A Literature Review of Sentiment Classification Algorithms- Including an Experimental Implementation

Nikki Kyllonen

CSCI 5512, May 2019

Abstract

Sentiment Classification (also known as **Opinion Mining**) is a subset of the field of **Natural Language Processing** (NLP) focused on extracting qualitative data (sentiments) from human (natural) languages. Many successful implementations make use of **Machine Learning** (ML) algorithms; as does the implementation discussed in this paper. In this paper, we discuss the simple approach of using a Bag of Words model, a Random Forest Classifier, and Google's Distributed Word Vector package, *Word2Vec*. Although not the most accurate analyzer, this implementation is simple and aids in understanding the processes behind Sentiment Classification.

1 Introduction

The core problem that NLP attempts to solve has fascinated philosophers for centuries. The goal of NLP is to allow machines to process human language; to empower a man-made device to be able to make sense of natural languages developed by humans. Defining languages as a container by which data is stored and shared, NLP essentially aims to translate: to fluidly transition between languages, from one container to another. Tasks related to this translation process can be coarsely classified into the following fundamental problems: Syntax, Semantics, Discourse, Speech, and Dialogue [1, 2].

1.1 NLP Before ML (pre-1980s)

The beginning of modern NLP can be roughly marked assigned to the publication of Alan Tur-

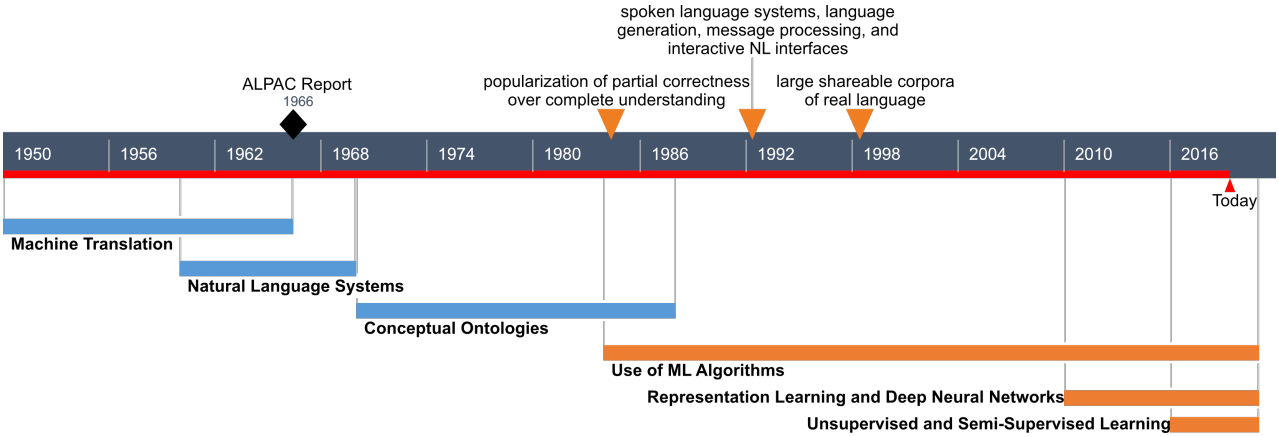

Figure 1: Rough timeline of the development of NLP research topics from the 1950s onwards.

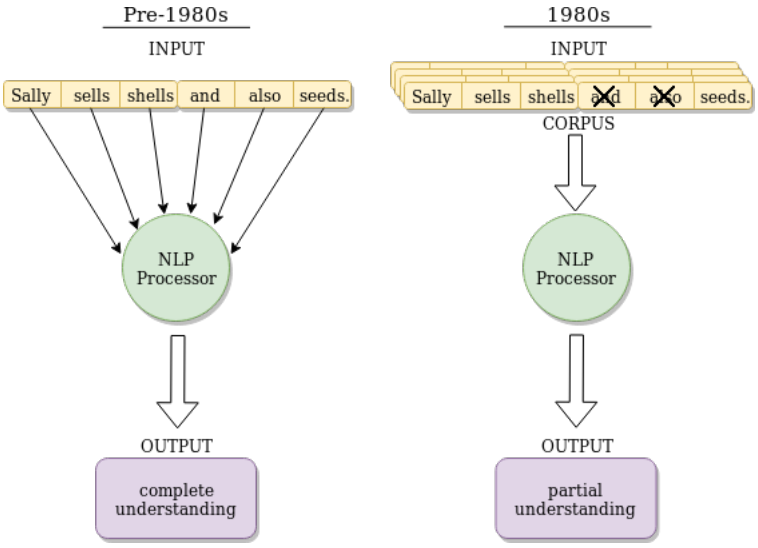

Figure 2: Diagram representing the change in the fundamental approach of NLP in the 1980s.

ing's article "Computing Machinery and Intelligence" where he proposed a set of criteria for determining intelligence that would later become known as the Turing Test [3]. This publication and the advances in **machine translation** that were happening alongside it marked the rise of NLP in the 1950s.

