Exploring Orwell's Rules of Writing

A Study of Natural Language Processing and Journalism

Michelle Carney, Gabe Nicholas, Sayan Sanyal, and Natasha Timakova

Introduction

Orwell's Rules

In his 1946 essay "Politics and the English Language," George Orwell laid out six rules for good writing. They are as follows:

1. Never use a metaphor, simile, or other figure of speech which you are used to seeing in print.
2. Never use a long word where a short one will do.
3. If it is possible to cut a word out, always cut it out.
4. Never use the passive where you can use the active.
5. Never use a foreign phrase, a scientific word, or a jargon word if you can think of an everyday English equivalent.
6. Break any of these rules sooner than say anything outright barbarous.

The objective of this project was to create a tool that would take in a text and highlight the words that break Orwell's rules. Our end product – an online interactive page – both takes in a text from a user and has several preloaded excerpts from Pulitzer Prize winning writing, local newspaper articles, and George Orwell's novels.

The original intent of this project was to create a metric for each of the five rules to measure to what extent a given text conformed to each rule. However, we found that identifying many of the rules was so difficult that our comparative metrics would be entirely meaningless (although they are still provided in the Results section.) Therefore, we instead created an exploratory tool that allowed users to see how our algorithms did on each of the tasks.

Defining the Rules

Coming up with operational definitions for these rules that could be translated into computational classifiers proved to be one of the toughest parts of this project. Some of the linguistic phenomena we were looking for, especially metaphors and jargon, were difficult to define because they were inherently semantic. Our research in the field as well as our conversations in class showed that methods to extract the figurative meaning unreliable, extremely complex, or both. Therefore, we used lexical and syntactical features instead of semantic ones coupled with probabilistic classifiers.

To make the scope for our project manageable, we defined and limited Orwell's rules in the following ways:

- **Rule 1**: We distinguished between trite and non-trite similes but did not make the same distinction for metaphors. We also ignored other figures of speech such as metonymy, synecdoche, oxymoron.
- **Rule 2**: We tried to find which words could be replaced by more simple words. Our definition of "simple" does not just mean shorter in number of syllables: it also needs to incorporate how often it appears in the English language. For example, "specs" should not be considered a replacement for "spectacles."
- **Rule 3**: Instead of trying to learn unnecessary words, we used a finite list of common extraneous words and phrases from the Purdue Online Writing Lab. We considered them to be able to be cut out in all cases.
- **Rule 4**: We considered passive voice to be always replaceable with active voice, although we admit, in practice, there are exceptions to this rule.
- **Rule 5**: We defined jargon (words in professional vocabularies), scientific words, and foreign words as words in these three domains that have everyday English equivalents.

The project can be seen running at http://tinyurl.com/nlporwell. The code for the project can be seen at GitHub here.

Results

Our end success at measuring Orwell's rules was very difficult to measure. We have texts that we marked up with Orwell's five rules but these measures are so subjective that we often disagreed on how to apply them. Since we each used our own word tokenizers, we were unable to check our results by word. Instead, we had to check results by character. Some of our rules also had overlapping tags (for example, two jargon words overlapping) and our final tag writing algorithms each handled this slightly differently. Only having character to character comparisons made it difficult when we accidentally changed something in our human marked texts. If we added one extra character, it made the text entirely useless. In a future version of this study, we would all share code for word chunking, sentence chunking, and text tagging so that these errors would not occur. We would also work together more closely on tagging the texts so the results would make sense.

Below are two human marked texts and computationally marked texts compared side by side. The first is an article from Bloomberg by Dana Hull and Christopher Martin entitled "Tesla seals $2 billion SolarCity deal set to test Elon Musk's vision." The second is a short story by George Orwell called "A Hanging."

Human Marked Up

> Tesla Motors Inc. and SolarCity Corp. shareholders approved the electric-car maker's purchase of the solar installer in a deal designed to test their shared Chairman Elon Musk's vision for a viable <rule5>one-stop shop</rule5> for clean energy consumers.
>
> More than 85 percent of Tesla shares voted in favor of the deal, the Palo Alto company said Thursday, adding that SolarCity shareholders also approved the <rule5>acquisition</rule5>. The deal, valued at about $2 billion, integrates the maker of Model S and upcoming Model 3 sedans with the San Mateo installer of rooftop solar panels.
>
> Shareholders are signing off on Musk's plan to combine and more efficiently run two companies that have <rule1m>a track record</rule1m> for fleeting profits and frequent <rule5>fundraising</rule5> needs. Tesla has lost about $4.8 billion in market <rule5>capitalization</rule5>

since its initial offer to buy SolarCity on June 21, while the latter company's value declined by about $86 million.

Tesla has forecast that SolarCity will add $1 billion in revenue to the combined company next year and $500 million in cash to its balance sheet over the next three years. Joining Tesla's retail network with SolarCity's installers and consolidating the two companies' supply chains may result in an estimated $150 million in savings within a year.

Musk owns 21 percent of Tesla and 22 percent of SolarCity, making him the largest shareholder of both companies. He and Antonio Gracias, who also serves as director at both companies, recused themselves from a board vote on the takeover July 30. <rule5>The all-stock deal</rule5> is worth $20.23 per share, a premium of 2 percent based on SolarCity's closing price Wednesday. The premium was about 35 percent when first announced.

Tesla <rule1m>shares rose</rule1m> 2.6 percent to close at $188.66, while SolarCity <rule1m>shares gained</rule1m> 2.9 percent to $20.40.

