Cindy Hernandez
CIS 3100
Professor Jairam
19 May 2024

Using KNN and Naïve Bayes Classifiers on Breast Cancer

Breast cancer is one of the most common cancers found within women across the world. Millions of people are diagnosed a year, and it is a problem that has yet to find a solid cure. Early detection and diagnosis are recommended and effective for treatment and increasing a patient's chance of being cancer free. For this project, I used a Breast Cancer dataset to implement two machine learning classifiers, known as K-Nearest Neighbors (KNN) and Naïve Bayes to predict whether a tumor is malignant or benign in a patient. The Class 2 is considered the malignant, while Class 4 is considered benign.

The Naive Bayes classification and K-Nearest Neighbors (KNN) classification are two machine learning approaches for classification applications. Naive Bayes is a probabilistic classifier that uses the Bayes' Theorem and then assumes strong (also known as naive) independence across its features. It calculates the probability for each class and assigns a class label with the highest input. Despite its assumption of feature independence, which can be a downside when classifying, Naive Bayes is useful in real-world applications such as detecting spam.

K-Nearest Neighbors is an instance-based learning method used in classification, as well as regression. It categorizes the data points based on the classification of their neighbors. It locates the K closest data points (known as neighbors) to the location of interest, usually through Euclidean distance, and then assigns the most frequent class among those neighbors. KNN makes no assumption of the data distribution making it non-parametric. This makes it beneficial when a decision boundary is irregular. However, it can quickly become computationally expensive when using larger datasets because it requires calculating the distance between the query and all points in the dataset.

For the K-Nearest Neighbors (KNN) Classifier, I used Python and imported csv, math, counter, and random shuffle libraries. Since, we were not allowed to use higher libraries such as NumPy and Pandas, we had to create our own constructors to run these processes. These were very helpful in managing our data and executing our mathematical operations. The first class that was formed was our DataHandler, which would manage data loading, preprocessing, and splitting our dataset. For this class, it was responsible for reading the csv file, transforming our feature values to floats for our analysis, and distinguishing from features and labels. The second class we used was labeled KNearestNeighbors, which would be responsible for our KNN analysis. It was responsible for creating methods to fit the model with training data, calculating the Euclidean distances between the data points to predict their classification, and determining the majority class of the nearest neighbors.

In our KNN class, we have a constructor that will initialize k values that we will use as our neighbors as well as the test and training sets. Then we have instance methods that will store the training sets and their labels (fit), calculate the distance between input features and training data and will sort the distance and find points for the neighbors (predict_neighbors), predicting the list of features and storing them (predict), determining the most common neighbor (common), calculating the Euclidean distance (_euclidean_distance), and lastly the classification report that will print out all of our points (classification_report). The Euclidean distance is used

to determine the similarity of data points. This is done in the KNN class's predict_neighbors method, which will calculate the distance between our input features and all our training data points.

For the Naïve Bayes classifier, I again used Python and imported math, csv, and shuffle libraries. I wanted to have a separate class for both KNN and Naïve Bayes to keep it organized as well as have its reusability be functional. The Naïve Bayes Classifier class offers methods for computing prior probabilities and conditional probabilities, as well as predicting class labels using these probabilities. The Naive Bayes technique was used to calculate the probability of each class based on the input features. This required establishing prior probabilities and likelihoods using the training data. I used the Gaussian Naive Bayes formula on continuous data, assuming that it is a normal distribution. This decision was made to properly manage the properties.

In our Naïve Bayes class, we have a class named DataHandler that initializes the filepath, as well as read the csv file, create our training and test sets, and separating our features and labels from the data. In our second class Naïve Bayes, I started by initializing a dictionary to store the means, standard deviations, and class probabilities for each class. I then created instance methods that would prepare and sort the data for analysis. For the first instance method fit, I trained the classifier by calculating the class probabilities, as well as the means and standard deviations for each feature. I then created an instance method named _calculate_means_stds, which would calculate the mean and standard deviation of each class based on the class and features. Then there's an instance method with predict_single, which would predict the class of a single feature set, and read the prior probability of the class and multiply it with the features to get the class with the highest probability. The _calculate_probability instance method would calculate probability using the Gaussian method. The predict instance method will simply predict a list of feature sets. Lastly, the classification_report instance method will generate a report with our findings and separate them into unique values.

One of the biggest challenges I had while working on this project was having a functioning csv file. I had tried multiple csv's, yet they were having issues with string that should have changed to a float. This caused major delays and issues to ensure the best analysis we could have gotten. Despite trying my best to rectify the problem, I was unable to fix it and opted for a new dataset. Once I switched my dataset, I was able to try out the code I had written. In the KNN analysis, I had issues formatting how I wanted the instance methods to run. I was confusing which operations were which, which effected the outcome of my analysis the first time I ran it. Since we are testing multiple neighbors, it was crucial for me to have the proper code to accurately get the information I needed. For example, in the predict_single instance method, I was mixing it up with _calculate_class_probabilities since they are both using the probability of classes in their instance variables. I was able to fix this by calculating the class probabilities first and then using the predict_single method to calculate the probabilities with said classes.

This project used machine learning methods called the KNN Classifier and Naive Bayes. These classifications can be used in real-world application, and in our case, within diagnosing medical conditions. Breast cancer detection is an important disease where these analyses can have a huge impact and could improve or speed the process of potential medical discoveries. These classifications can help detect and diagnose breast cancer early, resulting in faster recovery processes and successful patient outcomes. The information and execution learned from this project has shown the importance of information when performing these classifications.