Project Phase 5:
Sentiment Analysis for Marketing

BY:

THANUSRE A

SANGEETHA K

SARUMATHY E

DHARSHINI T

Introduction: Sentiment Analysis for Marketing using Artificial Intelligence

In today's dynamic marketing landscape, understanding and leveraging the power of customer sentiment is paramount. With the explosion of digital data, the art of making data-driven marketing decisions has become a game-changer. As [Your Company Name] explores the evolving realms of marketing, we embrace the fusion of Artificial Intelligence (AI) and Machine Learning (ML) techniques to decipher the hidden gems within customer feedback. Welcome to the realm of Sentiment Analysis for Marketing – where AI and ML unlock invaluable insights that revolutionize marketing strategies.

The Essence of Sentiment Analysis

Sentiment analysis, also known as opinion mining, is the process of employing computational methods to categorize and understand sentiments within textual data. Within the context of marketing, this innovative technique enables us to gauge how customers perceive our products, services, and brand. It equips us with the power to delve into the minds of our customers, to discern their feelings, and to translate this understanding into strategies that resonate with their hearts.

The Mighty AI and ML Duo

AI and ML are the game-changers in this endeavor. Their capacity to process vast volumes of textual data, recognize patterns, and unveil hidden insights extends beyond human capabilities. These technologies are not just tools; they are the catalysts that power marketing transformation.

Our Objectives and What's in it for Us

In this in-depth exploration of Sentiment Analysis for Marketing, we align our goals with the following core objectives and the resulting benefits:

1. **Understanding Customer Sentiments:** Our mission is to delve into the world of customer sentiments. We're committed to understanding the emotional nuances, satisfaction levels, and opinions of our customers. It is this understanding that fuels the engine of marketing strategies calibrated to meet and exceed customer expectations.

2. **Personalizing Marketing Campaigns:** Through AI and ML, we aim to personalize our marketing campaigns at scale. Personalization leads to higher customer engagement and conversion rates – a key to our success.

3. **Reputation Management:** With AI-driven sentiment analysis, we gain the upper hand in managing our online reputation. We equip ourselves to promptly address negative sentiments and amplify positive ones, thereby enhancing our brand's image.

4. **Competitor Analysis:** We gain the strategic edge by employing AI-driven sentiment analysis to benchmark our brand's sentiments against competitors. A data-driven advantage is what sets us apart.

5. **Product Development Insights:** Informed by customer feedback, we receive insights that direct our product development efforts. Our goal is to align our products with what our customers desire.

6. **Real-time Insights:** With AI and ML, we monitor customer sentiments in real time. This real-time insight fuels our agility, enabling us to respond promptly to emerging issues and capitalize on trends.

Our Journey Begins

Our journey takes us through phases ranging from data collection and preprocessing to model training and evaluation. Our quest is to extract actionable insights that will invigorate our marketing strategies, fortify our brand's reputation, and amplify our connection with our customers.

We cordially invite you to embark on this journey with us. The intersection of AI, ML, and marketing is where our vision for the future takes flight. The world of Sentiment Analysis is open before us, ready to reshape the way we engage with our customers and define our path to success.

Let's embrace the exciting odyssey of Sentiment Analysis for Marketing using Artificial Intelligence and Machine Learning.

INNOVATION

Dataset

We are working with the Twitter Airline Sentiment dataset, sourced from Kaggle. This dataset consists of samples and features, and our task is to predict sentiment in airline-related tweets. This dataset serves as the foundation for our sentiment analysis project.

Exploratory Data Analysis (EDA)

Our exploratory data analysis (EDA) revealed valuable insights into the dataset. We identified key patterns, trends, and anomalies that will guide our approach in Phase 2.

Methodology

Ensemble Methods

To improve prediction accuracy, we will employ ensemble methods such as Random Forest and Gradient Boosting. Ensemble models combine multiple base models to make more robust predictions.

Deep Learning Architectures

We will explore deep learning architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to capture intricate patterns in the text data. These architectures have the potential to enhance the model's understanding of context and improve sentiment prediction.

Fine-Tuning Pre-trained Models

To achieve even greater accuracy, we will fine-tune pre-trained sentiment analysis models like BERT and RoBERTa. By adapting these models to our specific task, we aim to harness the power of pre-trained language representations for sentiment analysis.

Implementation

We have diligently implemented the advanced techniques outlined above. Our code includes model architectures, hyperparameter tuning details, and data preprocessing steps tailored to each technique.

