CS 4740
Professor Cardie
Team members: John Trujillo (jpt86), Elaine Huang (yh467), Benjamin Steeper(bds238)

Assignment #1 Report

PREPROCESSING

We chose to code our n-gram model in Python for its weakly typed nature, many importable libraries (such as numpy) and its overall prominence in the NLP community.

We kept preprocessing relatively simple since the files were already tokenized according to the instructions. Our preprocessing can be summed up in our first two functions: clean(file_path) and sep_paras(file_path).

To initialize our corpus, we simply call clean with every file's path and store that value in a variable. Clean opens the file, replaces "\n" strings with spaces, and removes spaces after apostrophes. Clean then splits the text by space and returns a list of tokens. We did this to glue apostrophes with their following word to solve the contraction problem.

Our "sep_paras" function is necessary for classification later on and follows the same steps as "clean" only that it returns a triple nested list with paragraphs grouped and tokens within those paragraphs.

DATA STRUCTURE:

We store our unigram and bigram models in nested python dictionaries. For unigrams this means dictionary keys are words and values are frequencies, probabilities etc. For our bigram model, dictionary keys are the first word in the sequence and values are separate dictionaries storing all possible second words in the sequence as keys and frequencies or probabilities as values.

RANDOM SENTENCE GENERATION

Some Trump generated sentences by our "unigram_sentence" function :
['<s>', 'is', 'it', 'decline', 'vet', 'never', 'of', 'possible', 'deems', 'at', 'want', 'look', 'will', 'china', '</s>']
['<s>', '15', 'our', 'they', 'better', 'the', 'they', 'date', 'question', 'know', 'evangelical', 'running', 'in', 'read', '</s>']
['<s>', 'jobs', 'will', 'about', 'them', ',', 'you', '21', 'very', 'been', 'the', 'shooting', '"re", '.', '</s>']

Some Obama generated sentences by our "unigram_sentence" function :
['<s>', 'fiscal', 'and', 'task', 'responsibility', 'introducing', 'authorize', 'the', 'health', 'support', 'that', 'and', 'politics', 'insurance', '.', '</s>']
['<s>', 'united', 'his', 'spent', '.', '</s>']
['<s>', 'and', 'paperwork', 'the', 'pursue', ',', 'home', 'looks', 'to', 'nothing', 'she', 'values', 'an', 'colleges', 'a', 'you', 'after', 'triple', 'that', 'let', 'for', 'willing', 'history', '</s>']

Some Trump generated sentences by our "bigram_sentence" function :
['<s>', 'trump', 'tower', 'and', 'so', 'they', '’re', 'going', 'to', 'start', 'making', 'america', 'that', 'way', '</s>']

['<s>', 'we', '’re', 'going', 'to', 'hear', 'they', '’re', 'fine', '.', '</s>']
['<s>', 'in', 'these', 'people', 'that', 'bernie', 'sanders', 'lost', ',', 'this', 'before', 'this', 'because', 'i', '</s>']
['<s>', 'talking', '—', 'this', 'year', 'with', 'the', 'vets', 'that', 'much', '.', '</s>']

Some Obama generated sentences by our “bigram_sentence” function:
['<s>', 'shrapnel', 'in', 'capabilities', 'and', 'women', 'to', 'europe', 'to', 'defend', 'against', 'al', 'qaeda', 'captors', '.', '</s>']
['<s>', 'to', 'change', 'can', 'work', 'with', 'iran', '"s", 'legitimacy', ',', 'and', 'so', 'tonight', 'we', 'summon', 'the', 'people', 'of', 'ordinary', 'people', 'who', 'could', 'wear', '</s>']
['<s>', 'commitment', 'to', 'include', 'current', 'retirees', 'to', 'pursue', 'the', 'american', 'history', '.', '</s>']

Our unigram models show much less adherence to basic grammatical structures than our bigram models for both Obama and Trump corpuses. For example, one of obama’s generated sentence knows to put together the sequence “to defend against al qaeda”. A coherent sequence like this one would be highly improbable if coming from a unigram generator. However, the more tokens you observe in the sequence, the quicker semantics begin to fall apart. “To defend against al qaeda” is followed by “captors”. Why would we need to defend against captors?

When comparing both models (Trump / Obama), we find that in both unigram and bigram generated sentences Obama’s phrases are more fluid, while Trump’s tend to be choppier. This is likely due to his more fragmented style of speaking. Trump also tends to make references to himself, his opponents and more general pronouns, while Obama’s generated sentences tend to consist of more targeted fluid thoughts. Also, political climates and context is very much apparent in our generated sentences. For example, Obama references terrorism and healthcare frequently while Trump’s generated sentences introduce nouns like “Bernie Sanders” and protectionist rhetoric.

