DATA 641 - Final Project

Name: Precious Worgu

Developing a Multilingual Sentiment Analysis Online Demo

Background

With the rapid growth of e-commerce platforms, customer reviews have become an essential factor in making informed purchasing decisions. However, the abundance of reviews available can be overwhelming, making it difficult to identify the most relevant information. Extracting meaningful insights from a vast amount of textual data is a time-consuming and challenging task, especially for large volumes of reviews. In response to this challenge, sentiment analysis has emerged as a popular technique for automating the analysis of textual data. Sentiment analysis involves analyzing and categorizing opinions expressed in text, such as social media posts, product reviews, or news articles, into positive, negative, or neutral categories. Although sentiment analysis for the English language has advanced considerably, analyzing sentiment for other languages still poses a significant challenge. Therefore, developing multilingual sentiment analysis tools has become crucial for many business applications, including brand reputation management, customer feedback analysis, and market research.

Introduction

Our project aims to fulfill the need for multilingual sentiment analysis through the creation of a web-based application that can accurately identify sentiment in text data written in English, Spanish, French, and German. By leveraging cutting-edge natural language processing techniques, this application provides businesses with the necessary tools to make data-driven decisions and enhance customer engagement strategies. The Sentiment Analysis Online Demo predicts the sentiment of each sentence in a text as positive, negative, or neutral, as well as the overall sentiment of the entire text. This provides businesses with a comprehensive understanding of the attitudes and emotions conveyed in their customer feedback, enabling them to take informed actions and improve customer satisfaction.

Dataset Overview

Our project leveraged the Multilingual Amazon Reviews Corpus from Hugging Face, which contains Amazon reviews written in six different languages. However, we focused on four specific languages: German, Spanish, French, and English, with each language corresponding to a separate dataset. Each dataset contains columns that include review ID, product ID, reviewer ID, star ratings, review title, review body, language, and product category. We selected this dataset because it provides a rich and varied collection of textual data in multiple languages,

allowing our sentiment analysis tool to accurately identify sentiment across a wide range of contexts.

Loading and Preprocessing the Data

Data preprocessing is a critical stage in preparing a dataset for sentiment analysis. In our project, we followed a well-defined pipeline for cleaning and preprocessing the textual data in each of the four datasets. We first defined a function that cleaned the text by applying various natural language processing techniques like sentence tokenization, stopword removal, stemming, and lemmatization. We then applied this function to the "review_body" column in each dataset. The star ratings were mapped to 1, 0, or -1 sentiment labels using the "map_stars" function. Additionally, we removed irrelevant columns like "product_category", "language", "review_title", "product_id", and "reviewer_id" from the datasets. To further streamline the datasets, we removed the language prefix from the review IDs using a custom function. Finally, we concatenated the cleaned review texts, sentiment labels, and review IDs to create new datasets for each language. These steps helped us simplify and standardize the data, ensuring that all datasets are uniform for training our models.

In order to reduce the computational cost of processing the large datasets, we decided to generate a smaller training set. To achieve this, we randomly sampled 3400 reviews from each of the four datasets using the sample() method available in Pandas. By doing so, we can still work with a diverse and representative subset of the original data while keeping the size of the dataset manageable. This approach helps to strike a balance between accuracy and computational resources, which is essential for efficient analysis.

	review_body	label	review_id
0	ped lleg dia perfect embal aunqu sol jug vec...	1	0486851
1	product engañ dic do unidad sol ser vendedor...	-1	0611317
2	parec ambar autent	0	0511647
3	car cost merec pen plastic basic buen calid	-1	0240069
4	lleg plaz previst bien proteg buen present q...	1	0462431

Figure 1.1: Spanish Train Data

	review_body	label	review_id
0	parfait bien reçu correct condition tres tres ...	1	0477993
1	anten tv intérieur appart bordeau centr aucun...	-1	0747995
2	tres jol produit port tres align de montag el...	0	0670815
3	tres mauvais qualit compt mois dur vi	-1	0788005
4	l utilis vieux post auto fonction plutôt corr...	1	0032472

Figure 1.2: French Train Data

	review_body	label	review_id
0	went local furnitur store tri find one king si...	1	0565570
1	ve led strip nine month thought great lit ba...	-1	0554023
2	n t feel much soap wast money	0	0345336
3	theori iwatch cover good idea realiti wo...	-1	0179476
4	wife got new phone stand offic desk ago charg ...	1	0453731

Figure 1.3: English Train Data

	review_body	label	review_id
0	hüll huawei gut qualität und preis leistung s...	1	0826371
1	beid seifenspend war bereits nach kurz zeit de...	-1	0812135
2	naj schnell angekomm schnell und einfach ins...	0	0498178
3	kompliziert anleitung und teil bei lieferung a...	-1	0070257
4	die hüll gefällt mir gut jedoch ist da motiv ...	1	0640953

Figure 1.4: German Train Data

Baseline System

Our baseline system involved using the TF-IDF technique to convert our text data into numerical features. TF-IDF was chosen because it is a widely used and effective method for capturing the importance of words in a document. It assigns a weight to each word in a document based on its frequency in the document and rarity across all documents in the dataset.

