Project 2: K-Means vs. Hierarchical Agglomerative Clustering

Nazanin Yari
Johns Hopkins University

Introduction to Machine Learning - Section 8VL

Solution function

Sequential Forward Selection: This algorithm as one of the feature selection techniques is based on greedy search algorithms used to reduce the dimensionality of the dataset.

We first start with storing all the features in a list called `remaining_features` and initializing an empty list `selected_features` to add the best features to. Then a while loop used to compute the errors inside the function `sequential_forward(dataset)` and return the error list and selected features. We first find the errors for each single feature using Naive Bayes' algorithm we have already programmed in Project 1. The feature with the lowest error is then added to `selected_features` and removed from `remaining_features`. For the remaining features, we slice the dataset to the previously selected features as well as a new single feature from `remaining_features`. This subset of the original dataset is then used to train the Naive Bayes' classifier. We use 67% of the data to train, and the other 33% to test the algorithm. The loop terminates when the `remaining_features` list becomes empty.

K-Means: We used the below steps to program K-Means algorithm:

Step 1 - Choosing Initial Centroids: Instead of choosing one of the data points at random as a centroid for each cluster, we used a more sophisticated approach to choose the best available centroids. It is an algorithm called K-Means++ which was proposed in 2007 by David Arthur and Sergei Vassilvitskii, as an approximation algorithm for this K-Means problem. This algorithm avoids the poor clustering sometimes results from choosing random centroids. [1] The function `initial_centroids(k, dataset)` takes the given k number of clusters and the dataset and returns a $k\ x\ number\_of\_features$ matrix. The function first assigns a random data point as the first centroid. Then, to compute the rest of the $(k-1)$ centroids, for each data point, it computes its distance from the closest centroid. Then it selects the data point with maximum distance from the nearest centroid as the next centroid. It repeats the last 2 steps until all the k centroids have been selected.

Step 2 - Assigning Clusters to its Closest Centroid : For this step, the function `assign_clusters(centroids, dataset)` is used. It takes the centroids and the dataset and returns a $n\ x\ 1$ matrix, where n is the number of observations. For each data point, it first computes the distance from each centroid, and then adds the index of the data point to the closest cluster.

Step 3 - Updating the Centroids: After assigning the clusters, the program recalculates the centroids of the new clusters using `compute_centroids(k, clusters, dataset)` by taking the mean of all the data points in each cluster. If the cluster size is zero, then it randomly assigns a data point as the centroid of that cluster.

Step 4 - Repeating Steps 2 and 3: The program repeats the previous 2 steps until the centroids do not change. It uses a while loop inside the function `final_clusters(k, dataset)` which takes k number of clusters and the dataset, and returns the final cluster, final centroids, and the number of iterations took to converge.

[1] "k-means++ - Wikipedia." https://en.wikipedia.org/wiki/K-means%2B%2B. Accessed 10 Oct. 2020.

The function `k_mean(k, dataset)` which takes k number of clusters, and the dataset return the combined dataset by adding the calculated clusters in the last column. It assigns the name of the predominant class to the cluster.

To evaluate how well the algorithm works as a classifier, `k_mean_error(final_combined_data)` compares the actual classes to the classes computed through K-Means and return the error percentage as the sum of the number of data points with the predicted cluster different from the actual cluster over the total number of data points.

Hierarchical Agglomerative Clustering (HAC) with Single linkage function: The HAC is a clustering function to identify similar groups of data in a data set which uses the bottom-up approaches. The algorithm starts with initializing the `clusters_dict`, a dictionary which stores the index of the clusters as dictionary keys and the index of the data points as dictionary values in n single clusters, where n is the number of observations. Then, `distance_dict` has been initialized to store all the possible combinations of each 2 data points indices in a tuple as dictionary keys, and the Euclidean distance between the 2 data points as dictionary values. A while loop is then constructed to find the key of the minimum distance (value) in `distance_dict`, and add the tuple index (key) as new value to the `clusters_dict` with a new key which is 1 larger than the maximum existing key. For example, for a dataset with 11 data points that I used to test the program, the initial `clusters_dict` was:

$$\{0:[0],\ 1:[1],\ 2:[2],\ 3:[3],\ 4:[4],\ 5:[5],\ 6:[6],\ 7:[7],\ 8:[8],\ 9:[9],\ 10:[10]\}$$

