Detecting Spam in Web Corpora

Vít Baisa, Vít Suchomel

Natural Language Processing Centre
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
{xbaisa, xsuchom2}@fi.muni.cz

Abstract. To increase the search result rank of a website, many fake websites full of generated or semigenerated texts have been made in last years. Since we do not want this garbage in our text corpora, this is a becoming problem. This paper describes generated texts observed in the recently crawled web corpora and proposes a new way to detect such unwanted contents. The main idea of the presented approach is based on comparing frequencies of n-grams of words from the potentially forged texts with n-grams of words from a trusted corpus. As a source of spam text, fake webpages concerning loans from an English web corpus as an example of data aimed to fool search engines were used. The results show this approach is able to detect properly certain kind of forged texts with accuracy reaching almost 70 %.

Key words: web corpora, spam detection

1 Introduction

Web spamming has become a well know problem on the Internet. Spammed web pages contain hyperlinks, nonsense or very low quality texts in order to skew search engine results. The aim is to bring Internet users' attention to these in fact irrelevant pages. Seen through the eyes of an Internet browsing person, web spamming results in unwanted or unrelated content.

Another problem caused by web spam is distortion of frequency of words in collections of texts gathered from the Internet. This research is aimed at proposing a new method for detecting such unwanted spam contents. Comparing recently created web corpora for English, we observe more spammed data in a more recent corpus.

For example, word *viagra* is approximately hundred times more frequent in a web corpus from 2012 than in its predecessor from 2008. EnTenTen12, the focus corpus, was gathered from the web in May 2012 and cleaned using boilerplate removal and 7-gram based deduplication algorithms [1]. Comparable cleaning techniques were applied to the older corpus, therefore the cleaning procedure was unable to deal with increased presence of the word *viagra*.

Tables 1 and 2 show lemmas of words which are significantly more frequent in the most recent web corpus than in older web corpora. Looking at the top

Aleš Horák, Pavel Rychlý (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2012, pp. 69–76, 2012.

	lemma	2012	2008	RFR
1	**loan**	360.1	51.7	6.97
2	online	462.4	119.2	3.88
3	your	4194.4	1660.2	2.53
4	**insurance**	263.1	56.8	4.63
5	**credit**	321.7	119.9	2.68
6	buy	421.3	175.7	2.40
7	**mortgage**	132.4	22.9	5.78
8	product	502.6	219.6	2.29
9	brand	164.3	41.8	3.93
10	website	261.9	94.5	2.77
...				
21	**debt**	150.9	48.5	3.11

Table 1. Keywords from focus corpus enTenTen12 (2012), reference corpus enTenTen08 (2008).

	lemma	2012	2008	RFR
1	**loan**	360.1	65.1	5.53
7	**credit**	321.7	106.3	3.03
20	**mortgage**	132.4	32.3	4.10
26	**debt**	150.9	46.7	3.23
112	**insurance**	263.1	157.1	1.67

Table 2. Keywords from focus corpus enTenTen12 (2012), reference corpus ukWaC (1996).

items, there is a suspiciously high amount of words from domain of money: *loan, insurance, credit, mortgage, debt, etc.* RFR stands for relative frequency ratio between columns 2008 and 2012. The keywords extraction [2] function of SketchEngine was used. There are frequencies per million (FPM) in appropriate corpora in columns 2012 and 2008. The keywords are sorted by *keyness rank* which is order of a lemma in the comparison according to keyness score

$$\frac{\text{FPM in focus corpus} + 100}{\text{FPM in reference corpus} + 100}.$$

Authors of [3] also claim the amount of web spam increased dramatically and define several types of spam techniques. This work is interested in the following types:

- dumping of a large number of unrelated terms,
- weaving of spam terms into copied contents,
- phrase stitching (gluing together sentences or phrases from different sources).

Cleaning procedures applied to enTenTen12 were not designed to remove that type of spam, therefore the aim of this work is to detect such spam texts,

especially phrase stitching. Other spam categories can be dealt with boilerplate removal tools (e.g. pages containing only hyperlinks) or deduplication tools (e.g. pages copying whole paragraphs from other sources).

