Machine Learning Lab : Homework-1 Report

Vidushee Jain : 2020KUCP1014

March 25, 2023

1 Topic-A : Model selection and complexity control

This part of homework illustrates the use of resampling methods for model selection (complexity control), and for comparing prediction accuracy of a learning method.

1.1 Data Preprocessing

We began by importing the necessary libraries and loading the dataset into a pandas DataFrame. We then dropped the year_of_operation column from the dataset as it is not relevant to our prediction task.Next, we scaled the numerical variables in the dataset using the StandardScaler function from Scikit-learn. Scaling the variables ensures that each feature has equal weight in the prediction model.

1.2 Method Implementation (A1)

This code block defines a function called "KNN" that performs k-Nearest Neighbors classification on a pandas DataFrame object. The function takes three arguments: the DataFrame object to use as the training data, the value of "K" to use for the classification, and a new datapoint to classify represented as a pandas Series object.

The function first imports the "euclidean" function from the "scipy.spatial.distance" module. This function is used to calculate the Euclidean distance between two points.

The function then creates a new DataFrame object "KNN_indices" that represents the "K" nearest neighbors to the new datapoint. This is done by calculating the Euclidean distance between each row in the training DataFrame and the new datapoint using the "euclidean()" function. The resulting distances are then sorted in ascending order using the "sort_values()" method, and the first "K" indices are extracted using the "[:K]" syntax.

The function then creates a list "freq" to keep track of the frequency of the two possible classes (0 and 1) among the "K" nearest neighbors. The function then iterates over each index in "KNN_indices" and increments the corresponding value in "freq" depending on the class of the corresponding row in the training DataFrame.

Finally, the function uses an if-else statement to determine the final classification based on the frequencies of the two classes among the "K" nearest neighbors. If the frequencies are equal, then a random choice is made between 0 and 1 using the "random.choice()" function. Otherwise, the function returns 0 if the frequency of class 0 is greater than the frequency of class 1, and 1 otherwise.

1.3 Model Selection (A2)

1.3.1 LOO Validation

This code block defines a function called "LOO" that performs leave-one-out cross-validation on a pandas DataFrame object. The function takes one argument: the DataFrame object to use as the training data.

The function first creates a new DataFrame object unlabelled_df that represents the training data without the survival_status column. This column is dropped since it represents the class labels

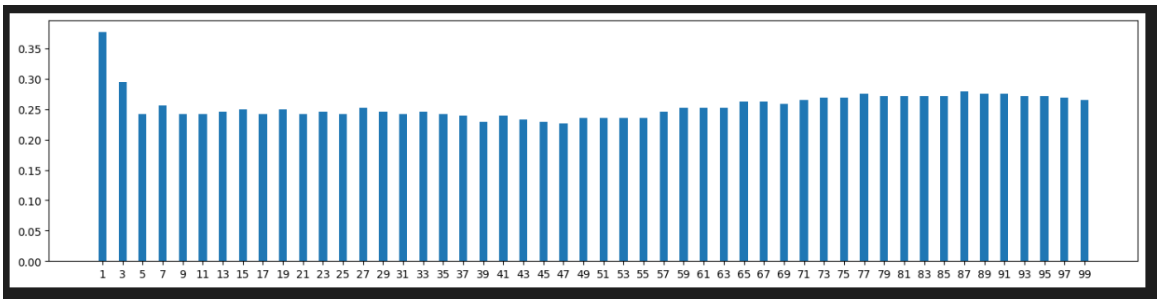
which we are trying to predict.
The function then creates an empty list "error" to store the mean squared error (MSE) for each value of "K" tested during cross-validation.

The function then iterates over each odd value of "K" from 1 to the smaller of 100 and the number of rows in the training DataFrame. For each value of "K", the function creates two lists: "actual_values" and "predicted_values". The "actual_values" list contains the true class labels for each row in the training DataFrame, while the "predicted_values" list initially contains None for each row. The function then iterates over each row in the training DataFrame and performs k-Nearest Neighbors classification on the remaining rows using the "KNN()" function defined earlier. The resulting predicted class label is stored in the "predicted_values" list for the corresponding row.

