From Classical Models to Fine-Tuned Transformers: A Comprehensive Analysis on IMDb Sentiment Classification

(Musab bin Jamil[1], Hussain Diwan[2])

About the Authors

This report has been authored by **Musab (ERP 29409)** and **Hussain (ERP 29410)**, who bring diverse academic and professional backgrounds to this collaborative project.

- **Musab** holds a background in **Astrophysics**, where he has honed analytical and computational skills to tackle complex, data-driven problems in the physical sciences. His expertise in mathematical modeling and statistical analysis has provided a strong foundation for exploring machine learning and natural language processing techniques.
- **Hussain**, on the other hand, is an **Electrical Engineer**, with a deep understanding of systems, algorithms, and optimization techniques. His engineering background complements the computational and theoretical aspects of this project, particularly in implementing efficient solutions and leveraging cutting-edge methods.

Both authors are currently working professionals, applying their skills to solve real-world challenges in their respective fields. This shared project reflects their mutual interest in machine learning and natural language processing, where they have combined their unique expertise to explore the performance of classical and advanced models on the IMDb sentiment classification task. Their professional experiences and diverse educational backgrounds have allowed them to approach this study with a multidisciplinary perspective, resulting in a thorough and well-rounded analysis. A GitHub link has been attached that can be found in the references section where you can find the notebooks and results for the work done for this project.

Introduction

Sentiment analysis, the process of extracting and analyzing subjective information from textual data, has become a cornerstone task in natural language processing (NLP). With the growing availability of large-scale textual datasets, researchers and practitioners have developed increasingly sophisticated models to interpret sentiment, ranging from classical machine learning algorithms to state-of-the-art deep learning approaches. One widely recognized benchmark for sentiment classification tasks is the **IMDb dataset**, which contains movie reviews labeled with binary sentiment (positive or negative). Its balanced structure and domain-specific focus have made it a standard dataset for evaluating sentiment analysis techniques.

This report investigates the performance of various machine learning models on the IMDb dataset, with a particular focus on comparing classical machine learning algorithms, pre-trained transformer-based models, and fine-tuned pre-trained models. The study highlights the evolution of NLP techniques and evaluates their efficacy in terms of accuracy, efficiency, and ability to generalize.

1. **Classical Machine Learning Models**: Classical algorithms, such as **Naïve Bayes, Random Forest, k nearest neighbours and Gradient Boosting,** are trained on text features extracted using methods like Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). These models rely heavily on feature engineering and have historically served as the foundation for sentiment analysis tasks.
2. **Pre-Trained Transformer-Based Models**: The advent of transformer architectures, such as **BERT (Bidirectional Encoder Representations from Transformers)**, **RoBERTa**, and **DistilBERT**, has revolutionized NLP by leveraging self-attention mechanisms and contextual embeddings. These models are pre-trained on massive corpora, enabling them to capture nuanced language representations that generalize well across tasks. In this study, pre-trained models are evaluated on the IMDb dataset without additional fine-tuning to establish a baseline for their out-of-the-box performance.
3. **Fine-Tuning Pre-Trained Models Using LoRA**: To further adapt pre-trained models to the IMDb dataset, a low-rank adaptation (LoRA) fine-tuning approach is applied. LoRA is a parameter-efficient fine-tuning method that updates only a small subset of the model's weights, significantly reducing computational and storage overhead while achieving competitive performance. This method allows for leveraging the power of large pre-trained models while maintaining practical scalability.

The primary objective of this report is to compare the performance of these three approaches—classical machine learning, pre-trained models, and fine-tuned models using LoRA—on the IMDb sentiment classification task. The evaluation metrics include accuracy, precision, recall, F1-score, and inference speed. Additionally, the report explores the trade-offs between model complexity, resource requirements, and prediction performance.

By conducting this comparative analysis, this study provides insights into the strengths and weaknesses of different modeling paradigms for sentiment analysis. The findings are expected to guide practitioners in selecting the most appropriate approach based on the task requirements, computational constraints, and desired accuracy.

