A Comparative Study of Machine Learning Algorithms for News Classification

MSML 641

Project members: Xiyuan Liu, Xin Li, Gan Wang

1. Abstract

In this report, we present a comparative study of machine learning algorithms for news classification. We evaluated the performance of several popular classification algorithms, including Support Vector Machines (SVM), LSTM and BERT, on 7600 samples and trained the models with random 16000 samples from ag_news which consisted of 120000 news articles from 4 different categories, including world, sports, business , and technology. To compare the impact of different feature extraction techniques, we used a range of feature extraction techniques, such as one-hot encoding and encoding using Keras to represent the text data. We evaluated the models using various evaluation metrics, including accuracy, precision, recall, F1-score and accuracy score.

Our experimental results showed that LSTM with pretrained embedding layers from Glove outperformed the other algorithms in terms of accuracy and training time. We also conducted an ablation study to understand the impact of various architectures of embedding layers on the classification performance. Our findings suggest that the choice of feature extraction technique and embedding architecture plays a crucial role in the performance of the classification models. Overall, our study provides insights into the performance of machine learning algorithms for news classification and can serve as a reference for researchers working in this area.

2. Research description

The research problem addressed in this project is the efficient classification of news articles into different topics or classes using machine learning algorithms. The task of news classification involves analyzing the text of news articles and assigning them to appropriate categories, such as world, sports, business , and technology. There are many challenges associated with news classification, including the large volume of data, the variability in writing styles, and the need to accurately capture the nuances of language and meaning. This project aims to explore different approaches to news classification and identify the most effective models. Additionally, the project aims to investigate the factors that can enhance the performance of these models, such as the use of different feature extraction techniques, the impact of various model architectures, and the influence of hyperparameter settings.

3. Dataset selection and Data preprocessing

3.1 Database

The ag_news dataset provided by Hugging Face [10] is a large collection of news articles that have been manually categorized into four classes: World, Sports, Business, and Science/Technology. The dataset is composed of 120,000 articles in the training set and 7600 articles in the testing set. Each class is evenly distributed in both the training set (each class containing 30000 articles) and the testing set (each class containing 1900 articles). The articles were collected from more than 2000 news sources by ComeToMyHead in more than one year of activity. Each article is represented by a tuple containing two fields: the raw text of the news article and a corresponding label indicating the class of the article. The labels are represented as integers with a one-to-one mapping to the four classes mentioned above.

Overall, ag_news dataset has been widely adopted as a standard benchmark for text classification tasks. Many state-of-the-art machine learning models have been evaluated on this dataset.

3.2 Data preprocessing:

Data preprocessing consisted of two main steps. Firstly, a subset of 4000 unique samples was randomly selected from each class to construct the training set. The decision to use a smaller subset instead of the entire 120,000 samples in the training set was based on the research objective to evaluate the models and investigate the impact of different feature extraction techniques and embedding architectures on classification performance. It was determined that 16000 samples would provide a sufficient sample size to achieve this objective. By using this smaller subset of samples, we can focus on evaluating the quality of the features extracted and the impact of different models without being limited by the computational resources required to process a larger dataset. For the testing dataset, the original testing dataset provided by ag_news was used without any additional adjustments. This ensured that our models were evaluated on a realistic and representative test set.

The next step was text cleaning which involved removing stop words, punctuation, and special characters. Additional text cleaning techniques also were applied such as stemming and lemmatization to reduce the dimensionality of the data.

4. Feature Extraction

In this research, one-hot encoding and encoding using Kersa had been used as Feature extraction methods.

4.1 One hot encoding:

One-hot encoding is a binary representation technique used to convert categorical variables into a numerical format. This technique generates a binary vector for each category in the variable, where the length of the vector is equivalent to the total number of tokens in a dictionary [1]. For this project, a token dictionary was created by selecting the top 100 frequent tokens (this step was processed after data preprocessing) that appeared in both the training and testing sets. The resulting binary vector contains a 1 in the position corresponding to the category it represents and 0 in all other positions. One-hot encoding enables machine learning algorithms to process categorical data and is an essential technique in natural language processing.

