Comparing Text Vectorization Techniques for Sentiment Analysis Task

732A92 Text Mining Project Report

Bayu Beta Brahmantio (baybr878)

Department of Computer and Information Science
Linköping University
March 13, 2021

Abstract

Numerical representation of texts as vectors is necessary for text classification tasks, including sentiment analysis. This project compares the performance of different classifiers on IMDB review dataset based on four text vectorization methods: bag-of-words, bag-of-words with tf-idf weights, average of word vectors, and DistilBERT. The results showed that tf-idf vectorizer paired with logistic regression or linear support vector classifier has the best result with 88% accuracy on the test data.

1 Introduction

Text classification is a recurring problem in the natural language processing field. One of its many applications is sentiment analysis which tries to identify the subjective information in a given word or text. The usual task for sentiment analysis is classifying polarization of a text e.g. whether a text has positive or negative sentiment.

Generally, given a collection of N documents paired with its class, $(d_1, c_1), ..., (d_N, c_N)$, we want to find a classifier f that takes d as an input and gives us a correct class $c \in C$. There is a wide range of classifier suitable for this task. However, we first need to represent the documents numerically before we can feed them into classifiers.

There are many ways to represent documents as vectors. The simplest one is to represent a document as a set of unordered words along with their frequency, also known as bag-of-words. The documents are represented as vectors of same length in which each element represents a frequency of a term in a given document. The term frequency (tf) can also be weighted by its inverse document frequency (idf) which takes into account the frequency of the term appearing in the collection. The product of tf and its idf results in tf-idf value (Jurafsky and Martin [2009]).

Word embeddings techniques such as GloVe (Pennington et al. [2014]) and word2vec (Mikolov et al. [2013]) allows us to represent words as dense vectors and learn similarities between them. It is also possible to learn contextual embedding as done in BERT (Devlin et al. [2019]) where each word can have different representation depending on contexts.

In this project, different techniques for representing texts as vectors will be compared: bag-of-words with tf, bag-of-words with tf-idf weights, word embedding using spaCy's pre-trained word vectors (Honnibal et al. [2020]), and DistilBERT (Sanh et al. [2020]), a smaller version of BERT. In the case of word embedding, the average of word vectors will be used to represent individual document. In DistilBERT's case, the final hidden state of special classifier token ([`CLS`]) can be used as the document's representation. They will be compared in terms of their performance in a sentiment analysis task using different classifier models: multinomial naive Bayes, logistic regression, and linear support vector machine (SVM).

2 Theory

2.1 Term Frequency and Bag-of-words Model

A collection of documents can be viewed as a term-document matrix, where the columns represent documents and the rows represent words in the vocabulary. Each element in the matrix contains the term frequency or the number of times the word appears in the document. Hence, each column of the matrix can be considered as a point in a $|V|$-dimensional vector space, where $|V|$ is the vocabulary size (Jurafsky and Martin [2009]). Using this representation, a document is represented as a bag-of-words, which only keeps the frequency of terms but the orders are neglected.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 1: The term-document matrix for four words in four Shakespeare plays (Jurafsky and Martin [2009]). Each document is represented as a column vector containing values for each term frequency.

2.2 TF-IDF

In Figure 1, each document is represented by the frequencies of its words. However, not all words are relevant as discriminators just because they are abundant (e.g. the, at, in). Hence, rather than using raw frequency values, we can introduce weights to make some terms more relevant than the others. First, we define the term frequency for the word t and document d as:

$$\mathrm{tf}_{t,d} = \mathrm{count}(t, d). \tag{2.1}$$

One way to reduce the impact of highly-occuring terms is to apply the log function to (2.1):

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t, d) + 1). \tag{2.2}$$

With this weight, a term with 100 times more occurence will only have 2 times the weight.

