以文本情感分类为例的文本表征学习研究

**学号：SY2124108 姓名：孙浩哲**

（北京航空航天大学 中法工程师学院 北京 100191）1

**摘 要** 在机器学习中，获取语言序列的向量表示并不是一项简单的任务，因为语言本身具有离散性和含时性。文本情感分析是自然语言处理领域的标志性任务。本文以支持向量机作为模型框架，研究不同的文本表示方法对于情感分析的准确率的影响。

**关键词** 文本情感分析，自然语言处理，表征学习

**Natural language representation learning: a case study on sentiment analysis**

Haozhe Sun

(Ecole Centrale de Pékin, Beihang University, Beijing, 100191, China.)

**Abstract**: In machine learning, finding a vector representation of a text sequence (vector sentence model, VSM) is not trivial as language is intrinsically discrete and time-dependent. Sentiment analysis is one of the key tasks in modern natural language processing (NLP). In this work, we use support vector machines (SVM) to study different representation learning methods’ influence on the final classification precision.

**Keywords:** sentiment analysis, natural language processing, representation learning

Introduction

Natural language processing (NLP) is an important application of artificial intelligence that aims to study human language with computing systems. Recently with the rise of machine learning, research on NLP has been widely done with machine learning models and algorithms.

However, machine learning models suppose often that their inputs, also known as features, are continuous values in a vector space. To effectively study languages with these models, much attention has been drawn to a non-trivial problem: how to represent a text sequence, which is fundamentally discrete and order-gnostic, as a vector?

In this work, we compare three benchmark approaches:

1. Bag-of-words: for a given vocabulary, we add one-hot encodings of all word occurrences in a sentence;
2. Average pooling: we take the GloVe embeddings for every word in a sequence, and take their average as the final representation;
3. Pre-trained language model: we take a pre-trained BERT model and take the [CLS] encoding of a sentence as its representation.

To compare them empirically, we use sentiment analysis, a standard NLP task, to test the influence of different representation methods on the final classification performance. We use linear support vector machines (SVMs), which are a popular model choice for binary classification.

Background

Word embeddings

To represent a word as a vector, a traditional method was to use one-hot encodings: we prepare a vector whose dimension is equal to the size of the vocabulary, and allocate each dimension to each word; for a certain word, its dimension value is 1 while others are 0.

This intuitive method, however, shows two significant drawbacks:

Every word is treated equally, and no semantic information is considered: for example, the word “computer” is closer in meaning to “compute” than to “frog”, but one-hot encoding would treat them as being “equally different”;

One-hot encoding results in high-dimensional vectors (typically dim>100k for English). A large number of features not only slow down drastically the training stage but also would cause sparsity in data and therefore brings difficulty in parameter estimation. In literature, this phenomenon is called “the curse of dimensionality”.

To cope with these problems, [1] proposed a low-dimensional, dense representation of words known as distributional representations or word embeddings. We represent words using relatively low-dimensional vectors, with the principle that ideally, similar words have similar embeddings in the vector space (their cosine similarity is close to one).

The authors trained the neural model along with the downstream task (language modeling) to obtain these word embeddings. Later in [2], [3], [4] (GloVe), with the idea that semantic information generally shares across different language tasks, pre-trained word vectors were proposed: by initializing word vectors in a new task with these pre-trained vectors, we can hugely enhance the final performance. Note that usually, we retrain these word vectors as well.

Representation learning of sentences

Above the problem of representing a word, in natural language processing, it is equally important to determine how to use a vector to represent a sentence, which is an ordered combination of words. Commons practices are:

Bag-of-words (BOW): we neglect all order information in a sequence and see it as a set of occurrences of different words, and we use a pooling layer to extract information from it (common choices are average pooling (equivalent to adding when scaled) and maximum pooling);

Use order-aware models to calculate the representation for a sentence, e.g., RNN/LSTM [5] models or Transformer [6]-based models. To deal with order information, the former uses a recurrent structure that feeds the output of step t-1 into the model in step t, and the latter uses position embeddings, which are continuous learnable vectors representing the position of the token in the sequence.

Pre-trained language models

Language modeling is the de facto NLP task for language generation. For every possible sequence, we assign a value to it, indicating the probability that it occurs in human language.

The most intuitive way to do this is to count the frequency of n-grams, which are a specific sequence of words. However, just like one-hot encodings, sparsity (number of n-grams grows exponentially with n) and loss of semantic information are two significant drawbacks. Later, with the rise of Deep Learning, people have begun to use neural network models to compute probabilities for input sequences [1].

To generate text from a language model, the most common practice is argmax decoding: we begin with an empty sequence, and we find the most probable word that comes first. We iterate this process, each time finding the most probable word given all the precedents. Apart from this purely greedy method, other methods like beam search or sampling are also used widely.

Language, complicated in its nature, is very hard to model. To model natural language more precisely, recent work has proposed to use very large Transformer-based neural network models that often have millions of parameters or even more. This brings, of course, the problem of how to estimate these parameters using limited supervised data for each task, as the scarcity of data causes severe over-fitting. An elegant way to solve this is to adopt a pre-training + fine-tuning paradigm: language models are first pre-trained are large corpora of unsupervised language data, using generation or reconstruction objectives. Then these models are fine-tuned on a much smaller supervised dataset to solve the final task. Popular pre-trained models of this kind contain for example GPT [7], BERT [8], and BART [9]. Theoretical research [10] has shown that unsupervised pre-training can be viewed as a process of regularization of the model.

