**Header**

Group 36
Michael Downs (michaeld20312@gmail.com)
Max Hymer (hymermax1@gmail.com)
Project 1
8/30/2024

## System Requirements

- (1) Download the five (5) data sets from the UCI Machine Learning repository. You can find this repository at http://archive.ics.uci.edu/ml/ as well as in Brightspace. Alternatively, you may download the data sets from Brightspace.
    - This requirement is very straightforward - we will simply download the breast cancer, glass, iris, soybean (small), and vote .data and .name files from Brightspace. This is an important step in analyzing how the datasets are oriented so that we can fit our pre-processing algorithm to the data.
- (2) Pre-process the data to ensure you are working with complete examples (i.e., no missing attribute values) and discrete features only.
    - We intend to read in the data then replace all instances of "?" (as that is what marks an incomplete data point in the relevant data sets) in the data with a random value within the constraints of the feature (i.e. if the feature contains integers from one to ten, we replace any "?" within that feature with a random integer from 1-10). We want to do this because we are already going to have to pass some sort of data with 10% of a feature shuffled. This should help us be more successful on both data sets instead of being much more accurate on one. We also want to be able to provide the training algorithm with consistently formatted information, as well as something that is simple to analyze based on the data structure. We will do this by having multiple functions in our driver class that take in information and format it similarly.
- (3) Implement the above algorithm and test it on two different versions of the data. The first version is what you get from the repository without change (other than, possibly, data imputation). The second version goes through your data, selects 10% of the features at random and shuffles the values within each feature, thus introducing noise into the data.
    - We intend to implement the Naive Bayes algorithm as its own class. We will use a driver unique to each data set to process each data set. We will first read in the data, run data imputation, and run the training algorithm and tests on the data. Then, we will update the driver to select 10% of the features at random and shuffle the values within each feature before it runs the training algorithm and tests. We will hopefully be able to process this information similarly based on our preprocessing process. There is not much creativity here in the training as we simply record data till there is none left and calculate probabilities of classes and features.
- (4) Select and implement at least two (2) different evaluation measures (i.e., loss functions) that you will use to evaluate your algorithm. Example loss functions include 0/1-loss, precision, recall, and F1-score.
    - We intend to implement the precision and recall loss functions to evaluate the performance of our algorithm. We want to use these two measurements because they specifically tell you how often you are giving false positives and false negatives. This should be extremely helpful if we need to adjust our training algorithm or classification function because it will give us a sense of how we need to change our classification.

- (5) Develop a hypothesis for each data set based on expected performance on the different data sets.
    - We intend to develop a hypothesis for each data set on the expected performance on the different data sets based on the research we have done online regarding the Naive Bayes algorithm (i.e. we expect it to do very well with data sets with both highly independent and categorical features but relatively poorly with data sets with both highly dependent and quantitative features). We predict this because of the raw nature of how our algorithm values independent features.
- (6) Design and execute experiments using 10-fold cross-validation to test your hypotheses, comparing performance on the unaltered and altered data sets from the UCI repository.
    - We intend to create a two dimensional array to store our dataset information. We are simply going to take row and column values and split the information up into chunks. We will then randomly combine 9 of the 10 chunks and pass them through the training function. Once we have done that and we calculate all of the probabilities, we will pass the “test” set through our classifying algorithm and report analyses.
- (7) Write a final paper encompassing the results of our tests
    - This will be an extremely similar document to this one, with all of our testing, reflection on hypothesis and analysis of our data structuring, flow and test strategy.

