Assignment 1

NLP Text Preprocessing Techniques

1. Introduction

This report explains the implementation and investigation of fundamental NLP preprocessing methods. The goal was to gain hands-on experience with key techniques essential for preparing text data for further analysis or model training.

2. Techniques Covered

The assignment covered the following essential NLP preprocessing techniques:

- **Tokenization**: This process involves breaking down raw text into smaller units, known as tokens. The notebook explored both word tokenization (splitting text into words) and sentence tokenization (splitting text into sentences) using popular NLP libraries such as NLTK and spaCy.
- **Lemmatization**: Lemmatization is the process of reducing words to their base or dictionary form (lemma). Unlike stemming, lemmatization considers the word's context and part of speech to ensure that the resulting lemma is a grammatically correct word. For instance, "running," "ran," and "runs" would all be lemmatized to "run."
- **Stemming**: Stemming is a more aggressive method of normalizing words by reducing them to their root or stem, typically by chopping off suffixes. While faster and simpler than lemmatization, the resulting stem may not always be a valid word (e.g., "beautiful" might become "beauti").
- **Part-of-Speech (POS) Tagging**: POS tagging involves identifying the grammatical category of each word in a text (e.g., noun, verb, adjective, adverb). This technique provides valuable linguistic information, aiding in subsequent NLP tasks.
- **Named Entity Recognition (NER)**: NER is a subtask of information extraction that identifies and classifies named entities in text into pre-defined categories such as person names, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

3. Comparison of Lemmatization and Stemming

A major focus was on comparing lemmatization and stemming:

- **Linguistic Accuracy:** Lemmatization is more accurate, considering context to produce valid words, while stemming is less precise and can create non-words.
- **Handling Irregular Forms:** Lemmatization can deal with irregular forms (e.g., “better” → “good”), unlike stemming.

- **Speed vs. Accuracy:** Stemming is faster and simpler, suited for speed-critical tasks, whereas lemmatization, though slower, offers better linguistic precision.

4. Conclusion

The assignment successfully provided foundational knowledge of various NLP preprocessing techniques, including tokenization, lemmatization, stemming, POS tagging, and NER, along with a detailed comparison between lemmatization and stemming, equipping essential skills for text data preparation.

Word Embeddings and Visualization

1. Introduction

This report outlines the objectives, methodologies, and findings of the "Word Embeddings and Visualization" assignment. The core focus was to explore and implement two distinct word embedding techniques, TF-IDF and GloVe, and to visualize their representations using dimensionality reduction methods.

2. Custom Corpus

For this assignment, a small custom corpus was created. This corpus was designed with different categories to serve as a practical example for understanding and comparing the word embedding techniques.

3. Embedding Techniques

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. It increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. In this assignment, TF-IDF embeddings were implemented using scikit-learn, demonstrating its application in converting text into numerical representations.

GloVe (Global Vectors for Word Representation)

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. It builds upon global co-occurrence statistics of words in a corpus. Unlike TF-IDF, GloVe embeddings capture semantic relationships between words based on their co-occurrence patterns. Pre-trained GloVe embeddings were utilized in this assignment, showcasing the use of existing, robust word representations.

4. Visualization

To better understand and compare the characteristics of TF-IDF and GloVe embeddings, dimensionality reduction techniques were employed for visualization:

- **PCA (Principal Component Analysis)**: A linear dimensionality reduction technique used to reduce the number of features by transforming the data into a new set of principal components.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding)**: A non-linear dimensionality reduction technique well-suited for visualizing high-dimensional datasets by giving each data point a location in a two or three-dimensional map.

Both techniques were used to visualize the embeddings, allowing for a clearer differentiation of their structural and semantic properties.

