Sentiment Detection with Character n-Grams

Tino Hartmann, Sebastian Klenk, Andre Burkovski and Gunther Heidemann

***Abstract*—Automatic detection of the sentiment of a given text is a difficult but highly relevant task. Application areas range from financial news, where information about sentiments can be used to predict stock movements, to social media, where user recommendations can determine success or failure of a product.**

We have developed a methodology, based on character n-grams, to detect sentiments encoded in text. In the course of this paper we will present the founding idea and the algorithms as well as a usage scenario with an evaluation. We discuss the the obtained results in detail and a compare them with those of other popular sentiment detection methodologies.

I. Introduction

Sentiment detection is an important aspect of unstructured text analysis. Automatically determining the feelings that a text is expressing is becoming increasingly important as more and more content is generated. Especially for companies the knowledge about consumer sentiment is of high value. Social media and user generated content are more and more forming public opinion. A decade ago consumer decisions were mostly based on experiences of close friends and a selective list of publications. Today, social media gives access to experiences of several thousand consumers and public opinion is formed by a vast network of users contributing and sharing information. One aspect of this social opinion generation process is that the overall sentiment is not determined by a few individuals but by an aggregation of all the available sentiments. Therefore it is necessary to be able to automatically analyze user generated content.

In this paper we want to contribute to this reseach task by presenting a sentiment detection method based on character n-grams. Here, as opposed to word n-grams, one is capable – by a suitable choice of n – to detect interword dependencies without overboarding combinatorics. Word n-grams require an exact n-tuple word match, character n-grams require only n characters to match, which (*i*) is more likely (for small n) and (*ii*) allows to match text stems without any sophisticated language model.

Character n-grams are a rather popular and simple method in natural language processing and information retrieval [1], [2]. In the course of this paper we will present a method to compare character n-grams based on the cosine distance. There the so called "Length Delimited Dictionary Distance" (LD^3) forms a very simple but efficient way to measure the distance between documents. The originating idea thereof stems from the Normalized Compression Distance [3], [4].

In the course of this paper we will present character n-grams as a means to determine sentiments of texts. We will present the rationale behind it, the algorithm as well as some experiments that demonstrate its applicability. We will further analyze, whether character n-grams are suited for determining text sentiment. For this purpose, we try to classify the popular IMDb dataset [5], using n-grams as terms with a Naive Bayes classification, and compare the results with other existing methods.

Tino Hartmann Sebastian Klenk, Andre Burkovski and Gunther Heidemann are with the Intelligent Systems Department, University of Stuttgart, Stuttgart, 70569, Germany (phone: +49 (0)711 685 88-241; email: klenksn@vis.uni-stuttgart.de).

II. Related Work

Because of the inherent complexity of the task and the lack of a model that is generally agreed on, there is a vast variety of approaches trying to tackle the text sentiment problem. The most basic approach is to model text as a *bag of words*, neglecting all compositional structure. Every single word gets labeled with a polarity score, which represents the probability of the word being in a positive or a negative text. The polarity of the text is then defined as the sum of all word polarities. Polarity scores for terms can either be manually constructed [6] or inferred via machine learning techniques [7], [8]. Manually constructed reference sets have always the problem of coverage, which means that most of the domain specific words will not be enclosed in an universal reference set. Some work focuses on the construction of domain specific sets and the adaption of existing ones of other domains [9]. Such a domain specific set can be inferred with a number of techniques, for example seeds, i.e. words with known polarity like *"good"* and *"poor"* and a proximity measure between words, like mutual information [7] or the WordNet[10], [11].

One major problem is that most of the sentences of a document do not express any sentiment. They will only add noise to the classification process. Therefore, there has been the attempt to classify the objectivity on sentence level [12]. The polarity estimate is then based only on the sentences which were classified as *subjective* beforehand. It is possible to go even further and try to determine which topic a given sentiment addresses. Instead of assuming, that a text only consists of sentiments about a single topic, every document is modelled as a collection of sentiments about many topics. A review of a book may contain sentiment about the author, which can be a different from the sentiment about the book. For example, Mullen [11] tries to determine topic proximity via the open ontology tool [13].