**System Architecture**

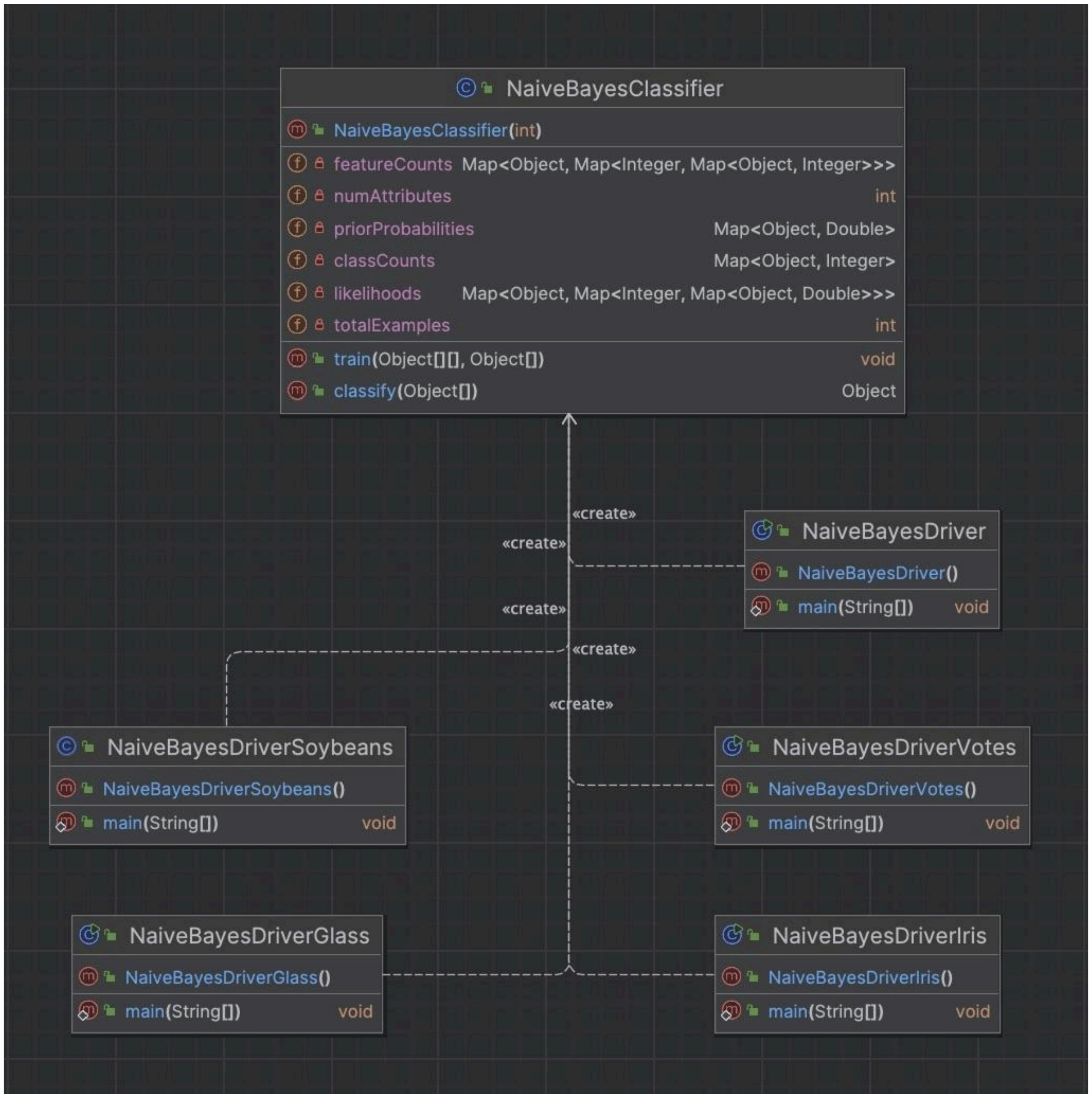

Figure 1: Project 1 UML diagram

The NaiveBayesClassifier class implements the Naive Bayes Classifier algorithm. The constructor takes the number of attributes (features) within the given data set (as an integer) as an argument and initializes the class data structures. Obviously, it outputs a NaiveBayesClassifer object. The train method is used to create the model that the algorithm will use to classify test data. It takes a 2D array of type Object (so that it works for any data type) containing every line of relevant data for the given data set and a 1D array of type Object (for the same reason) containing the correct classification for each line of the relevant data as arguments. It does not output anything; it simply fills the data structures that were initialized when the class object was created (i.e. featureCounts, numAttributes, etc.). The classify method takes a 1D array, again of type Object (again for the same reason) containing the line of test data as an argument and outputs a classification of type Object based on the results the algorithm produced in the train method.

