CS 51 Final Project
NLP Email Event Detection

Frances Ding, Raynor Kuang, Jimmy (Hsiu-Chi) Lin, Daniel Wang
Youtube link: `http://youtu.be/QZfLtUsi3cA`

May 1, 2015

1 Background

This project was conceived after inspiration from a different source: the Harvard Computer Society (HCS). HCS had faced the problem of classifying emails as events for a tangential project, and after hearing of the Bayesian spam filter, we decided to implement a similar project classifying emails as "public events"–physically attendable events intended for the Harvard Undergraduate public–or "non public events"–unrelated emails or emails for small, private events.

Since none in the group had machine learning experience, after some advice from our project TF, Peter Kraft, we decided to stick to a Naive Bayesian. In implementing this Naive Bayesian, we ultimately needed to determine "features" or attributes of emails that could be used to fit those emails to Gaussian distributions and thus classify them. In turn, we needed to calculate "rares" or words much more common in event emails than non-event emails. We would need a "learning set" of emails to teach the program on, a "guess set" of emails to implement the program on, and a "test set" of emails from which we could randomly sample the learning and guess sets to test the program on.

We felt we implemented these tools, an adequate naive Bayesian classifier, and a UI for this classifier. There was much toying around with certain features and rares, but as a group we are satisfied in the progress we made. On a test set of size 1210, we achieved mean accuracy rate of 78%, a mean positive ID rate (correctly identifying emails as events) of 67%, and a mean negative ID rate (correctly identifying emails as non-events) of 81%. Given that the balance of tested emails was approx. 20-80 events to non-events, we felt we had made significant progress on mere guessing.

This program is best used with python 3. Additionally, you may need to install the xlrd package located here (https://pypi.python.org/pypi/xlrd). Ultimately, we are very proud of this program and what we were able to accomplish. Should any problems or questions about the program arise, please contact Raynor Kuang or any other group member.

2 Operation Instructions

2.1 Export of Emails

The preprocessor of the event detector first uses an AppleScript to export emails from a preselected inbox to an Excel file. The export includes the send time, sender information, subject line, and message body for each email in the inbox. The script can be run until the desired number of emails is exported to the Excel file. Note that the program will ultimately only consider emails coming from an official HCS mailing list (as this is closest to the conditions this program might ultimately be run on); however, it is allowed to include other emails, as the program will properly filter them out.

A column indicating whether or not an email contains an event should be manually added to the Excel containing the exported emails. For our purposes, we chose to indicate an event email with "1" and a non-event email with "0". For this project, sample data sets will be included in the "data" folder. "learn_data.xlsx" will contain a set appropriate for learning (a set of emails with labeled events), ”guess_data.xlsx” will contain a set appropriate for guessing (a set of emails without labeled events), and ”test_data.xlsc” will contain a set appropriate for testing (a set of emails labeled with events). If you would like to provide your own data sets, follow the formats provided in these sets; this includes the exact column order, as well as a title first row, so that the program appropriately retrieves data.

The code for parsing in these emails and converting them from raw email data into usable data can be found in the Utility module.

2.2 UI

The program can be run from the command line by running ”python Detect.py,” upon which a simple text UI will appear allowing for setting of rares, learning, guessing, and testing similar to the following

1. Set rares
2. Learn
3. Guess
4. Test
5. Exit

Recommended operation is to perform each of these in the given order. It is also possible to import the Detector class from the Detect module, which will run the program once (upon which you can immediately exit), and then allow for calling the specific functions so that data can be stored.

This code can be found in the Detect module.

2.3 Rares

Rares must be set at least once before any learning or guessing, which can be done using the program by selecting ”Set rares.” Upon calculation, the rares will be stored in a file of your choice, and will default to ”rares.txt.” It is possible to skip this step, but you must supply a rares file of your own selection, which will be requested in later steps. The program will ask for how many rares to provide: this will vary by learning and guess sets, but it is appropriate to use ”100” for the test set we provide.

This code can be found in the set_rares function in the Detect module.

2.4 Learning

Once rares have been established, features can be calculated. After selecting ”Learn,” provide an appropriate learning set and rare file. The program will calculate the ”prior” and ”posterior” vectors for both event and non-event emails, which represent the statistical measures (means and standard deviations) for the features as calculated among all learning emails. The program will then store these vectors in the file ”distribution.txt” for future access; it is not recommended to modify this file as it is necessary for guessing. Consequently, at least one ”Learn” must be run before any guessing.

This section can be found in the learn function in the Detect module, as well as the Features module.

2.5 Guessing

Once priors and posteriors have been calculated, the program can make guesses about new emails. After selecting ”Guess,” provide an appropriate guess set and rare text file. The program will calculate features for each of the emails in the guess set and compare those features to the prior and posterior vectors to make educated guesses - this is the center of the Naive Bayes algorithm, in which our program iterates through each feature and updates its prior probability of an email being an event based on the relative likelihood of seeing the given feature value in event vs. non-event emails. Upon completion, the program will pretty print up the emails and their guesses (since guess sets tend to be rather large, these will be capped at 40; the entirety of the list will be returned by the method

”guess” that can be found in the Detect module).

This section can be found in the guess function in the Detect module, as well as the Features and Bayes modules.

2.6 Testing

Included for the sake of program development is the testing option. After selection ”Test,” provide an appropriate test set and rare file, as well as number of rounds of testing. The program will repeatedly randomly sample from the test set for learning and guess sets, make guesses, and then calculate accuracies. At the end, the program will output the mean accuracy, positive ID rate, and negative ID rate.

This section can be found in the multi_test and test functions in the Detect module, as well as the respective code for learning and guessing.

3 Final Report

3.1 Challenges

At the very beginning of this project, the most pressing issue was determining a language that was optimal for our purposes as well as a data structure that would allow us to freely store email features in a way that we could map back characteristics to the original email. We decided to code in python because of its thriving ecology and its nature as an interpreted language. We were able to find past examples of NLP algorithms for reference and the language made it easy for running out of VMWare, bypassing some grief from technical difficulties. For our learning set, after parsing the excel spreadsheet generated by the scraping done with applescript, we generated a tuple of two dictionaries, one for events and one for non-events. Each dictionary follows the structure

```
{
  time: [ strings ]
  message: [ arrays of strings ]
  sender: [ strings ]
  subject: [ arrays of strings ]
}
```

Each key of the dictionary maps to an array, the entries of which are indexed identically so that the first entry of each array is the characteristic of the first email. We preferred this structure because it allowed us to map to each value to the original email while keeping all like characteristics clustered for easier computations.

Another obstacle was reliably obtaining large datasets that contained adequate numbers of event emails for both our learning phase and our testing phase. We researched methods for email scraping and eventually decided upon utilizing AppleScript to scrape emails off of the Apple Mail app. Considering which features were necessary, we chose to retrieve time, sender, subject, and message body.

Throughout the course of this project, we were challenged to find metrics that accurately defined an email as an event. At the same time, we could not narrow-mindedly focus on accurately inferring significant features in the learning set data without risking over-fitting and losing robustness in detecting events in future data sets.

We considered many characteristics as features for our Naive Bayes filter, including the presence of a time expression, word frequency, word presence, sender address patterns, length of email, presence of words generally associated with events and number of matches with significant words. In the end, after significant testing of all possible combinations of features, we discarded presence of a time expression and word frequency after analyzing performance over our test sets. The Features module will iterate through emails and check for the presence and prevalence of our characteristics of interest, creating an array of weighted values. Thus feature values for an set of test data is maintained as an array of arrays, again with indexing such that the nth array corresponds to the feature values of the nth email.

After the establishment of email features that we believed to be significant markers for event emails, we were faced with the task of normalizing data and converting it to probability values that could be utilized in a final Bayesian calculation. We considered multiple options, such as normalizing across number of emails or across email lengths. We finally settled on the number of distinct words. The first two methods either overweighted or underweighted longer emails so we decided upon normalizing by number of unique words which gave a balanced method of weighting our features. Intuitively, this makes sense because though longer emails may be expected to have larger word variety, we can expect that gain in variety would decrease as more and more words have already been seen in the email.

After normalizing, it was necessary to choose which distributions we would use in selecting the values to adjust our prior probabilities. In this phase of consideration, we opted for the Gaussian Distribution over the Exponential Distribution simply because performance was better using the Gaussian. Initially, we had considered the Exponential Distribution because we believed that the number of emails may have been right-skewed, but it became apparent that we had no guarantee of this data characteristic as our trials with the Gaussian Distribuiton performed better. Therefore, in our final implementation all values are calculated from the PDF of the Normal given as

$$\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/(2\sigma^2)}$$

where μ and σ are generated empirically as population statistics from our data vectors.