The quarterly profit Tesla reported last month was the first in eight quarters. SolarCity has recorded losses in six of the last eight quarters. The two companies have <rule2>conducted</rule2> five separate equity offerings since SolarCity first sold shares to the public in December 2012.

Investor Jim Chanos, whose firm Kynikos Associates Ltd. saw weakness in Enron Inc. before its collapse, has been highly critical of the merger, in part because of the $2.89 billion in SolarCity debt Tesla will be taking on.

The deal drew mixed recommendations from proxy advisory firms, with Institutional Shareholder Services giving its blessing and Glass Lewis & Co. rejecting it as a "thinly veiled bailout plan." Institutional Shareholder Services said Tesla would be able to bridge <rule5>cash-burning</rule5> SolarCity's funding gap and called the deal a "necessary step" in the electric-car maker's push to become an integrated sustainable energy company.

The combined company's attention will be on <rule5>President-elect</rule5> Donald Trump's policies. The real estate mogul has vowed to relax environmental regulations and has tapped Myron Ebell, a climate-change skeptic, to head his Environmental Protection Agency transition team.

Gordon Johnson, an analyst at Axiom Capital Management Inc., downgraded seven solar companies Tuesday, citing his expectation for less favorable renewable energy policies from Trump's administration.

Computationally Marked Up

<rule1m>Tesla Motors <rule5>Inc</rule1m>.</rule5> and <rule1m>SolarCity <rule5>Corp.</rule5> shareholders approved the <rule5>electric-car</rule5> maker's</rule1m> purchase of the <rule1m>solar installer in a deal designed to test their shared Chairman Elon Musk's vision for a viable <rule5>one-stop</rule5> shop for clean energy consumers.

More than 85 percent of Tesla shares voted in favor of the deal, the Palo Alto company said Thursday, adding that SolarCity shareholders also approved the <rule5>acquisition</rule5>. The deal, valued at about $2 billion, integrates the maker of Model S and upcoming Model 3 sedans with the San Mateo installer of rooftop solar panels.

Shareholders are signing off on Musk's plan to combine and more efficiently run two companies that have a track record for fleeting profits and frequent fundraising needs. Tesla has lost about $<rule5>4.8</rule5> billion in market capitalization since its initial offer to buy SolarCity on June 21, while the latter company's value declined by about $86 million. Tesla has forecast that SolarCity will add $1 billion in revenue to the combined company next year and $500 million in cash to its balance sheet over the next three years. Joining Tesla's retail network with SolarCity's installers and consolidating the two <rule5>companies'</rule5> supply chains may result in an estimated $150 million in savings within a year.

Musk owns 21 percent of Tesla and 22 percent of SolarCity, making him the largest shareholder of both companies. He and Antonio Gracias, who also serves as director at both companies, recused themselves from a board vote on the takeover July 30. The all-stock deal is worth $<rule5>20.23</rule5> per share, a premium of 2 percent based on SolarCity's closing price Wednesday. The premium was about 35 percent when first announced.

Tesla shares rose <rule5>2.6</rule5> percent to close at $<rule5>188.66</rule5>, while SolarCity shares gained <rule5>2.9</rule5> percent to $<rule5>20.40</rule5>.

The quarterly profit Tesla reported last month was the first in eight quarters. SolarCity has recorded losses in six of the last eight quarters. The two companies have conducted five separate equity offerings since SolarCity first sold shares to the public in December 2012.

Investor Jim Chanos, whose firm Kynikos Associates <rule5>Ltd.</rule5> saw weakness in Enron <rule5>Inc.</rule5> before its collapse, has been highly critical of the merger, in part because of the $<rule5>2.89</rule5> billion in SolarCity <rule5>debt</rule5> Tesla will be taking on.

The deal drew mixed recommendations from proxy advisory firms, with Institutional Shareholder Services giving its blessing and Glass Lewis & <rule5>Co.</rule5> <rule2>rejecting</rule2> it as a "thinly veiled bailout <rule5>plan."</rule5> Institutional Shareholder Services said Tesla would be able to bridge <rule5>cash-burning</rule5> SolarCity's funding gap and called the deal a "<rule5>necessary <rule5></rule5>step"</rule5> in the <rule5>electric-car</rule5> maker's push to become an integrated sustainable energy company.

The combined company's attention will be on <rule5>President-elect</rule5> Donald Trump's policies. The real estate mogul has vowed to relax environmental regulations and has tapped Myron Ebell, a <rule5>climate-change</rule5> skeptic, to head his Environmental Protection Agency transition team.

Gordon Johnson, an analyst at Axiom Capital Management <rule5>Inc.</rule5>, downgraded seven solar companies Tuesday, citing his expectation for less favorable renewable energy policies from Trump's administration.

As you can see, not only were our results sometimes off but our placement of the tags were as well. We also ran a character-by-character comparison analysis on the two texts. Unfortunately, this means that any errors in parsing quickly become systematic errors shifting the whole document over one character and thus throwing off the analysis entirely. This definitely seems to have happened with the Orwell document as you can see below. This way of measuring results greatly values underfitting. We decided not to expend much effort in finding a more balanced way of measuring our results because they were so incomparable anyways. You can see the percentages below:

Text	**Rule**	**Percent Correct**
Bloomberg	1s	N/A
Bloomberg	1m	0.964
Bloomberg	2	0.994
Bloomberg	3	N/A
Bloomberg	4	N/A
Bloomberg	5	0.957
Orwell	1s	0.115
Orwell	1m	0.080
Orwell	2	0.065
Orwell	3	N/A
Orwell	4	0.065
Orwell	5	0.070

Table 1: Character-By-Character Results

Navigating This Project

The results of this project are less interesting than the methods we used. In the sections below, we will go rule by rule and discuss our background research, how we implemented it, how we think it did, and what we think we could improve in the future. For those looking to follow long in the code, the project is organized into different folders for each rule. (Note that for the rest of this project, metaphors and similes are considered to be separate rules.) All of the rules are brought together in the file `orwell.py` in the `Web App` directory. The web app itself is instantiated in `webapp.py` in the same directory.