We created a unique driver for each unique data set. Each driver inputs the corresponding data set, performs data imputation on that data set if needed, puts the lines of data and the classifications in arrays of type Object, creates a new NaiveBayesClassifier object with the corresponding number of attributes, runs the train method on the arrays, runs the classify method on the test data, and prints the classification found by this method. This will satisfy the first half of the testing requirement. The drivers will be updated to randomly select 10% of the features within the data set and shuffle the feature values after the initial tests are conducted. This will happen after the creation of the NaiveBayesClassifier object and before the train method is run and will satisfy the second half of the testing requirement.

## System Flow

For the flow of our system we want to use a larger number of classes to make different datasets smoother to process through our main training and classification algorithm. That way we will have them all formatted similarly so that our algorithm will be successful in all the data that we need to process. Starting from the beginning of reading in the data, we will store it in a 2 dimensional array. We are then going to pass that data into our pre-processing aspect of our algorithm. This algorithm is going to output two versions of the data, one that will be the same as the data excluding unknowns. Then, output a dataset that will choose a random feature and shuffle 10% of the values in that feature. We will then devise the data into 10 random sections. Using a random number generator, piece nine of them back together as the training data. We are going to then pass our shuffled dataset for training into the training and classifying class. We will process the probabilities of each class likelihood and feature likelihood. After this, we will input the test data (other 1/10 of the dataset) into the classification algorithm. This algorithm will take known probabilities and calculate which classification is most likely based on feature values. While running our classification algorithm, we are going to return information about how the classification algorithm is doing in terms of precision and recall. Then, we will repeat the above steps after implementing the first version of the training set into our trainer, with our dataset containing noise. This process should make our program easy to debug and give us valuable information that follows a logical process of the information to complete our machine learning algorithm.

## Test Strategy

We plan on using nine tenths of the data set for training and the other tenth for testing our algorithm. We will also be randomizing our data set so that we do not have any strong correlations between clumps of data sets that would be present in our portions of training sets. We will also be providing precision and recall functions regarding the accuracy so that we can tell if we are over or underfitting classes. This should be very helpful if we find that our classification algorithm is not extremely successful, how we should change it. Especially in circumstances where there are decimal valued features, where we plan to use ranges or “bin method” to be able to make more similar instances. We also plan on using text return messages to the console that give us information about how our code is working as we test it. We have made many different classes for processing each of the required data sets, so that we can be able to process different syntax and meanings for different data sets. This should be helpful in saving us time in the long run once we start testing.

## Task Assignments and Schedule

Michael Downs

Download and analyze the data sets to satisfy requirement 1 - 8/30
Design and code the drivers to properly preprocess each data set to satisfy requirement 2 - 9/4
Implement the Naive Bayes algorithm and code driver to run raw data through the algorithm to satisfy the first half of requirement 3 - 9/6
Update drivers to select 10% of features within each data set and shuffle the feature values and run this data through the algorithm to satisfy the second half of requirement 3 - 9/8
Final paper - 9/13

Max Hymer

Develop a hypothesis for each data set to satisfy requirement 5 - 9/4
Design and execute experiments using 10-fold cross-validation to test your hypotheses on the raw data to satisfy the first half of requirement 6 - 9/7
Design and execute experiments using 10-fold cross-validation to test your hypotheses on the updated (10% features then value shuffled) data to satisfy the second half of requirement 6 - 9/9
Implement loss functions to satisfy requirement 4 - 9/11
Final paper - 9/13

We plan to communicate through text messages and meet in person to work on the project. As well as put into our paper what we worked on in the program. We also plan to make the video together.