Dataset

The dataset used in this report is the IMDb dataset as provided for the assignment. The IMDb dataset is a widely used benchmark for sentiment analysis and natural language processing tasks. It contains movie reviews sourced from the Internet Movie Database (IMDb), focusing on binary sentiment classification. The dataset was curated by researchers at Stanford University, designed to evaluate models' abilities to distinguish between strongly positive and strongly negative sentiment in textual data. To achieve this, only reviews with clear sentiment polarity were included: positive reviews correspond to ratings of 7 or higher, while negative reviews are drawn from ratings of 4 or lower on IMDb’s 1–10 scale. Neutral reviews with

ratings between 5 and 6 were deliberately excluded to avoid ambiguity. This ensures a clean and focused dataset for training and evaluating sentiment models.

The dataset consists of 50,000 reviews, split into two subsets: 30,000 for training and 20,000 for testing. Each subset is balanced, containing an equal number of positive and negative reviews. The reviews are stored as raw text comma separated files, with separate columns for reviews and sentiments. The dataset includes a mix of review lengths, ranging from short comments to extensive multi-paragraph critiques, with an average of approximately 230 words per review. The text is in English and largely unprocessed, retaining informal user-generated language that may include typos, slang, and inconsistent grammar. This raw nature of the data makes it particularly valuable for developing robust natural language processing models that can handle real-world text.

Data Preprocessing

For Classical Models

In the preprocessing of textual reviews with sentiment labels (positive or negative) for classical machine learning models, we followed several steps to clean and prepare the text data. Here's a breakdown of each step we utilized:

1. **HTML Tag Removal**:

To remove unwanted HTML tags (like “<”, “.”, “*”, “?” and “>”) and non-alphanumeric characters from the text, as these are irrelevant for sentiment analysis and can introduce noise into the data. Regular expressions or Regex was used to identify said characters. This step ensures that only the actual content of the review remains, making it easier to focus on the sentiment.

2. **Lowercasing**:

To ensure consistency and avoid treating the same word in different cases as distinct (e.g., "Happy" and "happy"). Every word in the text was converted to lowercase using python’s lower() function, ensuring that variations in letter case don’t result in unnecessary duplications or discrepancies during analysis.

3. **Tokenization**:

Tokenization breaks the corpus into smaller units (tokens), typically words, which are easier to analyze and process in models. The word tokenize function from the Natural Language Toolkit (NLTK) was used. The word tokenize function utilizes the Punkt tokenizer from NLTK, which is a pre-trained model specifically designed to handle the complexities of tokenization. This tokenizer can handle sentence segmentation and word tokenization effectively. Punkt uses unsupervised learning to detect sentence boundaries and tokenize text, meaning it can recognize when a sentence ends (e.g., at periods, exclamation marks, or question marks) and appropriately tokenize the words that follow.

4. **Stopword Removal**:

Stopwords are common words like "the," "is," and "in," which don't contribute much to the sentiment of the text and can introduce noise. Removing them reduces dimensionality and improves model performance. NLTK's predefined list of stopwords was used to filter out these common words. After removing stopwords, the remaining text contains only meaningful words that are more likely to influence sentiment analysis.

5. **Lemmatization**:

Lemmatization reduces words to their base or root form. For example, "running" would be reduced to "run," and "better" would be reduced to "good." This helps in standardizing words, which might appear in different forms but have the same meaning. **Method**: The WordNetLemmatizer from NLTK was used. This lemmatizer uses WordNet, a lexical database of English, to ensure that words are reduced to their dictionary forms, thereby improving the model's ability to recognize word meaning without being affected by tense or plurality.

Vectorization Techniques Applied

Once the text was cleaned, the next step was to convert the text data into numerical representations using vectorization methods. We applied two popular techniques:

1. **TF-IDF Vectorization**:

TF-IDF (Term Frequency-Inverse Document Frequency) measures the importance of a word in a document relative to its frequency across all documents. It helps in giving higher weights to words that are frequent in a document but rare across other documents. The TfidfVectorizer was used, considering both unigrams (individual words) and bigrams (pairs of adjacent words) as features. The maximum number of features was limited to 5000, which means that only the 5000 most important terms (based on TF-IDF scores) were kept, reducing the dimensionality of the feature space and making the model more efficient.