All of these basic approaches have a good baseline performance, but there seems to be a certain barrier of accuracy that all of them can not overcome. Ironically, they perform only slightly better than simple machine learning approaches. It seems to be obvious that the reason for this lies in the neglection of words interdependencies, i.e. the structure and the context of the text. To model this structure, the compositional semantics, there is a variety of approaches.

A very basic approach to model sentence interdependencies is to look at negation only [14], more sophisticated is it to try to build semantic hierarchies via manually constructed rules [15] or to improve the classification based on words by simple linguistic rules [16]. The most promising approach seems to be a combination of machine learning techniques (like SVMs), pattern and sub pattern recognition and analysis of the grammatical structure of sentences [17]. Further information on the subject of sentiment detection can be found in the very extensive survey on sentiment detection, that has been done by Pang and Lee [12]. In this paper we are not concerned with more advances models. We are trying to detect sentiments with as little prior knowledge as possible.

III. The Movie Database

Sentiment detection, like any other pattern recognition and machine learning problem is highly depending on the quality of the data. We chose the IMDb movie review database as test scenario, because it is probably one of the most commonly used data sets for sentiment detection. The IMDb is a freely accessible library containing information to countless movies. Besides featured actors or information on the director the site also contains movie reviews which can be found at http://reviews.imdb.com/Reviews. There, one has access to over 41,000 movie reviews, written in plain English. Unfortunately, the data format varies and there is no common rating scale, which makes an automated use of this dataset difficult. However, a formatted dataset has been made available at http://www.cs.cornell.edu/people/pabo/movie-review-data/ which has grown very popular among sentiment detection researchers (used in [12], [17] for example). It consists of 1,000 both positive and negative reviews.

The IMDb dataset has proven to be especially difficult. One problem all algorithms have, that try to tackle sentiment detection with word counting, is that for example *"good"* and *"not good"* have opposite meanings. Algorithms based on word occurrence will match *"good"* both times which means that both phrases get a high positive weighting due to the occurrence of *"good"* which in the latter case is plain wrong. Das and Chen [14] tried to eliminate this problem by marking all words between a negating word and the next punctuation with a special tag, so that "good" and the word good in "not good" will count as different word. We will call the IMDb dataset, which is tagged with this rule IMDb-NOT.

IV. Text classification

When classifying text there are a large number of possible methods to choose from. Probably the most well known is the Naive Bayes classifier [18] with its simplistic approach. Besides that we will present two rather new approaches to text classification based on character n-grams.

A. Naive Bayes

The Naive Bayes classifier is a very common approach to statistical text classification. It is based on the – obviously naive – assumption that the occurrence of one term $t \in D$, given a document class C, is independent of the occurrence of any other term. Therefore, if we disregard the interdependency of term $t \in D$ with all other terms $t' \in D$ the conditional probability of the document D being a member of class C is simply:

$$P(C|D) = P(C) \prod_{t \in D} P(t|C) \tag{1}$$

The prior probability $P(C)$ of any document being in class C is estimated as follows:

$$P(C) = \frac{\#D_C}{N} \tag{2}$$

Where $\#D_C$ is the number of documents in the training set that are in class C and N is the number of documents in the training set. If we use a balanced dataset, $P(C)$ is identical for all classes. $P(t|C)$ is estimated as the relative frequency of term t in all documents belonging to class C.

$$P(t|C) = \frac{\#t_C}{\sum_{t' \in C} \#t'_C}$$

Here $\#t_C$ is the number of occurrences of term t in class C. To obtain a probability we are normalizing it with the sum of the occurrence of all terms in C. Although the assumption of positional independence is far from reality, Naive Bayes performs quite well for sentiment detection.

