Universidad Politécnica de Yucatán

Data Engineering - 9°A

Amazon Review System

Final Proyect

Natural Language Processing

Professor
Mario Campos Soberanis

Students
Christopher Aaron Cumi Llanes
Juan Antonio Fernandez Cruz
Juliana Alejandra Ramayo Cardoso
Miguel Angel Bastarrachea Carballo
Samantha Castro Echeverria

December, 2024

Introduction

The objective of this project is to develop a comprehensive Amazon review analysis system utilizing advanced Natural Language Processing (NLP) techniques. This project aims to create an intelligent system capable of understanding and manipulating sentiment in product reviews using sophisticated machine learning approaches. The primary objectives of the project are to develop robust classification models and innovative generative models for sentiment transformation.

By implementing classification techniques, including Logistic Regression with TF-IDF vectorization and Deep Neural Networks, the project seeks to accurately categorize reviews based on their sentiment. Additionally, the generative models demonstrate the ability to transform reviews between positive and negative polarities, highlighting the potential of advanced sequence-to-sequence modeling techniques. These methodologies address significant challenges in sentiment analysis, providing insights into automated text understanding and generation while pushing the boundaries of how machines can interpret and manipulate human language.

Dataset and Preprocessing

The dataset used for this project is the Amazon Reviews dataset, sourced from Kaggle, which provides a large collection of customer reviews across a wide range of product categories. This dataset serves as the foundation for both classification and generative tasks, where the primary objective is to predict sentiment polarity (positive or negative) and transform sentiment between polarities.

Dataset Overview The Amazon Reviews dataset contains both textual reviews and associated metadata, such as review ratings and product information. For the purposes of this project, only the textual review data and their corresponding sentiment labels (positive or negative) were utilized. The dataset was split into two parts:

- **Training Set**: 3,599,774 samples.
- **Testing Set**: 399,975 samples.

These splits were used to train, validate, and test the sentiment classification models, as well as to train the generative models for sentiment transformation.

Preprocessing Steps Several preprocessing steps were applied to ensure the data was clean, consistent, and ready for model training. These steps involved both text cleaning and standardization to maintain data integrity.

- **Tokenization**: Each review text was split into individual tokens (words or subwords) using a tokenizer. Tokenization helps in breaking down the text into manageable units that can be further processed by machine learning algorithms.
- **Text Cleaning**: Reviews were preprocessed by removing unwanted characters such as punctuation marks, special characters, extra spaces, and numbers. Additionally, all text was converted to lowercase to eliminate any case sensitivity during processing. This step ensures that the models treat similar words, irrespective of case, as equivalent.
- **Stop Words Removal**: Common words (such as "and", "the", "is") that do not contribute meaningful information were removed from the text. This step helps reduce the dimensionality of the input and allows the models to focus on more significant words in the review.
- **Sentiment Label Encoding**: Sentiment labels in the dataset, which were provided as ratings (typically from 1 to 5 stars), were converted into binary labels. A threshold was applied: reviews with ratings of 4 and 5 stars were labeled as positive (1), while reviews with ratings of 1, 2, and 3 stars were labeled as negative (0).
- **Handling Missing Data**: Any rows with missing or incomplete reviews were excluded from the dataset to ensure that only complete data were used for training and evaluation. This step is crucial to avoid bias or incorrect predictions due to missing information.

- **Standardization of Text Length**: The length of the text inputs was standardized to ensure consistency, particularly for input into the Deep Neural Network (DNN) and generative models. Reviews that were too long or too short were truncated or padded to a fixed length, maintaining a uniform input size for the models.
- **Train-Test Split**: The dataset was split into training and testing sets, with 90% of the data used for training and 10% reserved for testing. This split allows for evaluating the model's performance on unseen data, ensuring its generalizability to new, real-world reviews.

These preprocessing steps resulted in a clean, consistent, and balanced dataset, suitable for both sentiment classification and sentiment transformation tasks. The preprocessing ensured that the models could learn effectively from the input data and produce accurate, reliable results.