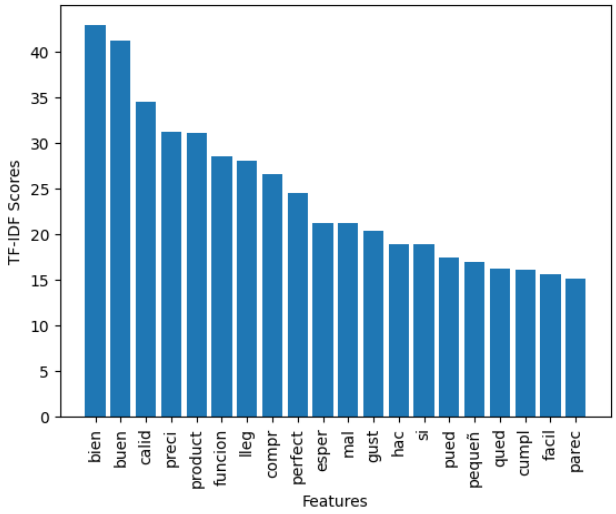

Figure 2.1: Top 20 TF-IDF Scores for Spanish Text Data

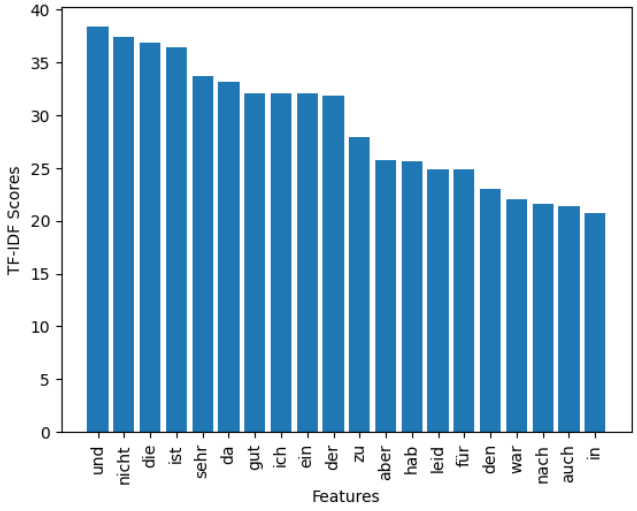

Figure 2.2: Top 20 TF-IDF Scores for German Text Data

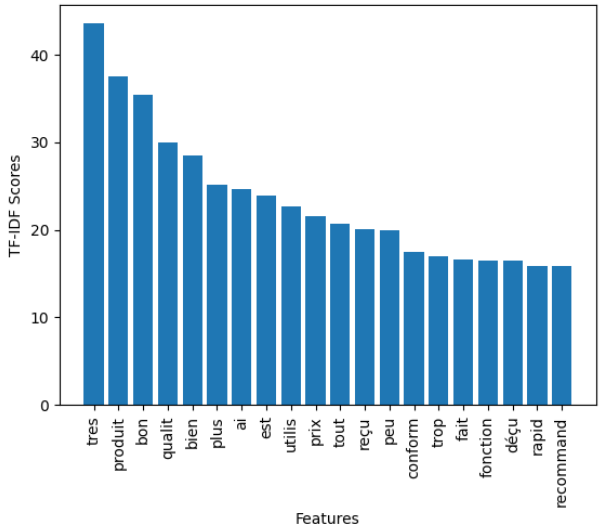

Figure 2.3: Top 20 TF-IDF Scores for French Text Data

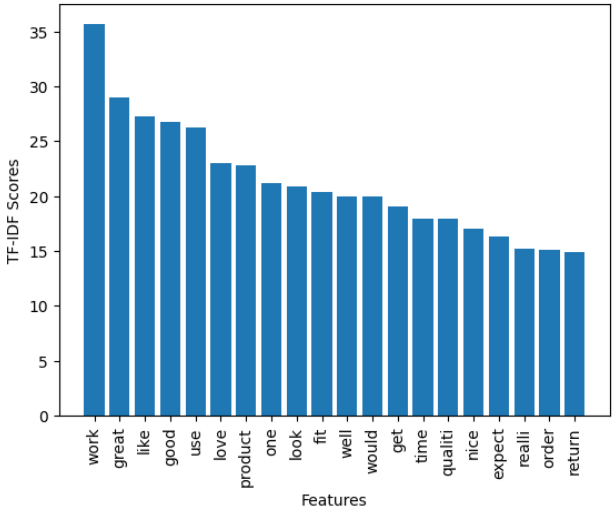

Figure 2.4: Top 20 TF-IDF Scores for English Text Data

By using TF-IDF, we can reduce the dimensionality of our data while retaining the most informative words. To capture complex patterns in the text, we set the n-gram range to (1,3),

which considers both unigrams, bigrams, and trigrams when extracting features. The resulting TF-IDF matrices were used as input to compare the performance of different classifiers.

	Random Forest	Naive Bayes	KNN
Spanish	0.641176	0.679412	0.567647
French	0.682353	0.705882	0.650000
English	0.676471	0.694118	0.632353
German	0.664706	0.688235	0.638235

