Srikar Arani | Perm: 3223005 | srikar.arani@gmail.com

Professor Xifeng Yan

CS 165A

16 February 2021

# Machine Project 1: Naive Bayes Classifier

## Architecture

The Naive Bayes Classifier was written using C++, due to the author's familiarity with C++ as opposed to Python. The project consists of 3 files — "mp1.cpp", :naiveBayes.cpp", and "naiveBayes.h" — all of which are compiled together using the included Makefile.

The "naiveBayes.h" file is a header file that includes a constructor, destructor, and two member functions: *readTraining()* and *readUnknown().* The header file also initializes all the variables required to construct the probabilities in the Bayes Classifier. All variables and member functions are left public for ease of production and testing.

The "naiveBayes.cpp" file consists of the 4 functions outlined in the header file. The constructor sets all the initial probabilities and counts to 0. The destructor is left blank as there are no dynamically allocated elements in the project. The function *readTraining()* is used to fully read and process all the training data (this will be more closely examined in the Preprocessing section). The *readUnknown()* function is used to read the public training set and use the information to calculate both the probability that a specific patient has either died or survived. It will then compare both probabilities and then make a prediction based on the probabilities (this will be more closely examined in the Model Building and Results sections).

## Preprocessing

Preprocessing the input information into usable information represented the most important challenge of the build. The project will pass the training document argument into the function *readTraining()*, where the function will make use of an ifstream object to open up the csv file. From there the csv file can be seen as a series of comma separated attributes, where each line represents a new patient. The document is read by entering a while loop and making use of the *getline()* function to split the csv by line and loop through each line one at a time. In the while loop, the *getline()* function is used again with a comma (',') used as a delimiter to split each line by commas and store the result into an associated value. This allows immediate access to the value of each attribute for each patient to be used while iterating through. This process is repeated exactly with the function *readUnknown()* with the public dataset passed in as an argument to perform the same preprocessing on the public dataset.

**Model Building**

The actual Bayes Classifier training happens using the function *readTraining()*. As explained in the previous section, each line is split up and iterated through one at a time, while variables are declared to keep track of the value of each attribute for each line. The model first splits the data by checking if a line represents a living or dead person. If the person is dead (the date_died attribute is not equal to "9999-99-99") the program will check the value of each attribute. If the attribute is numeric, it checks to see if the value is either "1" or "2". If the value is 1, it iterates the value of a variable that keeps track of that variable's "1" results. In the case that it is 2, it iterates the value of a variable that keeps track of that variable's "2" results. It will repeat this for each attribute in the line as long as it is numeric (the model does not handle dates; see weaknesses). This process is then repeated for all living patients, except using a separate set of variables to represent the number of each value of attributes in the case that a patient lives. The total number of dead and living patients is also accounted for, helping to create the probabilities of each attribute given that they are alive or dead. At the end of the function, all variables should have a value between 0 and the total number of patients.

The function *readUnknown()* then uses these totals to calculate probabilities. The basic method for prediction using Naive Bayes Classification requires summing the chance of each probability for an unknown line of attributes. It works by first preprocessing the csv as before. It then checks the value of each attribute — if "1" it will multiply the running product by the probability of that attribute being a "1" (the total numbers calculated by the *readTraining()* function divided by the total number of living and dead), and vice versa with the value "2". There are two running totals, the probability that the unknown patient is alive, and the probability that the unknown patient is dead. These are both required as the greater of the two probabilities represent the prediction for the model. This is repeated for every attribute, and at the end, before moving to the next line, the function checks to see if the probability of death is larger than the probability of survival — if survival probability is higher than probability of death, then the predictor outputs 0, otherwise, it outputs 1.

**Results**

With the provided datasets, the approximate runtime is 3.06ms with the provided dataset as well as an accuracy of 87.4642%. Given that the approximate correct implementation accuracy is about 87%, this approach seems to provide a mostly correct implementation.

The most important factors using an information gain calculation are given below:

1. covid_res
2. contact_other_covid
3. intubed
4. icu
5. patient_type
6. pneumonia
7. diabetes
8. hypertension
9. renal_chronic
10. other_disease

**Challenges**

The project seemed initially rather complex to even get a handle on. In order to combat this, the author first started by recreating a table from the homework and trying to first calculate probabilities that could easily be checked by hand. After devising the best way to calculate probabilities, the next step was to recreate the prediction method as was used by hand for the homework problems. After figuring this out, the next challenge was to expand the parameters of the homework classification system to the covid testing set. This was made significantly easier by slowly changing parameter names and one at a time adding columns to the table until it was easy to understand the patterns and finish all the numeric columns.

**Weaknesses**

This classifier is not perfectly calibrated even for the current dataset. Like mentioned in the Model Building section, the classifier treated every column as a binary, thus disregarding unknown values. This loses a sense of accuracy in the final product as any column that has high information gain but an unknown value is completely disregarded. The other issue is that any column that is not able to be represented by a single integer is disregarded, which contributes to an overall loss of precision. This could be rectified with a more careful approach to such parameters.