Comparison of different sentiment analysis algorithms on reviews' datasets

Zeen Wang 21-741-038

October 27, 2023

https://github.com/Zion-W9/Sentiment_analysis

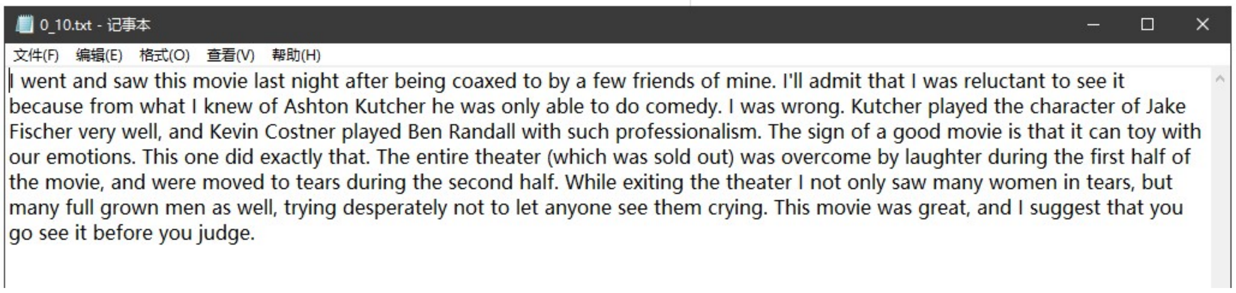
1 Introduction

Sentiment Analysis, also called opinion mining, stands out as a pivotal component in Natural Language Processing (NLP). Sentiment analysis main job is to mine insights from user-generated opinions. By evaluating text, audio, or visual content, it captures and gauges the prevailing sentiments, attitudes, and emotions expressed by individuals. In this paper, I want to compare and contrast several of the more commonly used sentiment analysis algorithms.

2 Data set

I choose the "Large Scale Movie Review Dataset" [3], more commonly referred to as the IMDb Dataset. Specifically designed for sentiment analysis endeavors, it encompasses movie reviews directly from the IMDb platform, thereby offering a genuine snapshot of user-generated content. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. It provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing.

For illustration, a positive example is provided below:



I went and saw this movie last night after being coaxed to by a few friends of mine. I'll admit that I was reluctant to see it because from what I knew of Ashton Kutcher he was only able to do comedy. I was wrong. Kutcher played the character of Jake Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The sign of a good movie is that it can toy with our emotions. This one did exactly that. The entire theater (which was sold out) was overcome by laughter during the first half of the movie, and were moved to tears during the second half. While exiting the theater I not only saw many women in tears, but many full grown men as well, trying desperately not to let anyone see them crying. This movie was great, and I suggest that you go see it before you judge.

Figure 1: Positive example

Each review is text-based, paired with metadata that signifies its sentiment (positive or negative).

3 Preprocessing

In the domin of sentiment analysis, preprocessing is important to ensure the data is aptly structured for subsequent machine learning tasks. At the core of this preprocessing is TorchText https://pytorch.org/text/stable/index.html, a PyTorch library designed for natural language processing. One of the principal components leveraged from TorchText is the ‘Field‘, which delineates how data should be processed. For our reviews, the ‘TEXT‘ field was defined, utilizing the spaCy tokenizer, specifically the ‘en_core_web_sm‘ model https://spacy.io/models/en, to split raw texts into tokens. This choice reflects spaCy’s reputation for precise tokenization. Contrarily, the sentiment, which can be either positive or negative, was processed by the ‘LABEL‘ field, a subclass of ‘Field‘ in TorchText optimized for categorical data.

The IMDb dataset was then automatically fetched using TorchText’s utilities and partitioned into training and test sets. Given that IMDb data inherently lacks a validation split, a portion of the training data was carved out to form a validation. It’s a common wy to hold out a validation set to fine-tune models without touching the test data, ensuring unbiased evaluation later.

A critical phase in our preprocessing is vocabulary construction. Vocabulary, in this context, is a mapping from tokens to unique integers, pivotal for converting textual data into numerical form that neural networks can process. Due to computational constraints and to expedite training, the vocabulary was truncated to the top 25,000 most frequent tokens from the training set. Tokens outside this vocabulary are represented using a special placeholder: the ¡unk¿ or unknown token. Additionally, to handle variable-length sentences within the same batch during training, a padding token, ¡pad¿, was introduced.

To ensure streamlined data flow during model training, we leveraged the ‘BucketIterator‘ from TorchText. This iterator returns batches of sentences, where sentences within a batch are of comparable lengths, minimizing the need for excessive padding and optimizing computational efficiency. These batches were configured for GPU processing on my pc.

