THE GEORGE WASHINGTON UNIVERSITY
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CLASSIFIER FOR CONGRESSIONAL BILLS USING MACHINE LEARNING ALGORITHMS

Final Project Report

By

Brandon Bernier

Prepared for:
Professor Howie Huang
ECE 6130 Section 10:
Grid and Cloud Computing

Submitted: May 4, 2016

Abstract

This report describes a classifier for Congressional bills that attempts to predict how a given legislator will vote on future legislation. Using publicly available voting histories for all Senators and Representatives in the House, combined with the full text of all legislation voted on, the goal is to find similarities between bills that could potentially be a good enough predictor for future votes. The algorithm creates two sets of words for each legislator unique to bills they voted Yea or Nay on and is able to compare the words in new bills to those lists. In addition to looking at the text of a given piece of upcoming legislation, the political party of both the sponsor and the voter are weighted heavily in making the final prediction for how the legislator will vote. Important aspects of this project include the natural language processing that must be done on the full text of the legislation to prepare it for analysis as well as the algorithm used to be able to actually classify them as potential Yea or Nay votes. Overall, the results of this project show a functional proof of concept design that, with more work, could potentially be a fairly accurate predictor of how a legislator will vote on legislation.

Table of Contents

Table of Figures

1. Introduction

1.1. Project Goals

The original goal of this project was to be able to use natural language processing on collected Congressional legislation, relate it to individual legislators' voting history, and determine their voting patterns and tendencies to predict future votes. How Congress votes on legislation is not only crucially important to the way in which the government functions, but it is also an interesting data analytics problem to be able to work with the very large dataset available and pull the desired information from it.

The overall intention of the project is to set this algorithm or system up on a server in the cloud such that anyone would be able to easily work with this information and check on how a specific legislator is going to vote on a specific bill. Eventually, the goal would be to produce a website that users can interact with to quickly and easily access the desired information, however, that part of the system has not yet been completed. Additionally, the algorithm itself is functional and can be run on an Amazon EC2 instance, however, it requires more than 1GB of RAM, which exceeds the free tier, and testing of it running in the cloud was not completed for this reason.

2. System Design Overview

2.1. Overall Design

The overall design of this project consisted of three main parts: acquiring the necessary data, analyzing the data, and storing the data. Each part of the system works in conjunction with the others to be able to implement the functionality of predicting future votes on legislation.

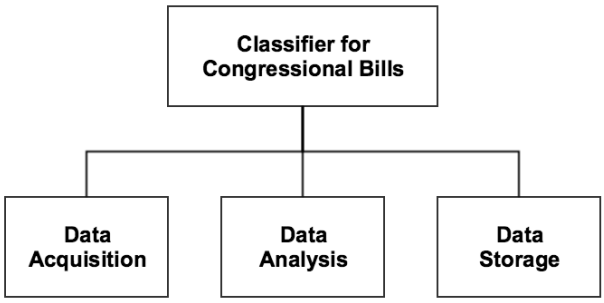

Figure 1: Overall System Tree Diagram

2.2. Datascraper

The core of this project depends on a large dataset available to train the algorithm, namely the full text of all legislation in Congress since 2009. Originally, I thought the full text was available directly through the Sunlight Congress API, which will be discussed later; however, I came to find out that there was no way to simply request the full text of a given piece of legislation. You can search over the full text, but you cannot get it back directly. To overcome this obstacle, I download and worked with the datascraper used by the Sunlight Congress API to be able to obtain the raw full text documents [1].

The datascraper is an open source project made specifically for producing public domain data about Congress and has a lot of functionality to be able to gather all types of information. Relevant to this project, I am using their fdsys.py script to be able to download text files of all Congressional legislation. While it takes hours to download the thousands of pieces of legislation from both the House of Representatives and the Senate, that luckily only needs to be run once, and all successive calls check the latest sitemap at https://www.gpo.gov/smap/fdsys/sitemap.xml for any changes and downloads new documents. Run the script by executing the following line, specifying the collection of BILLS from the 114th Congress to be stored as text files.

```
./run fdsys --collections=BILLS --congress=114 --store=text
```

2.3. Sunlight Congress API

The Sunlight Congress API is maintained by the Sunlight Foundation, a nonprofit group that uses technology to make the government and politics more transparent. Their public API is the backbone of this project, and I have spent most of my time on the project so far working on collecting and organizing the data I have obtained through a variety of the provided API calls. The main data being pulled from the API is information about legislators, the vote history on individual bills, and some related information about the bills that have been voted on [2].

```
https://congress.api.sunlightfoundation.com/[method]
```

