Kellen Hurley, Alex Ellingsen
Team 38
30 August 2024

Project 1 Design Document

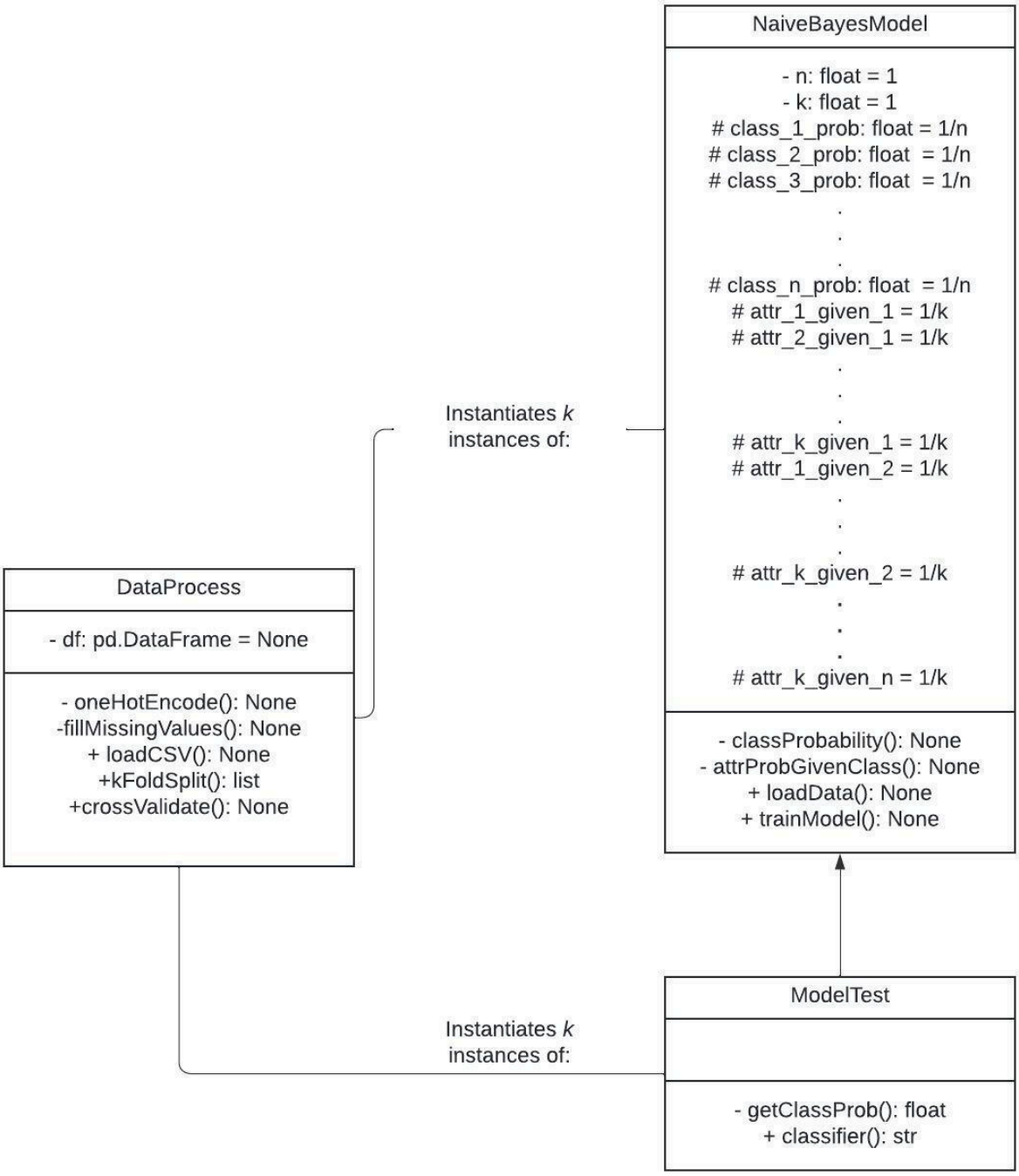
System Requirements

There are three portions of the assignment other than this design document that must be completed: the programming portion, the paper portion, and the video portion. For the programming portion, five requirements must be met. The first requirement is to pre-process the data ensuring that we are working with complete examples and discrete features only. In order to obtain complete examples only, we intend to use data imputation. We plan on discretizing continuous features by using a binning method and using one-hot encoding for each of the categories. The second requirement is to implement the algorithm given in the project document and test it on two different versions of the data. The first version is what you get from the repository without change, and the second version goes through the data, selects 10% of the features at random, and shuffles the values within each feature, thus introducing noise into the data. To train and test the algorithm, we are going to separate the training and testing steps into two separate classes, the NaiveBayesModel class, and the ModelTest class. The NaiveBayesModel class will handle the steps of the training algorithm, and The ModelTest class will then classify the examples from the test set (the ModelTest objects will be used in a class called DataProcess). The third requirement is to select and implement at least two different evaluation measures that we will use to evaluate the algorithm. We will be using recall and 0/1-loss to perform our evaluation. We chose recall in particular because in the case of the breast cancer data, we want to pay especially close attention to the false negatives. The fourth requirement is to develop a hypothesis for each data set based on the expected performance on the different data sets. We will form our hypotheses by analyzing each data set for things like conditional dependence of the features since this is a fundamental assumption of the Naive Bayes model. Lastly, the fifth requirement is to design and execute experiments using 10-fold cross-validation to test our hypotheses, comparing performance on the unaltered and altered data sets from the UCI repository. This requirement is important because a k-fold cross-validation test ensures that many different sections of the data are used for training and testing. For the 10-fold cross-validation tests, we will ensure the data is stratified (each fold will contain a roughly even number of samples from each class).

For the paper portion, we are to write a brief paper of length greater than or equal to five pages, but no more than ten pages using the JMLR (Journal of Machine Learning Research) format. This page limit includes figures, tables, and references. However, it does not include the appendix. Achieving the minimum page requirement should be no problem given the large number of requirements for this project. However, in order to avoid going over ten pages, we will have to carefully size graphics to balance readability and size, and proofread many times to ensure our statements are conveying all of the necessary information in a concise manner. In this paper, there are many elements that must be included. These elements include: title and author name(s), problem statement (including hypothesis), description of our experimental approach and program design, presentation of the results of our experiments, a discussion of the behavior

