Classification of the 20 Newsgroups Dataset

Objective

The primary objective of this project was to develop machine learning models capable of accurately classifying articles in the 20 Newsgroups dataset into their respective categories. The dataset consists of a variety of articles from 20 distinct newsgroups, and the goal was to explore various techniques, from basic preprocessing to the application of advanced embeddings, to build effective classification models.

Dataset Overview

The 20 Newsgroups dataset is widely used for text classification tasks. A consolidated dataframe was created to combine all the data from this dataset for easier manipulation and analysis.

Key Statistics and Insights from the Dataset:

- **Total Articles**: The dataset contains 18,846 articles categorized into 20 distinct groups.
- **Largest Category**: The category `rec.sport.hockey` contains the highest number of articles, with just under 1,000 articles.
- **Smallest Category**: The category `talk.religion.misc` has the fewest articles, with just over 600 articles.

Article Length:

- **Average Length**: The average length of an article is 1,902 characters.
- **Longest Article**: The longest article in the dataset contains 16,000 characters.
- **Shortest Article**: The shortest article contains only 115 characters.

Most Common Word: The most common word in the dataset is "ax," appearing across multiple articles.

Exploratory Data Analysis (EDA)

An exploratory analysis was conducted to gain deeper insights into the dataset, including visualizations and statistical summaries of the text length, category distributions, and word frequency. Detailed findings can be found in the file `eda.ipynb`.

Machine Learning Experiments

Using GloVe Embeddings

In the `ML_with_GloVe.ipynb` file, experiments with GloVe (Global Vectors for Word Representation) embeddings were conducted to see how they affect model performance.

Preprocessing:

- The text data was lemmatized to reduce inflectional forms and derivationally related forms of words to their base form.
- The GloVe 6B 100-dimensional model was used to create an embedding dictionary. This model transformed the text data into vectorized format.
- These vectors were stacked to create a feature matrix (X), which was fed into the machine learning models.

Modeling and Results:

1. **Support Vector Classifier (SVC):**
   - **Grid Search Hyperparameters**:
     - C: 10
     - gamma: scale
     - kernel: rbf
   - **Accuracy**: 0.60
2. **Random Forest Classifier:**
   - **Hyperparameters**:
     - bootstrap: False
     - max_depth: 20
     - min_samples_leaf: 2
     - min_samples_split: 2
     - n_estimators: 300
   - **Accuracy**: 0.5667
3. **Gradient Boosting Classifier:**
   - **Accuracy**: 0.55

Using Word2Vec Embeddings

In the `ML_with_Word2Vec.ipynb` file, Word2Vec embeddings were used to convert text data into vectorized representations. These embeddings were then used to construct the feature matrix (X) for the machine learning models.

Preprocessing:

- After lemmatization, the text data was tokenized.
- Word2Vec embeddings were used to create dense vector representations of the documents. These embeddings served as features for training the machine learning models.

Scaling the Data:

For most of the experiments, the data was **scaled** to improve the performance of models sensitive to feature magnitudes. Scaling was particularly important for algorithms like SVM and XGBoost, which perform better when features have the same scale. **Scaling was done using StandardScaler**, which standardizes the data by removing the mean and scaling to unit variance.

Note: The first Random Forest experiment was conducted on **non-scaled data**. Random Forests, being tree-based models, are generally not sensitive to the scale of the data, which is why scaling was not applied to this specific test.

Modeling and Results:

1. **Random Forest Classifier (Non-Scaled Data)**:
   - **Hyperparameters**:
     - bootstrap: False
     - max_depth: None
     - min_samples_leaf: 1
     - min_samples_split: 2
     - n_estimators: 300
   - **Accuracy**: 0.6908
2. **Stacking Classifier (Random Forest + XGBoost)**:
   - **Base Estimators**:
     - Random Forest (using the best parameters from grid search)
     - XGBoost (with `use_label_encoder=False` and `eval_metric='mlogloss'`)
   - **Final Estimator**: Logistic Regression
   - The data was **scaled** before training.
   - **Accuracy**: 0.66
3. **Support Vector Machine (SVM)**:
   - **Hyperparameters**:

        - C: 1
        - gamma: scale
        - kernel: rbf
    - The data was **scaled** before training.
    - **Results**:
        - Unscaled Test Accuracy: 0.7061
        - Scaled Test Accuracy: 0.7119

Key Findings with Scaling:

- **Scaling improved performance** for models like SVM and XGBoost, as evidenced by the increase in accuracy from 0.7061 to 0.7119 in SVM with scaling.
- **Random Forest performance** (non-scaled) was stable, showing that tree-based models do not require scaling to perform well.
- The **Stacking Classifier** and **SVM** both benefited from scaling, highlighting the importance of this step for algorithms that are sensitive to feature scaling.
        -

Using TF-IDF

In the `ML_with_TF-IDF.ipynb` file, experiments were conducted using TF-IDF features based on unigrams for SVC, Multinomial Naive Bayes (NB), and Random Forest classifiers.

1. **Support Vector Classifier (SVC):**
    - **Grid Search Hyperparameters**:
        - C: 100
        - gamma: scale
        - kernel: rbf
    - **Best Cross-Validation Accuracy**: 0.7289
    - **Test Accuracy**: 0.7255
2. **Multinomial Naive Bayes (NB):**
    - **First Run (alpha = 1, fit_prior = true)**:
        - **Accuracy**: 0.7045
    - **Second Run (alpha = 0.01, fit_prior = true)**:
        - **Accuracy**: 0.7552
3. **Random Forest Classifier:**
    - **Hyperparameters**:
        - bootstrap: False
        - max_depth: None
        - min_samples_leaf: 1
        - min_samples_split: 10
        - n_estimators: 300
    - **Accuracy**: 0.6735

Key Findings

- **TF-IDF SVC**: The SVC model performed well with TF-IDF features, achieving a test accuracy of 0.7255. A grid search helped identify the optimal hyperparameters, leading to a best cross-validation accuracy of 0.7289.
- **Multinomial Naive Bayes (NB)**: The Multinomial NB model showed the highest accuracy in its second run with `alpha=0.01`, achieving a test accuracy of 0.7552. The first run with `alpha=1` performed reasonably well at 0.7045, indicating that adjusting the alpha parameter can significantly impact performance.
- **Random Forest**: The Random Forest model achieved a test accuracy of 0.6735 when using TF-IDF features, which is lower than the other models.

Neural Network Experiments

To explore the performance of neural networks on the 20 Newsgroups dataset, several models were tested, including simple feedforward models, recurrent neural networks (RNNs) with LSTM layers, and models with custom attention mechanisms. Below is a summary of the experiments conducted and their results.

Model 1: Simple Feedforward Neural Network (Fully Connected)

In this experiment, a basic feedforward neural network was used, employing a simple embedding layer followed by dense layers.

- **Results**:
    - **Training Accuracy**: 86.58%
    - **Training Loss**: 0.4385
    - **Validation Accuracy**: 61.09%
    - **Validation Loss**: 1.8632

While the model performed well on the training data, the validation accuracy was significantly lower, suggesting overfitting or room for improvement in regularization.

Model 2: Bidirectional LSTM with Dropout

The second experiment utilized a **Bidirectional LSTM** layer, which can capture both past and future context in the sequences. The model also included **dropout layers** for regularization to prevent overfitting.

- **Results**:
    - **Training Accuracy**: 78.95%
    - **Training Loss**: 0.6147
    - **Validation Accuracy**: 58.75%
    - **Validation Loss**: 1.8230

This model showed better generalization than the previous one but still struggled to achieve high validation accuracy. The higher dropout rate helped reduce overfitting to the training data.

Model 3: L2 Regularized LSTM

The third model introduced **L2 regularization** to the LSTM and dense layers, aiming to further prevent overfitting by penalizing large weights.

- **Results**:
    - **Training Accuracy**: 4.61%
    - **Training Loss**: 2.9905
    - **Validation Accuracy**: 4.56%
    - **Validation Loss**: 2.9923

Despite the use of regularization, this model showed extremely poor performance, suggesting that the added regularization may have been too strong or that hyperparameters (e.g., learning rate) might need adjustment.

