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 $\sim$85% vs. 57% for humans).

1 Introduction

The arXiv is a free online repository maintained by Cornell for preprints of scientific papers (many of which go on to be published in journals) from fields including mathematics, physics, and computer science. Submissions are moderated via an endorsement system to ensure that only legitimate papers are posted, making it a valuable resource for scientists in many disciplines. In many areas of theoretical physics, almost all new papers are posted to the arXiv before publication.

The snarXiv is a computer program created by physicist David Simmons-Duffin that generates titles and abstracts in the style of arXiv's high-energy theory section (`hep-th`) using a context-free grammar. Simmons-Duffin created a browser game, "arXiv vs. snarXiv,"[2] in which readers are presented with two title/abstract pairs—one from `hep-th`, the other generated with snarXiv — and are asked to choose which one is real. After 750,000 guesses, the success rate was only 59%; i.e., people mistakenly identified a randomly-generated snarXiv title/abstract as real 41% of the time. This turns out to be a task that humans are remarkably (and amusingly) bad at, so we here investigate whether machines are any better.

In the first part of this report, we discuss the principles of our text representation (§2.1) and classification (§2.2) schemes. In the second, we present the results of parameter tuning and final classification accuracies. Details of implementation are collected in Appendix A.

2 Theory

Each of the abstract classification algorithms we use consists of three main parts: a parser, a language model, and a classifier. Here we discuss the theory behind our language models and classifiers; details about our parsing choices can be found in Appendix A.1.

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

[2] Now sadly defunct, it appears, but some results are summarized at `http://davidsd.org/2010/09/the-arxiv-according-to-arxiv-vs-snarxiv/`

2.1 Language models

2.1.1 n-grams

Assume that each abstract X has a probability $\mathbb{P}(X|Y)$ of being from source $Y \in \{-1, 1\}$, where $Y = 1$ represents the label "snarXiv" and $Y = -1$ represents the label "arXiv." Suppose $X = (w_1, w_2, \ldots, w_N)$ is an abstract containing N words $\{w_i\}_{i=1}^N$, and let w_i^{i+k} denote the word sequence $(w_i, w_{i+1}, \ldots, w_{i+k})$. The n-grams language model approximates $\mathbb{P}(X|Y)$ as

$$\mathbb{P}(X|Y) = \mathbb{P}(w_1^N|Y) = \prod_{i=1}^{N} \mathbb{P}(w_i|w_1^{i-1}, Y) \tag{1}$$

$$\approx \prod_{i=1}^{N} \mathbb{P}(w_i|w_{i-n+1}^{i-1}, Y) \quad (n\text{-grams approx.}) \tag{2}$$

In other words, in the n-grams model we assume that the probability of a given word w occurring in some abstract only depends on what the previous $n-1$ words in the abstract are, where n is some (typically small) positive integer; the $n-1$ words on which w depends are called the "context" of w.

In the simplest case $n = 1$, each word occurrence is assumed to be completely independent of any context, which is why the 1-gram language model is often referred to as the "bag-of-words" (BoW) model. Given some training set of abstracts from source Y, we define our BoW training estimate $\widehat{\mathbb{P}}(X|Y)_{\text{BoW}}$ as

$$\widehat{\mathbb{P}}(X|Y)_{\text{BoW}} = \widehat{\mathbb{P}}(w_1^N|Y)_{\text{BoW}} \equiv \prod_{i=1}^{N} \widehat{\mathbb{P}}(w_i|Y), \tag{3}$$

$$\text{where} \quad \widehat{\mathbb{P}}(w_i|Y) \equiv \frac{C_Y(w_i)+1}{\sum_w [C_Y(w)+1]} = \frac{C_Y(w_i)+1}{\sum_{w\in\mathcal{V}} C_Y(w)+V}, \tag{4}$$

$C_Y(w_i)$ is the number of times that w_i appears, $\mathcal{V}$ is a vocabulary of size $V = |\mathcal{V}|$ containing all of the words that we wish to learn about, and $\sum_{w\in\mathcal{V}} C_Y(w)$ is the total number of words in the training set. The extra $+1$ on each word count $C_Y(w)$ comes from Laplace smoothing, which is a precautionary measure taken to prevent zero probabilities from occurring in cases where we need to classify an abstract that contains a word that did not appear in the training set. Other technical details regarding bag-of-words and n-grams implementation can be found in Appendix A.

