The Differences Between Various Models in Processing Language

By Jack Brassil

Table of Contents

Introduction & Objective

As a computer scientist, I am interested in the concept of machine learning. Given the increased prevalence of AI in our society, learning how these tools are trained and fine-tuned will prove to be an important skill throughout my career. Given these factors, I decided to learn more about the concepts within natural language processing and train several models to understand the differences between them.

The first model I utilized was a Logistic Regression model. Text is first vectorized, often converted into a “Bag of Words”, meaning that the text is fed into the model without regard for its placement in a given sentence (“Text Classification using Logistic Regression”, n.d.). Logistic Regression is relatively simple, and the model was by far the easiest to implement; however, its less contextual nature theoretically makes its predictions less accurate.

‘ The second model I used was a Long Short-Term Memory (LSTM) model. This one worked better in regards to context — as its name suggests, it can store data in memory (“Long Short-Term Memory (LSTM)”, n.d.). While this works better in theory due to its increased memory capabilities, the LSTM showed some issues and required more computation to train than the Logistic Regression model.

The third and final model was the Bidirectional Encoder Representations from Transformers (BERT). BERT is a transformer model: a model that can learn meaning based on tracking relationships through sequential data, such as the ordering of the words in a sentence (Merritt, 2022). Additionally, transformer models utilize a technique called self-attention, which allows them to weigh the importance of words within the same input (Merritt, 2022). BERT in particular uses a form of training called bidirectional training.

While most models analyze sequences of text in one direction only, BERT looks in both directions to learn from its text. Additionally, BERT is pretrained, with data from Wikipedia and several books (Baheti, 2023).

Overall, these three gave surprising results. However, before analyzing them, I had to feed them data.

The Dataset

I thought about issues that could be solved through machine learning, and pondered where large amounts of data exist in the real world. I decided on news for several reasons: first, news is abundant. News is, in a sense, life itself, leading to a large amount of it being stored. Additionally, news takes several forms, allowing it to easily be divided into different categories. Finally, news is important, so training a model to streamline its delivery or classification quickly becomes an important task. In an ideal world, our models would initially know the difference between different types of news articles. Unfortunately, computers are not people, so I needed to train them on existing data. Before I could train the models, however, I needed a dataset to feed them. I decided to utilize a BBC dataset utilized by Greene and Cunningham (2006), reuploaded to Kaggle in a CSV format by user alfathterry. Essentially, in the table, one column stored a given article while the other stored the type of news article it was. Before training the model, I needed to make sure to tokenize the data so all three models would be able to read it.

Methodology & Process

To train my lovely models, I used Google Colab. Initially, I considered training them locally; however, due to my lack of an NVIDIA GPU, and the fact that Google gives out a free version of Colab Pro to students, I decided to utilize the service to train the model. Initially, I planned on using the Visual Studio Code Colab extension; however, Colab gave me trouble as it couldn't find the file path in my Google Drive. Therefore, I ended up using the online version instead. In the process, however, I forgot that the GPU configuration wouldn't sync over to the new file, and wasted 2 hours training on a CPU. Once I realized my mistake, I changed over to a T4 GPU — not top of the line by any means, but a solid contender that would help with training. As for my data, I decided to go with an 80/10/10 split: 80% was used to train the models, 10% was used to validate their training, and the last 10% was used to test them. Four CPU cores were assigned to help preprocess the data. Each model had different hyperparameters: for Logistic Regression, the maximum features was set to 5000. Essentially, this limited the dictionary to only the 5000 most important words. Any words beyond this wouldn't matter too much, allowing the model to focus on the ones that mattered and stay on track. Additionally, I set the maximum number of iterations to 1000. This felt like a balanced value, as I wanted it high enough to be accurate while not using up too many resources.

As for the LSTM model, there were more parameters to tune, the first of which being the maximum sequence length. I set this to 200, which essentially cut every article down to 200 words. Articles cut to the chase, so a given article's classification would absolutely be established within this window. The next parameter for the LSTM model was the embedding dimension. This essentially affected how much detail each word was given once vectorized. I gave LSTM a dimension of 128 units, granting it space without contributing to a large file size.