Model 4: Attention Mechanism with LSTM

In the final experiment, an **Attention** mechanism was added to the Bidirectional LSTM to allow the model to focus on important parts of the sequence. The attention mechanism weights different parts of the input sequence to enhance relevant information for classification.

- **Results**:
    - **Training Accuracy**: 89.73%
    - **Training Loss**: 0.3453
    - **Validation Accuracy**: 64.52%
    - **Validation Loss**: 1.7523

The addition of the attention mechanism led to a significant improvement in the model's performance, with a higher training accuracy and better validation accuracy compared to the previous models. This suggests that the attention mechanism was beneficial for capturing important features in the text data.

Summary of Neural Network Experiments:

- The **simple feedforward neural network** showed promising results but had overfitting issues.
- The **Bidirectional LSTM** improved generalization, but the validation accuracy still lagged behind the training accuracy.
- The introduction of **L2 regularization** did not improve performance, and it may have been too restrictive.
- The **Attention mechanism** improved performance significantly, achieving the highest validation accuracy of 64.52%.

These experiments highlight the potential of neural networks for text classification tasks, with attention mechanisms showing the most promise in enhancing model performance.

Model Training and Evaluation for Text Classification Using BERT

In this section, we describe the steps taken to train and evaluate a BERT model for text classification on the 20 Newsgroups dataset.

1. Data Preparation

The 20 Newsgroups dataset was loaded using the `fetch_20newsgroups` function. To ensure that only relevant content was considered for training, headers, footers, and quotes were removed from the dataset. After loading the data, it was split into training and validation sets, with 80% of the data used for training and 20% for validation.

2. Tokenization

The text data in the dataset was tokenized using BERT's tokenizer. Tokenization is the process of converting raw text into input IDs and generating attention masks. The tokenizer converts each word into a token and ensures that all sequences are padded to a fixed length (128 tokens in this case). The attention masks allow the model to differentiate between actual tokens and padding tokens.

3. Model Definition

A pre-trained BERT model for sequence classification (`bert-base-uncased`) was used as the starting point. This model was configured to handle 20 output classes, corresponding to the 20 categories in the Newsgroups dataset. The model was moved to a GPU (if available) for faster training.

4. DataLoader Setup

To efficiently process the data during training and evaluation, `DataLoader` objects were created for both the training and validation datasets. The training data was shuffled using a `RandomSampler`, while the validation data was processed sequentially using a `SequentialSampler`. Batching was done with a batch size of 16.

5. Optimizer and Scheduler

The AdamW optimizer was used for fine-tuning the BERT model, with a learning rate of 2e-5. A learning rate scheduler was also employed to adjust the learning rate during training, ensuring stable and efficient model convergence.

6. Training Function

The training process involved iterating over the training data for several epochs. During each epoch, the model performed a forward pass to compute the loss, followed by a backward pass to update the model's weights using the optimizer. The loss was accumulated over the course of the epoch to track the model's progress.

7. Evaluation Function

After each epoch of training, the model was evaluated on the validation set to assess its performance. Accuracy was calculated for each batch, and a classification report was generated to provide detailed metrics (precision, recall, F1 score) for each category.

8. Training and Evaluation Loop

The model was trained for 3 epochs. After each epoch, the training loss and validation accuracy were printed, along with a classification report showing how well the model performed across different categories. This process allowed for monitoring the model's progress and making adjustments if needed.

Model Performance

After training the BERT model for text classification on the 20 Newsgroups dataset, the model achieved an **expected validation accuracy of 0.7387**. This indicates that the model correctly predicted the category for approximately 73.87% of the validation samples.

Additionally, detailed performance metrics such as precision, recall, and F1 score were computed for each of the 20 categories, offering a comprehensive view of the model’s ability to classify each news group accurately.

Conclusion

The overall performance of the models tested resulted in mid-range accuracy scores. Several factors contributed to this outcome, as outlined below:

1. **Data Complexity and Quality**: The 20 Newsgroups dataset is diverse and noisy, containing text from different sources like news articles, forums, and group discussions. This can make it challenging for models to generalize effectively. The dataset also has some class imbalances, which can hurt model performance, especially in multi-class classification tasks.
2. **Basic Preprocessing**: The preprocessing steps (removal of stopwords, HTML tags, and conversion to lowercase) were relatively basic. More advanced text preprocessing techniques such as lemmatization, domain-specific filtering, or removing irrelevant words could have enhanced the data quality. Additionally, feature extraction methods like TF-IDF might not have captured deeper semantic relationships, which may have hindered the performance of traditional models like Logistic Regression, Naive Bayes, and Random Forest.

