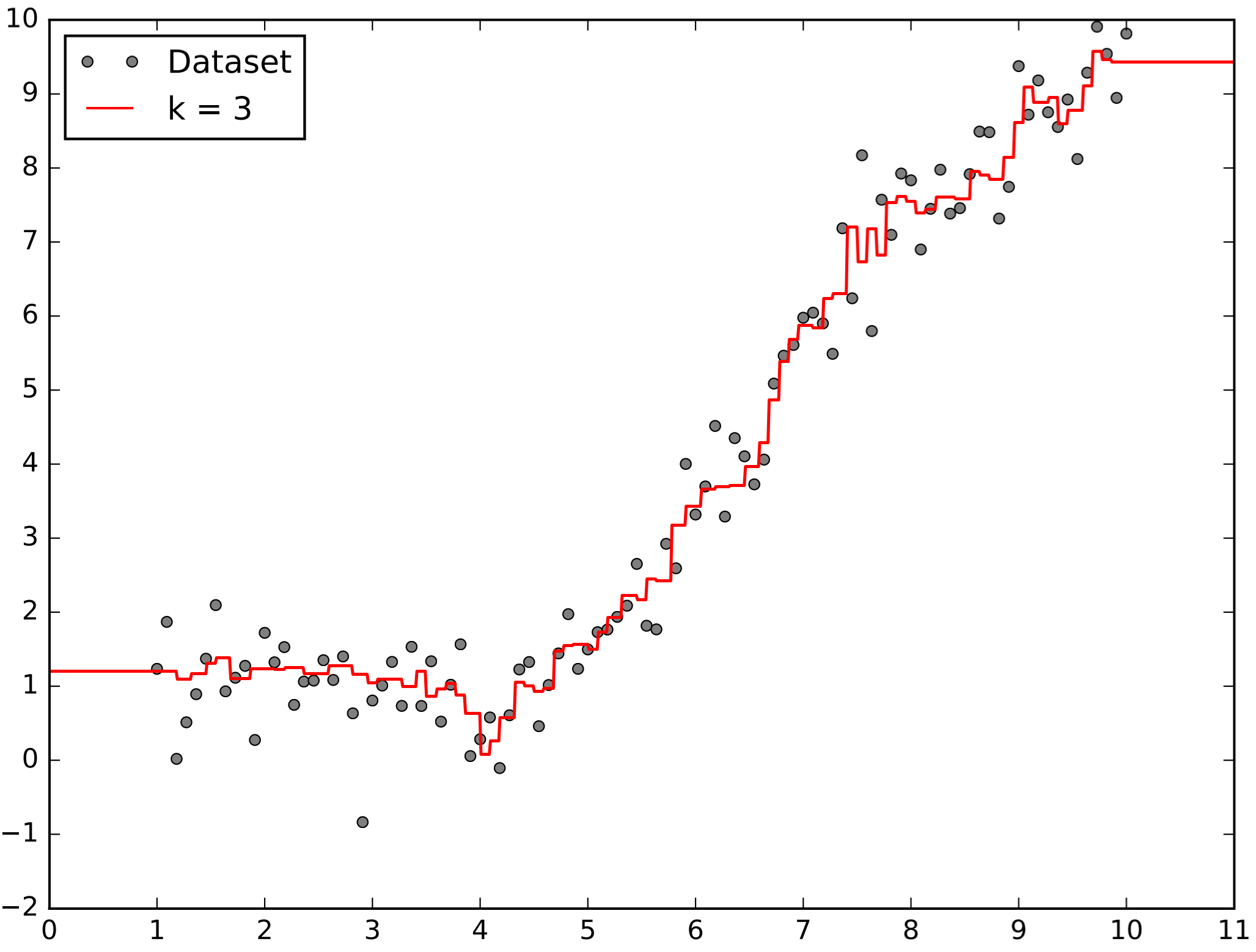
|  |
| --- |
| **import** tensorflow **as** tf  **import** numpy **as** np  **import** matplotlib**.**pyplot **as** plt  **from** kNN\_classification **import** **\***  name\_task **=** 0  gender\_task **=** 1  **def** data\_segmentation**(**data\_path**,** target\_path**,** task**):**  # task = 0 >> 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**)**    # 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**)** |

