Project 1

Team 14
Josh Aney - josh.aney@icloud.com
Owen Cool - ohcool57@gmail.com
30 August 2024

Requirements

For this project, it is required to download the five datasets from the UCI Machine learning repository. Once the data sets are downloaded it will be required to do some data preprocessing. This includes imputation and discretization. **Imputation** will be accomplished by assigning attributes according to the probability they occur in the data set (e.g. if 30% of examples have a given attribute and 70% do not, we will use a random number generator to assign a value for this attribute, with a 30% probability the example is given the attribute). **Discretization** will be accomplished through binning based on frequency (e.g. continuously valued attributes are divided into four groups; each quartile is given a different discrete value for that attribute).
It is also required to implement the **Naive Bayes Algorithm**. For the Naive Bayes Algorithm, you will first need to implement the training algorithm.

To do this:

1. For each class in the training set, calculate
   $Q(C = c_i) = \frac{Number\ of\ examples\ of\ class\ C}{Total\ Number\ of\ examples\ in\ the\ data\ set}$
2. Split up the data into their respective classes
3. For each attribute A_j in the class-specific training set, calculate
   $F(A_j = a_k, C = c_i) = \frac{Number\ of\ examples\ in\ class\ C\ where\ the\ attribute\ value\ matches + 1}{number\ of\ examples\ in\ class\ C + D}$

Once the training algorithm has been implemented, to classify an example from the testing set do the following for each class. Calculate only for the attribute values a_k that exist in the example

$$C(x) = Q(C = c_i) * \prod_{j=i}^{d} F(A_j = a_k, C = c_i)$$

Then return the class with the highest value for $C(x)$.
We will also implement the hyperparameter a for our algorithm. While this value will have nothing to do with the performance of the algorithm, it will be used to test a **grid search hyperparameter tuning** strategy.
Once the algorithm has been implemented we will run the algorithm on the preprocessed data that came directly from the UCI Machine Learning Repository. Once this is done, our next requirement is **noise-making**. We will go through the dataset and select 10% of the features at random and shuffle the values within the features, creating a second, "noisier" data set.
We will then formulate our **hypothesis**, predicting which dataset will see better performance using the Naive Bayes algorithm, and which hyperparameter values will be most successful.
Once we have our data preprocessed, our algorithm and hyperparameter tuning implemented, and our noisy dataset, we are ready to begin testing. We will test the algorithm using **10-fold cross-validation**, comparing results across hyperparameter values and datasets. We will compare these results using the metrics of **precision, recall, accuracy, and F1-score**. Comparing these results, we will be able to **evaluate our hypothesis**.