2.3.1. Legislators

The legislators method is used to obtain the names, and more importantly, the bioguide IDs for all Congressmen. The bioguide ID is a unique identifier used throughout the project to identify legislators, especially connected to their voting history. The following Python snippet shows the function used to gather and store this data. The first section makes the API call, retrieves the JSON, parses it, and extracts the results from it. The second section iterates through every Congressman and creates a nested dictionary consisting of their bioguide ID, name, and a blank voting history.

```
def getCongressmenData():
      """ Get all necessary data and create dictionary of Congressmen."""
      congressmen_query = url_base + "legislators?per_page=all" + api_key
      congressmen       = urlopen(congressmen_query)
      congressmen_data  = congressmen.read()
      congressmen_dict  = json.loads(congressmen_data)
      congressmen_results = congressmen_dict["results"]

      for result in range(len(congressmen_results)):
            congress_dict = congressmen_dict["results"][result]
            first_name    = congress_dict["first_name"].encode('ascii', 'ignore')
            last_name     = congress_dict["last_name"].encode('ascii', 'ignore')
            bioguide_id   = congress_dict["bioguide_id"].encode('ascii', 'ignore')
            party         = congress_dict["party"].encode('ascii', 'ignore')
            name          = first_name + " " + last_name
            vote_dict[bioguide_id] = {"name":name, "party":party, "vote_history":{}}
```

2.3.1. Votes

The votes method works similar to the legislators API call, however there is a per page limit of 50 votes. With over 7000 votes, this resulted in 150+ different pages to comb through in order to collect the voting histories of all of the Congressmen. This proved to be a time-consuming process iterating page by page, so I used Python's multithreading module to speed up this task, which worked well. Python has some limitations that limit true parallelism, however, because this is an IO bound task, the speedup was still significant. Only the votes related to bills being passed are added to the vote dictionary to eliminate votes on amendments and procedural votes, etc., and those corresponding bills are added to a set to be analyzed later.

2.3.2. Bills

Finally, the bills method is used to obtain the last version of a bill. When the datascraper downloads the text of all bills, it creates multiple folders with multiple versions of the bill, for instance when it is introduced into the Senate (IS) and when it becomes an enrolled bill (ENR). By collecting the last version from the API, I am able to determine which downloaded copy to analyze with the natural language processing portion of the project later on. In essence, the version of the bill will direct the text parser to grab that specific version of the bill.

2.4. Python

2.4.1. JSON Storage

One challenge working with huge datasets and testing many aspects of the code was how to store the data. Without storing the data, I had to pull it from the API calls every time I executed the script, and in the case of reading in the full text of every bill, this was quite time consuming. To avoid this, data is saved as a JSON that can be quickly loaded back later [3]. Loading the entire voting history and the full text of the bills only takes a few seconds to start up, opposed to the many minutes it would otherwise require.

One of the quirks about working with JSON files in Python 2.7.11 is that when the JSON file is loaded in, all of the text is Unicode rather, which is standard in Python 3, rather than ASCII strings, which are common to Python 2.7.11. Figuring out that this was an issue was somewhat time consuming, but a simple function that is able to encode the Unicode text from the JSON into ASCII strings to work in Python 2.7 was implemented.

2.4.2. Natural Language Toolkit (NLTK)

The only major change to the project since the project proposal has been the decision against implementing the machine learning algorithms using TensorFlow because it seemed to be more complex than necessary for this specific task.. While TensorFlow sounded appealing to start, I quickly realized that it would simply overcomplicate the project with little benefit because I would not be able to run the

project across multiple machines, as TensorFlow is intended for, without making use of multiple machines and instances in AWS. TensorFlow also relies on writing machine learning algorithms from scratch using the primitives it provides, which is way beyond what I felt I would be able to accomplish for this project, having no previous background with machine learning.

So, instead of TensorFlow, I chose to work with the widely-used Natural Language Toolkit for Python to implement the natural language processing aspect of this project [4]. I used this toolkit to get a frequency distribution of the words in each bill as well as some of its functionality to actually pull words from bills. This code snippet shows the necessary preprocessing of the full text of the bill to be able to get it into a useful form. I make sure all of the text is lowercase, remove any numbers, replace all punctuation with spaces, then split the string at every space to break it into a list of words. I also filter out stopwords included in NLTK such as "the," "a," and "an," that are unimportant to the overall meaning of the bill as well as any other word it finds that is less than three characters.

```
def getFreqDist(bill_id):
      bill_text         = full_texts[bill_id]
      replacement       = string.maketrans(string.punctuation, ' '*len(string.punctuation))
      bill_formatted    = bill_text.lower().translate(replacement, string.digits)
      all_words         = filter(lambda w: not w in stop_words, bill_formatted.split())
      bill_words        = [ word for word in all_words if len(word) >= 3 ]
      freq_dist         = nltk.FreqDist(bill_words)

      for word in freq_dist.most_common():
            print word
      return freq_dist
```

2.4.3. Challenges

The biggest challenge in choosing to use Python for this project is that I had never used it before, so part of my goal for the midterm project milestone included the learning curve I had to overcome to get started. I was able to teach myself Python and relatively quickly get to work coding the many functions required to process the data. The next section goes into more specific system implementation details.

3. System Design

3.1. Data Acquisition

The most important part of the code attached in the appendix is the loadData() method that allows me to acquire all of the necessary data and proceed accordingly. If the data has not changed, I simply load the data that was previously acquired. As was mentioned earlier, a large part of the data acquisition section of this project relies on the datascraper as well as some functions I created in Python to grab all of the required information. Figure 2 shows the full tree of function calls within loadData() to show exactly what is being used and how it all relates. Once the loadData() function is called, the other functions are called from within it. Three major pieces of data are necessary for the operation of this project including data about legislators, data about the votes cast, and the full text of all of the bills. Each of the inner functions eventually saves the necessary data in the dictionaries as shown. However, if the data is already stored, those files are read instead of executing these functions.