Another way to weight frequencies is to use inverse document frequency (idf):

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right) \tag{2.3}$$

where N is the total number of documents in the collection and df_t is the number of documents in which term t appears. This gives more importance to terms that are less likely to be found on many documents. Finally, the product of (2.2) and (2.3) is the tf-idf weight:

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t. \tag{2.4}$$

In the `scikit-learn` package (Pedregosa et al. [2011]) tf-idf function is defined by default as:

$$\text{tf-idf}_{t,d} = \mathrm{count}(t, d) \times \log\left(\frac{N+1}{\mathrm{df}_t + 1} + 1\right). \tag{2.5}$$

2.3 Word Embeddings

Word embeddings allow us to represent words as dense vectors with a significantly smaller dimension than the size of vocabulary. Besides that, they also encode similarities between words in a high-dimensional space (Maas et al. [2011]). There are many ways to generate the mapping. Bengio et al. [2003] used neural model to learn a distributed representation for each word together with its probability function for word sequences. GloVe (Pennington et al. [2014]) used global word-word co-occurence matrix from a corpus to produce a vector space. In word2vec (Mikolov et al. [2013], Mikolov et al. [2013a]), word embeddings are learned using two kinds of method: continuous bag-of-word model and continuous skip gram model. Continuous bag-of-word model (CBOW) predicts a word between multiple context words. Continuous skip gram model predicts words before and after the current word.

All the examples mentioned above are context-free embeddings, where a model learns a fixed representation for each word no matter the contexts. To generate contextual word embeddings, we need a model that can learn a representation of a word based on other words in the sentence.

BERT (Devlin et al. [2019]) uses multi-layer bidirectional Transformer (Vaswani et al. [2017]) to learn contextual word representations. Standard approaches to train language models using BERT includes pre-training with a large, unlabelled dataset and fine-tuning the model on a smaller, labelled dataset. A pre-trained BERT model can also be used as a feature-based model, where it generates activation values based on a sequence of tokenized words corresponding to a sentence. These activation values are contextual embeddings that can be used as inputs for other models.

A smaller and faster variant of BERT is DistilBERT (Sanh et al. [2020]). By reducing the size of a BERT model and using knowledge distillation (Hinton et al. [2015]) in the pre-training, it managed to achieve similar performance while also being faster.

2.4 Multinomial Naive Bayes Classifier

A naive bayes classifier uses the naive assumption that all features are mutually independent given a class $c \in C$. The best class c given a set of words w is described as:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}}\, p(c|w) = \underset{c \in C}{\operatorname{argmax}}\, p(c) \prod_{w \in V} p(w|c) \tag{2.6}$$

where w are words in the vocabulary V. The maximum likelihood estimates for $p(c)$ and $p(w|c)$ are given by:

$$\begin{aligned} \hat{p}(c) &= \frac{N_c}{N_{doc}} \\ \hat{p}(w_i|c) &= \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)} \end{aligned} \tag{2.7}$$

where N_c is the number of documents in class c, N_{doc} is the total number of documents, and $\text{count}(w_i, c)$ is the number of occurrences of word w_i in all documents that belong to class c.

To deal with words that are not in training data, an additive term is used in $\hat{p}(w_i|c)$ so that $p(c|w)$ is not multiplied to zero:

$$\hat{p}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V} \text{count}(w, c)) + |V|} \tag{2.8}$$

(Jurafsky and Martin [2009]).

2.5 Logistic Regression

Given an observation $\mathbf{x} = (x_1, ..., x_n)$ and classes $y \in \{0, 1\}$, we want to find the probability of the observation is from each class, $p(y|\mathbf{x})$. In this case, $p(y = 1|\mathbf{x})$ could be the probability of "positive sentiment" and $p(y = 0|\mathbf{x})$ is for "negative sentiment". To model the probability using linear functions of $\mathbf{x}$, we can use the sigmoid function so the output is going to be in the range [0,1]:

$$\begin{aligned} p(y=1|\mathbf{x}) &= \boldsymbol{\sigma}(w \cdot \mathbf{x} + b) \\ &= \frac{1}{1+\exp(-(w \cdot \mathbf{x} + b))} \\ p(y=0|\mathbf{x}) &= 1 - \boldsymbol{\sigma}(w \cdot \mathbf{x} + b) \\ &= 1 - \frac{1}{1+\exp(-(w \cdot \mathbf{x} + b))} \\ &= \frac{\exp(-(w \cdot \mathbf{x} + b))}{1+\exp(-(w \cdot \mathbf{x} + b))} \end{aligned} \tag{2.9}$$

for a set of weights w and a bias term b. We define our decision boundary as:

$$\hat{y} = \begin{cases} 1 & p(y=1|\mathbf{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

To learn w and b, we first define a loss function:

$$\begin{aligned} L(\hat{y}, y) &= -[y \log \hat{y} + (1-y)\log(1-\hat{y})] \\ &= -[y \log \boldsymbol{\sigma}(w \cdot \mathbf{x} + b) + (1-y)\log(1 - \boldsymbol{\sigma}(w \cdot \mathbf{x} + b))] \end{aligned} \tag{2.11}$$

and use an optimizer algorithm to find w and b that minimize the loss function given labels y and our predictions $\hat{y}$ (Jurafsky and Martin [2009]). By default, `scikit-learn` uses Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS).