Figure 3.1

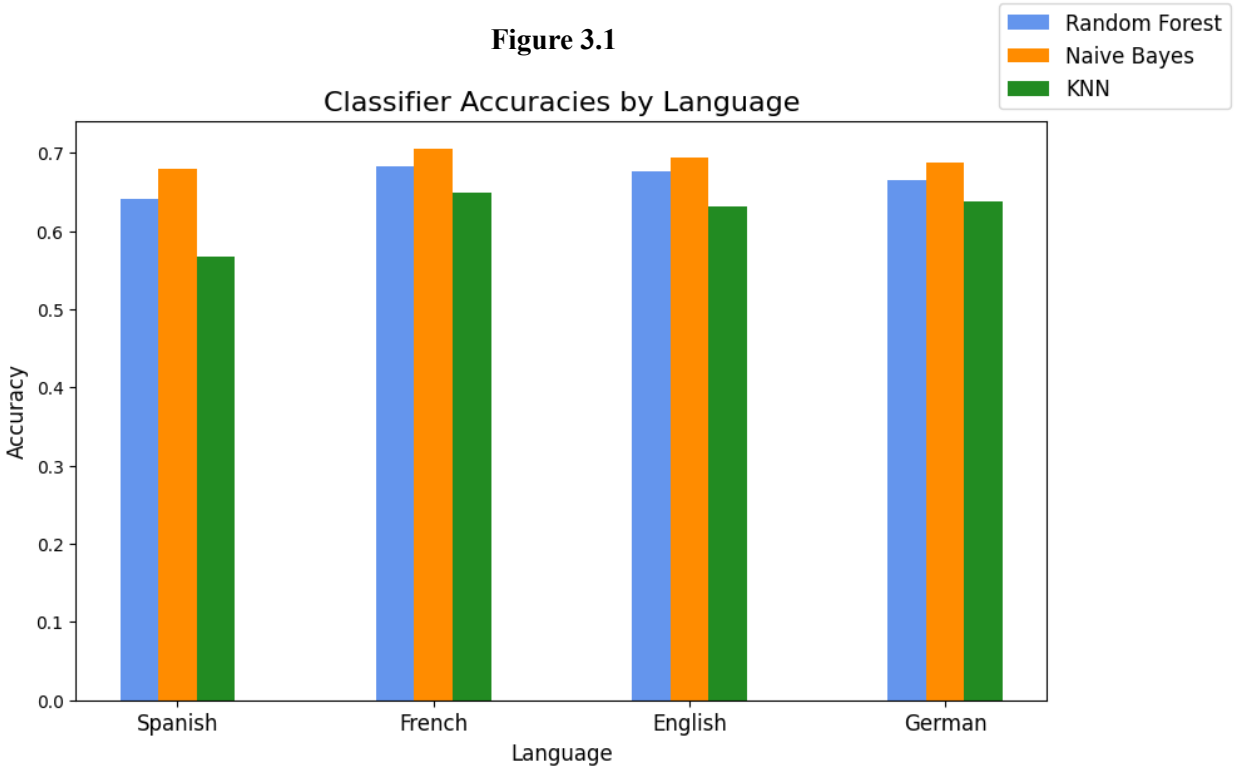

Figure 3.2

From the results of our experimentation with Naive Bayes, kNN, and Random Forest classifiers, it was evident that Naive Bayes produced the highest accuracies. However, we observed that it failed to classify the neutral class.

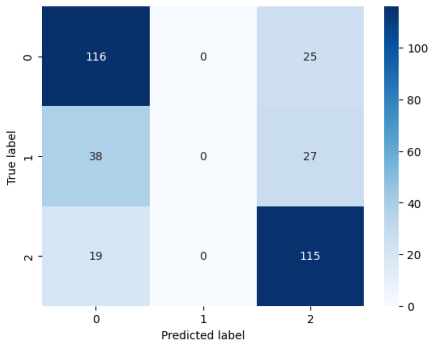

Figure 4.1: Naive Bayes Performance for Spanish

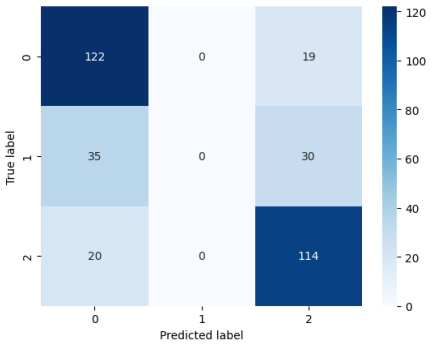

Figure 4.2: Naive Bayes Performance for English

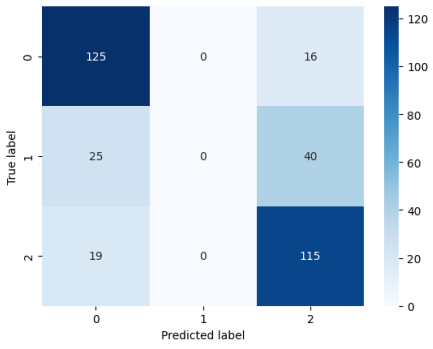

Figure 4.3: Naive Bayes Performance for French

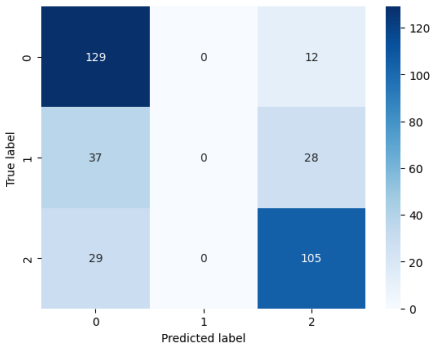

Figure 4.4: Naive Bayes Performance for German

Given that the neutral class is an essential component of our dataset, we chose not to use Naive Bayes. Instead, we chose the next best classifier, Random Forest, which produced accuracies only slightly different from Naive Bayes but performed better in classifying the neutral class. On the other hand, kNN produced the least accuracy of the three classifiers, and we decided not to use it. Therefore, after considering the performance of the three classifiers, we chose to use Random Forest as our classifier.

Fine-tuning BERT model

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based neural network architecture developed by Google AI Language. It has revolutionized the field of natural language processing (NLP) by achieving state-of-the-art results on a variety of NLP tasks. BERT is pre-trained on a large corpus of text data and fine-tuned on specific downstream tasks, making it highly effective for transfer learning. The BERT model is available in various sizes, from a few hundred million to several billion parameters, and pre-trained on various languages. For our project, we have chosen the bert-base-multilingual-cased model, which is pre-trained on a large corpus of 104 languages and has 110 million parameters. The model has shown impressive performance on various multilingual NLP tasks, and its ability to handle multiple languages makes it an ideal choice for our multilingual sentiment analysis task. In the following paragraphs, we will describe in detail how we fine-tuned the BERT model on our dataset to build an accurate multilingual sentiment analysis model.

To address the imbalanced distribution of labels in the datasets, we employed a stratified approach when splitting each dataset. Firstly, we utilized the map_polarity function to derive the polarity column from the label column of each review. This function assigns 'positive', 'neutral', or 'negative' as possible values to each review. Subsequently, we transformed these values into numerical labels for modeling purposes. Afterward, we partitioned the dataset into training and validation sets using the split ratio of 85% and 15%, respectively, and labeled each row based on its corresponding set. Finally, we verified the effectiveness of the stratified splitting by using the groupby method to count the number of reviews for each polarity, label, and data type, ensuring that the distribution of labels was preserved in both sets.

			review_body
polarity	label	data_type	
negative	1	train	1165
		val	206
neutral	2	train	572
		val	101
positive	0	train	1153
		val	203

Figure 5: Label distribution after splitting

After splitting the dataset, we preprocessed the textual data by tokenizing the reviews using the BERT tokenizer provided by the transformers library. We first loaded the pre-trained BERT tokenizer for the multilingual cased model provided by the transformers library, and then tokenized each review using the tokenizer's encode function. This function converts the text to a sequence of token IDs, which represents the input data that can be fed to the BERT model. To ensure the consistency of the input data, we set the maximum sequence length to the maximum length of all the encoded reviews in the dataset. The tokenizer's batch_encode_plus function was used to encode the training and validation sets separately. Special tokens were added to mark the beginning and end of each sequence, and attention masks were returned to indicate the position of the tokens in each sequence. Additionally, we padded the sequences to the maximum length and converted them to PyTorch tensors. Finally, we created PyTorch datasets for the training and validation sets, which consisted of the input IDs, attention masks, and labels. This approach ensures that the textual data is properly encoded and ready for modeling using the BERT model.

