A PROJECT REPORT

on

“Sentiment Analysis on Google Play Store Reviews”

Under the guidance of

Prof. Deependra Singh

School of Computer Engineering

Kalinga Institute of Industrial Technology

BHUBANESWAR, ODISHA-751024

April2024

By

Anish Singh	2105442
Anwesha Sahu	21051379
Bodhisatta Bhattacharjee	22057025
Sankalp Chauhan	22057053
Shreya Kumari	22057058

Abstract

This project explores the intersection of machine learning (ML) and natural language processing (NLP) to develop a comprehensive sentiment analysis tool. Utilizing state-of-the-art models from the Hugging Face transformers library (especially twitter's RoBERTa) and techniques from the Python library NLTK, alongside alternative approaches without NLTK, we aim to provide nuanced sentiment detection across various text inputs (such a google play store reviews). The core of this tool is a Flask-based backend, which interfaces with both the Hugging Face API for advanced model inference and NLTK for foundational NLP operations. The frontend, developed with React and styled using TailwindCSS, offers a user-friendly interface for live interaction with the sentiment analysis tool. This full-stack web development project not only incorporates interactive demonstrations of the sentiment analysis process via Google Collab Notebooks embedded within the web interface but also provides a theoretical background on the underlying ML and NLP concepts. The project's architecture supports live testing of models, enabling users to understand the practical applications of sentiment analysis in real-time. Through this project, we demonstrate the integration of cutting-edge ML models and NLP techniques in a full-stack application, highlighting the accessibility and potential of current technology in processing and understanding human language.

Visit the Web Application – **Opini-Analyzer-Client**

Table of Contents

1. Introduction

1.1 What is Data Science?

Data science is the study of data to extract meaningful insights for business. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyse large amounts of data. This analysis helps data scientists to ask and answer questions like what happened, why it happened, what will happen, and what can be done with the results.

1.2. What is Data Science used for?

1. Descriptive analysis - Descriptive analysis examines data to gain insights into what happened or what is happening in the data environment. It is characterized by data visualizations such as pie charts, bar charts, line graphs, tables, or generated narratives. For example, a flight booking service may record data like the number of tickets booked each day. Descriptive analysis will reveal booking spikes, booking slumps, and high-performing months for this service.

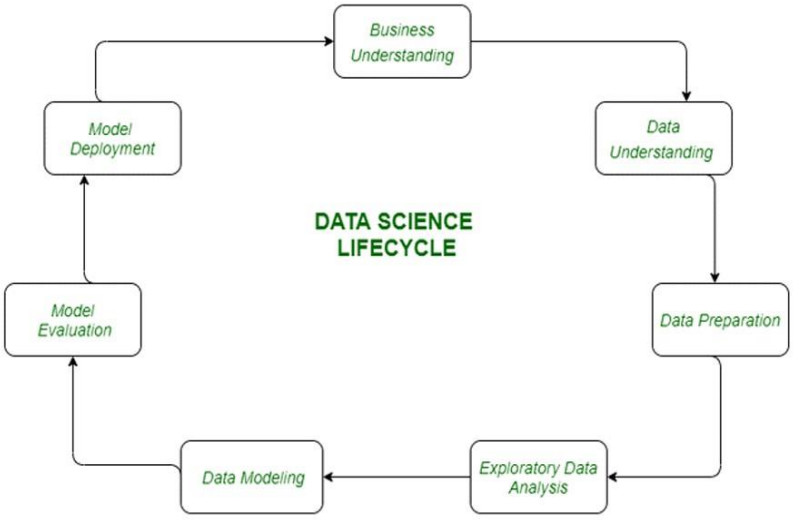
2. Diagnostic analysis - Diagnostic analysis is a deep-dive or detailed data examination to understand why something happened. It is characterized by techniques such as drill-down, data discovery, data mining, and correlations. Multiple data operations and transformations may be performed on a given data set to discover unique patterns in each of these techniques. For example, the flight service might drill down on a particularly high-performing month to better understand the booking spike. This may lead to the discovery that many customers visit a particular city to attend a monthly sporting event.

