LING 83600/CSCI 76000
Prof. Levitan

HW2: Text Classification

The goal of this HW is to gain experience using NLTK to preprocess text and extract features and use scikit-learn to classify text.

Deadline: Thursday September 22, 11:59pm

Submission instructions: Submit a document with your answers to the HW questions (.pdf or .doc) as well as your code (.ipynb or .py). Your code must run with python3.

Motivation
There is an overwhelming amount of news information available online. Some news headlines are known as clickbait – they aim to attract users to click on a link but the articles that they link to may not be of value or interest to the reader. In this HW we aim to answer the following question: C*an we automatically distinguish between clickbait and non-clickbait headlines, using a variety of linguistic cues?*

Data
You are provided with a corpus of clickbait and non-clickbait article headlines. The corpus is described in detail in this paper: https://arxiv.org/pdf/1610.09786.pdf
The data is stored in two files:
Non-clickbait: http://www.cs.columbia.edu/~sarahita/CL/non_clickbait_data.txt
Clickbait: http://www.cs.columbia.edu/~sarahita/CL/clickbait_data.txt
Note: this data is an extended version of the data in the paper – there are 16,000 headlines in each file.

Task
You are provided with starter code that loads the data, extracts a set of features, and then trains a Naïve Bayes classifier using those features and outputs the classifier accuracy. Your job is to extract additional feature sets using NLTK and report the classifier performance for each set of features. (Note: the starter code is optional – you can write your own)

Starter code
https://colab.research.google.com/drive/1fpmrn9eXJ0mbk02tmMafO5ESegnr0Nok?usp=sharing
Extract the following features:
1. Stop words: counts for each function word (from the NLTK stopwords list)
***this function is already implemented for you!**
2. Syntactic: counts for the following 10 common POS tags -- ['NN', 'NNP', 'DT', 'IN', 'JJ', 'NNS','CC','PRP','VB','VBG']
3. Lexical: counts for 30 most common unigrams in entire corpus (remove stopwords and punctuation for unigram count)

4. Punctuation: Counts for each punctuation mark in string.punctuation
5. Complexity:
   - average number of characters per word
   - #unique words/#total words
   - number of words
   - Count of “long” words - words with >= 6 letters
6. Your own proposed feature set: Think about what other features may be useful for clickbait identification and implement them. You can get ideas from this paper: https://arxiv.org/pdf/1610.09786.pdf

You are encouraged to play around with preprocessing and normalization that might help performance. You can also try other classification models in addition to Naive Bayes.

Record your results and answer the questions below.

1. Results (5pts)
Report the accuracy obtained using the following individual feature sets, as well as using all features combined.

Feature Set	**Accuracy**
Function words	0.874
Syntax	0.767
Lexical	0.71
Punctuation	0.501
Complexity	0.723
Your feature set (Interrogative words)	0.581
All features combined	0.911

2. Based on the features you extracted and the results, what observation(s) can you make about the nature of clickbait headlines? (2pts)

Out of the feature vectors extracted in this task, it seems that function words are the best predictor of clickbaiting. The presence of common English stop words allowed the classifier to correctly categorize the data ~87.4% of the time. This already remarkable performance can be attributed to the fact that traditionally clickbaits seem to include a significant number of stop words. According to Chakraborty et al. (2016, p. 3) the count of stop words in clickbaits is almost 2.5 times higher than in their non-clickbait counterparts. There is therefore little surprise that stop words are such a good predictor of whether a headline is a clickbait.

The second-best predictor of clickbaiting appears to be syntactic composition, expressed here as a vector of frequencies of 10 common POS tags. Based on this feature set alone, the classifier was able to correctly label the data in 76.7% of cases. While this level of performance is not disappointing, it seems that the feature set could be improved. Vectorizing the counts of common POS tags is certainly a good starting point for syntactic analysis, but it glances over the whole dimension of syntactic dependencies, which may provide useful modeling cues. Here I wonder if encoding syntactic dependencies (e.g. noun phrases, verb phrases, etc.) may help improve the classifier. In addition to computing POS tag vectors, it may be worthwhile to compute the distribution of selected phrase types (which themselves may be isolated with regular expressions) and employ those distribution values as feature vectors. Chakraborty et al. (2016) seem to acknowledge that classification may benefit from a more elaborate syntactic feature set by implementing syntactic N-grams (N-node paths generated by depth-first reversal of the syntactic tree of a headline). They also examine the length of syntactic dependencies, which they observe is longer for clickbaits.

With a 72.3% success rate, the third-best predictor is complexity. This feature set is composed of four subsets: the average number of characters per word, the TTR score, the number of words per headline, and the count of long (at least 6 characters) words in each headline. While computing these metrics was relatively intuitive, the TTR score in particular is ripe for adjustment, as it seems to be particularly susceptible to normalization (see previous homework). To see if normalization would have any effect on TTR as a component of the complexity feature set, two complexity vectors were computed - one with a normalized and one with a raw TTR score. Surprisingly, the accuracy in either case was 72.3% indicating that either normalization doesn't seem to matter all that much, or that its effect was negligible when applied only to the TTR submetric, as opposed to all complexity submetrics.

To implement the lexical feature set, the 30 most common unigrams in the whole corpus (clickbaits and non-clickbaits) were isolated. Thus obtained list was then used to generate unigram count vectors for each headline. At 0.71% success rate the classifier did only slightly worse than for the previous features set. Here it seems that the performance can be improved by expanding the set to focus on more complex N-grams, as examined by Chakraborty et al. (2016). In particular, the N-gram uniqueness seems to be a good predictor, with clickbait headlines significantly more likely to follow the same repeated word patterns.