The generalization to $n > 1$ is straightforward: we define our estimate $\widehat{\mathbb{P}}(X|Y)_{n\text{-gram}}$ for a given training set from source Y to be

$$\widehat{\mathbb{P}}(X|Y)_{n\text{-gram}} = \widehat{\mathbb{P}}(w_1^N|Y)_{n\text{-gram}} \equiv \prod_{i=1}^{N} \widehat{\mathbb{P}}(w_i|w_{i-n+1}^{i-1}, Y), \tag{5}$$

$$\text{where} \quad \widehat{\mathbb{P}}(w_i|w_{i-n+1}^{i-1}, Y) \equiv \frac{C_Y(w_{i-n+1}^{i})+1}{C_Y(w_{i-n+1}^{i-1})+V}, \tag{6}$$

$C_Y(w_{i-n+1}^i)$ is the number of times that the n-gram w_{i-n+1}^i occurs, and $C_Y(w_{i-n+1}^{i-1})$ is the number of times that the $(n-1)$-gram w_{i-n+1}^{i-1} appears.

Note that since a typical arXiv/snarXiv abstract contains roughly 100 words, so the abstract probability $\mathbb{P}(X|Y) = \prod_{i=1}^N \mathbb{P}(w_i|w_1^{i-1}, Y)$ is a product of $N \approx 100$ different word probabilities, each of which is typically much less than one. As a result, the size of $\mathbb{P}(X|Y)$ is heavily dependent on the abstract length and is usually extremely small, causing our estimates to hit numerical underflow in some cases. To combat the issue of length dependence, we prefer to work with the perplexity

$$\text{PP}(X|Y) \equiv \sqrt[N]{\frac{1}{\mathbb{P}(X|Y)}} = \left[\prod_{i=1}^{N} \mathbb{P}(w_i|w_1^{i-1}, Y)\right]^{-1/N}, \tag{7}$$

which normalizes the abstract probability by taking the geometric mean of the individual word probabilities (and inverting). To prevent numerical underflow from rendering our algorithms useless, we always work with logs of probabilities in our code, which turns products into sums and keeps the numbers reasonable. Note that the log of the perplexity is a simple arithmetic mean:

$$\log \mathrm{PP}(X|Y) = -\frac{1}{N}\log \mathbb{P}(X|Y) = -\frac{1}{N}\sum_{i=1}^{N} \log \mathbb{P}(w_i|w_1^{i-1}, Y). \tag{8}$$

In practice, we use the logs of the classification conditions given below in Section 2.2 to classify our test abstracts as arXiv or snarXiv; we only exponentiate back to perplexities for the purposes of figures.

2.1.2 tf–idf

The family of n-gram models can be used to produce term probabilities for naive Bayes or likelihood-ratio testing, but they also induce perfectly good $\frac{V!}{(V-n)!}$-dimensional vector representations of documents, where each entry records the occurrence of a given n-gram in the document. These vectors are large, but fairly sparse, and make rudimentary features for regression or clustering algorithms.

However, the *semantic* model implied by raw count vectors is extremely crude. For example, ubiquitous words like "the", "and", and "of" are assigned great weight, and pull most document vectors in a handful of directions, distorting the representation space in a way that can make clusters harder to separate. This motivates a weighting scheme that recognizes words more unique to particular documents or classes; in particular, we developed feature vectors using the term-frequency–inverse-document-frequency (tf–idf) scheme as described in Jurafsky and Martin [2014, ch. 6].

Each entry d_i in a document vector d corresponds to a term w_i in the vocabulary and receives a weight

$$d_i = \mathrm{tf}_{i,d} \cdot \mathrm{idf}_i. \tag{9}$$

We define

$$\mathrm{tf}_{i,d} \equiv \begin{cases} 1 + \log_{10} C_{i,d} & : \quad C_{i,d} > 0 \\ 0 & : \quad \text{else} \end{cases}, \tag{10}$$

where $C_{i,d}$ is the number of occurrences of w_i in the document d, and

$$\mathrm{idf}_i \equiv \log_{10} \frac{N+1}{\mathrm{df}_i + 1}, \tag{11}$$

where N is the number of documents in the training corpus, and df_i is the number of documents containing w_i. [3]

The principle of tf–idf is to weight vector components according to their discriminating power: our representation should emphasize the most unique parts of documents. Accordingly, the inverse document frequency metric completely ignores a word that occurs in every document, but gives great weight to a word occurring only once in the corpus. Further, our term-frequency metric grows sublinearly, since a word occurring ten times is not necessarily ten times as important as one occurring singly.

