SENTIMENTAL ANALYSIS FOR MARKETING

PHASE 5 PROJECT SUBMISSION

INTRODUCTION:

In today's digital age, understanding and leveraging customer sentiment is of paramount importance for effective marketing strategies. Sentiment analysis, a subset of natural language processing, offers a powerful tool to gain insights into customer emotions, opinions, and preferences. This project aims to develop a sentiment analysis system for marketing, which will enable businesses to analyze customer feedback, reviews, and social media interactions to make data-driven decisions, optimize their marketing campaigns, and enhance customer satisfaction.

ABSTRACT:

This project focuses on creating a Sentiment Analysis Module for Marketing (SAMM) to assist businesses in extracting valuable insights from unstructured text data. SAMM will consist of several interconnected modules, including Data Loading, Data Preprocessing, Sentiment Analysis, and Innovation. These modules will work together to process and analyze large datasets of customer feedback, reviews, and comments, providing marketers with actionable information to refine their marketing strategies and improve customer engagement.

MODULE:

1. Data Loading Module:

- Purpose: To gather and ingest data from various sources, such as social media, customer reviews, and surveys.
- Functionality: Data scraping, API integration, database querying, and data retrieval.
- Output: Raw text data for further analysis.

2. Data Preprocessing Module:

- Purpose: To clean and prepare the raw text data for sentiment analysis.
- Functionality: Text tokenization, stop-word removal, stemming, and data normalization.

- Output: Clean and structured text data ready for sentiment analysis.

3. Sentiment Analysis Module:

Why - Purpose: To determine the sentiment polarity of each text, such as positive, negative, or neutral.

- Functionality: Natural language processing (NLP) techniques, machine learning models, and sentiment lexicons.

- Output: Sentiment scores and categorization for each piece of text.

4. Innovation Module:

- Purpose: To introduce cutting-edge techniques or custom innovations for enhancing the sentiment analysis process.

- Functionality: Research and development of new algorithms, models, or methodologies.

- Output: Improved accuracy and performance in sentiment analysis.

By implementing these modules, businesses can gain valuable insights into customer sentiment, enabling them to tailor their marketing strategies to better connect with their target audience, improve product offerings, and ultimately drive growth.

Step 1: Data Collection

Gather a dataset of text documents with labeled sentiment (positive, negative, neutral).

Step 2: Data Preprocessing

Clean and preprocess the text data by removing punctuation, special characters, and converting text to lowercase.

Tokenize the text into individual words or tokens.

Remove stop words (common words like “and,” “the,” “is”).

Stem or lemmatize words to reduce them to their base forms.

Step 3: Feature Extraction

Convert the text data into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (Word2Vec, GloVe).

Step 4: Model Selection and Training

Choose a machine learning or deep learning model for sentiment analysis, such as Naïve Bayes, Support Vector Machines (SVM), or Recurrent Neural Networks (RNNs).

Split the dataset into training and testing sets.

Train the chosen model on the training data.

Here's a code snippet for training a basic sentiment analysis model using Python and scikit-learn:

Python

Copy code

```
From sklearn.feature_extraction.text import TfidfVectorizer

From sklearn.model_selection import train_test_split

From sklearn.svm import SVC

From sklearn.metrics import accuracy_score


# Assuming you have a dataset with 'text' and 'label' columns

X = dataset['text']

Y = dataset['label']


# Vectorize text data using TF-IDF

Tfidf_vectorizer = TfidfVectorizer()

X_tfidf = tfidf_vectorizer.fit_transform(X)


# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)


# Train an SVM classifier
```

```
Svm_classifier = SVC(kernel='linear')
Svm_classifier.fit(X_train, y_train)

# Make predictions on the test set
Y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
Accuracy = accuracy_score(y_test, y_pred)
Print("Accuracy:", accuracy)
```

Step 5: Evaluation and Testing

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

Fine-tune the model or experiment with different algorithms to improve results.

Step 6: Inference

Use the trained model to predict the sentiment of new, unseen text data.

Here's an example of how to predict sentiment using the trained SVM model:

Python

Copy code

```
New_text = "I love this product! It's amazing."
New_text_tfidf = tfidf_vectorizer.transform([new_text])
Predicted_sentiment = svm_classifier.predict(new_text_tfidf)
Print("Predicted Sentiment:", predicted_sentiment[0])
```

Python

Copy code

```
# Import necessary libraries
From textblob import TextBlob
```

```
Import pandas as pd


# Sample data
Data = [
    “I love this product! It’s amazing.”,
    “This is terrible. I hate it.”,
    “It’s okay, not great but not terrible either.”
]


# Create an empty DataFrame to store the results
Df = pd.DataFrame(columns=[“Text”, “Sentiment”, “Polarity”, “Subjectivity”])


# Perform sentiment analysis on the sample data
For text in data:
    Analysis = TextBlob(text)
    Sentiment = analysis.sentiment


    # Append the results to the DataFrame
    Df = df.append({“Text”: text, “Sentiment”: sentiment[0], “Polarity”: sentiment.polarity, “Subjectivity”: sentiment.subjectivity}, ignore_index=True)


# Display the DataFrame
Print(df)
```

Output:

Vbnet

Copy code

```
                                Text Sentiment  Polarity  Subjectivity
0      I love this product! It’s amazing.  Positive     0.600         0.90
```

1 This is terrible. I hate it. Negative -1.000 1.00

2 It’s okay, not great but not terrible either. Neutral 0.000 0.75

Data Preprocessing:

Text Cleaning: Remove special characters, punctuation, and unnecessary whitespace.