of the algorithms combined with any conclusions we can draw relative to our hypothesis, a summary, a references section (if we use resources other than course content), and finally an appendix listing who did what on the assignment.

For the video portion, we need to create a video that demonstrates the code we wrote in the programming portion. The video will focus on the behavior, not on walking through the code. In the video, we need to show input, data structure, and output. There is also a set of minimum requirements that the video will satisfy. We will need to ensure that the video is no longer than five minutes, that the video is provided in mp4 format (or uploaded to a streaming service with a link provided), that fast-forwarding is not done whenever there is a voice-over or when the results are being presented, that we provide verbal commentary on all of the elements we are demonstrating, that we demonstrate our discretization method for the real-valued features, that we show a sample trained model that will consist of the set of class parameter values as well as the class-conditional attribute parameter values, that we demonstrate the counting process by showing the corresponding counts for a class as well as for class-conditional attributes, that we provide sample outputs on one fold's hold-out set for one of the data sets showing classification performance on both versions of our algorithm, and lastly, that we show the performance on one fold's hold-out set for both versions of our algorithm on one of the data sets. Many of these requirements are straight-forward to achieve. We will most likely just choose to print the relevant information to the console. However, one requirement in particular may be hard to achieve, that being, the five-minute maximum length. In order to show this many things at a normal pace in five minutes, we will need to carefully plan out the sequence of events and write a script to follow to ensure no time is wasted.



System Architecture

Figure 1: UML diagram of the proposed model

Our design revolves around the use of three classes, as seen in Figure 1: DataProcess, NaiveBayesModel, and ModelTest. DataProcess will be used first to process and hold the data taken from the .DATA files. The loadCSV() method will be called to convert the raw data into a Pandas data frame and shuffle it. It will also call the methods oneHotEncode() and fillMissingValues(). oneHotEncode() will change any categorical columns into multiple one-hot encoded columns. fillMissingValues() will convert any missing values to the mean of that column. The method kFoldSplit() will take in a number, *k*, as a parameter, and divide the data into that many folds. The crossValidate() function will then instantiate *k* instances of both the NaiveBayesModel and ModelTest classes, which will be used to perform k-fold cross-validation on the data to test our model.