After preprocessing our textual data using the BERT tokenizer, we fine-tuned the BERT model for our task, using the BertForSequenceClassification class provided by the transformers library. We specified the pre-trained model to use as "bert-base-multilingual-cased". To adapt the model for our specific task, we set the num_labels parameter to the number of labels in our dataset, which is three (positive, neutral, and negative). We also turned off the output_attentions and output_hidden_states options since we didn't need them for our task.

During the training and validation of our BERT model, we utilized PyTorch data loaders to feed the preprocessed data. We specified a batch size of 32 for both the training and validation sets, which allows us to process the data in smaller chunks and reduce memory usage. To sample the data during training, we used the RandomSampler to randomly select batches of data from the training set. For the validation set, we used the SequentialSampler to ensure that the data was processed in order during validation. We then created data loaders for the training and validation sets using the DataLoader class, which takes the PyTorch datasets and the samplers as inputs. Additionally, we ensured that the model and data were on the same device by transferring the model to the available GPU or CPU using the to() method.

We then employed the AdamW optimizer, a variant of the Adam optimizer that incorporates weight decay, to optimize the parameters of our BERT model. The learning rate was set to 1e-5, a common value for fine-tuning BERT models, and the epsilon value to 1e-8. We used a linear scheduler provided by the transformers library, which increases the learning rate gradually during the warm-up phase and linearly decreases it during training. We set the number of warm-

up steps to 0, indicating that the learning rate would start increasing from the first step, and the number of total training steps to the product of the number of batches per epoch and the number of epochs. We chose to train the model for 3 epochs based on empirical evaluation and computational resource constraints. A higher number of epochs may lead to overfitting, whereas a lower number of epochs may not capture the underlying patterns in the data.

To evaluate the performance of our model, we used two common metrics: F1 score and overall accuracy. We defined a function to calculate the F1 score, which is a weighted average of the precision and recall, for our multiclass classification task. The function takes the predicted and true labels as inputs, and returns the F1 score. We also defined another function to calculate the overall accuracy of the model, which is the percentage of correctly classified samples in the dataset. The function takes the predicted and true labels as inputs, and prints the overall accuracy of the model. We used these metrics to evaluate the performance of our model on both the training and validation sets.

In our training loop, we start by setting a random seed to ensure reproducibility of our results. Then, we define a function to evaluate the model's performance on the validation set. The function sets the model to evaluation mode and iterates through the validation data to calculate the loss and predictions for each batch. It then returns the average loss and the predictions and true labels for the entire dataset.

We then iterate through the specified number of epochs using a tqdm progress bar to track the progress of each epoch. Within each epoch, we set the model to training mode and iterate through the training data to calculate the loss and update the model's parameters using the optimizer and scheduler. We also use gradient clipping to prevent the gradients from exploding during training. After each epoch, we evaluate the model's performance on the validation set using the previously defined evaluation function and print the training and validation losses and the F1 score. This process helps us to monitor the model's progress and identify any overfitting or underfitting issues.

```
English
  Language  Epoch  Training Loss  Validation Loss  F1 Score (Weighted)
6  English      1       1.009612         0.907302             0.541246
7  English      2       0.857522         0.843173             0.565554
8  English      3       0.794218         0.832819             0.565554

French
  Language  Epoch  Training Loss  Validation Loss  F1 Score (Weighted)
3   French      1       1.014892         1.004634             0.469612
4   French      2       0.865063         0.876547             0.551511
5   French      3       0.781589         0.884834             0.566356

German
   Language  Epoch  Training Loss  Validation Loss  F1 Score (Weighted)
9    German      1       0.935249         0.834825             0.578070
10   German      2       0.760781         0.789961             0.595631
11   German      3       0.683947         0.785754             0.621621

Spanish
  Language  Epoch  Training Loss  Validation Loss  F1 Score (Weighted)
0  Spanish      1       1.005242         0.959984             0.491749
1  Spanish      2       0.863872         0.887547             0.548436
2  Spanish      3       0.780352         0.882993             0.558024
```

The results obtained from training our fine-tuned BERT model shed light on its performance across various languages. The primary indicators of performance are the loss values, where lower values indicate better model performance. Additionally, the F1 score, which considers accuracy while accounting for class imbalances, serves as an important measure.

Analyzing the results, we observe that the model exhibits stronger performance on the German language, as evidenced by higher F1 scores and lower loss values across all epochs. This suggests that the model is better able to capture the nuances of the German language and make accurate sentiment predictions. However, it's worth noting that the model also demonstrates reasonable performance on other languages, such as English and Spanish, albeit with slight variations in the F1 scores and loss values.

After repeating the training and evaluation process for the four different languages, we proceeded to compare the accuracy of our fine-tuned BERT model with the baseline accuracy for each language.

	Random Forest	BERT
Spanish	0.641176	0.6235
French	0.682353	0.6333
English	0.676471	0.6431
German	0.664706	0.6725

