Project Milestone: arXiv vs. snarXiv

Tyler Blanton **Sam Kowash**

Abstract

We give a brief update on our project to develop a classifier that can accurately distinguish between real arXiv `hep-th` abstracts and fake abstracts generated from a context-free grammar by the program snarXiv[1]. Our progress consists of three major parts: obtaining large amounts of data in an efficient way, parsing the abstract text to prepare it for analysis, and implementing a simple classification algorithm – a naive Bayes classifier using a bag-of-words model. We chose this extremely simple classifier to serve as a proof of concept, though it already outperforms humans (successful classification rate is 75-80% vs. 59% for humans).

1 Progress

1.1 Obtaining abstracts

The arXiv exposes its search API through a well-documented HTTP interface, but this has limitations in the context of our project as it is configured to find no more than 30000 results, and return no more than 2000 of them at a time. We estimate that `hep-th` has $\sim$85000 articles, and we'd ideally use all of them, so we need a more robust data acquisition tool. Fortunately, arXiv also furnishes an Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) interface, which is better-suited to full-repository harvesting. We use the `pyoai` module[2] to retrieve the full metadata set from `hep-th`, extract the abstract text from the returned XML structure, and feed it to the parser, which normalizes it for analysis.

The snarXiv abstracts are even easier to obtain, since we can generate them at will. We use a modified grammar file to produce only abstracts rather than titles and authors, then call the compiled snarXiv generator repeatedly to produce a corpus, which can be fed to the parser. (The snarXiv generator could of course be modified to produce the desired format in the first place, but we prefer to use as much of the original tool as possible.)

1.2 Parsing abstract text

The raw response from an arXiv API call is a bunch of text consisting of full `hep-th` papers (which often contain TeX symbols like \$ and \) along with several tags specifying sections as title, author, abstract, etc. For simplicity, we are currently only focusing on the abstracts, which we pull out and organize into a list using Python's `feedparser` package. We also store snarXiv abstracts in this form, so the rest of the parsing process is the same for the arXiv and snarXiv data.

The next step is to split the abstract text into regular words that are easily identifiable between different abstracts. We split on all spaces and newlines, and for uniformity we make all words lowercase and strip out all punctuation. This method works for the most part, but it leaves traces of some TeX commands; for example, it sends `\mathbb{Z}` $\rightarrow$ `mathbbz`. In the future we may try a more sophisticated approach where we deal with TeX commands at the beginning before stripping all punctuation, but for now this issue is only a minor nuisance. In any case, the output of our parsing

[1] `http://davidsd.org/2010/03/the-snarxiv/`

[2] `https://github.com/infrae/pyoai`

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

procedure is two lists of lists (one arXiv, one snarXiv), where each inner list is comprised of formatted words from a single abstract.

1.3 Classifying abstracts

For our first attempt at creating a program that distinguishes between arXiv and snarXiv abstracts, we built a naive Bayes classifier. The theory behind this classifier is simple: assume that each abstract X has a probability $\mathbb{P}(Y|X)$ of being from source $Y \in \{\text{arXiv, snarXiv}\}$. If $\mathbb{P}(\text{arxiv}|X) > \mathbb{P}(\text{snarxiv}|X)$, then we should classify X as arXiv; otherwise, we should classify it as snarXiv. This is an intuitive classification condition – the only challenge is approximating $\mathbb{P}(Y|X)$ for any abstract X. We can simplify matters by noting that

$$\mathbb{P}(Y|X) = \frac{\mathbb{P}(Y)\mathbb{P}(X|Y)}{\mathbb{P}(X)}, \tag{1}$$

which means we should classify X as arXiv iff

$$\mathbb{P}(\text{arxiv}|X) > \mathbb{P}(\text{snarxiv}|X) \tag{2}$$

$$\Rightarrow \mathbb{P}(\text{arxiv})\mathbb{P}(X|\text{arxiv}) > \mathbb{P}(\text{snarxiv})\mathbb{P}(X|\text{snarxiv}). \tag{3}$$

We estimate each probability in (3) by training on a large number of arXiv and snarXiv abstracts. Since we are in control over how many arXiv vs. snarXiv abstracts will be in the train and test sets, $\mathbb{P}(Y)$ is not too interesting – we "approximate" it as the fraction of training abstracts which are from source Y, and its "accuracy" is given by how close it is to the corresponding fraction of test abstracts which are from Y.