Of all the computed features, punctuation seems to be the poorest predictor of classification accuracy. At 50.1% success rate, it is marginally better than a coin toss. As such it likely diminishes the overall performance of the classifier, and should probably be excluded

from the features set. Here it would be interesting to see which feature sets aid and which ones harm the classifier, or in other words, what combination of distributional features generates highest accuracy. While it may be tempting to discard the punctuation metric, the way that it interacts with other metrics may still improve performance in a statistically significant way (even though the metric in isolation barely does better than chance).

3. Describe the additional feature set that you implemented and the motivation for choosing that set of features. Were these features useful for clickbait classification? (2pts)

Personally, when thinking of clickbaits the first thing that comes to my mind are questions (e.g. “How did she lose 50 pounds in just two months?” or “What was behind their success?”). I was therefore slightly surprised that the authors didn’t include questions as a metric in their paper. When developing feature sets they did touch on determiners, which includes a handful of interrogative words, however they did not look at the interrogative words per se. Nor did they consider question marks, which would be another plausible way to approach questions. Then again, seeing how poorly interpunction performed as a predictor, it is perhaps not surprising that question marks were excluded from analysis.

Considering that questions were absent from the study I thought it may be useful to employ them as a feature set. Out of a number of ways to compute an interrogative feature vector I found the following three most intuitive:

- Vectorize the counts of interrogative words in each headline.
- Check if the headline is a question, i.e. does it begin with one of the selected interrogative words.
- Check if the headline ends in a question mark.

In light of punctuation’s poor performance, the latter of these approaches was dismissed a priori. The first approach yielded the accuracy rate of 58.1%, which while statistically significant, is just a bit better than chance. With the accuracy rate of 51.7%, the feature set generated as a result of the second approach performed even worse (just barely better than interpunction). Due to its disappointing performance its score was not reported. Here it may be useful to mention that since we are checking if a headline is a question or not, we are computing a binary feature. This in turn makes it sensible to apply a different Naive Bayes classifier, more suitable for binary features. `BernoulliNB` seems like a good choice here as it assumes features are *Bernoulli-distributed*; i.e., that they are binary-valued. The submitted code reflects this adjustment.

Seeing how poorly both feature sets did in terms of correctly classifying the headlines, it is not unreasonable to suspect that their inclusion in the final features set would negatively impact the classifier’s performance. In fact, that is exactly what happened. Whereas the accuracy after combining all feature sets was 91.1%, that score went up to 91.2% once the interrogative word counts vector was excluded from the model. Since the assignment requires that all features be combined into a single vector, the question words feature set was included in the final calculation in spite of this detrimental effect.

4. What are some ideas you could try to further improve performance? (1pt)

A handful of ideas that may be useful in improving the classifier's performance were already mentioned in my answer to the second question. Some of these ideas may be further expanded. For instance, when it comes to complexity, the task employs long-word counts as one of the possible measures. While it is no doubt a valid metric, another way to approach lexical complexity may be to compute word-entropy values for each word and calculate the mean for each headline. Thus computed word-entropy means may be then used to expand the complexity metric for the dataset. As stated by Chakraborty et al. (2016), and reflected in the results reported here, clickbait headlines tend to contain more function words and stop words than their non-clickbait counterparts. Conversely, the latter will contain more function words, which tend to be longer, and presumably yield higher entropy values. Should that be the case, the mean entropy per clickbait headline should be significantly lower than that for non-clickbait headlines. A feature design around that difference may improve the classifier..

Another feature worth exploring may be to look at the verbs in each headline category, which at least in terms of non-clickbaits should be characteristic of the reporting genre. If we assume that non-clickbait headlines generally refer to reports, then they may also contain more past-tense tokens, seeing how event reports tend to be relayed in the past tense. This may be in contrast to non-clickbait headlines, where there may be fewer instances of past tense use. If the difference in past tense verb distribution between the two categories is significant enough, it may help improve the overall performance of the classifier.

They say that if you want to learn about people's failures, start reading a newspaper from the first page. If you are interested in their successes, start from the end. As anecdotal as it is, this saying may be making a larger point, namely that reporting tends to be negative in sentiment. If we accept that observation, what follows is that the headlines that are not clickbaits are more likely to contain negative language than clickbaits. Analyzing both categories for sentiment, and using the findings as a feature set may be another way to further improve the model's performance.

The above suggestions relate more to feature design and extraction than the employed classification models. The features computed for this task were bag-of-words vectors, which work well with Naive Bayes (NB) classifiers. The NB classifier employed in this exercise was `MultinomialNB`, which assumes that features are essentially counts of some phenomenon and as such, it is commonly used in text classification. Another NB classifier, `ComplementNB`, which is a variant of `MultinomialNB` is designed for heavily imbalanced data; i.e., data where some classes are much more frequent than others, may have yielded different results. It may also be useful to increase the cross-validation parameter from 10 to 100-fold, though I am not sure if such an adjustment would significantly improve the score.

What is left may be to forgo generative classifiers (such as Naive Bayes) altogether, and instead use their discriminative counterparts, such as logistic regression or neural nets. Doing so may require adjusting the features, but should ultimately produce best results as suggested by Chakraborty et al. (2016) and their use of SVMs.