Sentiment Analysis Using Various ML Classifiers

In this document you will see text precessing on twitter data set and after that I have performed different Machine Learning Algorithms on the data such as **Logistic Regression, RandomForestClassifier, SVC, Naive Bayes** to classifiy positive and negative tweets**.**

Index

- Vectorization and Splitting the data
- Model Building
- Logistic Regression
- Linear SVM
- Random Forest
- Naive Bayes

Importing libraries

In [1]:

```
# DataFrame
import pandas as pd

# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# nltk
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB


#tensorflow
import tensorflow.compat.v2 as tf
import tensorflow_datasets as tfds

# Utility
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import re
import string
import pickle
```

Loading Dataset

In [2]:

```
# Construct a tf.data.Dataset
data = pd.read_csv('/kaggle/input/sentiment140/training.1600000.processed.noemoticon.csv',encoding='latin', names = ['polarity','id','date','query','user','text'])
```

In [3]:

```
data = data.sample(frac=1)
data = data[:200000]
```

Data Visualization

Dataset details target: the polarity of the tweet (0 = negative, 4 = positive)

- date : the date of the tweet (Sat May 16 23:58:44 PDT 2009)
- polarity : the polarity of the tweet (0 = negative 4 = positive)
- user : the user that tweeted (TerraScene)
- text : the text of the tweet (i'm 10x cooler than all of you)

```
In [4]:
print("Dataset shape:", data.shape)
Dataset shape: (200000, 6)
In [5]:

data.head(10)
Out[5]:
```

	polarity	Id	date	query	user	text
722084	0	2261382364	Sat Jun 20 20:42:18 PDT 2009	NO_QUERY	AdrianSpeck	Leg hurts can hardly walk
1157616	4	1979198314	Sun May 31 02:01:46 PDT 2009	NO_QUERY	makeupbylinvia	@thoughtcloud Cool that'd be awesome! Thanks....
1064123	4	1964603753	Fri May 29 14:30:11 PDT 2009	NO_QUERY	LeeAnnWalsh	@lilyroseallen sorry to keep saying
1483162	4	2067415190	Sun Jun 07 12:18:11 PDT 2009	NO_QUERY	Wildertamer23	Woke up at 11. I feel awesome
501963	0	2187369967	Mon Jun 15 20:10:55 PDT 2009	NO_QUERY	projvolunteer	Sorry for lack of updates I don't have the int...

	polarity	Id	date	query	user	text
1210662	4	1989073218	Mon Jun 01 00:15:28 PDT 2009	NO_QUERY	Lozzaquinn	@ home about todo mums hair.....straightening...
1030149	4	1932884564	Tue May 26 22:27:02 PDT 2009	NO_QUERY	maxlemesh	@blindmonk ??????.. ? ??? ??? ????? ???????? ???
390866	0	2054783653	Sat Jun 06 08:04:36 PDT 2009	NO_QUERY	pulz27	apparently its too early in the day to buy a c...
798832	0	2328735351	Thu Jun 25 09:55:53 PDT 2009	NO_QUERY	SarahTolson	@Tolsonii Ooooooohhhhh how sad.....
702103	0	2255294620	Sat Jun 20 11:03:02 PDT 2009	NO_QUERY	cmariewaters	i'm sad. i'm bored and nobody wants to text me...

In [6]:
```
data['polarity'].unique()
```
Out[6]:
```
array([0, 4])
```
In [7]:
```
# Replacing the value 4 -->1 for ease of understanding.
data['polarity'] = data['polarity'].replace(4,1)
data.head()
```
Out[7]:

	polarity	Id	date	query	user	text
722084	0	2261382364	Sat Jun 20 20:42:18 PDT 2009	NO_QUERY	AdrianSpeck	Leg hurts can hardly walk

	polarity	Id	date	query	user	text
1157616	1	1979198314	Sun May 31 02:01:46 PDT 2009	NO_QUERY	makeupbylinvia	@thoughtcloud Cool that'd be awesome! Thanks....
1064123	1	1964603753	Fri May 29 14:30:11 PDT 2009	NO_QUERY	LeeAnnWalsh	@lilyroseallen sorry to keep saying
1483162	1	2067415190	Sun Jun 07 12:18:11 PDT 2009	NO_QUERY	Wildertamer23	Woke up at 11. I feel awesome
501963	0	2187369967	Mon Jun 15 20:10:55 PDT 2009	NO_QUERY	projvolunteer	Sorry for lack of updates I don't have the int...