The project can be seen running at http://tinyurl.com/nlporwell. The code for the project can be seen at on GitHub here.

Rule 1: Similes

Research

Not every comparison is a simile. Similes are a type of comparisons, which, like metaphors, involve the juxtaposition of one thing with another thing of a different kind and are used to make a description more vivid. Compare a simile "…her eyes, like diamonds…" to a plain comparison "…her eyes, like his, were brown."

However, unlike metaphors, similes have a marker word – a comparator. Most often, words "like" or "as" are used as comparators. For example, "as brave as a lion," "they were like two hummingbirds." Therefore, finding a pattern and using it to create a dependency parsing model to detect similes seems a feasible solution.

As some researchers state, the typical simile can be represented as [tenor comparator vehicle], where tenor is usually a noun phrase or verb phrase, comparator is 'like' or 'as' (however, "as if," "like if," "than," and other are also possible), and vehicle is another noun or verb phrase (Fishelov, 1993). Also, additional adjectives and adverbs can come into the picture. According to other researchers (Hanks, 2012), the components of a simile are as follows:

- Topic (typically, noun phrase): obligatory
- Event or state (verb): obligatory
- Property (typically, adjective): can be either explicit or implicit
- Comparator: optional
- Vehicle (noun, verb, or adj.): obligatory. Just to make things more complicated, let's look at one of the most used comparators – "like." It can be used as a verb, a noun or as a preposition, when it actually serves as a comparator. But as a preposition, it also has several applications. According to Patrick Hanks, aside from use in similes, main uses of "like"-preposition include (but not limited to):
- Plain comparison ("John is like his father")
- Ad-Hoc set creation (inclusive set: "people like doctors and lawyers")
- Reporting perception: looks like, feels like, etc. ("it feels like a velvet").

Taking everything above into account, is it even possible to build a robust pattern for simile detection? Vlad Niculae (Niculae, 2013) explored this problem, comparing precision and recall in simile recognition using 53 tagged similes from Amsterdam Metaphor Corpus[1]. Apparently, dependency parsing or constituent parsing don't work significantly better than Lexical Baseline method (LexBL) – simple extraction of sentences containing comparators "like" and "as" after they are POS-tagged as prepositions. Recall for the latter is, obviously, 1.0, while dependency parsing omits some similes and gives R= 0.717 at most (table 1). Precision is a little lower in simple extraction (P = 0.166), but neither dependency, not constituent parser yield a significantly better result (P = 0.303-0.241).

System	**P**	**R**
LexBL	0.166	1.00
Constituent parsing with GLARF	0.303	0.434
Dependency pattern matching	0.241	0.717
DepSem*	0.252	0.717

Table 2: Excerpt from Niculae, 2013.
** The DepSem heuristic filters out matched comparisons with similarity scores above a manually set threshold, under the assumption that comparisons against highly similar things are unlikely to be figurative (Niculae, 2013).*

At that point, we decided to sacrifice precision for recall and continue with a simple extraction, hoping to filter out not relevant phrases later. Since our task is to extract overused similes, and not just similes, we assume that frequent stable collocations with "like" and "as <…> as" will be either grammatical patterns like "as soon as possible", or trite similes. Grammatical patterns are finite and therefore can be filtered out with regular expressions. This is how we should to be left with trite similes, which is the goal. The only problem now is that trite similes may differ in comparator ("as slow as a turtle" is essentially the same as "slow like a turtle") and have some modifications (for example, include adjectives: "slow like a slowest turtle"). To account for these two issues, we can replace a comparator with a generic word (for example, "comparator") and write a fuzzy matching search algorithm.

With the code described in the previous paragraph, we mined overused similes from a big corpus containing 5000 titles of pulp fiction books and mediocre romance novels. The more times a simile candidate appears in the resulting list, the bigger chance it is a trite simile. We manually set this threshold at 20 simile repetitions, as this seemed to be catching an optimally banal similes (subjectively). In theory, the bigger the initial corpus, the higher should be this threshold. All mined similes were recorded into a pkl-file (29.4 Mb) and stored.

With a testing text, we reused part of the above code in order to extract similes candidates. Then, we fuzzy-matched these candidates with the trite similes corpus from our pkl-file. Candidates that matched considered to be trite similes. However, candidates that don't - might be not similes at all.

Description of the Data

We used Archive.org and Gutenberg.org to source our data. Over 1,7 Gb of pulp fiction and romance books and magazine publications – about 5,000 texts – were traversed to extract almost 30 Mb of repetitive similes. Initially, we wanted to use The Daily Mail corpus, kindly shared by Google DeepMind and compiled by Kyunghyun Cho of NYU. But these articles turned out to be relatively poor in figurative language, so we opted for fiction. Generating the trite similes corpus took under 10 machine-hours.

Description of Algorithm

First part of the algorithm is building an overused similes corpus. Iteratively, file by file, take in the large fiction corpus; tokenize and normalize sentences, traverse through them, POS-tagging each and looking for sentences containing prepositions (tag = "IN") "like" and "as-as", and exclude, using regular expressions, grammatical constants like "as soon as," "as well as," "as usual," "like that." Out of these sentences, cut out simile candidates, which is essentially a segment of a sentence, containing comparator; replace ”likes“ and ”ases“ with a ”comparator”.

