Sentiment Analysis of Characters and Episodes from *The West Wing*

Natural Language Processing

Spring 2016

Prof. Yamangil

Obaid Farooqui and Taher Mun

Abstract

The West Wing is an intricately produced political show, with rich, dynamic characters and intelligent plot arcs. For this project we created a sentiment analysis system that utilizes two techniques, bag-of-words and MaxEnt, to label each line in an episode as being either positive, negative, or neutral. We use these polarity scores to determine each character's agreeability, as this is an important trait when it comes to political maneuverability, and we use our findings from this analysis to derive insights about character development on the show.

Introduction

We were interested in exploring sentiment analysis, and it seemed a natural choice to perform it on a TV show. *The West Wing* in particular is really interesting because it revolves around political debates and maneuvering, with characters having to carefully deal with all sorts of situations, crises, and interactions.

Agreeability is very important for characters to do well, and we hypothesized that we could take a read on a character's agreeability-- as well as the overall mood of an episode-- by comparing the number of lines delivered that had a positive sentiment to the ones with a negative sentiment. Characters that were more positive would be more agreeable; those who had more negative lines were disagreeable. The interesting thing we wanted to look into, however, was how characters

evolve over time in the show, or over the course of specific episodes: do plot twists and arcs lead to noticeable shifts in a character's agreeability?

There were two NLP techniques that we chose to explore for this, as essentially what we had was a classification problem. The first was a naive bag-of-words method that used a dictionary of positive and negative words and simply labeled a line according to whichever sentiment it had seen more words of (it would label neutral in the case that none of the words of a line matched belonged to either the positive or negative dictionaries).

The second technique we employed was MaxEnt. We came up with a list of features: does a line have an exclamation point, how many emphatic, positive, and negative words it has, as well as the unigrams and bigrams and how often those were found to be parts of positive or negative lines in the training set. We calculated weights using maximum likelihood estimate.

We did a fair amount of reading before starting. We had originally aimed to explore characters' emotions, but we found that it was hard to find useful data sets relating to emotions, there was a lot more work done and data available for simple polarity. We chose to focus on agreeability since we could use this data to extrapolate agreeability. We also did some reading to find the best techniques to use for sentiment classification, and we found that MaxEnt had been used with decent results and given proper training data could perform well.

Dataset(s)

For our training data, we had two datasets. The first were two lists of positive and negative words that we got from the NRC Emotion Lexicon. The NRC Emotion Lexicon has a long list of words and emotions, and a bit representing whether or not that word indicates that emotion. We parsed through this list and for each word added it to a positive list if that word's positive bit was on, and added it to the negative list if its negative bit was on. For bag-of-words we'd simply look up a word in these two lists and assign a line a sentiment based on that. So a sentence like "I am

happy to see you" would be classified as positive because of a positive word "happy" and a lot of neutral words, whereas a sentence such as "I hate this" would be negative because "hate" is a negative word.

For our MaxEnt model we used the Stanford Sentiment Analysis corpus. It has a corpus of movie reviews, each of which is on a scale of 0 to 1, with 0 being the most negative and 1 being the most positive. The readme for the corpus explained that values between 0.4 and 0.6 were neutral, and how high or low a number was above or below that range indicated how strongly positive or negative the sentiment was. What's really useful about this dataset however is that the reviews are also broken up into phrases, and each phrase is individually labeled as well. So for instance, "a nightmare on elm street" gets a score of 0.33333 which is solidly negative, whereas "a perfect family film" gets 0.93056, which is very strongly positive. Using phrases was much more helpful than using all out sentences, since it is far more likely to see phrases from the movie reviews come up in an episode of *The West Wing* than entire movie review sentences themselves. So, we used the labeled phrases to train our features and our weights for MaxEnt.