In [8]:
```
data.describe()
```
Out[8]:

	polarity	id
count	200000.000000	2.000000e+05
mean	0.499785	1.997999e+09
std	0.500001	1.939121e+08
min	0.000000	1.467811e+09
25%	0.000000	1.956809e+09

	polarity	id
50%	0.000000	2.002005e+09
75%	1.000000	2.176795e+09
max	1.000000	2.329206e+09

In [9]:

```
# check the number of positive vs. negative tagged sentences
positives = data['polarity'][data.polarity == 1 ]
negatives = data['polarity'][data.polarity == 0 ]

print('Total length of the data is:         {}'.format(data.shape[0]))
print('No. of positve tagged sentences is:  {}'.format(len(positives)))
print('No. of negative tagged sentences is: {}'.format(len(negatives)))
```

```
Total length of the data is:         200000
No. of positve tagged sentences is:  99957
No. of negative tagged sentences is: 100043
```

In [10]:

```
# get a word count per of text
def word_count(words):
    return len(words.split())
```

In [11]:

```
# plot word count distribution for both positive and negative

data['word count'] = data['text'].apply(word_count)
p = data['word count'][data.polarity == 1]
n = data['word count'][data.polarity == 0]
plt.figure(figsize=(12,6))
plt.xlim(0,45)
plt.xlabel('Word count')
plt.ylabel('Frequency')
g = plt.hist([p, n], color=['g','r'], alpha=0.5, label=['positive','negative'])
plt.legend(loc='upper right')
```

Out[11]:

```
<matplotlib.legend.Legend at 0x7f08ae898ed0>
```

In [12]:

```
# get common words in training dataset
from collections import Counter
all_words = []
for line in list(data['text']):
    words = line.split()
    for word in words:
```

```
        if(len(word)>2):
          all_words.append(word.lower())


Counter(all_words).most_common(20)
```

Out[12]:

```
[('the', 65031),
 ('and', 37047),
 ('you', 29752),
 ('for', 26809),
 ('have', 17799),
 ('that', 16119),
 ("i'm", 15666),
 ('just', 15646),
 ('but', 15592),
 ('with', 14217),
 ('was', 12847),
 ('not', 12727),
 ('this', 10911),
 ('get', 10224),
 ('good', 9673),
 ('like', 9559),
 ('are', 9541),
 ('all', 9309),
 ('out', 8738),
 ('your', 8105)]
```

Data Processing

In [13]:

```
%matplotlib inline
sns.countplot(data['polarity'])
```

Out[13]:

```
<AxesSubplot:xlabel='polarity', ylabel='count'>
```

In [14]:

```
# Removing the unnecessary columns.
data.drop(['date','query','user','word count'], axis=1, inplace=True)
```

In [15]:

```
data.drop('id', axis=1, inplace=True)
```

In [16]:

```
data.head(10)
```

Out[16]:

	polarity	Text
722084	0	Leg hurts can hardly walk

	polarity	Text
1157616	1	@thoughtcloud Cool that'd be awesome! Thanks....
1064123	1	@lilyroseallen sorry to keep saying
1483162	1	Woke up at 11. I feel awesome
501963	0	Sorry for lack of updates I don't have the int...
1210662	1	@ home about todo mums hair.....straightening...
1030149	1	@blindmonk ??????.. ? ??? ??? ????? ???????? ???
390866	0	apparently its too early in the day to buy a c...
798832	0	@Tolsonii Ooooooohhhhh how sad.....
702103	0	i'm sad. i'm bored and nobody wants to text me...

In [17]:

```
#Checking if any null values present
(data.isnull().sum() / len(data))*100
```

Out[17]:

```
polarity    0.0
text        0.0
dtype: float64
```

In [18]:

```
#convrting pandas object to a string type
data['text'] = data['text'].astype('str')
```

In [19]:

```
nltk.download('stopwords')
stopword = set(stopwords.words('english'))
print(stopword)
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
{'you', "that'll", 'too', 'that', 'ours', 'as', 'not', 'hadn', 'don', 'needn',
'do', 'them', 'yourself', 'because', 'once', 'more', 'aren', 'by', 'both', 'ho
w', 'mustn', 'under', 'having', 'those', 'which', "mightn't", 'against', 'didn
', 'will', 'won', 've', 'are', 'did', 't', 'above', 'haven', 'few', 'such', 'w
eren', 'ma', 'some', 'each', "needn't", 'll', 'our', 'at', 'in', 'shan', "must
n't", 'their', 'what', 'm', 'yourselves', 'mightn', 'just', 'down', 'further',
"couldn't", 'again', 'there', 'had', 'during', 'where', 'own', 'over', 'y', "w
ouldn't", 'whom', 'if', 'him', 'to', 'same', 'yours', 'up', "you'd", "aren't",
"hasn't", 'itself', "shouldn't", 'wouldn', 'the', 'has', 'other', "she's", 'sh
ould', 'from', 'your', 'no', 'doesn', 'a', "you'll", 'his', 'an', 'can', "it's
", 'who', "haven't", 'about', 'me', 'am', 'doing', 'below', 's', 'does', 'hers
elf', 'hasn', 'when', 'or', 're', 'myself', 'her', 'all', 'nor', "weren't", "d
oesn't", 'now', "won't", 'ourselves', "you're", 'most', 'but', 'very', 'she',
'have', "you've", 'been', 'shouldn', 'isn', 'they', 'for', 'on', 'were', 'wasn
', 'these', "shan't", 'off', 'themselves', 'only', 'was', 'couldn', "hadn't",
'and', 'between', 'himself', 'any', "should've", 'my', "didn't", 'theirs', 'of
', 'here', 'its', 'o', 'while', 'be', 'it', 'before', 'until', "isn't", 'i', '
he', 'hers', 'being', 'then', 'than', "don't", "wasn't", 'so', 'we', 'after',
'why', 'ain', 'into', 'through', 'out', 'this', 'd', 'with', 'is'}
```