We discuss BERT in some detail. BERT is an encoder model of the Transformer architecture that is pre-trained to reconstruct some corrupted text. To do a forward pass, we first add two special tokens, [CLS] at the beginning and [SEP] at the end. Then we take embeddings of every token, and after interactions between them, BERT finally returns the same number of vectors. The corresponding output vector of [CLS] is what we focus on in this work.

Sentiment analysis

Sentiment analysis is a standard NLP task aiming to identify a text sequence's implied sentiment. Normally it is expressed as a binary classification problem that determines whether a sentence has positive or negative sentiment, based on features extracted from input text data.

Support Vector Machines

Support vector machines (SVMs) are a category of machine learning models that can be applied both to classification and regression. They are discriminant [11] models that use a separating hyperplane to decide the corresponding class of new data. SVMs are normally linear models, but with the usage of kernel tricks, we can use them to deal with linearly inseparable data. In this work, we use linear SVMs as they scale better when we have a large dataset.

Experiments

Setup

We use *sklearn*’s implementation of linear SVMs as our model. As SVMs are sensible to the norm of input vectors, we normalize all input vectors to norm 1 before feeding them into the model.

For the dataset, we use the IMDB dataset [12] which contains 25000 training samples and 25000 testing samples. These samples are movie reviews from the IMDB website, along with annotated sentiment labels. We download the dataset from Hugging Face, which is a popular NLP platform that provides an API to easily download pre-trained models and datasets.

The BOW approach is the hardest to implement efficiently with SVMs, as the resulting vocabulary can be too big to fit into the RAM. To cope with this problem, we restrict the maximum vector size to 10000 which attains a balance between efficiency and completeness.

For the GloVe + average pooling method, we download the word vectors from GloVe’s official website and choose the 50-dimensional vectors pre-trained on Wikipedia 2014 + Gigaword 5. We then use the Gensim package to read these files and use SpaCy to tokenize our input to match words with their corresponding word embeddings. We use zero vector to represent unrecognized words. We do an extra experiment to test whether average pooling outperforms maximum pooling.

For the pre-trained language model approach, we use the DistilBERT model from Hugging Face, which is a lighter version of the original BERT model. We tokenize the inputs with the model’s tokenizer and feed them into the model. Then, as BERT models follow a seq2seq architecture, we take the output of [CLS] token as the representation of the whole sequence.

Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Approach | BOW | GloVe + avg | GloVe + max | BERT |
| precision | 0.811 | 0.751 | 0.655 | 0.854 |

We obtain the highest classification precision with the pre-trained language model approach, which is expected as BERT takes into account both semantics (dense representation) and order (self-attention mechanism) information. As for efficiency, BERT is the most resource-demanding because although the output dimension of the model is 768 (a lot less than 10000 in BOW) which alleviates the training burden, the feature extraction process requires a huge number of calculations as it involves a forward pass of the BERT model. To extract features for all data instances in the IMDB dataset, we used a T4 GPU and the speed was 2500 samples/min.

For the GloVe + pooling approach, we tried two different pooling methods: average pooling and maximum pooling, which are different ways to choose a value for each input dimension given several instances. We see that average pooling performs significantly better than maximum pooling, and we hypothesize that this is because maximum pooling only captures the maximum value for every dimension, and results in a set of representations that is hardly linearly separable. Its performance drop compared to BERT can be explained by the loss of order information. As for efficiency, contrary to intuition, 50-dimensional GloVe vectors aren’t less time-consuming compared to the BOW approach as the feature extraction step would require a table lookup in a huge dictionary (400K in our case). Our experiment ran at a processing speed of 1000 samples/min, which is even slower than using the BERT model and a consumer-level GPU.

The BOW approach gave a reasonable precision of over 0.8 and was easy to implement. By restraining the vocabulary size at 10000, we managed to avoid RAM explosion and the training process was done in 5 minutes, which was the slowest of all three approaches. However, tokenization took hardly any time and when considering the sum of training time and pre-processing (feature extraction) time, it was the fastest of the three.

**Conclusion**

In this work, we illustrated three methods for sentence representation and compared their training/inference efficiency and embedding correctness (similar sequences have similar representations) using a traditional sentiment analysis pipeline.

We didn’t cover every sentence representation method proposed in the literature, e.g., sentenceBERT [13] is another insightful method that fine-tunes a BERT model using a training task of text entailment. These methods can be further explored in future work.

**References**

[1] Bengio Y, Ducharme R, Vincent P. A neural probabilistic language model. Advances in neural information processing systems. 2000;13.

[2] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. 2013 Jan 16.

[3] Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems. 2013;26.

[4] Pennington J, Socher R, Manning CD. Glove: Global vectors for word representation. InProceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) 2014 Oct (pp. 1532-1543).

[5] Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997 Nov 15;9(8):1735-80.

[6] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. Advances in neural information processing systems. 2017;30.

[7] Devlin J, Chang MW, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. 2018 Oct 11.

[8] Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training.

[9] Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461. 2019 Oct 29.

[10] Erhan D, Courville A, Bengio Y, Vincent P. Why does unsupervised pre-training help deep learning?. InProceedings of the thirteenth international conference on artificial intelligence and statistics 2010 Mar 31 (pp. 201-208). JMLR Workshop and Conference Proceedings.

[11] Bishop CM, Nasrabadi NM. Pattern recognition and machine learning. New York: springer; 2006 Aug 17.

[12] Maas A, Daly RE, Pham PT, Huang D, Ng AY, Potts C. Learning word vectors for sentiment analysis. InProceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies 2011 Jun (pp. 142-150).

[13] Reimers N, Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084. 2019 Aug 27.