Another big issue is a rapid growth of content farms (see the second item in Table 3) – low quality articles promoting goods, services or webpages also made just to increase a page rank. Although being sparsely informative and much repetitive, such text is syntactically and sematically correct. That is why we do not intend to remove it from text corpora and this work does not aim to detect such kind of web content.

2 Sources of spam data

For the purpose of evaluating our method, we selected two sources of generated data: web documents about loans and fake scientific articles. The data was obtained in November 2012 from the web. Boilerplate removal [1] was applied.

2.1 Recent web corpus

Since there is a whole group of *loan* (and generally money) related words among the top keywords in the comparison in Tables 1 and 2, we chose to study documents containing word *loan* in enTenTen12. 200 documents were randomly chosen and the source web pages displayed in a browser for examination. However, only 92 pages were successfuly downloaded – this work was done 6 months after crawling the pages, many of them were not found or contained a different text.

Table 3 shows classification of web pages in the collection. We classified 44.5 % of documents not suitable for a text corpus as a spam. We selected 406 paragraphs from random documents and evaluated them once again, since some paragraphs in spam documents were not spam and vice versa. Finally, 199 (49 %) spam and 207 (51 %) not spam paragraphs were used in further experiments.

text category	**class**	**% doc**
nice text	OK	37.0 %
low quality text (possibly a content farm)	OK	18.5 %
fluency slightly broken (sentence or paragraph stitching)	spam	9.8 %
fluency broken (sentence or phrase stitching)	spam	13.0 %
not fluent (triplets of words stitching)	spam	14.1 %
nice text, unrelated words (spam terms weaving)	spam	7.6 %

Table 3. Classification of texts from the collection of 92 web documents containing word *loan*. Texts not containing fluent paragraphs were marked as spam.

3 All n-grams approach to detect generated texts

N-grams are the most common resource for statistical language modeling. Language models are used in many areas, mainly in speech analysis and in machine translation. In the latter, a language model is responsible for a fluent translation.

It was shown that language models assign more probability to a fluent text than to a random sequence of words. In well-known evaluation method for quality of machine translation BLEU [4], n-grams (where n is from 1 to 4) are used for measuring fluency of a candidate sentence and this method correlates reasonably well with human evaluations.

Usually, n-grams with n up to 4 are used. It means that higher-order n-grams are not taken into account. There are several reasons why not to use higher-order n-grams: slower performance, higher hardware requirements and mainly sparse data.

In our method, we suppose that a fluent text can be achieved by using a language model as in machine translation. When generating non-sense text, a language model relies on n-grams up to some n. Let us suppose now $n = 2$. Since the model has information about bigram counts (and probabilities) but knows nothing about trigrams, we might recognize this fact simply by checking frequencies of trigrams from the generated text taken from a reference corpus. Generally, if a model used n-grams, we could always check n+1-grams.

In other words, we suppose that n-grams of an order higher than an order used in a model will have much lower frequencies since the model simply does not know about them. Using a trigram model, generated quadrigrams will be more or less random.

3.1 Frequencies of all n-grams in a corpus

Our method does not use standard probabilities as in usual language models. We use simple frequencies of all n-grams of all orders. For a given sentence, we check all unigrams, bigrams, trigrams, ... n-grams where n is sentence length.

To be able to do that, we need a reference corpora (we used British National Corpus, BNC) and a procedure which quickly gets counts of all possible n-grams. There are $O(m^2)$ of all possible n-grams in corpus with m word positions. We used algorithm described in [5], which produces these counts in $O(n)$ time.

The algorithm uses *suffix array* [6], *longest common prefix array* [7] and fact that majority of all n-grams are unique in a corpus. In Table 4 you can see a part of suffix array built from the BNC.