2. **Count Vectorization**:

Count vectorization simply counts the occurrences of words in the document. It provides a straightforward representation of word frequency but doesn't account for word importance (like TF-IDF does). The CountVectorizer was applied, also limited to 5000 of the most frequent terms. This ensures that only the most common terms across the dataset are used, avoiding rare words that might be noisy or too specific to a single review.

For Pre-Trained Models

For pre-trained models, the preprocessing steps differed slightly:

1. **Converting Sentiment Column to Numeric**:

Machine learning models, including pre-trained models, typically require numerical input. The sentiment labels ("positive" and "negative") were converted into numerical labels (e.g., 1 for positive and 0 for negative) to enable the model to work with them. Sentiment labels were mapped to numeric values using label encoding, ensuring compatibility with the pre-trained model's requirements.

2. **Using Each Model's Tokenizer**:

Pre-trained models (especially deep learning models like BERT) use their own specific tokenizers, which are designed to handle text in a way that fits the model's architecture. For each pre-trained model, the corresponding tokenizer (e.g., BertTokenizer for BERT or RobertaTokenizer for RoBERTa) was used to tokenize the text. These tokenizers are typically trained on a vast amount of text and know how to handle complex inputs like sub word tokens, punctuation, and other linguistic nuances that are important for pre-trained models to work effectively.

3. **Sub-Sampling for Faster Training**

Due to long training times, sub-sampling was applied to use smaller datasets for faster experimentation:

- Training Samples: 1000
- Testing Samples: 200

Sub-sampling significantly reduced training runtime but led to similar results to classical models

Classical Machine Learning Models

Classical models are widely used for sentiment analysis, where the goal is to classify text into categories like positive, negative, or neutral. In this report we used the Naïve Bayes Classifier, Random Forest, k-NN and Gradient Boosting to perform sentiment analysis. For classical models, performance is often measured using metrics like accuracy, precision, recall, and F1 score. Accuracy is useful but can be misleading in imbalanced datasets, where a high accuracy may result from predicting the majority class without identifying minority classes properly. Precision is important when false positives are costly, while recall is crucial when false negatives are more problematic. The F1 score balances both, making it particularly valuable for imbalanced datasets by accounting for both false positives and false negatives.

Training time varies across classical models. Naive Bayes is quick to train, especially with large datasets, while KNN can be slow during prediction due to distance computations. Random Forests require more computational resources because of the need to build multiple trees but can be parallelized. Gradient Boosting, while powerful, can be computationally intensive and slow to train due to the sequential nature of building multiple trees to correct errors, particularly with large datasets. Despite their varying training times, classical machine learning models are generally

faster to train than deep learning models, making them more suitable when computational resources are limited or when faster training is needed.

1. Naive Bayes Classifier

The Naive Bayes classifier is a simple yet powerful probabilistic algorithm commonly used for sentiment analysis. It is based on Bayes' Theorem, which calculates the probability of a given sentiment (positive or negative) based on the features (words or phrases) present in the text. The "naive" assumption made by the model is that all features (words) are independent, even though they may not be in reality. Despite this simplifying assumption, Naive Bayes often performs well in sentiment analysis tasks due to its efficiency and ability to handle large datasets. It is particularly effective for text classification problems, where it can predict sentiment by learning from labeled training data.

Training Time: ~0.5 seconds

Performance:

- Accuracy: 0.859
- Precision: 0.86
- Recall: 0.86
- F1-Score: 0.86

2. Random Forest Classifier

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve classification accuracy, making it a robust choice for classification tasks. It works by training numerous decision trees on random subsets of the training data and features, then aggregating their predictions through a majority vote (for classification). Each tree in the forest is built independently, which helps reduce overfitting compared to a single decision tree. For sentiment analysis, Random Forest can capture complex patterns and interactions between words, making it effective at distinguishing between positive, negative, or neutral sentiments. Its ability to handle large datasets and provide feature importance further enhances its utility in text classification tasks.