Once all rows have been classified, the function calculates the mean squared error (MSE) between the "actual_values" and "predicted_values" lists. This is done by calculating the absolute difference between each corresponding pair of actual and predicted values, summing these differences, and dividing by the number of rows in the training DataFrame minus one (since we left out one row during classification).The resulting "K" value and MSE are then appended to the "error" list.

Finally, the function returns a new DataFrame object cross_val_err that represents the cross-validation error for each value of "K" tested. The DataFrame has two columns: "K" and "error". The "K" column contains the tested values of "K", while the "error" column contains the corresponding cross-validation error.

Then the next code block uses the "matplotlib" library to create a bar chart showing the cross-validation error for each tested value of "K" in the "cross_val_err" DataFrame object generated by the "LOO()" function.



1.3.2 Finding Optimal K value

This code block finds the row with the lowest error value in the "cross_val_err" DataFrame object and extracts the value of "K" corresponding to that row.
The first line uses the "idxmin()" function to find the index of the row with the lowest error value in the "error" column of the "cross_val_err" DataFrame object.
The second line uses the "iloc[]" function to select the row with that index and extracts the value of "K" from that row. The value is then converted to an integer using the "int()" function.
The third line extracts the error value from the same row as the optimal "K" value.
Finally, the fourth and fifth lines print out the optimal value of "K" and the corresponding error value, respectively.

- Optimal value of K: 47
- Error with the optimal value: 0.2262295081967213

1.3.3 Decision Boundary

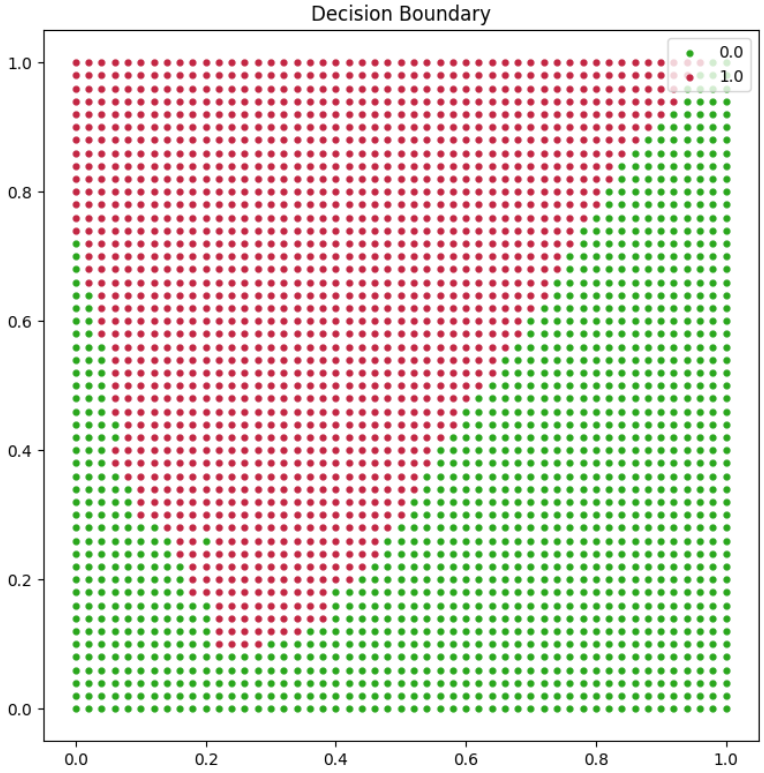
This code block generates points for a 2D grid and predicts their class using the optimal value of "K" obtained from LOO cross-validation. It then plots the predicted classes of the points on a 2D scatter plot.

The first two lines initialize an empty list "points" and set the dimension of the grid to 50.

The for loop that follows iterates through each coordinate of the 2D grid, converts the coordinates to the range [0,1], and passes them as a new data point to the KNN function with the optimal value of "K" and the scaled dataset "scaled_df". The predicted class of the new data point is then appended to the "points" list as a 3-tuple containing the x-coordinate, y-coordinate, and predicted class.

The "points" list is then converted to a pandas DataFrame object with columns "x", "y", and "class" representing the x-coordinate, y-coordinate, and predicted class of each data point.
The "grouped_points" variable groups the data points by their predicted class.