3.2 Project Experience and Takeaways

Working on this project was an informative and enjoyable experience, as we gained skills with Python, machine learning algorithms, designing our own modules and abstractions, and teamwork. A large challenge throughout the project was learning the syntax of Python and using it to implement the filter, since most of our teammates had little to no experience in the language. On top of understanding the statistics, it was very fulfilling to come out of this project with a better sense of another programming language. One important lesson learned from this project was that a complex model is not necessarily a good one. Machine learning appreciates the principle of Occam's Razor and many times we saw that a more parsimonious model with fewer features may more accurately model datasets than a model with more features. To some extent, this was a counterintuitive lesson in which valuing high accuracy on a particular test set could induce overfitting, leading to a poor model for future predictions on more diverse sets. In essence, this was a lesson of balance in finding the optimal number of characteristics that could also be generalized to data beyond what we had in our learning set and test set.

3.3 Contributions

While we all tried to have a role in each major component of the project, we eventually focused on more particular parts of the project, as follows:

Frances Ding:
Researched Naive Bayes classifiers and natural language processing, including strategies for feature selection. Helped write functions in Bayes module and functions for testing accuracy of predictions. Created video.

Raynor Kuang:
Wrote wrapper code. Wrote code for Bayesian and added utility code for parsing emails. Tested and implemented code for rare setting and feature calculation. Tested and implemented full code for accuracy.

Jimmy Lin:
Wrote prototypes for Bayesian and experimented with different stat distributions. Did preliminary research on the Naive Bayes filter and helped work out method to select p-values from features. Helped develop significant features.

Daniel Wang:
Wrote script for exporting emails to Excel files, wrote preprocessor code to parse email output into dictionaries of strings, and helped with labeling exported data

3.4 Future Advice

We learned that there were many possibilities for this final project - far too many to explore and definitely too many to implement. As with many projects, it is very beneficial to the team and project to meet early on. This is a good time to spend time exploring as many potential project paths as possible, outlining expectations within the group, and coming to a consensus on what the team wants to achieve for the project.

3.5 Future Considerations

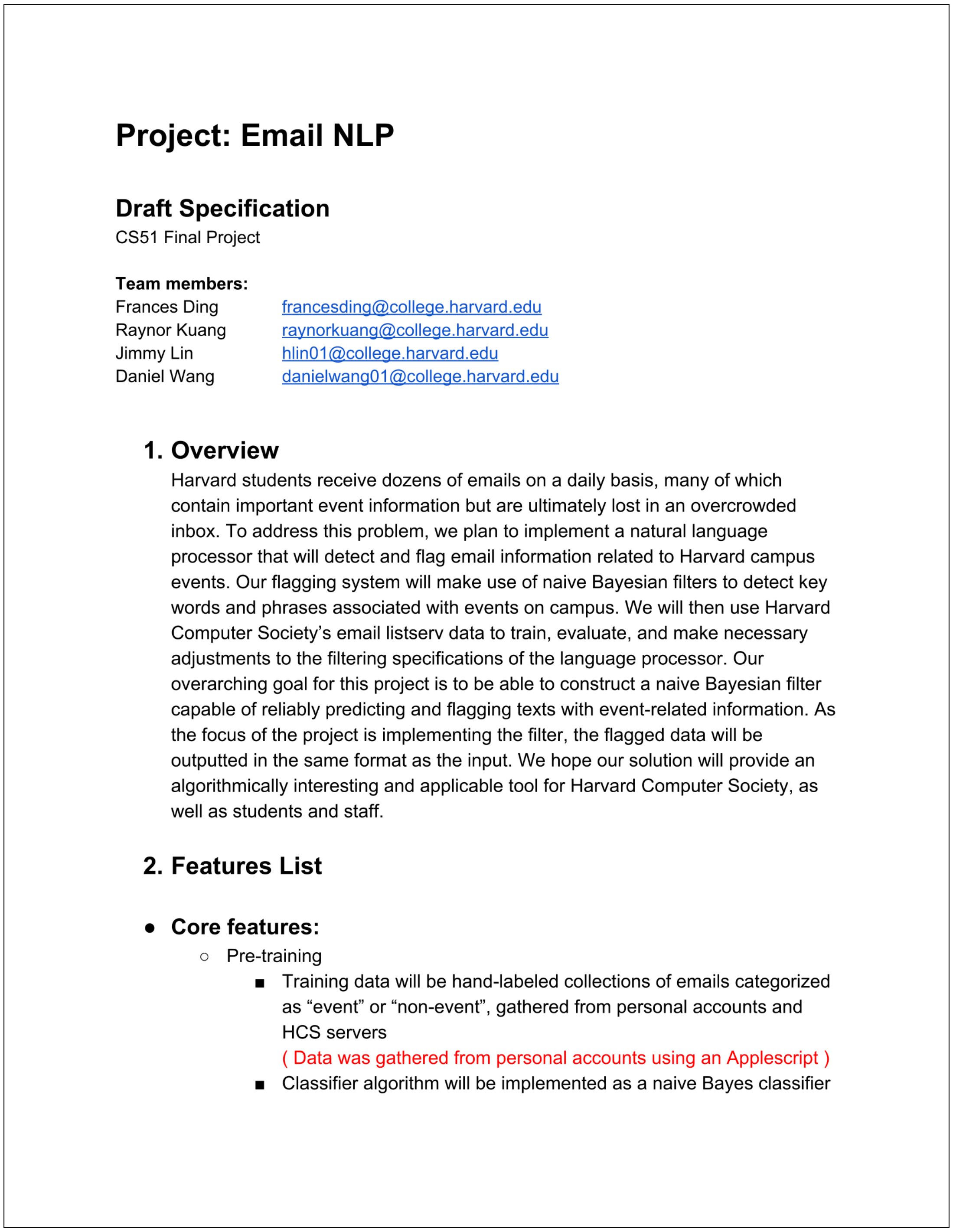
If more time had been available, it would have been interesting to implement other algorithms typical for text classification, particularly SVM. Ideally we could make comparisons between the different algorithms and their accuracy, as some software engineers claim that SVM is better than Naive Bayes, while others argue that Naive Bayes can achieve incredible accuracy if properly trained.

Additionally, next time we would have parsed our data in a way that would let us more easily retain patterns of words. We would have liked to gain inference from close proximity of different words, allowing phrases, dates, and times to be clustered as such. It also might have been informative to classify ”regions” in an email (i.e. beginning, middle, end). Our current model ignores most forms of punctuation and though we estimate that including punctuation would not alter prediction values significantly, it would be a worthwhile endeavor to confirm them as influential features or not.

4 Original Drafts and Final Specs

Attached are our original draft and final specifications. Annotations in red indicate how our final implementation reflects the plans we defined in each of these documents.

4.1 Original Drafts



Project: Email NLP

Draft Specification

CS51 Final Project

Team members:

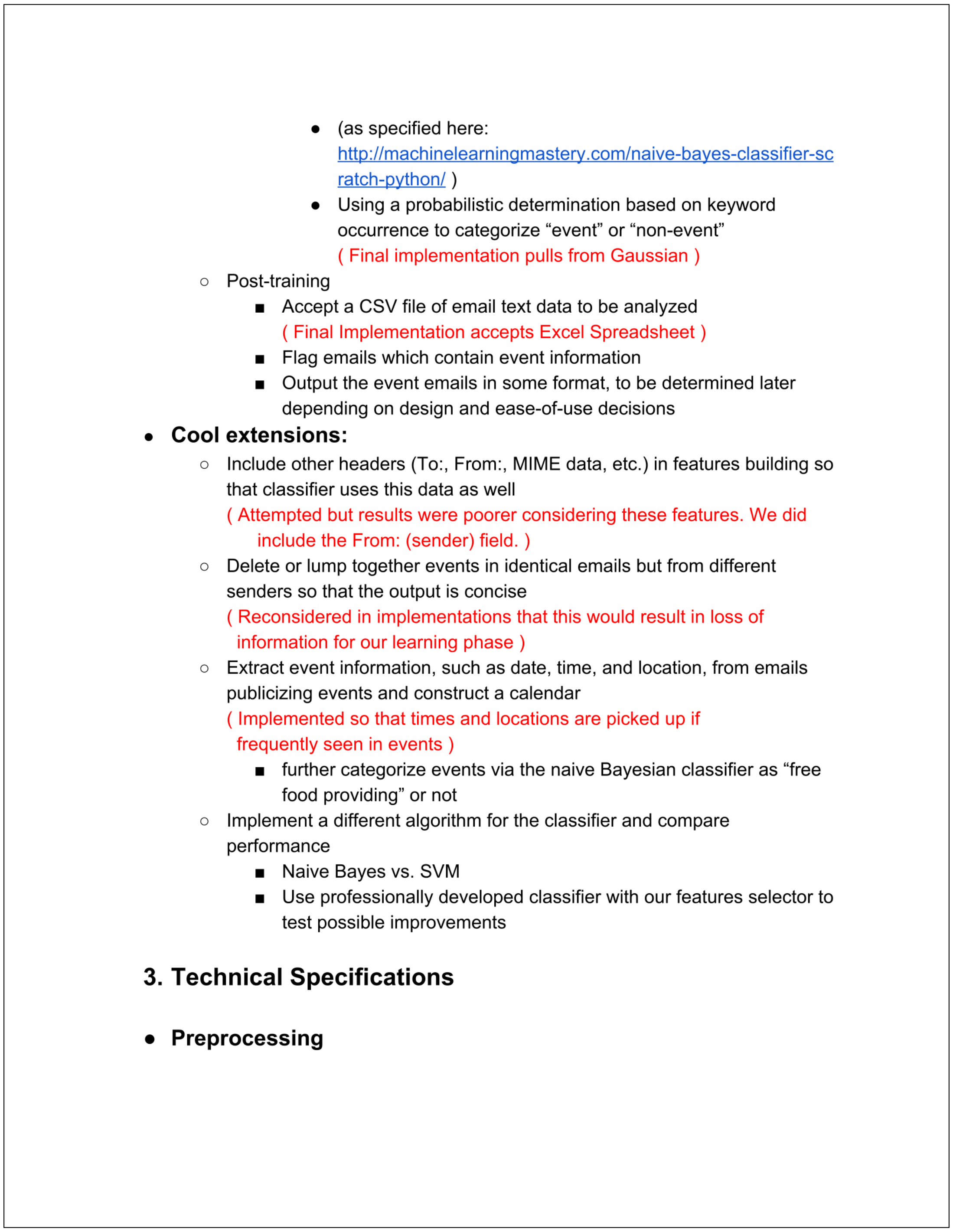
Frances Ding	francesding@college.harvard.edu
Raynor Kuang	raynorkuang@college.harvard.edu
Jimmy Lin	hlin01@college.harvard.edu
Daniel Wang	danielwang01@college.harvard.edu