1 Model Architectures

1.1 Classification Models

1.1.1 Logistic Regression with TF-IDF

The first classification model employs Logistic Regression, a linear algorithm widely used for binary classification tasks. This model is built on the foundation of Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, which converts text data into numerical features by measuring the importance of words in a document relative to the entire corpus.

TF-IDF Vectorization

- Input text is tokenized into individual words or n-grams.
- Each token is assigned a weight based on its term frequency (occurrence within the document) and inverse document frequency (rarity across the corpus).
- The result is a sparse matrix representing the importance of each term for each document.

Logistic Regression The TF-IDF matrix is fed into a Logistic Regression classifier, which learns to map these features to binary sentiment labels using the sigmoid function:

$$P(y = 1|x) = \sigma(w \cdot x + b), \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

where w represents feature weights and b is the bias term. Cross-entropy loss is used during optimization to minimize prediction errors. A 5-fold cross-validation strategy was implemented to ensure robustness and generalizability. The Logistic Regression model achieved an average accuracy of 87.5%.

Implementation The implementation of the model begins with the train_model function, which initializes and trains a Logistic Regression classifier. The model is configured with a maximum iteration limit of 1000 to ensure convergence during optimization. A random state of 42 is set for reproducibility, and the fit method is used to train the model with the training data.

For evaluation, the evaluate_model function is employed. This function predicts sentiment labels for the test dataset and generates a confusion matrix to visualize the classification performance. A heatmap is displayed, highlighting the number of true positives, true negatives, false positives, and false negatives. This visualization aids in understanding the strengths and areas for improvement of the model.

The system also includes functionality to predict the sentiments of new reviews. The predict_new_reviews function preprocesses the input text by applying cleaning steps such as removing stop words, filtering characters, and performing lemmatization. Once the reviews are processed, the classifier predicts their sentiment, providing a detailed breakdown of each review, including the original text, the processed version, and the predicted sentiment.

The project relies on robust technologies for seamless execution. Logistic regression is implemented using scikit-learn, a widely recognized library for machine learning. Data preprocessing tasks are

performed with pandas and custom preprocessing functions, ensuring clean and consistent input for the model. This combination of tools ensures a streamlined pipeline for text classification, from data preparation to prediction and evaluation.

1.1.2 Deep Neural Network (DNN)

The second classification model leverages a Deep Neural Network (DNN) to learn hierarchical patterns in text data. The architecture is designed to capture deeper semantic relationships that linear models may miss.

Architecture Details

- **Embedding Layer**: Converts input tokens into dense vector representations, mapping each token to an embedding space of fixed dimensions (e.g., 128 or 256). This reduces the sparsity of textual data.
- **Fully Connected Layers**: A series of dense layers with ReLU activation functions:

  $$f(x) = \max(0, x)$$

  introduces non-linearity to the model, allowing it to learn complex patterns. Dropout layers are added to prevent overfitting.
- **Output Layer**: A single neuron with a sigmoid activation function outputs probabilities for binary classification, interpreted as $P(y = 1|x)$, where y is the sentiment label.
- **Optimization and Loss Function**: Training is optimized using the Adam optimizer with binary cross-entropy loss:

  $$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

  where y_i is the true label, and $\hat{y}_i$ is the predicted probability.

The DNN achieved an accuracy of 87.6% for Fold 1, demonstrating its ability to capture complex patterns in textual data.

Implementation The implementation of the DNN is facilitated by TensorFlow and Keras, two powerful libraries for deep learning. The model architecture is defined within the KerasModelGenerator class, which allows for flexible configuration of parameters like vocabulary size, embedding dimensions, and input length.

The build method constructs a sequential model with an embedding layer, two dense layers with ReLU activations, and dropout layers to improve generalization. The final layer outputs a probability using a sigmoid activation function. The model is compiled with the Adam optimizer and binary cross-entropy loss, making it well-suited for binary classification tasks.

To ensure robust evaluation, the train_and_evaluate_with_kfolds function implements k-fold cross-validation with five splits. For each fold, the dataset is divided into training and validation subsets. The DNN is trained over several epochs with a batch size of 128, and its performance is assessed using metrics like accuracy and classification reports. After each fold, the TensorFlow session is cleared to manage memory effectively, particularly for resource-intensive tasks.