Next goes a fuzzy matching stage. Take the list of simile candidates and compare each of them to the rest of them. If 75% words match (for example, 3 out of 4), count this match as 1. All similes that match with other 20 similes are considered to be trite (20 is subjective; can put a higher num to get really trite similes; generally, the bigger the training set, the higher might be the number). Write all trite similes into a pkl-file. This is our final collection.

With a testing set, repeat all steps up to fuzzy matching. Then, instead of fuzzy matching candidates across the testing set, fuzzy match them with the trite similes collection. Output sentences that match. In our presentation webpage, we highlighted trite similes with dark blue. As of today, we are not aware of a reliable way to tell a comparison or ad-hoc set creation from a simile using NLP tools. However, we considered identifying *all* similes to be outside the scope of this project.

In order to assess quality of the solution, we compared our testing set results to the tagged version of this testing set. Recall of 96% shows that almost no trite simile was left out.

Conclusion and Future Work

For the future work, we consider calculating precision of this method; using even bigger and versatile collection of initial data for trite simile mining; apply a bigger array of regexp to filter out non-simile phrases. It might be also fruitful to look into applying several different dependency parsing patterns for better simile segmentation (right now, we are taking up to 6 words after "like" and up to 8 words after first "as" in "as-as" similes). But it will make sense only if we will develop a pattern for every type of simile. We can also employ WordNet to calculate semantic distance between a tenor and vehicle, and if the distance is short, discard these similes candidates as simple comparisons.

Work on similes may also inform efforts in detecting metaphors. In many cases, a metaphor is essentially a simile without a comparator. So we probably could use our simile corpus (after eliminating comparators) to train metaphor classifiers.

Rule 1: Metaphors

Background

Metaphors, like similes, are easy for humans to understand but difficult for computers to identify. Unlike similes, they do not have a consistent structure: they can be adjacent to the words they modify or span an entire text. They do not use a set of distinguishing words – virtually any word can be a metaphor at the discretion of the author. Even some of the most forefront research on this topic only achieves 80% accuracy with metaphor detection (Tsvetkov, Boytsov, Gershman, Nyberg, & Dyer, 2014). For the scope of this project, we were interested in applying the techniques we learned in this course to metaphor detection in order to better understand what is classified as a metaphor.

There are many types of metaphors, but we are most interested in trite metaphors. We have defined "trite metaphors" to be phrases of words that are used in everyday speech, and that have adjacent modifiers – whether it is adjective noun (i.e., “boiling rage”) or subject verb object (i.e., “excitement filled street”), similar to Tsvetkov et al. We will be excluding metaphors that are overarching throughout a text.

Description of Data

For this project, we used a list of subject-verb-object and adjective-noun metaphors and non-metaphors from Tsvetkov et. al (Tsvetkov, Github/metaphor, 2014) to create a feature vectors and classifiers. One of the most difficult tasks for this project was finding examples of non-metaphors, but we were fortunate that Tsvetkov et. al provided examples (adj-n metaphors n= 100, adj-n non-metaphors n=100, subj-v-obj n=111, subj-v-obj n=111). We also used the recently published Metanet (n=1184 sample metaphor high-level categories) (Lakoff, 2016), a list of children’s metaphors (n=71 sample metaphors) (yourdictionary.com, 2016), and a sample article on semantic meaning in the brain (Sample, 2016) for testing the metaphor detection. Michelle used BeautifulSoup to scrape the 1,184 high-level metaphor categories from Metanet, and the sample children’s metaphors. Metanet was also tested in the feature vectors and classifiers, but it often was too high level and proved to be less helpful than the Tsvetkov provided metaphor examples.

Description of Algorithms

Calculating Wordnet Distance

Our first hypothesis was that words that were “far away” from each other in Wordnet were more likely to be metaphors. In order to calculate the distance, we used path_similarity, wup_similarity, and lch_similarity. We quickly discovered that distances were not reliable predictors even within small sample tests. For example, in the metaphor “he is the head of the government” – if we take the key part of the metaphor (“head” and “government”) – it was difficult to get the correct synset of “head” and “government” without hand selecting the results, and hand selecting the parts of the sentence to compare similarity to.

Using CountVectorizer and Logistic Regression

Next, we tried using CountVectorizer with logistic regression, similar to the technique we used in the Kaggle assignment. The idea behind this technique is that if we have some features from the Tsvetkov et. al metaphor (and non-metaphor), we could try to predict if certain phrases were metaphors on the bigram or trigram level (the same format as the training data). This worked the best with Logistic Regression, accurately predicting the holdout set at 94% at the highest, but this may have also been because of the small sample size of the Tsvetkov dataset (n=422). It over fitted its predictions on datasets like the Children's metaphors, Metanet, and the sample article, predicting at least 1 metaphor per sentence, even with stopwords and punctuation removed.

Using Naïve Bayes Classifier with Parts of Speech

For our next attempt, we decided to try using the Part of Speech and Naïve Bayes classifier. We trained the nltk.NaiveBayesClassifier on the different parts of speech of each sample from the Tsvetkov et. al metaphor sample. This performed fairly well on the sample Children's Metaphors, and accurately predicted the development set of the data 57% of the time without additional adjustments to the incorrect POS tags. Also, interestingly enough, the POS tags NN=None, JJ=1, and NN=2 (no nouns, 1 adjective, and 2 nouns) were among the top 15 features for distinguishing if a trigram was or was not a metaphor. This was the sample we showed in the demonstration showcase, and we had to reverse engineer our result from being a bag of words to highlighting metaphors in a text, which proved to be fairly difficult.