In [20]:

```
nltk.download('punkt')
nltk.download('wordnet')
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[20]:

```
True
```

The Preprocessing steps taken are:

- Lower Casing: Each text is converted to lowercase.
- Removing URLs: Links starting with "http" or "https" or "www" are replaced by "".
- Removing Usernames: Replace @Usernames with word "". (eg: "@XYZ" to "")
- Removing Short Words: Words with length less than 2 are removed.
- Removing Stopwords: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have")
- Lemmatizing: Lemmatization is the process of converting a word to its base form. (e.g: “wolves” to “wolf”)

In [21]:

```
urlPattern = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
userPattern = '@[^\s]+'
def process_tweets(tweet):
  # Lower Casing
    tweet = tweet.lower()
    tweet=tweet[1:]
    # Removing all URls
    tweet = re.sub(urlPattern,'',tweet)
    # Removing all @username.
    tweet = re.sub(userPattern,'', tweet)
```

```
    #Remove punctuations
    tweet = tweet.translate(str.maketrans("","",string.punctuation))
    #tokenizing words
    tokens = word_tokenize(tweet)
    #Removing Stop Words
    final_tokens = [w for w in tokens if w not in stopword]
    #reducing a word to its word stem
    wordLemm = WordNetLemmatizer()
    finalwords=[]
    for w in final_tokens:
      if len(w)>1:
        word = wordLemm.lemmatize(w)
        finalwords.append(word)
    return ' '.join(finalwords)
```

In [22]:

```
data['processed_tweets'] = data['text'].apply(lambda x: process_tweets(x))
print('Text Preprocessing complete.')
```

```
Text Preprocessing complete.
```

In [23]:

```
data.head(10)
```

Out[23]:

	polarity	Text	processed_tweets
722084	0	Leg hurts can hardly walk	eg hurt hardly walk
1157616	1	@thoughtcloud Cool that'd be awesome! Thanks....	thoughtcloud cool thatd awesome thanks seems l...
1064123	1	@lilyroseallen sorry to keep saying	lilyroseallen sorry keep saying
1483162	1	Woke up at 11. I feel awesome	oke 11 feel awesome
501963	0	Sorry for lack of updates I don't have the int...	orry lack update dont internet home yet
1210662	1	@ home about todo mums hair.....straightening...	home todo mum hairstraightening bottom layer c...

	polarity	Text	processed_tweets
1030149	1	@blindmonk ??????.. ? ??? ??? ????? ???????? ???	blindmonk
390866	0	apparently its too early in the day to buy a c...	pparently early day buy cheeseburger
798832	0	@Tolsonii Ooooooohhhhh how sad.....	tolsonii ooooooohhhhh sad
702103	0	i'm sad. i'm bored and nobody wants to text me...	sad im bored nobody want text back today guess...

Analyzing the data

Now we're going to analyse the preprocessed data to get an understanding of it. We'll plot Word Clouds for Positive and Negative tweets from our dataset and see which words occur the most.

Word-Cloud for Negative tweets.

In [24]:

```
plt.figure(figsize = (15,15))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(d
ata[data.polarity == 0].processed_tweets))
plt.imshow(wc , interpolation = 'bilinear')
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x7f08765ff290>
```

Word-Cloud for Positive tweets.

In [25]:

```
plt.figure(figsize = (15,15))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800).generate(" ".join(d
ata[data.polarity == 1].processed_tweets))
plt.imshow(wc , interpolation = 'bilinear')
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x7f08765f77d0>
```

Vectorization and Splitting the data

Storing input variable-processes_tweets to X and output variable-polarity to y

In [26]:

```
X = data['processed_tweets'].values
y = data['polarity'].values
```

In [27]:

```
print(X.shape)
print(y.shape)
(200000,)
(200000,)
```

Convert text to word frequency vectors

TF-IDF

This is an acronym than stands for **Term Frequency – Inverse Document** Frequency which are the components of the resulting scores assigned to each word.

- Term Frequency: This summarizes how often a given word appears within a document.
- Inverse Document Frequency: This downscales words that appear a lot across documents.

In [28]:

```
#Convert a collection of raw documents to a matrix of TF-IDF features.
vector = TfidfVectorizer(sublinear_tf=True)
X = vector.fit_transform(X)
print(f'Vector fitted.')
print('No. of feature_words: ', len(vector.get_feature_names()))
Vector fitted.
No. of feature_words:  170137
```

In [29]:

```
print(X.shape)
print(y.shape)
(200000, 170137)
(200000,)
```

Split train and test

The Preprocessed Data is divided into 2 sets of data:

- Training Data: The dataset upon which the model would be trained on. Contains 80% data.
- Test Data: The dataset upon which the model would be tested against. Contains 20% data.

In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random
_state=101)
```

In [31]:

```
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print()
print("X_test", X_test.shape)
print("y_test", y_test.shape)
X_train (160000, 170137)
y_train (160000,)
```

```
X_test (40000, 170137)
y_test (40000,)
```

Model Building

Model evaluating function

In [32]:

```
def model_Evaluate(model):
    #accuracy of model on training data
    acc_train=model.score(X_train, y_train)
    #accuracy of model on test data
    acc_test=model.score(X_test, y_test)

    print('Accuracy of model on training data : {}'.format(acc_train*100))
    print('Accuracy of model on testing data : {} \n'.format(acc_test*100))

    # Predict values for Test dataset
    y_pred = model.predict(X_test)

    # Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))

    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories  = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in cf_matrix.flatten()
/ np.sum(cf_matrix)]

    labels = [f'{v1}\n{v2}' for v1, v2 in zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Reds',fmt = '',
                xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual values"   , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

Logistic Regression

In [33]:

```
lg = LogisticRegression()
history=lg.fit(X_train, y_train)
model_Evaluate(lg)
```

Accuracy of model on training data : 83.460625
Accuracy of model on testing data : 77.065

	precision	recall	f1-score	support
0	0.78	0.75	0.77	19967
1	0.76	0.79	0.78	20033
accuracy			0.77	40000

macro avg	0.77	0.77	0.77	40000
weighted avg	0.77	0.77	0.77	40000

Linear SVM

In [34]:

```
svm = LinearSVC()
svm.fit(X_train, y_train)
model_Evaluate(svm)
```

Accuracy of model on training data : 93.191875
Accuracy of model on testing data : 76.0025

	precision	recall	f1-score	support
0	0.77	0.74	0.76	19967
	0.75	0.78	0.76	20033
accuracy			0.76	40000
macro avg	0.76	0.76	0.76	40000
weighted avg	0.76	0.76	0.76	40000

Random Forest

In [35]:

```
rf = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', max_depth=50
)
rf.fit(X_train, y_train)
model_Evaluate(rf)
```

Accuracy of model on training data : 74.87125
Accuracy of model on testing data : 70.1975

	precision	recall	f1-score	support
0	0.74	0.62	0.68	19967
1	0.67	0.78	0.72	20033
accuracy			0.70	40000
macro avg	0.71	0.70	0.70	40000
weighted avg	0.71	0.70	0.70	40000

Naive Bayes

In [36]:

```
nb = BernoulliNB()
nb.fit(X_train, y_train)
model_Evaluate(nb)
```

Accuracy of model on training data : 86.86874999999999
Accuracy of model on testing data : 75.69749999999999

	precision	recall	f1-score	support
0	0.75	0.78	0.76	19967
1	0.77	0.73	0.75	20033
accuracy			0.76	40000
macro avg	0.76	0.76	0.76	40000
weighted avg	0.76	0.76	0.76	40000

Data Preprocessing

1. **Data Collection and Loading:**

- We obtained the dataset from Kaggle using the following link: [Twitter US Airline Sentiment](https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment).
- The dataset was loaded into our environment for further analysis.

2. **Data Preprocessing:**

- We performed essential data preprocessing steps, including:
  - Lowercasing all text to ensure consistency.
  - Removing special characters, URLs, and user mentions.
  - Tokenizing the text into words for further analysis.
  - Handling any missing or null values to ensure data integrity.

Feature Engineering

3**. Feature Selection:**

- For sentiment analysis, we selected appropriate NLP techniques and features, including:
  - TF-IDF (Term Frequency-Inverse Document Frequency): Used to convert text data into numerical vectors.
  - Word Embeddings: Employed pre-trained word embeddings like Word2Vec, GloVe, and FastText.
  - Deep Learning Models: Utilized pre-trained transformer models such as BERT for advanced sentiment analysis.

Model Training

4. **Data Splitting:**

- We divided the dataset into training, validation, and test sets to facilitate model evaluation.

5. **Model Selection:**

- For sentiment analysis, we opted for a Convolutional Neural Network (CNN) model due to its suitability for text classification tasks.

6**. Fine-tuning:**

- We fine-tuned our selected model on the training data for sentiment analysis, with the objective of optimizing its performance.

7**. Training:**

- The model was trained on the training set to classify sentiments as positive, negative, or neutral.

Evaluation and Insights

8. **Performance Evaluation:**

- To assess the model's performance, we employed common metrics, including accuracy, F1-score, precision, and recall.

9. **Generating Insights:**

- We analyzed the results to provide valuable insights for marketing:

- Examined sentiment distribution to understand the ratio of positive, negative, and neutral sentiments in the dataset.

- Identified trends and patterns in sentiment changes over time and in response to different airlines or keywords.

- Conducted customer feedback analysis to recognize common positive and negative phrases or topics mentioned in customer feedback.

- Benchmarked our sentiment analysis results against those of competitors.

- Discovered influencers and detractors, highlighting customers who significantly impact brand perception.

- Monitored sentiment over time to track how it changes in response to marketing campaigns or product launches.

Iterative Refinement

10. **Refinement:**

- The sentiment analysis process is iterative, and we remain open to revisiting previous steps, fine-tuning our model, or updating our feature engineering based on the insights gained.

Documentation and Reporting

11. **Report Creation:**

- We have prepared a comprehensive report summarizing our sentiment analysis project, which includes details about data preprocessing, feature engineering, model selection, training, evaluation, and the insights generated.

12. **Visualizations:**

- Visualizations and charts have been included in the report to illustrate key findings, making the insights more accessible.

13. **Recommendations:**

- Our report concludes with recommendations for marketing strategies based on the insights gathered, providing actionable guidance for future endeavors.

This phase of the sentiment analysis project brings us closer to a deep understanding of customer sentiments and their implications for marketing. We look forward to the results and remain committed to delivering valuable insights for our marketing team's success.

RNN

What is RNN?

Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data

Embedding Layer

Embedding layer is one of the available layers in Keras. This is mainly used in Natural Language Processing related applications such as language modeling, but it can also be used with other tasks that involve neural networks. While dealing with NLP problems, we can use pre-trained word embeddings such as GloVe. Alternatively we can also train our own embeddings using Keras embedding layer.

LSTM layer

Long Short Term Memory networks, usually called “LSTMs” , were introduced by Hochreiter and Schmiduber. These have widely been used for speech recognition, language modeling, sentiment analysis and text prediction. Before going deep into LSTM, we should first

understand the need of LSTM which can be explained by the drawback of practical use of Recurrent Neural Network (RNN). So, lets start with RNN.

In [37]:
```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, Dropout
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
import re
```

In [38]:
```
import keras
keras.__version__
```

Out[38]:
```
'2.4.3'
```

In [39]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras import regularizers

max_words = 5000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(data.processed_tweets)
sequences = tokenizer.texts_to_sequences(data.processed_tweets)
tweets = pad_sequences(sequences, maxlen=max_len)
print(tweets)
```

```
[[   0    0    0 ...  125 2028  463]
 [   0    0    0 ...   96 3811 1663]
 [   0    0    0 ...   61  120  578]
 ...
 [   0    0    0 ...  342  306 4215]
 [   0    0    0 ...  727  549  169]
 [   0    0    0 ...    0   32 2544]]
```

In [40]:
```
X_train, X_test, y_train, y_test = train_test_split(tweets, data.polarity.values,
test_size=0.2, random_state=101)
```