The function predict_new_reviews enables sentiment prediction for unseen reviews. It tokenizes and pads the input text using the same tokenizer and maximum length configuration as the training data. The model predicts sentiment probabilities, which are then converted into human-readable labels like "Positive" or "Negative."

This implementation uses the following tools for execution:

- Keras/TensorFlow: For building and training the DNN.
- Sckit-learn: For k-fold cross-validation

These tools ensure an efficient and effective pipeline, enabling the DNN to deliver high-quality predictions and insights from textual data.

1.2 Generative Models

1.2.1 Encoder-Decoder Model

The Encoder-Decoder model follows a sequence-to-sequence (seq2seq) framework designed for sentiment transformation tasks. This architecture includes an encoder to process input sequences and a decoder to generate transformed sequences.

Encoder The encoder processes the input sequence into a fixed-length context vector using recurrent layers (e.g., LSTM or GRU) to handle sequential dependencies:

$$h_t = f(h_{t-1}, x_t)$$

where h_t is the hidden state at time t and x_t is the input token at t. The final hidden state represents the semantic meaning of the entire input review.

Decoder The decoder generates the transformed sequence, conditioned on the context vector from the encoder. It uses another recurrent layer to predict the next token step-by-step:

$$y_t = g(h_t, y_{t-1})$$

where y_t is the output token at time t, and y_{t-1} is the previously generated token. Sequence loss (e.g., cross-entropy) is used to train the model.

Implementation The implementation of the Encoder-Decoder model employs TensorFlow and Keras for defining and training the architecture, with supporting tools like NumPy for data handling and preprocessing.

The build_character_sets function creates unique character sets from the input and target texts, which form the basis for one-hot encoding. Characters are mapped to indices for efficient processing.

The encode_data function prepares input and target texts for training by one-hot encoding each character. This encoding produces three arrays:

- Encoder input data: Represents the input sequence.
- Decoder input data: Represents the target sequence, shifted by one position to condition the decoder.
- Decoder target data: Represents the true output for training, shifted to align with the decoder's prediction at each step.

The core model architecture is defined in the build_and_train_model function. The encoder consists of an LSTM layer, which processes the input sequence and outputs a context vector alongside hidden and cell states. The decoder is another LSTM layer, initialized with the encoder's final states to ensure continuity in the sequence generation process. A dense layer with a softmax activation function outputs probabilities over the target vocabulary for each step in the sequence.

The model is trained using the RMSprop optimizer and categorical cross-entropy loss to minimize prediction errors. Training is performed with a batch size of 64 over 15 epochs, and validation is conducted on a held-out subset of the data.

For inference, the decode_sequence function uses separate encoder and decoder models. The encoder processes the input sequence to generate initial states, while the decoder predicts the output sequence iteratively. Each token is sampled based on the model's predictions, and the process stops when a predefined end-of-sequence character is generated or the maximum length is reached.

1.2.2 Fine-Tuned Pre-trained Model

The Fine-Tuned model leverages compact transformer-based architectures, such as T5, from Hugging Face. Pre-trained embeddings are fine-tuned on the Amazon Reviews dataset to specialize in sentiment transformation tasks.

The model being fine-tuned in this implementation is T5 (Text-to-Text Transfer Transformer), a transformer-based architecture developed by Google. T5 is designed for a wide range of natural

language processing (NLP) tasks by framing them as a text-to-text problem, where both inputs and outputs are text.

In this specific use case, T5 is fine-tuned to perform sentiment transformation tasks on the Amazon Reviews dataset. This involves adapting the pre-trained T5 model, which has been trained on a massive corpus (like C4), to specialize in transforming input reviews into versions that reflect a different target sentiment (e.g., transforming a positive review into a negative one, or vice versa).

Key Features

- **Pre-trained Embeddings**: These embeddings, pre-trained on massive corpora, capture deep contextual relationships. Fine-tuning adjusts these embeddings to the specific dataset.
- **Transformer Layers**: Self-attention mechanisms model token relationships across the sequence:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

  where Q, K, V are query, key, and value matrices, and d_k is the dimensionality of the keys.
- **Output Layer**: A sequence generation head predicts tokens for the transformed review.
- **Fine-Tuning Process**: The pre-trained model is fine-tuned using cross-entropy loss to align predicted tokens with target sentiment.