Since n-grams in the suffix array are sorted, we can observe, that there are at least six bigrams 'distinguish at', exactly three trigrams 'distinguish at least' and only one quadrigram 'distinguish at least between', 'distinguish at least four' etc. All n-grams starting with 'distinguish at least four' has frequency 1 and the algorithm exploits this observation.

distinguish	at	all	between	the	personal	...
distinguish	at	least	between	the	meaning	...
distinguish	at	least	four	sources	in	...
distinguish	at	least	three	cases	:	...
distinguish	at	once	from	the	destruction	...
distinguish	at	the	outset	between	the	...

Table 4. Part of suffix array built form BNC, n-grams starting with *distinguish.*

3.2 Frequency-drop between n-gram orders

For the classification we needed a list of features to be able to use a machine learning method. For that purpose we used frequency-drop between various levels of n-grams in a sentence. The first level (word counts) is compared with the second level (bigram counts) and so on. Counts are summed up for a level and these sums are simply divided between levels. A frequency-drop is ratio between a sum of n-grams frequencies from a level a and a sum of n-gram frequencies from a level $a + 1$. We suppose that the more fluent and natural a text is the smaller frequency-drops should be between its appropriate n-gram levels and that a significantly bigger frequency-drop is located between levels which correspond to an order of a language model used for generating the text.

On Figure 1 you can see an explaining diagram. Numbers bellow the words are frequencies of various n-grams. Frequencies on the first line with numbers correspond to unigrams. The second line contains frequencies for bigrams etc. On the right side, there are three frequency-drop numbers which stand for ratios between sums of appropriate n-grams levels.

Since the text is generated and it is not fluent English text, frequency-drops on various levels are very low (less than 0.01).

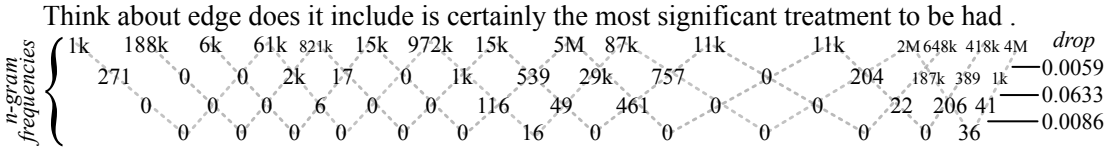

Fig. 1. Frequency-drop on 4 levels for apparently generated sentence.

Some low frequent words as e.g. proper names may introduce substantial frequency-drops between n-gram levels, but in one variant of our method we use average frequency-drops values, which should solve the problem especially for longer sentences.

On the contrary, some high frequent words may introduce the same from the other side, but this effect is weakened again by using an additional average value.

3.3 Classification

List of frequency-drops are vectors with values from 0 to 1. In our data maximum frequency-drop level with non-zero sum was 7 (i.e. there was an 8-gram with non-zero frequency). At first we used only frequency-drops from non-zero levels, but then added another value: average frequency-drop for all levels. In that case, accuracy was improved slightly. As another feature, paragraph length (in sentences) was added which also slightly improved the results.

For the classification we used both simple threshold tuning and Support Vector Machine (SVM) [8] for machine learning.

In the first experiment on development data, two thresholds were combined: frequency-drop between first two levels (0.015) and average frequency-drop on all levels (0.025).

For the SVM we used the mentioned n-tuples with two additional values – average frequency-drop and paragraph length. SVM automatically chose its kernel-function to maximalize accuracy of the trained model.

4 Results

For evaluation of these methods we used standard metrics: *precision, recall, accuracy* and *f-score*. In case of spam classification, it is usefull to express these metrics using terms *true positive* (*tp*), *true negative* (*tn*), *false positive* (*fp*) and *false negative* (*fn*). When a paragraph is annotated manually as 'spam' and classified by one of our methods as 'spam', then it is *true positive* match. The other matches are analogical. Standard metrics can be then expressed as follows.

