Fahimeh Ebrahimi

December 19, 2023

Human vs Machine Generated Text

In this project, I implemented a text classification algorithm to distinguish human vs bot generated texts. I used Google Colab for my implementation, and due to memory limits, I have not tried training the model for large iterations.

Before applying transformers, I first wanted to apply basic text classification algorithm, without incorporating any neural networks. So I applied text preprocessing techniques and then represented texts using both tf.idf and also Glove pretraining embeddings. Then I tried different classifiers (Logistic Regression, Random Forest, SVM). The best result was 0.72 accuracy. I also tried training a LSTM model, and the best result I achieved was again around 70% accuracy for both train and test sets. Finally, I applied the transformer architecture. The best result I achieved without overfitting was 0.9 accuracy on training data and 0.86 accuracy of test data. In what follows, I explained my model.

1. **Data-loader class**: In the project, a custom data loader class (`Dataset`) was implemented to handle the loading and pre-processing of text data. The data loader takes a DataFrame as input and utilizes the Transformers library's `BertTokenizer` for tokenization and encoding. It handles the extraction of labels, tokenization of text, and padding to ensure consistent input shapes. The class provides methods to access batches of inputs and labels during training.
2. **Text Tokenizer**: The `BertTokenizer` from the BERT model (`'bert-large-cased'`) was chosen for text tokenization. This tokenizer segments input text into subword tokens and maps them to corresponding indices. The choice of BERT-based tokenization is motivated by its effectiveness in capturing contextual information and semantic relationships within the text. I have tried `'bert-base-cased'` first, and as expected`'bert-large-cased'` variant performed better as due to its larger capacity, allowing the model to capture more complex patterns in the data.
3. **Neural Network for Classification**: The neural network used for classification is a custom `BertClassifier` built on top of the pre-trained BERT model (`'bert-large-cased'`). The classifier includes a dropout layer, a linear layer, and a `ReLU` activation function. The model leverages the BERT architecture's ability to understand contextual information and features within the text.

    a. Factors Considered for Neural Net Choice: The choice of using BERT-based architectures is motivated by their success in various natural language processing tasks, including text classification.
    b. Pre-training: The neural network is initialized with weights pre-trained on a large corpus, specifically the `'bert-large-cased'` variant. This pre-training is crucial for leveraging the knowledge gained from extensive language modeling tasks.
4. **Loss Function**: The cross-entropy loss (`nn.CrossEntropyLoss()`) was chosen as the loss function for the human/machine classification task, and I set the number of classes to two. Although I could use `nn.BCEWithLogitsLoss()` as well. This choice is optimal because it aligns with the nature of the classification task and provides a clear gradient signal for updating the model parameters during training. Also, you can easily change the number of classes and use the same code without changing the loss function.
5. **Optimization and Training Policy**: The Adam optimizer (`Adam`) was used for training the classifier with a specified learning rate. The choice of Adam is based on its effectiveness in adapting the learning rates for each parameter individually, leading to faster convergence.
6. **Data Augmentation Policy**: I have not implemented any data augmentation policy in the current version of the code. The reason was that according to my experiments, adding dropout and training for 5-10 epochs gave a good accuracy in both training and testing datasets without any visible overfitting. But data augmentation techniques such as random rotations or perturbations in the text could be explored to enhance classifier generalization and avoid overfitting.
7. **Training and Evaluation**: The classifier was trained using the provided train and validation sets. The training dataset was further split into train/val datasets in order to model parameter tuning. Training progress was monitored over multiple epochs, with performance metrics such as training loss, training accuracy, validation loss, and validation accuracy reported after each epoch. The model was evaluated on a validation set you provided, and I have not used it during training. And the final test set you provided was further used to predict (`predict` method) its labels using the trained model.
8. **Scalability for Newer LLMs**: The current codebase can serve as a foundation for incorporating newer large language models (LLMs). To scale for newer LLMs, minimal changes are required, primarily in the initialization of the BertClassifier and the choice of the tokenizer. By updating the model and tokenizer to the desired LLM variant, the code can seamlessly adapt to newer architectures without major modifications.