**DS-595 Natural Language Processing**

**Assignment-2**

**Nitya Phani Santosh Oruganty**

* **Exploratory Data Analysis (EDA):**

**Basic Data Set Information ( 10% of whole data ):**

**A screenshot of a computer

Description automatically generated**

**1.Wordcloud for Text column in Data frame:**

A close-up of words

Description automatically generated

**2. Target Variable Distribution ( Score Based on Reviews which is used for calculating sentiment ):**

**A white background with black text

Description automatically generated**

* **Created labels based on the score, e.g., score>3: positive, score<3: negative.**

**A screenshot of a computer

Description automatically generated**

* **Sentiment Distribution:**

**A black text on a white background

Description automatically generated**

* **Resampling for imbalanced dataset :**

**A close-up of a number

Description automatically generated**

* **Dataset After Preprocessing and Resampling :**

**A text on a white background

Description automatically generated**

**A screenshot of a menu

Description automatically generated**

* **TF-IDF Approach:**

The code performs sentiment analysis using various machine learning models and evaluates their performance using cross-validation and test data.

A screenshot of a document

Description automatically generatedA screenshot of a computer screen

Description automatically generated

**Methodology:**

1. Data Preparation: The dataset is split into training and testing sets using the `train\_test\_split` function from scikit-learn.

2. Model Selection: Several classification models are chosen for sentiment analysis:

- Logistic Regression

- Multinomial Naive Bayes

- Random Forest

- LightGBM (Gradient Boosting Machine)

3. Pipeline Construction: Each model is combined with a TF-IDF vectorizer within a scikit-learn pipeline. This allows text data to be transformed into numerical features using TF-IDF before being fed into the classification model.

4. Cross-Validation: Cross-validation is performed using stratified k-fold (5 folds) to assess the generalization performance of each model on the training data.

5. Model Evaluation: After cross-validation, each model is fitted on the entire training data and evaluated on the test data. Performance metrics include accuracy and a detailed classification report containing precision, recall, and F1-score for each class.

**Results:**

- Logistic Regression: Achieves a mean cross-validation accuracy of around 0.80 and a test accuracy of around 0.81. Provides a detailed classification report.

- Multinomial Naive Bayes: Achieves a mean cross-validation accuracy of around 0.76 and a test accuracy of around 0.77. Provides a detailed classification report.

- Random Forest: Achieves a mean cross-validation accuracy of around 0.70 and a test accuracy of around 0.71. Provides a detailed classification report.

- LightGBM: Achieves a mean cross-validation accuracy of around 0.77 and a test accuracy of around 0.77. Provides a detailed classification report.

Logistic Regression and Multinomial Naive Bayes perform relatively well and consistently across cross-validation and test data.

Random Forest shows slightly lower performance compared to Logistic Regression and Multinomial Naive Bayes.

Light-GBM performs competitively, with performance similar to Multinomial Naive Bayes.

Logistic Regression and Multinomial Naive Bayes are recommended for sentiment analysis due to their robust performance and simplicity.

Light-GBM also shows promise and can be further optimized for potentially higher performance.

Random Forest may not be the best choice for this task given its lower performance compared to other models.

* **Word2Vec Approach:**

This code segment performs text classification using word embeddings generated by a Word2Vec model and various classifiers. Here's a breakdown of what's happening:

**1.Tokenization and Word Embeddings Generation:**

The text data (`X\_train` and `X\_test`) is tokenized into words.

A Word2Vec model is trained on the tokenized text data (`X\_train\_tokenized`). This model learns to map words into fixed-size dense vectors (word embeddings).

**2.Word Embeddings Generation Function:**

A function `generate\_word\_embeddings` is defined to generate word embeddings for each text.

This function iterates through each word in a text, retrieves its word embedding from the trained Word2Vec model, and computes the average of all word embeddings in the text.

If a word is not found in the Word2Vec model, it returns a zero vector.

**3.Classifier Definitions:**

Three classifiers are defined: Logistic Regression, Random Forest, and LightGBM.

Each classifier is instantiated with its default hyperparameters.

**4.Classifier Training and Evaluation:**

For each classifier:

A pipeline is created with the classifier.

Cross-validation is performed using `cross\_val\_score` to estimate the model's accuracy.

The classifier is trained on the training set (`X\_train\_embeddings`) and evaluated on the test set (`X\_test\_embeddings`).

Classification report metrics (precision, recall, F1-score, and support) are computed using `classification\_report`.

Cross-validation accuracy, accuracy on the test set, and the classification report are printed.

For each classifier, the cross-validation accuracy (mean and standard deviation), accuracy on the test set, and the classification report are printed.

This code demonstrates how to train and evaluate text classification models using Word2Vec word embeddings and different classifiers, providing insights into the models' performance through metrics such as accuracy, precision, recall, and F1-score.

A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated

* **Bert without fine tune:**

This code segment performs sentiment analysis using a pre-trained BERT-based model and evaluates its performance using various metrics.