4.2 encoding using Keras:

The encoding process was also implemented using Keras library functions [2]. A tokenizer class was created from the library with a parameter 'num_words' set to 20,000, which specified that the tokenizer should only consider the 20,000 most frequent words in all the given corpus. The tokenizer was then fitted on the text data from both the training and testing sets to update the internal vocabulary of the tokenizer based on the frequency of words in the data. The 'texts_to_sequences' method was then used to convert the text data into a sequence of integers, where each word was replaced with its corresponding integer index in the tokenizer's vocabulary. Finally, the 'pad_sequences' method was used to ensure that all input sequences had a length of 100, which is the maximum length of a sequence that will be accepted by the models used in this project.

5. Baseline : Support vector machine

Support Vector Machines (SVM) is a well-established and widely used machine learning algorithm for classification tasks. The objective of SVM is to find an optimal hyperplane that effectively separates data points from different classes in the feature space. This hyperplane is selected to maximize the margin, which represents the distance between the hyperplane and the closest data points of each class [3]. By maximizing the margin, SVM aims to enhance the model's generalization ability and robustness when dealing with unseen data.

In this project, a linear support vector classifier with calibration was chosen as the baseline model with one-hot encoding as feature extraction technique. Although news classification can be a complex task with potentially non-linear separability, a linear support vector classifier is still a suitable choice for establishing a performance benchmark. The goal is to provide a reference point against which the effectiveness of alternative models and techniques can be assessed.

To implement the linear support vector classifier, the LinearSVC classifier from the scikit-learn library was utilized. This class implements a support vector machine with a linear kernel specifically designed for classification tasks. Additionally, calibration of the decision function was performed using the CalibratedClassifierCV() function from the same library. This calibration step improves the reliability of the predicted probabilities. Once the model was constructed, it was trained on the training set using the fit() method, and the predict() method was employed to obtain predicted labels for the testing set.

6. Machine learning approaches

6.1 LSTM

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture designed to address the challenges of capturing and retaining long-term dependencies in sequential data. It overcomes the vanishing gradient problem commonly encountered in traditional RNNs, where gradients diminish exponentially during backpropagation through time steps. LSTM is capable of preserving relevant information over longer sequences, making it suitable for tasks involving sequential data [4].

In this project, an LSTM model was selected as the initial approach. The model was constructed using the Keras Sequential API and comprised several layers. The first layer was an Embedding layer, which took the size of the total unique vocabulary as the input dimension and 128 as the output dimension. This layer learned and mapped words to fixed-size dense vectors. Next, an LSTM layer with 25 units was added to capture the sequential patterns and dependencies within the text data. The LSTM layer was followed by a GlobalMaxPool1D layer, which extracted the maximum value over the temporal dimension (sequence length) to simplify the input for subsequent layers. To prevent overfitting, a Dropout layer with a rate of 0.2 was included. This layer randomly set input units to 0 during each training update, reducing interdependencies among units and enhancing model generalization. Two dense layers were then added. The first dense layer had 50 units with the rectified linear unit (ReLU) activation function, introducing non-linearity and enabling the model to learn complex patterns. The second dense layer had 4

units, corresponding to the 4 classes in the classification task, and utilized the softmax activation function to generate class probabilities.

By combining the Embedding layer, LSTM layer, and dense layers, the model effectively captured sequential dependencies and learned representations for text classification. The choice of LSTM as the initial approach allowed for the exploration of the model's performance and paved the way for evaluating alternative architectures and techniques.

6.2 LSTM-Large

If a neural network has more layers and neurons, it generally has the potential to learn more complex patterns and representations from the data. Increasing the depth (number of layers) and width (number of neurons) of a neural network can provide it with more capacity to capture intricate relationships and extract higher-level features.

In this study, we developed the LSTM-Large model as a second approach, which exhibits a more complex architecture compared to the simpler LSTM model. The LSTM-Large model incorporates bidirectional LSTM and GRU layers, each consisting of 128 units, along with additional layers such as a bidirectional LSTM layer and an extra LSTM layer. This increased complexity allows the LSTM-Large model to capture more detailed sequential dependencies and potentially discover intricate patterns in the text data. By incorporating more layers and neurons, the LSTM-Large model offers the potential for improved performance in text classification tasks.