2.6 Linear Support Vector Classifier

Suppose that we have n pairs of training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$. Define the classification rule in the hyperplane as:

$$\hat{y} = \begin{cases} 1 & w \cdot \mathbf{x} + b > 0 \\ -1 & w \cdot \mathbf{x} + b < 0 \end{cases} \tag{2.12}$$

where $||w|| = 1$. We can find a hyperplane that creates the biggest margin between classes 1 and -1 by performing optimization problem:

$$\begin{aligned} &\min_{w,b} ||w|| \\ &\text{subject to } y_i(w \cdot \mathbf{x}_i + b) \geq 1,\ i = 1, ..., n \end{aligned} \tag{2.13}$$

(Hastie et al. [2001]).

3 Data

The dataset used is a collection of 50,000 movie reviews along with its sentiment from Internet Movie Database (IMDB) (Maas et al. [2011]). It is split evenly into 25,000 reviews in the training and test set. There is also a balanced number of negative and positive reviews in each set.

Instead of taking into account all kinds of reviews, Maas et al. [2011] only collected highly-polarized reviews, that is, only reviews that are considered negative and positive are included. A negative review has an IMDB score of ≤ 4 while a positive one has a score of ≥ 7. The scores are in the range of $[0, 10]$. Since it is a case of balanced dataset where the classes are split evenly in training and test dataset, the expected accuracy of a random classifier will be around 50%. Overview of the training data can be seen in Table 1.

Review	Sentiment
Story of a man who has unnatural feelings for ...	negative
Airport '77 starts as a brand new luxury 747 p...	negative
This film lacked something I couldn't put my f...	negative
Sorry everyone,,, I know this is supposed to b...	negative
When I was little my parents took me along to ...	negative
⋮	
Bromwell High is a cartoon comedy. It ran at t...	positive
Homelessness (or Houselessness as George Carli...	positive
Brilliant over-acting by Lesley Ann Warren. Be...	positive
This is easily the most underrated film inn th...	positive
This is not the typical Mel Brooks film. It wa...	positive
⋮	

Table 1: Overview of the dataset. The reviews shown in the table are from the training set with the top five rows as the first five negative reviews and the five rows below that as the first five positive reviews.

4 Method

In this section, the technicalities behind the experiment will be discussed. In general, the same training and test set will be used for all classifier methods. Since they already comes in same sizes, a further split is not needed. The accuracy of each classifier in each method will then be calculated using `scikit-learn`'s `classification_report`. For each text vectorization method, all three classifiers will be used except for word embeddings and DistilBERT that are not compatible with multinomial naive Bayes classifier which does not accept negative values.

All classifiers (multinomial naive Bayes, logistic regression, and linear SVC) uses `scikit-learn`'s implementations with default settings. Specifically for logistic regression and linear SVC, the parameter `random_state` will be set to 1234 so the results are reproducible. A classifier will be trained on the training data and the accuracy of predicted classes based on the test data will be measured. The baseline random classifier is expected to achieve accuracy of around 50% since the classes are balanced.

Bag-of-words

The `CountVectorizer` method from `scikit-learn` transforms a given review text into a sparse vector of the same length as the number of words in the vocabulary of training data. This transforms each text into a "bag of words" containing word frequencies. Default parameters will be used for this method.

Bag-of-words with TF-IDF

This method is essentially similar to the previously mentioned bag-of-words but each word will have its own weight. The `TfidfVectorizer` method is used to transform each review text into a sparse vector of weighted frequencies. Default parameters are also used for this method.