System Architecture

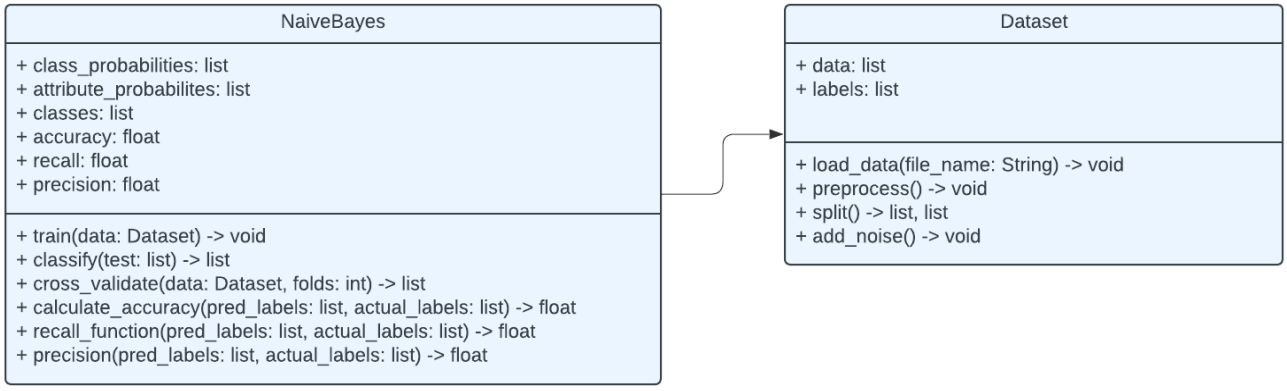

Figure 1. UML Class Diagram for Naive Bayes

NaiveBayes Class

- This class contains the entirety of the algorithm that was specified earlier in the design document. It has methods that contain the code for the training algorithm as well as the classifying algorithm. This class also contains methods that compute the loss function of the algorithm. The point of this class is to satisfy all the requirements that regard to the actual algorithm being implemented in this project.

  The attributes for this class are:

  1. Class_probabilites: Contains all the probabilities for each class that is specified in the data set
  2. Attribute_probabilities: Contains all the probabilities for each attribute in a specified class
  3. Classes: This is a list of all the classes that are present in the dataset
  4. Accuracy: This is a float that gets set by the calculate_accuracy() method and it represents the accuracy of the model.
  5. Recall: This is a float that gets set by the recall_function() method and it represents a metric that can be used to evaluate the model.
  6. Precision: This is a float that gets set by the precision() method and it represents a metric that can be used to evaluate the model.

  The methods for this class are:

  1. train(data: Dataset): This method takes in a Dataset object that contains all the data and it sets the class_probabilites attribute, the attribute_probabilites attribute, and the classes attribute. This method does not return anything.
  2. classify(test: list): This method takes in a list that comes from the split method in the DatasetClass and it returns a list of labels that the model predicted.
  3. cross_validate(data: Dataset, folds: int): This method takes in a Dataset object as well as an int that represents the number of folds. It then calculates and returns metrics for all the different cross validations that can be used to evaluate the model.
  4. calculate_accuracy(pred_labels: list, actual_labels: list): This method takes in two lists of strings that represent the predicted labels and the actual labels of the data. Then the

method calculates how accurate the model was and returns a float that represents the accuracy.

5. recall_function(pred_labels: list, actual_labels: list): This method takes in two lists of strings that represent the predicted labels and the actual labels of the data. Then the method calculates the recall function and returns a metric that can be used to evaluate the model.
6. precision(pred_labels: list, actual_labels: list): This method takes in two lists of strings that represent the predicted labels and the actual labels of the data. Then the method calculates the precision function and returns a metric that can be used to evaluate the model.

Dataset Class

- This class is where the data that is downloaded from the UCI Machine Learning Repository is stored. This class contains the code that is used to satisfy the requirements regarding any data preprocessing or data manipulation.

  The attributes for this class are:

  1. Data: This is a list that contains all the features for every single example in the entire dataset.
  2. Labels: This is a list that contains the labels for all the examples in the entire dataset.

  The methods for class are:

  1. load_data(file_name: String): This method takes in a string that represents a file name and then it will load the dataset into the data class attribute. It will also load the labels and set the labels class attribute. This method returns void.
  2. preprocess(): This method is meant to handle any missing values and discretize all values for the NaiveBayes Class. This method returns void.
  3. split(): This method is meant to split the data up into a training set and a testing set. These sets can then be used in the NaiveBayes class. This method is meant to be called after the preprocessing method. This method returns void.
  4. add_noise(): This method is meant to take 10% of the features at random and shuffle the values within that feature across the data. This method will update the data attribute. This method is meant to be called after the preprocessing method. This method returns void.