1. Overview

Harvard students receive dozens of emails on a daily basis, many of which contain important event information but are ultimately lost in an overcrowded inbox. To address this problem, we plan to implement a natural language processor that will detect and flag email information related to Harvard campus events. Our flagging system will make use of naive Bayesian filters to detect key words and phrases associated with events on campus. We will then use Harvard Computer Society's email listserv data to train, evaluate, and make necessary adjustments to the filtering specifications of the language processor. Our overarching goal for this project is to be able to construct a naive Bayesian filter capable of reliably predicting and flagging texts with event-related information. As the focus of the project is implementing the filter, the flagged data will be outputted in the same format as the input. We hope our solution will provide an algorithmically interesting and applicable tool for Harvard Computer Society, as well as students and staff.

2. Features List

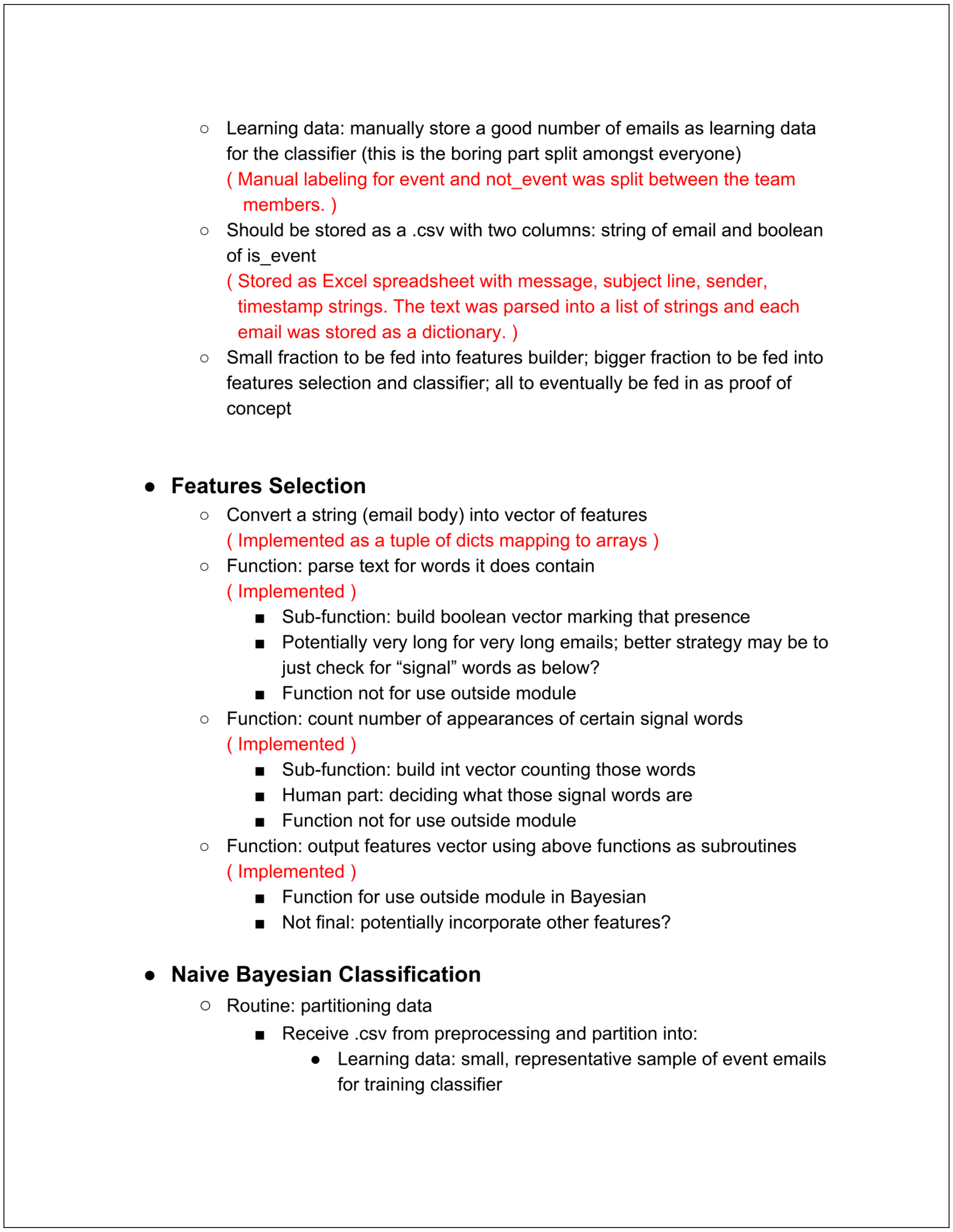
- **Core features:**
  - Pre-training
    - Training data will be hand-labeled collections of emails categorized as "event" or "non-event", gathered from personal accounts and HCS servers
      (Data was gathered from personal accounts using an Applescript)
    - Classifier algorithm will be implemented as a naive Bayes classifier



        - (as specified here: http://machinelearningmastery.com/naive-bayes-classifier-scratch-python/)
        - Using a probabilistic determination based on keyword occurrence to categorize "event" or "non-event"
          (Final implementation pulls from Gaussian)
    - Post-training
        - Accept a CSV file of email text data to be analyzed
          (Final Implementation accepts Excel Spreadsheet)
        - Flag emails which contain event information
        - Output the event emails in some format, to be determined later depending on design and ease-of-use decisions
- **Cool extensions:**
    - Include other headers (To:, From:, MIME data, etc.) in features building so that classifier uses this data as well
      (Attempted but results were poorer considering these features. We did include the From: (sender) field.)
    - Delete or lump together events in identical emails but from different senders so that the output is concise
      (Reconsidered in implementations that this would result in loss of information for our learning phase)
    - Extract event information, such as date, time, and location, from emails publicizing events and construct a calendar
      (Implemented so that times and locations are picked up if frequently seen in events)
        - further categorize events via the naive Bayesian classifier as "free food providing" or not
    - Implement a different algorithm for the classifier and compare performance
        - Naive Bayes vs. SVM
        - Use professionally developed classifier with our features selector to test possible improvements

