**ECE521: Assignment 1**

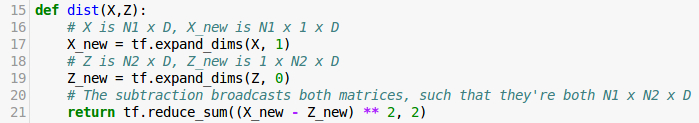
**K Nearest Neighbours (k-NN)**

**Younghoo Cho (1000594270), Deng Pan (1000548874), Keon Young Park (1004686253)**

**Equal contribution from each member.**

# **1 Euclidean Distance Function**

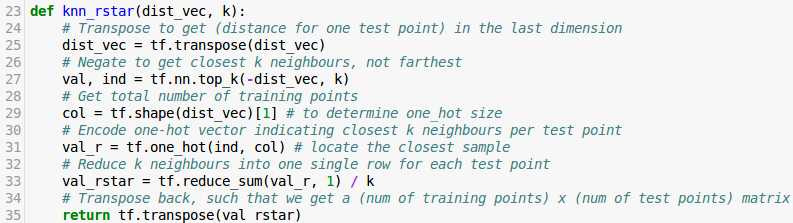
## **1.1 Pairwise distance:**



This function takes in two arguments. X is a N1 x D matrix and Z is a N2 x D matrix. In order to use Tensorflow broadcasting, lines 17 and 19 uses tf.expand\_dims() on both X and Z such that their dimensions become N1 x 1 x D and 1 x N2 x D, respectively. In this way, when doing the subtraction in the tf.reduce\_sum() call on line 21 between X\_new and Z\_new, Tensorflow broadcasting occurs such that both matrices become N1 x N2 x D in dimension. Then the reduce\_sum() sums over the last dimension (that has D elements) such that it returns an N1 x N2 matrix.

# **2 Making Predictions for Regression**

## **2.1 Choosing the nearest neighbours**



This function takes in the pairwise distance matrix “dist\_vec” and the number of neighbours to consider “k”. It returns the responsibility matrix which holds in its row the responsibility vector for each test point. Note that the pairwise distance matrix and the responsibility matrix have the same dimensions. Here’s an explanation for each line in the function:

|  |  |
| --- | --- |
| **Line** | **Description** |
| 25 | It transposes the pairwise distance matrix. This is done due to how tf.nn.top\_k() functions. tf.nn.top\_k() goes across the last dimension of the input matrix and finds the largest k values (ie. for k=6, the largest six values). Since this pairwise distance matrix comes from our dist() function (See Part 1), it is an N1 x N2 matrix, where N1 corresponds to the number of training points and N2 corresponds to the number of test points. Therefore, just using the non-transposed pairwise distance matrix with tf.nn.top\_k() returns the top k values across N2. Equivalently, it finds the top k values for one training point across all test points. Instead, what we want is to find the top k values for one *test* point across all training points. Therefore, a transpose needs to be done. |
| 27 | The parameter into tf.nn.top\_k() is negated (ie. -dist\_vec) because tf.nn.top\_k() gets the largest values. Since dist\_vec is the distance between test points and training points, getting the largest values means getting the points that are furthest away. |
| 29 | The size of distance matrix “dist\_vec” along axis 1 is being fetched into variable “col” using tf.shape(). Axis 1 represents the number of the elements inside each column of “dist\_vec”. This will be used inside one-hot encoding to generate the responsibility vectors. |
| 31 | Using the indices “ind” fetched from the top k elements inside the negative distance matrix with the help of tf.nn.top\_k(), combined with the depth (or equivalently the length) of the distance matrix’s columns, we can generate a tensor containing all individual vectors that have a single non-zero element with value 1 in a index where one of the top k elements inside the distance matrix occurs, for all top k elements. Doing this will necessarily add an extra dimension to the responsibility vectors, thus they will have to be reduced so that one single responsibility vector corresponding to each test point inside the predicted dataset will be generated. |
| 33 | The responsibility tensor val\_r is now passed into tf.reduce.sum(val\_r, 1) for reduction along axis 1. The resulting matrix will contain rows that corresponds to an unnormalized responsibility vector to each point inside the predicted dataset. This means that elements inside these rows will be 1s and 0s. The 1s will appear on indices where the top k nearest training data points occur with respect to a point in the test dataset. This is then divided by k so that each top k element will contribute 1/k to the final averaging of the training output values. |
| 35 | When returning this matrix of all responsibility vectors for each point inside the predicted set, these vectors should appear as columns of this matrix. Thus, a transpose is taken before returning the responsibility matrix. |