System Flow

To run the system and to make sure that all the requirements are fulfilled, start by loading all the data into the Dataset class. Depending on how the data is loaded you may have to run the preprocess() method on the Dataset class to handle missing values or any other issues with the data. Once this is done, you will spit the data into a training and test set for the model. The training set will then be fed into the NaiveBayes class through the train() method. Once this has been completed the NaiveBayes class will be able to run the classify() method on the testing set. Once this is done you will have the predicted labels from the model and they can be fed into all of the loss function methods. These will then be compared to the actual labels from the test set and the loss function methods will return a metric that can be used to evaluate the function. This will complete the requirement that specifies no data manipulation. Next, you will want to create a new Dataset class with the same data from the UCI Machine Learning Repository. You will complete the same steps as specified above, but this time after you preprocess the data you will want to run the addNoise() method on your Dataset class. This will manipulate the data in the way that is specified in the requirements. After this is done you will follow the same steps to train, classify, and evaluate your NaiveBayes class.

Test Strategy

To ensure that we meet project requirements, we will first validate that our data has been effectively preprocessed, containing only **complete examples** with entirely **discretized** attributes. This will be quite simple to validate, as our Naive Bayes algorithm will only run if this condition is met.
To validate that the **Naive Bayes algorithm** is implemented correctly, we will check that the implementation matches our planned system flow and architecture.
To test that our **grid search hyperparameter tuning** is working as intended, we will print values of the hyperparameter as we run the algorithm, checking to see that it changes values, moving through the whole grid. We will not be validating its impact on the algorithm's accuracy, as it will have no impact.
To ensure that **noise-making** functioned as intended, we will check that the original dataset and the noisy dataset have a reasonable number of distinct values.
Upon obtaining our results, we will check to make sure that **precision, accuracy, recall, and F1 score** have been calculated correctly by printing the intermediate values used to calculate these metrics and carrying out the calculations manually. We will check that **10-fold cross-validation** went as intended by ensuring that we get different results on different trials of the experiment and that we get reasonable results.

Task Assignments and Schedule

Throughout the course of the project, each team member is expected to complete their work that is assigned in a timely manner. On the parts of the project that require team members to meet in person, it is expected that the team members will communicate to make a time work where they can meet and complete the task at hand. This communication can be done directly with the other team member since there are only two members in the team. All of the code that is written in this project will be uploaded to a team repository so that it can be accessed at any time if the other team members need it. It is expected that team members will keep their code up to date in the repository if they have made any changes.

<u>Task Descriptions</u>

1. Create Design Document - Follow the Creating a Design Document PDF of D2L to create a design document specific to project 1
2. Create Dataset Class - Follow the description of the Dataset class in the Project 1 Design Document. The structure of the class should be finished after this but methods do not need to be implemented.
3. Implement Methods for Dataset Class - Follow the method descriptions for the Dataset class in the Project 1 Design Document. After this task is complete the class should be fully utilizable.
4. Create NaiveBayes Class - Follow the description of the NaiveBayes class in the Project 1 Design Document. The structure of the class should be finished after this but methods do not need to be implemented.
5. Implement Methods for NaiveBayes Class - Follow the method descriptions for the NaiveBayes class in the Project 1 Design Document. After this task is complete the class should be fully utilizable.
6. Final Code Check - For this task, the team members will need to meet in person to make sure all the code runs as expected so that the data that gets collected later in the project is reliable.
7. Run Code and collect results - For this task the code will need to be run on one team member's machine and all the results will need to be collected from the model.

8. Create Figures for Paper - For this task, a team member will be required to create all the figures that will be used in the final paper. These figures will come from any metrics that can be used to evaluate the model.
9. Create a Rough Draft of the Paper - This task is relatively self explanatory. Once the experiment has been completed and the results have been collected, a team member will be required to create a very rough outline of how the paper will be structured and what content will be in each section.
10. Complete the Paper - This will be a team effort. This task will not be done by a single teammate and it will require the entire paper to be completed and submitted.
11. Plan Video Essay - This task requires the general layout of the video essay to be complete. This allows the actual recording of the video to go much smoother and so there are no last minute changes while trying to record.
12. Record Video Essay - For this task, all of the team members will meet and record the video essay. The editing will also take place and it will be used as a final check for anything missing in the project. Once this is done they will submit the video