The less trivial task is estimating $\mathbb{P}(X|Y)$ – the probability that source Y will produce a given abstract X. Perhaps the simplest estimation scheme is the bag-of-words model, in which each word $x_i \in X$ is assumed to be independent of all other words in the abstract. This model is obviously flawed since certain word sequences are much more likely than others, but it can still be useful if some words are more likely in one source than another, so we adopt it to compute our initial estimate. In this model, we define our training estimate $\widehat{\mathbb{P}}(X|Y) \approx \mathbb{P}(X|Y)$ as

$$\widehat{\mathbb{P}}(X|Y) \equiv \prod_i \widehat{\mathbb{P}}(x_i|Y), \tag{4}$$

$$\text{where} \quad \widehat{\mathbb{P}}(x_i|Y) \equiv \frac{\#(x_i \in Y)_{\text{train}}}{\sum_j \#(x_j \in Y)_{\text{train}}} \tag{5}$$

is the frequency with which word x_i appears in source Y inside the training set; $\#(x_i \in Y)$ is the number of times word x_i appears in source Y in the training set, and thus $\sum_j \#(x_j \in Y)$ is the total number of words in Y inside the training set.

Testing this method on new abstracts after training on 800 arXiv and 800 snarXiv abstracts, we find that the algorithm correctly classifies an abstract as arXiv or snarXiv roughly 85% of the time. This accuracy is not fantastic (which is to be expected given the simplicity of the model), but it is higher than we expected, and is already well higher than the average human classification accuracy of 57%.[3] Looking a little more closely at the results, we find that the only misclassifications made are arXiv abstracts that the classifier mistakes as snarXiv – when a snarXiv abstract is tested, it is correctly identified as snarXiv 100% of the time. The explanation for this finding is relatively simple: snarXiv only has about 1000 unique words that it pulls from when generating fake abstracts, so each of these words occurs with a significant probability. Any reasonably large training set exposes this, and thus the classifier has no problem identifying snarXiv abstracts correctly. On the other hand, 800 arXiv abstracts contain roughly 6000 unique words (making for lower individual word probabilities), and thus the classifier can struggle when an arXiv abstract contains several snarXiv words.

2 Plans

2.1 Feature development

Currently, our feature model represents a given abstract as a dictionary of occurrence counts for each word in the abstract and each word known from the training corpus. This is called a bag-of-words

[3] According to `http://snarxiv.org/vs-arxiv/highscores/`

model and is completely insensitive to word order, meaning that it is impossible for a classifier to take into account word co-occurrence rates or sentence position, which obviously contain most of the information in natural language. (That our simple classification scheme achieves such reasonable performance given an extremely reductive model is surprising, and might suggest that the problem as posed is too easy; more thoughts on that below.)

The next step from a context-insensitive model is, obviously, the incorporation of context: instead of looking at how many times a given word appears in an abstract, we can look at the number of times it appears adjacent to each other word, or the number of times it appears in a triad with each pair of other words. This leads generically to the class of n-gram models, which incorporate length-n sequences of words as elementary objects of analysis.

Concretely, the Brown model is a common framework in which language production is treated as a Markov process with hidden states and transition probabilities, and frames the learning task as estimating these properties from n-gram occurrence data in the corpus. This task is complicated, however, by the rapidly ballooning amount of data that is produced if we must account for the occurrence of *every* possible n-gram, even considering relatively small maximum correlation lengths. This leads naturally to the field of word embeddings, which aim to map this painfully high-dimensional space into a lower-dimensional feature space in a way that preserves useful information about relationships between words from the n-gram structure. Stratos et al. [2015] and Stratos et al. [2014] discuss efficient methods for generating embeddings from corpora, which will be our next implementation step in feature design.

2.2 Possible extension

Our results thus far suggest that the language produced by snarXiv may just be too artificial and constrained to put up much of a fight in the classification task. The naive Bayes classifier never mistakes a snarXiv paper for real, although it still accidentally flags a decent chunk of real papers as fake. We will continue this work with more sophisticated feature generation and classifiers to explore different methods and try to reduce the remaining error, but it is worth looking in other directions as well.

In particular, if snarXiv cannot produce abstracts that our convincing to our classifier, what can? The field of generative adversarial networks has become prominent recently, as reviewed in Creswell et al. [2018], and we are considering extending our project in this direction. The aim would be to produce, effectively, a next-generation snarXiv that achieves a meaningful error rate against its discriminating counterpart. Several sources note that GANs are not naturally suited to contexts with discrete data streams, so this may prove unworkable, but is one direction to pursue if our original plan leads to shallow results.

References

Charu C. Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. In *Mining Text Data*. Springer, Boston, Massachusetts, 2012. URL `https://doi.org/10.1007/978-1-4614-3223-4_6`.

William Cavnar and John Trenkle. N-gram-based text categorization. 05 2001.

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1): 53–65, 2018.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks, May 2015. URL `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105. Madison, Wisconsin, 1998.

Karl Stratos, Do-kyum Kim, Michael Collins, and Daniel J Hsu. A spectral algorithm for learning class-based n-gram models of natural language. In *UAI*, pages 762–771. Citeseer, 2014.

Karl Stratos, Michael Collins, and Daniel Hsu. Model-based word embeddings from decompositions of count matrices. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1282–1291, 2015.

Shrawan Kumar Trivedi. A study of machine learning classifiers for spam detection. In *Computational and Business Intelligence (ISCBI), 2016 4th International Symposium on*, pages 176–180. IEEE, 2016.