The next parameter was the number of LSTM units, representing the size of its memory. I chose 64 units here, allowing for suitable memory without risking overfitting. The next parameter was dropout, which ignores a certain percentage of neurons during training to prevent the model from becoming too dependent on certain words. I set this value to 0.2, which dropped 20% of the neurons. Finally, I chose 5 epochs. By training the model 5 times, the model would become smart without a risk of overfitting.

The final model, BERT, had its own hyperparameters as well. I used a small learning rate of 2×10^{-5}; as BERT was already beautifully trained, I didn't dare overstep on its existing knowledge. The weight decay was set to 0.01 — while I wanted the model to be interested in the data, I also wanted to make sure it understood it in a general sense rather than a specific one. That is to say, I made sure it picked up on overall trends rather than the specific details of any one article. I used three epochs — this felt like a solid number of iterations that would allow the model to comprehend the data without overdoing it. Finally, I used a batch size of 16; this allowed the model to only look at 8 articles at a time, but made it only update itself every two batches. This gave it plenty of data at a time without crashing it.

Other tools were used as well. For example, I utilized Hugging Face to access BERT, as well as to obtain BERT's trainer. For the Logistic Regression model, I imported Scikit-learn to access its vectorizer and the logistic regression algorithm. Finally, for the LSTM model, I utilized TensorFlow's Keras API to tokenize it and for its neural network layers. As mentioned above, Google Colab and Google Drive were used in all three models. As I initially started with BERT, I first wrote its training code in Colab and then ran tests locally. However, as I did the other two models, I realized it was far easier to simply include their respective tests in their Colab code.

Another tool I utilized was the Streamlit library. This allows users to quickly and easily build simple frontends for testing machine learning applications. I utilized the BERT model with this website, allowing users to enter any string of text that they would like to classify. This extends the models beyond just their own data, forcing them to consider any new information they are fed.

Overall, while there was some setup required, training all three was thankfully quick and relatively simple.

Model Performance

Each of the models tested with different accuracies. For each of them, I found their accuracies as a percentage, their Matthews Correlation Coefficient (MCC), and ran a few of the same tests across all of them to notice differences. As seen in table 1 below, the Logistic Regression model had an accuracy of 0.99 with an MCC score of 0.98. The LSTM model, on the other hand, had an accuracy of 0.86 with an MCC score of 0.83. Finally, the BERT model had both an accuracy and MCC score of 1.0.

Model	Accuracy	MCC Score
Logistic Regression	0.99	0.98
LSTM	0.86	0.83
BERT	1.0	1.0

Table 1: The Accuracies of Each Model

These results shocked me — the simplest model was far better than the LSTM, and the BERT model was perfect. I created a quick app with Streamlit to allow users to play around with the BERT model — if a user inputs text into the website, it will return one of five categories. As I played around on the Streamlit website, I wondered if BERT was truly the best in every category. After all, its MCC of 1.0 meant it worked perfectly. As I typed strings into the website, I decided to try “football”; this changed my whole perception of these models.

BERT’s “football” score was 30.08% in the entertainment category. On the other hand, the LSTM model scored 99.61% in the sports category, while the Logistic Regression model scored 52.93% — also in the sports category.

This perplexed me for a moment. However, I soon came to theorize and understand these results. First, the Logistic Regression model saw "football" as part of the bag of words. It knew that it appears in sports articles, but not much else. Therefore, it placed it into the sports category, but only on that attribute alone. The LSTM, on the other hand, knew more about "football" through its embedding layer. It was aware of the contexts it often appeared in, leading to its relation to the sports category.

Finally, the elephant in the room is BERT. After all, the best model scored a low confidence in the wrong category. However, these results made more sense once I remembered how BERT works. BERT often replaces random words with masked tokens, requiring context to guess what those words are (Baheti, 2023). The model is clearly the best of the three when considering large sequences of text, as it quite literally blocks text out and forces itself to predict what that text is. However, in doing so, it places less emphasis on individual words. Football likely appeared in a sports article, sure — but it could have also appeared in an entertainment article in regards to popular culture. One word isn't enough for BERT to be able to guess, as it thrives with larger amounts of text. For instance, I tried another string of text with BERT: "The football player kicked a field goal and earned 3 points." This text gave a score of 60.43% in the sports category, clearly showing BERT's strength with long text.