Figure 6.1

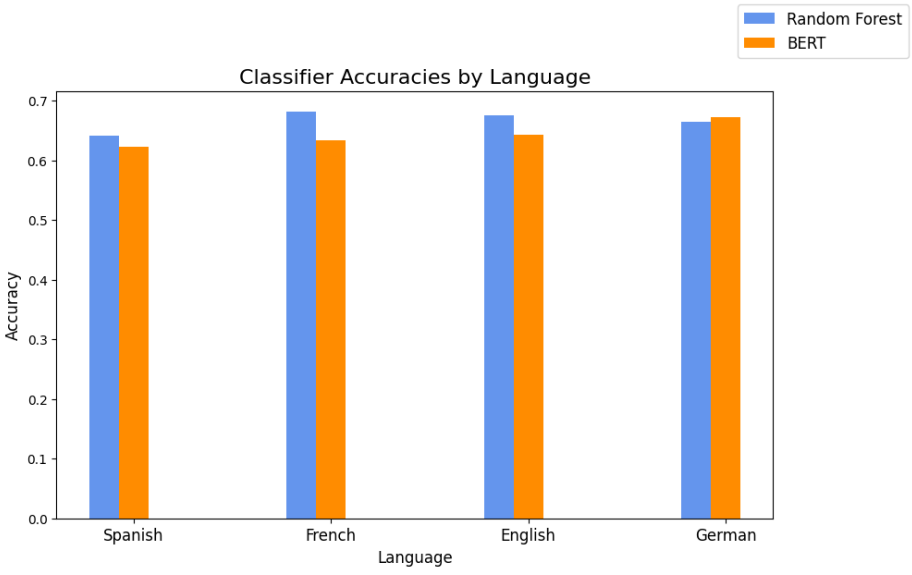

Figure 6.2

The performance of our fine-tuned BERT model was compared to the baseline Random Forest classifier, and the results are presented in Figure 6.1 and Figure 6.2. The analysis reveals that the BERT model achieves competitive performance across the four languages considered. Specifically, in the Spanish language, the BERT model demonstrates an accuracy of 0.6235, slightly lower than the Random Forest accuracy of 0.6412. Similarly, for French, the BERT model achieves an accuracy of 0.6333, while the Random Forest accuracy is slightly higher at 0.6824. The trend continues for the English language, with the BERT model achieving an accuracy of 0.6431 compared to the Random Forest accuracy of 0.6765. However, in the case of German, the BERT model excels with an accuracy of 0.6725, surpassing the Random Forest accuracy of 0.6647. These results clearly demonstrate the potential of the fine-tuned BERT model in accurately predicting sentiment across the four different languages, exhibiting competitive performance when compared to the baseline classifier.

Finally, to ensure the accessibility and seamless deployment of our fine-tuned BERT models for each language, we implemented a streamlined process to save and store the models. Leveraging the capabilities of the Hugging Face API, we instantiated the HfApi class, enabling us to interact with the Hub effortlessly. Utilizing the model.push_to_hub() method, we successfully uploaded the fine-tuned BERT model for each language to the Hub.

Language	Model Name
English	Worgu/Final_Project_finetuned_bert-base-multilingual-cased_english
German	Worgu/Final_Project_finetuned_bert-base-multilingual-cased_german
French	Worgu/Final_Project_finetuned_bert-base-multilingual-cased_french
Spanish	Worgu/Final_Project_finetuned_bert-base-multilingual-cased_spanish

We took a similar approach to save and push the associated tokenizer for each fine-tuned BERT model to the Hub. The tokenizer plays a critical role in preprocessing input text, so it was important to ensure its availability for deployment alongside the model. By saving both the model and tokenizer to the Hub, we established an efficient and streamlined process for accessing and deploying these resources in the User Interface.

Deploying the User Interface

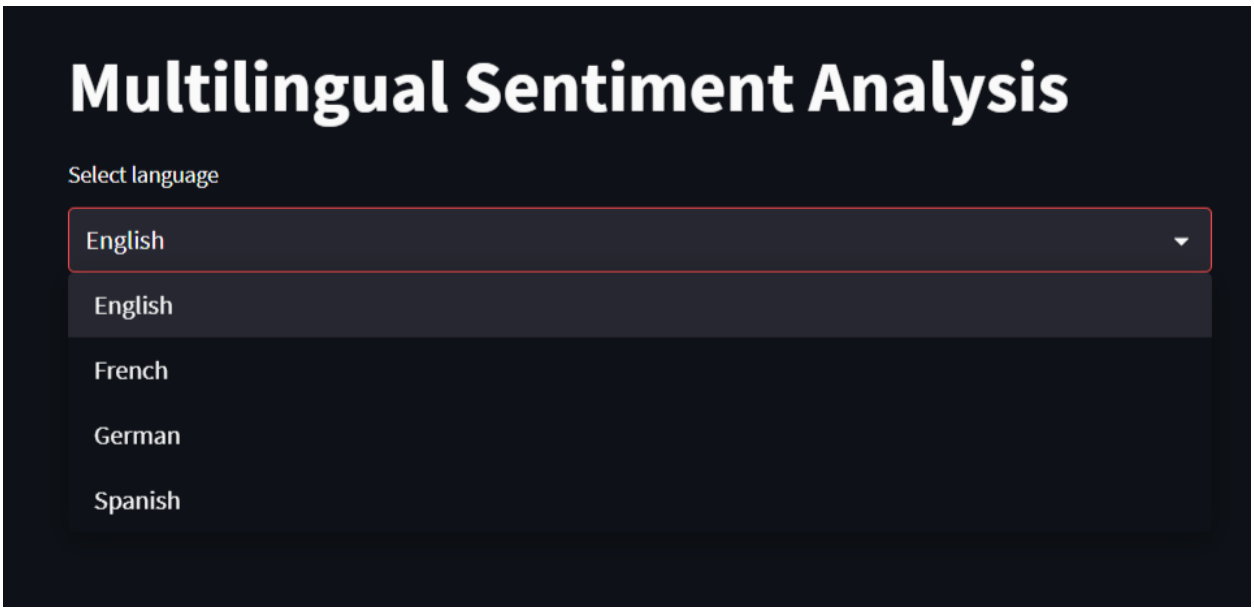

Figure 7.1: Our Multiligual Sentiment Analysis Online Demo

Our interface was developed on Streamlit.

Link: https://huggingface.co/spaces/Worgu/DATA_641_Project

Language Selection: The application provides a dropdown menu where users can select the language of the text they want to analyze. This is implemented using the st.selectbox function. The available languages in the application are English, French, German, and Spanish.

After the language is selected, the corresponding model is loaded for sentiment analysis. We have trained and stored different models for different languages. The load_model() function takes the selected language as an argument and loads the corresponding model and tokenizer.