3. Technical Specifications

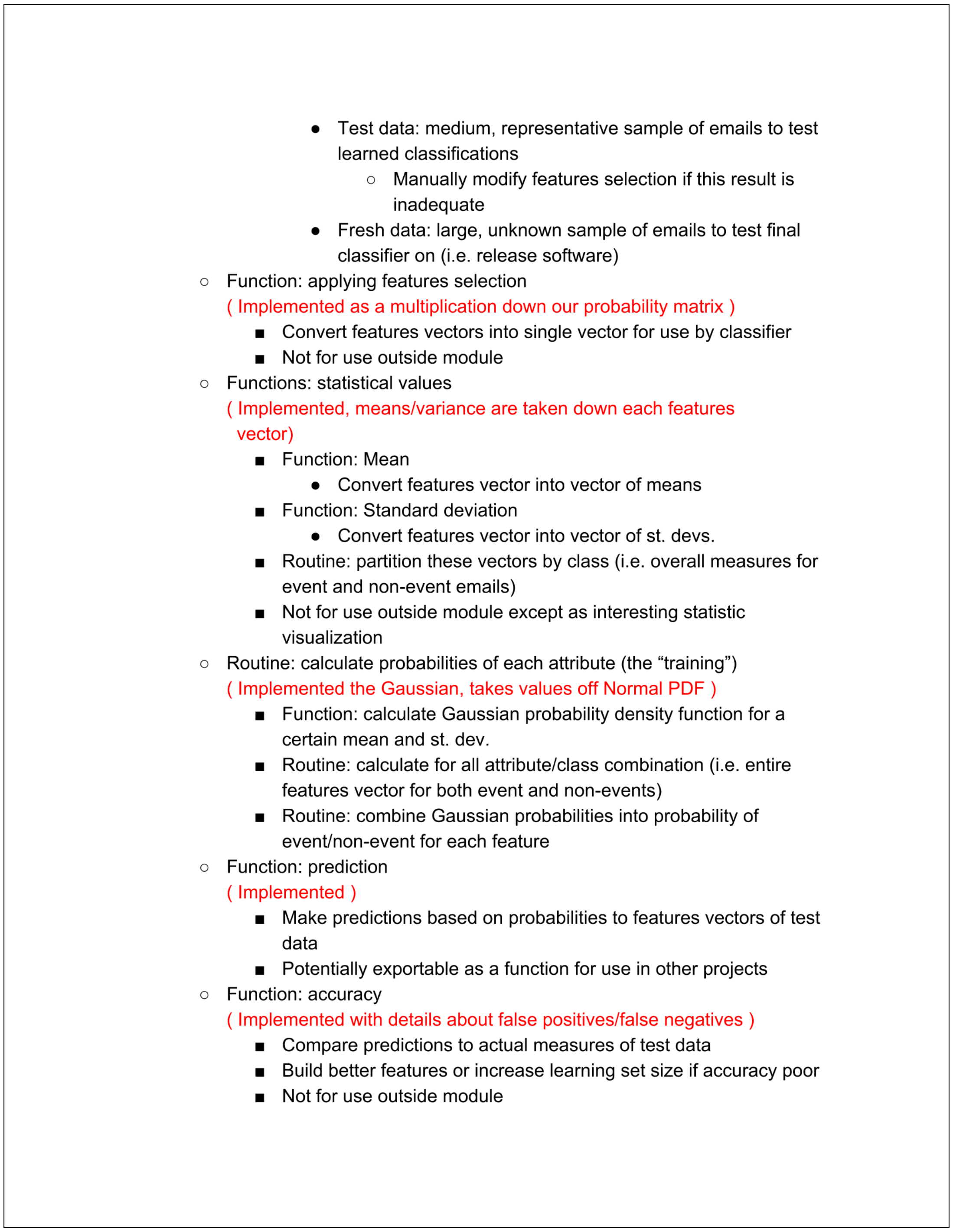
- **Preprocessing**



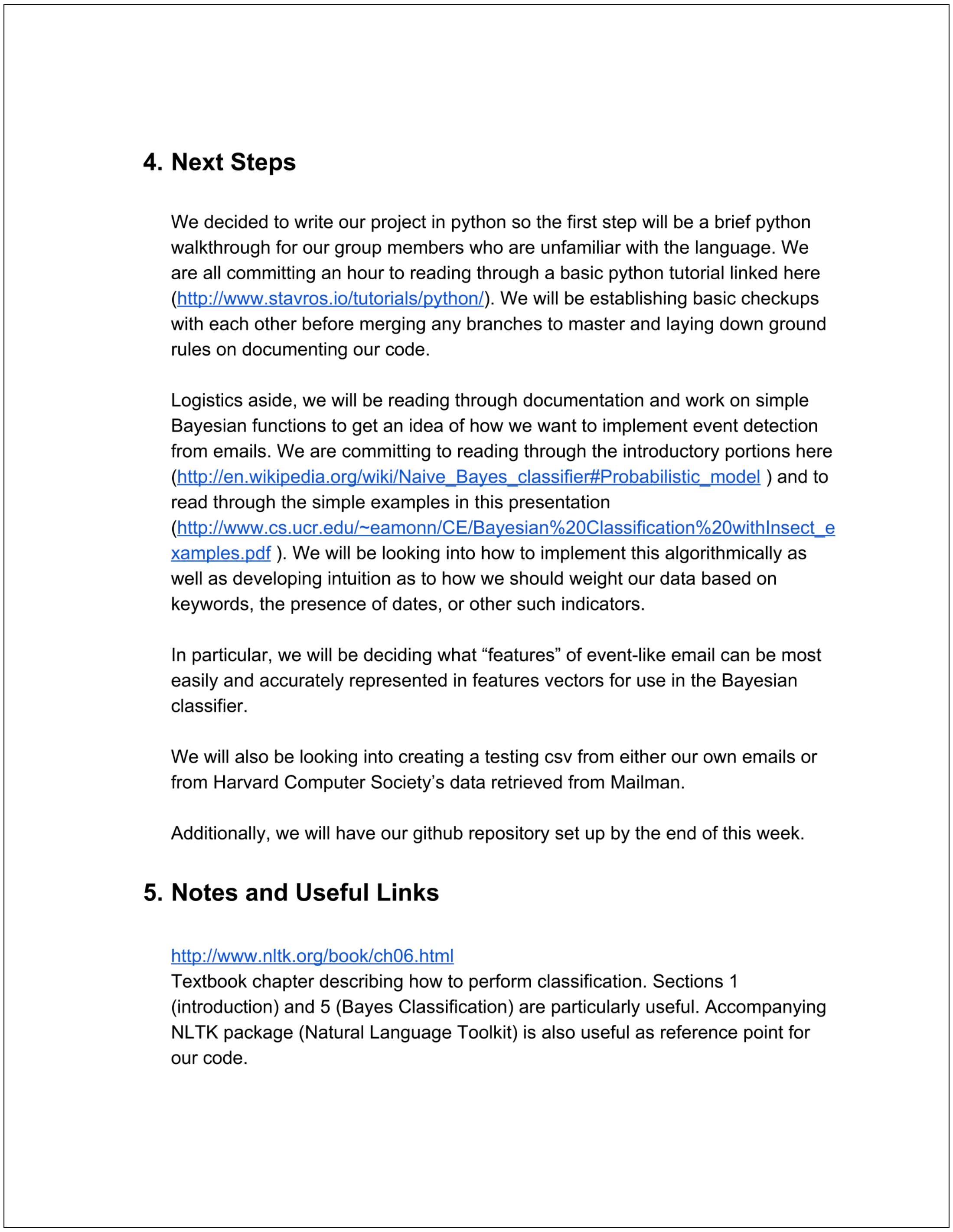
- Learning data: manually store a good number of emails as learning data for the classifier (this is the boring part split amongst everyone)
  (Manual labeling for event and not_event was split between the team members.)
- Should be stored as a .csv with two columns: string of email and boolean of is_event
  (Stored as Excel spreadsheet with message, subject line, sender, timestamp strings. The text was parsed into a list of strings and each email was stored as a dictionary.)
- Small fraction to be fed into features builder; bigger fraction to be fed into features selection and classifier; all to eventually be fed in as proof of concept

- **Features Selection**
  - Convert a string (email body) into vector of features
    (Implemented as a tuple of dicts mapping to arrays)
  - Function: parse text for words it does contain
    (Implemented)
    - Sub-function: build boolean vector marking that presence
    - Potentially very long for very long emails; better strategy may be to just check for “signal” words as below?
    - Function not for use outside module
  - Function: count number of appearances of certain signal words
    (Implemented)
    - Sub-function: build int vector counting those words
    - Human part: deciding what those signal words are
    - Function not for use outside module
  - Function: output features vector using above functions as subroutines
    (Implemented)
    - Function for use outside module in Bayesian
    - Not final: potentially incorporate other features?

