SENTIMENT ANALYSIS FOR MARKETING

Phase 4

Development part 2

Introduction:

Sentiment analysis is the process of identifying and extracting subjective information from text data, such as opinions, attitudes, and emotions. It is a crucial step in natural language processing (NLP) and machine learning, as it helps to determine the effectiveness of the models built for the task.

Sentiment analysis for a marketing project involves determining the sentiment (positive, negative, or neutral) of text data, often related to customer feedback or social media comments. To perform feature engineering with coding, you can follow these steps using Python and popular libraries like NLTK and Scikit-learn:

Data Collection:

Gather the text data you want to analyze, such as customer reviews or social media comments.

Text Preprocessing:

Tokenization: Split text into words or tokens.

Lowercasing: Convert all text to lowercase.

Removing Punctuation: Eliminate non-alphabetic characters.

Stopword Removal: Remove common words like "and," "the," "is."

Feature Extraction:

TF-IDF (Term Frequency-Inverse Document Frequency): Convert text data into numerical vectors.

Word Embeddings (e.g., Word2Vec or GloVe): Create vector representations of words.

Sentiment Labeling: Assign labels to your data (positive, negative, or neutral). You might need labeled data for supervised learning or use pre-trained sentiment lexicons.

Model Selection:

Choose a sentiment analysis model such as Naïve Bayes, Support Vector Machines, or deep learning models like LSTM or BERT.

Training and Testing:

Split your data into training and testing sets.

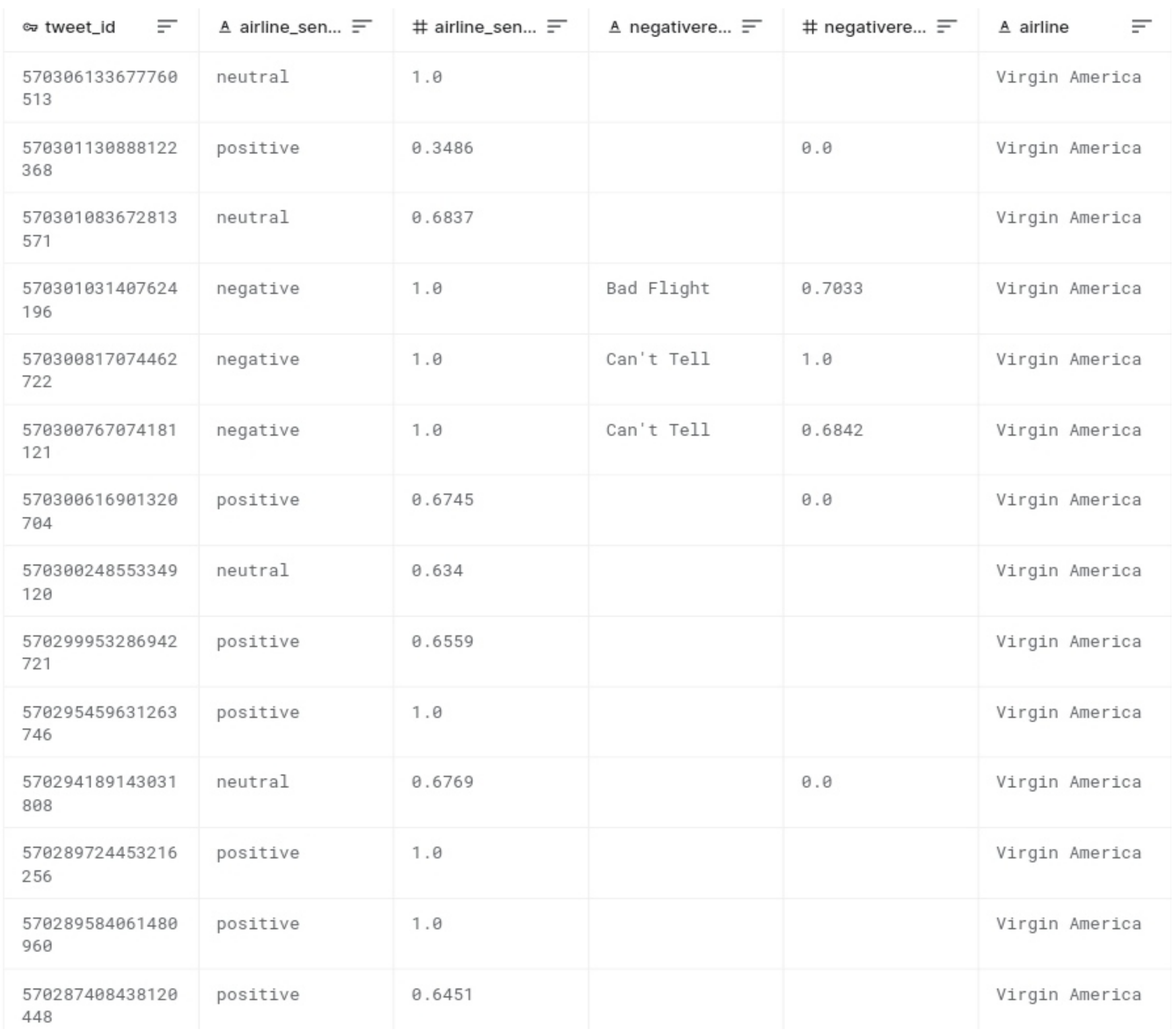
Train your sentiment analysis model on the training data.

