```
#######################
# TESTING NAIVE BAYES #
#######################
c(k) values:
trump spoke 637 statements
clinton spoke 455 statements

c(k, w) values:
trump said the word country 161 times.
trump said the word president 39 times.
clinton said the word country 84 times.
clinton said the word president 182 times.

p(k) values:
P(trump): 0.20183776932826364
P(clinton): 0.14416983523447402

p(w | k) values:
P(country|trump): 0.004146301915637812
P(president|trump): 0.0010063339844906174
P(country|clinton): 0.0015826447291908564
P(president|clinton): 0.0034268680759293097

p(k | d) values:
perry: 3.6761347263356324e-98
chafee: 6.3426514016717605e-31
walker: 1.2539533988082282e-09
bush: 0.1448595595573337
sanders: 1.9321657669665616e-15
clinton: 0.00037500915248108097
carson: 1.4169347807709393e-14
rubio: 0.8547260721427091
o'malley: 1.6654698701578745e-20
webb: 3.313895649400917e-13
huckabee: 1.9490221315039155e-14
paul: 3.9354021386725584e-05
```

```
trump: 2.133306632004047e-13
cruz: 1.8648430223363127e-09
kasich: 1.7509441241604307e-13
christie: 2.006537208557656e-09
fiorina: 7.587658713319151e-18
```

Final tests:

```
253 correct out of 400 using unigrams
117 correct out of 400 using bigrams
98 correct out of 400 using parts of speech
223 correct out of 400 using all of these factors
```

Implementation Choices:
I did add n smoothing, with n = 0.1. This type of smoothing resulted in the largest accuracy on dev.
I also had to adjust the proportional probability values by a constant multiplicitive factor to avoid underflow when taking the exponential

Bigrams:
I attempted to use bigrams in replacing the unigram model we started with. This meant that instead of finding the probability of a word given a speaker, I was finding the probabilities of pairs of words based on a gived speaker. My new accuracy was 30% with just bigrams, and 53% using both bigrams and unigrams. While this is a decrease in correctness, I have some reasons as to why this was. To start, bigrams are a fairly small n-gram to use. Two words together are not often enough to map to a specific person where one of those words individually would not already do that. The other reason is how I used the bigrams in addition to the unigrams. I simply averaged the probabilities of the unigram and bigram models, where a weighted result may have proven better.

Parts of Speech:
I used parts of speech as my second feature. To do this, I used the NLTK library to map each word to a part of speech, and then treated the calculations exactly as I did for the unigram model. This had a 24.5% accuracy alone, with 55.7% combined. This decreased accuracy also has some possible explanations. First, NLTK was only using individual words, and some words cannot be given accurate parts of speech without the context of the surrounding sentence/statement. Also, many statements are only a few words long, and did not have unique enough parts of speech where

any kind of accurate prediction could be made from it.

Note:
I did not complete these features with the logistic regression model.

```
#################################
# TESTING LOGRITHMIC REGRESSION #
#################################
Iteration number: 1
Negative log likelihood: 1252.6212548609506
Accuracy on dev: 0.315

Iteration number: 2
Negative log likelihood: 699.1032284309434
Accuracy on dev: 0.435

Iteration number: 3
Negative log likelihood: 737.9647865960252
Accuracy on dev: 0.445

Iteration number: 4
Negative log likelihood: 657.4210457482573
Accuracy on dev: 0.48

Iteration number: 5
Negative log likelihood: 801.0429149791579
A ccuracy on dev: 0.4475

Iteration number: 6
Negative log likelihood: 619.8046596108312
Accuracy on dev: 0.5175

Iteration number: 7
Negative log likelihood: 631.3737946357152
Accuracy on dev: 0.5075
```

```
Iteration number: 8
Negative log likelihood: 610.5652554667198
Accuracy on dev: 0.4975

Iteration number: 9
Negative log likelihood: 605.9635755961223
Accuracy on dev: 0.5

Iteration number: 10
Negative log likelihood: 612.5272749159129
Accuracy on dev: 0.51

Iteration number: 11
Negative log likelihood: 604.3126290643296
Accuracy on dev: 0.52

Iteration number: 12
Negative log likelihood: 620.4040423873965
Accuracy on dev: 0.51

Iteration number: 13
Negative log likelihood: 614.5549756020687
Accuracy on dev: 0.525

Iteration number: 14
Negative log likelihood: 617.2588352987474
Accuracy on dev: 0.53

Iteration number: 15
Negative log likelihood: 613.0065283425197
Accuracy on dev: 0.53

Iteration number: 16
Negative log likelihood: 610.7431564497435
Accuracy on dev: 0.525
```

```
Iteration number: 17
Negative log likelihood: 613.3593159962102
Accuracy on dev: 0.5225

Iteration number: 18
Negative log likelihood: 624.3212133513899
Accuracy on dev: 0.53

Iteration number: 19
Negative log likelihood: 617.3258336338514
Accuracy on dev: 0.5225

Iteration number: 20
Negative log likelihood: 620.5504337090883
Accuracy on dev: 0.515

Iteration number: 21
Negative log likelihood: 619.5286480856632
Accuracy on dev: 0.5175

Iteration number: 22
Negative log likelihood: 617.2303937615001
Accuracy on dev: 0.5175

Iteration number: 23
Negative log likelihood: 617.9295195887673
Accuracy on dev: 0.515

Iteration number: 24
Negative log likelihood: 625.7016540508471
Accuracy on dev: 0.525

Iteration number: 25
Negative log likelihood: 618.9934556811277
Accuracy on dev: 0.53
```

```
Iteration number: 26
Negative log likelihood: 614.7562982858399
Accuracy on dev: 0.525

Iteration number: 27
Negative log likelihood: 616.7185485137913
Accuracy on dev: 0.5225

Iteration number: 28
Negative log likelihood: 626.2105335925158
Accuracy on dev: 0.5125

Iteration number: 29
Negative log likelihood: 620.2356344797174
Accuracy on dev: 0.525

Iteration number: 30
Negative log likelihood: 621.4660756421484
Accuracy on dev: 0.52

λ(k) values:
λ(trump): 1.8127036265623255
λ(clinton): 0.8635480491745451

λ(k, w) values:
λ(trump, country): 0.5382576122215567
λ(trump, president): -0.41727063404993636
λ(clinton, country): -0.2595728887723306
λ(clinton, president): 0.5948347676228543

P(k | d) for the first line of dev:
{'sanders': 0.17058255880726814, 'rubio': 0.07264706293781135, 'walker': 0.0037137370031804277,
'clinton': 0.034555860758487184, "o'malley": 0.07085132035788302, 'carson': 0.05643290970302147,
'perry': 0.002197091253335659, 'bush': 0.052207772060201926, 'webb': 0.006281927380168701,
'huckabee': 0.004748764141122602, 'trump': 0.4207475887615625, 'christie': 0.00698679037689651,
'cruz': 0.024162126947650308, 'kasich': 0.04383773735405467, 'paul': 0.015799220240314747,
'fiorina': 0.012284444945761844, 'chafee': 0.0019630869712790114}

Accuracy on test: 0.56
```

Implementation Choices:
I randomly shuffled the training lines before each iteration, as well as each test set I tested on
I started with a learning rate of 0.1. This allowed for quick increases in accuracy initially, but I decreased this value by 5% each iteration. This made the steps smaller towards the end.
I chose 30 iterations because at this point, the learning rate is small enough that the model should be hovering around its maximum. Note: $.95^{30} \sim= 20\%$ of the original learning rate.
for $\lambda(k)$, I assumed there was a dummy word (the empty string in my case) that occurred once per document.