Training Time: ~15 seconds

Performance:

- Accuracy: 0.847
- Precision: 0.85
- Recall: 0.85
- F1-Score: 0.85

3. k-Nearest Neighbors (k-NN)

K-Nearest Neighbors (KNN) is a simple, non-parametric algorithm. It classifies a given text based on the majority sentiment of its 'k' nearest neighbors in the feature space. In the context of sentiment analysis, the feature space is typically composed of word vectors, where each data point represents a text document, and the distance between points is usually measured using metrics like Euclidean distance. When a new text is introduced, KNN finds the k closest training texts and assigns the most common sentiment label among them. KNN is effective for smaller datasets but can be computationally expensive for larger datasets due to the need to calculate distances between points. Its strength lies in its simplicity and the fact that it doesn't require training, making it a straightforward method for sentiment classification.

Training Time: ~30 seconds

Performance:

- Accuracy: 0.734
- Precision: 0.74
- Recall: 0.73
- F1-Score: 0.73

4. Gradient Boosting Classifier

Gradient Boosting is an ensemble learning technique that builds a strong predictive model by combining multiple weak models, typically decision trees, in a sequential manner. In sentiment analysis, it works by fitting a series of decision trees, each of which corrects the errors made by the previous tree. Each new tree is trained on the residuals (errors) from the previous tree, and the final prediction is made by combining the outputs of all trees in the sequence. This method gradually improves the model's accuracy, making it highly effective for tasks like sentiment analysis, where nuanced patterns in text data need to be captured. Gradient Boosting models are known for their high predictive power and ability to handle complex data, though they can be prone to overfitting if not properly tuned. Popular variations like XGBoost and LightGBM further enhance performance and efficiency, making them popular choices for text classification tasks.

Training Time: ~2 minutes

Performance:

- Accuracy: 0.808
- Precision: 0.81
- Recall: 0.81
- F1-Score: 0.81

Pre-Trained Transformer Models

Pre-trained transformer models, such as GPT and BERT, have revolutionized sentiment analysis by leveraging deep learning and large-scale pre-training on vast amounts of text data. These models are based on the transformer architecture, which is highly effective at capturing contextual relationships between words, even across long sentences. In sentiment analysis, pre-trained transformers are fine-tuned on labeled datasets to predict sentiment, such as positive, negative, or neutral, by understanding the underlying meaning and context of the text.

What makes these models powerful is their ability to understand the semantic meaning of words in context, which allows them to perform well on a variety of text classification tasks, including sentiment analysis. For instance, BERT uses bidirectional attention to consider both the left and right context of each word, making it especially effective for nuanced sentiment understanding.

These models can outperform traditional machine learning approaches due to their deep understanding of language and ability to generalize well across different domains. However, they are computationally intensive and require substantial resources for training and inference. The following transformer models were fine-tuned for the sentiment classification task:

1. DistilBERT

DistilBERT is a smaller, more efficient version of the BERT model, designed to retain much of BERT's performance while being faster and requiring less computational power. It is created using a technique called knowledge distillation, where a smaller "student" model is trained to mimic the predictions of a larger, pre-trained "teacher" model (in this case, BERT). DistilBERT achieves this by reducing the size of the model (by about 60%) while maintaining 97% of BERT’s language understanding capabilities.

In sentiment analysis, DistilBERT is highly effective due to its ability to capture contextual relationships and nuances in text, similar to BERT. However, it is much more efficient, making it suitable for environments with limited computational resources or when real-time processing is needed. Its smaller size allows it to perform faster while still providing strong performance on text classification tasks like sentiment analysis, where understanding the sentiment of the text (positive, negative, neutral) is crucial.

Training Arguments:

- Epochs: 3
- Batch Size: 8 (train), 16 (eval)
- Learning Rate: Optimized using a warm-up scheduler
- Weight Decay: 0.01

Training Time: ~3 minutes

Performance:

- Accuracy: 0.865
- Precision: 0.892
- Recall: 0.859
- F1-Score: 0.871

2. RoBERTa

RoBERTa is a robust variant of BERT, designed to improve upon BERT's performance by training on more data and using optimized training strategies. Unlike BERT, RoBERTa removes the Next Sentence Prediction task and is trained with larger mini-batches and for longer periods, making it more efficient in learning contextual representations. By leveraging a vast amount of text data, RoBERTa achieves superior performance across several natural language processing tasks, such as text classification, question answering, and named entity recognition. It outperforms BERT in many benchmarks, demonstrating a stronger ability to understand the nuances of language and complex contexts, making it highly effective in tasks like sentiment analysis, where discerning the subtle emotional tone of text is key.