The initial `distance_dict` was:

$$\{(1,0):2.0,\ (2,0):1.41,\ (2,1):1.41,\ (3,0):2.83,\ (3,1):2.0,\ (3,2):1.41,\ (4,0):6.0,\ (4,1):6.32,\\ (4,2):5.1,\ (4,3):4.47,\ (5,0):8.0,\ (5,1):8.24,...,(10,\ 7):\ 15.0,\ (10,\ 8):\ 11.18,\ (10,\ 9):\ 10.44\}$$

The keys of the `distance_dict` are the indices of a symmetric lower triangular matrix with size $n\,x\,n$, where n is the number of observations. The minimum value or distance in `distance_dict` is 1.41 with key $(2,0)$. This means we need to create a new cluster with data point indices $[2,0]$ and add it to `clusters_dict` with a new key and remove the two single clusters $[0]$ and $[2]$. The `clusters_dict` would be:

$$\{1:[1],\ 3:[3],\ 4:[4],\ 5:[5],\ 6:[6],\ 7:[7],\ 8:[8],\ 9:[9],\ 10:[10]\},\ 11:[0,2]\}$$

Assuming the triangular matrix storing the distances, when $[0],[2]$ are merged, and replaced with only on cluster $[0,2]$, then the associated rows and columns should be eliminated from the matrix and a new row and column added. This means we need to remove all the entries which includes 0 and 2 in their keys from `clusters_dict` and add the distances of all the possible combinations of key 11 (related to the cluster $[2,\ 0]$) with existing keys; 1 through 10, in the dictionary. To calculate the distance between 2 clusters when the cluster size is greater than 1, the function `cluster_distance(dataset, C1, C2)` has been created. This function uses a single linkage approach which measures the Euclidean distance between the closest members of the two clusters. The program exits the loop when it reaches the `k` number of clusters.

The silhouette coefficient calculated using the function `silhouette(dataset, clusters_dict)`. The function takes the dataset and the `clusters_dict` and returns the average silhouette coefficient as well as a test coefficient which uses sklearn built-in function to validate the answer. The function creates two matrices. First, a matrix $n\,x\,n$, where n is the number of observations, called `distance_matrix` stores all the distances between any 2 data points. It is similar to the triangular matrix already mentioned, but instead it is a full symmetric matrix. The second matrix is a $n \, x \, k$ matrix where k is the number of clusters. It is called `cls_mask_matrix` which stores cluster membership masks. Instead of using 1 for cluster membership, we divided the numbers by the cluster size, so the dot product produces a_i's and the values for b_i's. Choosing the minimum number in each row of the new matrix would create the b_i vector. Please note that in order to be able to compare the coefficient calculated using our program and the one from sklearn, we followed the general function used for silhouette where for a_i's, instead of dividing by cluster size, we divided the delta by $(cluster\ size\ - \ 1)$. Also, the $s_i = 0$ where the size of the cluster is 1.[2] The clustering of the data then evaluated by the average of the s_i's.

What We Know About the Datasets

Preprocessing Datasets: There was no missing data found in the datasets. The mean and covariance for datasets are calculated using `data_stats(dataset)` function in reading_data.py file.
Below is the mean and covariance of the synthetic dataset per class where the keys represent classes and the values are means:

```
{0.0: array([0.98286273, 4.01281616, 2.18858511, 6.99351062, 0.46997995]), 1.0: array([ 4.00617298,
        0.97958118,  7.49743591, -2.00798054,  5.03176237])}
```