Figure 2: Tree of Function Calls and Output within loadData()

3.2. Data Analysis

There are a few separate methods used to perform the data analysis in this project. First, the data has to be prepared for analysis, which in the case of the full texts of bills, the words need to be separated, formatted uniformly, and counted. Once this is done, the actual classification of bills for individual legislators can be done, yielding the predicted votes.

3.2.1. Text Parser

One of the more important pieces of this project is a small, yet extremely powerful function that returns a list of all of the words in a bill. While that may not sound all that impressive, the complicated part is choosing what words and characters not to include in this list to ensure that the list only has those words most important or relevant to the overall meaning of the bill. In order to properly count the words in a bill, they must all be identical, including uppercase or lowercase letters. To account for this, the parser makes all characters lowercase and removes any digits in the bill as these are unimportant for this circumstance. Figure 3 shows the rest of the processing done on the text until the result, a list of important words in a bill, is returned.

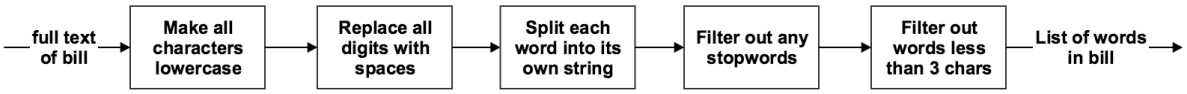

Figure 3: Processing of Bill Text

3.2.2. Frequency Distribution

Once the list of words in a bill is obtained, we can achieve a frequency distribution of the words in that bill. To do this, I use the FreqDist() function included in the Natural Language Toolkit. By applying the most_common() method on that frequency distribution, a list of tuples including the word and the count of that word is returned.

An extension of this capability for a single bill is being able to do it for a single legislator using their ID. The getIDFreqDist() function iterates over every bill a

legislator has voted on, gathers the words from those bills, and creates a master list of all of the words in bills the legislator voted Yea on and those they voted Nay on. This distribution of the most common words is important later when comparing the words from upcoming bills for each legislator. To narrow these lists even further, the list of yea words and nay words specifically exclude any words common to bills that received a yea and a nay vote. This is done with a list comprehension. The reason for doing this was to eliminate many of the Congress specific stopwords that add little meaning to the bill overall, but are more procedural. For instance, the word deleted is often used throughout bill drafts, but since it is in both sets of bills, it is automatically excluded.

3.2.3. Classifier

The classifier itself is the final piece of the algorithm that decides whether a legislator is like to vote Yea or Nay on a given bill. To accomplish this, I am using two different methods that complement each other. Originally, this was intended to use more complex machine and deep learning algorithms, however, without any background in those algorithms they were quite difficult to implement in a meaningful way. Also, after doing some more research on the topic, it is possible to make a very good prediction based solely on a legislator's political party. In fact, Congress just set a new record high percentage for voting along party lines of 92% of the time for Republicans and 94% of the time for Democrats [5].

Using political party alone would yield high prediction accuracy, however, it was not the main goal of the project. So, in addition to making a prediction based on political parties, the classifier includes information based on the number of times words appear in the upcoming bill and how often they appeared in previous bills voted on by the legislator. The words of upcoming bills are parsed as previously explained, then compared against the list of words existing in bills a legislator voted "Yea" on and those they voted "Nay" on. To make an actual prediction based on this, two counters exist for each bill that add up the frequencies of the words stored in the lists. For instance, say "livestock" was in a bill 54 times that a legislator voted in favor of, if it appears in the upcoming bill, 54 is added to the yea count.

The following figure better simplifies how exactly the classifier makes decisions. It is also worth noting that the option to classify independently by political party or by words is still available. The goal is that combining knowledge of political parties as well as the language processing of the word counting method will increase the accuracy of voting prediction in this system.

