Homework 1

Shrey Patel

January 15, 2024

1 Combining all data files

Starting off this homework, I first downloaded the zipped file off the patients from UC Irvine Repository. After unzipping all the files, I found that they had no extension. To make it easier on the task ahead, I uploaded all the files to Jupyter Notebook. After, I utilized the pandas library to load all the files, and add them to a pandas dataframe. I did this using a for loop as it makes it easier than manually adding to the dataframe each time. Finally, using the pandas ability to make csv files, I saved the dataframe as a csv file. These files will be in the submission called CombiningData.ipynb and combined$_dataset.csv$.

2 Task 1: KNN Classifier

2.1 Loading data and EDA

Starting off the KNN task, I uploaded the prior combined dataset.csv to the python file. Knowing that the first 2 columns are not needed, I dropped them. Then I went over the exploratory data analysis. In this, I found the length of the dataset, the unique values in the 'Code' column and the unique values in the 'Value' column. Furthermore, I found the number of occurence for each of these unique values. Another key aspect in exploratory data analysis is to look for null values in the dataset, thus I used diabetesdf.isnull().sum() to find the number of null values for each column. The result was 0 for both. Finally, I made two plots. One was to show the Code column versus the number of occurences in a bar plot. The second graph shows the correlation between the Code column and Value column on a split plot.

2.2 One-Hot Encoding

Next, I turned the original dataset into a one-hot encoded one. This will create binary columns and will only return a 1 if there is a Code value present there. Each row should only have a single 1 value. This transformed the shape of the dataframe from 29330 rows by 2 columns to 29330 rows by 25 columns. 24 columns are the Code features.

2.3 Training-Validation-Test Sets

This step utilized scikit-learn's train test split module to make a training, validation and test set. Each was split into 60, 10, and 30 respectively. Next, I split the features into the input(x) and target(y) variables. All of the Code columns would be the inputs, and Value would be the target. Next, I printed out each of the sets to see their sizes, training had a size of 17598 rows, validation had a size of 2933, and test had 8799 rows. All of them had 25 columns

2.4 KNN Algorithm

Finally, we have reached the KNN function itself. The first thing I did was import tqdm to keep a progress bar on the code running at hand, since there were a lot of rows. To start off with the KNN, I first made a euclidian distance function that would calculate between 2 instances. Then, I made the KNN prediction function. This function would take the training data, Value as the target, testing data, and k as its parameters. There are 2 other parameters, one being a subsample size in case the program is not able to run. The other one is the number of rows I want the prediction for. k would represent how many of its nearest neighbors to find. Next, the function loops through the rows in the test data, and calculates its euclidan distance to the row in the training data. Based off that calculation, the k nearest neighbors would be determined. If k is 3, then it would find 3 nearest neighbors. k can be any value as long as it is within the index of the dataframe. Now based on those nearest neighbors, the KNN function is able to make a prediction as to what the target 'Value' should be. Now after, running the code, it was very possible to find predictions for all 8799 rows of the test data. It will just take a while.

2.5 Metrics

In this section, I found the metrics of the KNN predictions. However, an issue I ran across was that my code could not run for all 8799 rows of the test data. Thus, I only chose 2 rows to make metric calculations on. This section, I made another KNN prediction function that would also inlcude the metrics calculations in its functions. After running the code, I saw that I had 0 percent across the board for accuracy, precision, recall, etc. This makes sense as when I ran the prior KNN algorithm, none of my predictions matched the actual target 'Value'. Thus, it made sense when my plot showed 0 for accuracy as k changed.

3 K-Means

3.1 Loading Data and One-Hot Encoding

This time, instead of loading the csv file, we will start again with the 70 original data files. This is to make it easier to break the files into vectors for each patient, and then combine them into one single final matrix instead having to break down the csv dataset then recombining. After loading the files, each file's(patient's) sequence is transformed into one-hot encoded vectors. They will then all stack into one final matrix or numpy array, comprising of 70 rows. 70 rows for 70 patients.

3.2 the K-Means function

Once again, I imported the tdqm libary to have a progress bar on the code running. First, I randomly intialized all the k clusters. To make the sequences equal length, I padded them with zeroes. Next, I assigned each data point to its nearest centroid and then made recalulations based on those points. Finally, the K-Means function runs with the updated centroids and repeats for as many iterations possible. Finally, to put it all together, I first call upon the pad sequences function on the original matrix to make it equal length. Then, I call the K-Means function on that padded matrix and then print it to get the results. After doing that, I print out the silhouette score. Something that I noticed was that as k, the number of clusters increased, the silhouette score decreased. Thus, it became weaker as k increased

4 Weka

Weka is a phenomenal tool that helps in learning machine learning algorithms. However, I found that when I tried to load in the datasets for KNN classification, it would not let it run. However, I was able to utilize the clustering tools. What I found was that with the Code and Value columns, it returned out a 88.6737 percent of incorrectly clustered instances.

References

Dataset: https://archive.ics.uci.edu/dataset/34/diabetes