Finally, the scatter plot is created using the matplotlib library. The for loop iterates through each group of data points, represented by "pts", and plots them as scatter points using the "scatter()" function. The color of the scatter points is determined by the "color" variable, which corresponds to the predicted class. The "s" parameter controls the size of the scatter points, and the "label" parameter sets the label of the legend for each class.The last three lines set the title of the plot, create a legend, and display the plot.



1.4 Prediction accuracy of a learning method (A3)

This code block is creating a list of 5 dataframes from the given input dataframe scaled_df. The dataframes in the list represent the folds created for performing cross-validation. The sort_values method is called on the scaled_df dataframe to sort the data according to the "age_at_operation" column. Then, the iloc method is called to get every fifth row starting from the first row, then every fifth row starting from the second row, and so on, up to the fifth row. These rows are used to create five different dataframes, one for each fold. The reset_index method is called on each dataframe to reset the indices after the rows are selected.

The next block of code is performing the k-fold cross-validation method to estimate the performance of the KNN model on the given dataset. The dataset is first divided into five folds. For each fold, LOO (leave-one-out) cross-validation is performed to compute the error for each value of K ranging from 1 to 99. The optimal value of K for each fold is determined by finding the K that results in the lowest error. The LOO error for the optimal K value for each fold is stored in the loo_val_errs list. The optimal_Ks list stores the optimal K value for each fold. cross_val_errs is a list of data frames,

where each data frame corresponds to a fold and stores the K and the corresponding error for each value of K ranging from 1 to 99.

The est_test_errs list contains the test errors estimated using KNN with the optimal k value, for each of the five folds. The test() function takes in a training set (composed of four of the five folds), a test set (the remaining fold), and the optimal k value for the corresponding fold. It returns the mean squared error (MSE) of the predictions made by k-NN on the test set.
The est_test_errs list can be used to estimate the overall performance of the KNN classifier on new, unseen data, by taking the mean of the test errors over the five folds.

	Fold number	Optimal value of K	LOO validation error	Estimated test error
0	1	11	0.245902	0.245902
1	2	11	0.300000	0.236735
2	3	5	0.150000	0.261224
3	4	1	0.283333	0.326531
4	5	9	0.233333	0.273469

1.5 Performance Evaluation

The final code block calculates the average LOO cross-validation error, the average estimated test error, and the optimal K error. It then prints the results and calculates the percentage difference between the average estimated test error and the optimal K error.It's worth noting that the performance evaluation step only calculates the mean squared error and doesn

- Average LOO cross validation error: 0.2425136612021858
- True test error: 0.2687721646035463
- Optimal K (from A2) error: 0.2262295081967213
- The true test error is 18.81 percent more than that of Optimal K error (from A2)

---

2 Topic-B: Parametric Models

This part of homework illustrates the estimation of parametric models using maximum likelihood estimation (MLE) methodology.

2.1 Method Implementation (B1)

2.1.1 Covariance and discriminant

This code defines three classes IndependentCovar, EqualCovar, and DiagonalCovar for classification using a Gaussian mixture model.

The IndependentCovar class assumes that the covariance matrices of the two classes are independent of each other. It has two methods: getCovar() and classify(). getCovar() takes two data frames C1 and C2 containing the training data for two classes, and their respective probabilities P_C1 and P_C2, and computes the covariance matrices S1 and S2 of each class. classify() takes a series x, probabilities P_C1 and P_C2, mean vectors m1 and m2, and covariance matrices S1 and S2 of each class, and returns the predicted class label based on a Gaussian mixture model with independent covariance matrices.

The EqualCovar class assumes that the covariance matrices of the two classes are equal. It has the same methods as IndependentCovar, but in getCovar(), it computes a single covariance matrix S as a weighted sum of the covariance matrices of the two classes.

The DiagonalCovar class assumes that the covariance matrices of the two classes are diagonal matrices. It also has the same methods as IndependentCovar, but in getCovar(), it computes the diagonal covariance matrices S1 and S2 by setting all non-diagonal elements to zero.

Overall, these classes provide different ways to compute the covariance matrices and hence different ways to classify data points based on a Gaussian mixture model.

2.1.2 Test Error Calculation

Test error calculation using mean square error as the metric The testError function takes a trained model, a test dataset, and the required input parameters for the model. It returns the Mean Squared Error (MSE) between the actual and predicted values of the test dataset.