Below shows the Synthetic Dataset covariance per class, where the keys represent classes and the values are covariances:

```
{0.0: array([[ 9.89876705e-01,  7.04548402e-01, -1.83775662e-02, 6.91017209e-01,  5.05432201e-01],
        [ 7.04548402e-01,  2.01918397e+00, -1.87520929e-03,1.01173323e+00,  5.62769205e-02],
        [-1.83775662e-02, -1.87520929e-03,  1.96505897e+00,7.77620217e-01,  6.42406954e-01],
        [ 6.91017209e-01,  1.01173323e+00,  7.77620217e-01,9.93431904e-01,  5.03036838e-01],
        [ 5.05432201e-01,  5.62769205e-02,  6.42406954e-01,5.03036838e-01,  2.91298982e+00]]),
1.0: array([[ 1.00698062,  0.73677077, -0.00765858,  0.71527025,  0.44119449],
        [ 0.73677077,  2.03904971,  0.0276622 ,  1.04214177, -0.03460628],
        [-0.00765858,  0.0276622 ,  2.03364389,  0.81871643,  0.71590962],
        [ 0.71527025,  1.04214177,  0.81871643,  1.03038357,  0.46692262],
        [ 0.44119449, -0.03460628,  0.71590962,  0.46692262,  2.89354161]])}
```

Below is the mean for the first class in spambase dataset:

[2] "Silhouette (clustering) - Wikipedia." https://en.wikipedia.org/wiki/Silhouette_(clustering). Accessed 10 Oct. 2020.

```
[7.34791966e-02 2.44465567e-01 2.00581062e-01 8.85939742e-04
 1.81040172e-01 4.45444763e-02 9.38307030e-03 3.84146341e-02
 3.80487805e-02 1.67170014e-01 2.17109039e-02 5.36323529e-01
 6.16642755e-02 4.24031564e-02 8.31779053e-03 7.35868006e-02
 4.83464849e-02 9.72919656e-02 1.27034075e+00 7.57890961e-03
 4.38701578e-01 4.52259684e-02 7.08751793e-03 1.71377331e-02
 8.95473458e-01 4.31994261e-01 1.26526542e+00 1.93805595e-01
 1.62794118e-01 1.65853659e-01 1.06032999e-01 7.73063128e-02
 1.50986370e-01 7.77869440e-02 1.69454806e-01 1.41671449e-01
 1.97743902e-01 1.87230990e-02 1.21678623e-01 8.31169297e-02
 7.20265423e-02 2.16807747e-01 7.05810617e-02 1.26635581e-01
 4.15760402e-01 2.87184362e-01 8.19225251e-03 5.12266858e-02
 5.02808465e-02 1.58578192e-01 2.26836442e-02 1.09983501e-01
 1.16484935e-02 2.17130560e-02 2.37730093e+00 1.82144907e+01
 1.61470947e+02]
```

And the mean for the second class:

```
[1.52338665e-01 1.64649752e-01 4.03794815e-01 1.64671815e-01
 5.13954771e-01 1.74875896e-01 2.75405405e-01 2.08141202e-01
 1.70060673e-01 3.50507446e-01 1.18433536e-01 5.49972421e-01
 1.43546608e-01 8.35741864e-02 1.12079426e-01 5.18361831e-01
 2.87506895e-01 3.19227799e-01 2.26453944e+00 2.05521236e-01
 1.38036955e+00 2.38036404e-01 2.47054606e-01 2.12879206e-01
 1.74793161e-02 9.17264203e-03 1.54991726e-03 1.87975731e-02
 6.83949255e-04 5.96800883e-03 1.27413127e-03 5.18477661e-04
 1.45615003e-02 1.77606178e-03 6.92774407e-03 2.95146167e-02
 4.34693878e-02 4.71042471e-03 1.24269167e-02 3.67181467e-02
 5.51571980e-05 2.44346387e-03 8.45008274e-03 6.24379482e-03
 1.25091009e-01 1.47269719e-02 1.21897408e-03 2.10148924e-03
 2.05730833e-02 1.08970215e-01 8.19856591e-03 5.13712631e-01
 1.74478213e-01 7.88769994e-02 9.51916492e+00 1.04393271e+02
 4.70619415e+02]
```

The covariance matrices for the spambase dataset is not shown because of the large amount of data. For the HAC algorithm, the datasets are rescaled to center at zero with variance of 1. The `standardize_data(dataset)` in the reading_data.py file used for this purpose. The output files are saved in the output_files folder.