I ran other tests on all three as well. The first string of text was "Ar ar ar ar ar ar ar ar." This was clearly gibberish, and I tested it to see how the models might interpret it. The Logistic Regression and LSTM models both put it in the sports category, with confidence scores of 33.25% and 99.65%. Perhaps the LSTM's higher rating was due to the repetition aspect — after all, chants are often repeated several times at sporting events. This would explain the Logistic Regression's lower score, as it lacked the context of the repetition. Bert, on the other hand, placed it in the politics category, with a score of 39.22%. BERT has the advantage of being pretrained, so seeing a completely random string like this may have caused it to look back at its previous training. Perhaps the string "Ar ar ar ar ar ar ar ar" sounded like the constant bickering of politics, or resembled something it once saw in a book.

The second string I tested was "Bleepity bloopity blop! Blorg blorg blop." Once again, this was a string of complete gibberish. The Logistic Regression model and LSTM models both put it in the sports category yet again, with confidence scores of 33.25% and 99.65% respectively. The reasoning for this likely lines up with the former test; perhaps the repetition and unique text was associated with the chaos and often upbeat absurdity of any given sporting event. BERT took another path yet again, placing it in the entertainment category with a confidence score of 90.57%. Given its previous training, BERT likely associated this text with Sci-Fi; after all, aliens and robots have often been associated with "bleeps" and "blorgs". This connection especially shows how BERT's previous training shapes its responses.

The last string I tested was a simple one: "The central bank raised interest rates to combat inflation today." Most people would arguably associate this with business, given its financial nature. The Logistic Regression model and the BERT model agreed with this, giving confidence scores of 69.75% and 95.70% respectively. As BERT is more knowledgeable on English as a whole, its increased confidence in this obviously business-related sentence makes sense. However, LSTM deviated from the other two here, scoring in the sports category with a 93.55% confidence. Whereas the Logistic Regression model saw this string in its bag of words and BERT knew what these words meant from previous and more extensive training, the LSTM model likely grew more dependent on the flow of the string. Certain words like "central", "raised", and “combat” may have grown more associated with sports, while words such as “bank” and “inflation” may have been overlooked.

Conclusion & Future Work

Overall, I enjoyed the process of training these three NLP models. While Colab gave me trouble near the start, the rest of the project was overall successful and revealed a lot of information. These three models each had their advantages; the Logistic Regression model, for example, trained the fastest and was mostly accurate for this small dataset. The LSTM model, on the other hand, was less accurate as a whole, perhaps overfitting on the initial dataset. BERT was the clear winner in most scenarios; however, its lesser accuracy regarding singular words and its longer training time showed some issues.

While this was a simple project, its implications carry over to the real world. While I only trained my models on 1780 articles and tested them with 223 articles, real datasets are far larger and require more extensive training. In extensive real-world applications like these, Logistic Regression models will likely prove to be inaccurate, whereas LSTM may show to be more useful than it was here.

As a whole, this project taught me a lot about machine learning, and I hope to do more research in this field in the future.

Citations

Baheti, P. (2023, September 11). BERT: State-of-the-Art Model for Natural Language Processing. *Comet*. https://www.comet.com/site/blog/bert-state-of-the-art-model-for-natural-language-processing/

Greene, D., & Cunningham, P. (2006). Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering. *Proc. 23rd International Conference on Machine Learning (ICML'06)*, 377–384. https://doi.org/10.1145/1143844.1143892

Long Short-Term Memory (LSTM). (n.d.). NVIDIA Developer. Retrieved December 22, 2025, from https://developer.nvidia.com/discover/lstm

Merritt, R. (2022, March 25). What Is a Transformer Model? *NVIDIA Blog*. https://blogs.nvidia.com/blog/what-is-a-transformer-model/

Text Classification using Logistic Regression. (11:52:52+00:00). GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/text-classification-using-logistic-regression/