4 Model training

4.1 convolutional neural network

The Convolutional Neural Network (CNN) has been a prevalent machine learning model. Distinctive for its nonlinear characteristics and its capacity for regional learning embedding, a typical CNN comprises an embedding layer, a convolution layer, a pooling layer, and an output layer. Within the embedding layer, text is embedded at the word level, forming a matrix representation. The convolutional layer, with a filter width fixed at the word vector’s dimension, is designed to capture relationships between adjacent words. Subsequently, the pooling layer employs a max-over-time pooling operation to extract the maximum value from each feature map. The output layer extracts features, and the fully connected layer establishes a probability distribution over the outputs.

In certain configurations, a deep CNN might encompass an input layer, dual convolutional layers, two max pooling layers, culminating in a fully connected layer processed by a Softmax classifier. ZHANG et al. [5] indicated that character-based deep CNNs excel in text classification.

4.1.1 Model Architecture

Our model architecture encompasses the following principal components:

- **Embedding Layer**: This layer is used for representing discrete tokens as continuous dense vectors. It is initialized with a size of (vocab_size, embedding_dim). A padding index, denoted as *pad_idx*, ensures uniform sequence lengths.
- **Convolutional Layers**: The model employs multiple convolutional layers tailored for varying filter sizes. This diversity allows the detection of different n-gram sizes. Post-convolution, a ReLU activation function is applied.

- **Max-pooling Layers**: Following each convolution, max-pooling is executed. This reduces dimensionality and retains only the salient features.
- **Dropout Layer**: Prior to the fully connected layer, a dropout is applied on the concatenated pooled features, ensuring regularization.
- **Fully Connected Layer**: The features are then processed through this layer to produce the classification logits.

Hyperparameters
The model's primary hyperparameters include:

- **Vocabulary size** (*vocab_size*): Dictates the vocabulary's extent.
- **Embedding dimensions** (*embedding_dim*): Denotes the dense vector's size for each token.
- **Number of filters** (*n_filters*): The count of filters in each convolutional layer.
- **Filter sizes** (*filter_sizes*): Represents the varying filter dimensions in the convolutional layers. Our model employs sizes: 3, 4, and 5.
- **Output dimensions** (*output_dim*): Represents the output's size or the classification classes count.
- **Dropout rate** (*dropout*): Specifies the fraction to be dropped during the dropout operation.

4.2 Recurrent Neural Networks

Compared with Convolutional Neural Networks (CNNs), the Recurrent Neural Network (RNN) model possesses two significant characteristics. First, while CNNs have different parameters in each layer, RNNs share the same parameters across all layers. In RNNs, each stage's output depends on the previous stage, which necessitates a considerable amount of memory. Consequently, RNNs hold an advantage over CNNs in processing sequential information. RNNs can leverage this property to map a sequence of arbitrary length to a fixed-length vector.

However, during the backpropagation process of a simple RNN, training encounters several challenges, primarily the vanishing gradient problem, where the gradient value approaches zero, and the exploding gradient problem, which may lead to instability in the learning process.

4.2.1 Model Architecture

I adopt a Recurrent Neural Network (RNN) model, specifically utilizing the Long Short-Term Memory (LSTM) cells, ideal for handling sequence data such as text.

The LSTM model comprises the following key components:

- **Embedding Layer**: This layer transforms discrete tokens into continuous vectors of size *embedding_dim*. It is initialized with a shape of $(\text{vocab\_size}, \text{embedding\_dim})$. To accommodate consistent sequence lengths, a padding index, *pad_idx*, is used.
- **LSTM Layer**: The core of our model. The LSTM processes sequences while adeptly handling long-term dependencies. Our implementation leverages multiple layers and supports bidirectional processing. This means the model processes the text from start to end and vice versa, enhancing the feature extraction process.

- **Dropout Layer**: Applied post-embedding and before feeding the sequence into the LSTM. Also, utilized after retrieving the hidden states from the LSTM. This layer randomly sets a fraction of input units to 0 at each update, aiding in regularization.
- **Fully Connected Layer**: After processing through the LSTM, the hidden states (from both the forward and backward LSTMs in the final layer) are concatenated and passed through this layer to produce the classification logits.

Hyperparameters
The model is parameterized by the following hyperparameters:

- **Vocabulary size** (*vocab_size*): Defines the extent of the vocabulary.
- **Embedding dimensions** (*embedding_dim*): Dictates the size of the dense vector for each token.
- **Hidden dimensions** (*hidden_dim*): Specifies the size of the hidden states in the LSTM.
- **Number of layers** (*n_layers*): The count of LSTM layers stacked together.
- **Bidirectionality** (*bidirectional*): Indicates if the LSTM processes the sequences in both forward and backward directions.
- **Dropout rate** (*dropout*): Represents the fraction of units to be dropped during the dropout operation.
- **Output dimensions** (*output_dim*): Specifies the size of the output.