B. Length Delimited Dictionary Distance

In this section we will introduce what we are calling the Length Delimited Dictionary (LDD_k). It is based on the idea of compression based pattern recognition [4], where it is possible to determine dissimilarity of two objects by looking at the ratio of joint compression against individual compression. Let C be a compression algorithm and $C(s)$ be the length of the compressed string s. The normalized compression distance (NCD) is defined as follows:

$$NCD(s_1, s_2) = \frac{C(s_1, s_2) - min\{C(s_1), C(s_2)\}}{max\{C(s_1), C(s_2)\}} \tag{3}$$

Most discrete compression algorithms generate a dictionary $W(D)$ to compress a document D. This dictionary is simply a list of substrings (*words*) w, all of which have preferably high frequency in D. If the compression algorithm finds a word w of the dictionary in the string, it replaces this word with a shorter one. If there is no occurrence of w in D, w does not contribute to the compression of w. If there is no occurence of *any* word of the dictionary, the document D will not be compressed or might even get larger. If a dictionary can be used to highly compress a document D_1, but does not compress another document D_2, we can assume that D_1 and D_2 are very dissimilar.

The joint compression of two strings can be very effective, if a dictionary $W(D_1, D_2)$ can be found that compresses both strings effectively. We assume that in this case the dictionaries $W(D_1)$ and $W(D_2)$ are very similar and it is sufficient to compare the dictionaries to determine dissimilarity.

In order to have an intuitive and highly flexible dictionary that can be used to measure the distance of any type of data,

we use a very basic approach for the generation of W: the Length Delimited Dictionary (LDD). Formally speaking the LDD_k of a document D is the set of all substrings of length k (character k-grams).

The Length Delimited Dictionary Distance LD_k^3 of two documents D_1 and D_2 is the number of elements that are common to both dictionaries normalized by the number of unique elements in both dictionaries joint together:

$$LD_k^3(D_1, D_2) = 1 - \frac{|LDD_k(D_1) \cap LDD_k(D_2)|}{|LDD_k(D_1) \cup LDD_k(D_2)|} \quad (4)$$

Interesting to note here is that the LD^3 is identical with the Jaccard similarity coefficient [19] and as such related to the cosine distance for character n-grams.

For sentiment detection we create two dictionaries (consisting of k-grams) $LDD_k(D_0)$ and $LDD_k(D_1)$ where D_i is the class document represented by the concatenation of all documents of class $i \in \{0|1\}$. For each class $C \in \{0, 1\}$ we determined the class membership $C_k(D)$ of document D by calculating the dissimilarity between D and each of the class documents D_0 and D_1.

$$C_k(D) = \underset{i \in \{0,1\}}{\arg\min} \{LD_k^3(D_i, D)\} \quad (5)$$

C. Character n-grams with Naive Bayes

LD^3 determines dissimilarity in a black and white manner, either an n-gram exists or not. Naive Bayes on the other hand weights the existence or non-existence. Unfortunately it is too restrictive in such a way that only words or even worse word n-grams are used. We implemented Naive Bayes with character n-grams as a trade-off between the flexibility of the Length Delimiting Dictionary – depending on the length, LDD is capable of representing inter word dependencies – and the problem adaption of Naive Bayes which learns the relevance of strings. This way, as we will demonstrate later on, we are capable to increase the recognition performance beyond that of either one of them alone.

The algorithm is as follows: instead of calculating $P(C|D)$ with word occurrences within a document D, we define d_k to be a LDD_n dictionary element, i.e. a substring of length n. Thus the Naive Bayes formula will be rewritten to

$$P_n(C|D) = P(C) \prod_{d \in LDD_n(D)} P(d|C)$$

with

$$P(d|D) = \frac{\#d_C}{\sum_{d' \in C} \#t'_C}.$$

Here $\#d_C$ is the number of occurrences of dictionary element d in the dictionary consisting of all documents in class C. We will call this classifier $NB(LD_n)$ as opposed to $NB(n)$ for plain Naive Bayes.

V. Evaluation

We tested the $NB(n)$ classifier with 10-fold cross validation on the two datasets IMDb and IMDb-NOT. We compared the results to a Naive Bayes classifier using word n-grams as feature. We call the classifier that uses Naive Bayes with word n-grams $NB(n)$, so $NB(1)$ is a Naive Bayes approach operating on unigrams, $NB(2)$ on bigrams and so forth.

The results of our evaluation can be observed in Figures 1,2 and 3. Detailed information can be found in Table I. For the IMDb-NOT dataset details are presented in Table II. Here there is a slight increase in performance due to the prior information (inform of the encoded negation) stored in the data.