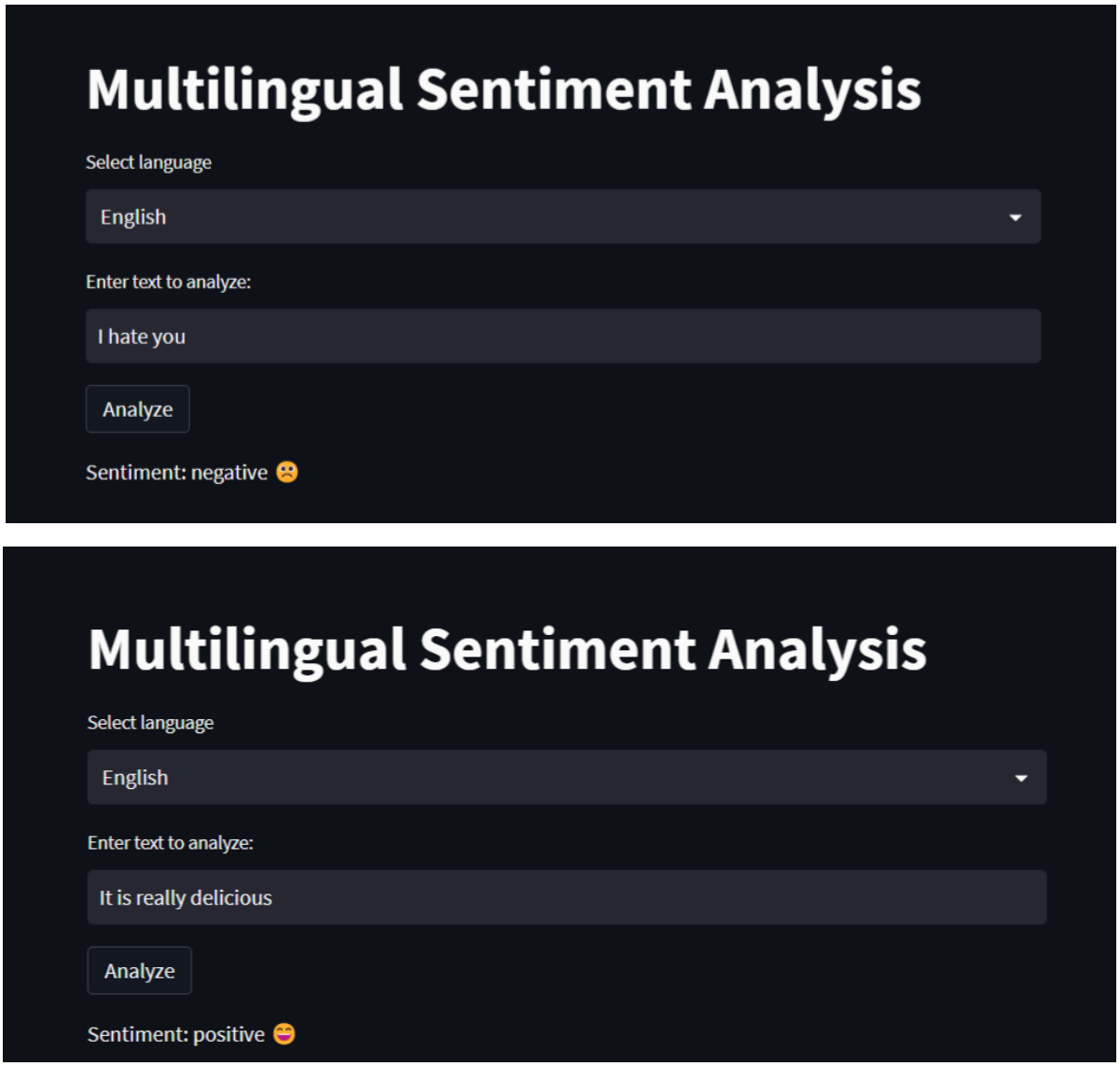

Figure 7.2: Sentiment prediction for English language

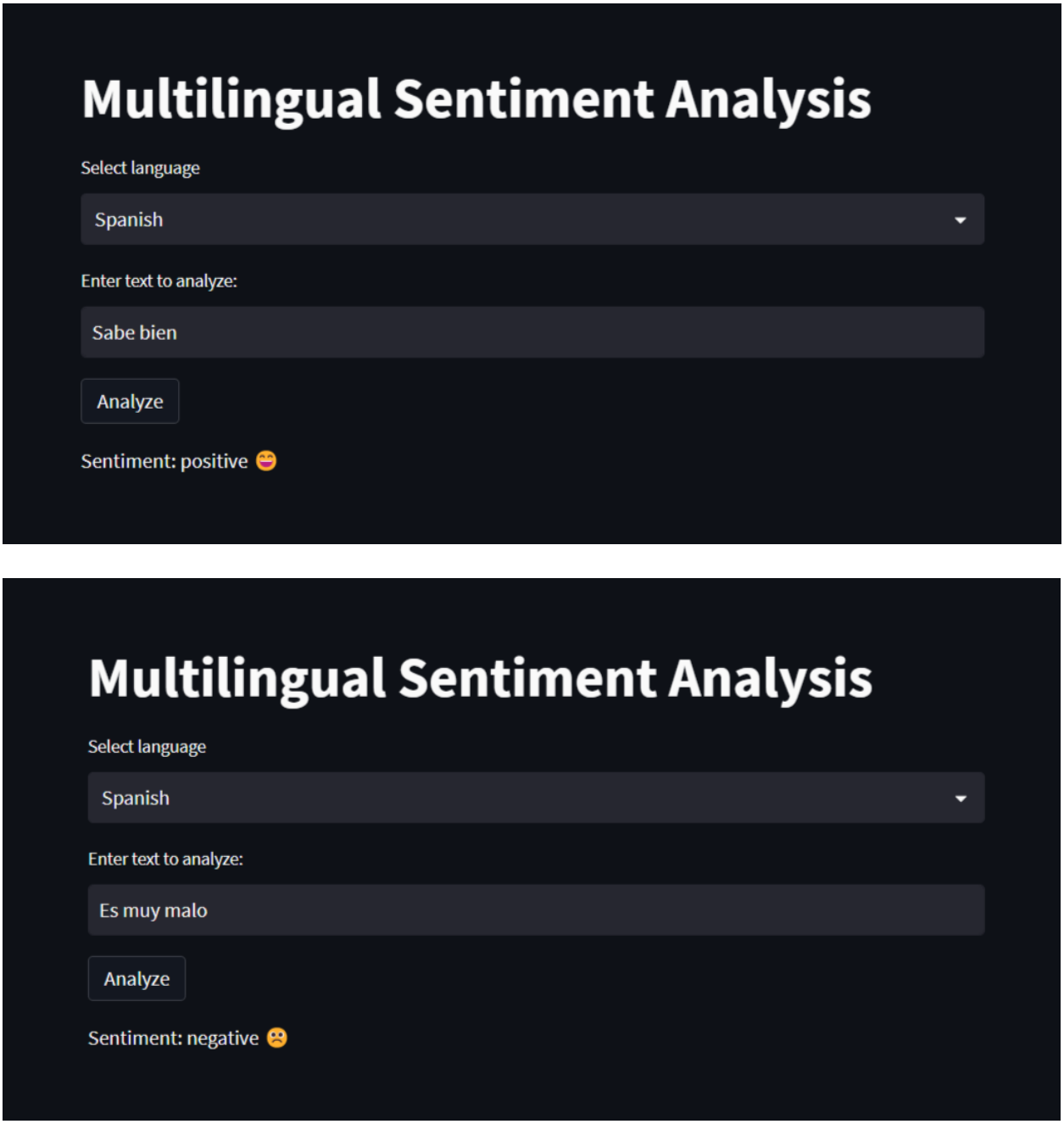

Figure 7.3: Sentiment prediction for Spanish language

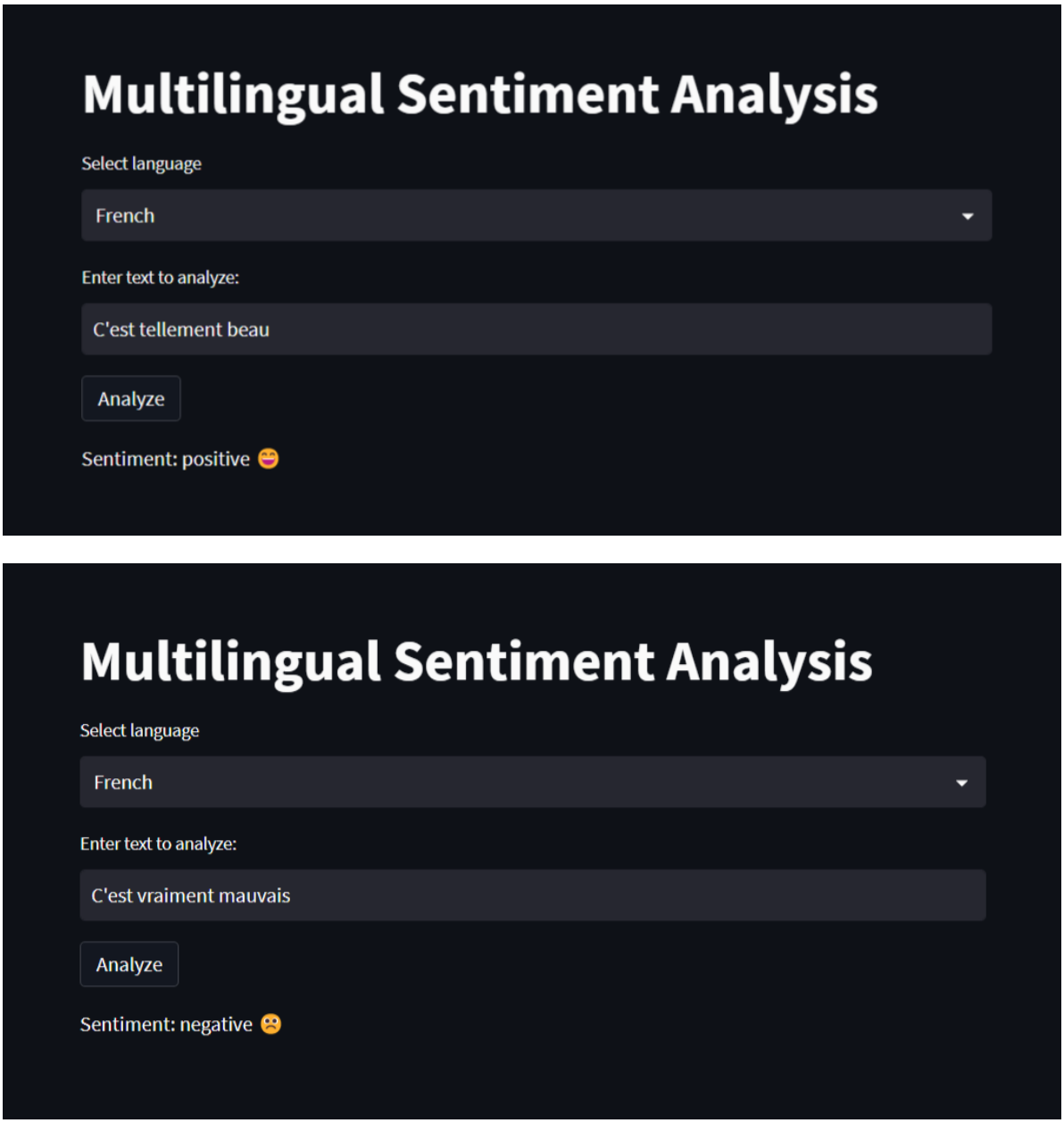

Figure 7.5: Sentiment prediction for French language

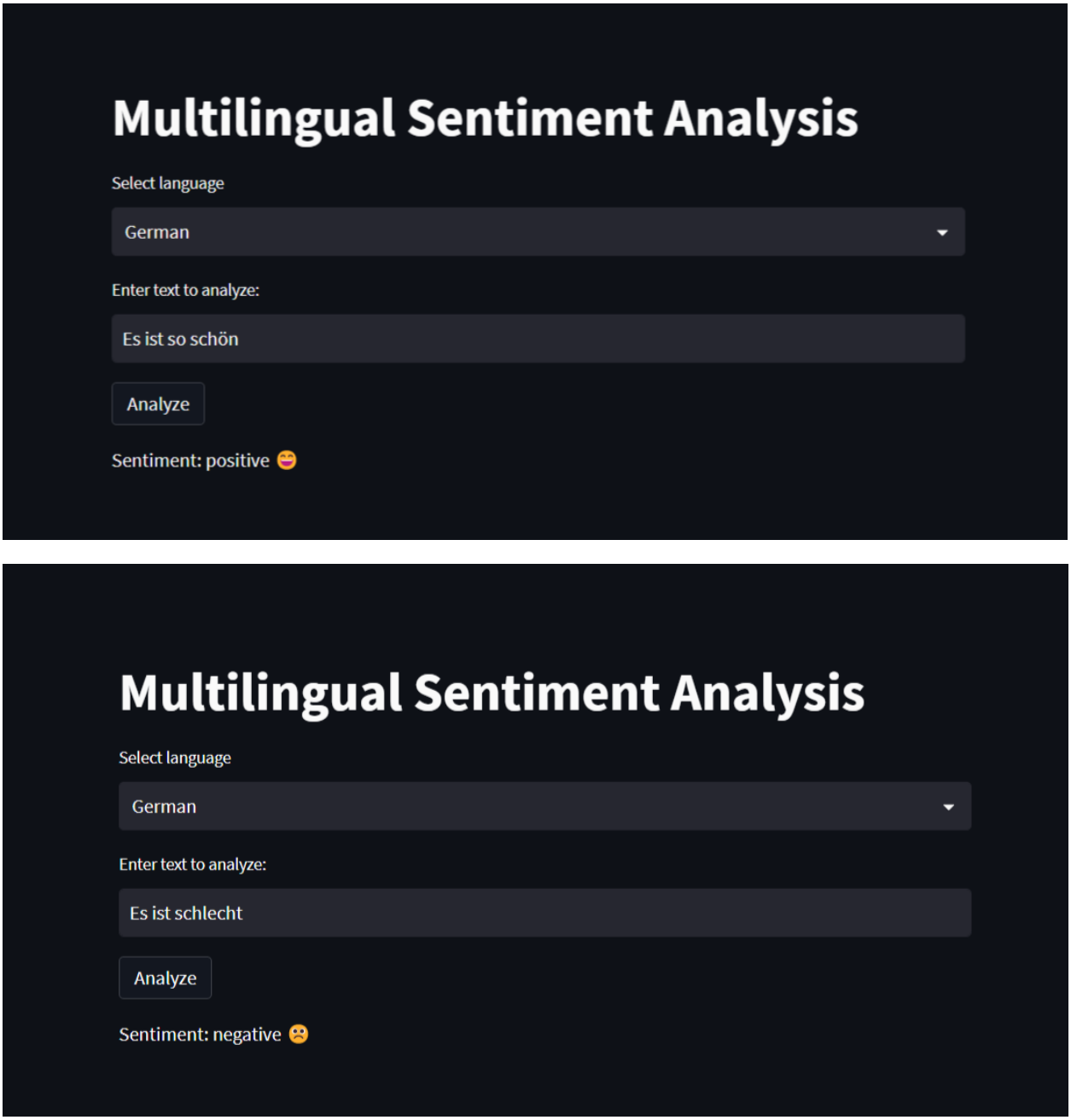

Figure 7.4: Sentiment prediction for German language

1. **Analyze Button:** There's an "Analyze" button in the application interface. When this button is clicked, the entered text is processed and analyzed. This is implemented with the **st.button()** function.

2. **Text Cleaning:** Before analysis, the text is cleaned. The **clean_text()** function performs several operations to prepare the text for analysis. It converts the text to lowercase, tokenizes it into sentences and words, removes stopwords, applies stemming and lemmatization, and removes non-alphanumeric characters and digits. The resulting clean text is a version of the original text that's easier to analyze.

3. **Model Loading:** Depending on the selected language, a different model is loaded using the **load_model()** function. Each model has been fine-tuned on a specific language (English, French, German, or Spanish).

4. **Sentiment Analysis:** The clean text is analyzed using the appropriate model. The application uses a **TextClassificationPipeline** from the **transformers** library to do this. The pipeline takes the model and tokenizer as input and provides a simple interface for classifying the sentiment of the cleaned text.

5. **Result Display:** The result of the sentiment analysis is a label indicating the sentiment: positive, neutral, or negative. This label is displayed to the user. The application uses emojis to make the output more user-friendly.