Calculating a word's "metaphor index" by using glosses

Lastly, after a discussion with Andrea, we decided to take a different approach to determining metaphors: can we determine if a word is likely to be a trite metaphor or not? For this, we returned to Wordnet: trite metaphors could have a variety of definitions in Wordnet, and there should be a range of similarity between the definitions. Our hypothesis is to compare the different glosses of each word for how similar they are, and if they have a low similarity score they are more likely to be metaphorical.

In the example of the word 'up' – a common word in metaphors: we can calculate the similarity of each of the glosses (definitions from the different word senses) to the other glosses and create a similarity matrix that can be converted into a heat map (see Figure 1 below). We then can sum all of the similarity scores and divide by two (since we are comparing A to B and B to A, and we only want to do this once), and then we can divide by the number of unique similarity comparisons. We subtract this value from 1 to get a “metaphor index” – words with a metaphor index > 0.9 are more likely to be used in the metaphorical sense since it is built into their definition on WorldNet.

In order to better inform our metaphor index, we checked with the top occurring words of Metanet and calculated their index below. We found that top-occurring Metanet words like “verticality” do not match this index (see calculated indexes below), but that may be because they are too high-level and only their concepts are used in metaphors, not the word itself. Interestingly enough, words like “hand” and “head” have high metaphor index scores >0.93, matching our hypothesis that these are used often as trite metaphors and are built into their definitions on WordNet.

Code

```
l = defaultdict(dict)
a = wn.synsets('up')
b = wn.synsets('up')
for val in a:
    for val2 in b:
        val_comp = ' '.join([word for word in val.definition().split()
          if word not in stopwords.words("english")])
        val2_comp = ' '.join([word for word in val2.definition().split()
          if word not in stopwords.words("english")])
        l[val][val2] = SequenceMatcher(None, val_comp.split(),
          val2_comp.split()).ratio()
sample_df = pd.DataFrame.from_dict(l)
sample_df
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(18.5, 10.5)
plt.pcolor(sample_df)
plt.yticks(np.arange(0.5, len(sample_df.index), 1), sample_df.index)
plt.xticks(np.arange(0.5, len(sample_df.columns), 1), sample_df.columns,rotation = -45)
plt.show()
numer = (((sum(sample_df.sum()))/2))
denom = sum(sample_df.count())/2
meta_index_up = 1- numer/denom
meta_index_up
```

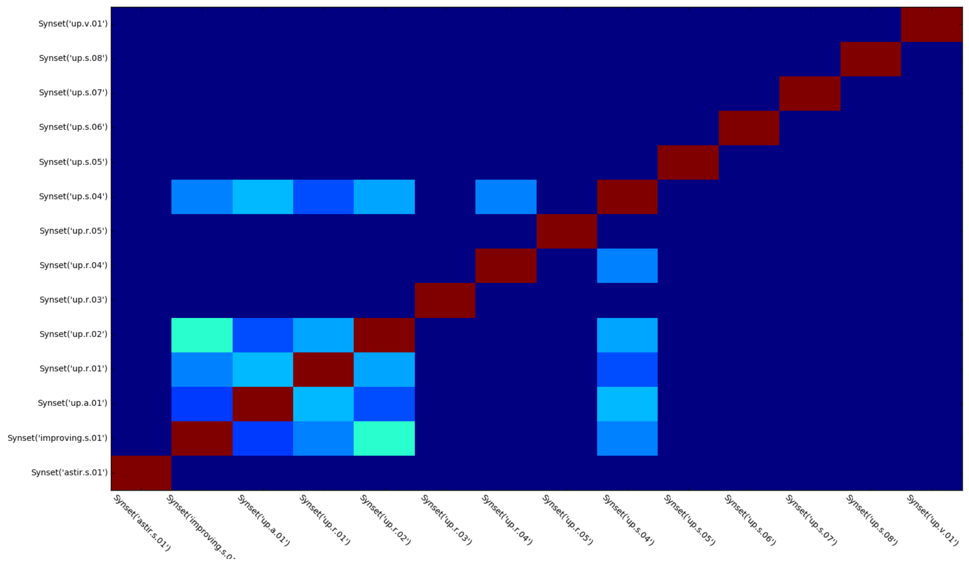

Figure 1: Heat map of the similarities between different glosses of the word "up" and "hand" respectively. Red indicates a value of 1.0, which is where the gloss is 100% similar to itself. The word "up" has more similar glosses than the word "hand," supporting our hypothesis that perhaps the trite metaphor words have these glosses built into their definitions.

Metaphor index of other words:

```
meta_index_hand: 0.933693910256
meta_index_dog: 0.867063492063
meta_index_government: 0.634722222222
meta_index_quilt: 0.518518518519
meta_index_head: 0.966518113147
meta_index_war: 0.789333333333
meta_index_turkey: 0.777142857143
meta_index_physical: 0.820699708455
meta_index_verticality: 0.0
meta_index_up: 0.89878947583
meta_index_metaphor: 0.0
```

Note that words 'government,' 'verticality,' 'war,' and 'physical' were some of the most occurring words in Metanet (all occur >= 15 times), but they have low metaphor indices. This may be due to their concepts being used, but not the words themselves (i.e., "This election was a battle between left and right" does not use the metaphor word 'war').