6.3 Word2vec_doc

In contrast to the previous LSTM model, our model initializes the embedding layer with random values and trains it from scratch during the training process. This approach enables the embedding layer to learn a high-dimensional vector representation of each input token, capturing both the semantic and syntactic properties of the context of the given set. To achieve this, we utilized the Word2Vec embedding method, which involves pre-trained word embeddings generated using the Word2Vec algorithm.

When Word2Vec is used in conjunction with LSTM, the model can leverage the semantic relationships between words that are captured by Word2Vec, leading to improved performance on downstream NLP tasks. Word2Vec is a technique used to learn distributed representations of words, also known as word embeddings. These embeddings are dense vector representations of words in a

high-dimensional space, where words with similar meanings are closer to each other. Word2Vec accomplishes this by training a neural network to predict context words given a target word or vice versa.

We developed our own Word2Vec from scratch, using the entire training set from the dataset, which contained 120,000 samples, instead of a small randomly selected subset. Notably, Word2Vec has two different models, namely skip-gram and CBOW. In our Word2Vec model, we employed skip-gram rather than CBOW because the former excels at capturing the nuances and associations of rare words.

In CBOW, the model predicts the target word given a context of surrounding words. This method works well when context words are informative and distinct, but may not be effective for rare or infrequent words that appear infrequently in the training data. Conversely, skip-gram predicts the context words given a target word, allowing the model to learn more about the meaning and usage of the word by considering multiple contexts in which it appears. By capturing different nuances and associations, skip-gram more accurately represents rare words in the embedding space. In summary, skip-gram is better for capturing rare words and larger datasets, while CBOW is faster to train and more suitable for smaller datasets.

Through our project, we gained three insights into Word2Vec. First, compared to normal embedding methods, Word2Vec generalizes better. Second, it reduces the time required to train an LSTM model as embeddings are pre-computed and can be loaded directly into the model. Third, Word2Vec captures word relationships and patterns that are specific to our domain or task. Additionally, we had more control over hyperparameters and the training process, which improved our model's performance on our tasks.

6.4 Word2vec_glove

To explore the performance of pre-trained word embeddings, we opted to use GloVe: Global Vectors for Word Representation rather than writing our own word2vec from scratch. Pre-trained embeddings are often trained on larger and more diverse corpora of text data than what is typically available for individual projects, enabling them to capture a broader range of semantic relationships between words and potentially improve model performance.

One of the benefits of using pre-trained word embeddings is their consistency and reproducibility. They are standardized and publicly available, providing a reliable baseline for comparing NLP models across different projects and domains. GloVe, in particular, is an unsupervised learning algorithm that uses

global word-word co-occurrence statistics to generate 300-dimensional vector representations for 1.9 million words across a corpus of 42 billion tokens.

By generating the embedding matrix directly from GloVe, we were able to bypass the need for additional neural network models, which can be computationally expensive, particularly for larger corpora. This allowed us to save significant computation time and resources, while still leveraging the benefits of pre-trained embeddings.

6.5 Transformer (BERT)

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing (NLP) model developed by Google. It is based on the Transformer architecture, which introduces attention mechanisms for capturing contextual relationships in text data. BERT is pre-trained on a large corpus of unlabeled text data using masked language modeling and next sentence prediction tasks, allowing it to learn rich representations of words and sentences [5].

Compared to LSTM-based models, BERT offers several significant advantages. First, BERT incorporates attention mechanisms, which enable it to capture dependencies between words in both directions and consider the entire context. Second, BERT utilizes a mask language mechanism, allowing it to handle bidirectional context and further enhance its understanding of language. Additionally, BERT employs multiple embedding methods, including token, segment, and position embeddings, which contribute to its comprehensive understanding of the input data [5].

Improving the performance of machine learning models often depends on the input perspectives provided to the model. In this regard, BERT offers a broader range of perspectives compared to LSTM-based models, providing it with a higher capability to effectively process the News classification task.