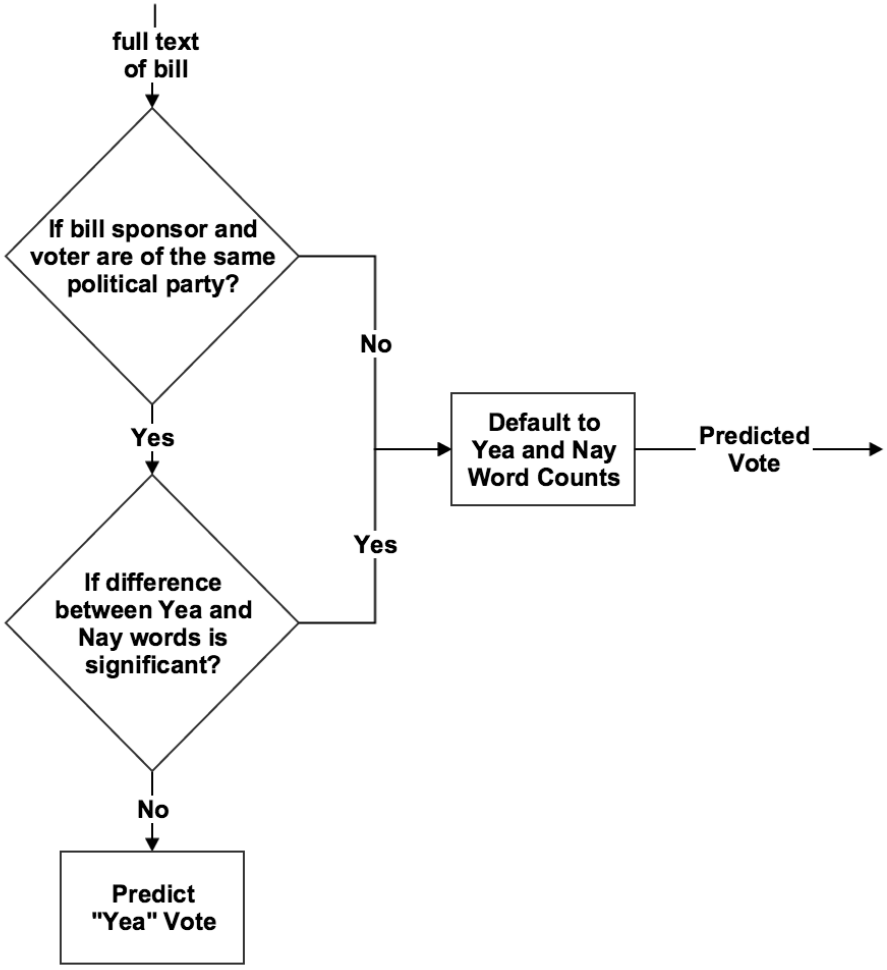

Figure 4: Basic Overview of Classifier

4. Future Work

4.1. Natural Language Processing

Looking into the actual feasibility or necessity of improving the natural language processing capabilities of this system would be interesting. As already mentioned, there is good data that proves voting in favor of the sponsor's political party happens just over 90% of the time. Originally I planned to do some form of k-means clustering to be able to classify the bill, but it proved that other algorithms such as counting the words in combination with knowledge of the legislator's political party might work just as well.

4.2. Running on Amazon EC2

The goal of this project was always to be able to run this algorithm remotely in the cloud. This will not be a big deal because it is mainly as simple as copying the directory to an EC2 instance and running the Python script from there. However, in attempting to test this capability, I realized that the free tier of AWS would not provide enough memory necessary to load in all of the full texts of the bills and worth with the large dataset. Whenever I would attempt to run it on the cloud with it loading in the full texts, the process would automatically be killed. Commenting out the load of the texts proved that that was indeed the problem.

4.3. Website Front End

Ideally, my hope was to be able to have the backend functioning sooner so that I could turn this project into an interactive website that provides the data in a nicer way; however, with the amount of work left to be done figuring out the classifier algorithm, that was not possible this semester. Working with a partner would have probably been more ideal to being actually able to complete both the data processing and a nicer interface. Nonetheless, this is an interesting project I would like to continue working on and eventually get that finished.

5. Results

5.1. Compared with Bills Already Voted On

One attempt that I made to be able to test this project was to run every legislator and bill through the classifier to see if it would be able to correctly predict the outcome based on the words in it. There seemed to be two sets of legislators when I performed this test. Those who had a few hundred votes came out with a very high 90s percentage of correct predictions, while many legislators who had over 1000 or 2000 votes landed around 65% correct. I believe the results were this way for a few reasons. Those with the lower total vote count were Senators, where there are less bills overall and most are of a decent length. The House of Representatives, on the other hand, often has many more bills before them and many are short resolutions that do not have very many unique words in them. This causes there to be many cases where it is difficult to predict the vote based solely on words.

Additionally, this system does not account for abstentions, or Not Voting votes. It also does not keep track of Independent Congressmen. It will only ever predict a "Yea" or a "Nay." Another problem that I noticed while trying to test the algorithm is that the system is severely unbalanced in that most of the votes it is based on were "Yea" votes, and it seems that there are few bills for each legislator that they voted "Nay" on. This makes it difficult to create a broad enough set of Nay words for a legislator to properly be able to decide on future bills.

5.2. For Upcoming Bills

Testing for upcoming bills shows some promise. When testing for a Republican, two current Republican sponsored bills came up as predictions of "Yea." However, because of the limited timeframe and the nature of Congress, it will be some time before future votes are actually counted and can be compared against the predictions made by the algorithm today. There are also only currently two upcoming pieces of legislation, so to properly test the system, it will take a lot of time to get enough data to accurately suggest some meaningful results.

6. Conclusions

Although there were some minor tweaks to the project implementation as I went along, I was able to successfully meet my midterm project goals, and now also the goals I set in the midterm project report. I would consider the project functioning in that it will make predictions based on various sources of information and puts a lot of information about Congress and legislation very quickly at your fingertips. I would like to have more substantial test data before suggesting that the method by counting words is any better than simply predicting based on political party, but the amount learned to get to this point has been extensive.

This project forced me to learn a lot on my own and delve deeper into topics of cloud computing, machine learning, and Python programming. It was very interesting to be able to use public APIs with Python to quickly grab and store tons of data that could then be manipulated and worked with. With more work, this project could continue to be developed into something more fully functional and better tested, with a much nicer interface.