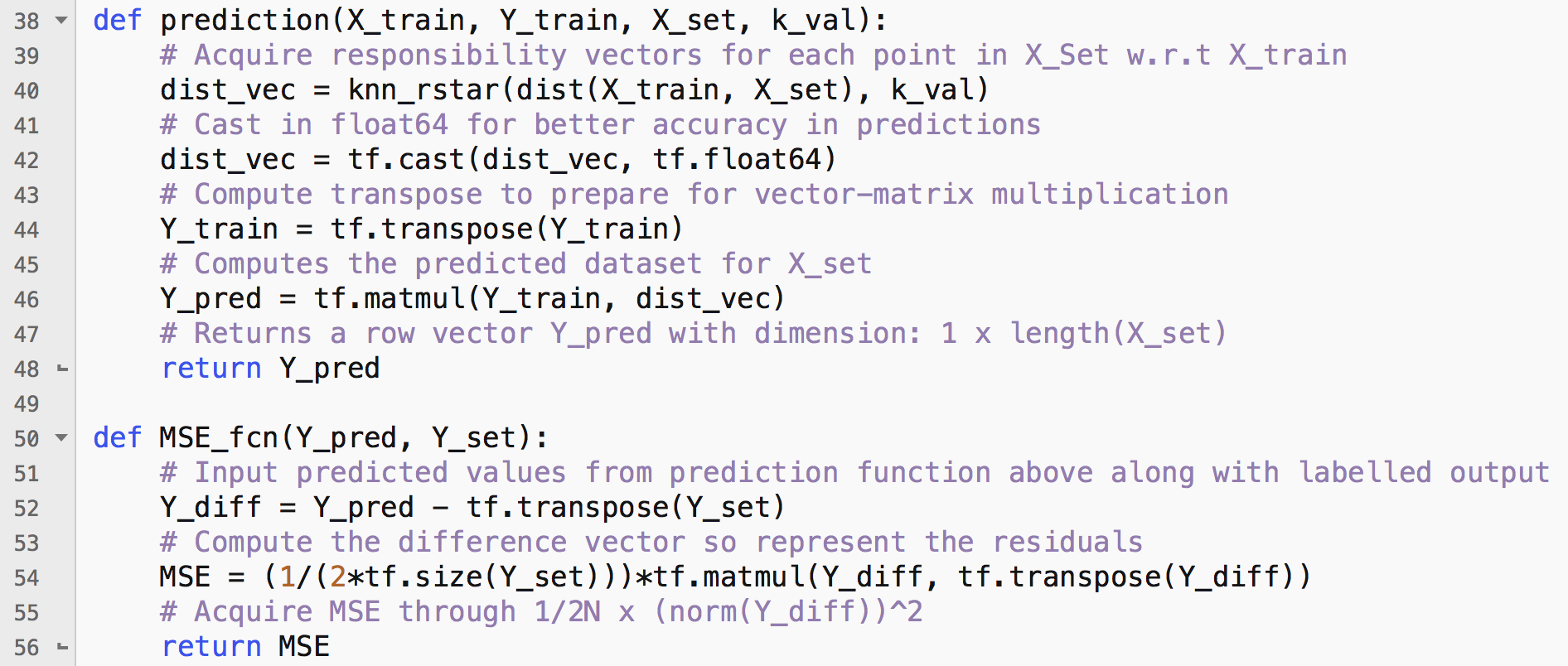
## **2.2 Choosing the nearest neighbours**

The function knn\_rstar is used in the calculation of MSE value for each of these pairs datasets:

1. Training vs (**Predicted)** **Training**
2. Training vs **Valid**
3. Training vs **Test**

The responsibility vectors are generated for the latter set (highlighted in bold) using the “prediction” and “MSE\_fcn” functions. Applying supplied datasets, the following results yield:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **k** | **1** | **3** | **5** | **50** |
| **Training MSE** | 0.0 | 0.10646488943195342 | 0.1210685023615439 | 1.245985396492448 |
| **Valid MSE** | 0.2715496697215269 | 0.32437628398600493 | 0.31669897837510275 | 1.2287016697580737 |
| **Test MSE** | 0.13943234651568068 | 0.15879689754963655 | 0.18509598110335512 | 0.7026316020129829 |