3. Predictive analysis - Predictive analysis uses historical data to make accurate forecasts about data patterns that may occur in the future. It is characterized by techniques such as machine learning, forecasting, pattern matching, and predictive modelling. In each of these techniques, computers are trained to reverse engineer causality connections in the data. For example, the flight service team might use data science to predict flight booking patterns for the coming year at the start of each year. The computer program or algorithm may look at past data and predict booking spikes for certain destinations in May. Having anticipated their customer's future travel requirements, the company could start targeted advertising for those cities from February.

4. Prescriptive analysis - Prescriptive analytics takes predictive data to the next level. It not only predicts what is likely to happen but also suggests an optimum response to that outcome. It can analyse the potential implications of different choices and recommend the best course of action. It uses graph analysis, simulation, complex event processing, neural networks, and recommendation engines from machine learning.

Back to the flight booking example, prescriptive analysis could look at historical marketing campaigns to maximize the advantage of the upcoming booking spike. A data scientist could project booking outcomes for different levels of marketing spend on various marketing channels. These data forecasts would give the flight booking company greater confidence in their marketing decisions.

1.3. Data Science Life Cycle



1.4. What is Machine Learning?

Machine learning (ML) is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn, gradually improving its accuracy.

How does machine learning work? UC Berkeley (link resides outside ibm.com) breaks out the learning system of a machine learning algorithm into three main parts.

1. A Decision Process: In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labelled or unlabelled, your algorithm will produce an estimate about a pattern in the data.

2. An Error Function: An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.

3. A Model Optimization Process: If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this iterative "evaluate and optimize" process, updating weights autonomously until a threshold of accuracy has been met.

1.5. Machine Learning Methods

Machine learning models fall into three primary categories.

1. Supervised machine learning - Supervised learning, also known as supervised machine learning, is defined by its use of labelled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, the model adjusts its weights until it has been fitted appropriately. This occurs as part of the cross validation process to ensure that the model avoids overfitting or under fitting. Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. Some methods used in supervised learning include neural networks, Naive Bayes, linear regression, logistic regression, random forest, and support vector machine (SVM).

2. Unsupervised machine learning - Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyse and cluster unlabelled datasets (subsets called clusters). These algorithms discover hidden patterns or data groupings without the need for human intervention. This method's ability to discover similarities and differences in information make it ideal for exploratory data analysis, cross-selling strategies, customer segmentation, and image and pattern recognition. It's also used to reduce the number of features in a model through the process of dimensionality reduction. Principal component analysis (PCA) and singular value decomposition (SVD) are two common approaches for this. Other algorithms used in unsupervised learning include neural networks, k-means clustering, and probabilistic clustering methods.

3. Semi-supervised learning - Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labelled data set to guide classification and feature extraction from a larger, unlabelled data set. Semi-supervised learning can solve the problem of not having enough labelled data for a supervised learning algorithm. It also helps if it's too costly to label enough data.

1.6. Common Machine Learning Algorithms:

A number of machine learning algorithms are commonly used. These include:

1. Neural networks - Neural networks simulate the way the human brain works, with a huge number of linked processing nodes. Neural networks are good at recognizing patterns and play an important role in applications including natural language translation, image recognition, speech recognition, and image creation.

2. Linear regression - This algorithm is used to predict numerical values, based on a linear relationship between different values. For example, the technique could be used to predict house prices based on historical data for the area.