Conclusion and Future Work

Metaphors are hard to detect for humans and even harder to detect for machines. We have learned a lot about metaphor detection through using a survey of the different techniques used throughout this course. For each attempt we used, it was interesting to see what was and was not tagged as a metaphor (see ipython notebook). The metaphor index is an interesting start for identifying trite metaphors, but it is most interesting to see what it does miss (i.e., 'war,' 'verticality'). Could this be a byproduct of how Metanet is structured and written? If Metanet had more examples of these metaphors rather than high-level categories, one could imagine the initial techniques may have been better for metaphor prediction. For future work, we would recommend repeating a similar technique with hypernyms and hyponyms – this would possibly capture highly metaphorical words like 'verticality.'

Rule 2: Replacing Long Words with Short Words

Background

The most common definition of text readability comes from the Flesch-Kincaid readability score. This measurement considers the number of words per sentence and the number of syllables per word (Kincaid, Fishburne, Rogers, & Chissom, 1975.) Therefore, in our definition of "long words," we considered length to be the same thing as number of syllables. We also included an optional flag for including information about how frequently a word appears in the Google N-Grams corpus because we believed that more common words could be considered more "readable."

Description of Data

To calculate the readability of a word we needed data sources to give us the number of syllables in a word and the commonality of a word in the English language. To get the number of syllables in a word, we used the Carnegie Mellon Pronouncing dictionary, which contains over 134,000 words. For any word that was not in the pronouncing dictionary, we used our own rough algorithm (see below.) To get the commonality, we used the Google N-Grams corpus. We also used a list of the 20,000 most common words in the N-Grams corpus to make our algorithm run more efficiently. Finally, to determine the whether or not two words are the same, we used synonymy in WordNet.

Description of Algorithms

There were two different problems we used algorithms to try to solve: figuring out the length of a word (which we considered to by synonymous with complexity) and figuring out the synonyms of a word.

For the length of the word, we used a score that considered both the tried to weight the complexity and the number of syllables equally against each other. To calculate the number of syllables, we first checked to see if the word was in the CMU Pronouncing Dictionary. If it was there, we used that number. If it was not, made our own rough calculation that counted the number of vowels and subtracted one if there was an “e” at the end. The code looked as follows:

```
def syllable_count():
  syllable_count = 0
  for word in word.split():
      if word in syllable_dict:
          # Shout out to StackOverflow for this snippet of code
          # http://stackoverflow.com/a/4103234/1031615
          syllable_count += [len(list(y for y in x
            if y[-1].isdigit())) for x in syllable_dict[word]][0]
          continue
      # If it's not in the dictionary count the number of vowels and
      # ignore an e at the end not preceded by another vowel. It's
      # rough, but there will be few cases if any cases in which
      # a word is not in the CMU dictionary but in WordNet
      if word[-1] == 'e':
          word = word[:-1]
      word = re.sub(r'[^aeiou]', '', word)
      syllable_count += len(word)
   return max(syllable_count, 1)
```

To calculate the frequency of a word in Google N-Grams, we used the google_ngram_downloader library. We then calculated the percentage of total words in the corpus of a given word. Since this number was always extremely small, we used the log of it. Here is what our calculation looks like:

```
# The number of total ngrams in Google N-Grams, according to their blog
# Blog link here
NGRAM_TOKEN_COUNT = 1024908267229
```

```
def find_frequency_score(word):
    unigram_count = find_google_ngrams_word_count(word)
    percent_occurrence = unigram_count/NGRAM_TOKEN_COUNT
    # Get the log of the frequency to make our number manageable
    # This will be a number between -12 and 0
    freq_val = math.log(percent_occurrence)
    # The frequency value for if the ngram occurred once in the corpus
    min_ngram_val = math.log(1/NGRAM_TOKEN_COUNT)
    # Return the ratio of the word's frequency value to the lowest possible one
    # This is a value between 0 and 1
    relative_freq = ((freq_val - min_ngram_val)/(-min_ngram_val))
    return relative_freq
```

Finally, we combine multiply these two values together. Importantly, we consider any words that are within the 10,000 most common words on Google N-Grams (done in the `is_non_replaceable_word()` function.)

```
def reading_difficulty_for_word(word, ignore_common_words=False, use_ngrams=False):
    if word is None:
        return BIG_NUMBER
    word = word.lower()
    # If it's in the top 10000 most common words, we assume it is readable
    if ignore_common_words is True and is_non_replaceable_word(word) is True:
        return 0
    syllables = syllable_count(word)
    if use_ngrams == False:
        return syllables
    freq_score = find_frequency_score(word)
    return syllables * freq_score
```

To figure out the synonyms of a word we run over the text one sentence at a time and Lesk's algorithm on each sentence tokenized by word. We get use the part of speech according to spaCy to improve the performance of the algorithm. Once we have what we assume is the proper synset paCy to improve the performance of the algorithm. Once we have what we assume is the proper synset, we use the `lemma_names` function to determine the synonyms. We then run the the `reading_difficulty_for_word` function on each lemma and if there is one shorter than the one used, we consider that word to be replaceable by a shorter word.

If you notice, we passed `use_ngrams` as a flag into the `reading_difficulty_for_word` method We did this because although the n-grams frequency score improved our algorithm, it took several seconds per word to run. This meant that it was simply too slow for us to use in our live demonstration.