**2.2.1 Best “k” Using Validation Error**

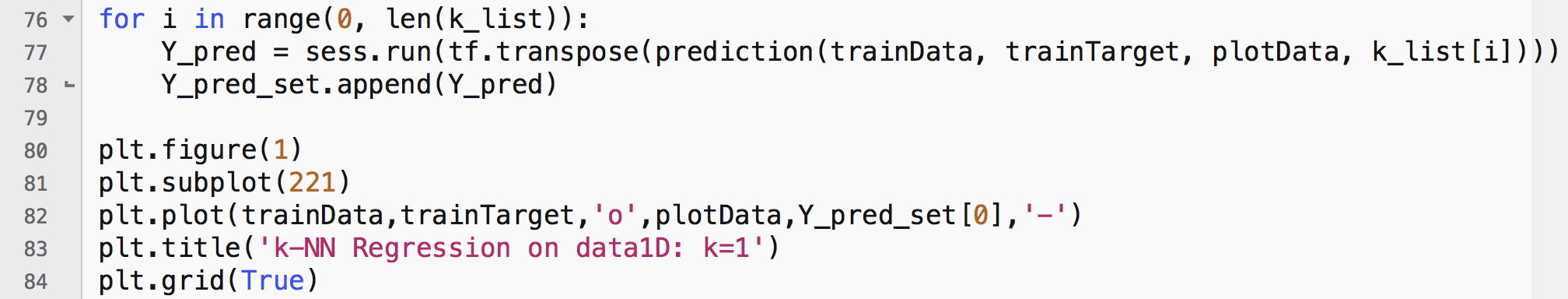
Calculated values show that with *a single nearest data (k = 1), it generates the lowest MSE for the list of k values tested*.

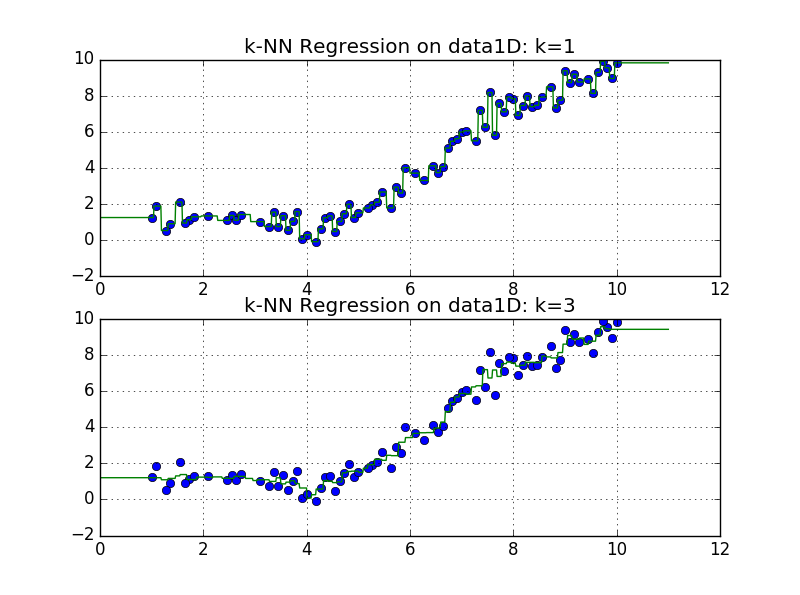
When we further investigate the sequence of MSE values inside the validation set, a local minimum is observed at k = 5. This is potentially the best fit to the data in a stochastic environment as it will achieve a reasonable balance between the accuracy of predictions and tolerance of noise. The k = 1 MSE value, although the lowest, is ***potentially more prone to sudden or dramatic changes in the labelled data which may occur as outliers or unwanted perturbations inside the dataset***. Such noisy variations can be minimized for the predicted dataset by increasing the value of k to an optimal value. Over-averaging, however, with large k values will result in reduction to capture accuracy of changes inside datasets.