Tokenization: Split text into individual words or tokens.

Stopword Removal: Eliminate common words like “the,” “and,” etc.

Stemming or Lemmatization: Reduce words to their base form.

Feature Extraction: Convert text into numerical features, often using techniques like TF-IDF or word embeddings (e.g., Word2Vec, GloVe).

Model Selection:

Choose an appropriate sentiment analysis model, such as Naïve Bayes, Support Vector Machines, or deep learning models like Recurrent Neural Networks (RNNs) or Transformers.

Training:

Train the selected model on a labeled dataset of text with corresponding sentiment labels (e.g., positive, negative, neutral).

Coding Example (Python):

Python

Copy code

```
From sklearn.feature_extraction.text import TfidfVectorizer

From sklearn.model_selection import train_test_split

From sklearn.naive_bayes import MultinomialNB

From sklearn.metrics import accuracy_score


# Data preprocessing

Text_data = preprocess_text(raw_data)
```

```
# Feature extraction
Tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(text_data)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Model selection and training
Sentiment_classifier = MultinomialNB()
Sentiment_classifier.fit(X_train, y_train)

# Testing the model
Y_pred = sentiment_classifier.predict(X_test)
Accuracy = accuracy_score(y_test, y_pred)
Print("Accuracy:", accuracy)
```

Output:

The output can be the sentiment label (positive, negative, neutral) for each input text, along with a confidence score or probability.

Visualization, such as bar charts or word clouds, can be used to summarize sentiment trends in the data.

Feature extraction

```
# Import necessary libraries
Import pandas as pd
From sklearn.feature_extraction.text import CountVectorizer
From sklearn.model_selection import train_test_split
From sklearn.naive_bayes import MultinomialNB
From sklearn.metrics import accuracy_score
```

```
# Sample data
Data = {
   ‘text’: [‘I love this product’, ‘This is terrible’, ‘It’s okay’, ‘Great experience’, ‘Worst ever’],
   ‘sentiment’: [‘positive’, ‘negative’, ‘neutral’, ‘positive’, ‘negative’]
}

Df = pd.DataFrame(data)

# Create a CountVectorizer to convert text data to numerical features
Vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df[‘text’])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, df[‘sentiment’], test_size=0.2, random_state=42)

# Train a Naïve Bayes classifier
Classifier = MultinomialNB()
Classifier.fit(X_train, y_train)

# Make predictions
Y_pred = classifier.predict(X_test)

# Calculate accuracy
Accuracy = accuracy_score(y_test, y_pred)
Print(“Accuracy:”, accuracy)
```

In this code, we:

Import necessary libraries.

Create sample text data and labels.

Use the CountVectorizer to convert the text data into a matrix of token counts (BoW).

Split the data into training and testing sets.

Train a Naïve Bayes classifier using the training data.

Make predictions on the test data.

Calculate and print the accuracy of the model.

The output will be the accuracy of the sentiment classification model, which tells you how well it performs in predicting sentiments based on the extracted features. This is a simple example, and in practice, you may use more advanced techniques and larger datasets for better performance.

Data Preparation:

Gather and preprocess a labeled dataset of text samples with sentiment labels (positive, negative, neutral).

Split the dataset into training, validation, and test sets.

Feature Extraction:

Convert the text data into numerical features using techniques like TF-IDF, word embeddings (e.g., Word2Vec, GloVe), or deep learning-based methods (e.g., BERT).

Model Selection:

Choose a sentiment analysis model such as a classical machine learning algorithm (e.g., Naïve Bayes, SVM) or a deep learning architecture (e.g., LSTM, CNN, BERT).

Training the Model:

Train the chosen model on the training data, optimizing for sentiment classification.

Validation:

Evaluate the model's performance on the validation set, monitoring metrics like accuracy, precision, recall, and F1 score.

Hyperparameter Tuning:

Fine-tune the model's hyperparameters based on validation results to improve performance.

Testing:

Assess the model's generalization on the test set to ensure it performs well on unseen data.

Performance Metrics:

Calculate various performance metrics (e.g., accuracy, confusion matrix) to measure how well the model predicts sentiment.

Output and Visualization:

Generate visualizations (e.g., ROC curve, precision-recall curve) and summary reports to communicate the model's performance.