Training Arguments:

- Epochs: 3
- Batch Size: 8 (train), 16 (eval)
- Learning Rate: Optimized using a warm-up scheduler (500 warm-up steps)
- Weight Decay: 0.01

Training Time: ~6 minutes

Performance:

- Accuracy: 0.860
- Precision: 0.884
- Recall: 0.851
- F1-Score: 0.867

3. ALBERT

ALBERT (A Lite BERT) is a more parameter-efficient version of BERT, designed to reduce the memory footprint and computational requirements while maintaining high performance on natural language processing tasks. ALBERT achieves this by sharing parameters across layers and employing factorized embedding parameterization, which significantly reduces the number of parameters without sacrificing the model's ability to capture complex language patterns. As a result, ALBERT is faster and more efficient than BERT, with much fewer parameters but still achieving comparable or even superior results on tasks like text classification, sentiment analysis, and question answering. Its reduced size makes it particularly useful in resource-constrained

environments where computational efficiency is critical, while still providing strong performance in understanding and interpreting text.

Training Arguments:

- Epochs: 3
- Batch Size: 8 (train), 16 (eval)
- Learning Rate: Optimized using a warm-up scheduler (500 warm-up steps)
- Weight Decay: 0.01

Training Time: ~6 minutes

Performance:

- Accuracy: 0.855
- Precision: 0.855
- Recall: 0.879
- F1-Score: 0.866

LoRA Fine-Tuning

LoRA (Low-Rank Adaptation) is a technique designed to efficiently fine-tune large pre-trained models by introducing low-rank matrices during the training process. Instead of updating all the parameters of a large model, LoRA focuses on adding a small number of additional parameters, which can capture the necessary adjustments for specific tasks. This method allows the model to retain its pre-trained knowledge while adapting to new tasks with minimal computational cost. LoRA is particularly beneficial for fine-tuning large models, like BERT or GPT, in scenarios where there are resource constraints or when only limited task-specific data is available. By using low-rank approximations, LoRA can significantly reduce the time and memory required for fine-tuning, making it an efficient and scalable approach for task adaptation.

Training Arguments:

- Rank (r): 16
- Scaling Factor (lora_alpha): 32
- Target Modules: "query", "value"
- LoRA Dropout: 0.1
- Bias: None
- Task Type: Sequence Classification (Sentiment Analysis)

Training Time: ~5 minutes

Performance (LoRA):

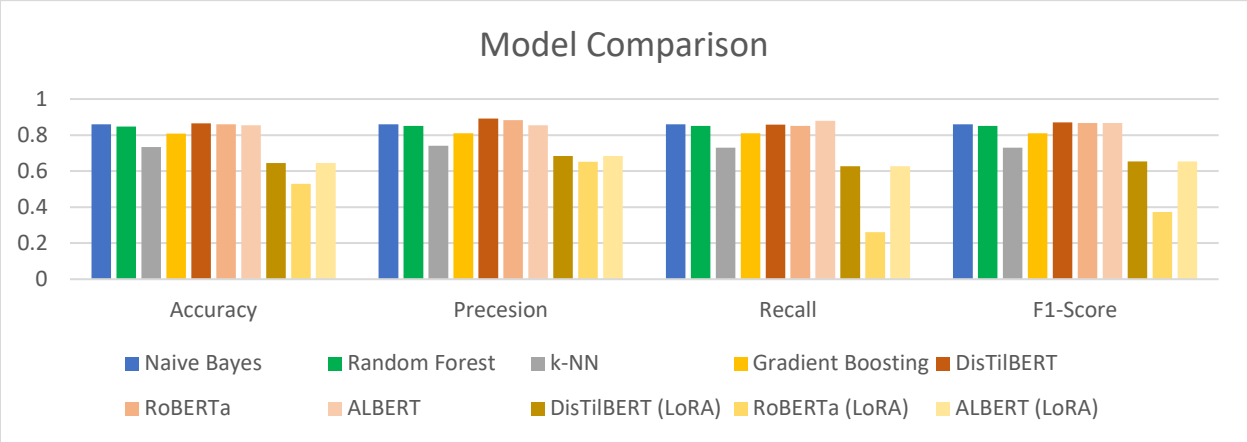
- **DistilBERT**:
  - Accuracy: 0.645
  - Precession: 0.6837
  - Recall: 0.6262
  - F1-Score: 0.654
- **RoBERTa**:
  - Accuracy: 0.530
  - Precession: 0.6512
  - Recall: 0.2617
  - F1-Score: 0.373
- **ALBERT**:
  - Accuracy: 0.645
  - Precession: 0.6837
  - Recall: 0.6262
  - F1-Score: 0.654