7. Performance evaluations

	Precision				Recall			
Model	Label 0	Label 1	Label 2	Label 3	Label 0	Label 1	Label 2	Label 3
SVM	0.68	0.71	0.75	0.52	0.65	0.66	0.55	0.73
LSTM	0.88	0.92	0.85	0.85	0.87	0.96	0.82	0.85
LSTM_large	0.88	0.94	0.82	0.82	0.85	0.93	0.81	0.86
Word2vec_doc	0.85	0.9	0.85	0.85	0.88	0.96	0.78	0.84
Word2vec_glove	0.85	0.95	0.87	0.85	0.88	0.97	0.81	0.86
BERT	0.91	0.89	0.98	0.92	0.88	0.91	0.98	0.94

	F1 score						AVERAGE ACC	0.84296
Model	Label 0	Label 1	Label 2	Label 3	Accuracy score	Training time(s)	Effective Training Time	
SVM	0.67	0.69	0.63	0.61	0.64711	4	200	
LSTM	0.88	0.94	0.83	0.85	0.87354	63	63	
LSTM_large	0.87	0.94	0.82	0.84	0.86308	44	44	
Word2vec_doc	0.86	0.93	0.82	0.84	0.86374	43	43	
Word2vec_glove	0.87	0.96	0.84	0.86	0.8803	27	27	
BERT	0.9	0.9	0.98	0.93	0.93	600	600	

Based on the charts above, we can see that each model performs differently across the different labels and evaluation metrics. We use yellow and cyan to represent the maximum number and minimum number in each model. And we use pink and green colors to represent the worst and best performance in terms of accuracy and effective training time.

The performance of SVM is consistently poor for Label 3 based on the F1 score, while it demonstrates strong performance for Label 1, a trend that is similarly observed in all models except for

BERT. Notably, LSTM, LSTM_large, Word2vec_doc, and Word2vec_glove all exhibit a common performance pattern of achieving high F1 score for Label 1 and poor performance for Label 2. Conversely, BERT demonstrates a divergent performance pattern, with strong performance for Label 2 and weaker performance for Label 1.

The aim of this project was to compare the performance of several models on news classification tasks in terms of accuracy and training time. The baseline model, SVM, was found to have an accuracy rate of 64.7% and the shortest training time of 4 seconds.

BERT achieved the highest accuracy rate of 93%, but its training time of 600 seconds cannot be ignored. Among the four LSTM-based models, including LSTM, LSTM_large, Word2vec_doc, and word2vec_glove, the accuracy range was between 86% and 88%, with a maximum gap of only 2%. However, the difference between their training times was significant, with a range of up to 36 seconds.

In order to compare all models on the same page, we add a new parameter called Effective Training Time to evaluate all seven models. The formula is shown below:

$$Effective\ Training\ Time = Training\ times + (average\ accuracy - model\ accuracy) * 1000$$

We set the average accuracy as our target accuracy, any models that have lower accuracy than the average will take a penalty, and the original training time will add 100 times the difference between the average accuracy and true accuracy. Overall, word2vec_glove is our best model in this project.

8. Conclusion

According to our current assessment, the word2vec_glove model yields the most optimal results. Incorporating pretrained embeddings has proven to be a valuable asset in enhancing model performance, underscoring the significance of pretrained materials. For optimal results, we recommend using Skip-gram if seeking better performance, or CBOW if prioritizing faster training time.

Although BERT is a highly robust model, its cost of training is prohibitively expensive, even with fine-tuning a pretrained-bert model. Alternatives like Colbert or Tinybert may prove to be more cost-effective. It's important to note that increasing the complexity of the model does not necessarily equate to an improvement in its performance.

9. Limitations

9.1 dataset limitation

During our experiment, we observed that the performance of the LSTM_large and Word2vec_glove models was relatively similar. Despite having the same architecture, it was expected that Word2vec_glove would outperform LSTM_large due to its embedding being trained on a much larger corpus, whereas LSTM_large had a randomly assigned embedding layer. Notably, LSTM_large was trained on a relatively small dataset of 16,000 samples, making it unlikely for the model to train a generalized embedding layer.