7. References

[1] @unitedstates. GitHub. [Online]. https://github.com/unitedstates/congress

[2] Sunlight Foundation. Sunlight Congress API. [Online]. https://sunlightlabs.github.io/congress/index.html

[3] Python Software Foundation. Python 2.7.11 Documentation. [Online]. https://docs.python.org/2/library/json.html

[4] Steven Bird, Ewan Klein, and Edward Loper. (2009, June) Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. [Online]. http://www.nltk.org/book_1ed/

[5] National Journal. (2014, February) National Journal. [Online]. https://www.nationaljournal.com/congress/2014/02/03/congress-sets-record-voting-along-party-lines

8. Appendix – Project Proposal

8.1. Project Goals

The overall goal of this project is to create a classifier capable of predicting how a Congressman will vote on future legislation based on their past voting history. To accomplish this, machine learning algorithms will need to be used to train the classifier based on the huge amount of data available on past bills and how each Congressman has previously voted.

The content of all legislation on record must be analyzed along with how each member of Congress voted on each piece of legislation. By comparing the language in previous bills with the language of upcoming bills, it should be possible to predict how likely someone is to vote Yea, Nay, or Abstain. For instance, a Senator with a long history of supporting gun rights and voting against gun control measures is very likely to continue to vote against gun control legislation. It will require complex machine learning algorithms to be able to accurately create such a classifier. Google's recently open-sourved machine learning library TensorFlow will be used.

If the backend algorithm comes together as described and proves to be functional, this project can also be easily deployed to the cloud with Docker as TensorFlow has built-in support for this. Docker allows the TensorFlow installation to be completely isolated in a container from other packages and things on the system. A simple frontend website could be created to show the results from the algorithm.

8.2. Plan

8.2.1. Sunlight Congress API

Sunlight Foundation provides a live JSON API for the people and work of Congress. This API will be critical in providing all of the necessary data for this project. It provides access to all legislation in both the House and Senate dating back to 2009, a history of all roll call votes, and live updates on upcoming bills. The voting history on legislation since 2009 will be examined and studied to create the classifier, and the information regarding upcoming bills will be used as the test cases for the classifier.

8.2.2. TensorFlow

TensorFlow is Google's open-sourced software library for machine learning. It provides a flexible architecture that makes it easy to do computation across one or more CPUs or GPUs. I will use this library as the basis for creating the machine learning algorithm for this project.

8.2.3. Docker

Docker provides the ability to easily install and run TensorFlow in a container isolated from other packages on the machine. The use of Docker will simply be an added element to this project because it was an interesting topic we have discussed in class, but the project relies on getting the machine learning algorithms working.

8.3. Milestones

8.3.1. Midterm Report

For the midterm report, I plan to have the basic functionality and theory fully understood so that I can move forward with actual implementation. I have no background in machine learning, so I am fully expecting there to be a very steep learning curve to climb to be able to figure out exactly how to go about implementing this project. The first step that should be easy enough to accomplish is getting familiar with the Sunlight Congress API, how exactly I will be accessing the necessary data, what format it is provided in, and how that data will be fed into the machine learning algorithm.

8.3.2. Final Report

By the final report, I hope to have the project completed and fully functional. This includes having a classifier that has been fully trained using the data provided by the Sunlight Congress API as well as functional machine learning algorithms that accurately predict how Congressmen will vote on future legislation and the overall outcome of that legislation (whether or not it will pass, etc.). How that data is finally presented will be entirely dependent on it working first, but it would be ideal to have some basic website set up that shows upcoming legislation as well as the predictions made by the algorithm.