NaiveBayesModel and ModelTest make up the design of the model itself. NaiveBayesModel will be used first, as it contains the methods and data attributes that will train the model. Two important data attributes, *n* and *k*, will be initially set 1. Once the data is loaded into the model, *n* will be changed to represent the number of classes found in the data. *k* will also be modified to represent the number of attributes found in each example in the data set. This will allow us to then set values for *n* more data attributes known as *class_1…n_prob*, each representing the probability of each class being found in the data. These will start with equal value (1/*n*) but will be modified to accurately represent the data after training is complete. The next set of data attributes will be called *attr_1…k_given_1…n*. There will be *n*k* of these values. These represent the probability of an attribute being selected from any row, given a specific class. These will all have an equal initial value of 1/*k*, but will also be updated to more accurate values as training commences. *class_1…n_prob* and *attr_1…k_given_1…n* are protected variables to allow them to be accessed by child classes, specifically ModelTest where they will be used as the parameters to test the model. ModelTest has no data attributes of its own, instead inheriting all the data attributes from NaiveBayesModel.

There are 6 methods used in our model, four belonging to NaiveBayesModel and two belonging to ModelTest. The classProbability() method will calculate the probability of each class being selected out of the entire data set. It will modify the *class_1…n_prob* variables to accurately represent the data. The attrProbGivenClass() method will recalculate the correct probability of each attribute occurring given a specific class, and modify the *attr_1…k_given_1…n* variables accordingly. The loadData() method will be used to initialize the variables *n* and *k*, as well as create the correct number of *class_1…n_prob* and *attr_1…k_given_1…n* data attributes. The final method in the NaiveBayesModel class is trainModel(), which will be used to call the private methods classProbability() and attrProbGivenClass() to train the model according to the dataset. Two methods belong to the ModelTest class, one being public and one being private. The public method, classifier(), will be used to make a class prediction for a given example. This method will call getClassProb(), which does the heavy lifting of calculating the probability of the example belonging to a specific class, using the classification formula given in the Project 1 requirements.

System Flow

The process of working with our model would begin with the pre-processing of data. This is done by initially creating an instance of the DataProcess class. The loadCSV() method will then be called, which will convert the data to a Pandas data frame and begin to manipulate parts of the data. These manipulations are discretizing continuous values using equal-width binning and the Freedman-Diaconis rule, using one-hot encoding for categorical variables, and filling any missing values with the mean of the column. It will move the columns containing the group label for each example to the end of the data frame, and, if it exists, make the id column the index. This will allow our model to automatically load and process data without the need to change any code. Then, the kFoldSplit() method will be called. This will split the data into *k* folds, which according to the system requirements will be 10. Finally, the crossValidate() function will be called, which will use the split data from kFoldSplit() to perform k-fold cross-validation on the data using our model. The user will not have to interact with the program anymore after this step, however, there is much more happening behind the scenes. The crossValidate() function initializes *k* versions of NaiveBayesModel. These instances of NaiveBayesModel will then call the method loadData(), each using a specific fold as its parameter. This will create the variables needed for the training of the model. At this point, due to the nature of our model, you could immediately start testing it on new, classless data. However, for more accurate results, the method trainModel() should be called on the instance of the NaiveBayesModel class. Once the model has been trained, we can then create an instance of the ModelTest class. Finally, we can call the classifier() method, which will take in the testing data set of its specific fold and attach a new "class" column to the end of every sample. This will make up our predictions, which we can then analyze for accuracy using recall and 0/1 loss.

Test Strategy

Testing to check if the data pre-processing step has been successfully run, we will manually inspect a .CSV version of the Pandas data frame to check if continuous variables have been properly binned, and any categorical variable has been one-hot encoded. In order to test that we have shuffled 10% of the data for the second version, we're going to manually compare the unaltered dataset and the altered dataset and make sure they differ by about 10%. Then, to make sure the algorithm is implemented correctly, we plan to run through several iterations of the algorithm by hand and make sure it matches what the program is doing. Ensuring that the two evaluation measures are returning the correct outputs is simple, simply do the calculations by hand and compare answers between ourselves and the program. The next requirement is to develop a hypothesis about each dataset which does not require verification, one simply does or does not do it. Of course, we will take great care in making sure our hypotheses are based on something empirical about the data since it will be what "drives" the paper so to speak. To verify that we sufficiently satisfied the last requirement for the programming portion, that is, to design and execute experiments involving 10-fold cross-validation to evaluate our hypotheses, we will make sure we fully understand the concept of k-fold cross-validation to ensure we're doing the correct experiments and make sure the code reflects that understanding. To ensure the video has been successfully recorded, we will have the person who did not record the video watch it and verify its accuracy and quality. This will allow both partners to sign off on the final video product. We will use a very similar method to confirm the success of the paper. Each partner will proof-read and edit the sections they did not write to ensure quality.