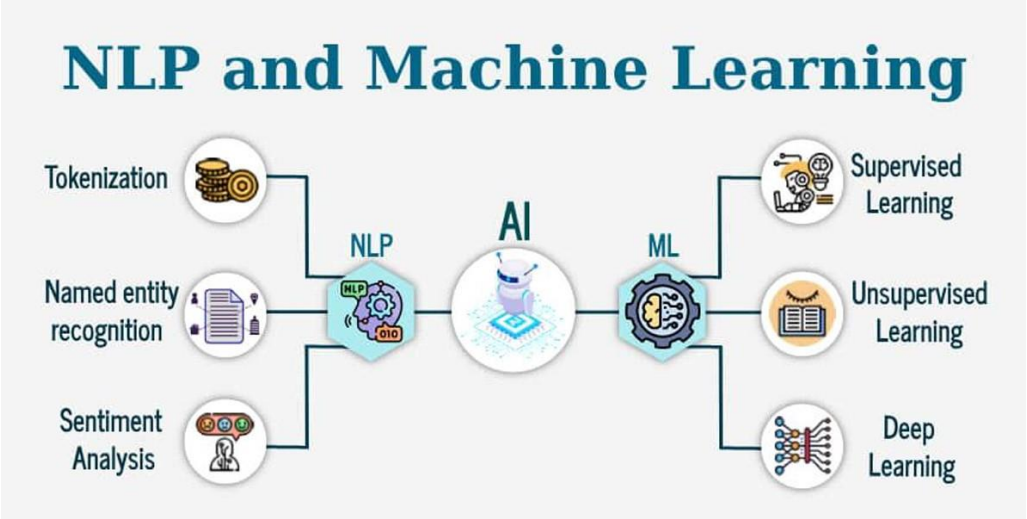
3. Logistic regression - This supervised learning algorithm makes predictions for categorical response variables, such as "yes/no" answers to questions. It can be used for applications such as classifying spam and quality control on a production line.

4. Clustering - Using unsupervised learning, clustering algorithms can identify patterns in data so that it can be grouped. Computers can help data scientists by identifying differences between data items that humans have overlooked.

5. Decision trees - Decision trees can be used for both predicting numerical values (regression) and classifying data into categories. Decision trees use a branching sequence of linked decisions that can be represented with a tree diagram. One of the advantages of decision trees is that they are easy to validate and audit, unlike the black box of the neural network. 6.Random forests: In a random forest, the machine learning algorithm predicts a value or category by combining the results from a number of decision trees.

6. Random forests - In a random forest, the machine learning algorithm predicts a value or category by combining the results from a number of decision trees.

1.7. How is ML different from NLP?



1.8. How do we measure accuracy in ML and NLP models?

1. Accuracy - Accuracy is mainly used in classification problems to get the appropriate percentage of the models being executed to know how much of it is correctly predicted. In any model, there will be several classification problems will present when we execute them, then the accuracy plays the main role by talking about whether all the problems are correctly predicted or not with respect to the total number of predictions made by the model. We can calculate the accuracy of any model by dividing the correctly predicted problems by the total number of predictions made.

2. Accuracy in Binary Classification - In the binary classification case, we can communicate precision in True/False Positive/Negative qualities. The accuracy recipe in Machine learning is given as follows: How to Check the Accuracy of your Machine Learning Model in the accuracy classes, there are only two classes present. They are positive/negative:

TP: TP stands for True positive, i.e., These are the classes that are correctly predicted, and the correctly predicted classes are marked as positive.

FP: FP stands for false positive, i.e., These are the classes that are falsely predicted, and the falsely predicted classes are marked as positive.

TN: TN stands for True positive, i.e., These are the classes that are correctly predicted, and the correctly predicted classes are marked as negative.

FN: FN stands for False negative, i.e., These are the classes that are falsely predicted, and the falsely predicted classes are marked as negative.

1.9. Available Libraries that help doing sentiment analysis

1. NLTK (Natural Language Toolkit) - It offers sentiment analysis and various NLP tasks like breaking down text into words, identifying parts of speech, and analyzing sentence structure in Python.

2. TextBlob - It provides a straightforward Python interface for sentiment analysis, part-of-speech tagging, extracting key phrases, and language translation, all built on top of NLTK.

3. VADER(Valence Aware Dictionary and sEntiment Reasoner) - It is a Python library specifically designed for sentiment analysis, particularly well-suited for working with social media text.

4. Stanford CoreNLP - It is a Java-based toolkit that can perform sentiment analysis, identify named entities, tag parts of speech, and more.

5. Gensim - It is a Python library useful for topic modeling, document indexing, and finding textual similarities, which can enhance sentiment analysis and text clustering.

6. FastText - It is developed by Facebook's AI research team, allows for efficient learning of word representations and sentence classification, boosting sentiment analysis and text categorization.

7. Scikit-learn - It is a widely used Python library for machine learning, including sentiment analysis tasks through classifiers and text processing tools.