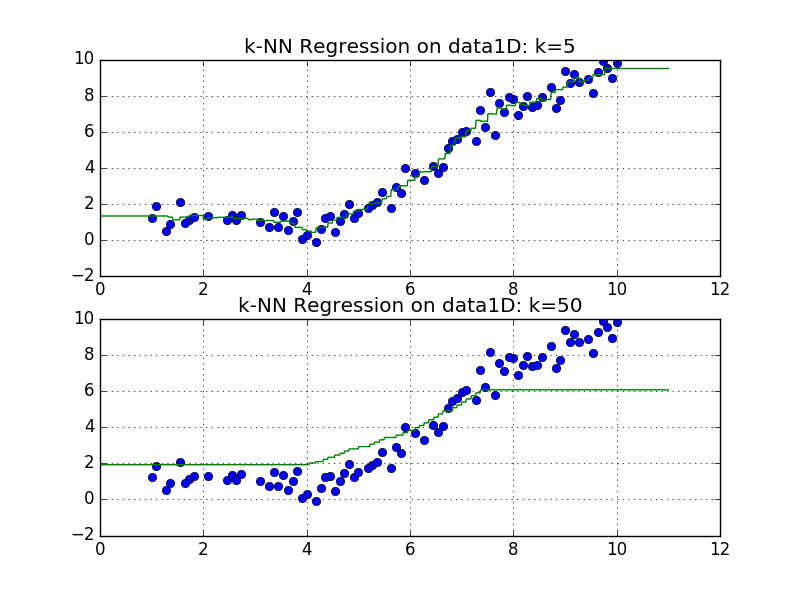
***Therefore, by accounting for accuracy and noise tolerance, k = 5 is an optimal value to use within k-NN algorithm for the given validation dataset***.

**2.2.2 Prediction**

For k = {1,3,5,50}, the prediction function will be used to predict the unknown output values for the dataset X = np.linspace(0.0, 11.0, num = 1000)[:, np.newaxis] using labelled training data. This generates the plots below with following code snippet (graphing shown for one plot):







**Figure Dataset Generation**

The input data is generated using X = np.linspace(0.0, 11.0, num = 1000)[:, np.newaxis]. This is compared against the set of labelled training data used in 2.2.1. **Y\_pred** is the vector output of a given value containing the k-NN predicted outputs. This is then appended to a 2D matrix, **Y\_pred\_set**, containing such sets for k = {1,3,5,50}. ***(X, Y\_pred\_set[i]) is shown in a green regression line with original training dataset shown in blue scatter plots***.

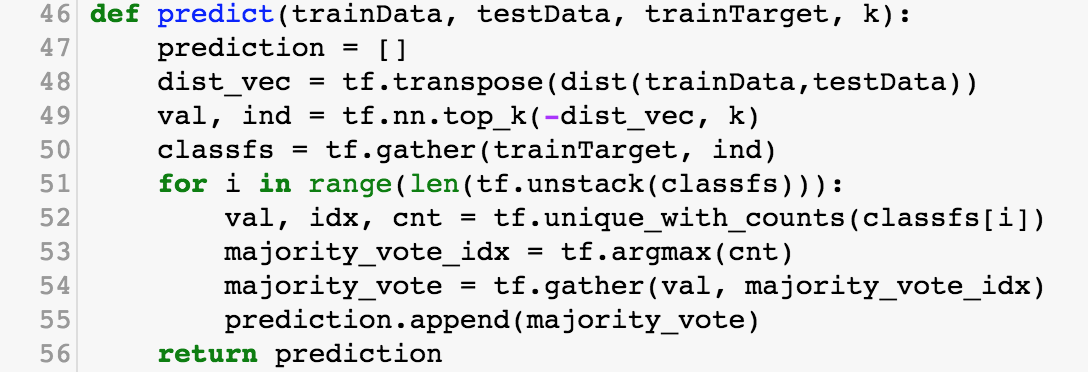
**Comments on Graphs**

Observing the above plotted graphs, the *trade-off between accuracy in representation and tolerance of noisy data is evidently displayed*. For k = 1 and k = 2, there are sudden rises and drops in the regression line. This shows that although we can capture a higher accuracy of data variations, inherently, if there are huge noise variations, they cannot be mitigated.