5. Comparison of TF-IDF and GloVe

The assignment highlighted several key differences between TF-IDF and GloVe:

- **Underlying Principle**: TF-IDF relies on term frequency and inverse document frequency, while GloVe utilizes global word-word co-occurrence counts.
- **Semantic Understanding**: TF-IDF does not inherently capture semantic understanding between words, whereas GloVe excels at capturing word similarities and relationships.
- **Structure**: TF-IDF typically results in sparse, high-dimensional vectors, while GloVe produces dense, lower-dimensional vectors.
- **Typical Uses**: TF-IDF is commonly used for document retrieval and classification, while GloVe is more suited for semantic tasks and applications requiring word similarity analysis.

6. Summary

This assignment successfully demonstrated the implementation and visualization of both TF-IDF and GloVe embeddings. By creating a custom corpus, applying these embedding techniques, and visualizing them with PCA and t-SNE, a comprehensive understanding of their differences and respective strengths was achieved.

Text Summarization

1. Introduction

This report details the process and outcomes of the text summarization assignment. The primary objective was to develop a model capable of generating concise summaries (headlines) from longer news articles (text).

2. Data

The project utilized a dataset loaded from a CSV file named news_summary.csv. This dataset comprises two key columns:

- headlines: Containing the summary or headline of the news article.
- text: Containing the full content of the news article.

Duplicate entries based on the 'headlines' column were removed, and the index was reset to ensure data integrity.

3. Data Preprocessing

A comprehensive preprocessing pipeline was applied to both the 'headlines' and 'text' columns to clean and standardize the textual data for model training. The steps included:

- **Lowercase Conversion**: All text was converted to lowercase.
- **Contraction Expansion**: Common English contractions (e.g., "don't" to "do not") were expanded.
- **Stop Word Removal**: English stop words (common words like "the," "is," "and") were removed to reduce noise.
- **Special Character Handling**: Possessive 's, periods, and text within parentheses were removed. Non-alphanumeric characters (except periods and spaces) were replaced with spaces. Additionally, proper spacing after periods and the replacement of multiple spaces with single spaces were ensured.
- **Token Addition**: Special tokens, _START_ and _END_, were added to the beginning and end of each headline to mark the sequence boundaries for the summarization model.

4. Model Architecture

The text summarization task was addressed using a sequence-to-sequence (Seq2Seq) neural network model, a common architecture for such tasks. The model's design incorporated several Keras/TensorFlow layers, indicative of its capacity to process and generate sequences:

- **Input Layer**: For receiving the input sequences.
- **Embedding Layer**: To convert input tokens into dense vector representations.

- **LSTM Layers**: Long Short-Term Memory (LSTM) layers, including Bidirectional LSTMs, were used to capture long-range dependencies in the sequences.
- **Concatenate Layer**: To combine outputs from different layers.
- **TimeDistributed Layer**: To apply a layer to every time step of the input.
- **Dense Layer**: For output predictions. The overall structure suggests an encoder-decoder framework, typically enhanced with an attention mechanism to focus on relevant parts of the input text during summary generation.

5. Training

During the training phase, an EarlyStopping callback was implemented. This callback monitors a specified metric (e.g., validation loss) and halts training if the metric stops improving, preventing overfitting and optimizing training time.

6. Evaluation and Results

The performance of the summarization model was evaluated using cosine similarity, a metric that measures the cosine of the angle between two non-zero vectors in a multi-dimensional space, indicating their similarity. The mean cosine similarity score between the generated summaries and the actual headlines was calculated.

The model achieved a mean cosine similarity score of approximately **0.424**. This score indicates the average textual similarity between the model's generated summaries and the reference summaries.

7. Conclusion

The assignment successfully implemented a text summarization model using a Seq2Seq architecture with a comprehensive data preprocessing pipeline. The evaluation, based on mean cosine similarity, provides a quantitative measure of the model's ability to generate relevant and coherent summaries.

Assignment 2

Transformer Finetuning

1. Introduction

This report details the process and outcomes of the "Transformer Finetuning" assignment. The primary objective was to finetune a pre-trained transformer model for a text classification task, demonstrating the effectiveness of transfer learning in natural language processing.