Conclusion and Future Work

Our program took a very limited definition of what words were considered "replaceable." It missed out on many words that we in our marked texts considered replaceable by a shorter word. Using only words within the same synset overlooked instances where someone uses a word that is not just unnecessarily long but also unnecessarily specific. For example, even though a "tumbler" is something more specific than a "glass," in American English the prior is often gratuitously verbose (same goes with the word "gratuitous.") We also tried checking the hypernym of the word and this sometimes worked surprisingly well when used in concert with the Google N-Grams checker. However, it used too many obscure words without checking the n-grams frequency so we used the shared synset in the final web app.

Were we to improve this algorithm, we could swap out Google N-Grams and instead try to check to see if words get used in the same context. This way, we could combine words in the same synset, the hypernym, and maybe even other hyponyms of the hypernym of the word in question. Then, as we intended to do originally, we could check to see whether the comparison word was ever used in the same context as the original word. In our original write up, we gave the example of checking to see whether "glasses" and "spectacles" were actually interchangeable in the context of the sentence "He wears glasses" by checking to see whether "spectacles" is ever the direct object of "wears." In our preliminary attempts, we found that Brown corpus was not nearly large enough to do this task. This means our corpus would have to be truly huge.

Rule 3: Cut Out Unnecessary Words

Research and Description of Data

Finding unnecessary words was the problem we spent by far the energy time on. In our research, we could not find any papers on the topic, which meant that we could not find any useful data sets of useless words. Fortunately, the Purdue Online Writing Lab came up with two lists of often unnecessary words in their section on Conciseness. The list of words can be found below:

- kind of
- sort of
- type of
- really
- basically
- for all intents and purposes
- definitely
- actually
- generally
- individual
- specific
- particular
- past memories
- various differences
- each individual *
- basic fundamentals
- true facts
- important essentials
- future plans
- terrible tragedy
- end result
- final outcome
- free gift
- past history
- unexpected surprise
- sudden crisis
- large in size
- often times
- of a bright color
- heavy in weight
- period in time
- round in shape
- at an early time
- economics field
- of cheap quality
- honest in character
- of a * condition
- in a * state
- * in nature
- * in degree
- of a * type
- the reason for
- for the reason that
- due to the fact that
- in light of the fact that
- considering the fact that
- on the grounds that
- this is why
- owing to the fact
- on the occasion of
- in a situation in which
- under circumstances in which
- as regards
- in reference to
- with regard to
- concerning the matter of
- where * is concerned
- it is crucial that
- it is necessary that
- there is a need/necessity for
- it is important that
- cannot be avoided
- is able to
- has the opportunity to
- has the capacity for
- has the ability to
- it is possible that
- there is a chance that
- it could happen that
- the possibility exists for

Description of Algorithm

Our algorithm for this was very simple. First, we converted common unnecessary phrase to regular expression. The regular expression excluded anything that was within quotes. Then, we ran that regular expression over the document and considered anything. Finally, we returned the indexes of where each unnecessary phrase was. You can find the code below:

Future Work

For future work to be done on this, we would need a corporus marked with unnecessary words. Then we could train a classifier on that set. Although our approach was far from perfect, it worked surprisingly well. Yes, sometimes there are non-extraneous uses for phrases in our set like “kind of” and “basically.” Many of them though are aphorisms with no use at all—”for all intents and purposes,” “at an early time,” etc. Perhaps the one thing we could have done is try to identify redundancies like “unexpected surprise” or “past history,” but event that would be very difficult to do without a data set to train on.

Rule 4: Never Use the Passive When You Can Use the Active

Research

Passive Voice detection has reached the status of being included in commercially available software like MS Word and Grammarly and is not a general area of computational research. Dependency Parsing, however, is an active area of research with parsers such as Parsey McParseyface from Google's SyntaxNet representing the state of the art in the field.

Description of Data

There was no external dataset used for this part.

Description of Algorithm

We used the dependency parser from spaCy, an industrial strength NLP library. After parsing the text, we went through the sentences which had tokens tagged as 'nsubjpass', 'csubjpass' or 'auxpass' and marked them as passive voice sentences. This meant that sentences that either had a noun, clause or auxillary word being modified by a passive verb were tagged as passive sentences.

```
def rule4_ranges_in_text(article, parser):
    '''
    This function accepts a string of sentences
    and prints them out classifying them into
    active or passive. It returns a list of tuples
    in the format (starting_char_of_passive_sentence,
    length_of_passive_sentence) of sentences that are
    passive.
    '''
    edited_article = remove_quotes_from_text(article)
    parse = parser(edited_article)
    passive_list = []
    for sentence in parse.sents:
        sent = str(sentence)
        hasPassive = False
        passive_indicators = []
        for word in sentence:
            if word.dep_ in ['nsubjpass', 'csubjpass', 'auxpass']:
                passive_indicators.append((word.dep_, word.text))
                hasPassive = True
        if hasPassive:
            passive_list.append((article.find(sent), len(sent)))
            print("Passive Voice Sentence: {0}.\n Passive Voice indicators: {1}"
            .format(sentence, passive_indicators))
        else:
            continue
    return passive_list
```

Future Work

Our approach considered and marked up every sentence in the text that were in the passive voice. However, we did not consider sentences that were in the passive voice but could possibly not be written in the active voice. While a tighter definition of the rule could only include sentences which had an object to the verb, we felt that would skip sentences where that object was implicit. Perhaps an approach that would exhaustively figure out sentences where active voice formulations were impossible could be a way to improve this rule.

Rule 5: Foreign Phrase, Scientific Word, Jargon

Research

While jargon detection from a linguistic standpoint was not something we were able to easily find specific research on, there was literature on jargon detection that included semi-supervised methods. However, these methods all included gathering data on what is considered jargon and learning from that data.