First, the function creates two lists: actual_values and predicted_values. The actual_values list contains the values from the last column of the test dataset, while the predicted_values list is initialized with None.

Next, the 8th column of the test dataset is dropped, as it contains the actual values that were added to the actual_values list earlier.

Then, the function iterates over each row of the test dataset, and for each row, it calls the classify method of the provided model object with the input parameters required by the model, and the features of the current row from the test dataset. The predicted class label is then added to the predicted_values list.

Finally, the function calculates the MSE between the actual and predicted values using a list comprehension. The absolute difference between the actual and predicted values is summed up, and the result is divided by the total number of rows in the test dataset to obtain the MSE. The MSE is then returned as the output of the function.

2.1.3 Main function implementation:

The MultiGaussian() function is to implement the entire process of training and testing a multivariate Gaussian classifier.

The function reads the training and testing data from the provided file paths and calculates the mean and covariance matrices for the two classes based on the training data.

Depending on the value of the Model parameter, it creates an object of one of the three classes: IndependentCovar, EqualCovar, or DiagonalCovar. These classes implement different methods to calculate the covariance matrices for the two classes.

Once the mean and covariance matrices are calculated, the testError() function is called to calculate the mean squared error on the test data.

The function then returns a pandas DataFrame containing the prior probabilities, mean vectors, covariance matrices, and test error for the two classes.

2.1.4 Output

After calculation of parameters and error rates for all combinations of models and datasets now we are printing that parameters.

```
Model 1:
  P(C1): 0.3
  P(C2): 0.7
  m1: [ 0.43  2.02  3.18 -2.43 -2.52  3.24 -5.52 -6.69]
  m2: [ 4.58  6.49  6.43  1.69  2.29  8.36 -0.17 -1.8 ]
  S1: [[ 1.86  0.23  0.75  1.    0.42  1.22  1.13 -1.19]
       [ 0.23  3.54  0.3  -0.13  1.53  1.   -0.19  3.17]
       [ 0.75  0.3   7.84  1.29 -0.41  1.72  0.34  0.22]
       [ 1.   -0.13  1.29  4.09  0.92  0.72  1.03  1.91]
       [ 0.42  1.53 -0.41  0.92  4.    0.97 -0.53  3.32]
       [ 1.22  1.    1.72  0.72  0.97  3.93 -0.19  2.22]
       [ 1.13 -0.19  0.34  1.03 -0.53 -0.19  4.08 -1.65]
       [-1.19  3.17  0.22  1.91  3.32  2.22 -1.65 16.53]]
  S2: [[ 3.42  2.07  2.57  2.61  1.77  1.83  2.68  2.93]
       [ 2.07  5.78  2.18  2.72  3.16  2.89  2.76  5.85]
       [ 2.57  2.18  8.71  3.38  2.83  2.23  2.75  5.18]
       [ 2.61  2.72  3.38  8.17  3.58  2.66  2.02  8.4 ]
       [ 1.77  3.16  2.83  3.58  5.57  2.91  3.25  4.82]
       [ 1.83  2.89  2.23  2.66  2.91  3.73  2.23  4.48]
       [ 2.68  2.76  2.75  2.02  3.25  2.23  8.21  4.31]
       [ 2.93  5.85  5.18  8.4   4.82  4.48  4.31 19.85]]

Model 2:
  P(C1): 0.3
...
       [ 0.    0.    0.    0.    5.57  0.    0.    0.  ]
       [ 0.    0.    0.    0.    0.    3.73  0.    0.  ]
       [ 0.    0.    0.    0.    0.    0.    8.21  0.  ]
       [ 0.    0.    0.    0.    0.    0.    0.   19.85]]
```

2.2 Prediction accuracy of the learning method (B2)

Displaying error rate of each dataset with each model:

```
Error rates:
  For dataset 1:
    M1: 0.22
    M2: 0.17
    M3: 0.18

  For dataset 2:
    M1: 0.23
    M2: 0.55
    M3: 0.52

  For dataset 3:
    M1: 0.11
    M2: 0.45
    M3: 0.06
```

At the end we can conclude that :

- For dataset 1 : Model 2 (Equal Covariances)
- For dataset 2 : Model 1 (Independent Covariances)
- For dataset 3 : Model 3 (Diagonal Covariances)