- **Naive Bayesian Classification**
  - Routine: partitioning data
    - Receive .csv from preprocessing and partition into:
      - Learning data: small, representative sample of event emails for training classifier



        - Test data: medium, representative sample of emails to test learned classifications
            - Manually modify features selection if this result is inadequate
        - Fresh data: large, unknown sample of emails to test final classifier on (i.e. release software)
- Function: applying features selection
  (Implemented as a multiplication down our probability matrix)
    - Convert features vectors into single vector for use by classifier
    - Not for use outside module
- Functions: statistical values
  (Implemented, means/variance are taken down each features vector)
    - Function: Mean
        - Convert features vector into vector of means
    - Function: Standard deviation
        - Convert features vector into vector of st. devs.
    - Routine: partition these vectors by class (i.e. overall measures for event and non-event emails)
    - Not for use outside module except as interesting statistic visualization
- Routine: calculate probabilities of each attribute (the "training")
  (Implemented the Gaussian, takes values off Normal PDF)
    - Function: calculate Gaussian probability density function for a certain mean and st. dev.
    - Routine: calculate for all attribute/class combination (i.e. entire features vector for both event and non-events)
    - Routine: combine Gaussian probabilities into probability of event/non-event for each feature
- Function: prediction
  (Implemented)
    - Make predictions based on probabilities to features vectors of test data
    - Potentially exportable as a function for use in other projects
- Function: accuracy
  (Implemented with details about false positives/false negatives)
    - Compare predictions to actual measures of test data
    - Build better features or increase learning set size if accuracy poor
    - Not for use outside module



4. Next Steps

We decided to write our project in python so the first step will be a brief python walkthrough for our group members who are unfamiliar with the language. We are all committing an hour to reading through a basic python tutorial linked here (http://www.stavros.io/tutorials/python/). We will be establishing basic checkups with each other before merging any branches to master and laying down ground rules on documenting our code.

Logistics aside, we will be reading through documentation and work on simple Bayesian functions to get an idea of how we want to implement event detection from emails. We are committing to reading through the introductory portions here (http://en.wikipedia.org/wiki/Naive_Bayes_classifier#Probabilistic_model) and to read through the simple examples in this presentation (http://www.cs.ucr.edu/~eamonn/CE/Bayesian%20Classification%20withInsect_examples.pdf). We will be looking into how to implement this algorithmically as well as developing intuition as to how we should weight our data based on keywords, the presence of dates, or other such indicators.

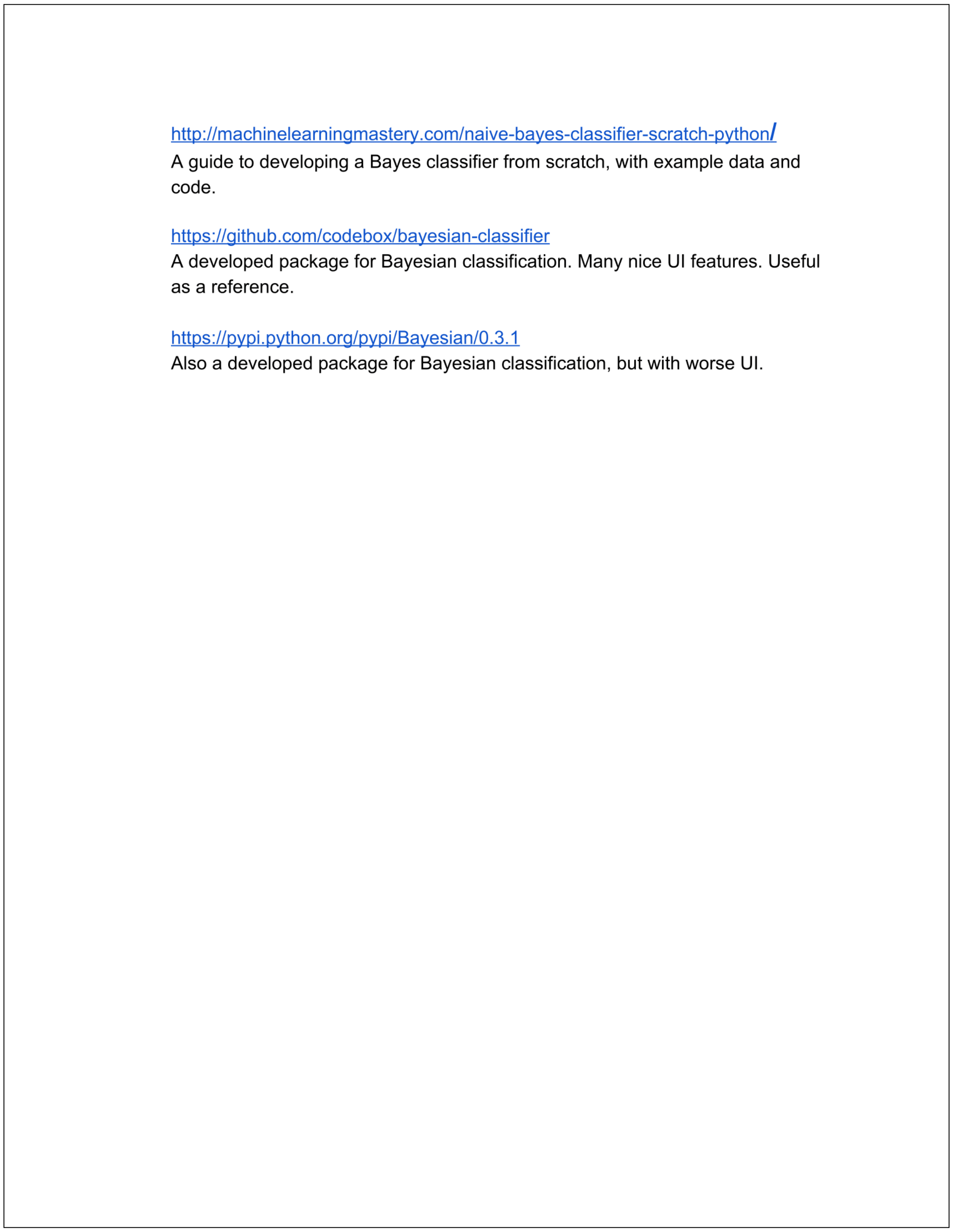
In particular, we will be deciding what “features” of event-like email can be most easily and accurately represented in features vectors for use in the Bayesian classifier.

We will also be looking into creating a testing csv from either our own emails or from Harvard Computer Society’s data retrieved from Mailman.

Additionally, we will have our github repository set up by the end of this week.

5. Notes and Useful Links

http://www.nltk.org/book/ch06.html
Textbook chapter describing how to perform classification. Sections 1 (introduction) and 5 (Bayes Classification) are particularly useful. Accompanying NLTK package (Natural Language Toolkit) is also useful as reference point for our code.

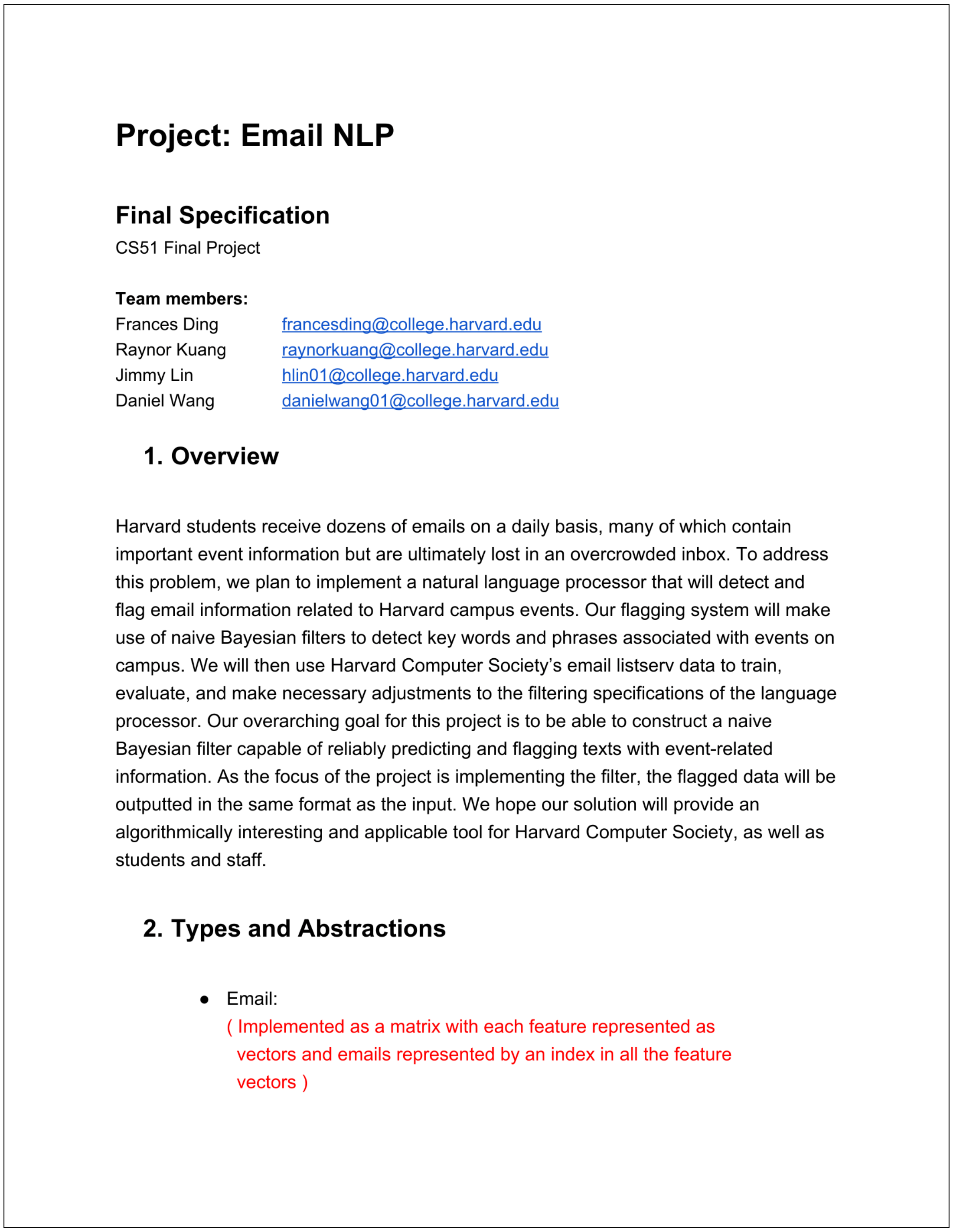


http://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
A guide to developing a Bayes classifier from scratch, with example data and code.

https://github.com/codebox/bayesian-classifier
A developed package for Bayesian classification. Many nice UI features. Useful as a reference.

https://pypi.python.org/pypi/Bayesian/0.3.1
Also a developed package for Bayesian classification, but with worse UI.