It turns out that the simple LD_n^3 classifier cannot outperform Naive Bayes, but $NB(LD_n)$ performs slightly better than $NB(n)$, i.e. character n-grams are better features than word n-grams. This is interesting, because character n-grams make less assumptions on the underlying data than regular n-grams. The document does not have to be tokenized into words, a simple substring routine is sufficient. As a result, character n-grams can be used on all kinds of data.

As a baseline for the evaluation of the $NB(LD_n)$ classifier we are referencing Pang and Lee. They are classifying the exact same dataset with a number of different classifiers [20]. There Support Vector Machines with unigram feature presence got the best result of 82.9% accuracy. It should be mentioned though that they evaluated the classifier with 3-fold cross-validation which generally yields worse results.

Better results were obtained by Matsumoto *et. al.* [17] with an accuracy of 88.3%. Their solution uses much more knowledge about the underlying data. They incorporate information on text and language, like the grammatical structure of sentences returned by a natural language parser which is not always available or even desirable.

We are also comparing the LD_k^3 distance measure with its origin, the Normalized Compression Distance. Therefore we are creating an intuitive classifier based on compression. The document D was classified as a positive review if the average normalized compression distance to all positive reviews was smaller than the average distance to all negative reviews of the training data. Ten-fold cross validation result obtained for the NCD classification is 63.5%.

VI. Discussion

Originating from the Normalized Compression Distance, the LD_k^3 distance measure does fairly well at the text sentiment classification task. Whereas an NCD classification with 10-fold cross validation reached only 63.5 percent, the $ld(17)$ classifier can achieve an accuracy of 80.4. This is not much compared to other classification solutions done by other authors, but it shows, that the LD_k^3 distance is a suitable and efficient substitution for the NCD distance when it comes to large documents.

A very interesting result is, that character n-grams are a better choice than word n-grams when used with Naive Bayes. This could mean that word n-grams are either too strict in counting evidence or, which is more probable, are requiring a larger training data set. A classification with tri-grams, given the size of the training set, is not optimal, because there simply are not enough tri-gram intersections

TABLE I
ACCURACY RESULTS FOR DIFFERENT CLASSIFIERS BASED ON THE IMDB DATASET.

C	ld1	ld2	ld3	ld4	ld5	ld6	ld7	ld8	ld9	ld10	ld11	ld12	ld13	ld14	ld15	ld16	ld17	ld18
μ	14.7	49.7	50.9	50.3	50.8	51.3	52.2	54.0	57.3	62.3	68.0	73.0	76.5	78.3	79.4	80.3	80.4	79.8
σ	13.20	1.59	0.19	0.02	0.03	0.01	0.04	0.06	0.06	0.14	0.12	0.19	0.07	0.18	0.03	0.15	0.20	0.24

C	nb.ld1.	nb.ld2.	nb.ld3.	nb.ld4.	nb.ld5.	nb.ld6.	nb.ld7.	nb.ld8.	nb.ld9.	nb.ld10.
μ	50.0	52.8	75.6	80.7	82.0	83.4	84.5	84.7	84.9	84.8
σ	0.01	0.06	0.08	0.06	0.10	0.03	0.10	0.09	0.05	0.09

nb.1.	nb.2.	nb.3.
81.7	83.5	81.2
0.08	0.13	0.13

TABLE II
ACCURACY RESULTS FOR DIFFERENT CLASSIFIERS BASED ON THE IMDB-NOT DATASET.