The k = 5 scenario seems to be good choice if one requires *a* ***relatively smooth regression with good noise tolerance***. Similar to part 2.2.1, with k too large, the accuracy in prediction will significantly decrease as shown in k = 50 case. ***This means that the best k-value to choose is the k = 5 case***.

# **3 Making Predictions for Classification**

## **3.1 Predicting class label**



|  |  |
| --- | --- |
| **Line** | **Description** |
| 48 | Transpose the pairwise distance matrix to find top k values *for one test point* across all *training points.* This is because tf.nn.top\_k() function goes across the last dimension of the input matrix to find the largest k values. |
| 49 | Get values and indices of top k values using tf.nn.top\_k() function with the negated pairwise distance function as an input in order to get the nearest distances. |
| 50 | Get target values according to the *indices* from tf.nn.top\_k() function by using tf.gather() function. |
| 51 | Because tf.unique\_with\_counts() takes in 1D tensors as an argument, the classfs must be separated into 1D tensors. The tf.unstack() call is used to do this. |
| 52 | Get *values, indices, counts* of each *target value* which is returned from the line 50. This will get the frequency of each classification given from the nearest neighbours. |
| 53 | Get indices of the largest value among *count* values from tf.unique\_with\_counts by using tf.argmax() function. The indices we get from this line are later used to locate the classification prediction among *values* from line 52 since the *values* and *counts* are corresponsive. |
| 54 | Using tf.gather() function and *indices* from tf.argmax() function, get the *classification label* with majority votes. |

## **3.2 Face Recognition Using k-NN**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **k** | **1** | **5** | **10** | **25** | **50** | **100** | **200** |
| Accuracy Over Validation Set | 66.304% | 60.870% | 57.609% | 59.783% | 57.609% | 47.826% | 31.522% |

Therefore, the k value that has the highest accuracy is k=1. Its accuracy over the test set is **70.968%.**

**3.3 Gender Recognition Using k-NN**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **k** | **1** | **5** | **10** | **25** | **50** | **100** | **200** |
| Accuracy Over Validation Set | 91.304% | 91.304% | 89.130% | 90.217% | 89.130% | 85.870% | 78.261% |

Therefore, the k value that has the highest accuracy is k=1. Its accuracy over the test set is **92.473%**.

**3.4 Analysis**

The test image in Figure 3.4.4 failed on both face and gender classification. Looking at the images in Figure 3.4.4 and Figure 3.4.5 below, we can see that the nearest neighbours are the same classification as the test image (ie. Gerard Butler, and male). However, the proceeding images are not the same classification in both gender and face/person. This may indicate that there were either not enough training data points or the k-value was too high, since the nearest neighbours were the correct classification, but they were not enough to be the majority and so the next nearest neighbours of a different classification influenced the prediction. The pairwise distance between this test image and its 10 nearest neighbours is as follows:

Table 3.4.1: Pairwise distance between a test image and its 10 nearest neighbours.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 25.057 | 29.654 | 32.184 | 34.767 | 34.810 | 35.321 | 36.220 | 36.469 | 37.188 | 38.198 |

The following are the average pairwise distances for correctly classified test images and incorrectly classified test images, respectively:

Table 3.4.2: Average pairwise distance between all correctly classified test images and their 10 nearest neighbours.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 19.969 | 25.585 | 27.945 | 29.581 | 30.884 | 31.668 | 32.319 | 33.049 | 33.668 | 34.310 |

Table 3.4.3: Average pairwise distance between all incorrectly classified test images and their 10 nearest neighbours.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24.613 | 30.600 | 31.911 | 33.282 | 33.905 | 34.469 | 35.344 | 36.170 | 36.723 | 37.213 |

Therefore, it can be seen that the highlighted test image here has 9 out of 10 pairwise distance values that are larger than the average of the incorrectly classified test images. This indicates that more data was needed (ie. more likely for training images to be closer to this test image) for this test image to be classified correctly.