These libraries can also be applied to a variety of other natural language processing tasks, such as emotion detection, aspect-based sentiment analysis, opinion mining, topic modeling, named entity recognition, and document clustering. This expands their utility beyond just sentiment analysis, making them valuable tools for a range of text-based applications.

1.10. Deep Learning, NLP and DNLP

Deep Learning - In deep learning, artificial neural networks are trained with large amounts of data for pattern recognition and decision-making. It involves several layers of interconnected neurons that enable the model to learn complex representations of data.

Natural Language Processing - NLP is a field in artificial intelligence focusing on the interaction between computers and humans through natural language. Deep learning has greatly contributed to NLP by enabling models to learn intricate language features and perform tasks such as translation, sentiment analysis as well as text generation with high precision.

Deep NLP - DNLP combines deep learning methods with NLP techniques to construct sophisticated models that can understand, generate, manipulate human language. Frequently, DNLP models employ architectures such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer models for efficient sequential data processing and capturing contextual information effectively. In essence, deep learning enhances NLP by creating more powerful models, and DNLP integrates these technologies to develop systems that understand and generate human language with precision and complexity.

1.11. Existing Models

BERT (Bidirectional Encoder Representations from Transformers) - is a natural language processing model developed by Google that is more advanced. By considering both the left and right sides of words in any sentence it is put through; it has been found to be effective at understanding the context of words. BERT has improved several NLP tasks such as text classification, question answering and named entity recognition greatly by pre-training on large text datasets and fine-tuning on specific tasks.

"twitter-roberta" - is a variant of BERT that was designed specifically for social media texts like Twitter data. It has been trained on massive amounts of twitter text in order to capture more subtle nuances as well as informalisms used in tweets making it possible for people to create their own classifiers.

“Hugging Face” - serves as a popular library and platform where one can work with the most recent NLP models such as those called BERT and “twitter-roberta”. Therefore, anyone wishing to install these models into projects without paying any money should consider hugging face inference.

2. Implementation

2.1. Processes involved in Sentiment Analysis without NLTK Library-

1. Text Pre-processing - Before analysing sentiment, text data needs pre-processing to clean it. Common steps include removing punctuation, converting text to lowercase, and handling special characters.

2 Feature Extraction - Convert text into numerical or categorical features that can be used by machine learning algorithms. Common techniques include bag-of-words, TF-IDF, or word embedding like Word2Vec or GloVe.

3. Model Selection - Choose an appropriate model for sentiment analysis. Popular choices include Support Vector Machines (SVM), Naive Bayes, Logistic Regression, or deep learning models like Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs).

4. Training - Train the selected model using labeled data. Labeled data consists of text examples paired with their corresponding sentiment labels (e.g., positive, negative, neutral).

5. Testing and Evaluation - Evaluate the trained model using a separate set of labeled data not seen during training. Common evaluation metrics include accuracy, precision, recall, and F1-score.

6. Inference - Once the model is trained and evaluated, it can be used to predict sentiment on unseen text data.

7. Post-processing - Optionally, apply post-processing techniques to refine sentiment predictions. For example, you might use rules or heuristics to adjust predictions based on context or domain-specific knowledge.

8. Deployment - Deploy the trained model into production systems where it can analyse sentiment in real-time or batch processing pipelines.

9. Feedback Loop (Optional) - Continuously improve the model's performance by collecting feedback from users or monitoring its predictions in production. This feedback can be used to retrain the model on updated data or to fine-tune its parameters.