Results

Sequential Forward Selection - Synthetic Dataset: Running the algorithm on the synthetic dataset with 5 features results in the below Bayes' error in the first run for single features:

Feature_0	Feature_1	Feature_2	Feature_3	Feature_4
0.06	0.13	0.029	0	0.086

Therefore, feature 3 with Bayes' error of 0 is selected. The remaining features are now $\{0, 1, 2, 4\}$. There is no need to proceed since no better error can be found although the program again computes the error of the combined dataset as below:

Features_3 & 0	Feature_3 & 1	Feature_3 & 2	Feature_3 & 4
0	0	0	0

The validity of feature 3 error has been confirmed using sklearn built-in function. We also ran the SFS program on iris dataset to test validity of the algorithm and as expected, petal-length and petal-width have been selected as the best features.

Sequential Forward Selection - Standardized Synthetic Dataset: Running the algorithm on the standardized version of the synthetic dataset, features 3 and 0 are selected. The below table depicts the Bayes' error in the first iteration:

Feature_0	Feature_1	Feature_2	Feature_3	Feature_4
0.5072	0.5185	0.5151	0.4997	0.5051

Therefore, feature number 3 is selected. The remaining features again are $\{0, 1, 2, 4\}$. The errors for the combination of feature 3 with the remaining features are:

Features_3 & 0	Feature_3 & 1	Feature_3 & 2	Feature_3 & 4
0.4961	0.5164	0.5158	0.5024

Features 3 and 0 have the smallest Bayes' error rate among others. This error is compared to the feature number 3 error from the previous step. Since $e_{[3]} = 0.4997 > e_{[3,0]} = 0.4961$, the selected features in this step are 0, and 3. The error rates did not decrease in the next steps, therefore the final selected features are features 0 and 3.

Sequential Forward Selection - Synthetic Dataset _ 4000 Observations: Using a subset of the synthetic data yields the same results as when we ran the program on the whole dataset. The selected feature is feature number 3 with error 0.

Sequential Forward Selection - Standardized Synthetic Dataset _ 4000 Observations: Running the SFS algorithm on the standardized version of the synthetic data with 4000 observations results in the below error rates:

Feature_0	Feature_1	Feature_2	Feature_3	Feature_4
0.517	0.5121	0.5242	0.5356	0.5090

Thus, feature 4 is selected. The remaining features are $\{0, 1, 2, 3\}$. The next step includes the errors of the combinations of feature 4 with the remaining features:

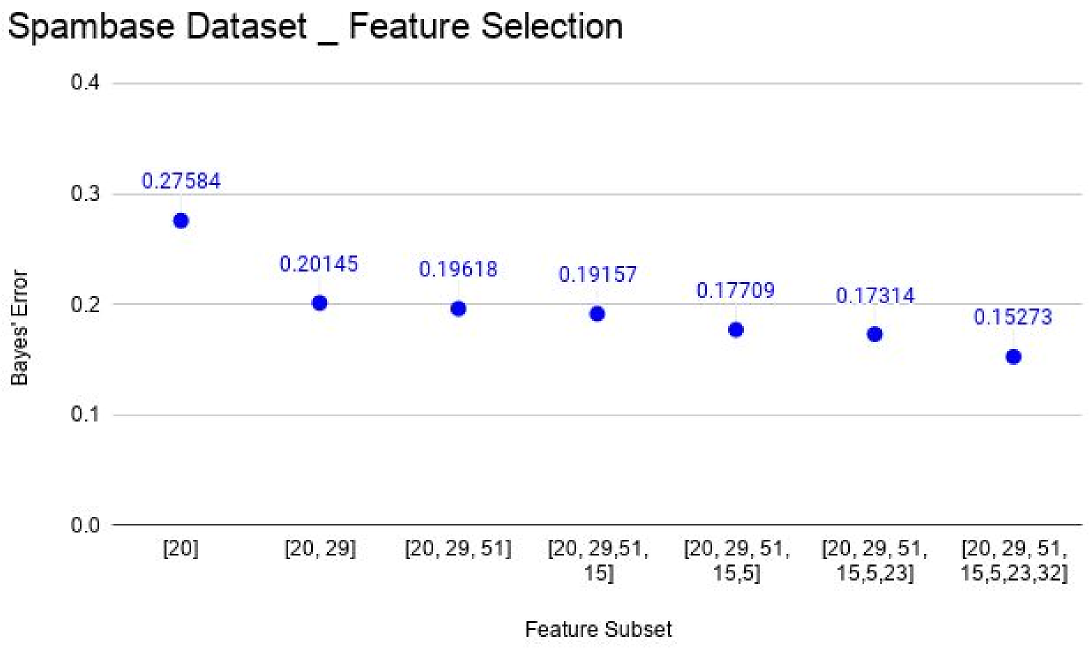
Features_4 & 0	Feature_4 & 1	Feature_4 & 2	Feature_4 & 3
0.5242	0.4985	0.5068	0.5379