2.2 Classifiers

2.2.1 Naive Bayes classifier

Our first attempt at creating a program that distinguishes between arXiv and snarXiv abstracts utilized a naive Bayes (NB) classifier. The theory behind this classifier is simple: if $\mathbb{P}(Y = 1|X) > \mathbb{P}(Y = -1|X)$, then we classify X as snarXiv; otherwise, we classify it as arXiv. We can use Bayes' theorem $\mathbb{P}(Y|X) = \frac{\mathbb{P}(Y)\mathbb{P}(X|Y)}{\mathbb{P}(X)}$ to rewrite this classification condition as

$$\widehat{Y}_{NB} \equiv \arg \max_{Y \in \{-1,1\}} \mathbb{P}(Y)\mathbb{P}(X|Y). \tag{12}$$

[3]Our definition of idf_i varies from that given in the reference by smoothing terms which deal with the possibility of developing features from a corpus smaller than the one that produced the vocabulary and prevent infinite logs.

We estimate each probability in (12) by training on a large number of arXiv and snarXiv abstracts. We define our estimate[4] for $\mathbb{P}(Y)$ as the number of abstracts in training set Y over the total number of abstracts in both the arXiv and snarXiv training sets. The conditional probabilities $\mathbb{P}(X|Y)$ are more complicated, and are approximated using a language model from Section 2.1.

2.2.2 Likelihood-ratio test

A slightly more sophisticated classification rule is the likelihood-ratio (LR) test. For a given abstract X, define the likelihood ratio $\Lambda(X) \equiv \frac{\mathbb{P}(X|Y=1)}{\mathbb{P}(X|Y=-1)}$. While we certainly want our classifier to correctly label all snarXiv abstracts as fakes, we also want to minimize the probability that an arXiv abstract gets incorrectly classified as fake. The Neyman-Pearson lemma states that the optimal classifier

$$\delta^*_{LR} \equiv \arg\max_{\delta} \mathbb{P}(\delta(X) = 1|Y = 1), \quad \text{subject to} \quad \mathbb{P}(\delta(X) = 1|Y = -1) \le \alpha \tag{13}$$

for any fixed false-positive tolerance α takes the form

$$\mathbb{P}\left(\delta^*_{LR}(X) = 1\right) = \begin{cases} 1, & \Lambda(X) > \eta \\ \gamma, & \Lambda(X) = \eta \\ 0, & \Lambda(X) < \eta, \end{cases} \tag{14}$$

where η and γ can be treated as hyperparameters to be tuned in cross-validation. In practice, γ is almost completely irrelevant since for any η, the odds that a language model like n-grams would yield an estimate for $\Lambda(X)$ that is *exactly* equal to η are essentially zero.[5] We arbitrarily use $\gamma = 0.5$, meaning the only hyperparameter we tune is η. Because of the issues of small $\mathbb{P}(X|Y)$ probabilities discussed in Section 2.1.1, it is more practical to instead compare the perplexities $\text{PP}(X|Y) = \mathbb{P}(X|Y)^{-1/|X|}$. The relevant parameter in the LR-test is thus $\eta^{1/|X|}$. We would like our hyperparameter to not depend on the specifics of the input though, so we note that the average abstract length is about 120 words and define $\eta_{\text{PP}} \equiv \eta^{1/120}$ to be our new LR-hyperparameter.

2.2.3 Logistic regression

For weighted vector representations as produced by the tf–idf scheme, it is not obvious how to produce term or document likelihoods for use in a Bayes or LR classifier, so we must turn to other classification methods. Fortunately, there are a host of methods for binary classification of vector data, and it is convenient and familiar to choose regularized logistic regression. We adopt a model

$$\mathbb{P}(Y = y \mid x) = \frac{1}{1 + e^{-yw^T x}} \tag{15}$$

where $y \in \{-1, 1\}$ represents the class of a document (negative for arXiv, positive for snarXiv), x is a tf–idf document vector, and w and b are parameters to be learned. We accomplish this by minimizing the convex regularized objective

$$J(w) = \sum_{i=1}^{N} \log\left[1 + \exp\left(-y_i(x_i^T w + b)\right)\right] + \lambda \|w\|_2^2, \tag{16}$$

where λ is a hyperparameter. The optimum lacks a closed form, but it can be found easily in CVXPY even for the $V \sim 15000$-dimensional space created by our vocabulary. After training w and b, we classify a document according to the sign of the exponent $(x_i^T w + b)$, since it can be related to the log-ratio of probabilities to be in each class.