SEEDING
For seeding, we simply pass in a second argument into our sentence generating functions (“unigram_sentence” and “bigram_sentence”). This second argument is a string. The string will either be empty, in which case seeding is ignored, or populated with a list of words separated by spaces, in which case it will seed by grabbing the last word (unigram) in the string as the temporary start token in the sentence generation. We follow a similar procedure for bigram generation, such that our algorithm gets the last bigram of the given seed, and from there builds an adequate random generated bigram sentence similar to that of our sentence generation. Examples below for both unigram and bigram models:

Trump unigram seeding:
Function call: unigram_sentence(trumptrain,"do this fast")
['<s>', 'do', 'this', 'fast', '?', '</s>']
['<s>', 'do', 'this', 'fast', 'mexico', 'know', 'about', 'that', 'unions', 'to', 'correct', 'christians', 'cutting', '.', '</s>']

Trump bigram seeding:
Function call: bigram_sentence(trumptrain,"week")
['<s>', 'week', 'later', 'time', 'there', 'i', '’ll', 'have', 'a', 'businessman', '.', '</s>']
['<s>', 'week', 'that', '’s', 'the', 'cards','no', 'i', '’m', 'forced', 'to', 'thank', '</s>']

['<s>', 'week', ',', 'where', 'we', 'have', 'rubio', 'and', 'you', 'allow', 'you', 'have', 'no', ',', '</s>']

Function call: bigram_sentence(trumptrain,"cover for").....
['<s>', 'cover', 'for', 'a', 'lot', 'of', 'miles', '.', '</s>']
['<s>', 'cover', 'for', 'our', 'inner', 'cities', ',', 'as', 'saudi', 'arabia', 'didn', '’t', 'want', 'some', '</s>']
['<s>', 'cover', 'for', 'our', 'country', '.', '</s>']

Function call: bigram_sentence(trumptrain,"kind of the donors") …….
['<s>', 'kind', 'of', 'the', 'donors', 'to', '$', '21', 'trillion', 'going', 'to', 'end', 'up', '.', '</s>']
['<s>', 'kind', 'of', 'the', 'donors', 'or', 'lives', 'than', 'they', 'were', 'supposed', 'to', 'stay', 'forever', '</s>']
['<s>', 'kind', 'of', 'the', 'donors', 'or', 'redundant', 'by', 'a', 'deal', 'because', 'he', '’s', 'not', '</s>']

Obama unigram seeding:
Function call: unigram_sentence(obamatrain,"big")
['<s>','big', 'and', 'encourage', 'world', '.', '</s>']
['<s>', 'big', 'the', 'inherit', 'and', 'but', 'about', 'indian-americans', ',', 'as', 'has', '’a', 'ohio', 'that', 'did', 'and', 'our', 'astonished', 'behind', 'debates', 'his', 'grassley', '"s", '</s>']

Obama bigram seeding:
Function call: bigram_sentence(obamatrain,"certainly")
['<s>', 'certainly', 'my', 'job', '.', '</s>']
['<s>', 'certainly', ',', 'we', 'have', 'their', 'obligations', ',', 'not', 'feared', 'our', 'programs', ',', 'public', 'universities', 'educate', 'a', 'larger', 'cause', '.', '</s>']
['<s>', 'certainly', 'be', 'paying', 'a', 'chance', '.', '</s>']

Function call: bigram_sentence(obamatrain,"institution about")
['<s>', 'institution', 'about', 'our', 'government', 'that', '"s", 'strength', 'at', 'this', 'truth', 'is', 'worthy', 'of', 'the', 'market', ',', 'women', 'were', 'talking', 'about', 'the', 'world', '</s>']
['<s>', 'institution', 'about', 'it', 'comes', 'responsibility', 'to', 'vote', 'on', 'right', '.', '</s>']
['<s>', 'institution', 'about', 'america', 'the', 'incredible', 'pride', 'that', 'because', 'we', '’ve', 'got', 'companies', 'that', '’s', 'how', 'great', 'depression', ',', 'this', 'state', 'of', 'our', '</s>']

SMOOTHING AND UNKNOWN WORDS
For handling unknowns, we used a method from the slides to take care of unknown words. This involved substituting all tokens that only appeared once as “<unk>”. We chose this method because it was simple, and wouldn’t result in too many words being marked as unknown.

To smooth our unigrams, we chose to implement add-1 smoothing. Although it is not the most recommended method, we felt that it was simple and straightforward, and would work well enough for unigrams.