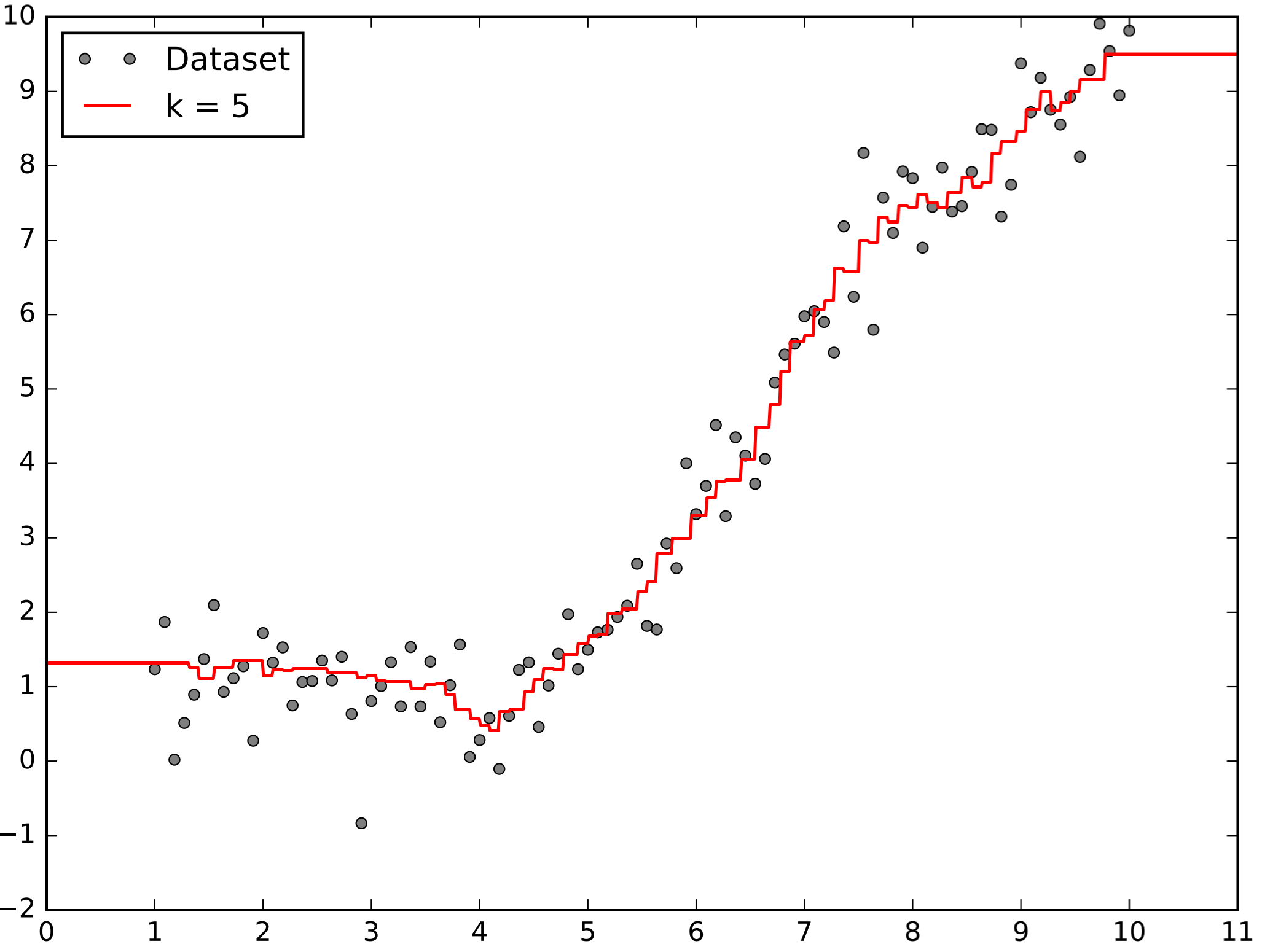
# Appendix B – Individual plots of the prediction function