Testing data was a different beast, however, since we needed a labeled corpus of *The West Wing* scripts. We looked a lot, even for other TV shows, but we could not find such a corpus, so we buckled down for two and a half hours and hand-labeled each line of Season 1 Episode 9, and then used that episode as our gold-standard. Labeling this episode was not easy-- there were a number of lines where it was difficult to decide between the three categories, as often sarcasm or underhanded comments would be delivered in such a way that they could very easily be neutral and not decidedly positive or negative. We discussed every line that we did not readily agree on and took into account the context, the delivery, the facial expressions, and the intent when making our decisions so that we could label them as accurately as possible. Some example tagged lines look like this (the label is the second line):

```
C.J.
neg
```

```
Yeah. I can't believe my psychic didn't tell me, Toby. Rest assured,
I'm gonna get my twenty bucks back.

TOBY
pos
It's gonna be an excruciating battle, Mr. Justice, one I have no
intention of losing.
```

Methods for parsing each of these datasets into python dictionaries are found in data_processing.py

Experimental method

To conduct a sentiment analysis of our television series data, we used two kinds of classifiers - a "bag of words" classifier, and a multinomial regression, or MaxEnt, classifier.

The bag-of-words model simply looks at a string of text and counts the number of positive, negative and neutral words, assigning a value of 1, 0.5, and 0 to them, respectively, and averages these values. To classify the text, it labels strings with averages between the range of 0.5-1.0 as positive, between 0 and 0.5 to negative, and any string with a 0.5 average as neutral. The positive and negative words are derived from the NRC Emotion Lexicon dataset as described above. Any word in a string that is not defined in this dataset is assumed to be neutral. Because this model naively assigns sentiment based only only on the ratio of the sentiments of the individual words in the string, it was used as a baseline classifier.

The code of the bag-of-words model is found in bag_of_words.py. To conduct a bag-of-words analysis on TV show episode, simply load a list of positive and negative words and an episode into python, and then run the function bag_of_words to update the episode with the sentiment weights.

```
from data_processing import parse_NRC, parse_goldstandard
# load NRC pos/neg words
pos_negs = parse_NRC(
"data/NRC-Emotion-Lexicon-v0.92/NRC-Emotion-Lexicon-v0.92/NRC-emotion-lexicon-wordlevel-alphab
etized-v0.92.txt")
s1e9 = parse_goldstandard("s1e9.txt", 1, 9)
bag_of_words(s1e9)
print s1e9['script'][0]['bow_score']
```

Our main classifier assigns sentiments to strings of text using multinomial regression. This technique requires assignments of features to strings and a set of weights that correspond to these features. The model is trained on a set of previously labeled strings by using the features associated with these strings to create weights specific to each class. These weights, in effect, determine which features are associated with a class.

Features can represent any characteristic of the string in question, ranging from length to the number of times it contains the word “the”. Because features can be so arbitrary, feature selection is an important aspect of this model. We chose to use the following characteristics as features: number of positive words, number of negative words, number of exclamation points, number of “emphatic” words, and the number of appearances for each unigram and each bigram. To determine the weights for each class, we calculate the maximum likelihood estimate of the probability of seeing each class given the features found in the dataset. The MLE estimate for each weight is calculated by dividing the total sum of the values for that feature seen for a particular class by the total sum of the values for that feature seen for all classes. The methods used to develop the MaxEnt model are contained in MaxEnt.py. To train a MaxEnt model, first load the training set and train the weights, then load an episode and use the weights to calculate sentiments:

```
from data_processing import parse_stanford, parse_goldstandard
from MaxEnt import train_MaxEnt, run_MaxEnt
# load training set
```

```
training_set =
parse_stanford("data/stanfordSentimentTreebank/stanfordSentimentTreebank/dictionary.txt",
"data/stanfordSentimentTreebank/stanfordSentimentTreebank/sentiment_labels.txt")
# calculate weights
weights = train_MaxEnt(training_set)
# load test episode
s1e9 = parse_goldstandard("data/s1e9_gold.txt", 1, 9)
# updated sentiment scores for episode
run_MaxEnt(s1e9, weights)
print s1e9['script'][0]['MaxEnt_score']
```

The only third party tools we opted to use were NLTK for its word_tokenize method, and matplotlib for its plotting capabilities.