The hype surrounding machine translation and its seemingly predictable rapid growth came to an end in the US with the release of a report by the Automatic Language Processing Advisory Committee (ALPAC). The American organization concluded that the field had *not* advanced as rapidly as predicted and that human translators were by far still more effective and efficient. The ALPAC report marked the 1960s as being NLP's shift to **natural language systems**. This shift, however, wasn't as strong elsewhere as it was in the US. Other countries, such as Canada and many European countries, continued to pour funding into machine translation. In these countries, there were strong bilingual or multilingual cultures that demanded a high quantity of translated documents [4].

Natural language systems aimed to respond to typed queries and there was some success with systems such as SHRDLU[5] and ELIZA[6]. However, these approaches were strongly knowledge-based approaches and relied completely on a database of possible responses created by the researchers themselves. This hard-coded, strictly structured approach to representing data about languages can be referred to as the creation of **conceptual ontologies**, or representations of structured data about the real-world. The databases of responses used in the 1960s are an example of a conceptual ontology. In the 1970s, these ontologies were expanded to contain complex rules that attempted to distill grammar and other linguistic knowledge into something a machine could understand.

Up until the 1980s, all NLP tasks used every part of provided input while processing. At the time, researchers believed that to generate a complete understanding of the input, the entire input needed to be used. This theology changed in the 1980s for a few reasons. The first being that the level of complete understanding NLP had been aiming to achieve up now seemed more than just improbable, it seemed infeasible. The second reason being the result of Moore's Law: greater computational power. And finally, the availability of **corpus linguistics** that contained large samples of real-world text or corpora. Researchers now would be able to process more input and faster.

By throwing out a portion of the input, researchers began to see the benefits of striving for

partial understanding. Although they were unable to achieve as high of a level of correctness on their outputs, they were able to overall achieve better success since the new algorithms failed much less often. This shift in mindset helped to open up the field to a new set of algorithms: **Machine Learning Algorithms**.

1.2 Incorporating ML (post-1980)

Learning algorithms are characterized by their flexibility and adaptability. They lack a reliance on exactness and are able to improve their own performance mid-execution "whether by adjustment of variables (which are regarded as data) or rules (which are regarded as program)". In the 1980s, NLP research began to accept these approaches as valid, noting that they very closely resembled how natural language is acquired by humans [7].

When ML algorithms were first applied to NLP, they manipulated tree representations of the input corpora. These algorithms added a new level of flexibility that set them apart from previous approaches [4]. However, tree algorithms, such as decision trees, still built ontologies similar to what had been done before: hard if-then rules [1].

In the late 1980s, part of what drove the rapid development of NLP was the commercialization of NLP technology. The shift to processing larger corpora as input led to the rise of grammar and style checkers [2, 8]. The combination of these developments partially drove the increased use of ML algorithms in NLP. With the use of part-of-speech tagging, Hidden Markov Models (HMMs) took over the field, paving the way for the ultimate ML transformation: the use of **statistical models**. The 1990s heralded ML's increasing focus on using statistical models and probability theory and therefore the birth of the "statistical revolution" in NLP [8, 1, 4].

Johnson differentiates the new *statistical approach* from the older *grammar-based approach* by describing that the latter relied on rules or grammars created by linguists while the former instead relied on corpora annotated by linguists [8]. By not relying on strict rules, soft, probabilistic decisions were made that were much more robust. This robustness was invaluable when the input corpora were increasingly *not* lab-grown material. The 1990s marked the use of *real-world* texts as input corpora; expanding NLP into an even more valuable tool.

The use of ML has only continued to expand NLP research and applications. In the past decade, **representation learning**, **deep neural networks**, and **unsupervised and semi-supervised algorithms** have also found their places within the field of NLP [1].

2 Related Work

Sentiment Classification fits into the aforementioned Semantics category as it uses the words themselves to extract meaning. We can describe Sentiment Classification as a subcategory of NLP which aims to translate a collection of human words into a sentiment value. This translation is described by Li et al. as "a special task of **text classification** whose objective is to classify a text according to the sentimental polarities of opinions it contains (Pang et al., 2002 [10]), e.g., *favorable* or *unfavorable*, *positive* or *negative*" [9].

In the proceedings referenced by Li, Pang et al. used three ML algorithms on movie reviews to analyze the effectiveness of each in extracting sentiment. The classifiers employed were **Naive Bayes**, **maximum entropy classification**, and **support vector machines** (SVMs). The results of the paper showed that Naive Bayes performed the worst and SVMs performed the best. However, Pang et al. make the note that the differences between the methods were *not* very large [10].