One possible explanation for this observation relates to the dataset used in our experiment. Both the training and testing sets were generated from the ag_news dataset, which may have contained text of high quality and significant similarity. By high quality, we mean the text had no misspellings, grammar mistakes, or logic errors, which could have aided the model's performance. Moreover, the similarity between the training and testing data might have allowed the model to learn specific patterns unique to the ag_news dataset. However, if the testing set were switched to another source, it is possible that the model's performance would significantly decline.

9.2 computational power limitation

Computation power is an essential factor in machine learning, especially when dealing with complex models. The available computational resources directly influence the size and complexity of the tasks that can be effectively executed. During our experimentation, we conducted training for a fixed number of epochs for different models. We observed that certain models, like Word2Vec_glove, did not exhibit substantial improvements beyond a certain number of epochs, suggesting that their performance plateaued. However, models such as BERT, which require significant computational resources to train, may benefit from additional epochs to continue learning and refining their representations. The decision to limit the number of epochs is often influenced by a trade-off between computational costs and the potential for further improvement in model performance.

9.3 layer selection limits

In this project, we followed a trial-and-error approach by randomly adding or deleting layers to the model architecture. However, we acknowledge that this is not an optimal or systematic way to define the architecture. Designing an effective model architecture requires careful consideration and planning

based on domain knowledge, problem requirements, and previous research. It involves making informed decisions about the number and type of layers, their connectivity, activation functions, and other architectural elements. A well-defined architecture can improve model performance, training efficiency, and interpretability.

9.4 model interpretation

In this project, we aimed to gain insights into the training process and understand how the models were learning to perform the classification task. To achieve this, we attempted to visualize some of the layers, particularly the trained embedding layers from the LSTM and Word2vec_glove models. However, this step proved to be computationally intensive and required substantial computation power. While we were able to generate some visualizations, the interpretation of these visualizations was challenging. The complexity and size of the models, coupled with the limitations in computational resources, made it difficult to fully comprehend the underlying patterns and representations learned by the models.

References

[1]: Brownlee, Jason. “Why One-Hot Encode Data in Machine Learning? - MachineLearningMastery.com.” *Machine Learning Mastery*, 28 July 2017, https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. Accessed 16 May 2023.

[2]: “tf.keras.preprocessing.text.Tokenizer.” TensorFlow, 23 March 2023, https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer. Accessed 16 May 2023.

[3]: Gandhi, Rohith. “Support Vector Machine — Introduction to Machine Learning Algorithms | by Rohith Gandhi.” Towards Data Science, 7 June 2018, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Accessed 16 May 2023.

[4]: Miguel, Tiago. “How the LSTM improves the RNN. Understand the differences between… | by Tiago Miguel.” Towards Data Science, 1 January 2021, https://towardsdatascience.com/how-the-lstm-improves-the-rnn-1ef156b75121. Accessed 16 May 2023.

[5]: Horev, Rani. “BERT Explained: State of the art language model for NLP.” Towards Data Science, 10 November 2018, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270. Accessed 16 May 2023.

[6]: word2vec. “Word2vec.” TensorFlow, 2022, www.tensorflow.org/tutorials/text/word2vec#skip-gram_sampling_table. Accessed 16 May 2023.

[7]: Pennington, Jeffrey. “GloVe: Global Vectors for Word Representation.” Stanford.edu, 2014, nlp.stanford.edu/projects/glove/. Accessed 16 May 2023.

[8]: “Gensim: Topic Modelling for Humans.” Radimrehurek.com, 2016, radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html. Accessed 16 May 2023.

[9]: İrsoy, Ozan, et al. “Corrected CBOW Performs as Well as Skip-Gram.” ArXiv.org, 2020, arxiv.org/abs/2012.15332. Accessed 16 May 2023.

[10]: Batista, Elisa. “ag_news · Datasets at Hugging Face.” Hugging Face, https://huggingface.co/datasets/ag_news. Accessed 16 May 2023.