It appears that the complexity of our approach is not too large - only approaching O(n). This is because the features that we calculate for each of the phrases in the training set only perform one pass through the string. Also, calculating the weights only requires a single pass through all the features. Therefore, this algorithm might be able to scale to larger datasets. To see how the features were calculated, see features.py, which is used in maxent.py

Perhaps the largest challenge in creating a MaxEnt classifier was in choosing the features. Factors such as the number of positive and negative words are important, however, depending on the length of the line these features could be “diluted” by the large amount of bigrams and unigrams present. Therefore, we had to determine a way to pick the best “subset” of features, which is described in the evaluation section below.

Evaluation

For our evaluation we used our hand-labeled gold-standard episode 9 of season 1 of *The West Wing.* The method employed in developing this test set is described at the end of the Datasets section. Our method of evaluation was to run both of our techniques, bag-of-words and MaxEnt, on each line of the test set and then compare their classifications to the gold-standard

classifications, and then see what percent were correct. We also chose bag-of-words to be our baseline, since it's a naive algorithm that simply counts the number of known positive and negative words. It is unrefined and doesn't learn anything except from the lists of words it is fed, and so we would hope to be able to improve on that with more sophisticated models.

We wanted to avoid using too many features, so we sorted our weights by value and tested accuracies using only the top x weights (meaning their associated features). We got the following result:

```
10000 max_ent vs gold: 0.387054161162
20000 max_ent vs gold: 0.387054161162
30000 max_ent vs gold: 0.387054161162
40000 max_ent vs gold: 0.387054161162
50000 max_ent vs gold: 0.387054161162
60000 max_ent vs gold: 0.387054161162
70000 max_ent vs gold: 0.38177014531
80000 max_ent vs gold: 0.406869220608
90000 max_ent vs gold: 0.384412153236
100000 max_ent vs gold: 0.404227212682
110000 max_ent vs gold: 0.392338177015
120000 max_ent vs gold: 0.392338177015
```

We chose to use 80,000 features since this maximized the accuracy of our MaxEnt model. Even with this decision, however, our accuracy for bag-of-words was 0.391, and our accuracy for MaxEnt was 0.407. These are two numbers are too close for comfort, but there are a few reasons why we believe that MaxEnt did not blow out bag-of-words, and also why we still believe MaxEnt to be the better method.

First and foremost, our training data was not great. It would have been helpful to have had a labeled corpus of conversations, as all of the lines in the TV show are in the form of dialogue. We can learn a lot from phrases used in movie reviews, but they are not a perfect substitute for dialogue between people. In addition, the topics covered in the movie reviews tend to revolve around film, and dabble only a bit into the subjects of the films themselves. The scripts that we are analyzing, however, tend to talk about people, events, political issues, and a lot of office banter. So having more relevant training data would have helped, but unfortunately such data is very hard to come by, but hopefully in the near future.

The second reason is that we pruned our features by picking the ones with the biggest weights, since we reasoned that those were the ones that made a difference; the features with the lowest weights were the most inconsequential. The problem with this though is that while we checked to make sure that features (like unigrams or bigrams, since we used these as features) that only appeared once were not used, it is possible that many of our strongest features with the biggest weights only appeared a small number of times in the training set but always indicated a certain sentiment. However, these features may not have appeared in the test set, and because of that were entirely unhelpful. Again, more labeled datasets that would give us more training and testing sets would be helpful.

We still believe that MaxEnt is the better method, however, because it is more careful in making its predictions by defaulting to neutral. When we tried computing accuracy by treating a neutral classification as worth 75% of a correct classification, so as to reward classifying something that's positive as neutral rather than as flat-out negative, we saw that MaxEnt's accuracy jumped to 0.809 and bag-of-word's accuracy jumped to 0.715. This is a much more pronounced difference between the two models, but it also tells us that MaxEnt tends to default to neutral a lot, which is better than it making wild guesses at sentiments. It is better to classify something as neutral rather than the opposite of what it is, and this is something that MaxEnt does better than bag-of-words.