9. Appendix – Python Code

```
#######################################
# Brandon Bernier
# ECE 6130 - Grid and Cloud Computing
# Final Project
# Classifier for Congressional Bills
# Using Machine Learning Algorithms
#######################################

import re
import string
import json

import os.path
import subprocess

from urllib2 import urlopen
from threading import Thread

import nltk
from nltk.corpus import stopwords


url_base    = "https://congress.api.sunlightfoundation.com/"
api_key     = "&apikey=99edce157d934983ad380f55ae4c1757"
per_page    = "&per_page=50"

stop_words  = set(stopwords.words("english"))

vote_dict   = {}
full_texts  = {}
bill_ids    = {}
versions    = set([ 'eh', 'is', 'cdh', 'ih', 'cds', 'lts', 'es', 'pap', 'pp',
                            'rs', 'pcs', 'rcs', 'enr', 'rh', 'eah', 'pcs2', 'eas',
                            'rfs2', 'rfs', 'rds', 'ath', 'ats', 'rfh'])


class VoteWhip(Thread):
      """Thread class for multithreading to get vote history."""
      def __init__(self, thread_id):
            Thread.__init__(self)
            self.thread_id = thread_id

      def run(self):
            getVoteDataSinglePage(self.thread_id)


def getCongressmenData():
      """ Get all necessary data and create dictionary of Congressmen."""
      congressmen_query       = url_base + "legislators?per_page=all" + api_key
      congressmen             = urlopen(congressmen_query)
      congressmen_data  = congressmen.read()
      congressmen_dict  = json.loads(congressmen_data)
      congressmen_results     = congressmen_dict["results"]
```

```
      for result in range(len(congressmen_results)):
            congress_dict     = congressmen_dict["results"][result]
            first_name        = congress_dict["first_name"].encode('ascii', 'ignore')
            last_name         = congress_dict["last_name"].encode('ascii', 'ignore')
            bioguide_id       = congress_dict["bioguide_id"].encode('ascii', 'ignore')
            party             = congress_dict["party"].encode('ascii', 'ignore')
            name              = first_name + " " + last_name
            vote_dict[bioguide_id] = {"name": name, "party": party, "vote_history": {}}


def getVoteDataSinglePage(page):
      """Get vote data for single page of results."""
      page_num          = "&page=" + str(page)
      vote_query        = url_base + "votes?fields=voter_ids,bill_id,vote_type" + per_page
+ page_num + api_key
      votes             = urlopen(vote_query)         #instance
      vote_data         = votes.read()                      #JSON
      vote_dict         = json.loads(vote_data)       #dict
      vote_results      = vote_dict["results"]        #list

      for result in range(len(vote_results)):
            result_dict = vote_results[result]
            if "bill_id" in result_dict and result_dict["vote_type"] == "passage":
                  addVotes(result_dict)


def getVoteData():
      """Get all of the vote data available."""
      vote_query        = url_base + "votes?fields=voter_ids,bill_id,vote_type" + api_key
      votes             = urlopen(vote_query)         #instance
      vote_data         = votes.read()                      #JSON
      vote_dict         = json.loads(vote_data)       #dict
      vote_count        = vote_dict["count"]          #int
      page_count        = (vote_count/50) + 1
      threads           = []

      for page in range(page_count):                        #spawn 1 thread per page here
            thread = VoteWhip(page+1)
            thread.start()
            threads.append(thread)

      for thread in threads:
            thread.join()


def addVotes(result_dict):
      """Function to add a vote to Congressman's voting history."""
      bill_id = result_dict["bill_id"].encode('ascii','ignore')
      if bill_id not in bill_ids:
            bill_ids.update({bill_id: {"sponsor_party": getBillSponsor(bill_id)}})
      for voter in result_dict["voter_ids"]:
            vote = result_dict["voter_ids"][voter].encode('ascii','ignore')
            if voter in vote_dict:
                  vote_dict[voter]["vote_history"][bill_id] = vote
```

```
def getBillSponsor(bill_id):
      """Get the political party of the sponsor of a bill."""
      bill_query        = url_base + "bills?bill_id=" + bill_id + api_key
      bill              = urlopen(bill_query)
      bill_data         = bill.read()
      bill_dict         = json.loads(bill_data)
      bill_results      = bill_dict["results"]

      if bill_results and "sponsor_id" in bill_results[0]:
            sponsor_id  = bill_results[0]["sponsor_id"]
            if sponsor_id in vote_dict:
                  sponsor_party     = vote_dict[sponsor_id]["party"]
                  return sponsor_party
      else:
            return "NA"

def getUpcomingBills():
      """Get info for all upcoming bills."""
      new_bills_query = url_base + "upcoming_bills?" + per_page + api_key
      new_bills_url     = urlopen(new_bills_query)          #instance
      new_bills_data    = new_bills_url.read()              #JSON
      new_bills_dict    = json.loads(new_bills_data)  #dict
      results           = new_bills_dict["results"]         #list
      new_bills         = {}

      for result in range(len(results)):
            new_bill          = results[result]["bill_id"].encode('ascii', 'ignore')
            sponsor_party     = getBillSponsor(new_bill).encode('ascii', 'ignore')
            new_bills.update({new_bill: {"sponsor_party": sponsor_party}})
            readBill(new_bill)

      return new_bills


def printBills():
      """Print all bills in the vote history."""
      for bill in bill_ids:
            print bill, bill_ids[bill]["sponsor_party"]


def printBillInfo(bill_id):
      """Print all info for bill with bill_id."""
      print bill_id, bill_ids[bill_id]["sponsor_party"]


def printCongressmen():
      """Print list of all Congressmen."""
      for congressman in sorted(vote_dict):
            print congressman, vote_dict[congressman]["name"],
vote_dict[congressman]["party"]


def getIDVoteHistory(bioguide_id):
