****

**User Guide for REHS: Text Classification via Supervised Machine Learning for an Issue Tracking System Software**

Nicholas Clark, Roxane Martin, Dustin Wu

**Required Libraries**:

Our software uses the following packages:

* numpy 1.14.5
* tqdm 4.23.4
* matplotlib 2.2.2
* scikit\_learn 0.19.2

The software runs on python 3. The packages can all be installed at once using the command “pip3 install -r requirements.txt”.

**File Description:**

Installation of the software should come with the following python files:

* main.py: contains of the driver programs to run the functions of other files.
* dataset.py: contains all of the functions involved in email collection, processing, and organization.
* classifier.py: contains all of the functions that train and use our implementation of a Naïve Bayes text classification algorithm.
* parse.py: contains functions that use regular expressions to extract information out of emails
* scikit\_classifier.py: contains all of the functions that train and use scikit-learn’s implementation of Naïve Bayes text classification algorithm.
* stemming.py: contains a python implementation of a stemming algorithm;
* table.py: contains the code for the data structure of our Naïve Bayes algorithm.
* visual.py: contains functions that generate statistical graphs.
* auto\_responder.py: contains functions that could be used to automatically respond to support tickets using the classifier. However, due to a lack of testing, it is still very much a prototype.
* flags.py: contains the global flag variables.

In addition, the installation should come with the directory named bin, which contains the text files named help.txt and stopwords.txt.

**Command Line Usage:**

python3 main.py [command][input][flag1] [flag2] ...

The order that flags appear in does not matter, but the command must come first. If there is an input associated with the command, it must come directly after the command, before any flags. Only one command can be run at a time, but multiple flags can be used.

**Workflow:**

**[1] manual:**

*dwnl → strp → catg → dict → tble → crsv*

**[2] automatic:**

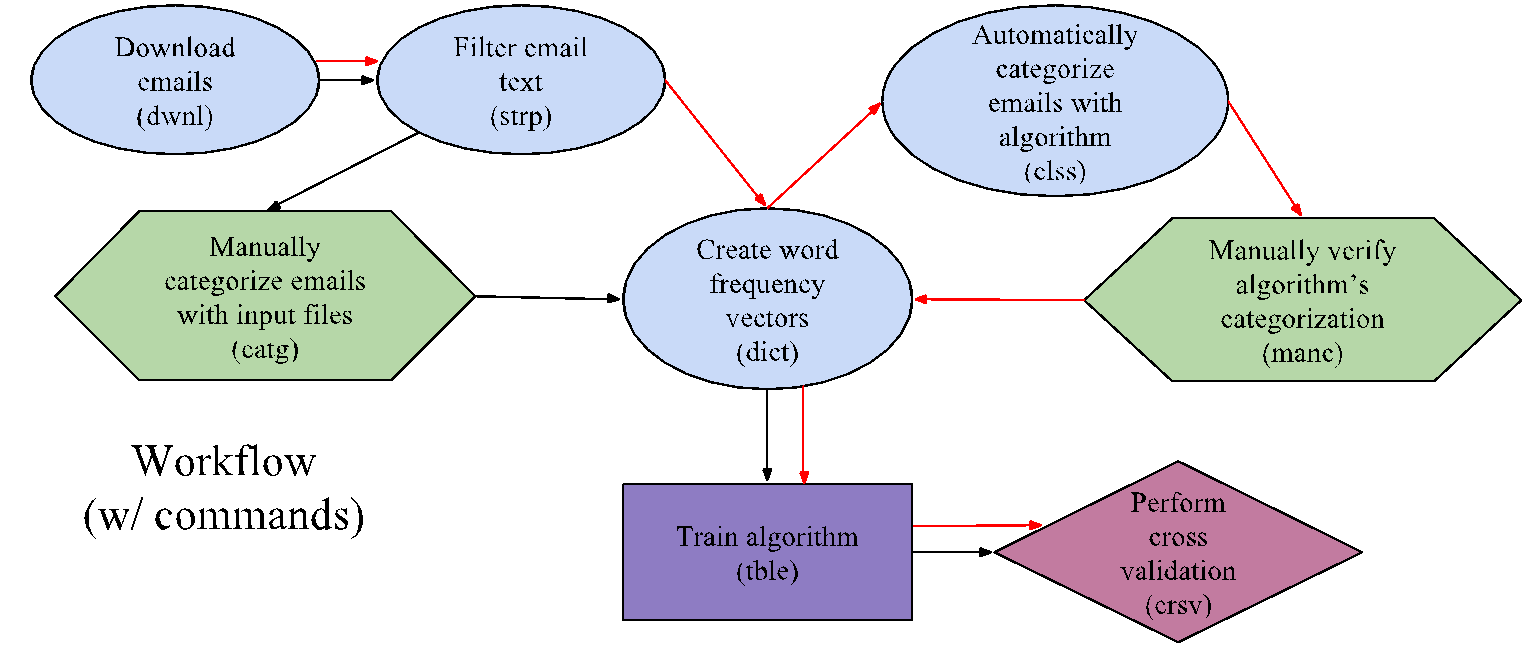
*dwnl → strp → dict → clss → manc → dict → tble → crsvcatg*