While analyzing their final results, Pang et al. investigated the effects of **feature frequency** vs. **feature presence** in their Naive Bayes and SVM classifiers. In text classification, feature frequency takes into account the total number of appearances a feature makes. Depending on the **N-gram model** used by the classifier, features can vary in the number of words they represent: each feature be a value linked

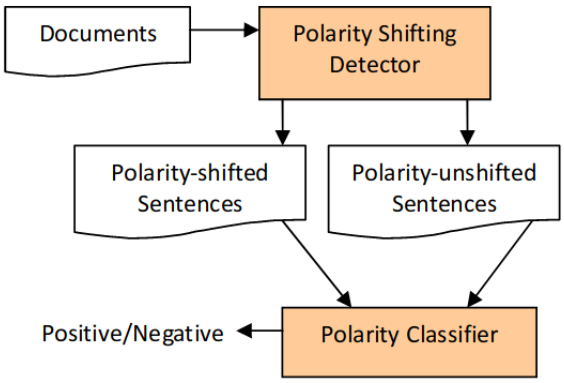

Figure 3: General framework of the approach used in Li et al. 2010 to incorporate polarity shifting into their sentiment classification algorithm [9].

to a group of n words. In order to remove the frequency variable, the **document vectors** employed by the classifiers $(n_0(d), n_1(d), ..., n_m(d))$ were adjusted so that $n_i(d) = 1$ to indicate the presence of and $n_i(d) = 0$ to indicate the absence of the ith feature. This isolation showed that much better performance is achieved by disregarding frequency, indicating that in sentiment classification, presence is a greater indicator of sentiment than frequency [10].

Another variable tested by Pang et al. was how to build each feature itself: what value of n was more effective? For each classifier, a **unigram model** was compared against a **bigram model**, corresponding to $n = 1$ and $n = 2$ respectively. The results of each classifier were the same: bigrams do *not* perform better than unigrams and they "[cause] accuracy to decline by as much as 5.8 percentage points"[10].

Li et al. expanded on these and other findings in order to try to address one of the challenges of sentiment classification: **polarity shifting**. Of the four different levels of sentiment classification (word, phrase, sentence, and document) document-level classification was chosen. Document-level classification is characterized by two approaches: **term-counting** and **machine learning**, which Li et al. describes as lexicon-based and corpus-based, respectively [9].

Li et al. successfully addressed the challenge of polarity shifting by using the framework shown in Figure 3. This framework trains two machine-learning-based classifiers, one for polarity-shifted data and one for polarity-unshifted data. By adding this extra intermediate step, Li et al. was able to "consistently improve the overall performance across different domains and training data sizes" [9].

In 2013, Li et al. once again faced the challenge of polarity shifting. However, they chose to use a **bag-of-words** based machine learning approach as their base algorithm. Expanding on their previous findings, they once again injected an intermediate step to detect polarity-shifted sentences. They then used the results of that classifier to train a "novel term counting-based classifier" with much success. Even greater performance was achieved when this counting-based classifier was combined with a machine-learning classifier [11].

3 Experiment

In my approach, I also chose to implement document-level classification using a bag-of-words model, a **random forest classifier**, and Google's *Word2Vec* **distributed word vector** package. To do this, I followed Kaggle's *Bag of Words Meets Bags of Popcorn* tutorial competition instructions. These can be found here at this link: `Word2Vec NLP Tutorial`.

As hinted in the title of the tutorial, the dataset used consisted of 50,000 IMDB movie reviews. The aim was to perform a binary senti-

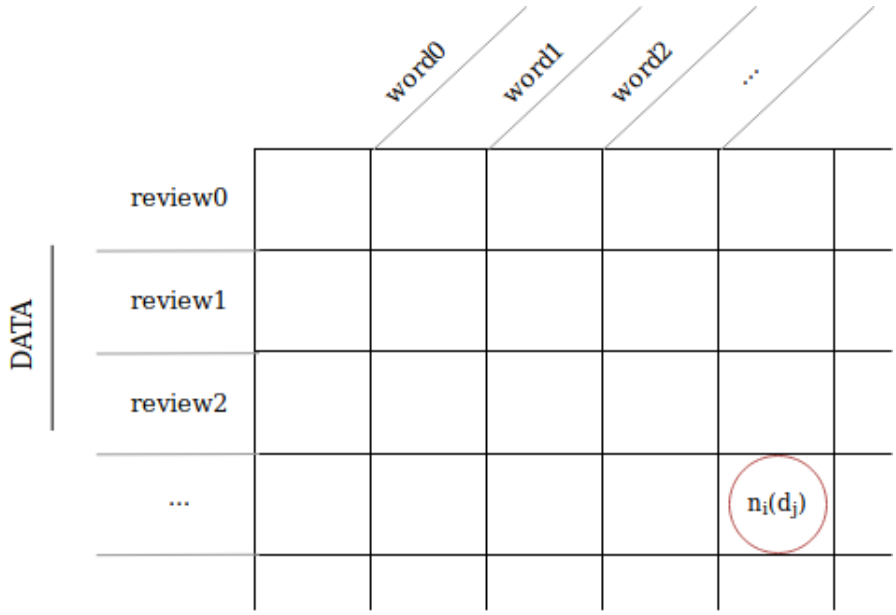

Figure 4: Visual representation of the bag of words produced by *build_bag*(). Each feature $n_i(d_j)$ is the count of $word_i$ in $review_j$.