2.2 Example Code for Sentiment Analysis without NLTK Library

```
import string
 def remove_punctuation(text):
 # Remove punctuation from the text
 return text.translate(str.maketrans( ', ', string.punctuation))
 def tokenize(text):
 # Tokenize the text by splitting on whitespace
 return text.split()
 def stem(word):
 # Simple stemming by removing common suffixes
 if word.endswith('s'):
 return word[:-1]
 return word
 def preprocess_text(text):
 # Preprocess the text by removing punctuation and tokenizing
 text = remove_punctuation(text.lower())
 tokens = tokenize(text)
 # Stem each token
 stemmed_tokens = [stem(token) for token in tokens]
 return stemmed_tokens
 def analyze_sentiment(tokens):
# Analyze sentiment based on token list
 positive_words = ["good", "happy", "joyful", "great"]
 negative_words = ["bad", "sad", "unhappy", "terrible"]
 sentiment_score = 0
```

```
for token in tokens:
 if token in positive_words:
 sentiment_score += 1
 elif token in negative_words:
 sentiment_score-= 1
 if sentiment_score > 0:
 return "Positive"
 elif sentiment_score < 0:
 return "Negative"
 else:
 return "Neutral"
# Main function
 if __name__ == "__main__":
 # Input text
 input_text = input("Enter a sentence: ")
 print("Input Text:", input_text)
 # Preprocess text
 preprocessed_text =
preprocess_text(input_text)
 print("Preprocessed Text:", preprocessed_text)
 # Analyze sentiment
 sentiment =
analyze_sentiment(preprocessed_text)
 print("Sentiment:", sentiment
```

2.3. Processes involved in Sentiment Analysis with NLTK Library

1. Import Libraries and Data - ○ You'll start by importing NLTK and any other necessary libraries like pandas for data manipulation. ○ Load your text data, which could be reviews, social media posts, or any other text source.

2. Pre-process Text Data - ○ This involves cleaning the text by removing punctuation, stop words (common words like "the" or "a"), and converting everything to lowercase. ○ You might also perform stemming or lemmatization to reduce words to their root form.

3. Lexicon-Based Analysis (VADER) - ○ NLTK offers a pre-trained sentiment analyser called VADER (Valence Aware Dictionary and sEntiment Reasoner). ○ VADER works by assigning sentiment scores to words based on a predefined lexicon. ○ It considers punctuation and capitalization for better analysis of informal text. ○ VADER outputs a dictionary with scores for positive, negative, neutral sentiment and a compound score representing the overall sentiment.

4. Machine Learning Approach (Optional) - ○ NLTK allows building custom sentiment classifiers using machine learning. ○ This involves creating a training dataset with labeled text data (positive, negative, neutral). ○ You can then train a model like Naive Bayes or Support Vector Machine to classify new text.

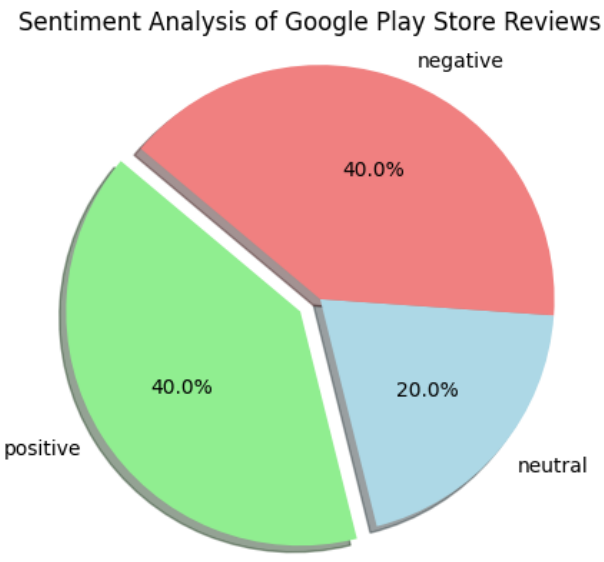
5. Analyse Sentiment - ○ After using VADER or your trained model, you'll get sentiment scores for your text data. ○ You can interpret these scores to understand the overall sentiment expressed in the text. ○ Depending on your needs, you can classify the sentiment as positive, negative, or neutral based on predefined thresholds. Pros and Cons of Lexicon-Based Approach (VADER): ● Pros: Easy to use, fast, good for informal text. ● Cons: Less accurate for complex or formal text, may not capture sarcasm or negation.