**[3] scikit manual:**

*dwnl → strp → catg → dict → stble → scrsv*

**[4] scikit automatic:**

*dwnl → strp → dict → sclss → manc → dict → stble → crsv*

See Figure 1 for a visualized workflow.

*Figure 1: The workflow of our software. Workflow [1] arrows are in black, workflow [2] arrows are in red.*

To build a dataset and train the model, run workflow [1]. The process of this workflow can be described as:

1. Download the emails with the *dwnl* command.

2. Strip out unnecessary words from the emails using *strp*.

4. Organize emails into category directories using manually-created input files and *catg*.

5. Create feature vectors out of the stripped email bodies using *dict*. Also create the global/master feature vector using the training set emails.

6. Using the feature vectors created by *dict* and the file organization created by *catg*, train the model using *tble*.

7. Test the accuracy of the model using the *crsv* command. Alternatively, the *list* and *acrc* commands can be used.

Once the model has been trained, workflow [2] can be used to expand the training set. This workflow which uses the model to automatically classify new emails. The process of this workflow can be described as:

1. Download new emails with the *dwnl* command.

2. Strip out unnecessary words from the emails using *strp*.

3. Create feature vectors out of the stripped email bodies using *dict*. The global dictionary will not be affected by this step, since these emails have not yet become training data.

4. Automatically classify the emails into categories using the trained model and *clss*.

5. Manually verify the correctness of the classifications of *clss* using *manc*.

6. Recreate the global feature vector with the new training data using *dict*.

7. Train the model using the new training data created by *manc* using *tble*.

8. Test the new accuracy of the model using the crsv command. Alternatively, the *list* and *acrc* commands can be used.

To re-train the model, modify the category text files as needed and rerun the *dict* and *catg* commands with the -re flag enabled. Then, rerun the *tble* command, which will re-train the model with the new categories.

Workflows [3] and [4] operate like workflows [1] and [2] but using the scikit-learn implementation of the model instead.

**File Organization:**

As the software is run, the following directory structure will develop. Unless otherwise stated, these directories will be automatically created by the software.

* **UID:** Each email is represented with its own directory. Its name takes the form “UID#X”. UID stands for “universal identifier”, and serves as a unique identifier for each email that gets downloaded. Each UID directory contains a few text files, one containing the header, one containing the body, and one containing the subject.After the *strp* command is run, two new text files will appear in each email: one containing the filtered body text, ‘body\_stripped.txt’, and another, ‘info.txt’ containing miscellaneous information that was parsed from the email, such as the original email address, the sender’s name, and any phone numbers.
* **Ticket:** Each support ticket is represented with its own directory. Its name takes the form “ticket#X”. Because a support ticket can contain multiple emails, the first “issue” email and the later “response” emails, each ticket directory contains multiple emails.
* **Emails:** The directory named “emails” is where downloaded emails are first placed. It contains ticket directories. As UID directories are downloaded one-by-one, they are placed into their respective ticket directory. Only the first “issue” email with the earliest UID is of interest.
* **Extra Emails:** A ticket directory can be placed into the directory named “emails/extra” for two reasons. The first is if the *strp* command finds that the ticket directory’s earliest UID email is not actually the first “issue” email (due to the lack of a secondary header). The second is if, during the *manc* command, the user finds that the ticket is not part of any current category. For the former reason, the ticket’s UID is placed in the special “extra” category in the file categories/extra.txt, which means that the *catg* command will permanently designate the email as part of the “extra” category. For the latter reason, this is not the case, so the user can use the *ucat* command to place the “extra” ticket back into the emails for reclassifying.
* **Training Emails**: The directory named “training\_emails” is where the emails used as training data, referred to as “training emails,” are contained. This directory contains issue categories filled with ticket directories. Emails in the “emails” directory may be classified into the “training\_emails” directory in two ways: one, through the *catg* command; and two, through the *clss* and *manc* commands.
* **Testing Emails:** The directory named “testing\_emails” is where the emails classified by the model are placed after the *clss* command is run. The name is somewhat misleading as the user can only use training emails, and not testing emails, to test the accuracy of the model. Emails are placed here because the model may not classify perfectly, and the model should not be trained with its own classifications. To promote emails in this category to the “training\_emails” directory, run the *manc* command.
* **Bin:** The directory named “bin” contains various files used by the software, such as the stopwords and the usage guide text files. It should come with the installation.
* **Categories:** The directory named “categories” is where the user should place with category text files that will be used to promote the training data by the *catg* command. It should come with the installation.
* **Graphs:** The directory named “graphs” is where the software will place statistical graphs created by the *crsv* and *bcsv* commands.