MaxEnt also handles sarcasm better, and given better training data would learn to discern tone even better too. Some telling examples of why MaxEnt is better include the following:

```
  {'bow_score': 'neutral',
   'character': 'MANDY',
   'gold_score': 'negative',
   'MaxEnt_score': 'negative',
   'text': "This isn't funny, Josh."},
```

In this above example, “isn’t” cancels out “funny”, so bag-of-words classifies it as neutral. MaxEnt learned however that the bigram (“isn’t”, “funny”) usually indicates negative, and so it was able to correctly classify this line.

CJ is being sarcastic towards Toby

```
  {'bow_score': 'positive',
   'character': 'C.J.',
   'gold_score': 'negative',
   'MaxEnt_score': 'negative',
   'text': "Yeah. I can't believe my psychic didn't tell me, Toby.
Rest assured, I'm gonna get my twenty bucks back."},
```

In this example, there are more positive words than negative words so bag-of-words fails to recognize that it is actually a negative line. MaxEnt picks up on the dripping sarcasm however (movie reviews often have their fare share of sarcasm in them), and so it correctly labels the line as negative.

```
  {'bow_score': 'negative',
   'character': 'JOSH',
```

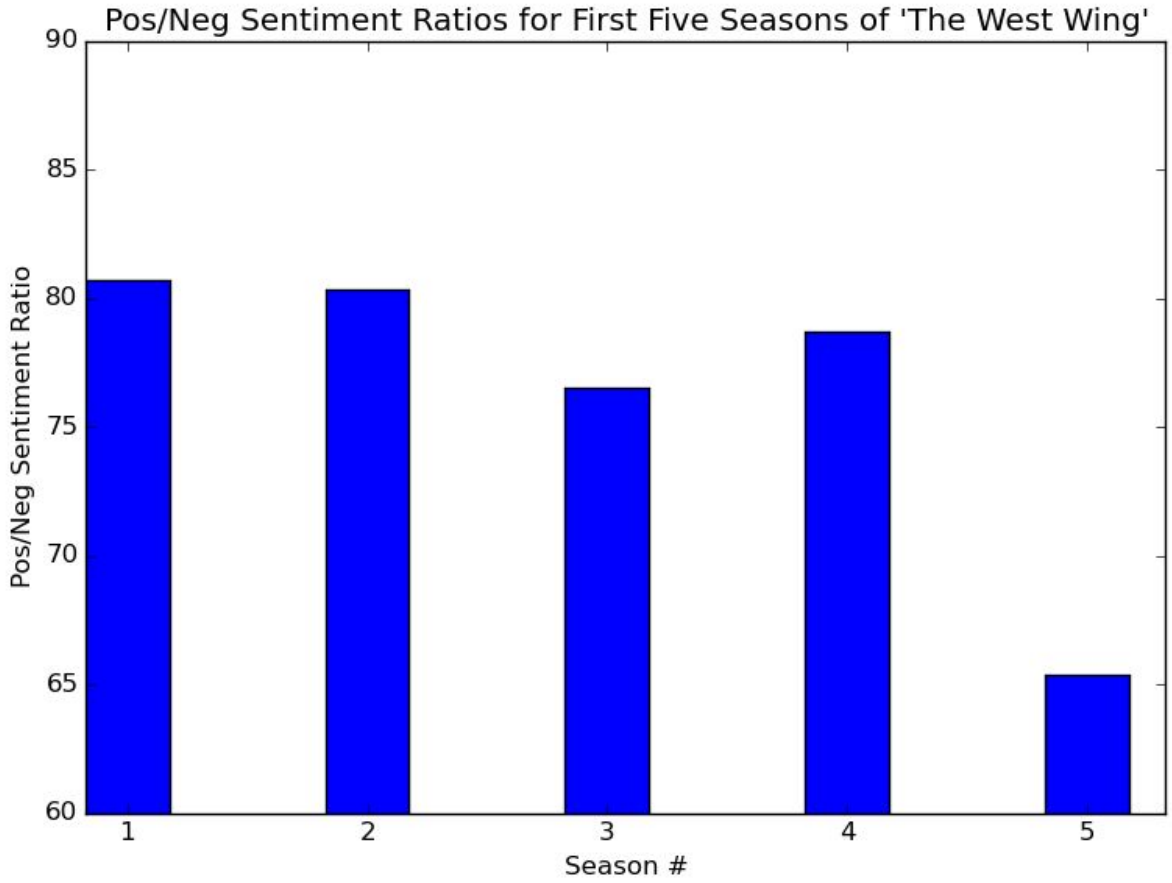
```
   'gold_score': 'positive',
   'MaxEnt_score': 'negative',
   'text': "[laughs] I was interrogating this intern from the
Legislative Liaison's Office, and she broke down crying while telling
me about the bong she had made out of an eggplant."},
```