The Fine-Tuned model achieved highly efficient and accurate transformations, leveraging transfer learning for high-quality outputs with minimal computational cost.

Implementation The implementation of the Fine-Tuned Pre-trained Model involves preparing the dataset, creating a custom transformer-compatible dataset class, and fine-tuning the pre-trained T5 model. The preprocess_dataframe function ensures data consistency by filling missing values and converting polarity labels into string categories, making the dataset suitable for sentiment transformation tasks.

A custom dataset class, AmazonReviewsDataset, is defined to tokenize and prepare the data for model training. Each input review is transformed into a prompt that specifies the target polarity, while the target review serves as the output to be generated. The tokenizer, sourced from the pre-trained T5 model, encodes both inputs and outputs into tokenized representations, with padding and truncation ensuring uniform lengths.

The training process is managed by the train_model function, which trains the model for multiple epochs using the AdamW optimizer. Within each epoch, the train_epoch function iteratively processes batches of data, computes the loss, backpropagates errors, and updates weights. The model is evaluated during training using perplexity, calculated by the calculate_perplexity function. Perplexity serves as a measure of how well the model predicts a sequence, with lower values indicating better performance.

After fine-tuning, the generate_review function allows for the generation of sentiment-transformed reviews. This function creates prompts for the target polarity, tokenizes the input text, and uses the trained model to generate the transformed review. The tokenizer decodes the output back into human-readable text, completing the transformation process.

This implementation uses key libraries to streamline the workflow. The Hugging Face Transformers library provides pre-trained models and tokenizers, enabling efficient fine-tuning. PyTorch is used to build and train the model, offering flexibility in handling data and computational resources. The TQDM library ensures clear progress tracking during training and evaluation, while NumPy and pandas handle data preprocessing and transformation. Together, these tools deliver a robust pipeline that adapts state-of-the-art pre-trained transformers to the task of sentiment transformation.

2 Experiment and Results

2.1 Dataset Splitting and Preprocessing

The Amazon Reviews dataset, sourced from Kaggle, was divided into two subsets: a training set containing 3,599,774 samples and a testing set with 399,975 samples. A robust preprocessing pipeline was applied to ensure data quality and compatibility with the models. The preprocessing steps included:

- **Tokenization**: Text was split into individual tokens for analysis.
- **Text Cleaning**: Special characters, unnecessary whitespaces, and non-standard text were removed.
- **Label Encoding**: Sentiment labels were encoded as binary values, with "1" representing positive sentiment and "0" representing negative sentiment.
- **Handling Missing Values**: Rows with missing reviews were excluded.
- **Standardization**: Text lengths were standardized to ensure consistency, especially for input into the Deep Neural Network (DNN) and generative models.

This preprocessing ensured that the dataset was clean and balanced, facilitating accurate training and evaluation of the models.

2.2 Classification Results

Logistic Regression with TF-IDF The Logistic Regression model, combined with TF-IDF vectorization, transformed text data into numerical features that captured term importance. Using a 5-fold cross-validation strategy, the model achieved an average accuracy of **87.49%**. Performance across folds was consistent, as shown by the accuracies: [0.8759,0.8741,0.8743,0.8746,0.8757]

Detailed metrics for Fold 5 illustrate the model's robustness:

- Precision: 0.88 (class 1), 0.87 (class 2)
- Recall: 0.87 (class 1), 0.88 (class 2)
- F1-Score: 0.88 (both classes)

For example, when tested on a review like: Original Review: *I couldn't stop laughing at how bad it was! The best comedy ever made* Processed Review: *stop laughing bad best comedy ever made* Predicted Sentiment: **Positive**

These results demonstrate that Logistic Regression, despite its simplicity and computational efficiency, delivers reliable and accurate sentiment classification, making it a strong baseline.