**Commands:**

***dwnl*:**

Downloads emails from an office 365 email address (specified in main.py) and organizes those emails into directories. When run, the user will be prompted for the email account’s password. The 'emails' directory created will contain ticket directories corresponding to each ticket ID downloaded. Each ticket directory contains a set of UID (universal identifier) directories, the earliest of which represents the earliest email of that ticket.

Ticket directories are named ticket#X and hold all of the emails in a user support ticket, including responses. Each of these emails is represented by a UID directory named UID#Y, which holds the subject, body, and header of the email. X and Y are natural numbers that increment as more emails are downloaded.

In order to jump to the most recently downloaded email, a number in 'last\_num.txt' updates after downloading is complete. If the -re flag is enabled, the last-num feature is disabled and every email will be redownloaded.

Downloading emails can take up to 1-3 hours. If control-c is used to quit out of the downloading process, the command will save the last-num but an email may not have finished downloading. It’s recommended to run the *dwnl* command in the background with the –nl flag enabled.

**strp [FILENAME1.txt] [FILENAME2.txt] ...:**

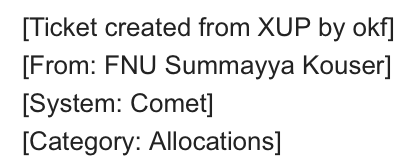
Strips downloaded email body text, removing words that occur in stopwords.txt and the input .txt file. The resulted “stripped body” is stored in body\_stripped.txt.

It also removes characters like punctuation and digits, as well as urls and file paths. Strip text also implements a classic stemming algorithm, the Porter stemmer, which shortens words to their stem as a way to combine different verb forms and related words in the dictionary frequency count. If the –re flag is enabled, body text will be re-stripped, otherwise emails whose body text has already been stripped will be ignored.

The strip text method relies on a “secondary header” being present in each email. If the secondary header is not found, the strip text method will designate that email as an “extra” by adding its ID to a category text file called “extra.txt”. This will cause the *catg* command to move this email to the extra directory located in the emails directory. See figure 2 for an example of a secondary header.

Note that only emails in the emails directory will be stripped. This means that if the user wants to re-strip an email already in a category, the *ucat* command must be run first.

Stripping is necessary in order to limit the number of words that appear as input for the text classification model.