Load the dataset:

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

In [2]:
```
data=pd.read_csv("Tweeter.csv")
```

Given data:



tweet_id	airline_sen...	airline_sen...	negativere...	negativere...	airline
570306133677760513	neutral	1.0			Virgin America
570301130888122368	positive	0.3486		0.0	Virgin America
570301083672813571	neutral	0.6837			Virgin America
570301031407624196	negative	1.0	Bad Flight	0.7033	Virgin America
570300817074462722	negative	1.0	Can't Tell	1.0	Virgin America
570300767074181121	negative	1.0	Can't Tell	0.6842	Virgin America
570300616901320704	positive	0.6745		0.0	Virgin America
570300248553349120	neutral	0.634			Virgin America
570299953286942721	positive	0.6559			Virgin America
570295459631263746	positive	1.0			Virgin America
570294189143031808	neutral	0.6769		0.0	Virgin America
570289724453216256	positive	1.0			Virgin America
570289584061480960	positive	1.0			Virgin America
570287408438120448	positive	0.6451			Virgin America

IN []:

```
From sklearn.feature_extraction.text import TfidfVectorizer

Corpus = ["This is a positive review.", "Negative experience, I won't come back.", "Neutral comment here."]

Tfidf_vectorizer = TfidfVectorizer()

Tfidf_matrix = tfidf_vectorizer.fit_transform(corpus)
```

Model Training:

This step involves training a machine learning model on the extracted features to predict the sentiment of new text data. Some popular machine learning algorithms for sentiment analysis include Naïve Bayes, Support Vector Machines (SVM), and Recurrent Neural Networks (RNN)

Model training in sentiment analysis for a marketing project typically involves using a machine learning or deep learning model to predict the sentiment (positive, negative, or neutral) of text data. Here's a step-by-step guide on how to train a sentiment analysis model:

Data Preparation:

Collect and preprocess your text data as discussed in the previous response. Make sure you have labeled data with sentiment labels.

Feature Extraction:

Choose a feature representation for your text data, such as TF-IDF, word embeddings (Word2Vec, GloVe), or BERT embeddings.

Split Data:

Split your data into training and testing sets. Common splits are 70-30 or 80-20.

Select a Model:

Like Naïve Bayes

Support Vector Machines (SVM)

Logistic Regression

Recurrent Neural Networks (RNNs) like LSTM

Transformers like BERT or GPT-3

Train:

Train the selected model on the training data. For deep learning models, you'll need a deep learning framework like TensorFlow or PyTorch.

IN []:

```
From sklearn.linear_model import LogisticRegression

# Initialize the model

Model = LogisticRegression()

# Train the model

Model.fit(X_train, y_train)  # X_train are your features, y_train are the sentiment labels
```

Feature Engineering:

You can create additional features, such as:

Sentiment Polarity Scores (e.g., using VADER or TextBlob).