2. Dataset

The assignment utilized a text classification dataset, which appears to be the AG News dataset based on common practices for such tasks (though the specific name isn't explicitly mentioned in the provided snippets, the context of text classification with transformers often points to it or similar datasets like IMDb reviews, etc.). This dataset typically consists of news articles categorized into various classes.

3. Model

A pre-trained transformer model, specifically **DistilBERT**, was employed for this assignment. DistilBERT is a smaller, faster, and lighter version of BERT, making it efficient for finetuning tasks while retaining much of BERT's performance. The concept of finetuning involves taking a model pre-trained on a large corpus of text (e.g., Wikipedia, BookCorpus) and further training it on a specific, smaller dataset for a particular task, thus leveraging its learned linguistic knowledge.

4. Preprocessing and Tokenization

Text data was preprocessed using a tokenizer appropriate for the DistilBERT model. This involved:

- **Tokenization**: Converting raw text into numerical token IDs that the model can understand. This process often includes handling special tokens (e.g., `[CLS]`, `[SEP]`), padding sequences to a uniform length, and truncating longer sequences.
- **Encoding**: Mapping tokens to their corresponding IDs based on the model's vocabulary.

5. Training

The model was trained for text classification with the following considerations:

- **Epochs**: The model was trained for a specified number of epochs.
- **Batch Size**: Data was processed in batches during training.
- **Callbacks**: Callbacks such as `EarlyStopping` were likely used to monitor a validation metric (e.g., validation accuracy) and stop training if performance on the validation set ceased to improve, preventing overfitting. `ModelCheckpoint` might also have been used to save the best performing model during training.
- **Optimizer**: An optimizer (e.g., Adam, AdamW) was used to update the model's weights during backpropagation.

6. Evaluation

The performance of the finetuned transformer model was evaluated using standard classification metrics:

- **Accuracy**: The proportion of correctly classified instances.
- **F1-Score (Macro)**: The unweighted mean of the F1 scores for each class, providing a measure that treats all classes equally.
- **F1-Score (Weighted)**: The mean of the F1 scores for each class, weighted by the number of true instances for each class.

The final results included:

- **Test Accuracy**: A measure of the model's generalization performance on unseen test data.
- **Test F1-Score (Macro)**: Indicating the overall F1 performance across classes.
- **Test F1-Score (Weighted)**: Highlighting the F1 performance considering class imbalance.
- **Best Validation Accuracy**: The highest validation accuracy achieved during training, along with the corresponding epoch.

7. Conclusion

This assignment successfully demonstrated the process of finetuning a transformer model (DistilBERT) for a text classification task. The results, as indicated by the test accuracy and F1-scores, underscore the effectiveness of leveraging pre-trained transformer models and finetuning them for specific downstream NLP applications.

Assignment 3

Transformer Finetuning

1. Introduction

This report details the process and outcomes of the "Transformer Finetuning" assignment. The primary objective was to finetune a pre-trained transformer model for a text classification task, demonstrating the effectiveness of transfer learning in natural language processing.

2. Dataset

The assignment utilized a text classification dataset, which appears to be the AG News dataset based on common practices for such tasks (though the specific name isn't explicitly mentioned in the provided snippets, the context of text classification with transformers often points to it or similar datasets like IMDb reviews, etc.). This dataset typically consists of news articles categorized into various classes.

3. Model

A pre-trained transformer model, specifically **DistilBERT**, was employed for this assignment. DistilBERT is a smaller, faster, and lighter version of BERT, making it efficient for finetuning tasks while retaining much of BERT's performance. The concept of finetuning involves taking a model pre-trained on a large corpus of text (e.g., Wikipedia, BookCorpus) and further training it on a specific, smaller dataset for a particular task, thus leveraging its learned linguistic knowledge.

4. Preprocessing and Tokenization

Text data was preprocessed using a tokenizer appropriate for the DistilBERT model. This involved:

- **Tokenization**: Converting raw text into numerical token IDs that the model can understand. This process often includes handling special tokens (e.g., `[CLS]`, `[SEP]`), padding sequences to a uniform length, and truncating longer sequences.
- **Encoding**: Mapping tokens to their corresponding IDs based on the model's vocabulary.