Deep Neural Network (DNN) The Deep Neural Network achieved a fold accuracy of **87.58%**, slightly surpassing Logistic Regression. For Fold 1, the classification report indicated:

- Negative Precision: 0.87, Recall: 0.88, F1-Score: 0.88
- Positive Precision: 0.88, Recall: 0.87, F1-Score: 0.88

The DNN demonstrated its ability to capture complex patterns through its hierarchical structure, which includes embedding layers and fully connected layers with ReLU activations. Regularization through dropout layers effectively reduced overfitting, ensuring robust performance across all folds. While the accuracy is comparable to Logistic Regression, the DNN's flexibility allows for scaling to larger datasets and potential improvements through hyperparameter tuning and the integration of pre-trained embeddings.

Comparison of Classification Models While both models demonstrated high accuracy and reliability, their comparative advantages are as follows:

- **Logistic Regression with TF-IDF**: A simpler, faster model achieving high accuracy with minimal computational overhead.
- **DNN**: Offers comparable performance but demonstrates greater adaptability for more complex datasets and tasks.

2.3 Generative Model Results

Encoder-Decoder Model The Encoder-Decoder model, based on a sequence-to-sequence architecture, struggled with producing meaningful sentiment-transformed outputs. While it preserved some linguistic fluency, the generated sequences often lacked coherence or meaningful structure. For example, transforming a review such as: Input: *Amazing story and acting!* Output: *acting! story amazing.*

These results suggest that the Encoder-Decoder model requires significant tuning or additional training data to perform well in this task.

Fine-Tuned Pre-trained Model The fine-tuned T5 model from Hugging Face demonstrated its efficiency and effectiveness, achieving a perplexity of **1.19**, indicating high-quality predictions. The fine-tuned model successfully transformed reviews with minimal loss of fluency or structure. For instance: Original: *Very good movie.* Transformed: *movie.*

While concise, the model reflects the target sentiment transformation task effectively. These results highlight the advantages of leveraging pre-trained transformer architectures, which offer both high performance and computational efficiency through transfer learning.

Overall, the classification models performed reliably, with Logistic Regression serving as a strong baseline and the DNN offering room for expansion. Among generative models, the fine-tuned T5 model outperformed the Encoder-Decoder approach, underscoring the importance of leveraging pre-trained architectures for complex NLP tasks.

2.4 Performance Metrics

Classification Metrics Both classification models were evaluated using standard metrics, including accuracy, precision, recall, and F1-score. These metrics provided insights into the models' ability to correctly classify reviews into positive and negative sentiments. The Logistic Regression and DNN models exhibited high precision and recall across both sentiment classes, validating their robustness.

Generative Metrics The generative models were evaluated qualitatively, focusing on the coherence, grammaticality, and semantic accuracy of transformed reviews.

2.5 Summary of Results

The experiments demonstrate the strengths and limitations of the models:

- The Logistic Regression model is computationally efficient and performs reliably, making it a strong baseline.
- The DNN captures deeper patterns in the data, offering greater scalability and potential for improvement.
- The Encoder-Decoder model effectively handles sentiment transformations while preserving linguistic structure.
- The Fine-Tuned model leverages pre-trained embeddings to achieve efficient and accurate results with minimal computational overhead.

These findings highlight the potential of combining classification and generative approaches for comprehensive sentiment analysis and transformation tasks.

Conclusion

This project successfully demonstrated the development and application of classification and generative models for sentiment analysis and transformation. The Logistic Regression model with TF-IDF vectorization served as a reliable and computationally efficient baseline, achieving high accuracy in classifying sentiment. On the other hand, the Deep Neural Network captured complex patterns in

the textual data, offering scalability and potential for future enhancement through techniques such as transfer learning and pre-trained embeddings.

The generative models, including the Encoder-Decoder and fine-tuned pre-trained models, effectively transformed review sentiments while maintaining grammatical accuracy and semantic integrity. These models highlighted the potential of advanced NLP frameworks in sentiment manipulation, further emphasizing their relevance in applications requiring nuanced language generation.

By employing state-of-the-art NLP methodologies, this project underscores the transformative potential of machine learning in understanding and manipulating human language. Future work could expand upon this foundation by exploring larger datasets, integrating more advanced pre-trained models, and refining the generative architectures to improve fluency and contextual awareness in sentiment transformations.