      """Get the vote history of a single Congressman by ID."""
      return vote_dict[bioguide_id]["vote_history"]
```

```
def printIDVoteHistory(bioguide_id):
      """Print the vote history of a single Congressman by ID."""
      print bioguide_id + ":", vote_dict[bioguide_id]["name"],
vote_dict[bioguide_id]["party"]
      vote_history = getIDVoteHistory(bioguide_id)
      for vote in sorted(vote_history):
            print "\t" + vote + ":", vote_history[vote]

def printVoteHistoryAll():
      """Print the vote history of all Congressmen."""
      for congressman in sorted(vote_dict):
            printIDVoteHistory(congressman)


def getBillVersion(bill_id):
      """Get the last version of a bill."""
      bill_query        = url_base + "bills?bill_id=" + bill_id + api_key
      bill              = urlopen(bill_query)
      bill_data         = bill.read()
      bill_dict         = json.loads(bill_data)
      bill_results      = bill_dict["results"]

      if bill_results and "last_version" in bill_results[0]:
            version =
bill_results[0]["last_version"]["version_code"].encode('ascii','ignore')
            return version
      else:
            return "NA"


def getBillVersionsAll():
      """Create a set of all possible bill versions."""
      for bill in bill_ids:
            version = getBillVersion(bill)
            versions.add(version)
      print versions


def readBill(bill_id):
      """Read in the full text version of a bill."""
      split       = bill_id.split("-")
      bill        = split[0]
      session           = split[1]
      bill_split  = re.split('(\d+)', bill)
      bill_type   = bill_split[0]
      version     = getBillVersion(bill_id)
      folder_path = "congress-master/data/" + session + "/bills/" + bill_type + "/" + bill
+ "/text-versions/"
      file_path   = folder_path + version + "/document.txt"

      print file_path

      if os.path.exists(file_path):
            with open(file_path, "r") as full_text:
                  bill_text = full_text.read()
```

```
                  full_texts[bill_id] = bill_text
      else:
            for vers in versions:
                  file_path = folder_path + vers + "/document.txt"
                  if os.path.exists(file_path):
                        with open(file_path, "r") as full_text:
                              bill_text = full_text.read()
                              full_texts[bill_id] = bill_text
                              break
                  else:
                        pass


def readAllBills():
      """Read in the full text version of all bills."""
      for bill_id in bill_ids:
            print bill_id
            readBill(bill_id)


def getBillWords(bill_id):
      """Parse a bill and return a list of all meaningful words in the bill."""
      bill_words  = []
      if bill_id in full_texts:
            bill_text         = full_texts[bill_id]
            replacement       = string.maketrans(string.punctuation, '
'*len(string.punctuation))
            bill_formatted    = bill_text.lower().translate(replacement, string.digits)
            all_words         = filter(lambda w: not w in stop_words,
bill_formatted.split())
            bill_words        = [ word for word in all_words if len(word) >= 3 ]
      else:
            pass

      return bill_words


def getFreqDist(bill_id):
      """Gets the frequency distribution for a given bill."""
      bill_words  = getBillWords(bill_id)
      freq_dist   = nltk.FreqDist(bill_words)

      print bill_id
      for word in freq_dist.most_common():
            print word

      return freq_dist.most_common()


def getIDFreqDist(bioguide_id):
      """Gets the frequency distribution of words in all bills voted on by specific
legislator."""
      vote_history                  = getIDVoteHistory(bioguide_id)
      word_dict                     = {"bill_words": [], "yea_words": [], "nay_words": []}
      temp_words                    = []
```

```
      yea_words_set                 = set()
      nay_words_set                 = set()

      for vote in sorted(vote_history):
            temp_words = getBillWords(vote)
            word_dict['bill_words'] += temp_words
            if vote_history[vote] == "Yea":
                  word_dict['yea_words'] += temp_words
                  for word in temp_words:
                        yea_words_set.add(word)
            elif vote_history[vote] == "Nay":
                  word_dict['nay_words'] += temp_words
                  for word in temp_words:
                        nay_words_set.add(word)
            else:
                  pass

      yea_wordss = yea_words_set.difference(nay_words_set)
      nay_wordss = nay_words_set.difference(yea_words_set)

      yea_words  = [x for x in word_dict['yea_words'] if x in yea_wordss]
      nay_words  = [x for x in word_dict['nay_words'] if x in nay_wordss]

      yea_freq_dist     = nltk.FreqDist(yea_words).most_common()
      nay_freq_dist     = nltk.FreqDist(nay_words).most_common()

      freq_dist         = {"yea": yea_freq_dist, "nay": nay_freq_dist}

      return freq_dist


def ascii_encode_dict(data):
    ascii_encode = lambda x: x.encode('ascii', 'ignore')
    return dict(map(ascii_encode, pair) for pair in data.items())


def loadData():
      """Either gets all necessary data or loads previously stored data."""
      global vote_dict, bill_ids, full_texts
      print "Loading Data"
      subprocess.Popen("./run fdsys --collections=BILLS --congress=114 --store=text",
                              cwd="congress-master/", stdout=subprocess.PIPE, shell=True)
      if os.path.exists("json/vote_history.json") and os.path.exists("json/bill_ids.json"):
            with open('json/vote_history.json','r') as votes_json:
                  vote_dict   = json.load(votes_json)
            with open('json/bill_ids.json','r') as bill_ids_json:
                  bill_ids    = json.load(bill_ids_json)
      else:
            getCongressmenData()
            getVoteData()
            with open("json/vote_history.json", 'w') as votes_json:
                  json.dump(vote_dict, votes_json)
