HW3 Write-up

Elias Crum

Preprocessing:
I chose to use a wine dataset for HW3 analysis. Loading the wine dataset was done with sci-kit learn using the `datasets.load_wine()` command. The data values were standardized using the sklearn `StandardScalar()` method. The data was then split into 85% training and 15% testing data, then the training data (85% of original data) was split into 70% training and 30% development data (this gives roughly a 70%, 15%, 15% split), all using the `train_test_split()` sklearn method.

I thought accuracy would work fine as the evaluation metric because the wine data set has 178 total instances, of these n[0] = 59, n[1] = 71, and n[3] = 48. Since all classifications are fairly well represented within the dataset (i.e. there is not one label with only 5 or 10 instances), the data is fairly balanced and accuracy should be adequate for model evaluation.

1. Using the Support Vector Machines implementation in sklearn, I trained a default model on the wine training data using the method `LogisticRegression()`. The default hyperpaprameters were used by only inputting `LogisticRegression(randomstate=0)` to make sure the same model is used each time. When evaluated against the development data, the classifier was 94.7% accurate.

2. To tweak the default SVM classifier, I first changed the C value. When the C value was increased, even very high, there was no change in model classification accuracy. By contrast, when the C value was decreased to 0.001, the accuracy dropped to 0.158. When the C value with slightly decreased to 0.1, the accuracy decreased to 0.921. Interestingly, when the C value was decreased to 0.01, the accuracy returned to 0.947. When the gamma value was altered, there was no observed change in accuracy, both with large and small changes. The model I chose to use moving forward was the one with C=1.0 and gamma=0.001. I moved the C value up and down because the C value controls the size of the margin on either side of the hyperplane used to separate the data points and assign label designations. The larger the C value, the smaller the margin while the smaller the C value, the larger the margin. I observed that the misclassification error increased as the C value decreased and stayed the same as the C value increased. This suggests the wine data is almost totally linearly separable and that shrinking the margin does little to impact the accuracy of the SVM model. I settled on using a C value of 1 because it provided the largest accuracy while not venturing into the high C values that could lead to over-fitting. Changes to the gamma value, both up and down, did not change the accuracy, likely because the data is almost entirely linearly separable the changing of the curvature of the hyperplane (which is essentially what gamma controls) was not impactful.

3. The KNN algorithm I developed is very simple and has 4 parameters. The training data, training data labels, query, and K value. The query is the row (also can be thought of as a data point) of a dataset being tested. The rows being assessed by the KNN algorithm are those of the

development dataset. These rows are inputted into the KNN algorithm recursively (so all rows will act as query once) and the KNN algorithm makes predictions about the label of these rows based on their relationship to the training data labels and the designated k value.

The algorithm works by first calculating the Euclidean distance between the inputted query (development data row) and a row of the training dataset via the `euclidian_distance(row1, row2)` method which returns a value representing the distance between the two points. The query is compared to every row of the training dataset separately. During each iteration, the calculated distance between the query row and training data row as well as the index of training data row is appended to the `distances` list. This loop recursively runs until all rows of the training data have distances recorded in the `distances` list. The `distances` list is then sorted using the `sorted()` method so that the rows with the shortest distances (most closely related to the query) are in the front of the list. The algorithm then takes the first k terms of the `sorted_distances` list. Using a for loop, the index associated with the first k terms of the `sorted_distances` list are used append the label of these points to a list named `k_nearest_labels`. The `Counter()` method then determines the most common label of the most closely related training rows. The most common label is returned as the prediction for that query. The prediction is appended to the `predicted_labs` list. Once all rows of the development dataset have been used as the query, the `predicted_labs` list is compared to the `y_dev` list of development dataset labels and accuracy is calculated.

The accuracy of the KNN algorithm with a k of 5 was 0.895 on the development dataset, if the k was decreased the accuracy did not change. When the k value was increased, the accuracy did not change until a k value of 10, where the accuracy decreased to 0.868. Thus, the k-value with the best performance for the development data was between 5 and 9. I chose to use a k value of 9 for the test data. With a small k value, the influence of noise (or points that do not cleanly fit into a classification category) is larger which can impact accuracy. While with a large k value, the model can be under-fit due to the decrease in decision impact of the closest neighbors. Thus, because the wine dataset has fairly defined linear boundaries between classes (as seen in the SVM optimization), I chose a k value at the top end of the range that contained the highest accuracy on the development data; k=9.

4. Using the DummyClassifier implementation in sklearn, the baseline models were trained using the 'most_frequent' and 'stratified' strategies. The 'most_frequent' model, `DummyClassifier(strategy="most_frequent")`, exhibited an accuracy of 0.132 on the verification data, and the 'stratified' model, `DummyClassifier(strategy="stratified")`, exhibited an accuracy of 0.368. The "most_frequent" classifier simply predicts the most common class within the dataset for every row. The "stratified" classified predicts a random label based on the distribution probabilities of each class in the dataset. These accuracy values are very very low and act as controls to ensure our models are working correctly.

5. Compare model in #2 to KNN via evaluating on test set (Include the performance of your baseline systems from step (4) for comparison.)

Test data baseline 'most_frequent' accuracy: 0.241
Test data baseline 'stratified' accuracy: 0.296
Test altered SVM accuracy: 0.981
Test KNN accuracy: 0.926

The SVM accuracy was slightly higher than the KNN accuracy during testing but both models performed substantially better than the baseline models. The KNN model likely performed worse because it is a much simpler algorithm that 'memorizes' the training data and makes predictions based on input similarity opposed to the SVM algorithm which is much more powerful because of its ability to predict based on complex hyperplanes established between classes during training.