ment classification that decided if a review was positive (1) or negative (0). The binary classifications of the training data were determined using the associated IMDB rating: < 5 resulting in 0 and ≥ 7 resulting in 1. The training set consisted of 25,000 of the reviews while the testing set consisted of the remaining 25,000.

For this report, I was only able to complete Part 1 of the tutorial, **Basic Natural Language Processing**, which did not employ **deep learning** techniques. In the future, I would like to continue through Parts 2 and 3: **Deep Learning for Text Understanding**.

3.1 Implementation

1. Bag-of-Words

The first step in any machine-learning algorithm is build a numeric representation of the input data. This representation contains specific feature values and are often stored in a **feature vector**. The input data is transformed into a series of vectors representing each of the documents (reviews). To do this transformation, I used the bag-of-words model.

The bag-of-words model distills a sentence or a document into a feature vector characterized by words of size n, depending on the n-gram model in use. For further simplicity, I chose to use a unigram model for my features, therefore dividing my documents into individual words. The output of my bag-building can be visualized as a large matrix such as the one in Figure 4, where each row is a feature vector.

Before vectorizing, however, I sent each review through a preprocessing step. The tutorial recommended removing punctuation, numbers, and stopwords for simplicity, but commented on this step as being one open to interpretation. I decided to be more strict and to remove all instances of these characters. Since my time was very limited, I wanted to be able to more quickly complete my classifier. I have, however, noted this step as a place to return to in the future. By preprocessing differently, I could potentially improve my results without changing any other parts of the overall algorithm.

My cleaned data then consisted of strings of completely lowercased words contained within an array. This array was then given to a `CountVectorizer` object. `CountVectorizer` is part of the `sklearn` python3 package and is used

to fit a bag-of-words model, learn vocab, and transform the input data into feature vectors.

2. Random Forest Classifier

- included in scikit-learn
- uses multiple tree-based classifiers to train
- supervised learning

3. Distributed Word Vectors (*Word2Vec*)

- built by Google's Word2Vec algorithm
- "distributed representations" of words
- does not need labels
- with enough training data, able to create unique and clustered vectors – some word relationships can be reproduced using vector math

Word2Vec:

- uses a neural network to learn
- able to greatly reduce the amount of time needed to train deep/recurrent neural network models

3.2 Results

First 15 out of 25,000 lines of output csv file: [12]

id	sentiment
"12311_10"	1
"8348_2"	0
"5828_4"	1
"7186_2"	1
"12128_7"	1
"2913_8"	0
"4396_1"	0
"395_2"	1
"10616_1"	0
"9074_9"	1
"9252_3"	1
"9896_9"	1
"574_4"	1
"11182_8"	1

From spotchecking the test data of reviews, against the output sentiment, my results seem roughly 50% accurate (at best).

4 Conclusion

Conclusion/summary and future work

References

[1] "Natural language processing," *Wikipedia*, May 2019.

[2] M. Bates, "Models of natural language understanding," pp. 9977–9982, Oct. 1995. Proceedings of the National Academy of Sciences of the United States, vol. 92, no. 22.

[3] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, p. 433, 1950.

[4] W. Hutchins, "Machine translation over fifty years," *Histoire Épistémologie Langage*, vol. 23, pp. 7–31, 2001.

[5] "Shrdlu," *Wikipedia*, Feb. 2019.

[6] "Eliza," *Wikipedia*, April 2019.

[7] D. M. Powers and C. C. Turk, *Machine Learning of Natural Language*. Springer Science Business Media, 2012.

[8] M. Johnson, "How the statistical revolution changes (computational) linguistics," in *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics*, pp. 3–11.

[9] S. L. et al., "Sentiment classification and polarity shifting," in *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pp. 635–643, Aug. 2010.

[10] L. L. Bo Pang and S. Vaithyanathan, "Thumbs up? sentiment classification using

machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing (EMNLP)*, vol. 10.

[11] S. L. et al., "Sentiment classification with polarity shifting detection," in *Proceedings of the 2013 International Conference on Asian Language Processing*, pp. 129–132, Aug. 2013.

[12] N. Kyllonen, "Project github repository." https://github.com/nkyllonen/CSCI5512-AI2/tree/master/nlp.