CS221
Project Progress Report
Matthew Radovan (mradovan 06250254)
Christopher Cross (chrisglc 06281089)
Sasankh Munukutla (sasankh 06124893)

Labeling News Headline Topics through Unsupervised Learning

Task Definition

Our project continues to follow its description in our proposal pretty closely. We are continuing on our goal to implement text feature extraction and topic modeling that can extract a set of keywords. Those keywords should describe the topic they are drawn from accurately and distinctly from any other topic in the model.

Dataset

Our dataset is a list of strings where each string is a news article headline combined with its description snippet, obtained from the News API. We currently have a dataset consisting of over 7000 headlines related to politics, which grows as news is released daily. Referring to an "article", "article headline", etc. refers to this combination of headline + description.

Pipeline

Because we are analyzing both feature extractors and clustering models, we've created a "mix and match" pipeline in which different feature extractors can be connected to different models and then run through the same quantitative analysis script.

Partial Feature Extractors

In general, text feature extractors (featurizers) convert string data into numeric data for models to ingest. We establish a set of featurizers to work as different text preprocessors for the models. However, our featurizers are purposefully only partially complete - rather than being full featurizers, they simply extract pare down the original text, and the models fully featurize the resulting text. This is because our models don't necessarily expect integers as input (such as the LDA model). And, because keyword extraction from clusters is drawn from the model but requires knowing how strings map to integers, we chose to include the generation of that mapping in our models.

Simple featurizer

As an initial featurizer, we simply removed all punctuation and split the string by spaces to get the words without punctuation. This featurizer keeps nearly all of the article's original text and only removes punctuation to avoid non-unique words accidentally being considered unique. However, because it leaves all words in, it tends to cause our models to have much higher dimensionality, leading to higher time and a potential decrease in clustering ability.

spaCy Noun Tagging ("noun featurizer")

We used spaCy, a natural language parsing library for Python, to automatically identify all nouns and proper nouns and parse down the full text to only use those identified terms as input to the models. The motivation behind this was that nouns are always the subject of a sentence, so identifying all nouns should give an initial idea into the possible topic of an article.

Example: “UK Supreme Court hears government side in vital Brexit case LONDON (AP) — The British government was back at the country's Supreme Court on Wednesday, arguing that Prime Minister Boris Johnson's decision to suspend Parliament just weeks before the country is set to leave the European Union was neither improper…” becomes ["UK", "Supreme", "Court", "government", "side", "Brexit", "case", "LONDON", "AP", "government", "country", "Supreme", "Court", "Wednesday", "Prime", "Minister", "Boris", "Johnson", "decision", "Parliament", "week", "country", "European", "Union"]

Word Length

Similarly to the noun featurizer, we built a featurizer that removes all words below a certain length. This featurizer was motivated by the speed of the noun featurizer - which was by no means fast. Instead, by removing all words with length lower than the median, we created a featurizer that inherently keeps more unique, and likely more important words, without having to use another model to tag nouns in a word. In the case of our dataset, the median word length was 5, so we cleaned punctuation and removed all words with length less than 5.

Stopword and Punctuation Removal (“clean featurizer”)

Next, rather than only caring about nouns, we built a featurizer using the nltk package that removes punctuation and stopwords in the text. Unlike the noun featurizer, this featurizer includes all parts of speech and instead removes common English words, so that verbs may be included but not words like “a”, “an”, and “the”. This approach gives more freedom to the resulting text than the noun featurizer, which may allow models to capture the topic of an article more accurately.

Example: The previous example instead becomes ["uk", "supreme", "court", "hears", "government", "side", "vital", "brexit", "case", "london", "ap", "", "british", "government", "back", "countrys", "supreme", "court", "wednesday", "arguing", "prime", "minister", "boris", "johnsons", "decision", "suspend", "parliament", "weeks", "country", "set", "leave", "european", "union", "neither", "improper"]

N-gram (N = 2 at the moment)

The general n-gram featurizers were built on top of the “clean featurizer”. We wanted to generally create n-grams to allow words to in sequence to have an effect on the clustering, especially grouping words in the case of names (such as “new york”, where “new” could be interpreted as being its own thing wording…) or phrases (“voters supported”, where “voters” has a wide representation across topics). These groupings allow for an automatic collection of words into ideas rather than clustering on words in, essentially, a void, where their proximity doesn’t affect the clustering. This featurizer ended up not working, even with N = 2, because topic keywords are 2-grams. Because of this increased specificity of features, there’s no guarantee that a given article contains even a single topic, which leads to empty articles when removing all non-keywords in the article.

Models

As mentioned previously, a model takes a featurizer run on our article dataset and creates a set of clusters of articles. In addition to this topic clustering, we ensured that each model had a method to extract its keywords from its topics - inherent in some models, such as the latent dirichlet allocation, but not in others, such as K-means. As an initial step, we set the number of clusters in each model to 10, although this can easily be changed.