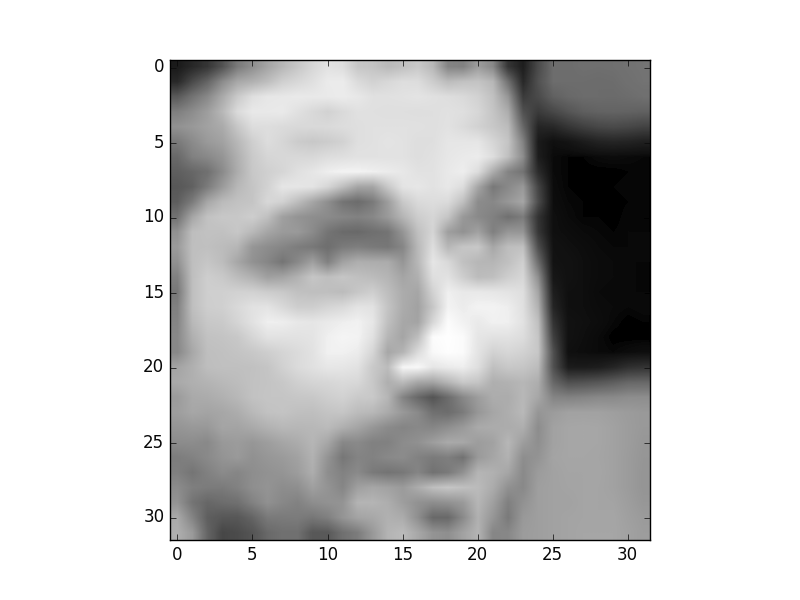
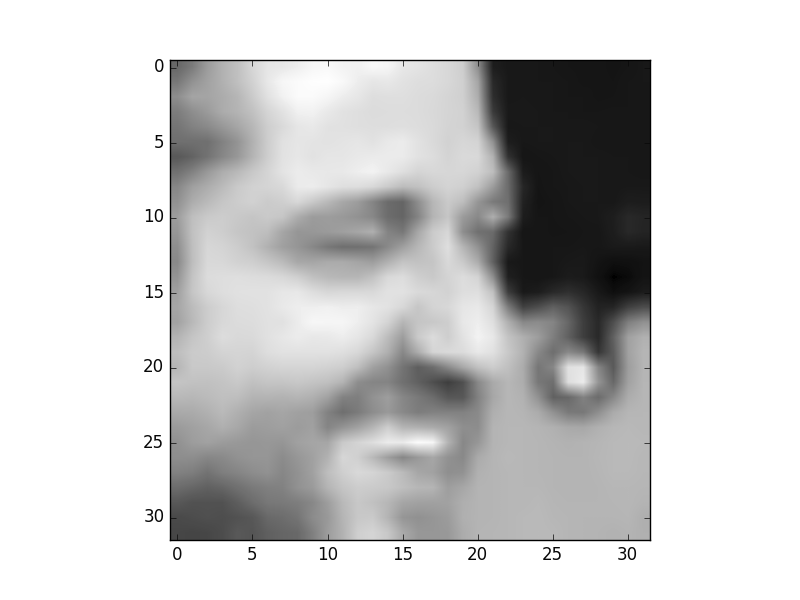
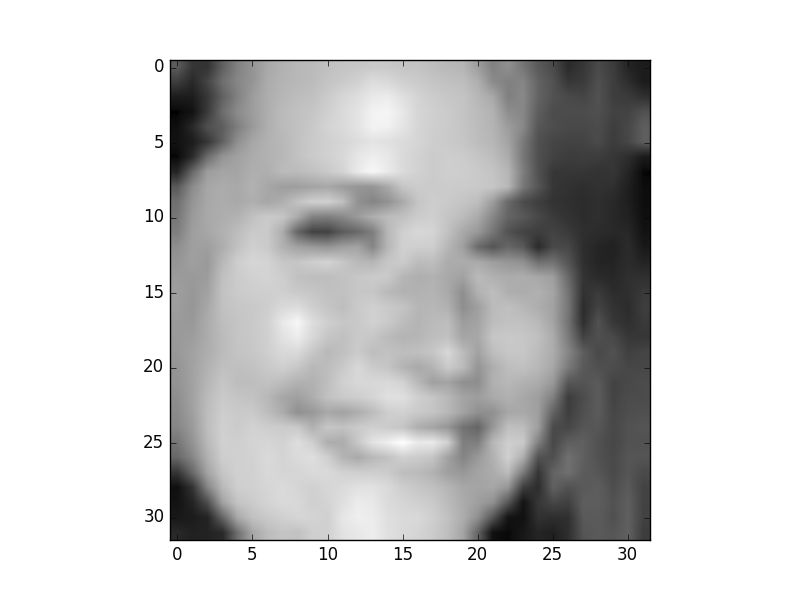
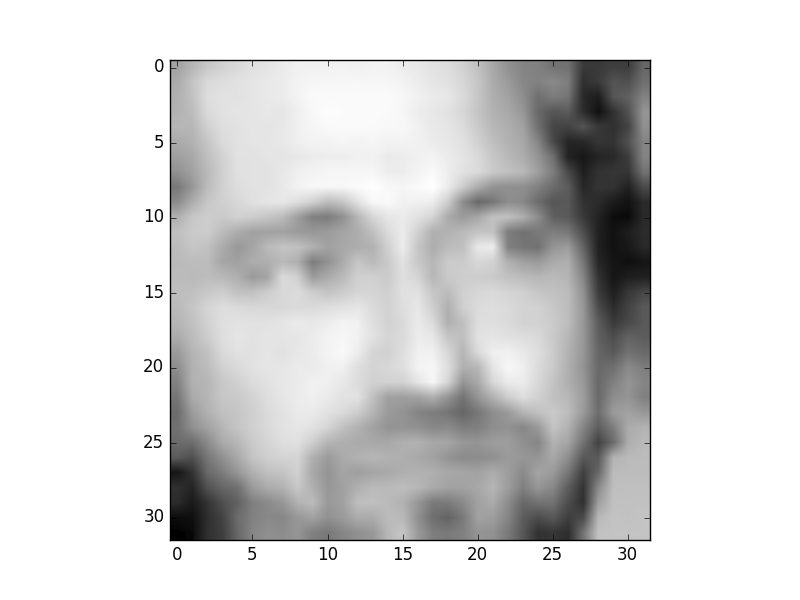