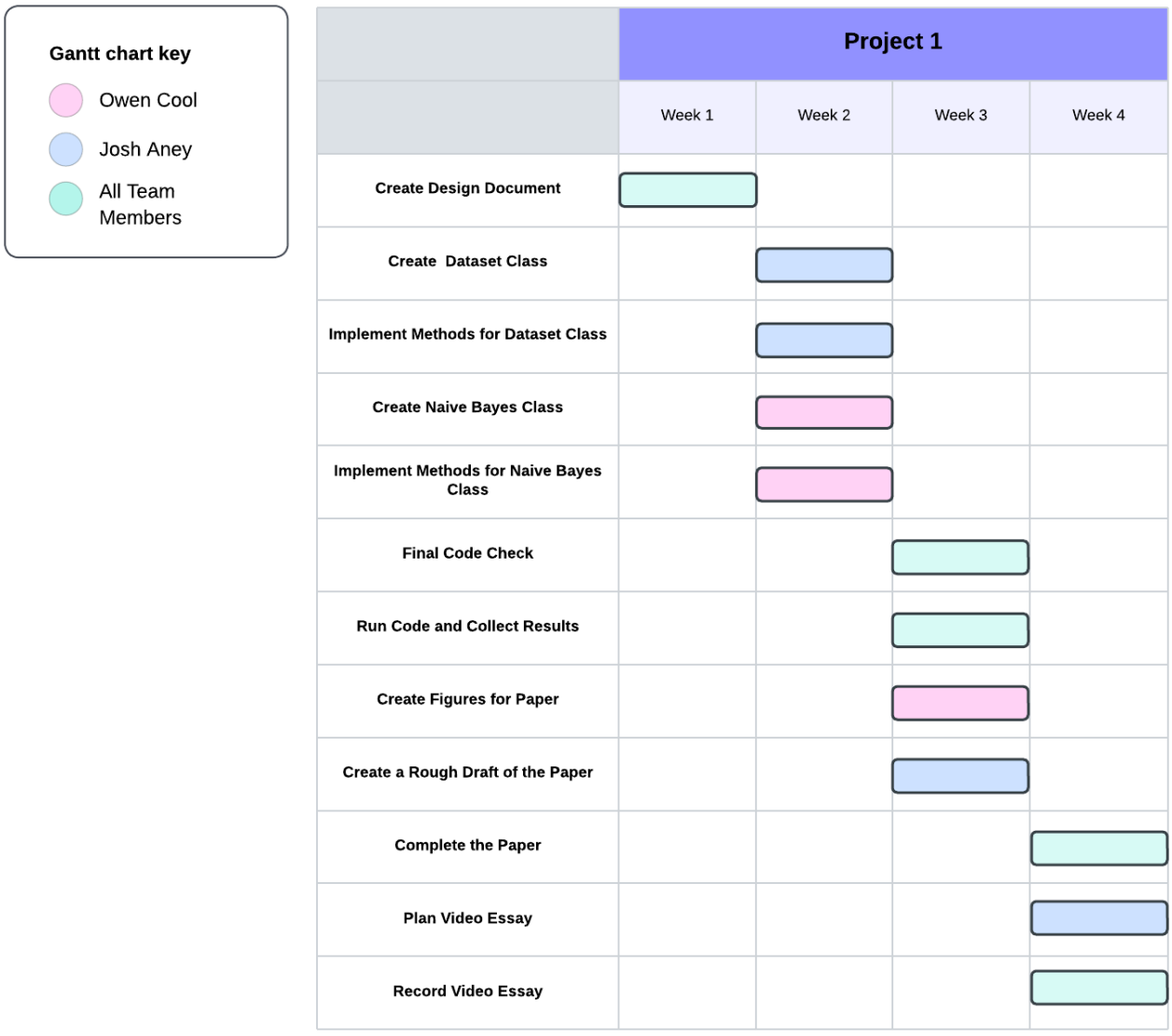

Figure 2. Gantt chart for schedule of tasks on Naive Bayes project.