For bigrams, we chose to implement the Kneser Ney method for smoothing as it was taught in class and is widely considered one of the most efficient methods. We referenced the Jurafsky textbook for the formula. For better time efficiency on code, we created Lambda and PKN dictionaries to pass into our

"smoothed_bicount" function. Furthermore, we set *d* to 0.75 since professor cardie used the same constant for n-grams where *n* >= 2 in lecture.

PERPLEXITY

Using the second perplexity formula on the handout, we calculated the perplexity of our smoothed unigrams *and* bigrams for Trump and Obama's speeches and ran them on the development set for both Trump and Obama. Trump's perplexity came out to 58.5715979117 and Obama's perplexity came out to 122.184285917 for bigrams, and for unigrams, Trump's perplexity came out to ~ 20 and Obama's to ~24.

CLASSIFICATION

We used perplexity to compute speech classification, as it is straightforward and simple and suggested by the handout. To predict the classification, we first computed Trump's train dataset perplexity on the development set and Obama's train dataset perplexity on the development set. Then for each paragraph, we calculated its perplexity using Trump's smoothed bigram probability and its perplexity using Obama's smoothed bigram probability. We then used these two perplexities to see which one has a tighter fit to their respective calculated perplexity from the development set. From here, we then know that whoever's perplexity delta is the smaller one of the two is thus our predicted speaker.

EVALUATING WORD EMBEDDINGS

In this section of the assignment we chose to work with word2vec and glove, as they were the first two mentioned. In the table below we report the percentage accuracy and the total number correct responses over the total number of analogies, in other words each line from the dataset. For word2vec and glove we used the Google News dataset.

As seen below, word2vec is slightly worse at predicting the fourth word in each line of analogy_test.txt than glove. As the assignment explicitly stated to discuss each pre-trained word embedding's *results*, we will avoid diving into technicalities surrounding setup for the sake of brevity.

	word2vec	glove
Percentage Accuracy	68.78%	72.13%
Fractional Accuracy	9409/13681	9875/13681

We decided to design two analogy types: shapes to their number of sides and antonyms. Example of this can be seen below.

	Shapes to Its Number of Sides Analogy	**Antonym Analogy**
Example 1	square is to four as pentagon is to five	evening is to morning as black is to white
Example 2	triangle is to three as square is to four	past is to future as light is to dim

Example 3	hexagon is to six as pentagon is to five	black is to white as breakfast is to dinner

S = “Shapes to Its Number of Sides Analogy” column
AA = “Antonym Analogy” column

Example 1 for S (square: four, pentagon: __): “five” was second result
Example 2 for S (triangle: three, square: __): “four” was second result
Example 3 for S (hexagon: six, pentagon: __): “five” was third result

Example 1 for AA (evening: morning, black: __): “white” was first result
Example 2 for AA (past: future, light: __): “dim” was second result
Example 3 for AA (black: white, breakfast: __): “dinner” was 9th result/ but “supper” was 10th
* “dinner” and “supper” are synonyms and vary in use across dialects

Word	**Top 10 Similar Words Based on Cosine Metrics**
Sit	sitting, sat, sits, plop, Sitting, flying_thermos_bottle, go, **stand**, stare,
Win	victory, win, triumph, clinch, victories, won, **defeat**, winning, beat, vicory,
Increase	**decrease**, increases, increased, **reduction**, increasing, **decreasing, decline,** rise
Enter	entering, entered, reenter, enters, entry, Entering, participate, **leave**, join, register
Open	Opened, **closed**, opens, opening, reopen, **closes**, **close**, 9am_5pm_Mon, reopened, ajar

To further explain the above table, the results we amassed prove that a given word and its antonyms tend to have similar word embeddings. For example, for the last row in the table, “open”, the word with the highest cosine distance was “opened” at 0.596660, and “closed” (open’s antonym) was a close second at 0.581966. In the case of “increase”, its word with the highest cosine distance was in fact “decrease” at 0.837032. In second place came “increases” at 0.770938. In the case of “win”, “defeat” was in 8th place with a cosine distance of 0.624235. The word “defeat” is an interesting case as it could either occupy the role of a verb or a noun. In the case of a verb it would be a synonym, while in the case of a noun, an antonym to “win”. These examples illustrate a known issue with word embeddings - their tendency to predict high similarities among words with opposing meanings.

Section 6 KAGGLE SCORES: **0.81** -- jpt86, yh467, bds238
Section 8 KAGGLE SCORES: **0.73** -- jpt86, yh467, bds238

SPEECH CLASSIFICATION WITH WORD EMBEDDINGS
EVERYTHING BESIDES sec_7.py IS WRITTEN FOR PYTHON 3
sec_y.py IS WRITTEN FOR PYTHON 2.7

Our code for this can be seen in sec_7.py
It is worth mentioning that we referenced a small chunk of our sec_7.py file from (lines 81-89)
http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/