CSC 240 Text Processing Project

Purpose

Many programming tasks require processing text from increasingly large data sources. This project is meant to exercise the design and programming tasks required to successfully complete any similar project that you choose to take on. In this project, you will produce at least one of each of the exemplars of our learning and program outcomes for this course. These exemplars must be applied for the purpose of text processing, since this is the focus of the project.

Deliverables

Group/Individual project submissions

Group/Individual presentation

Schedule

3/17 – 3/28: Form the group, explore data set, and research classification methods. Write down the plan, which include the UML for each of the class, some of the key function for the class, and the workload for each group member. The plan is due on 3/27. 3-5 groups will be selected for a presentation on 4/1.

3/31 – 4/4: Split the dataset into learning set and test sets. Then, only work on the learning set. Further split it up to Spam and Not Spam. Write code that automatically extract the feature from each set.

4/7 – 4/11: keep working on automatically extract features from the data set. Pin down 3-5 features that best differentiate Spam from Not Spam. Keep research on classification methods. 3-5 groups will be selected to present their algorithms that automatically generate the features from the dataset on 4/15

4/14 – 4/25: Try some of the classification algorithms and use test data set to test whether the accuracy of spam filter. 3-5 groups will be selected to present their final implementation of the classification algorithm and accuracy of the spam filter on 4/24.

4/28 – 5/1: Continue improve the spam filter. Submit implementation by 5/1.

Task Description

You will write a program to read the Spam or Not Spam Dataset (https://www.kaggle.com/datasets/ozlerhakan/spam-or-not-spam-dataset), and then process each email to create features for the email. The features could be the counts of each word, the number of words, the counts of each bigram, statistics about the size of each word, or any other feature about the email that you think may be relevant and that your program will be able to compute by processing the text. These features are ultimately going to be a list of values where each value is associated with a named feature. This can be accomplished in many ways. A very simplified tabular example is below (the first row is the feature name, and rest of the rows are the values, the first column is not a feature but just an id for the email):

email	a_count	the_count	is_count	day_count	bright_count	...
1	10	5	3	1	1	...
2	0	0	1	0	0	...

Notice that some emails may not have a value for the count of a particular word, so the full list of features isn't known until after all the emails are processed.

Once the features are created, your program should have a way to display the features of an individual email. It should also have a way to display a summary of the features of a group of emails. You will decide what this summary will look like. You could use basic statistics such as min, max, median, mean, and standard deviation. You could list which words appear in all the documents and which words appear in different fractions of the documents.

Since, this could be a lot of information, your program should save the email feature data and summary feature data to two separate csv files so that they could be read by a spreadsheet or statistics package. In addition, you may want to create a GUI for visualization.

In addition to creating the features, there should be a method that can take one set of email features and another set of email features and produce a numeric distance between the two emails. Similarly, a set of email features should be able to have a distance from the summary data of a group of emails.

Your program should be able to read in a model of spam and a model of not spam, then it should be able to read in a set of email features to determine the distance from each model, and it will classify the email as being spam if it is closes to the spam model or not spam if it is closest to the not spam model.

Stage 1 Planning

First, based on the Task Description above, create a list of requirements for your program. Next, for each requirement, describe specifically what the code will do for that requirement. Include UML for classes that will be needed. Include pseudocode and/or an algorithm description when it would clarify the requirement. Questions that you may want to answer (answers should be included in your plan):

1. How do you plan to read the .csv file and process the individual cells of data?
2. Which features do you plan to compute for each email?
3. How do you want to represent an email in code? Use a class? Just the features? Include the whole raw text? Should the known class (spam/not spam) be part of the email? What if the class (spam/not spam) isn't known?
4. How do you plan to save the computed features and summaries?
5. How do you plan to display the computed features and the summaries?
6. What distance metric do you plan to use for distance between two emails? Euclidean distance? Manhattan distance? Edit distance between features?
7. How will you compute distance from an email to the summary features?

Once you have a plan of requirements and how to code them, make a plan of how to test that each requirement is met. For a first step, one test per requirement is sufficient.

Finally, create a tentative schedule and work specification for the first draft and completion of each requirement.

Stage 2 Plan critique and revision

During the presentation of the plan, please write down comments, and discuss the plan with other groups. You should then revise your plan based on the discussion provided by your group members and other groups. It's okay to use ideas from a plan from other groups, just make sure to give them credit for the ideas. Also, implementation of the code will need to be your own.

The goal of the critique is to give each other guidance and encouragement to complete the project. Rather than saying something is missing, ask to expand on parts that could be clearer. If you don't understand part of a plan, then ask for clarification.

When evaluating other teams' project proposals, there are some high-level questions to consider:

1. Is the plan clear?
2. Does the plan identify
    a. an end goal?
    b. The steps to get to the goal?
    c. Risks to completing steps?
    d. Missing knowledge?
    e. Specific software requirements?
    f. A testing plan for those requirements?
3. Is the schedule reasonable?
4. Does the plan identify learning outcomes?

Finally, if there is any part of your colleagues' plan that you are interested in for your project, but you didn't include it, then you may want to ask your colleague to make a module that you could use. To do this, you would need to agree upon an API or a Java interface that can be used as a placeholder until the module is in working order.

Stage 3 Implementation

Once your plan is in place, you'll be spending the first half of the semester completing your implementation based on your schedule and work specification.

During the implementation phase, you may want to explore additional functionality in your project. It is ok to share implementation modules with other groups, however the implementations should be self contained. Integrating the module should not involve changing any code to the module. Instead, if the module is not working to a specification, then notify the author of the module, let them know your test results, and task them to provide a working module.

Any implementation that you didn't write should be documented as such, and the project should not rely on the implementation. That is, it should be supplemental to your program. The requirements in your plan should still be met without it.

To demonstrate that your program works, you will implement testing code based on your testing plan. In addition, your program will be given a training and a testing file generated from the spam/not spam data

set. The training file will have the label column, but in the testing file, the label column will not have a meaningful value. Your program will need to read the training file to create models of spam and not spam, and then create predictions for each email in the testing file. These predictions should go to a file called predictions.txt. There should be one prediction per line.

Exploring the data

This project is meant to create an end-to-end prediction system. However, most of the time a person analyzes the data along the way. If you are interested in doing some data exploration, you may start that once your program is able to make features and save them to a .csv file.

As a first step, you might try to import the .csv file into Excel, Numbers, or Google sheets. However, Exploratory, Weka, and R could be used for this as well. If you have questions getting started with this, please come to office hours.

Classification Methods

This section will describe 2 classification methods that can be used once you have models. The simplest method is to have your model consist of the entire data set. This is called a nearest neighbor approach. In a nearest neighbor approach, when a new email is tested, the distance between it and each email in your model is computed. Then the top neighbor or the top n neighbors are kept. The majority class of the n neighbors is the predicted class of the email. The next simplest method is to use the summary data for spam and the summary data for not spam as your models. You can use the distance from the email to the summary data, and use the closest summary to predict the type of email.

There are other classification methods you could try, but either of these are a good first step.

Submission

You will have one submission area for your project plan that will be shared with your group. All planning documents and critiques will be put there. When you submit your plan, create a directory called with the word Plan followed by your name such as Plan-Cui, and then place your plan file as a .pdf file or a markdown file (.md) if you know markdown. When submitting critiques, they should be in the same directory as the plan that they are critiquing.

You will have a submission area for your project implementation, that will be where you put your work. This should include your finalized plan, the program, the testing programs, and a self-assessment narrative write-up describing how much of your plan you completed, your prediction methods, and program accuracy, and which learning exemplars you completed and how they are shown in your project.