3. **Neural Networks (MLP)**: The neural networks, while capable of capturing complex relationships within the data, did not perform significantly better than traditional models in this case. This could be due to several reasons:
    - **Insufficient training**: Neural networks, especially when using basic architectures like MLP (Multi-Layer Perceptron), require proper tuning of hyperparameters (e.g., number of layers, neurons per layer, activation functions) and longer training times to perform well on complex tasks like text classification.
    - **Dataset Size and Model Overfitting**: Neural networks are highly flexible, which can make them prone to overfitting, especially with limited data or inadequate regularization techniques. More data and better regularization methods (e.g., dropout, L2 regularization) would help reduce overfitting and improve performance.
    - **Hyperparameter Optimization**: As with other models, the lack of extensive hyperparameter tuning could have restricted the neural network's ability to reach its full potential.
4. **BERT (Transformer-based Model)**: BERT, being a transformer-based model pre-trained on large text corpora, is inherently better suited for understanding complex linguistic patterns and semantic relationships in text. However, even BERT's performance was not outstanding in this case. The following factors might explain why:
    - **Insufficient Training Time**: BERT requires a significant amount of training time to perform well, especially on a dataset like 20 Newsgroups. With only three epochs, the model did not have enough time to fine-tune its weights sufficiently to achieve optimal performance.
    - **Batch Size and Learning Rate**: Suboptimal choices for batch size or learning rate can hinder BERT's training process, leading to subpar results. These parameters could be further tuned to enhance model performance.
    - **Computational Resources**: BERT is a computationally intensive model that requires powerful hardware and sufficient time to train effectively. If the training was conducted on limited resources, it could have restricted the model's performance.
5. **Hyperparameter Tuning and Optimization**: Across all models, hyperparameter optimization was limited. For instance, Random Forest's number of trees, Logistic Regression's regularization strength, and Naive Bayes' smoothing parameters could all be fine-tuned for better performance. For neural networks and BERT, key hyperparameters such as the number of layers, learning rate, and batch size were not optimized, which likely affected their performance.
6. **Evaluation Metrics**: While accuracy was used as the evaluation metric, it may not fully capture model performance, especially in multi-class classification problems. Other metrics such as precision, recall, and F1-score could have offered a clearer picture of the model's true effectiveness, especially when dealing with class imbalances.
7. **Training Time and Computational Resources**: Neural networks, especially deep learning models like BERT, are computationally expensive and require longer training times to perform well. With the current setup and constraints, the models did not have

enough time to reach optimal performance. Additionally, the lack of GPU utilization or limited GPU time could have restricted the models' learning capabilities.

Overall Insights

The moderate performance observed across all models can be attributed to several factors:

- The complexity and noise in the 20 Newsgroups dataset
- Basic preprocessing steps
- Insufficient hyperparameter optimization
- Limited training time

BERT, although having the potential to perform better, especially with more data and fine-tuning, did not outperform the other models in this case due to the limited number of training epochs and potential hardware constraints. Neural networks also showed mid-range performance, which could be attributed to insufficient training and hyperparameter tuning. Traditional models like Random Forest, Logistic Regression, and Naive Bayes provided solid baseline results but did not capture the deeper semantics of the data as effectively as BERT and neural networks.

For future improvements:

- **BERT**: More epochs, careful hyperparameter tuning (e.g., learning rate, batch size), and training on more data could lead to significantly better results.
- **Neural Networks**: Experimenting with more sophisticated architectures, longer training times, and better regularization techniques could help enhance their performance.
- **Traditional Models**: Further feature engineering and hyperparameter optimization could improve their ability to perform well on more complex datasets like 20 Newsgroups.

The results highlight the importance of balancing model choice, data preprocessing, and training duration to achieve the best outcomes in text classification tasks.