The subset $[4, 1]$ has the lowest error among other rates. This rate is also lower than feature 4 error of 0.5090. Thus, the best selected features are 4, and 1. The error has not been decreased in the next steps.

Sequential Forward Selection - Spambase Dataset: The same set of tables for step-by-step error rates can not be shown for the spambase dataset because of the large number of features. The below table shows the features with the minimum rates selected in each step:

[20]	[20, 29]	[20, 29, 51]	[20, 29,51, 15]	[20, 29, 51, 15,5]	[20, 29, 51, 15,5,23]	[20, 29, 51, 15,5,23,32]
0.27584	0.20145	0.19618	0.19157	0.17709	0.17314	0.15273

The below chart shows the decreasing error rates when adding a new feature to the best features list.



Sequential Forward Selection - Standardized Spambase Dataset: Running the SFS algorithm using the standardized version of the spambase dataset, feature 46 with Bayes error of 0.3687 is the only selected feature.

K-Means - Synthetic Dataset - All features: The error rate from k-means algorithm using all features of the synthetic dataset is zero, meaning the dataset is perfectly clustered using this algorithm.

K-Means - Synthetic Dataset - Selected Features: The error rate from k-means algorithm using only the selected feature; feature number 3, results in zero error rate. This means the K-means algorithm could cluster the data perfectly using only one feature of the dataset. This was expected as the Bayes' error for this specific feature was 0, which means this feature is good enough to classify the dataset.

K-Means - Spambase Dataset - All features: Running the K-Means algorithm on the spambase dataset with all features resulted in 0.394 error. This would be probably the result of using all the features including not important ones for clustering.

K-Means - Spambase Dataset - Selected features: The error rate for the K-Means algorithm using only the 7 selected features on spambase data is 0.088. The clustering results have been significantly improved in comparison to the rate of 0.394 using all features.

HAC - Standardized Synthetic Dataset _ 4000 Observations: For the HAC algorithm, the standardized version of the datasets have been used. For the subset of the standardized synthetic data with 4000 data points, features 1 and 4 were selected as the best features using the SFS algorithm. The computed silhouette coefficient using the HAC algorithm was 0.4984560980026387. This indicates the average distance to the points within the cluster is less than the minimum average distance to the point in other clusters. This means the clusters are well separated. Please note using the suggested silhouette coefficient on the handout, the coefficient would be higher than the calculated number.

HAC - Standardized Spambase Dataset: For this dataset, feature 46 was selected as the only useful feature to predict the classification. The silhouette coefficient using this feature was -0.08189953144191124. This coefficient is very close to zero and would probably be higher than zero using the suggested coefficient formula on the project handout. Since it might take a while to run the program on your side, I have included 2 screenshots of the HAC program results which shows the above-mentioned coefficients.