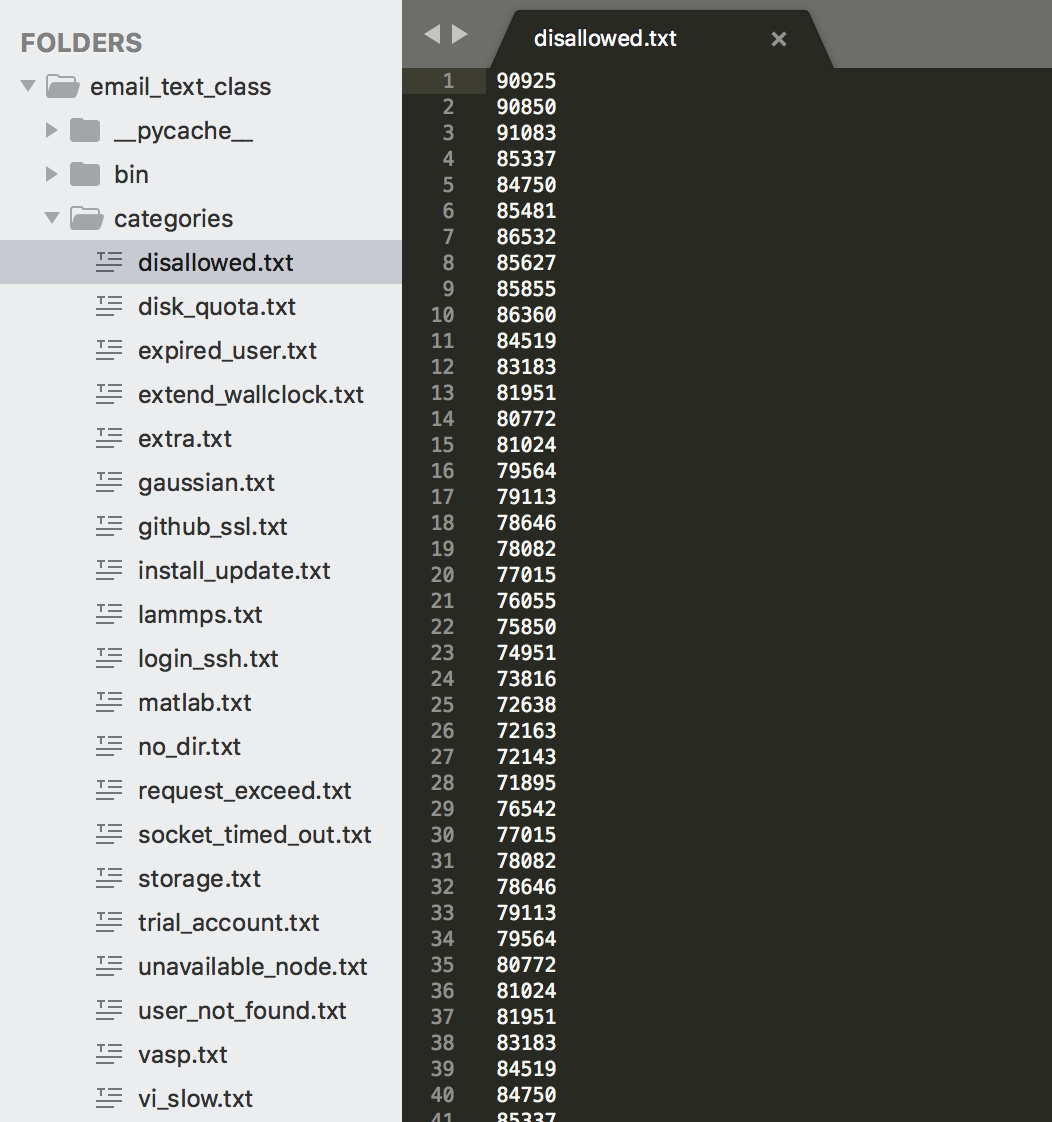


*Figure 2: An example of a secondary header*

**catg:**

Moves ticket directories into category directories using the .txt files in the directory named "categories". These files should contain the ticket ids that belong to a specified category, which is referenced from the name of the file. These ticket directories are moved to category directories in the training\_emails directory.

This command is in charge of organizing email data, creating the training data. The user must do the work of manually training the model by writing category text files. The installation of our software should contain a categories directory with some category files already written. See Figure 3 for an example of a category text file.



*Figure 3: An example category text file, located in the categories directory.*

**dict:**

Writes python dictionaries containing the frequency of each word, in the form of text files. A 'global' dictionary.txt will be created in the main directory with the total word frequencies of every email in the training set. A 'local' dictionary.txt will be created inside of each earliest UID directory containing the word frequencies of each stripped body file.

Note that an email must have a stripped body file for the command to write it a dictionary, so make sure to run the *strp* command before this command.

**tble:**

Creates a table using dictionaries of training\_emails emails created by the *dict* command by default, and stores it in a .p (pickle) file. This is the command that is responsible for training the model.

Each row of this table represents an email that was given as input via the *catg* command. Each column is an integer representing the occurrence of a particular word in that email’s stripped body. Thus, the table can be seen as a collection of the feature vectors of each email’s stripped body.

**clss:**

Automatically categorizes emails in emails directory using the trained model. Categorized emails will be transferred into the test\_emails directory, where they will *not* serve as training data until approved by the manc command.

This command depends on the frequency vectors/dictionaries to categorize emails. If an email does not have a dictionary it will not be categorized. Thus, the commands *strp* and *dict* must be run before this command.