C	ld1	ld2	ld3	ld4	ld5	ld6	ld7	ld8	ld9	ld10	ld11	ld12	ld13	ld14	ld15	ld16	ld17	ld18
μ	14.7	49.3	50.7	50.9	50.9	51.6	52.5	54.8	58.8	63.8	68.9	72.9	76.2	78.4	79.5	80.1	80.0	79.4
σ	18.82	7.46	0.27	0.12	0.02	0.01	0.04	0.05	0.11	0.13	0.14	0.16	0.22	0.12	0.13	0.14	0.23	0.09

C	nb.ld1.	nb.ld2.	nb.ld3.	nb.ld4.	nb.ld5.	nb.ld6.	nb.ld7.	nb.ld8.	nb.ld9.	nb.ld10.
μ	50.0	51.6	74.2	79.8	82.0	83.3	84.1	84.8	85.2	84.9
σ	0.004	0.032	0.206	0.083	0.050	0.120	0.108	0.063	0.139	0.115

nb.1.	nb.2.	nb.3.
81.2	83.8	81.7
0.06	0.02	0.18

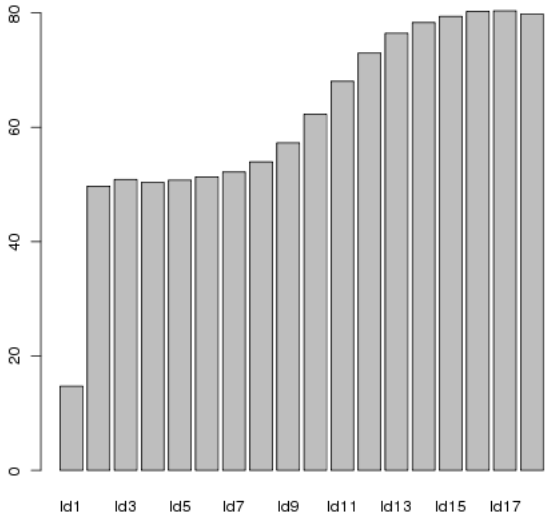

Fig. 1. Accuracy results for the LD_k^3 classifier.

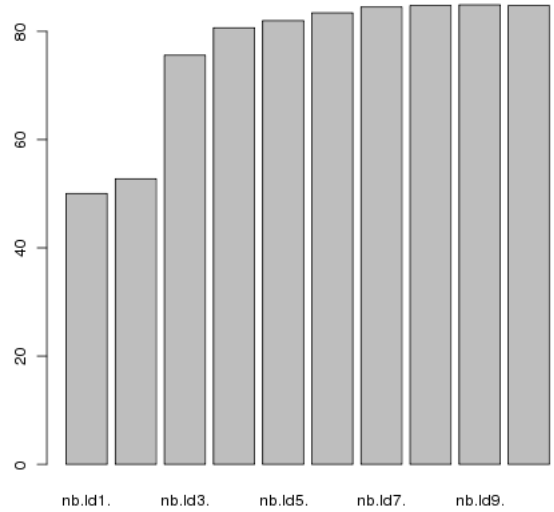

Fig. 2. Accuracy results for the Naive Bayes classifier based on character n-grams.

between the test documents and the training set. The character n-gram perform better, because the probability of finding an exact string of length n in the training set is much higher than finding an n-gram, which makes the character n-gram parameter more flexible. Here one has to keep in mind that in a text of length m there are almost $m * n$ character n-grams whereas only about $(m * n)/k$ word n-grams (given the average word length is about k). Here it would be very interesting to work with much larger labeled datasets and compare the performance.

We have also observed that the quality of the classification is strongly correlated with the number of reviews used, i.e. a classification with a 0.7 training/test ratio will lead to much less classification accuracy than a ratio of 0.9. This leads us to the assumption that there may not be enough reviews in the IMDb dataset, compared to the difficulty of the task. The variance in the data is too high in relation of the amount of data available. This also means that one has to be careful when comparing classification results from different authors even though they are using the exact same dataset. For datasets of similary size to the IMDb dataset classification accuracies calculated with different cross-validation strategies result in difference accuracy values for the exact same classifier.