Comparative Results

The top three models—Naive Bayes, DistilBERT, and RoBERTa—demonstrate strong performance in terms of accuracy and F1-score. Naive Bayes stands out for its efficiency, achieving an accuracy of 0.859 and balanced metrics (precision, recall, and F1-score of 0.860) while requiring only 0.5 minutes of training time, making it highly suitable for quick deployments with minimal computational resources. DistilBERT, a distilled version of BERT, performs slightly better in accuracy (0.865) and exhibits impressive precision (0.892), although it takes 3 minutes to train. This tradeoff between performance and training time makes DistilBERT a solid choice for NLP tasks where higher accuracy is prioritized. RoBERTa, with an accuracy of 0.860 and an F1-score of 0.867, also performs very well, though it requires a significantly longer training time of 6 minutes. RoBERTa balances high performance with computational demands, making it ideal when more robust models are necessary for complex NLP tasks, provided that computational resources allow.

Model Name	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Training Time (minutes)
Naive Bayes	0.859	0.860	0.860	0.860	0.5
Random Forest	0.847	0.850	0.850	0.850	0.25
k-NN	0.734	0.740	0.730	0.730	0.5
Gradient Boosting	0.808	0.810	0.810	0.810	2
DistilBERT	0.865	0.892	0.859	0.871	3
RoBERTa	0.860	0.884	0.851	0.867	6
ALBERT	0.855	0.855	0.879	0.866	6
DistilBERT (LoRa)	0.645	0.6837	0.6262	0.6537	5
RoBERTa (LoRa)	0.53	0.6512	0.2617	0.3733	5
ALBERT (LoRa)	0.645	0.6837	0.6262	0.6537	5

The following bar chart compares the performance the machine learning models and generally, DistilBERT, RoBERTa, and Naïve Bayes models show consistently strong performance across all metrics, often achieving scores above 0.8. Random Forest and Gradient Boosting also perform reasonably well, though slightly lower than the top performers. k-NN consistently shows the weakest performance across all metrics. The LoRA (Low-Rank Adaptation) versions of DistilBERT, RoBERTa, and ALBERT generally underperform their full model counterparts, especially in Recall and F1-Score, suggesting a potential trade-off between efficiency gained by LoRA and overall performance.



Conclusion

This project demonstrated the effectiveness of both classical machine learning classifiers and modern transformer-based models for sentiment analysis. Transformer models consistently outperformed classical models, with DistilBERT achieving the highest accuracy (0.865). LoRA fine-tuning provided a parameter-efficient approach, but with lower performance across other parameters compared to full fine-tuning.

References

- **Code and Results:** **https://github.com/MusabbinJamil/TextAnalytics/tree/main/Assignment%203**
- **IMDb Dataset:** https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
- **Naive Bayes:** Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning*, *29*(2-3), 103-137.
- **k-NN:** Fix, E., & Hodges Jr, J. L. (1989). Discriminatory analysis-nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 238-247.
- **Random Forest:** Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32.
- **Gradient Boosting:** Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.
- **DistilBERT:** Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.

- **RoBERTa:** Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.
- **ALBERT:** Lan, Z., Chen, M., Goodman, S., & Gimpel, K. (2019). Albert: A lite bert for self-supervised learning of language representations.
- **LoRA:** Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models.