Word Embeddings using spaCy Pre-trained Vectors

Pre-trained word embeddings from `spaCy` is used to transform each word into a 300-dimensional length vector. It is done by using `spaCy`'s large english pipeline that was pre-trained on web-text, `en_core_web_lg`. To make a representation of a review text, the average of vectors of all words in that text will be used. This method is commonly used as a cheap and fast method and as a baseline to be compared with more advanced document-level representation methods (Socher et al. [2013], Mikolov et al. [2013a]). The process is done by creating a method called `MeanSentenceVectorizer` that is compatible to be used in `scikit-learn`'s `pipeline` method.

DistilBERT

The model used is a pre-trained DistilBERT model made available by Hugging Face's `transformers` package (Wolf et al. [2020]). Each text will be feed to a tokenizer that is also available from `transformers`. The maximum number of tokens is set to 512. The tokenized text will be fed to the DistilBERT encoder and each tokenized word will have activation units that can be used as a vector representation, including the special token [`CLS`] that represents the whole text. The activation units for [`CLS`] is then used as features for classifier algorithms.

5 Results

The performances of text vectorization methods are compared in Table 2. It is based on the accuracy of predictions given the test data. We can see that the bag-of-words with

TF-IDF method (`TfidfVectorizer`) paired with logistic regression or linear SVC gives the best results with 88% accuracy on test data.

In general, logistic regression and linear SVC give similar performance in every text vectorization methods except in bag-of-words method (`CountVectorizer`). Both `MeanSentenceVectorizer` and DistilBERT do not manage to get better `TfidfVectorizer` and `CountVectorizer` results on logistic regression and linear SVC. On the other hand, the multinomial naive Bayes classifier gives relatively worse accuracy than all other models on all text vectorization methods.

	MultinomialNB	LogisticRegression	LinearSVC
CountVectorizer	0.82	0.86	0.85
TfidfVectorizer	0.83	0.88	0.88
MeanSentenceVectorizer	-	0.85	0.85
DistilBERT	-	0.85	0.85

Table 2: Accuracy of different text vectorization methods on different classifier algorithms.

6 Discussion

Despite being more complicated, `MeanSentenceVectorizer` and DistilBERT yield similar or even worse results than the relatively simpler bag-of-words methods. This should come in as no surprise since the bag-of-word methods (`CountVectorizer` and `TfidfVectorizer`) are fitted to the data rather than used as a method that is pre-trained on a corpus not related to the sentiment analysis problem. Still, they achieve competitive results that can be used as baselines compared to more advanced methods.

In DistilBERT's case (or BERT in general), the standard modelling procedure is to include *fine-tuning* phase where all pre-trained parameters are fine-tuned to better suit the downstream task (Devlin et al. [2019]). However, this is a relatively expensive process that could take hours even with GPUs. This project shows that even without the fine-tuning phase, DistilBERT can still be used in a feature-based approach to produce document-level representations and still achieve a reasonable result for sentiment analysis task.

The same can be said about the word vectors averaging (`MeanSentenceVectorizer`). It is not sophisticated enough to beat state-of-the-art methods, although it could be a good baseline. This is due to the average of word vectors being unordered, meaning that it ignores the orders of words, hence the context of words in a given document. A couple of methods such as Iyyer et al. [2015] and Arora et al. [2017] took a similar approach but managed to achieve comparable results to more sophisticated models including RNN and LSTM while also being simpler and faster.

Ultimately, bag-of-words based method is still a reliable way to vectorize texts for a sentiment analysis task. Its strength lies in its speed to represent texts numerically. This makes it an ideal method for baseline.

7 Conclusion

In this project, four different text vectorization methods (bag-of-words, bag-of-words with tf-idf weights, average of word vectors, and DistilBERT) are compared in terms of their performance in a sentiment analysis task. Bag-of-words with tf-idf weights managed to get the best results with 88% accuracy using logistic regression and linear support vector classifier. On the other hand, pre-trained methods (average of word vectors, and DistilBERT) performed similarly if not worse than the other two methods.

8 Appendix

All codes and dataset are available in this repository

References

Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, Upper Saddle River, N.J., 2009. ISBN 9780131873216 0131873210. URL http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020. URL https://doi.org/10.5281/zenodo.1212303.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P11-1015.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003. ISSN 1532-4435.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/D13-1170`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. 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)*, pages 1681–1691, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1162. URL `https://www.aclweb.org/anthology/P15-1162`.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*, 2017.