Figure 3.4.4: One of the mispredicted test pictures.



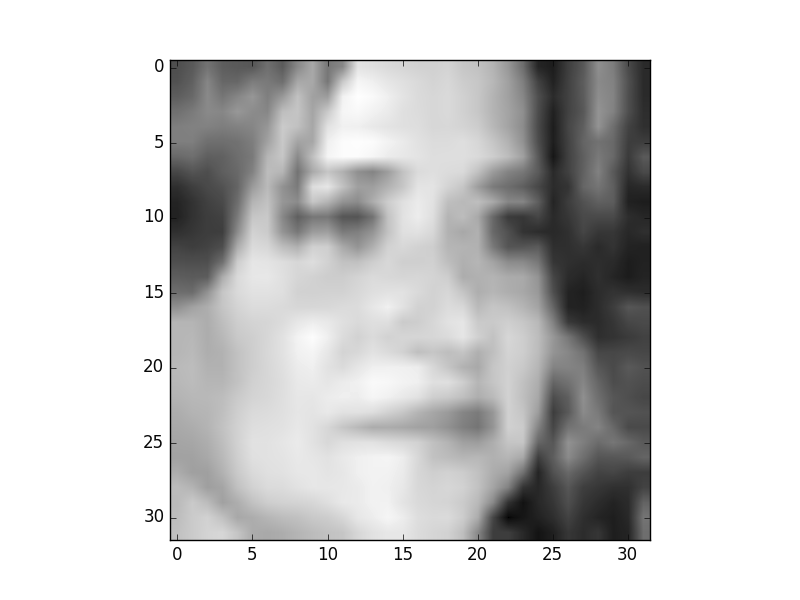
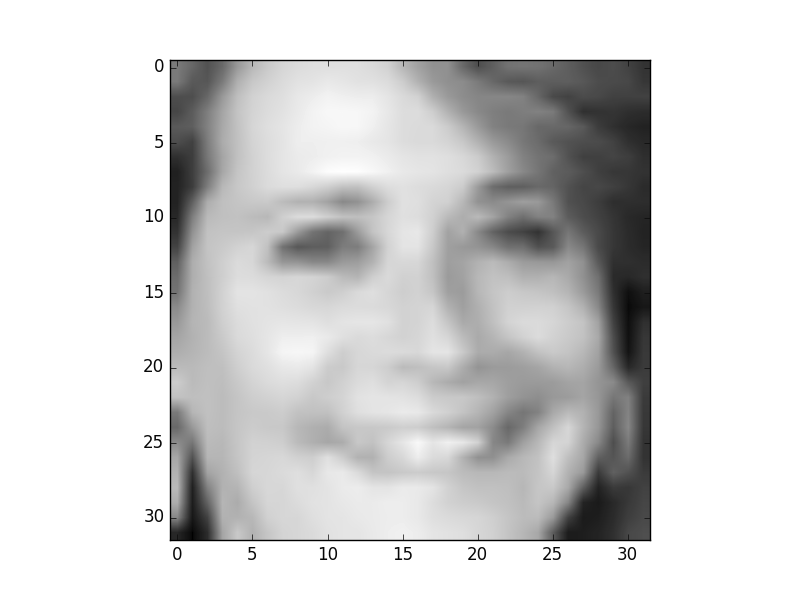
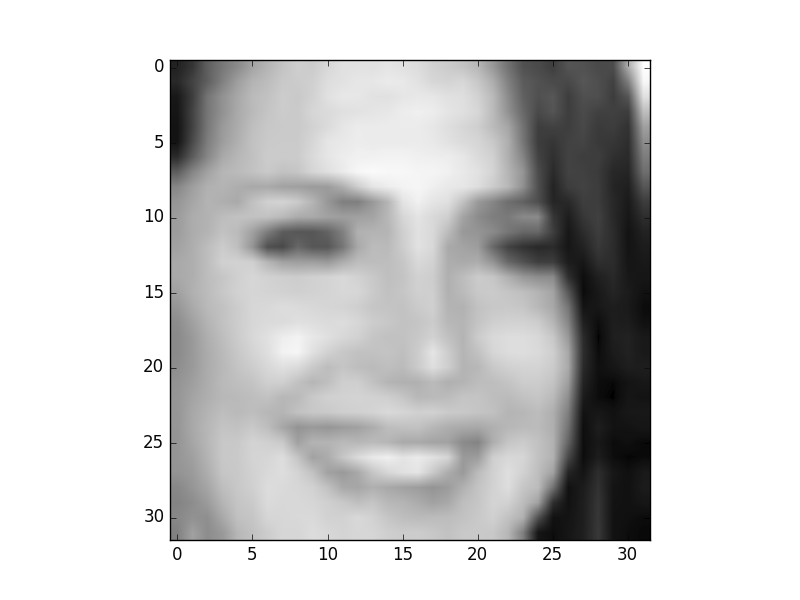
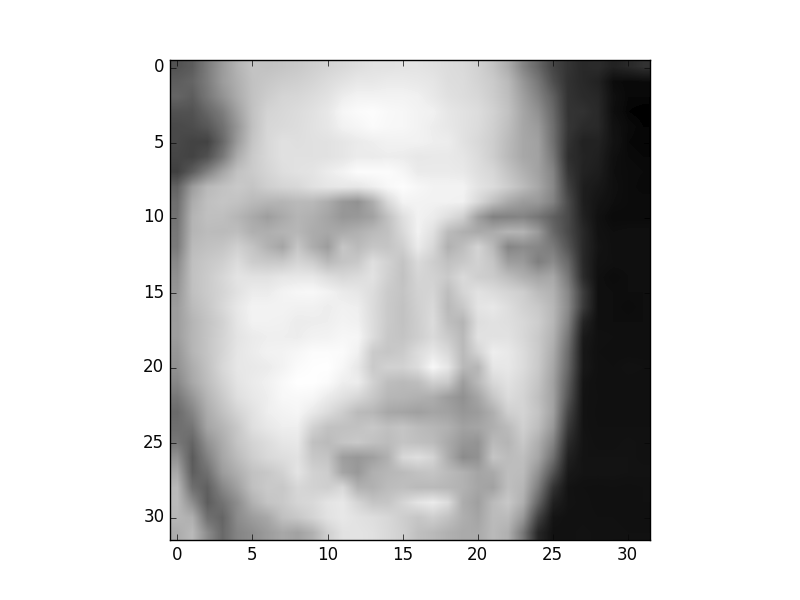


Figure 3.4.5: 10 nearest neighbours to the mispredicted image in Figure 3.3.1. Traversing left-to-right, top-to-bottom, we get farther neighbours (ie. top left is the nearest neighbour).