Plot of the Prediction Function for

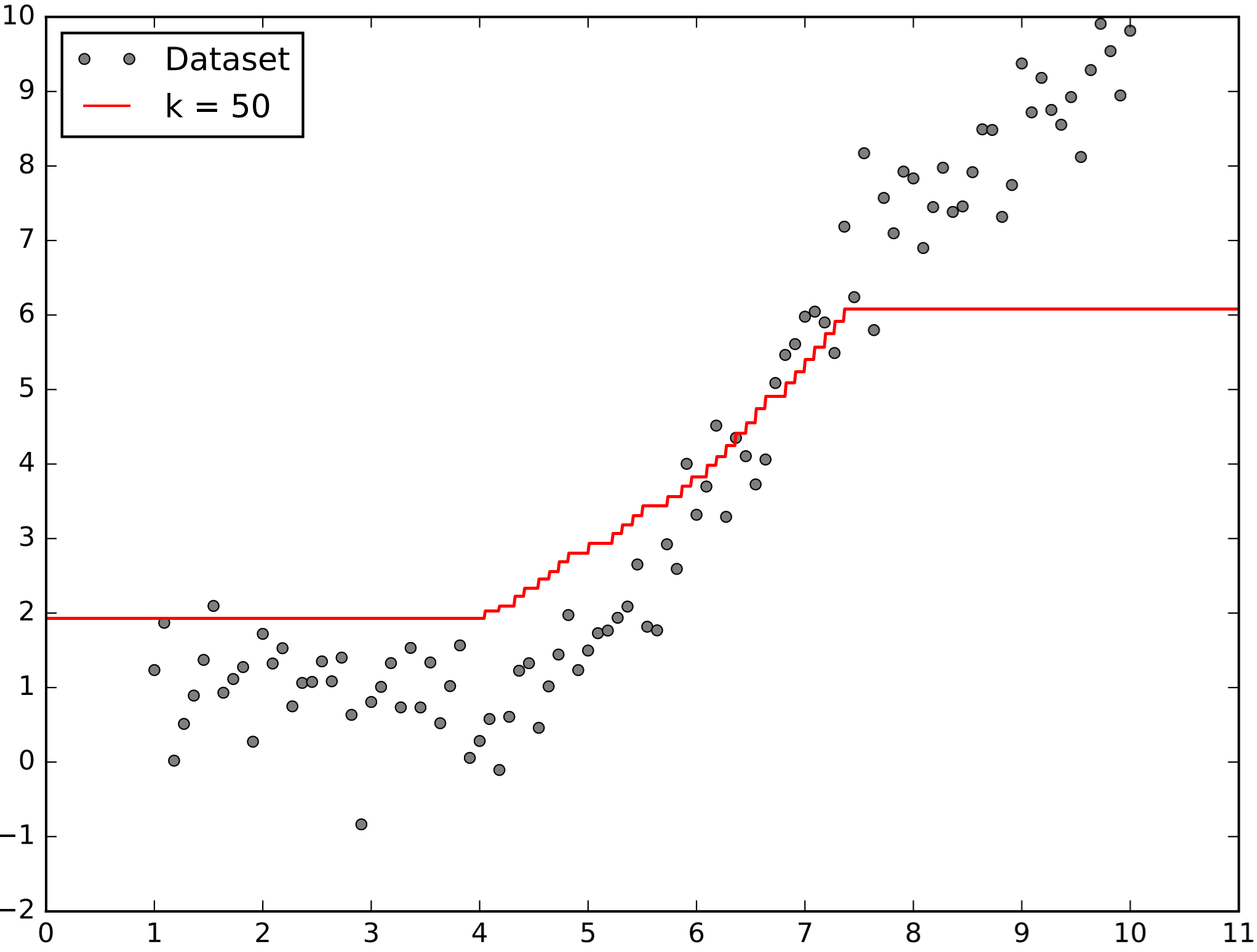


Plot of the Prediction Function for



Plot of the Prediction Function for 

Plot of the Prediction Function for



# 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

# 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