This is an interesting example where both MaxEnt and bag-of-words were wrong, and did not pick up on the sarcasm. To be fair, the line is packed with negative words like “interrogating”, “broke”, and “crying”, and the the bigrams of the line must also be more readily mapped to negative sentiments in the training set. The stage direction [laughs] also was not enough for either model to pick up on. However, when watching the episode we were able to correctly discern the joke from the tone and the circumstance.

Of the future work that we would like to do to improve our models, first and foremost is finding or creating better training data. Second, we’d like to take a better look at what features might be of use. It might be useful to take a look at what topics are being talked about in a line, for instance, and use topics as features. Or perhaps we could build up knowledge about a character, including their political leanings and temperament thus far in the show, and take that into account when making decisions. In essence, better features and better training data would help us improve.

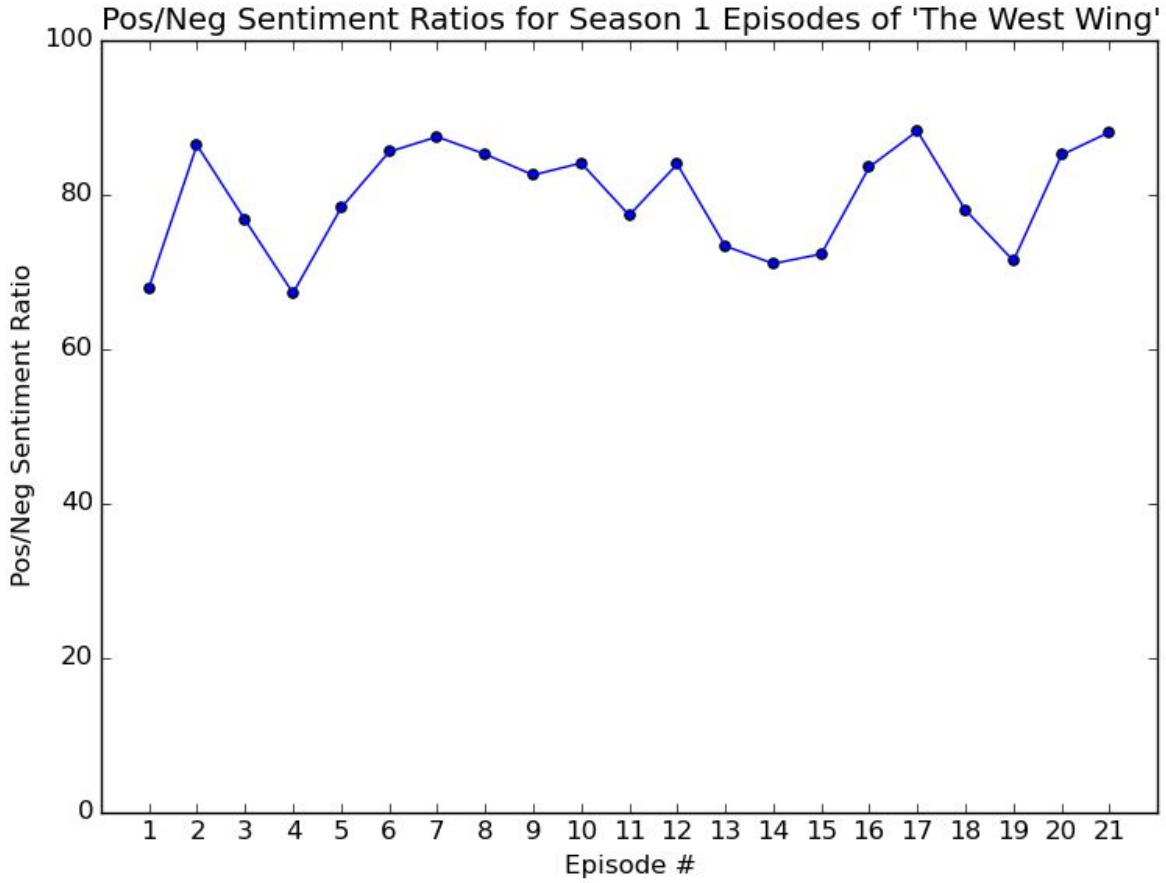
Interesting Insights into *The West Wing*