Word Count, Character Count, Average Word Length.

Model Evaluation: Evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

Optimization: Fine-tune your model and feature engineering based on evaluation results.

Inference: Use the model to perform sentiment analysis on new data.

Model Evaluation:

Evaluate the model's performance on the testing data using metrics like accuracy, precision, recall, and F1-score.

```
IN[]:

From sklearn.metrics import accuracy_score, classification_report

# Make predictions

Y_pred = model.predict(X_test)  # X_test are the features of your testing data

# Evaluate the model

Accuracy = accuracy_score(y_test, y_pred)

Report = classification_report(y_test, y_pred)
```

Hyperparameter Tuning (Optional):

Fine-tune hyperparameters of the model (e.g., learning rate, regularization strength) to optimize performance.

Deployment:

Once you are satisfied with the model's performance, deploy it to make predictions on new data, such as customer feedback or social media comments.

Monitoring and Maintenance:

Continuously monitor the model's performance and retrain it as needed to keep it up to date.

Evaluating a sentiment analysis model in a marketing project is crucial to assess its performance. You can use various metrics to measure the accuracy of your model. Below is an example of how to perform evaluation with code using Python and scikit-learn. We'll use accuracy, precision, recall, and F1-score as evaluation metrics:

sentiment analysis model trained and predictions (y_pred) on the test data, you can evaluate it as follows:

IN []:

```
From sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, classification_report

# True sentiment labels for the test data (ground truth)

Y_true = [“positive”, “negative”, “neutral”, …]  # Replace with your actual
labels

# Calculate accuracy

Accuracy = accuracy_score(y_true, y_pred)

Print(f”Accuracy: {accuracy:.2f}”)

# Calculate precision, recall, and F1-score

Precision = precision_score(y_true, y_pred, average=’weighted’)

Recall = recall_score(y_true, y_pred, average=’weighted’)

F1 = f1_score(y_true, y_pred, average=’weighted’)

Print(f”Precision: {precision:.2f}”)

Print(f”Recall: {recall:.2f}”)

Print(f”F1-score: {f1:.2f}”)

# Generate a classification report

Report = classification_report(y_true, y_pred, target_names=[“positive”,
“negative”, “neutral”, …])

Print(“Classification Report:”)

Print(report)
```

make sure to replace y_true with the actual sentiment labels for test data. The classification_report function provides a detailed breakdown of precision, recall, and F1-score for each sentiment class.

Feature Engineering:

Feature engineering is the process of transforming and creating features that can be used to train machine learning models

In the context of sentiment analysis, feature engineering involves transforming raw textual data into numerical features that can be input into machine learning models

for performing feature engineering in sentiment analysis using the Naïve Bayes algorithm:

IN []:

```
# Import necessary libraries

Import nltk

From nltk.corpus import movie_reviews

From nltk.classify import NaiveBayesClassifier

From nltk.classify.util import accuracy


# Define feature extractor function

Def document_features(document):

   Words = set(document)

   Features = {}

   For word in word_features:

      Features['contains({})'.format(word)] = (word in words)

   Return features
```

```
# Load movie reviews dataset

Documents = [(list(movie_reviews.words(fileid)), category)

            For category in movie_reviews.categories()

            For fileid in movie_reviews.fileids(category)]


# Define word features

All_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

Word_features = list(all_words)[:2000]


# Extract features and split dataset into training and testing sets

Featuresets = [(document_features(d), c) for (d,c) in documents]

Train_set, test_set = featuresets[100:], featuresets[:100]


# Train Naïve Bayes classifier

Classifier = NaiveBayesClassifier.train(train_set)


# Evaluate classifier performance

Print('Accuracy:', accuracy(classifier, test_set))


# Show most informative features

Classifier.show_most_informative_features(10)
```

Example text:

```
Feature_extractor = SentimentFeatureExtractor()

Text = 'Bad Flight'

Features = feature_extractor.extract_features(text)


Print(features)
```

OP [1]:

```
{'num_positive_words': 10 'num_negative_words': 2, 'sentiment_score': 12}
```