If none of the words in the email appear in the training data, the classifier will not be able to classify the email. In this case, the email will be left in the emails directory.

**manc:**

Prompts the user to verify if the classifications done by the *clss* command were correct. For each improperly classified email, it will prompt the user for the proper category, which can only be either one of the pre-existing categories in training\_emails, or, if the email does not belong in any category, the command will move the email into the “extra” directory.

The email will then be transferred to its proper category in training\_emails and will be considered fit for use as additional training data. Once this is done, the ticket ID of this categorized email will be appended to its respective category.txt file.

**ucat:**

Uncategorizes emails in both test\_emails and training\_emails. This command also uncategorizes emails in the “extra” directory. Thus, if an email were put into the extra category by the *manc* command, the *ucat* command will place it back into the emails directory, allowing it to be reclassified.

Due to the fact that *manc* updates the category.txt files after an email is categorized, the *ucat* command does not undo the work done by *manc* to promote emails to training data.

Note that after running the *ucat* command also re-run the *catg* command, since *ucat* uncategorizes all the training\_emails directories.

**list [PERCENT]:**

Randomly selects PERCENT% of the training emails to test against the algorithm. Creates a text file called testing\_list.txt which stores the emails chosen to be part of the test set, and also "blindfolds" the algorithm to these emails. If no PERCENT is given, defaults to 10%.

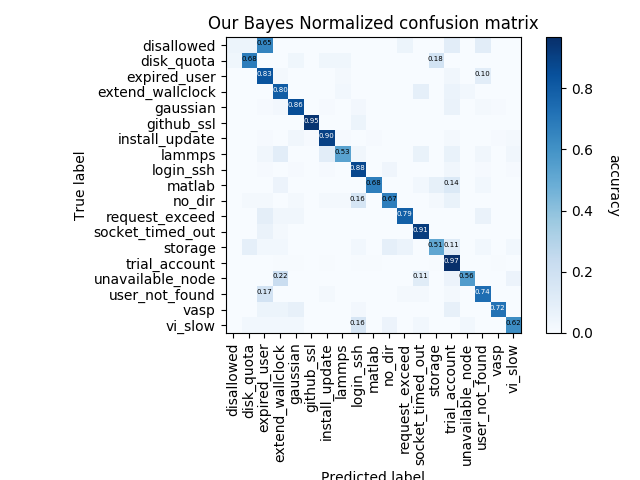
The *list* command was kept separate from the *acrc* command so that we could share testing\_list files with each other to compare model performance.

**acrc:**

Runs a single test on the accuracy of by training the model using all of the training emails EXCEPT the ones chosen by the *list* command and then using those in testing\_list.txt as test data. Since the *acrc* command depends on the *list* command, running *acrc* repeatedly without rerunning the *list* command each time will produce no effect on the resulting accuracy results.

**crsv [K]:**

Performs a K-fold cross validation to test the accuracy of the algorithm by temporarily retraining the algorithm on smaller sets of training data. For example, a 2-fold cross validation splits the data into 2 sections, trains the data on 50% and tests it on the other 50%, then retrains the data on the latter 50% and tests it on the first, finally averaging these accuracies and obtain a more reliable overall accuracy. Then, if the -vi flag is enabled, the command builds a confusion matrix displaying accuracy statistics for each specific category. See figure 4 for an example of a confusion matrix.