We were able to use our MaxEnt model to find some really interesting trends. We had the complete scripts for the five seasons of the show, so we ran a number of different queries to see how individual characters changed over time, as well as how the mood of the show itself, based on all of the characters’ lines, changed over the seasons.



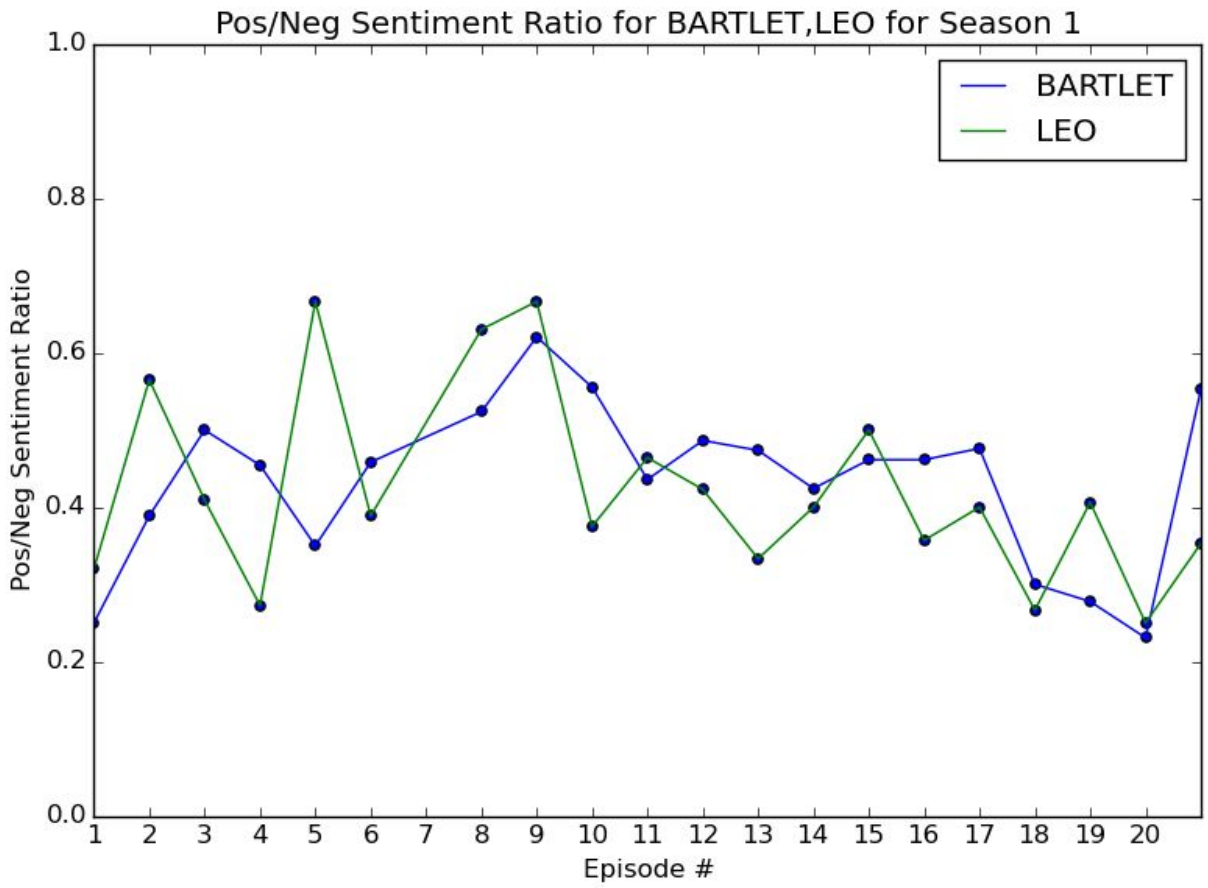
To start, we took a general aggregate temperature of the sentiment of the show by season. We did this by running MaxEnt on every line of every episode and taking the ratio of positive to negative lines in an episode . Next we took the average ratio across all of the episodes in a season (note that in this figure, the y-axis starts at 60%). For the first four seasons the show stays relatively stable, with about three times as many positive lines as negative ones. It dips a bit in season 3, which makes sense because the secret that President Bartlet suffers from Multiple Sclerosis is leaked, and much of the season deals with the personal and political fallout from that revelation. The fact that in addition to the normal ups and downs of political life something so personal is introduced to the show is reflected by a slight lowering of the ratio of positive to negative lines.