VII. CONCLUSION

We have demonstrated that for text sentiment classification, character n-grams perform at a high level and are capable of achieving results comparable to highly sophisticated methods. This is especially interesting as character n-gram require an almost minimal amount of prior knowledge, even

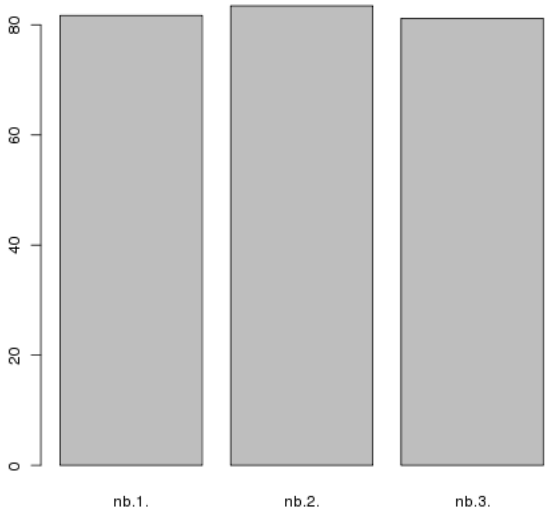

Fig. 3. Accuracy results for the Naive Bayes classifier based on word n-grams.

compared to word n-grams. Character n-grams make less assumptions about the data than, because they model a text as a collection of characters rather than words. Given the size of the available datasets we demonstrated that character n-grams are efficient than more intuitive approaches such as word n-grams. Here it would be of great interest to repeat the presented evaluation on much larger datasets.

REFERENCES

[1] P. Mcnamee and J. Mayfield, "Character n-gram tokenization for european language text retrieval," *Inf. Retr.*, vol. 7, no. 1-2, pp. 73–97, 2004.

[2] Y. Miao, V. Kešelj, and E. Milios, "Document clustering using character n-grams: a comparative evaluation with term-based and word-based clustering," in *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. New York, NY, USA: ACM, 2005, pp. 357–358.

[3] M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi, "The similarity metric," *Information Theory, IEEE Transactions on*, vol. 50, no. 12, pp. 3250–3264, Dec. 2004.

[4] R. Cilibrasi and P. Vitanyi, "Clustering by compression," *Information Theory, IEEE Transactions on*, vol. 51, no. 4, 2005.

[5] H. Tang, S. Tan, and X. Cheng, "A survey on sentiment detection of reviews," *Expert Syst. Appl.*, vol. 36, no. 7, pp. 10 760–10 773, 2009.

[6] M. Hurst and K. Nigam, "Retrieving topical sentiments from online document collections," in *Document Recognition and Retrieval XI*, 2004, pp. 27–34.

[7] P. Turney, "Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews," 2002, pp. 417–424. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.1992

[8] S.-M. Kim and E. Hovy, "Determining the sentiment of opinions," in *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2004.

[9] J. Blitzer, M. Dredze, and F. Pereira, "Biographies, Bollywood, boomboxes and blenders: Domain adaptation for sentiment classification," in *Proceedings of the Association for Computational Linguistics (ACL)*, 2007.

[10] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[11] T. Mullen and N. Collier, "Sentiment analysis using support vector machines with diverse information sources," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, July 2004, pp. 412–418, poster paper.

[12] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," in *In Proceedings of the ACL*, 2004, pp. 271–278. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.9144

[13] N. Collier, K. Takeuchi, A. Kawazoe, T. Mullen, and T. Wattarujeekrit, "A framework for integrating deep and shallow semantic structures in text mining," in *KES*, 2003, pp. 824–834.

[14] S. R. Das and M. Y. Chen, "Yahoo! for Amazon: Sentiment extraction from small talk on the Web," *Management Science*, vol. 53, no. 9, pp. 1375–1388, 2007.

[15] A. Fahrni and M. Klenner, "Old Wine or Warm Beer: Target-Specific Sentiment Analysis of Adjectives," in *Proc.of the Symposium on Affective Language in Human and Machine, AISB 2008 Convention, 1st-2nd April 2008. University of Aberdeen, Aberdeen, Scotland*, 2008, pp. 60 – 63.

[16] X. Ding and B. Liu, "The utility of linguistic rules in opinion mining," in *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2007, pp. 811–812.

[17] S. Matsumoto, H. Takamura, and M. Okumura, "Sentiment classification using word sub-sequences and dependency sub-trees," in *PAKDD*, 2005, pp. 301–311.

[18] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambridge University Press, July 2008.

[19] J. Han and M. Kamber, *Data mining*. Morgan Kaufmann Publ., 2001.

[20] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment classification using machine learning techniques," in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002, pp. 79–86.