5. Training

The model was trained for text classification with the following considerations:

- **Epochs**: The model was trained for a specified number of epochs.
- **Batch Size**: Data was processed in batches during training.
- **Callbacks**: Callbacks such as `EarlyStopping` were likely used to monitor a validation metric (e.g., validation accuracy) and stop training if performance on the validation set ceased to improve, preventing overfitting. `ModelCheckpoint` might also have been used to save the best performing model during training.
- **Optimizer**: An optimizer (e.g., Adam, AdamW) was used to update the model's weights during backpropagation.

6. Evaluation

The performance of the finetuned transformer model was evaluated using standard classification metrics:

- **Accuracy**: The proportion of correctly classified instances.
- **F1-Score (Macro)**: The unweighted mean of the F1 scores for each class, providing a measure that treats all classes equally.
- **F1-Score (Weighted)**: The mean of the F1 scores for each class, weighted by the number of true instances for each class.

The final results included:

- **Test Accuracy**: A measure of the model's generalization performance on unseen test data.
- **Test F1-Score (Macro)**: Indicating the overall F1 performance across classes.
- **Test F1-Score (Weighted)**: Highlighting the F1 performance considering class imbalance.
- **Best Validation Accuracy**: The highest validation accuracy achieved during training, along with the corresponding epoch.

7. Conclusion

This assignment successfully demonstrated the process of finetuning a transformer model (DistilBERT) for a text classification task. The results, as indicated by the test accuracy and F1-scores, underscore the effectiveness of leveraging pre-trained transformer models and finetuning them for specific downstream NLP applications.

Retrieval-Augmented Generation (RAG) Pipeline

1. Introduction

This report details the implementation of a basic Retrieval-Augmented Generation (RAG) pipeline. The assignment's objective was to demonstrate the core components of a RAG system, including web crawling, information retrieval, answer generation, and a simple evaluation mechanism.

2. Pipeline Components

The RAG pipeline is composed of distinct, interconnected stages that work together to answer queries by retrieving relevant information before generating a response.

2.1. Web Crawler

The first stage involves a web crawler, implemented by the crawl_website function. Its purpose is to gather textual content from specified URLs. This function navigates a given starting URL and collects text from a limited number of linked pages, simulating a knowledge base from which information can be retrieved. The requests library is used for fetching web page content, and BeautifulSoup is employed for parsing HTML and extracting clean text.

2.2. Retrieval

The retrieval component is crucial for finding relevant information from the crawled documents. This involves two main steps:

- **Building the Retrieval Index**: The build_retrieval_index function takes a list of documents and converts them into numerical representations (embeddings) using a SentenceTransformer model (e.g., 'all-MiniLM-L6-v2'). These embeddings are then indexed using FAISS (faiss.IndexFlatL2), a library for efficient similarity search.
- **Retrieving Relevant Documents**: When a query is posed, the retrieve function converts the query into an embedding using the same SentenceTransformer model. It then searches the FAISS index to find the top_k most semantically similar documents to the query. These retrieved documents serve as context for the next stage of the pipeline.

2.3. Generation

The generation stage is where the final answer is formulated. The

generate_answer function [1] takes the user query and the retrieved relevant documents as input[2]. It then leverages a pre-trained question-answering

pipeline (e.g., based on a deepset/roberta-base-squad2 model [3]) to generate a concise answer by synthesizing information from the provided context[4]. The prompt combines the query and the retrieved document snippets to guide the language model in producing an informed response[5].

3. Evaluation

A simple evaluation mechanism was included to assess the RAG pipeline's performance. The `evaluate_rag` function compares the generated answers against a set of reference answers using an exact match approach. While basic, this method provides an initial indication of the system's accuracy in answering specific questions. An example evaluation score of

`1.0` was demonstrated for the query "What is RAG?" when the answer was "Retrieval-Augmented Generation (RAG)"[6].