2.4. Example Code for Sentiment Analysis with NLTK Library

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# Download necessary NLTK resources (run once)
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('vader_lexicon')
# Step 1: User input
user_input = input("Enter a sentence to analyze sentiment: ")
# Step 2: Tokenization
tokens = word_tokenize(user_input)
print("\nStep 2: Tokenization - Tokens:", tokens)
# Step 3: Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
print("\nStep 3: Stopwords removal - Filtered Tokens:", filtered_tokens)
# Step 4: Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
print("\nStep 4: Lemmatization - Lemmatized Tokens:", lemmatized_tokens)
# Step 5: Sentiment Analysis
analyzer = SentimentIntensityAnalyzer()
sentiment_score = analyzer.polarity_scores(user_input)
# Step 6: Output
print("\nStep 5: Sentiment Analysis - Sentiment Score:", sentiment_score)
# Interpretation
if sentiment_score['compound'] >= 0.05:
print("\nOverall sentiment: Positive")
elif sentiment_score['compound'] <= -0.05:
print("\nOverall sentiment: Negative")
else:
print("\nOverall sentiment: Neutral")
```

2.5. Using Hugging Face Inference to use BERT models

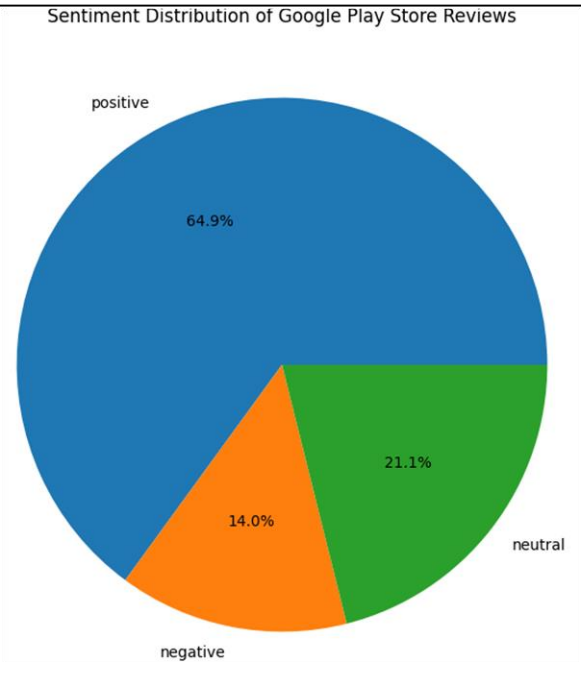
```
# Import necessary libraries
from transformers import pipeline
import matplotlib.pyplot as plt
import pandas as pd
# Load sentiment analysis pipeline from Hugging Face's "cardiffnlp/twitter-roberta-base-sentiment-latest" model
sentiment_pipeline = pipeline("sentiment-analysis", model="cardiffnlp/twitter-roberta-base-sentiment-latest")
# Define a dataset of 5 review sentences
reviews = [
    "This app is fantastic! Helped me a lot in organizing my tasks.",
    "Not what I expected, it's quite confusing and buggy.",
    "The app is okay, but there are some features that are missing.",
    "I absolutely love this app, it's user friendly and effective.",
    "Terrible experience, the app crashes frequently and lost my data."
]
# Load the dataset
# df = pd.read_csv('dataset_reviews_2.csv')
# Extract the reviews from the 4th column (assuming the first column is index 0)
# reviews = df.iloc[:, 3].tolist()
# Analyze the sentiments of the reviews
results = sentiment_pipeline(reviews)
# Initialize counters for each sentiment
sentiment_counts = {"positive": 0, "neutral": 0, "negative": 0}
# Count the number of each sentiment
for result in results:
    sentiment_counts[result['label'].lower()] += 1
# Print out the sentiment counts
print(sentiment_counts)
# Plot the percentages of positive, negative, neutral using a pie chart
labels = sentiment_counts.keys()
sizes = sentiment_counts.values()
colors = ['lightgreen', 'lightblue', 'lightcoral']
explode = (0.1, 0, 0)  # explode 1st slice (positive)
# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title('Sentiment Analysis of Google Play Store Reviews')
plt.show()
```



2.6. Analysing Google Play Store Reviews

Using dataset from KAGGLE – [dataset_reviews_2.csv]

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from textblob import TextBlob
# Load data from CSV
df = pd.read_csv('dataset_reviews_2.csv')
# Extract relevant columns and rows (from 10th to 20th)
relevant_data = df[['content', 'thumbsUpCount']].iloc[0:50000]
# Function to perform sentiment analysis
def analyze_sentiment(text):
    analysis = TextBlob(text)
    # Classify the polarity into positive, negative, and neutral
    if analysis.sentiment.polarity > 0.05:
        return 'positive'
    elif analysis.sentiment.polarity < -0.05:
        return 'negative'
    else:
        return 'neutral'
# Apply sentiment analysis to the review content
relevant_data['sentiment'] = relevant_data['content'].apply(analyze_sentiment)
# Initialize counters for each sentiment
sentiment_counts = {'positive': 0, 'negative': 0, 'neutral': 0}
# Aggregate thumbsUpCount based on sentiment
for index, row in relevant_data.iterrows():
    sentiment_counts[row['sentiment']] += row['thumbsUpCount']
# Convert counts to percentages
total_counts = sum(sentiment_counts.values())
for sentiment in sentiment_counts:
    sentiment_counts[sentiment] = (sentiment_counts[sentiment] / total_counts) * 100 if total_counts > 0 else 0
# Draw pie chart
plt.figure(figsize=(8, 8))
plt.pie(sentiment_counts.values(), labels=sentiment_counts.keys(), autopct='%1.1f%%')
plt.title('Sentiment Distribution of Google Play Store Reviews')
plt.show()
```