*Figure 4: An example of a confusion matrix.*

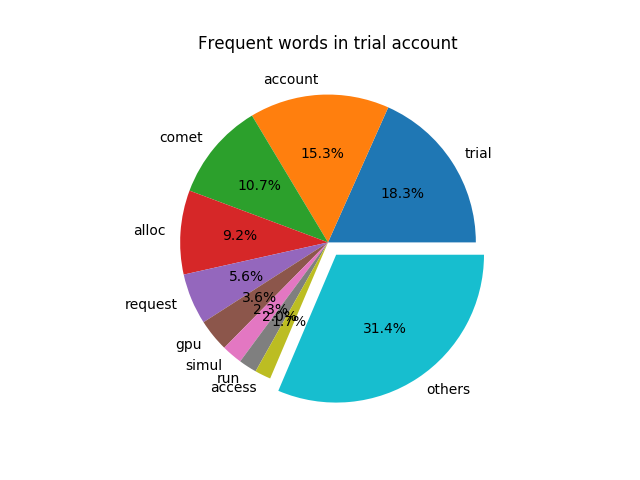
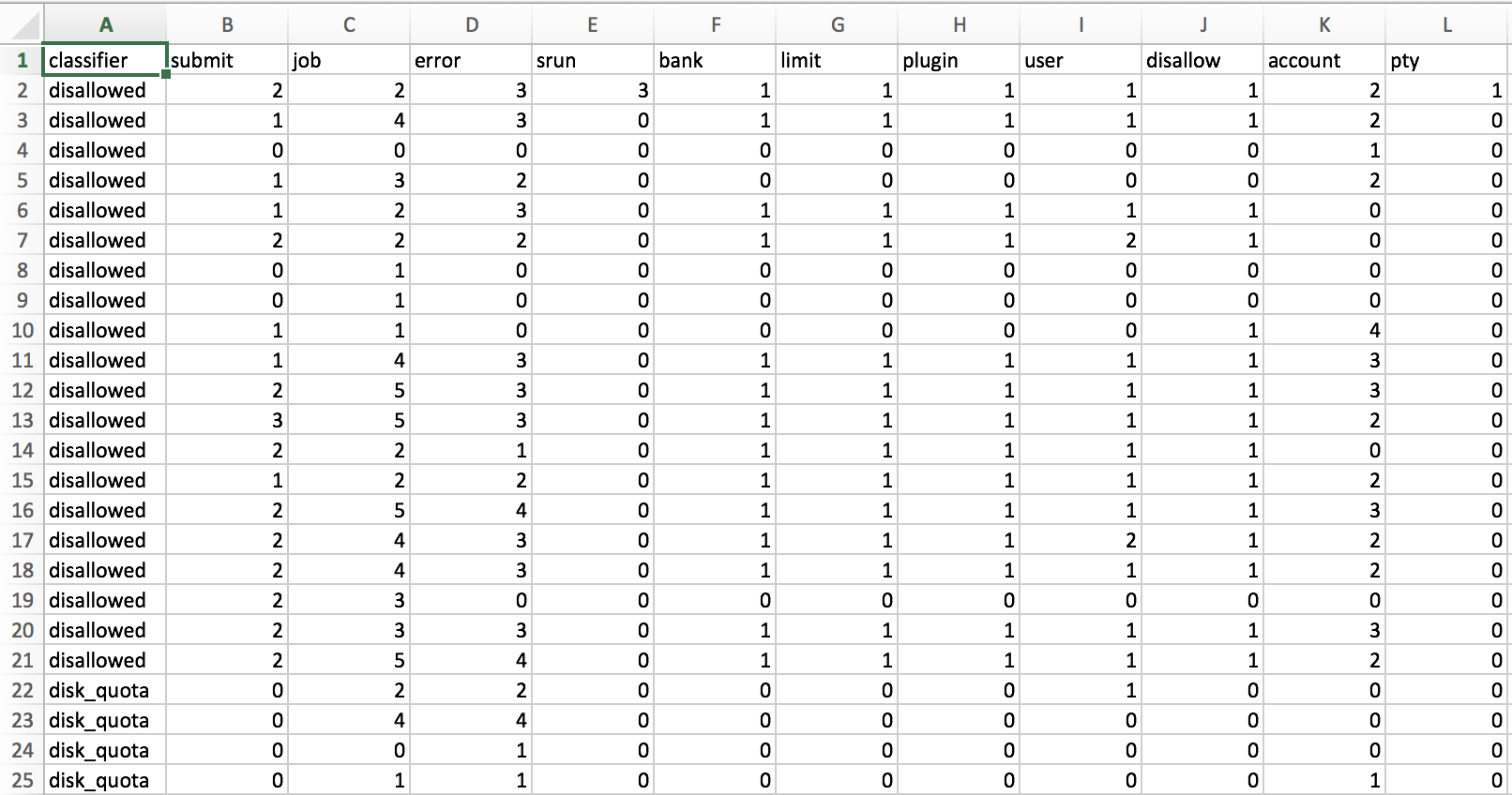
**bcsv:**

Makes a .csv file representing the table’s current trained state, which can be opened and examined by a program like Excel.

Then, if the -vi flag is enabled, the command then uses that file to build png file pie charts displaying the top 9 most common words found in the emails of each category.

Running the *list* command will change the amount of training data that is used and thus will affect the .csv file. To create a csv file and pie charts representing the table trained on *all* of the data, run the command “python3 main.py list 0” before running *bcsv*.

See Figures 5 and 6 for an example .csv file and an example pie chart, respectively.