4.2 Final Specs



Project: Email NLP

Final Specification

CS51 Final Project

Team members:

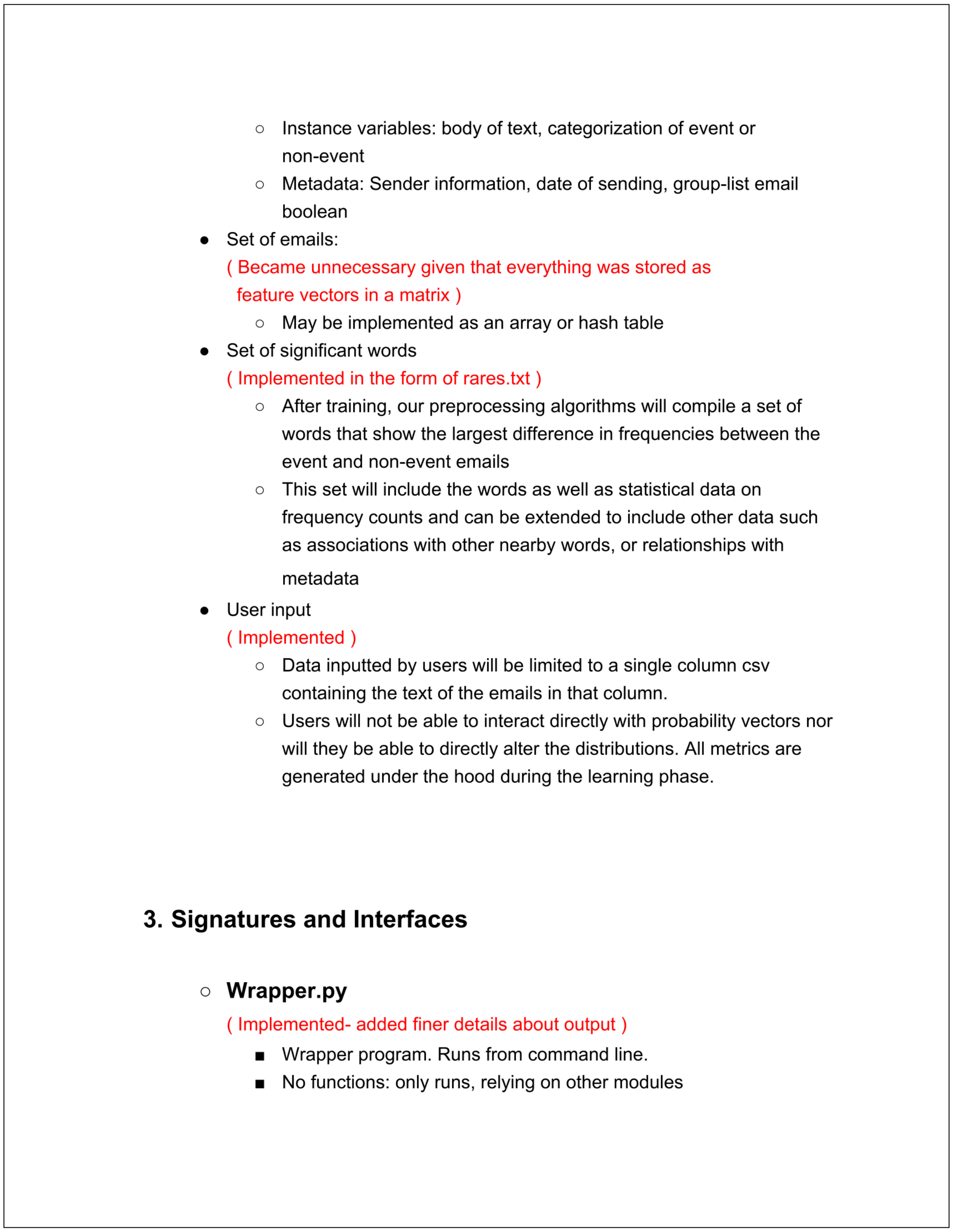
Frances Ding	francesding@college.harvard.edu
Raynor Kuang	raynorkuang@college.harvard.edu
Jimmy Lin	hlin01@college.harvard.edu
Daniel Wang	danielwang01@college.harvard.edu

1. Overview

Harvard students receive dozens of emails on a daily basis, many of which contain important event information but are ultimately lost in an overcrowded inbox. To address this problem, we plan to implement a natural language processor that will detect and flag email information related to Harvard campus events. Our flagging system will make use of naive Bayesian filters to detect key words and phrases associated with events on campus. We will then use Harvard Computer Society's email listserv data to train, evaluate, and make necessary adjustments to the filtering specifications of the language processor. Our overarching goal for this project is to be able to construct a naive Bayesian filter capable of reliably predicting and flagging texts with event-related information. As the focus of the project is implementing the filter, the flagged data will be outputted in the same format as the input. We hope our solution will provide an algorithmically interesting and applicable tool for Harvard Computer Society, as well as students and staff.

2. Types and Abstractions

- Email:
  (Implemented as a matrix with each feature represented as vectors and emails represented by an index in all the feature vectors)