4. Conclusion

The project, aimed at performing sentiment analysis of Google Play Store reviews, successfully harnessed the capabilities of machine learning (ML) and natural language processing (NLP) to derive insights into user sentiments towards various applications. By employing three distinct models—two from the Hugging Face Transformers library and one based on the NLTK framework—this tool was able to analyze and categorize reviews into positive, neutral, or negative sentiments with a high degree of accuracy. The development of a full-stack web application, utilizing Flask for the backend and React along with TailwindCSS for the frontend, provided an accessible platform for users to interact with and benefit from the sentiment analysis tool in real-time. This project not only demonstrated the practical application of NLP in interpreting human language but also highlighted the pivotal role of sentiment analysis in understanding user feedback on digital platforms like the Google Play Store.

NLP's ability to process and analyze vast quantities of text data made it an invaluable asset in this project, enabling the automated categorization of user reviews at scale. This technology facilitated the extraction of meaningful patterns and trends from user-generated content, providing app developers and marketers with actionable insights into the perceived strengths and weaknesses of their products.

5. Future Enhancements

Looking forward, there are several areas for enhancement and expansion that could further refine the tool's accuracy and utility:

Advanced Sentiment Detection: Implementing more sophisticated NLP models that can detect nuanced sentiments and emotions, such as frustration, excitement, or disappointment, could provide deeper insights into user reviews. This includes exploring transformer-based models specifically fine-tuned for emotion detection.

Aspect-Based Sentiment Analysis: Moving beyond general sentiment analysis to aspect-based sentiment analysis (ABSA) could allow for the identification of specific features or aspects of an application that users mention in their reviews. This approach can help pinpoint exactly what users like or dislike about a product, offering more granular feedback to developers.

Language and Regional Support: Expanding the tool to include support for multiple languages and regional dialects would make it more universally applicable, allowing developers to gather insights from a global user base.

Temporal Analysis Features: Incorporating temporal analysis to track sentiment trends over time could help developers understand how updates or changes to their app affect user sentiment. This feature would be particularly useful for assessing the impact of new features or bug fixes.

Interactive Data Visualization: Enhancing the web application with interactive data visualization tools for sentiment analysis results could offer users a more intuitive understanding of the data. Visual analytics could include sentiment distribution over time, word clouds for frequently mentioned terms, and sentiment breakdown by app features.

Feedback Loop for Continuous Learning: Creating a mechanism for users to provide feedback on the accuracy of the sentiment analysis could help in continuously training and improving the models. This feedback loop could also include manual review of misclassified instances to refine the models further.

Integration with Developer Tools: Developing plugins or integrations for popular app development and analytics platforms could streamline the process of sentiment analysis for app developers, making it easier to incorporate insights directly into the development lifecycle.

By addressing these areas for future development, the project can evolve into an even more powerful tool for leveraging sentiment analysis to understand user feedback, ultimately helping developers to enhance their products and align more closely with user expectations.

THANK YOU