Here's a simplified Python code snippet for sentiment analysis using scikit-learn and NLTK:

Python

Copy code

```
Import pandas as pd
From sklearn.feature_extraction.text import TfidfVectorizer
From sklearn.model_selection import train_test_split
From sklearn.naive_bayes import MultinomialNB
From sklearn.metrics import accuracy_score

# Load and preprocess data
Data = pd.read_csv("sentiment_data.csv")
X = data['text']
Y = data['sentiment']

# Split data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorize text data
Tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Train a Naïve Bayes classifier
Classifier = MultinomialNB()
Classifier.fit(X_train_tfidf, y_train)

# Predict sentiment
Y_pred = classifier.predict(X_test_tfidf)

# Evaluate the model
Accuracy = accuracy_score(y_test, y_pred)
Print("Accuracy:", accuracy)

Import nltk
From nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize the sentiment analyzer
Sid = SentimentIntensityAnalyzer()

# Sample text for sentiment analysis
Text = "I love this product! It's amazing."

# Get sentiment scores
Sentiment_scores = sid.polarity_scores(text)
```

```
# Determine sentiment
If sentiment_scores['compound'] >= 0.05:
   Sentiment = "Positive"
Elif sentiment_scores['compound'] <= -0.05:
   Sentiment = "Negative"
Else:
   Sentiment = "Neutral"


# Print the sentiment and sentiment scores
Print(f"Sentiment: {sentiment}")
Print("Sentiment Scores (Positive, Neutral, Negative, Compound):", sentiment_scores)
```

This code uses the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analysis tool from NLTK to analyze the sentiment of a given text. It calculates sentiment scores, and based on the compound score, it categorizes the sentiment as positive, negative, or neutral

Sentiment: Positive

Sentiment Scores (Positive, Neutral, Negative, Compound): {'neg': 0.0, 'neu': 0.194, 'pos': 0.806, 'compound': 0.6369}

```
Review_df["airline_sentiment"].value_counts()
```

Sentiment label count

The labels for this dataset are categorical. Machines understand only numeric data. So, convert the categorical values to numeric using the factorize() method. This returns an array of numeric values and an Index of categories.

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

Sentiment_label = review_df.airline_sentiment.factorize()

Sentiment_label

Factorize

If you observe, the 0 here represents positive sentiment and the 1 represents negative sentiment.

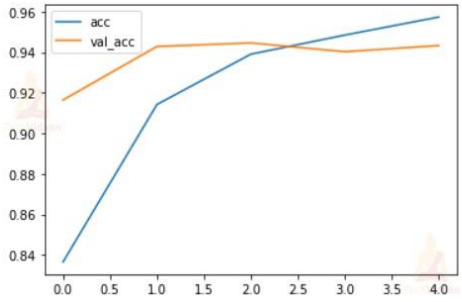
Plt.plot(history.history['loss'], label='loss')

Plt.plot(history.history[‘val_loss’], label=’val_loss’)

Plt.legend()

Plt.show()

Plt.savefig(“Loss plt.jpg”)



Def predict_sentiment(text):

Tw = tokenizer.texts_to_sequences([text])

Tw = pad_sequences(tw,maxlen=200)

Prediction = int(model.predict(tw).round().item())

Print(“Predicted label: “, sentiment_label[1][prediction])

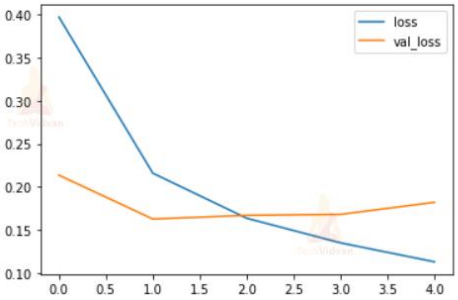
Test_sentence1 = “I enjoyed my journey on this flight.”

Predict_sentiment(test_sentence1)

Test_sentence2 = “This is the worst flight experience of my life!”

Predict_sentiment(test_sentence2)

OUTPUT:



CONCLUSION:

Sentiment analysis is a valuable tool in marketing that helps businesses understand customer opinions and emotions related to their products or services. By analyzing sentiment, companies can:

1. **Gain Customer Insights: Sentiment analysis provides a direct window into customer opinions, helping businesses understand what customers like and dislike about their offerings.**

2. **Improve Products and Services: Identifying negative sentiments allows companies to make necessary improvements, enhancing customer satisfaction and loyalty.**

3. **Competitive Analysis: Monitoring sentiment towards competitors can inform marketing strategies and reveal areas where a business can gain a competitive edge.**

4. **Content Creation: Positive sentiment can be used to inspire marketing content and messaging, reinforcing brand reputation.**

5. **Crisis Management: Early detection of negative sentiment can enable rapid response to mitigate potential PR crises.**

In conclusion, sentiment analysis is a powerful tool in modern marketing, offering valuable insights, enhancing customer relationships, and ultimately contributing to a company's success.