**1.Loading Sentiment Analysis Pipeline:**

The code imports the `pipeline` function from the Transformers library to load a pre-trained sentiment analysis pipeline.

**2.Data Preparation:**

It extracts the text and sentiment columns from the `final\_df` DataFrame.

The text data is truncated to a maximum sequence length of 512 tokens to fit the model input requirements.

**3.Sentiment Classification:**

The sentiment analysis pipeline is applied to each truncated text to predict the sentiment label (positive/negative).

Predicted sentiment labels are extracted from the pipeline's output and converted to lowercase.

**4.Evaluation Metrics Calculation:**

The true sentiment labels (binary numeric values) are obtained from the original DataFrame.

The predicted sentiment labels are converted to binary numeric values.

Evaluation metrics such as accuracy, precision, recall, and F1-score are calculated using functions from scikit-learn (`accuracy\_score`, `precision\_score`, `recall\_score`, `f1\_score`).

**5.Saving Tokenizer and Model:**

The code loads the BERT tokenizer and model using `BertTokenizer.from\_pretrained` and `BertForSequenceClassification.from\_pretrained`.

It specifies a directory to save the tokenizer and model files.

The tokenizer and model are saved to the specified directory using `save\_pretrained`.

**6.Optional: Verification of Saved Files:**

Optionally, the code prints a message confirming the successful saving of tokenizer and model files.

**7.Additional Cleanup:**

To avoid memory leaks, the code deletes the tokenizer and model objects and empties the CUDA cache (if applicable).

This code demonstrates how to perform sentiment analysis using a pre-trained BERT-based model, evaluate its performance, and save the tokenizer and model for future use. The evaluation metrics provide insights into the model's effectiveness in predicting sentiment labels.

A number on a white background

Description automatically generated

* **Bert with fine tune :**

The provided code segment performs fine-tuning and evaluation of a BERT-based sentiment classification model using the Hugging Face Transformers library.

**1.Data Preparation:**

i) A fraction (5%) of the `final\_df` dataset is sampled for training and evaluation.

ii) The sampled data is split into training and validation sets.

iii) Texts are tokenized using the BERT tokenizer, and labels are converted to tensors.

**2.Model Fine-Tuning:**

* + 1. The BERT model is loaded for sequence classification.
    2. Training arguments are defined, specifying the output directory for saving results and the number of training epochs.
    3. A custom data collator is defined for sequence classification.
    4. Trainer is initialized with the model, training arguments, training dataset, evaluation dataset, and custom data collator.
    5. The model is fine-tuned using the Trainer.

**3.Evaluation:**

The model is evaluated on the validation dataset using the Trainer's `evaluate` method.

Evaluation metrics such as loss, accuracy, precision, recall, and F1-score are calculated.

**4.Results and Metrics:**

Evaluation results and additional metrics (accuracy, precision, recall, F1-score) are printed.

Evaluation results and metrics provide insights into the performance of the fine-tuned model.

The report should include an overview of the fine-tuning process, mentioning the dataset sampling, model training, and evaluation steps.

Results should be presented, including evaluation metrics, such as loss, accuracy, precision, recall, and F1-score.

Insights into the model's performance and potential areas for improvement should be discussed.

Visualizations, such as training/validation loss curves or confusion matrices, can enhance the presentation of results.

By including the above information and insights from the provided code segment, you can create a comprehensive report on the fine-tuning and evaluation of the BERT-based sentiment classification model.

A close-up of a computer screen

Description automatically generated

* **Bert with LoRa:**

The provided code segment outlines a process for training and evaluating a sentiment classification model based on BERT with LoRA (Label Refinery Attention). Below is a detailed explanation for each part of the code:

**1. Data Preparation:**

- A fraction (5%) of the `final\_df` dataset is sampled for training and evaluation purposes.

- The sampled data is split into training and evaluation sets using the `train\_test\_split` function from scikit-learn.

**2. Custom Dataset Class:**

- `CustomDataset` class is defined to preprocess the data for model training.

- It takes a dataframe, tokenizer, and max\_length as input parameters.

- The `\_\_len\_\_` method returns the length of the dataframe.

- The `\_\_getitem\_\_` method tokenizes each text sample, converts the sentiment label to a binary numeric value, and returns a dictionary containing input\_ids, attention\_mask, and labels.

**3. Model and Optimizer Initialization:**

- BERT tokenizer is instantiated from the `'bert-base-uncased'` pretrained model.

- BERT model for sequence classification (`BertForSequenceClassification`) is initialized with the same configuration.

- LoRA configuration (`LoraConfig`) is defined with specific parameters (r=16, lora\_alpha=16, lora\_dropout=0.1, bias="none").

- The BERT model is integrated with LoRA using the `get\_peft\_model` function.

- AdamW optimizer is defined to optimize the parameters of the LoRA-integrated BERT model.

**4. Data Loading and Dataloaders:**

- `CustomDataset` instances are created for both training and evaluation datasets.

- Dataloaders are initialized for both datasets to facilitate batch-wise processing during training and evaluation.