Description of Data

The data set that we ended up using was curated by hand for this project. We tried to capture the different aspects and kinds of jargon that is commonly found in journalistic texts. We amassed a large list of words that were specifically considered jargon, foreign or scientific and tagged it as jargon. To mix in non jargon words, we used the most common words in the English language. We thought that training on the two extremes in terms of words that we would and would not consider jargon would have give us information that we would not be able to encode with marginal cases. The list of data sources were then cleaned up to exclude long phrases, full sentences and general features that made them unsuitable to be considered a single token. The sources and lists of data sources are given as follows:

- Legal Jargon
- Medical Jargon
- Generic Buzzwords
- Business Jargon, and link 2
- French Terms
- Latin Terms
- Non Jargon words

Description of Algorithm

We initially wanted to use a corpus that included the graphs to the Wikipedia corpus to understand how central terms were to topics that were being discussed in the article. However, given the non specificity of the Wikipedia graph and size of the graph, computing similarity of every single token in a graph was not computationally optimal for a real time application.

We tried to compute frequencies using word frequencies and use the word frequencies of WordNet definition of words, however the Google N-Grams API had too high a latency for real time applications.

Finally, we went down the route of the data set and trained classifier. We trained the classifier based on features such as character n grams of the words, the synsets and their height in the WordNet tree and concatenating it with character tfidfvectorizers. We initially used a naive Bayesian classifier, but on testing we found that using a SVM with a linear kernel worked better.

We found that the large number of non jargon text definitely helped us in getting an accuracy of 95.5%. However, this worked much better on our text which mirrored our dataset in that the larger number of non-jargon words. Note our lower recall of 0.79 for jargon texts indicating the lower accuracy when the number of non jargon words were reduced.

	Precision	Recall	F1-Score	Support
False	0.96	0.99	0.97	4018
True	0.92	0.79	0.85	745
Avg/Total	0.96	0.96	0.96	4763

To illustrated that point, we tried to predict which kind of jargon a word belonged to, and while our overall accuracy was still 93%, we could clearly see that certain categories like Latin and French are easier to predict than the others, and that there may not be enough instances in other categories to predict them accurately.

Jargon Type	Precision	Recall	F1-Score	Support
Businesses	0.40	0.27	0.32	44
Buzzwords	0.42	0.25	0.31	52
French	0.69	0.51	0.59	90
Latin	0.91	0.93	0.92	425
Medical	0.55	0.21	0.31	80
Most Common	0.96	0.99	0.97	4000
Average/Total	0.92	0.93	0.93	4763

Future Work

This algorithm only looked at the lexical and structural features of each othe training set without looking at the semantic and contextual meaning of the sentences. We could perhaps look at jargon in situ rather than as individual words to understand when words that are jargon in one context is not jargon in another. Rather than using individual words as datasets, we should use marked up documents in conjunction with a semantic graph in order to understand what features indicate that words are jargon or features in another.

Contributions

Each team member equally contributed to this project.

We divided up Orwell's Rules into:

- Rule 1 (Simile): Natasha
- Rule 1 (Metaphor): Michelle
- Rule 2: Gabe
- Rule 3: Gabe

- Rule 4: Sayan
- Rule 5: Sayan

Although each person took accountability for their Rule, there was some cross-collaboration, but each person assigned to the rule was responsible for designing and developing the approach and solution for their Rule.

Gabe and Michelle put together the interactive demonstration and Natasha put together the one page handout for the demonstration day. Sayan helped streamline the integration of metaphor and jargon detection into the interactive demo, most importantly reconstructing key phrases from a bag of words. Gabe streamlined all the rules together to create a single marked text and created this LaTex document. A friend of the group, Gabriel Fierro, also helped with consulting on the interactive demonstration.

References

- Fishelov, D. (1993). Poetic and Non-Poetic Simile: Structure, Semantics, Rhetoric. Poetics Today, 14(1), 1-23

- Hanks, P. (2012). The Roles and Structure of Comparisons, Similes, and Metaphors in Natural Language (An Analogical System). Presented at the Stockholm Metaphor Festival, September 6-8. Link.

- Kincaid, J. P., Fishburne Jr, R. P., Rogers, R. L., & Chissom, B. S. (1975). Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel (No. RBR-8-75). Naval Technical Training Command Millington TN Research Branch. Chicago

- Lakoff, G. (2016, 10 01). Metanet. Retrieved 11 08, 2016. Link.

- Niculae, V. (2013). Comparison pattern matching and creative simile recognition. Proceedings of the Joint Symposium on Semantic Processing. Textual Inference and Structures in Corpora. Link

- Pal, A. R., & Saha, D. (2013). Detection of Jargon Words in a Text Using Semi-Supervised Learning. Computer Science & Information Technology (CS & IT).

- Sample, I. (2016, April). Neuroscientists create 'atlas' showing how words are organized in the brain. The Guardian , 1.

- Tsvetkov, Y. (2014, 01 01). Github/metaphor. Retrieved 11 08, 2016, from https://github.com/ytsvetko/metaphor

- Tsvetkov, Y., Boytsov, L., Gershman, A., Nyberg, E., & Dyer, C. (2014). Metaphor Detection with Cross-Lingual Model Transfer. Proc. ACL , 1-14. yourdictionary.com. (2016, 11 01). Metaphor Examples for Kids. Retrieved 11 08, 2016, from http://examples.yourdictionary.com/metaphor-examples-for-kids.html

- Welcome to the Purdue OWL. (n.d.). Retrieved December 1, 2016, from https://owl.english.purdue.edu/owl/resource/572/02/