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

$$f\text{-}score = 2 \times \frac{precision \times recall}{precision + recall}$$

In Table 5 you can see results for the two methods. BASE is baseline method: classification of all paragraphs as 'spam'. sDEV is the simple threshold method run on development data, sTEST is the same method run on test data. In next columns, the SVM method with various training vectors are listed. SVM used n-tuples with up to 7 frequency-drop levels. SVM^c used one more value for a number of sentences in a paragraph, SVM_a used one more value for average frequency-drop and the last SVM_a^c used the two additional values together.

We can see that the simple threshold method is only slightly better than the baseline. So is the first SVM method using vector of frequency-drops for n-gram levels. The best method is SVM^c which gives almost 70 % accuracy.

	BASE	**sDEV**	**sTEST**	**SVM**	**SVM**^{c}	**SVM**$_a$	**SVM**$_a^c$
precision	48.53	55.36	49.72	31.31	83.84	72.73	84.85
recall	100.0	97.03	90.91	50.00	63.36	64.29	62.22
f-score	65.35	70.50	64.29	38.51	72.17	68.25	71.79
accuracy	48.53	59.41	50.98	51.47	68.63	67.16	67.65

Table 5. Results for various evaluated methods.

5 Conclusions and future work

Processing of test data was done on vertical files. Development and test sets have about 17,000 tokens respectively.

We classified spam on paragraph level. It might be more appropriate to classify spam on sentence level since as was said, sometimes only a part of a paragraph is generated and the rest is taken over from another web page. Moreover the mentioned deduplication tool removes duplicates on paragraph level so it can not solve the problem with sentences stitched together in a paragraph.

The accuracy of these proposed methods can be decreased also by imperfect training data. It was annotated manually by non-native speaker of English which probably influenced quality of the annotation. For further research we would like to use data annotated by more annotators and especially data annotated by native speakers.

The process of getting frequency-drops on all possible n-gram levels is very fast. After suffix array, longest commonest array and all n-gram frequencies are prepared, we are able to process cca 2,000,000 tokens per minute on a computer with 8 processors and 100 GB RAM. It allows us to process very large billion corpora within hours or days. Standard language modeling methods (using probabilities) would be much slower and thus unusable for big data.

In the future we would like to use n-grams extracted from larger corpora, e.g. from mentioned ententen08. As a reference corpus we used BNC. We would also like to try the method on bigger training and testing data for specific domain of scientific texts. For that purpose we intend to use fake scientific articles generated by Scigen[1]. As a counterpart to these fake articles, random documents published in Computer Science section in arXiv[2] could be used. For that data set we would use a corpus containing scientific documents.

Spam and other garbage on the web is increasing problem nowadays and we should try our best to be able to deal with it and filter it out. Without it, methods for machine translation, speech analysis etc. would be badly affected by low quality data used for building n-gram language models.

[1] An Automatic Computer Science Paper Generator, `http://pdos.csail.mit.edu/scigen/`

[2] Open access to natural sciences e-prints, `http://arxiv.org/list/cs/recent`

6 Acknowledgement

This work has been partially supported by the Ministry of Education of CR within the LINDAT-Clarin project LM2010013 and by EC FP7 project ICT-248307.

References

1. Pomikálek, J.: Removing Boilerplate and Duplicate Content from Web Corpora. PhD thesis, Masaryk University, Brno (2011)
2. Kilgarriff, A.: Getting to know your corpus, Springer (2012) 3–15
3. Gyongyi, Z., Garcia-Molina, H.: Web spam taxonomy. (2005)
4. Papineni, K., Roukos, S., Ward, T., Zhu, W.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting on association for computational linguistics, Association for Computational Linguistics (2002) 311–318
5. Yamamoto, M., Church, K.: Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. Computational Linguistics **27**(1) (2001) 1–30
6. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. siam Journal on Computing **22**(5) (1993) 935–948
7. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Combinatorial Pattern Matching, Springer (2006) 181–192
8. Suykens, J., Vandewalle, J.: Least squares support vector machine classifiers. Neural processing letters **9**(3) (1999) 293–300