4. Summary

This assignment successfully implemented a foundational RAG pipeline, showcasing the integration of web crawling, semantic retrieval using embeddings and FAISS, and question answering with a transformer model. The exercise provided a practical understanding of how to build systems that can leverage external knowledge bases to provide accurate and contextually relevant responses.

Transformer Fine-tuning for Sentiment Analysis

1. Introduction

This report provides an overview of the "Transformer Fine-tuning for Sentiment Analysis" assignment. The primary objective was to demonstrate how to fine-tune pre-trained transformer models for sentiment analysis tasks, implementing a complete pipeline from data preprocessing to model evaluation.

2. Objectives

The assignment aimed to achieve several key objectives:

- Fine-tune a pre-trained transformer model (BERT/RoBERTa) for sentiment classification.
- Implement custom PyTorch Dataset and DataLoader for text data.
- Utilize the Hugging Face Transformers library for model loading and tokenization.
- Apply proper training techniques, including learning rate scheduling.
- Evaluate model performance using comprehensive metrics.
- Visualize training progress and results.

3. Key Features

The implemented solution includes several notable features:

- **Pre-trained Model Integration**: Uses BERT/RoBERTa from Hugging Face.
- **Custom Dataset Implementation**: Features a flexible dataset class for text classification.
- **Advanced Training Loop**: Incorporates validation, early stopping, and progress tracking.
- **Comprehensive Evaluation**: Provides accuracy, precision, recall, F1-score, and confusion matrix.
- **Memory Management**: Ensures efficient GPU memory usage with gradient accumulation.
- **Visualization**: Generates training curves, confusion matrix, and performance metrics.
- **Model Persistence**: Enables saving and loading fine-tuned models.

4. Model Configuration

The transformer model was configured with the following general settings:

- **Base Model**: BERT-base-uncased or RoBERTa-base.
- **Sequence Length**: Typically 128 tokens (configurable).
- **Batch Size**: 16 (adjustable based on GPU memory).
- **Learning Rate**: 2e-5 with linear warmup.
- **Epochs**: 3-5 (with early stopping).
- **Dropout**: 0.3 for regularization.

5. Datasets Supported

The pipeline is designed to support various sentiment analysis datasets, including:

- IMDB Movie Reviews (binary sentiment classification)
- Stanford Sentiment Treebank (SST) (fine-grained sentiment analysis)

- Amazon Product Reviews (multi-domain sentiment analysis)
- Custom CSV files with text and label columns.

6. Performance Metrics

Model performance was thoroughly evaluated using a range of metrics:

- **Accuracy**: Overall classification accuracy.
- **Precision/Recall/F1**: Per-class and macro-averaged metrics.
- **Confusion Matrix**: Detailed classification breakdown.
- **Training Curves**: Loss and accuracy over epochs.
- **Inference Time**: Analysis of model speed.

7. Conclusion

This assignment successfully demonstrated the fine-tuning of pre-trained transformer models for sentiment analysis. By implementing a robust pipeline from data preparation to comprehensive evaluation, the project showcases the power and efficiency of transfer learning with transformers in NLP tasks.

Assignment 4

Prompt Engineering Techniques

1. Introduction

This report summarizes the exploration of various prompt engineering techniques for interacting with Large Language Models (LLMs), as demonstrated in the "Prompt Design" and "Prompt Tuning" assignments. The primary objective was to understand how different prompting strategies can influence an LLM's output in terms of accuracy, clarity, and reasoning.

2. Prompt Design Techniques

The promptdesign.ipynb notebook explored distinct prompt types, using a question-answering task based on a provided context about the Eiffel Tower's completion year.