Conclusion

In conclusion, our project focused on developing an online sentiment analysis demo using fine-tuned BERT models for multiple languages. Through rigorous training and evaluation, we have successfully demonstrated the competitive performance of these models compared to the baseline classifier. The fine-tuned BERT models have showcased their exceptional ability to accurately predict sentiment across various languages, highlighting their potential for multilingual sentiment analysis applications. Furthermore, we have designed and implemented a user-friendly interface that enables users to conveniently input text and obtain accurate sentiment analysis results. Overall, our project showcases the effectiveness of BERT models in sentiment analysis tasks and offers a practical solution for sentiment analysis across multiple languages. Future work may involve expanding the scope of model training to include more languages and integrating additional features to create a comprehensive sentiment analysis system.

Limitations

While our project has achieved success in developing sentiment analysis models and an interactive user interface, it is important to acknowledge certain limitations. One of the primary limitations is the constraint posed by computational power and its associated cost. Working with large datasets like The Multilingual Amazon Reviews Corpus and training complex models like BERT requires significant computational resources, which can hinder scalability and efficiency. Another critical aspect is the identification and handling of ethical issues within the training data. Ensuring diversity, representativeness, and mitigating biases are crucial considerations to avoid perpetuating discriminatory sentiments. Despite the promising results obtained, it is important to recognize that our methods do not capture the full complexity of sentiment analysis, and further research and advancements are needed in this area. Additionally, there were instances where the predictions made by the models were inaccurate or inconsistent, emphasizing the inherent challenges in sentiment prediction. Lastly, it is essential to acknowledge that machine learning models can inherit biases from the training data, leading to potential biases in their predictions. Addressing and mitigating these biases is an ongoing endeavor within the field of natural language processing.

References

Amazon_reviews_multi · datasets at hugging face. amazon_reviews_multi · Datasets at Hugging Face. (n.d.). https://huggingface.co/datasets/amazon_reviews_multi/viewer/de/train

Bert-base-multilingual-cased · hugging face. bert-base-multilingual-cased · Hugging Face. (n.d.). https://huggingface.co/bert-base-multilingual-cased

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: Analyzing text with the natural language toolkit*. O'Reilly.

Hugging face – the AI community building the future. Hugging Face –. (n.d.). https://huggingface.co/

GitHub. (n.d.). https://github.com/

Manning, C. D., Raghavan, P., & Schütze, H. (2019). *Introduction to information retrieval*. Cambridge University Press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (1970, January 1). *Scikit-Learn: Machine learning in Python*. Journal of Machine Learning Research. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

Streamlit • A faster way to build and share data apps. (n.d.). https://streamlit.io/