Task Assignments and Schedule

In order to show a general outline of our schedule and work distribution for this project, we have created a Gantt chart:

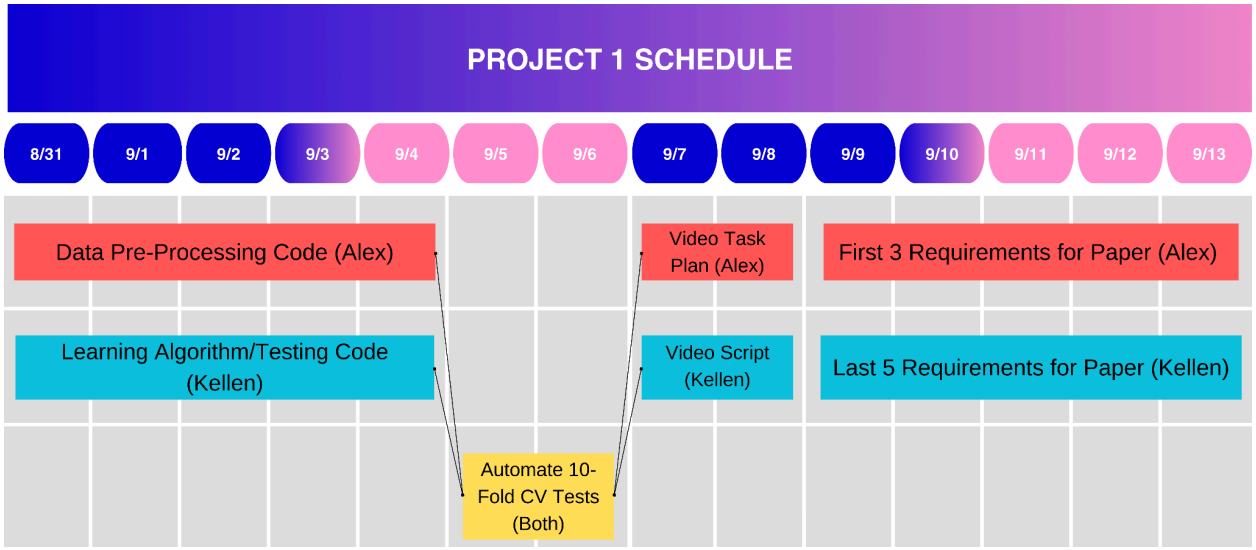


Figure 2: Schedule of Tasks

As shown in Figure 2, we plan to focus our efforts on the programming portion immediately after the design document is due. This is because every other task is dependent on us completing this portion. I (Kellen) will be responsible for writing the portion of the code that will implement the learning algorithm, as well as the testing formula. I will also be implementing the code that evaluates our algorithm using 0/1-loss and recall. Alex will be responsible for writing the code that pre-processes the data ensuring that only complete examples are being used, and all features are discretized. Once we complete those tasks individually, we will then both work on the code that will run the 10-fold cross-validation tests on our hypotheses. After this, the programming portion should be completed and then we will move onto the video. While we don't necessarily need to do the video next, we feel it follows naturally after doing the programming portion since they are heavily related. I will be responsible for writing the script for the video whereas Alex will work on the plan for the demo (i.e., which things will be shown in which order, how these things will be shown, etc.). We anticipate the video being fairly straightforward and we're confident we can complete it in a couple of days. Lastly, we will move onto the paper. Alex will be responsible for the title/author name, the problem statement (including hypothesis), and explaining our experimental approach and program design. I will be responsible for presenting the results of our experiments and discussing the behavior of our algorithms, as well as any conclusions we may draw relative to our hypothesis. I will also be writing the summary and the appendix (and potentially references if they exist). I am taking on more requirements for the paper because we feel as though many of my requirements are significantly easier, in particular, the summary, the appendix, and presenting the data. We anticipate the paper being a large part of

the workload, so we have set aside five days to complete it. If the distribution of tasks ends up being imbalanced, we may readjust slightly. However, we think this is a reasonable distribution of tasks. Alex and I have exchanged phone numbers in order to communicate about the progress of the project, and we plan to meet in person once per week to discuss in further detail. In order to ensure we're both keeping up on our portion of the programming section, we have set up a github repository to push our code to as we progress.