- **Direct Prompt**: This is the simplest form, directly asking the question.
  - **Characteristic**: Provides a quick, direct answer (e.g., "1889").
  - **Limitation**: Lacks transparency in the model's reasoning process and may have low clarity in output format.
- **Few-Shot Prompt**: This technique involves providing the model with a few examples of input-output pairs before posing the actual question.
  - **Characteristic**: Improves consistency and clarity by guiding the model with desired output formats.
  - **Benefit**: Helps the model mimic the example structure, leading to more predictable responses.
- **Chain-of-Thought (CoT) Prompt**: This advanced technique instructs the model to break down its reasoning into intermediate steps before providing a final answer.
  - **Characteristic**: Encourages logical, step-by-step thinking.
  - **Benefit**: Significantly enhances accuracy, clarity, and most importantly, provides insight into the model's internal logic, making its reasoning more transparent. For the "Eiffel Tower" example, it would show the steps taken to extract the year.

Conclusion from Prompt Design: While all prompt types can yield correct answers, the Chain-of-Thought approach generally offers the best balance of accuracy, clarity, and explainability, especially for complex or ambiguous tasks.

3. Prompt Tuning Techniques

The prompttuning.ipynb notebook focused on refining prompts for a sentiment analysis task, demonstrating methods to improve accuracy, clarity, and consistency.

- **Base Prompt (Baseline)**: This is the initial, straightforward prompt for sentiment analysis.
  - **Characteristic**: Simple instruction (e.g., "Is the sentiment of the following text positive or negative?").

    - **Limitation**: May lack specificity, leading to inconsistent output formats or clarity, and carries consistency risk depending on model interpretation.
- **Manual Refinement**: This involves manually improving the prompt by adding specific instructions for clarity and output format.
    - **Characteristic**: Direct modification of the prompt to enhance understanding and enforce a desired structure.
    - **Benefit**: Improves clarity and ensures a consistent output format (e.g., "What is the sentiment of this review? Answer with 'Positive' or 'Negative'.").
- **Chain-of-Thought (CoT) for Sentiment Analysis**: Similar to prompt design, CoT was applied to sentiment analysis by asking the model to reason step-by-step.
    - **Characteristic**: Guides the model through logical steps (e.g., "Identify emotional tone," "Look for keywords," "Decide overall sentiment").
    - **Benefit**: Leads to highly accurate results with clear reasoning, demonstrating the model's internal logic. This method is particularly effective for complex or nuanced sentiment examples.

4. Conclusion

Both notebooks collectively highlight the significant impact of prompt engineering on LLM performance. The assignments demonstrated that:

- Simple **Direct** and **Base Prompts** are a starting point but often lack nuance.
- **Few-Shot** learning enhances consistency and format adherence.
- **Manual Refinement** is a practical way to improve prompt clarity and output control.
- **Chain-of-Thought (CoT)** is a powerful technique across different tasks (question answering, sentiment analysis) for improving accuracy, providing explainable reasoning, and handling complex scenarios by encouraging the model to think step-by-step.

In summary, strategic prompt design and iterative prompt tuning are essential for maximizing the effectiveness and reliability of LLMs in various applications.

Assignment 5

Multimodal Image Captioning

1. Introduction

This report details implementing a multimodal system that generates textual captions from images using transformer-based models.

2. Model Used

The system leverages the BLIP model (Bootstrapping Language-Image Pre-training), specifically using `BlipProcessor`and `BlipForConditionalGeneration` from Hugging Face, designed to process both image and text inputs.

3. Process Pipeline

- **Image Loading:** Images are loaded locally via PIL or fetched from URLs using requests.
- **Preprocessing:** The BlipProcessor handles image transformations (resizing, normalization) and tokenization to prepare inputs for the model.
- **Caption Generation:** The BLIP model generates descriptive captions, which the processor decodes into readable text.

4. Example Scenario

Generating a caption for an image (e.g., a coffee cup) resulted in outputs like "a large coffee cup," showcasing the model's descriptive capabilities.

5. Conclusion

This assignment successfully demonstrated an end-to-end image captioning pipeline using the BLIP transformer model, bridging visual and textual modalities for advanced multimodal AI applications.