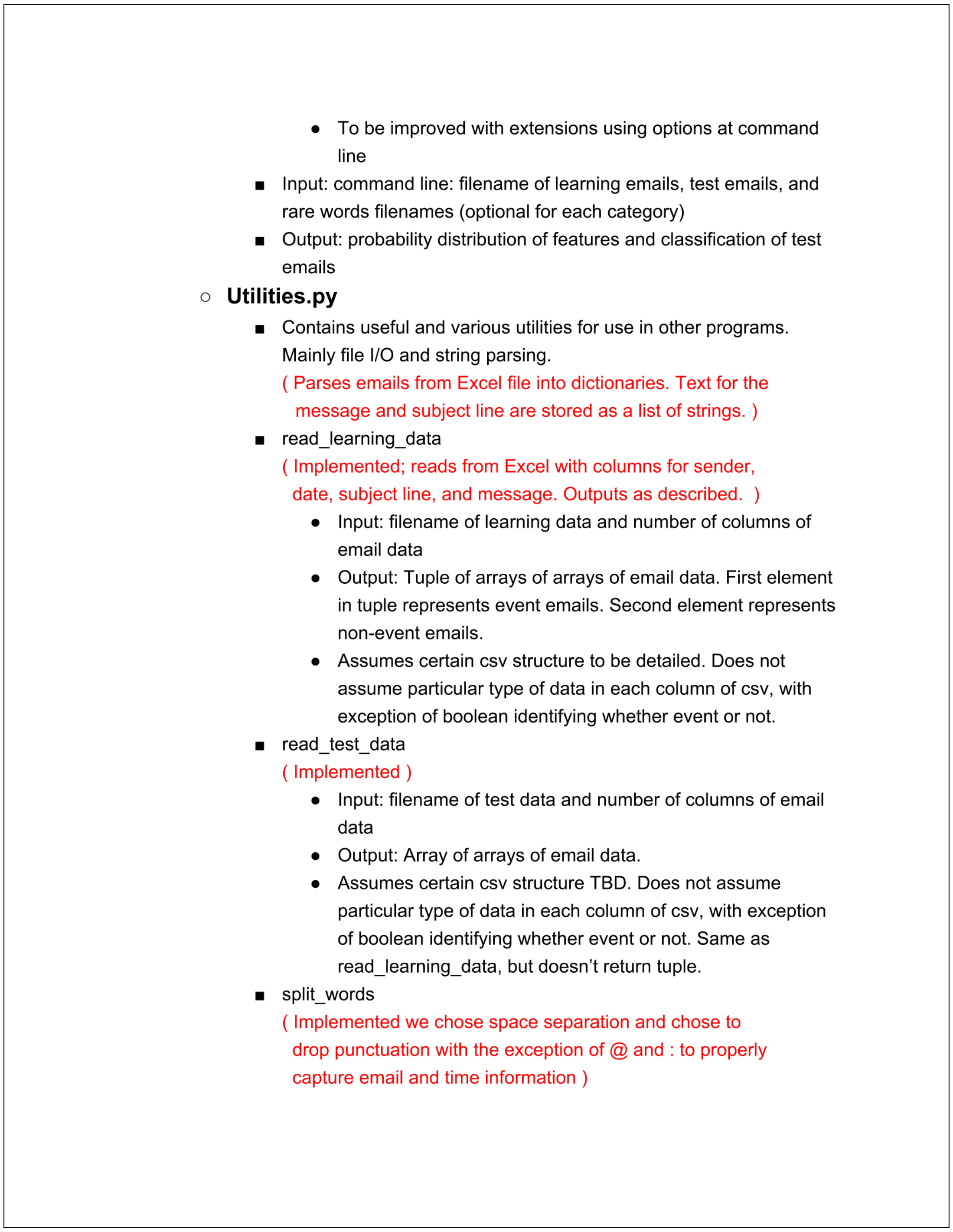


- Instance variables: body of text, categorization of event or non-event
- Metadata: Sender information, date of sending, group-list email boolean

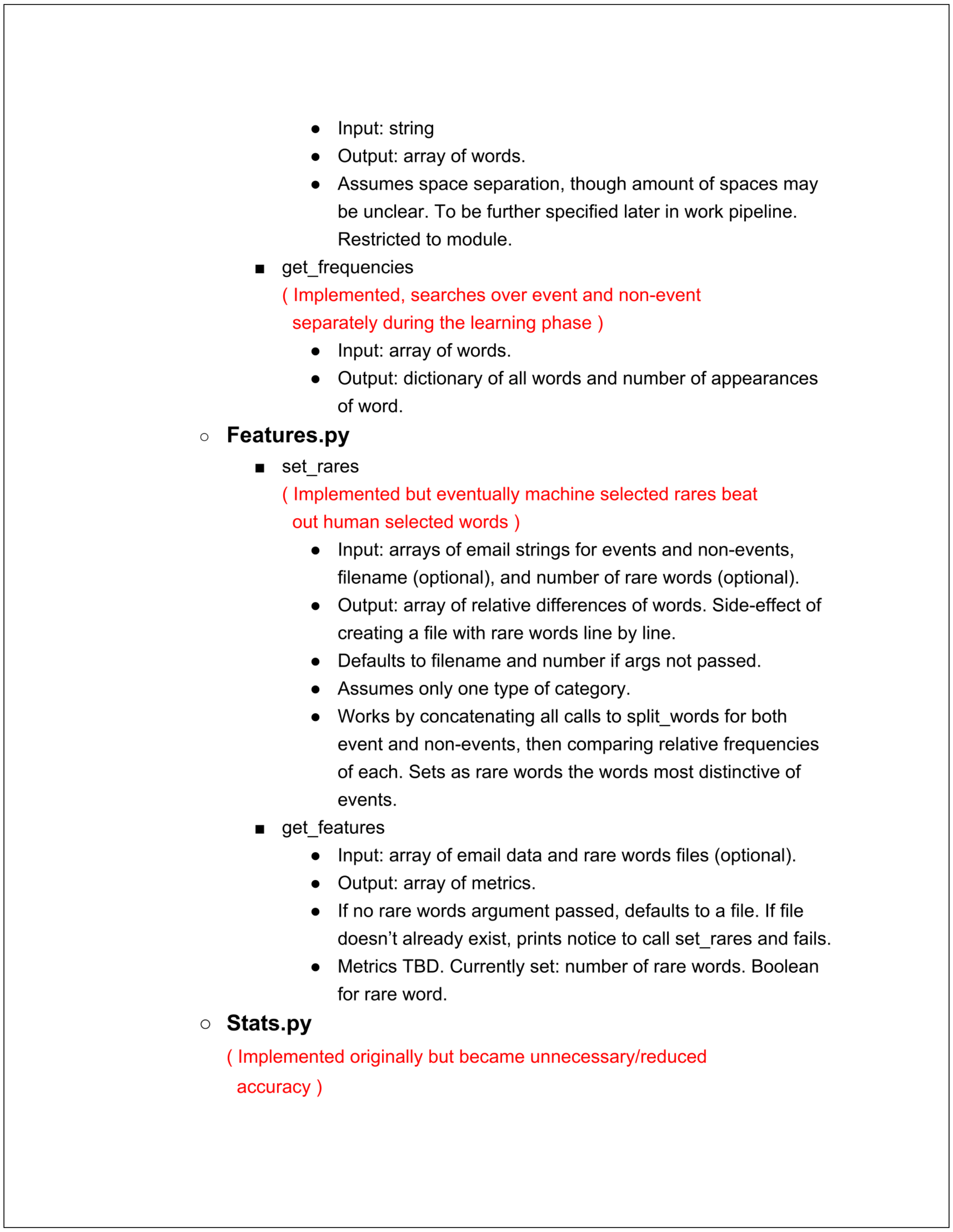
- Set of emails:
  (Became unnecessary given that everything was stored as feature vectors in a matrix)
  - May be implemented as an array or hash table
- Set of significant words
  (Implemented in the form of rares.txt)
  - After training, our preprocessing algorithms will compile a set of words that show the largest difference in frequencies between the event and non-event emails
  - This set will include the words as well as statistical data on frequency counts and can be extended to include other data such as associations with other nearby words, or relationships with metadata
- User input
  (Implemented)
  - Data inputted by users will be limited to a single column csv containing the text of the emails in that column.
  - Users will not be able to interact directly with probability vectors nor will they be able to directly alter the distributions. All metrics are generated under the hood during the learning phase.

3. Signatures and Interfaces

- **Wrapper.py**
  (Implemented- added finer details about output)
  - Wrapper program. Runs from command line.
  - No functions: only runs, relying on other modules



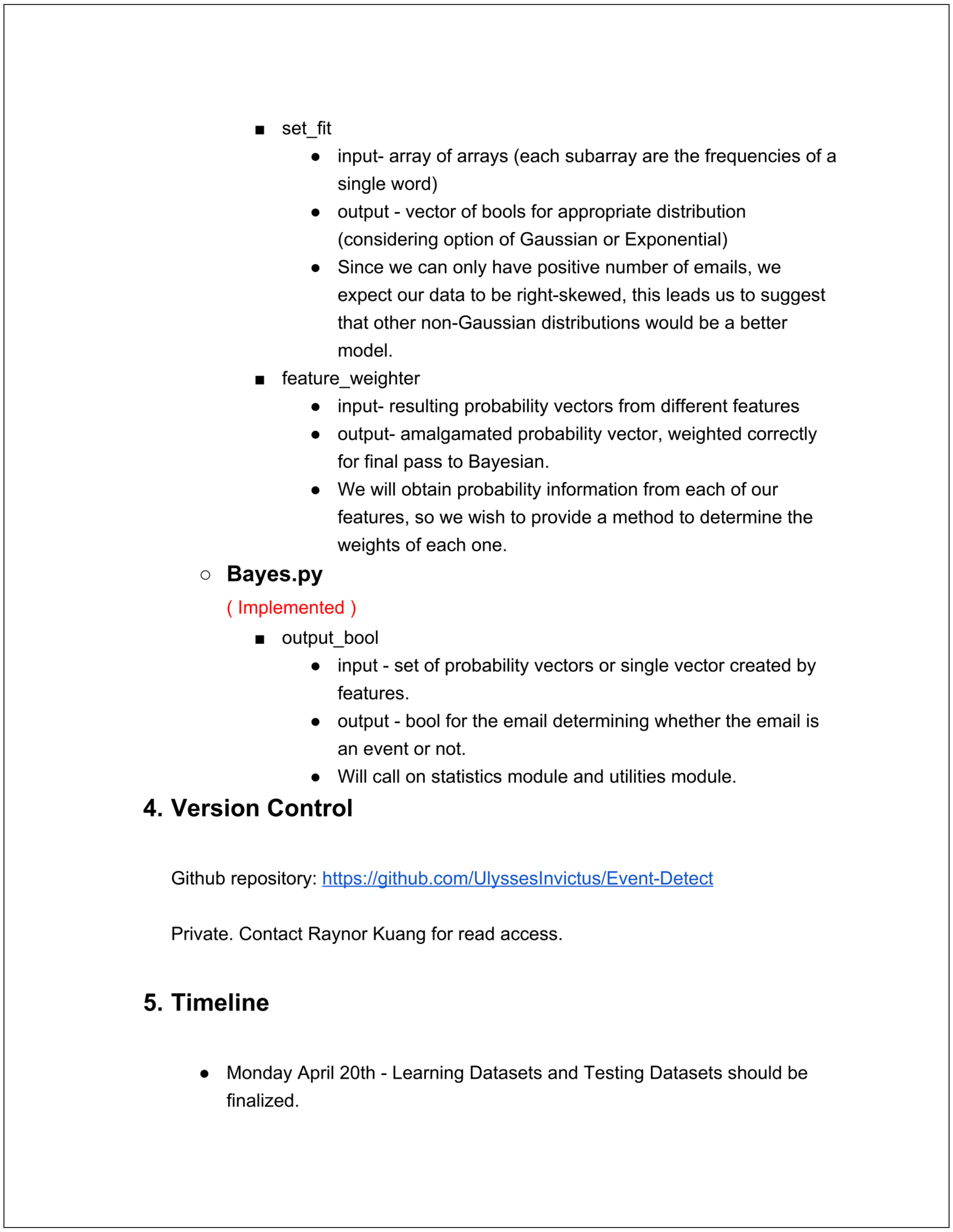
    - To be improved with extensions using options at command line
  - Input: command line: filename of learning emails, test emails, and rare words filenames (optional for each category)
  - Output: probability distribution of features and classification of test emails
- **Utilities.py**
  - Contains useful and various utilities for use in other programs. Mainly file I/O and string parsing.
    (Parses emails from Excel file into dictionaries. Text for the message and subject line are stored as a list of strings.)
  - read_learning_data
    (Implemented; reads from Excel with columns for sender, date, subject line, and message. Outputs as described.)
    - Input: filename of learning data and number of columns of email data
    - Output: Tuple of arrays of arrays of email data. First element in tuple represents event emails. Second element represents non-event emails.
    - Assumes certain csv structure to be detailed. Does not assume particular type of data in each column of csv, with exception of boolean identifying whether event or not.
  - read_test_data
    (Implemented)
    - Input: filename of test data and number of columns of email data
    - Output: Array of arrays of email data.
    - Assumes certain csv structure TBD. Does not assume particular type of data in each column of csv, with exception of boolean identifying whether event or not. Same as read_learning_data, but doesn't return tuple.
  - split_words
    (Implemented we chose space separation and chose to drop punctuation with the exception of @ and : to properly capture email and time information)