K-Means

The K-Means model is straightforward and as described in lecture. As of the time of writing, our feature dimensionality on the spaCy featurizer was around 9000 (varies based on the featurizer, and ambiguous because our dataset grows daily). It uses a simple count vectorizer to create a sparse feature vector where each word maps to the count of that word in the article title and outputs an integer determining which cluster a given input is assigned to.

One consideration for this model is that it may suffer from the curse of dimensionality, so although the model is not slow, we need to qualitatively as well as quantitatively analyze the model's output to see if it makes sense. Another is that the number of clusters must be manually assigned, although we may be able to guide this through qualitative analysis of a datapoint visualization (such as through PCA to display on a 2-d axis).

To determine the relevant topic keywords, we are currently just calculating the N most common words in each topic and setting those as keywords (N = 25 at the moment). However, because we found that certain terms like "trump" are extremely common across multiple topics, we've begun working on a second cluster keyword extractor where the terms are ranked by the formula $r = p_i * (p_i / sum(p_i))$: the rank is the proportion of the term in the topic multiplied by the topic-specific proportion divided by the sum of the proportions across all topics. Intuitively, this means that the more unique a term is to a single topic, the more important it is, as long as it does occur frequently in that topic.

Latent Dirichlet Allocation

The Latent Dirichlet Allocation model roughly parallels the one mentioned in the lecture. Instead of assigning a given article to a specific model based on sparse vector clustering - as in the case of the K-Means, the LDA instead represents each article as a mixture of topics. The LDA takes as input the words that are taken from a given article and tries to generate a distribution of topics from these words. The LDA then picks a given generated topic and generates a distribution of words associated with this given topic. From there, it then backtracks by using a given article's words and tries to find the topic distribution that best fits that article's distribution of words. Thus, the LDA would then output the most likely topic for a specific article given the keywords of an article. One major disadvantage of the LDA is its poor ability to generalize to short documents - in this case, tweets. Given that it's a statistical inference model that relies heavily on a large amount of data with dense features in order to fully explore the feature space and develop a fully comprehensive distribution of topics for each article, the LDA's accuracy becomes dubious when such articles are sparse.

Biterm Model

The Biterm model tries to solve many of the problems that the LDA encounters with regard to short documents. Essentially, the Biterm model uses a significantly less complex model in order to better deal with inherently sparse data points. Instead of inferring the mixture of topics over each of the documents, it models the mixture of topics over the entire word corpus. Additionally, instead of inferring a topic from a single word, as in the case of the LDA, the Biterm model infers topics from biterms (which essentially represent the co-occurrence patterns of a range of word pairs in the corpus). Similar to the general approach of N-grams, this method reduces the noise of analyzing individual words from the case of the LDA by incorporating more context. Thus, the

Biterm model is a simpler model that still captures the nuanced relationships between topics and words in a short sequence of text.

Analysis

Our proposal discussed comparing two metrics to analyze the success of the keyword extraction: the first metric being a classifier that predicts the cluster label given the full article headline, and the second metric being the same classifier only using the terms in the article headline that are in the full set of keywords across all topics. Building upon this, we created a final metric that is the difference between the second metric and the first to determine how well the keywords described the topic. More negative difference indicates descriptive keywords - that is, the more negative the final metric is, the better the performance of the keyword model.

These metrics have been implemented using a logistic regression classifier (rather than a Naive Bayes because words in the short article headline are not necessarily independent).

Initial Results

The input data (featurized data, and featurized data containing only model keywords) were shuffled randomly alongside their cluster assignments. Scores were then generated with cross validation over 5 folds and the final metric described in the analysis section was computed. Recall that a more negative number is generally better.

	K-Means (K=10, 25 keywords)	Same K-Means with proportion-based terms (times remain the same)	LDA (10 clusters, 25 keywords)
Simple (0:01)	-0.0071 (1:54)	-0.0150	Not run yet
No stopwords (0:07)	-0.0102 (1:05)	-0.0078	Not run yet
Noun tagging (2:12)	-0.0061 (0:40)	-0.0076	0.2259 (0:09)
Word length (0:01)	-0.0051 (0:36)	-0.0043	Not run yet

Note - results are defined as *final metric = first metric (accuracy of classifier that predicts the cluster label given the full article headline) - second metric (accuracy of same classifier only using the terms in the article headline that are in the full set of keywords across all topics)*. Model training times and featurizer runtimes are in parentheses in the following format: *mm:ss* run on a Dell XPS 9560, i7-7700HQ, on battery.

The LDA hasn't been run on other featurizers yet due to its poor performance on the first featurizer we used for it.