*Figure 5: An example .csv file. Figure 6: An example pie chart.*

**smod [FILENAME.txt]:**

Either appends or deletes words in FILENAME.txt to/from stopwords.txt (located in the bin directory), depending on if the -dl flag is not or is used. Defaults to appending words. FILENAME.txt should be located in the bin directory

**htxt:**

This command prints out a shortened user guide/help text to the terminal.

**Scikit Mode:**

Putting an 's' in front of the commands *tble*, *clss*, *list, acrc*, and *crsv (stble, sclss, slist, sacrc,* and *scrsv*) runs the commands above except using a Naive Bayes algorithm from the scikit-learn library. The scikit Bayes algorithm classifies faster than our implementation, but the accuracy difference is negligible.

**Flags:**

Flags can be used with all standard or Scikit commands. Flags come last, after the command and any input files. Multiple flags can be used and their order does not matter.

**-vb:**

This flag stands for verbose. When used, the program will print misc. info to terminal (defaults to false)

**-nl:**

This flag stands for no loadbar. When used, the tqdm loading bar that appears when running several commands is disabled; otherwise, loadbar appears by default.

**-re:**

This flag stands for redo. When used, it impacts the behavior of multiple commands (defaults to false):

* *dwnl*: emails will be redownloaded even if they already exist in emails, test\_emails, or training\_emails
* *strp*: all emails will be re-stripped instead of just those that have not been stripped before
* *clss*: emails of the unknown category in training\_emails will be put back into the emails directory to be re-categorized

**-dl:**

This flag stands for delete. When used with smod command deletes words from stopwords.txt rather than append (defaults to false)

**-vi:**

This flag stands for visualize. When used, commands *bcsv* and *crsv* will generate graphs displaying statistics of the algorithm.

**Example Commands:**

* **python3 main.py htxt:** Access the shortened user guide.
* **python3 main.py dwnl:** The command to download new emails.
* **python3 main.py dwnl –nl → control\_z → bg:** download emails in the background.
* **python3 main.py dwnl -re:** The command to re-download every email in the inbox.
* **python3 main.py strp –re:** Strip all of the emails in the emails directory, even if they have been stripped previously.
* **python3 main.py catg:** Build training set out of category files in the categories directory.
* **python3 main.py tble:**  Train the model on the training set created by *catg*.
* **python3 main.py stble:** Like the tble command, but trains a scikit-learn model instead.
* **python3 main.py list 20 → python3 main.py acrc:** Randomly select 20% of the training emails to be used as test data, then perform a single accuracy test.
* **python3 main.py crsv 6 -vi:** Perform a 6-fold cross validation, then generate a
* **python3 main.py bcsv –vi:** Create a .csv file representing the current trained state of the table, and then create pie charts displaying the top nine most common words in each category.

**Troubleshooting:**

**Q:** ImportError, NameError, decode() error, ConnectionResetError?

**A:** You might be running our software in python 2 instead of python 3. On some systems, you have to type “python3 code.py” instead of “python code.py” to run python 3. In addition, you might have used pip, which is for python 2, to install packages, instead of pip3, which is for python3. Try typing “pip3 install package” instead of typing “pip install package”.

**Q:** Master/Global dictionary empty? Table empty? Software crashing on training?

**A:** If your bin/dictionary.txt file just contains a pair of curly braces, it is empty. This happened because there were no training set for the *dict* command to use to create the global dictionary. Be sure to run the *dict* command AFTER the *catg* command.

**Q:** Why can’t I use the command line after putting a process in the background?

**A:** This is because the tqdm loading bar is enabled. To disable it, run the command with the –nl flag.

**Q:** Why are there still tickets in my emails directory even after running *clss*?

**A:** There are a couple of reasons. One is that you may have not downloaded enough emails before running the *catg* command, and as a result some of the emails in your emails directory actually belong in the training\_emails directory. To remedy this issue, run the commands *ucat* → *catg* → *clss*.

The other reason is normal, and occurs when the classifier is simply unable to determine a category for the email. This is because either none of the words in the email’s body were found in the classifier’s training set, or because there are too many words in different categories for the model to make a decision.

**Q:** Why do my confusion matrices have an empty row and column?

**A:** We found that this error occurs occasionally. A workaround is to uncomment the lines:  **C = np.delete(C, (16,), axis=0)**

**C = np.delete(C, (16,), axis=1)**

at around line 280 in main.py.