            with open("json/bill_ids.json", 'w') as bill_ids_json:
                  json.dump(bill_ids, bill_ids_json)
```

```
      if os.path.exists("json/full_texts.json"):
            with open("json/full_texts.json", 'r') as full_texts_json:
                  full_texts  = json.load(full_texts_json, object_hook=ascii_encode_dict)
      else:
            getBillVersionsAll()
            readAllBills()
            with open("json/full_texts.json", 'w') as full_texts_json:
                  json.dump(full_texts, full_texts_json)
      print "Loading Data Complete"


def classifierByParty(bioguide_id):
      """Returns predictions for all upcoming bills based solely on the voter and sponsor's
political parties."""
      voter_party = vote_dict[bioguide_id]["party"]
      new_bills   = getUpcomingBills()

      for bill in new_bills:
            sponsor_party = new_bills[bill]["sponsor_party"]
            if voter_party == sponsor_party:
                  new_bills[bill].update({"vote": "Yea"})
            else:
                  new_bills[bill].update({"vote": "Nay"})

      print bioguide_id, vote_dict[bioguide_id]["name"], voter_party
      for bill in new_bills:
            print bill, new_bills[bill]["sponsor_party"], new_bills[bill]["vote"]

      return new_bills

def classifierByWords(bioguide_id):
      """Returns a set of predictions for all upcoming bills based on the words in the
bill."""
      new_bills   = getUpcomingBills()
      freq_dist   = getIDFreqDist(bioguide_id)
      yea_common  = dict(freq_dist["yea"])
      nay_common  = dict(freq_dist["nay"])

      for bill in new_bills:
            bill_text = getBillWords(bill)
            yea_count = 0
            nay_count = 0
            for word in bill_text:
                  if word in yea_common:
                        yea_count += yea_common[word]
                  if word in nay_common:
                        nay_count += nay_common[word]
            if yea_count >= nay_count:
                  new_bills[bill].update({"vote": "Yea"})
            else:
                  new_bills[bill].update({"vote": "Nay"})
            if yea_count > 10*nay_count or nay_count > 10*yea_count:    # If difference is
overwhelming, set confidence high
                  new_bills[bill].update({"confidence": 1})
            else:
```

```
                  new_bills[bill].update({"confidence": 0})
            print "Yea: ", yea_count, "Nay: ", nay_count

      print bioguide_id, vote_dict[bioguide_id]["name"], vote_dict[bioguide_id]["party"]
      for bill in new_bills:
            print bill, new_bills[bill]["sponsor_party"], new_bills[bill]["vote"],
new_bills[bill]["confidence"]

      return new_bills


def classifier(bioguide_id):
      """Combines data from classifying by party and by words to return predictions."""
      by_party    = classifierByParty(bioguide_id)
      by_words    = classifierByWords(bioguide_id)
      predictions = {}

      for bill in by_words:
            if by_words[bill]["vote"] == by_party[bill]["vote"]:
                  predictions.update({bill: {"vote": by_party[bill]["vote"]}})
            elif by_words[bill]["confidence"]:
                  predictions.update({bill: {"vote": by_words[bill]["vote"]}})
            else:
                  predictions.update({bill: {"vote": by_party[bill]["vote"]}})

      print bioguide_id, vote_dict[bioguide_id]["name"], vote_dict[bioguide_id]["party"]
      for bill in predictions:
            print bill, by_party[bill]["sponsor_party"], predictions[bill]["vote"]


def classifierByWordsTEST(bioguide_id):
      """Tests the algorithm against the dataset itself."""
      new_bills   = getIDVoteHistory(bioguide_id) #Pull all bills from vote history.
      freq_dist   = getIDFreqDist(bioguide_id)
      yea_common  = dict(freq_dist["yea"])
      nay_common  = dict(freq_dist["nay"])
      new_bill_votes = {}
      right = 0
      total = 0

      for bill in new_bills:
            bill_text = getBillWords(bill)
            yea_count = 0
            nay_count = 0
            for word in bill_text:
                  if word in yea_common:
                        yea_count += yea_common[word]
                        print bill, word, yea_common[word], "Yea: ", yea_count, "Nay: ",
nay_count
                  if word in nay_common:
                        nay_count += nay_common[word]
                        print bill, word, nay_common[word], "Yea: ", yea_count, "Nay: ",
nay_count
            if yea_count == nay_count:
                  new_bill_votes.update({bill: {"vote": "EQUAL"}})
```

```
            elif yea_count > nay_count:
                  new_bill_votes.update({bill: {"vote": "Yea"}})
            else:
                  new_bill_votes.update({bill: {"vote": "Nay"}})

      for bill in new_bill_votes:
            if vote_dict[bioguide_id]["vote_history"][bill] == "Not Voting":
                  pass
            else:
                  if new_bill_votes[bill]["vote"] ==
vote_dict[bioguide_id]["vote_history"][bill]:
                        right += 1
                        total += 1
                        #print      "predicted: ", new_bill_votes[bill]["vote"],
                        #          "actual: ",
vote_dict[bioguide_id]["vote_history"][bill],
                  else:
                        total += 1
                        #print      "predicted: ", new_bill_votes[bill]["vote"],
                        #          "actual: ",
vote_dict[bioguide_id]["vote_history"][bill],

      print bioguide_id, "Right: ", right, "Total: ", total, "Percentage: ",
float(right)/total


def main():
      """Used for testing with different values."""
      loadData()
      printCongressmen()
      #printBills()
      #classifierByParty("Z000018")
      #classifierByWords("W000817")
      #classifierByWordsTEST("Z000018")
      #print getIDFreqDist("Z000018")
      #printIDVoteHistory("W000817")
      #getIDFreqDist("Z000018")
      #print full_texts["hr3521-113"]
      #getFreqDist("hr3521-113")
      #getFreqDist("hr4923-114")  #UPCOMING BILL


if __name__ == '__main__':
      main()
```