Machine Learning (CSCI 335)

Programming Assignment 1

Linear and Nearest-Neighbor Classification Writeup

Lynlee Hong

a. Briefly describe your implementation of the least squares and k-nn classification models (1-2 paragraphs each)

The least squares classifier uses a linear model to classify input data. In addition to the given starter code for *linear_classifier(weight_vector),* I implemented a helper function *find_optimal_weights(input_matrix),* which is responsible for computing the optimal weights. It uses the equation:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

After optimal weights are computed, *linear_classifier(weight_vector)* uses the computed weight vectors to classify the input data. The function returns an N x 2 result array of class scores, which is calculated from the dot product of the input data and the weight vector, and the predicted class.

So, the main execution flow first uses *find_optimal_weights()* to compute the weight vector. The weight vector is passed onto *linear_classifier()* to create the classifier, and the classifier uses this data to predict the class labels for the training data. The results are evaluated using a confusion matrix. This generates a linear decision boundary, which can be seen in the output *ls.pdf*

The knn classifier performs well where the linear classifier cannot. It computes the Euclidean distance between a test point and all training data points. *knn_classifier(k, data_matrix)* identifies the *k* nearest neighbors and assigns the majority class among them to the test point. Smaller values of *k* will give a more specified decision boundary, as seen in the output *knn-1.pdf*. Larger values will smooth the boundary but may not take enough data into account *(knn-15.pdf)*.

b. Briefly describe your implementation for drawing decision boundaries (1-2 paragraphs)

The decision boundary implementation for drawing decision boundaries is unified across all classifier types. Class predictions are converted into a grid that matches the space created by variables xx and yy. Then, it checks where the predicted class changes both horizontally and vertically. This determines the decision boundary between the two classes.

The implementation then draws small lines where the transitions occur horizontally and vertically. The vertical boundaries are drawn by averaging the x-coordinates where the class change happens, and horizontal boundaries are drawn by averaging the y-coordinates. Since the lines are drawn both horizontally and vertically, the decision boundary may not seem smooth to the viewer. I increased the *axis_tick_count* in hopes of a clearer figure, but I found that increasing this too much would cause the program to take longer to generate the figures.

c. Include your confusion matrices and classification diagrams in your report, and comment on the observed differences in decision boundaries and classification accuracy/errors for the training data.

```
>>>  Data vs. Itself Sanity Check

    Recognition rate (correct / inputs):
     100.0 %

    Confusion Matrix:
                0: Blue-True  1: Org-True
--------------------------------------
0: Blue-Pred |         100           0
1: Org-Pred  |           0         100
```

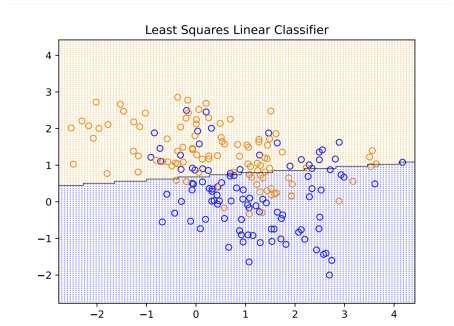
This shows the perfect matrix. When data is compared with itself, it identifies all data points without any errors, as expected.

```
>>>  Least Squares

    Recognition rate (correct / inputs):
     73.0 %

    Confusion Matrix:
                0: Blue-True  1: Org-True
--------------------------------------
0: Blue-Pred |          70          24
1: Org-Pred  |          30          76

Wrote image file: ls.pdf
```



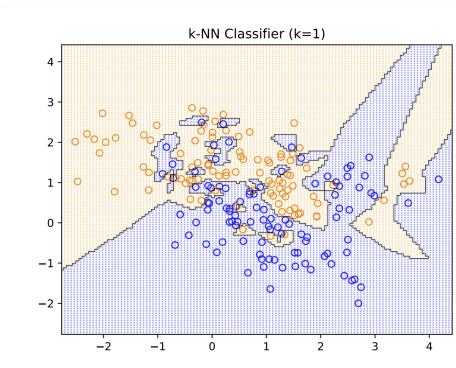
The least squares classifier has a recognition rate of 73%. This matrix shows that the classifier correctly identified 70 blue points and 76 orange points but made some errors: it classifies 24 blue points as orange and 30 orange points as blue. This shows that the linear boundary may not perfectly separate the classes. From the classification diagram for least squares, the recognition rate makes sense as there are blue points on the side classified as orange and orange points on the side classified as blue. Together, these two can show where a linear decision boundary has its limitations.

```
>>>  knn: k=1

    Recognition rate (correct / inputs):
     100.0 %

    Confusion Matrix:
                0: Blue-True  1: Org-True
--------------------------------------
0: Blue-Pred |         100           0
1: Org-Pred  |           0         100

Wrote image file: knn-1.pdf
```



For k=1, the k-NN classifier has a perfect recognition rate of 100%. This means it correctly classified all 100 blue points and all 100 orange points, as shown in the confusion matrix. This perfect result shows that the classifier perfectly fits the training data when k=1, but it also indicates a potential risk of overfitting. k=1 only considers the closest point for classification, which could lead to less generalization on unseen data.

Comparing it with the classification diagram, we can see why the classifier achieved perfect accuracy. The decision boundary is complex and follows the individual data points closely. Each point is classified according to its nearest neighbor, causing highly localized areas. The classifier basically memorizes the training data, resulting in the 100% recognition rate.

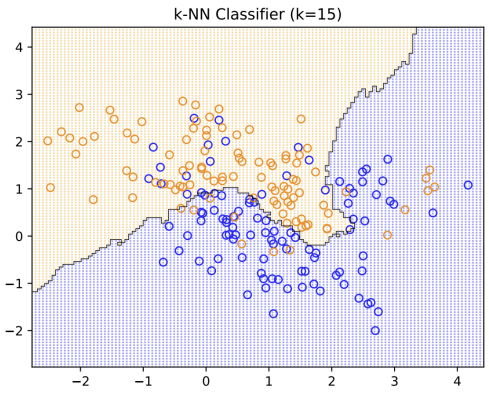
However, while perfect for the training data, this complex boundary may not generalize well to new data. If there are any small changes in the data, the classifier might incorrectly classify points because of its overfitting to the specific arrangement of the training points.

```
>>>  knn: k=15

    Recognition rate (correct / inputs):
     84.5 %

    Confusion Matrix:
                0: Blue-True  1: Org-True
---------------------------------------
0: Blue-Pred |          82           13
1: Org-Pred  |          18           87

Wrote image file: knn-15.pdf
```



For k=15, the k-NN classifier achieves a recognition rate of 84.5%, as seen in the confusion matrix. The classifier correctly identified 82 blue points and 87 orange points. However, it made some mistakes, classifying 13 blue points as orange and 18 orange points as blue. This indicates that with k=15, the classifier is less sensitive to small variations in the data compared to k=1, but it still makes a fair number of errors.

The classification diagram shows that the decision boundary for k=15 is much smoother and less complex compared to k=1. This smoother boundary shows that the classifier is considering a larger number of neighbors when making predictions, leading to more generalized decision regions. This helps reduce overfitting but also means that the classifier may struggle to correctly classify points near the boundary. However, it is more accurate than the linear classifier. Unlike k=1 this complex boundary may generalize better to new data but there is the concern of underfitting, where the classifier may miss important patterns in the data.