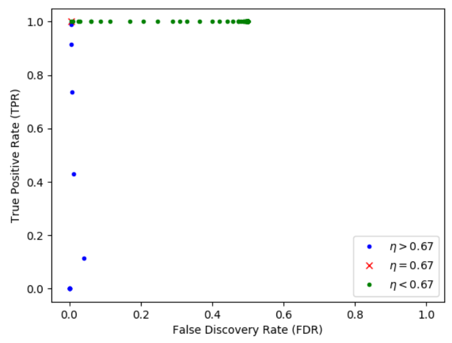
3 Results

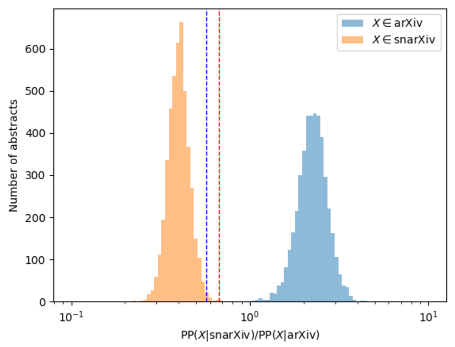
Testing the method on new abstracts after training on 800 arXiv and 800 snarXiv abstracts, we find that the algorithm correctly classifies an abstract as arXiv or snarXiv roughly 85% of the time. This

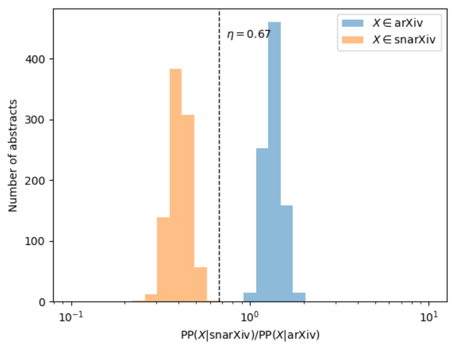
[4]The probabilities $\mathbb{P}(Y = \pm 1)$ are somewhat trivial since we generate the snarXiv abstracts ourselves and can control over how many arXiv vs. snarXiv abstracts will be in the train and test sets. Since the snarXiv was originally created to make abstracts that compete one-on-one with arXiv abstracts, we chose to set $\mathbb{P}(Y = \pm 1) = \frac{1}{2}$ for all tests reported in this paper.

[5]The borderline case of the naive Bayes classifier where $\mathbb{P}(Y = \pm 1|X) = \frac{1}{2}$ is irrelevant for the same reason.

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%.[6] 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.







A Technical details

A.1 Parsing abstract text

Our parser attempts to translate the raw abstract text (a string) to a list of formatted words that are easily identifiable between different abstracts. We first split the text on all spaces, newlines, and hyphens, and for uniformity we make all words lowercase. Next, we look for words that end in a period or question mark and insert the special word tokens `<e>` and `<s>` afterwards to mark the end of one sentence and the start of another. Lastly, we strip out all punctuation (except for the special tokens), prepend `<s>` to the list (marking the start of the first sentence), and delete the last word if the list ends `<e> <s>` (otherwise, we append `<e>`).

An example of parser input/output is

Input: `“I’m taking a CSE class on machine-learning. It’s a lot of work, but pretty interesting.”`

Output: `[<s>, im, taking, a, cse, class, on, machine, learning, <e>, <s>, its, a, lot, of, work, but, pretty, interesting, <e>],`

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

where each "word" in the output list is a string.

Our parsing scheme works well, but it leaves traces of some TeX commands from the raw abstract text; for example, it sends `\mathbb{Z}` $\to$ `mathbbz`. This would be fine if all arXiv and snarXiv abstracts have TeX commands written the same way, though it is possible in some cases for abstracts to write the same command in different ways. In practice, we find that these instances are rare enough for the concern to be negligible.

A.2 Vocabulary

We defined our vocabulary $\mathcal{V}$ as the collection of $V = 15,315$ unique words that appeared at least twice in a training corpus of 12,000 randomly chosen parsed arXiv abstracts and 12,000 (randomly generated) parsed snarXiv abstracts. We decided not to include the 13,960 words that only appeared once since they seem to be relatively uncommon (often they are the residuals of TeX commands), and including them would nearly double our vocabulary size. This pruning can be thought of as a feature-processing step.

A.3 n-grams

It is fairly common for a word to appear in the test set that is not in the vocabulary. When this happens, we change the word to a special token `<UNK>`. We then treat `<UNK>` as we would any other word when counting n-gram occurrences $C_Y(\cdot)$.

We prepend copies of the start sentence token `<s>` to each parsed abstract to ensure that the first n-gram consists of $n-1$ copies of `<s>` and one "real" word. Similarly, we append copies of `<e>` to the end of each parsed abstract.

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`.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

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

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

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.

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.