4.3 Dictionary-based approaches

Utilizing a dictionary-based method to gather sentiment words is a good choice, primarily because many dictionaries, such as WordNet [1], provide synonyms and antonyms for each entry. Here's how the method unfolds:

1. Begin with a small collection of sentiment words (seeds) that have a known positive or negative connotation. This initial collection is easily curated manually. 2. The algorithm then expands this set by searching for synonyms and antonyms of these seed words in WordNet or other online dictionaries. 3. Words discovered in the previous step are incorporated into the seed list. 4. The process iterates, and with each cycle, the seed list grows. 5. The iteration concludes when no additional words can be identified. 6. Upon completion, a manual review is conducted to refine and finalize the list.

Here I directly use the VADER (Valence Aware Dictionary and sEntiment Reasoner)[2] model for evaluation. VADER contains over 7500 lexical features, including slang, acronyms, and emoticons commonly found in social media text.

4.4 Large language model

The large language model is an artificial intelligence model designed to understand and generate human language. They are trained on large amounts of text data and can perform a wide range of tasks, including text summarization, translation, sentiment analysis, and more. LLM are characterized by their large scale, containing billions of parameters, helping them learn complex patterns in linguistic data.

I use the "distilbert-base-uncased-finetuned-sst-2-english" model`https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english`, which is based on DistilBERT[4]. DistilBERT is a distilled version of BERT, which is a popular transformer-based model for natural language processing tasks. The idea behind distillation is to train a smaller model (the "student") to mimic the behavior of a larger, more complex model (the "teacher"). In the case of DistilBERT, the student model is trained to approximate the outputs of the BERT model. DistilBERT offers a compelling balance between performance and efficiency.

5 Evaluation

Since sentiment analysis is a classification task, we use accuracy, recall, and precision as evaluation methods. The results are as follows:

	CNN	RNN	Dictionary-based	LLM
Accuracy	0.85	0.62	0.69	0.88
Precision	0.83	0.75	0.72	0.92
Recall	0.88	0.60	0.69	0.84

The Convolutional Neural Network (CNN) model exhibited a strong performance with an accuracy of 0.85. Its precision and recall values, at 0.83 and 0.88 respectively, suggest that the model was able to correctly identify a significant portion of the sentiments while also minimizing false positives.

The Recurrent Neural Network (RNN) model, on the other hand, had a lower accuracy of 0.62. Despite its decent precision of 0.75, its recall of 0.60 indicates that it might have missed a considerable number of positive instances.

The Dictionary-based approach, which is inherently different from the neural network models, achieved an accuracy of 0.69. Its precision and recall, both hovering around the 0.70 mark, suggest a balanced performance but with room for improvement.

Lastly, the Large Language Model (LLM) outperformed the other models in terms of accuracy and precision, achieving values of 0.88 and 0.92 respectively. However, its recall of 0.84, while still commendable, indicates that there were some instances it failed to capture.

In summary, while all models show their strengths and weaknesses, the LLM emerged as the top performer in this experiment. Its high accuracy and precision underscore the potential of large-scale neural models in sentiment analysis tasks. However, it's essential to consider the specific requirements and constraints of any application before selecting an appropriate model. The Dictionary-based approach, for instance, offers simplicity and might be more suitable for applications where interpretability is crucial. On the other hand, for tasks demanding high accuracy, neural network-based models, especially the LLM, seem to be the more promising choice.

6 Future work

Given the ever-expanding volume of text data, it's anticipated that large language models will dominate the future landscape of natural language processing. Observing the evolution of these methods suggests several focal points for future research in text sentiment analysis:

1. **Complex Sentence Analysis:** There's a growing need to refine sentiment analysis for intricate sentences. As internet jargon, idioms, and phrases with emotional connotations become increasingly prevalent, challenges arise, especially when the text includes ironic or metaphorical language. Detecting emotional polarity in such contexts demands further investigation.

2. **Fine-grained Sentiment Analysis:** While much of the current research revolves around binary sentiment analysis, there's a burgeoning interest in multi-class sentiment categorization. Delving deeper into nuanced sentiment analysis will likely be a focal point in upcoming research.

3. **Optimization of Pre-training Models:** Pre-training models are currently a hotbed of research. They address many limitations of traditional methods, such as non-parallelizable computations, and adeptly discern word relationships. Fine-tuning these models can yield superior results in downstream tasks. However, challenges like extensive model parameters and prolonged training durations persist. Exploring ways to maintain classification accuracy with fewer parameters and reduced training times is a promising research direction.

References

[1] Christiane Fellbaum. “WordNet”. In: *Theory and applications of ontology: computer applications*. Springer, 2010, pp. 231–243.

[2] Clayton Hutto and Eric Gilbert. “Vader: A parsimonious rule-based model for sentiment analysis of social media text”. In: *Proceedings of the international AAAI conference on web and social media*. Vol. 8. 1. 2014, pp. 216–225.

[3] Andrew L. Maas et al. “Learning Word Vectors for Sentiment Analysis”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[4] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv preprint arXiv:1910.01108* (2019).

[5] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems* 28 (2015).