**5. Training Loop:**

- The training loop runs for a specified number of epochs (`num\_epochs`).

- The model is set to training mode (`lorabert.train()`).

- Batch-wise training is performed using the defined dataloader.

- Optimizer gradients are zeroed, and backpropagation is applied to compute gradients and update model parameters.

- Training loss is displayed using tqdm progress bar.

**6. Evaluation Loop:**

- The evaluation loop runs on the evaluation dataset.

- The model is set to evaluation mode (`lorabert.eval()`).

- Batch-wise evaluation is performed using the defined dataloader.

- Model predictions and true labels are collected for evaluation metrics calculation.

**7. Evaluation Metrics:\*\***

- Accuracy, precision, recall, and F1-score are calculated using scikit-learn's metrics functions (`accuracy\_score`, `precision\_score`, `recall\_score`, `f1\_score`).

- A classification report is generated using the `classification\_report` function, providing detailed metrics for each class.

**8.Results Printing:**

- Evaluation metrics (accuracy, precision, recall, F1-score) are printed to assess the model's performance.

- The classification report is printed to provide a detailed breakdown of the model's performance across different classes.

In summary, this code segment demonstrates the entire pipeline for training and evaluating a sentiment classification model using BERT with LoRA, including data preprocessing, model initialization, training, evaluation, and result analysis. This detailed explanation can be used to document the process and findings in a report effectively.

A green bar with black text

Description automatically generatedA number of numbers on a white background

Description automatically generated

* **Result Analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Precision** | **Recall** | **Accuracy** | **F-1 Score** |
| TFIDF-model 1 – **Logistic Regression** | 0.88 | 0.88 | 0.88 | 0.88 |
| TFIDF-model 2 – **Multinomial Naïve bayes** | 0.88 | 0.87 | 0.87 | 0.87 |
| TFIDF-model 3 – **Random Forest** | 0.79 | 0.79 | 0.79 | 0.79 |
| TFIDF-model 4 – **Light-GBM** | 0.85 | 0.85 | 0.85 | 0.85 |
| word2vec-model-1-**Logistic Regression** | 0.81 | 0.81 | 0.80 | 0.81 |
| word2vec-model- 2 – **Random Forest** | 0.86 | 0.86 | 0.86 | 0.86 |
| word2vec-model- 3 – **Light-GBM** | 0.84 | 0.84 | 0.84 | 0.84 |
| BERT **w/o fine tune** | 0.82 | 0.82 | 0.82 | 0.82 |
| BERT **with fine tune** | 0.87 | 0.90 | 0.88 | 0.88 |
| BERT **with LoRA** | 0.91 | 0.76 | 0.82 | 0.83 |

**Interpretation of the results provided:**

**1.TFIDF Models:**

- TFIDF-model 1 (Logistic Regression) and TFIDF-model 2 (Multinomial Naïve Bayes) perform similarly in terms of precision, recall, accuracy, and F1-score, with high values around 0.88.

- TFIDF-model 3 (Random Forest) and TFIDF-model 4 (LightGBM) also perform similarly, but with slightly lower scores compared to the logistic regression and Naïve Bayes models.

**2.Word2Vec Models:**

- Word2Vec-model-1 (Logistic Regression) performs reasonably well, with scores around 0.81 for all metrics.

- Word2Vec-model-2 (Random Forest) and Word2Vec-model-3 (LightGBM) have similar performances, with slightly higher scores compared to the logistic regression model.

**3. BERT Models:**

- BERT without fine-tuning achieves moderate scores across all metrics, indicating that the pretrained BERT model alone provides decent performance.

- BERT with fine-tuning outperforms BERT without fine-tuning, showing improvements in precision, recall, accuracy, and F1-score.

- BERT with LoRA (Label Refinery Attention) exhibits high precision but lower recall compared to BERT with fine-tuning, resulting in a slightly lower F1-score and accuracy.

**Interpretation and Comparisons:**

**TFIDF vs. Word2Vec vs. BERT:**

- BERT models generally outperform both TFIDF and Word2Vec models across all metrics. This can be attributed to BERT's ability to capture complex contextual relationships in text compared to traditional methods like TFIDF and Word2Vec, which rely on simpler statistical and semantic representations.

- Word2Vec models perform better than TFIDF models, indicating that word embeddings capture more meaningful semantic information compared to TFIDF's bag-of-words approach.

**BERT with LoRA vs. BERT with Fine-tuning:**

- BERT with LoRA achieves higher precision but lower recall compared to BERT with fine-tuning. This suggests that LoRA may be refining the predictions to focus more on precision, possibly at the expense of recall.

- The difference in performance between BERT with fine-tuning and BERT with LoRA highlights the trade-off between precision and recall, with fine-tuning providing a better balance between the two metrics.

In summary, the superior performance of BERT models over TFIDF and Word2Vec models underscores the effectiveness of deep contextualized representations in natural language understanding tasks. Additionally, the comparison between BERT with fine-tuning and BERT with LoRA emphasizes the importance of considering trade-offs between precision and recall when selecting model configurations for specific use cases.