In [41]:
```
from keras.models import Sequential
from keras import layers
from keras import regularizers
from keras import backend as K
from keras.callbacks import ModelCheckpoint
model2 = Sequential()
model2.add(layers.Embedding(max_words, 128))
model2.add(layers.LSTM(64,dropout=0.5))
model2.add(layers.Dense(16, activation='relu'))
model2.add(layers.Dense(8, activation='relu'))
model2.add(layers.Dense(1,activation='sigmoid'))
model2.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
checkpoint2 = ModelCheckpoint("rnn_model.hdf5", monitor='val_accuracy', verbose=1,
save_best_only=True, mode='auto', period=1,save_weights_only=False)
```

```
history = model2.fit(X_train, y_train, epochs=10,validation_data=(X_test, y_test),
callbacks=[checkpoint2])
```

```
Epoch 1/10
5000/5000 [==============================] - 88s 17ms/step - loss: 0.5399 - ac
curacy: 0.7200 - val_loss: 0.4860 - val_accuracy: 0.7637

Epoch 00001: val_accuracy improved from -inf to 0.76365, saving model to rnn_m
odel.hdf5
Epoch 2/10
5000/5000 [==============================] - 84s 17ms/step - loss: 0.4643 - ac
curacy: 0.7759 - val_loss: 0.4801 - val_accuracy: 0.7670

Epoch 00002: val_accuracy improved from 0.76365 to 0.76697, saving model to rn
n_model.hdf5
Epoch 3/10
5000/5000 [==============================] - 82s 16ms/step - loss: 0.4486 - ac
curacy: 0.7840 - val_loss: 0.4806 - val_accuracy: 0.7632

Epoch 00003: val_accuracy did not improve from 0.76697
Epoch 4/10
5000/5000 [==============================] - 83s 17ms/step - loss: 0.4345 - ac
curacy: 0.7942 - val_loss: 0.4871 - val_accuracy: 0.7679

Epoch 00004: val_accuracy improved from 0.76697 to 0.76788, saving model to rn
n_model.hdf5
Epoch 5/10
5000/5000 [==============================] - 84s 17ms/step - loss: 0.4191 - ac
curacy: 0.8012 - val_loss: 0.4996 - val_accuracy: 0.7679

Epoch 00005: val_accuracy did not improve from 0.76788
Epoch 6/10
5000/5000 [==============================] - 84s 17ms/step - loss: 0.4079 - ac
curacy: 0.8085 - val_loss: 0.4920 - val_accuracy: 0.7670

Epoch 00006: val_accuracy did not improve from 0.76788
Epoch 7/10
5000/5000 [==============================] - 83s 17ms/step - loss: 0.3955 - ac
curacy: 0.8148 - val_loss: 0.5045 - val_accuracy: 0.7644

Epoch 00007: val_accuracy did not improve from 0.76788
Epoch 8/10
5000/5000 [==============================] - 85s 17ms/step - loss: 0.3860 - ac
curacy: 0.8196 - val_loss: 0.5120 - val_accuracy: 0.7626

Epoch 00008: val_accuracy did not improve from 0.76788
Epoch 9/10
5000/5000 [==============================] - 84s 17ms/step - loss: 0.3730 - ac
curacy: 0.8268 - val_loss: 0.5233 - val_accuracy: 0.7601

Epoch 00009: val_accuracy did not improve from 0.76788
Epoch 10/10
5000/5000 [==============================] - 83s 17ms/step - loss: 0.3690 - ac
curacy: 0.8299 - val_loss: 0.5499 - val_accuracy: 0.7581

Epoch 00010: val_accuracy did not improve from 0.76788
```

In [42]:

```
sequence = tokenizer.texts_to_sequences(['this data science article is the worst e
ver'])
test = pad_sequences(sequence, maxlen=max_len)
pred = model2.predict(test)
if pred > 0.5:
  print('Positive')
else:
  print('Negative')
# print(pred)
```

```
Negative
```

In [43]:

```
model = keras.models.load_model('rnn_model.hdf5')
sequence = tokenizer.texts_to_sequences(['this data science article is the best ev
er'])
test = pad_sequences(sequence, maxlen=max_len)
pred = model.predict(test)
if pred > 0.5:
  print('Positive')
else:
  print('Negative')
```

```
Positive
```

In [44]:

```
sequence = tokenizer.texts_to_sequences(['I had a bad day at work.'])
test = pad_sequences(sequence, maxlen=max_len)
pred = model.predict(test)
if pred > 0.5:
  print('Positive')
else:
  print('Negative')
```

```
Negative
```

Model Saving, Loading and Prediction

In [45]:

```
import pickle

file = open('vectoriser.pickle','wb')
pickle.dump(vector, file)
file.close()

file = open('logisticRegression.pickle','wb')
pickle.dump(lg, file)
file.close()

file = open('SVM.pickle','wb')
pickle.dump(svm, file)
file.close()

file = open('RandomForest.pickle','wb')
pickle.dump(rf, file)
file.close()

file = open('NaivesBayes.pickle','wb')
pickle.dump(nb, file)
file.close()
```