- Input: string
- Output: array of words.
- Assumes space separation, though amount of spaces may be unclear. To be further specified later in work pipeline. Restricted to module.

- get_frequencies
  (Implemented, searches over event and non-event separately during the learning phase)
  - Input: array of words.
  - Output: dictionary of all words and number of appearances of word.

- **Features.py**
  - set_rares
    (Implemented but eventually machine selected rares beat out human selected words)
    - Input: arrays of email strings for events and non-events, filename (optional), and number of rare words (optional).
    - Output: array of relative differences of words. Side-effect of creating a file with rare words line by line.
    - Defaults to filename and number if args not passed.
    - Assumes only one type of category.
    - Works by concatenating all calls to split_words for both event and non-events, then comparing relative frequencies of each. Sets as rare words the words most distinctive of events.
  - get_features
    - Input: array of email data and rare words files (optional).
    - Output: array of metrics.
    - If no rare words argument passed, defaults to a file. If file doesn't already exist, prints notice to call set_rares and fails.
    - Metrics TBD. Currently set: number of rare words. Boolean for rare word.
- **Stats.py**
  (Implemented originally but became unnecessary/reduced accuracy)



- ■ set_fit
    - input- array of arrays (each subarray are the frequencies of a single word)
    - output - vector of bools for appropriate distribution (considering option of Gaussian or Exponential)
    - Since we can only have positive number of emails, we expect our data to be right-skewed, this leads us to suggest that other non-Gaussian distributions would be a better model.
- ■ feature_weighter
    - input- resulting probability vectors from different features
    - output- amalgamated probability vector, weighted correctly for final pass to Bayesian.
    - We will obtain probability information from each of our features, so we wish to provide a method to determine the weights of each one.

○ **Bayes.py**

(Implemented)

- ■ output_bool
    - input - set of probability vectors or single vector created by features.
    - output - bool for the email determining whether the email is an event or not.
    - Will call on statistics module and utilities module.

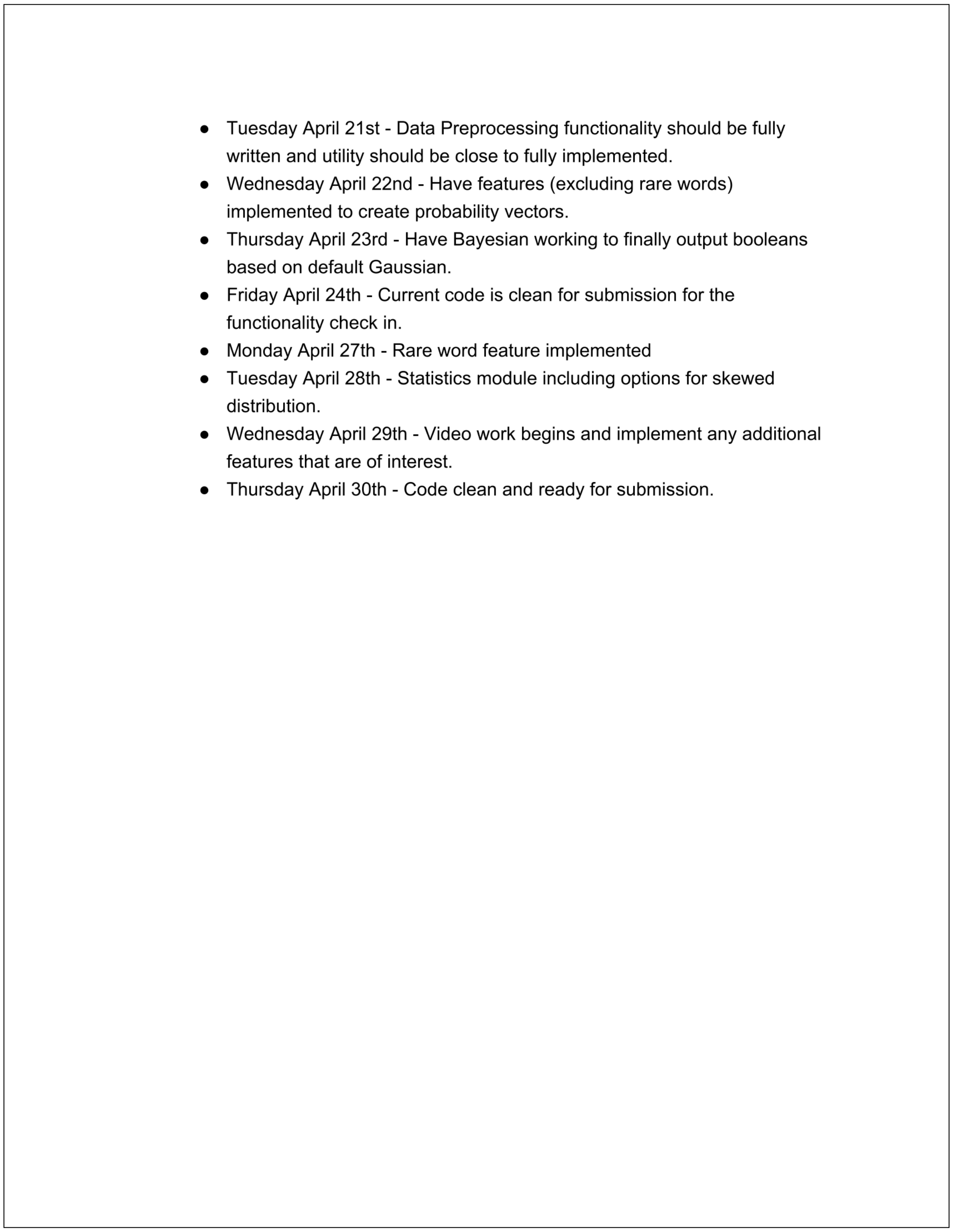
4. Version Control

Github repository: https://github.com/UlyssesInvictus/Event-Detect

Private. Contact Raynor Kuang for read access.

5. Timeline

- Monday April 20th - Learning Datasets and Testing Datasets should be finalized.



- Tuesday April 21st - Data Preprocessing functionality should be fully written and utility should be close to fully implemented.
- Wednesday April 22nd - Have features (excluding rare words) implemented to create probability vectors.
- Thursday April 23rd - Have Bayesian working to finally output booleans based on default Gaussian.
- Friday April 24th - Current code is clean for submission for the functionality check in.
- Monday April 27th - Rare word feature implemented
- Tuesday April 28th - Statistics module including options for skewed distribution.
- Wednesday April 29th - Video work begins and implement any additional features that are of interest.
- Thursday April 30th - Code clean and ready for submission.