There is a stark drop in season 5, however, and this is best explained by the show's writer Aaron Sorkins leaving the show after the first four seasons, and being replaced by John Wells. John's style is markedly different from Aaron's, and this is reflected by the lowered ratio as shown in the graph.



This chart shows an interesting trajectory of the show over the course of the first season. We see that it starts out with a fair amount of drama, with agreeability (the ratio of positive to negative lines) being about as low as it gets in the first episode before spiking up. For most of the season, especially from episodes 3 to 15, the episodes follow a sort of curve and has a more gentle change in tone. However, the positive-negative ratio drops significantly from episodes 17 to 19. Within the show's universe, this time point represents a period of great turmoil within the

fictional “Bartlet Administration” because it is when one of its members is facing accusations of drug abuse and the senior advisors are finding themselves politically stagnating. These issues get resolved in the next two episodes which constitute the season finale.



Lastly, we found it interesting to look at President Bartlet’s and his Chief of Staff Leo’s sentiment over the course of the first season. As we might expect, the Chief of Staff deals with a lot of the same issues that the President does, and so they both tend to move in the same direction.

One interesting data point is in episode 5, where Leo’s sentiment is noticeably more positive in relation to that of President Bartlet. This uptick can be attributed to Leo’s joy in making his

subordinates participate in a “Big Block of Cheese Day”, where they are required to listen to the (oftentimes silly) concerns of any concerned party who cannot otherwise gain access to the White House.

Episode 4 is also interesting because it also shows a noticeable difference in sentiment between President Bartlet and his Chief of Staff. In this episode, Leo has to juggle stress from an impending divorce with his wife and stress from trying to prevent the loss of an important vote, while the President is largely uninvolved in the episode.

Conclusions

This project was a very interesting foray into the Wonderland of sentiment analysis; there are so many minute cues that we as humans pick up on to discern sentiment that it is a formidable challenge to try and train computers to do the same.

With regards to this project, given more time we’d like to extend it to more than just TV shows to encompass other formats, like books, articles, and interviews. As mentioned at the end of the evaluation section, we’d like to find or create significant and better training and testing data to make our models more robust, and we’d like to try more involved features than the ones we’ve used here.

On the subject of features, one thing we’d also like to try is introducing context. We analyze each line in isolation; it would be interesting to split a script into different sections take into consideration the first line when classifying the second, and so forth. Often times a conversation will take a particular emotional arc; when one character starts being negative others follow suit, for instance. Figuring out how to track this and how much weight to give context would be an interesting challenge to take on.

We'd also like to explore using Support Vector Machines as an alternative to MaxEnt. SVMs are useful for classification problems, which is what we have, and it would be worth it to see how given the correct training how an SVM would stack up to MaxEnt in determining sentiment.

TV sentiment analysis could be very useful for reviewing and critically analyzing human speech in general: shows are structured, have plots, character development, drama, romance, and comedy. They offer snippets of life, and being able to train on them brings us closer to systems that can understand natural language more completely. The MaxEnt classifier, though not shown to be too accurate in this paper, can be an effective way to do this if better training data were available or if more useful features could be extracted. There is room for improvement as outlined above, and we hope to explore this in the future.