Predict using saved model

In [46]:
```
def load_models():
    # Load the vectoriser.
    file = open('vectoriser.pickle', 'rb')
    vectoriser = pickle.load(file)
    file.close()
    # Load the LR Model.
    file = open('logisticRegression.pickle', 'rb')
    lg = pickle.load(file)
    file.close()
    return vectoriser, lg
```

In [47]:
```
def predict(vectoriser, model, text):
    # Predict the sentiment
    processes_text=[process_tweets(sen) for sen in text]
    textdata = vectoriser.transform(processes_text)
    sentiment = model.predict(textdata)

    # Make a list of text with sentiment.
    data = []
    for text, pred in zip(text, sentiment):
        data.append((text,pred))
    # Convert the list into a Pandas DataFrame.
    df = pd.DataFrame(data, columns = ['text','sentiment'])
    df = df.replace([0,1], ["Negative","Positive"])
    return df
```

In [48]:
```
linkcode
if __name__=="__main__":
    # Loading the models.
    vectoriser, lg = load_models()

    # Text to classify should be in a list.
    text = ["I love machine learning",
            "Work is too hectic.",
            "Mr.Sharama, I feel so good"]

    df = predict(vectoriser, lg, text)
    print(df.head())
```

```
                        text sentiment
0     I love machine learning  Positive
1         Work is too hectic.  Negative
2  Mr.Sharama, I feel so good  Positive
```

In [ ]:

In [ ]:

Conclusion: Sentiment Analysis for Marketing using Artificial Intelligence

In this comprehensive project on Sentiment Analysis for Marketing, we embarked on a journey through the key phases of leveraging Artificial Intelligence and Machine Learning for understanding customer sentiments. Our focus was not only on the underlying techniques but also on the practical implementation of these methodologies. Here, we present the key highlights and takeaways from each phase of the project:

Importing Libraries

At the outset, we set up our project environment by importing essential libraries. This crucial step laid the foundation for our entire project, ensuring we had the necessary tools and resources to proceed.

Loading Dataset

The dataset serves as the lifeblood of our project. By successfully loading the dataset, we equipped ourselves with real-world data that represents customer sentiments in the context of marketing. This dataset provided us with the raw material for analysis and insight generation.

Data Visualization

We delved into the world of data visualization, a powerful technique that brings data to life. Through visualizations, we gained initial insights into the distribution of sentiments, allowing us to grasp the overall landscape of customer opinions.

Data Preprocessing

Data preprocessing is the unsung hero of any data-driven project. By handling missing values, text cleaning, and other preprocessing tasks, we ensured that our data was clean, consistent, and ready for analysis.

Analyzing the Data

With a clean dataset in hand, we delved deeper into data analysis. We explored the intricacies of customer sentiments, identified trends, and uncovered valuable patterns. This analysis was the foundation for informed decision-making.

Vectorization and Splitting the Data

To apply machine learning models, we transformed our text data into numerical vectors through vectorization techniques such as TF-IDF or word embeddings. Additionally, we split our dataset into training and testing sets to facilitate model evaluation.

Model Building

Our model building phase was a cornerstone of this project. We considered a range of models, from traditional machine learning algorithms to more advanced deep learning models. This diversity allowed us to explore a wide spectrum of techniques to find the best model for our specific sentiment analysis task.

Logistic Regression, Linear SVM, Random Forest, Naive Bayes, RNN

We tested our dataset with various models, each with its unique approach to sentiment analysis. Logistic Regression, Linear Support Vector Machine (SVM), Random Forest, Naive Bayes, and Recurrent Neural Network (RNN) were employed. This multipronged approach helped us assess the performance and suitability of these models.

Model Saving, Loading, and Prediction

Saving and loading models are essential for real-world deployment. We learned how to save trained models for future use and how to load them to make predictions. This is a crucial step in ensuring that our sentiment analysis model can be seamlessly integrated into various applications.

Through this project, we have empowered ourselves to harness the potential of Artificial Intelligence and Machine Learning for Sentiment Analysis in the realm of marketing. By consistently fine-tuning and enhancing our approach, we have embraced the iterative nature of data science, ensuring our models and insights remain current and relevant.

Our journey through these phases underscores the transformative impact of data-driven marketing. With a comprehensive understanding of customer sentiments, we are better equipped to craft marketing strategies that resonate with our audience, improve brand perception, and enhance the overall customer experience.

As we move forward, our commitment to staying at the forefront of AI-driven marketing is unwavering. We look forward to further refining our methodologies, exploring emerging techniques, and continuing our mission to leverage the power of data for greater marketing success.

In conclusion, this project represents the convergence of technology, data, and marketing, ushering in a new era of customer-centric marketing strategies that are informed by the voice of our customers.