# SURYA GROUP OF INSTITUTIONS

# NAAN MUDHALVAN

IBM-ARTIFICIAL INTELLIGENCE

VIMAL RAJ M

422221104047

Sentiment Analysis For Marketing

TEAM : 05

# Sentiment Analysis For Marketing Machine Learning Algorithm In Python

Sentiment analysis can be an invaluable tool for organizations to identify and address their customers' pain points. In a Repustate case study, a bank in South Africa noticed that many users had stopped doing business with them and they were concerned as to why this was happening. To gain further clarity on this issue, they collected social media data to understand what their customers were saying about them.

The bank realized that many of their clients were dissatisfied with the customer service: long waiting times (especially during lunch and peak hours) and even the operating hours were inconvenient. Performing sentiment analysis on over 2 million pieces of text data, they not only had identified the issue, but now knew how to resolve it.

Management improved the operating hours and increased the number of tellers in each branch. They also never had unmanned teller stations during lunchtime or peak hours to ensure that customers were served on time. As a result, there was a significant drop in customer churn rates and a rise in the number of new clients.

Essentially, sentiment analysis is the process of mining text data to extract the underlying emotion behind it in order to add value and pinpoint critical issues in a business. In this tutorial, I will show you how to build your own sentiment analysis model in Python, step by step.

## Table of Contents:


How to Perform Sentiment Analysis in Python?

- Python Pre-Requisites
- Reading the Dataset
- Data Preprocessing
- TF-IDF Transformation
- Building and Evaluating the ML Model

Sentiment Analysis with Python: Next Steps


# How to Perform Sentiment Analysis in Python?

You're probably already familiar with Python, but if not – it is a powerful programming language with an intuitive syntax. Not to mention it's one of the most popular choices across the data science community, which makes it perfect for our tutorial.

We will use the Trip Advisor Hotel Reviews Kaggle dataset for this analysis, so make sure to have it downloaded before you start to be able to code along.

# Step 1: Python Pre-Requisites:

First things first: installing the necessary equipment. You need a Python IDE – I suggest using Jupyter. (If you don't already have it, follow this Jupyter Notebook tutorial to set it up on your device.)

Make sure to have the following libraries installed as well: NumPy, pandas, Matplotlib, seaborn, Regex, and scikit-learn.

## Step 2: Reading the Dataset:

Let's start by loading the dataset into Python and reading the head of the data frame:.

```
import pandas as pd

df = pd.read_csv('tripadvisor_hotel_reviews.csv')

df.head()
```

The code above should render the following output:

This dataset only has 2 variables: "Review" which contains guests' impressions of the hotel and "Rating" - the corresponding numerical evaluation (or, in simpler terms, the number of stars they've left).

| | Review | Rating |
|---|---|---|
| 0 | nice hotel expensive parking got good deal sta... | 4 |
| 1 | ok nothing special charge diamond member hilto... | 2 |
| 2 | nice rooms not 4* experience hotel monaco seat... | 3 |
| 3 | unique, great stay, wonderful time hotel monac... | 5 |
| 4 | great stay great stay, went seahawk game aweso... | 5 |

Now, let's take a look at the number of rows in the data frame:

```
len(df.index) # 20491
```

## Step 3: Data Preprocessing:

As we already know the TripAdvisor dataset has 2 variables – user reviews and ratings, which range from 1 to 5. We will use “Ratings” to create a new variable called “Sentiment.” In it, we will add 2 categories of sentiment as follows:

0 to 1 will be encoded as -1 as they indicate negative sentiment

3 will be labeled as 0 as it has a neutral sentiment

4 and 5 will be labeled as +1 as they indicate positive sentiment

Let’s create a Python function to accomplish this categorization:

```
import numpy as np

def create_sentiment(rating):

  if rating==1 or rating==2:
    return -1 # negative sentiment
  elif rating==4 or rating==5:
    return 1 # positive sentiment
  else:
    return 0 # neutral sentiment

df['Sentiment'] = df['Rating'].apply(create_sentiment)
```

Now, let’s take a look at the head of the data frame again

| | Review | Rating | Sentiment |
|---|---|---|---|
| 0 | nice hotel expensive parking got good deal sta... | 4 | 1 |
| 1 | ok nothing special charge diamond member hilto... | 2 | -1 |
| 2 | nice rooms not experience hotel monaco seattl... | 3 | 0 |
| 3 | unique great stay wonderful time hotel monaco ... | 5 | 1 |
| 4 | great stay great stay went seahawk game awesom... | 5 | 1 |

First, however, we need to preprocess the “Review” column in order to remove punctuation, characters, and digits. The code looks like this:

```
from sklearn.feature_extraction.text import re

def clean_data(review):

    no_punc = re.sub(r'[^\w\s]', '', review)
    no_digits = ''.join([i for i in no_punc if not i.isdigit()])

    return(no_digits)
```

In this way, we will eliminate unnecessary noise and only retain information that is valuable to the final sentiment analysis.

```
df['Review'][0]
```

```
'nice hotel expensive parking got good deal stay hotel anniversary, arrived late evening took advice previous reviews did valet
parking, check quick easy, little disappointed non-existent view room room clean nice size, bed comfortable woke stiff neck hig
h pillows, not soundproof like heard music room night morning loud bangs doors opening closing hear people talking hallway, may
be just noisy neighbors, aveda bath products nice, did not goldfish stay nice touch taken advantage staying longer, location gr
eat walking distance shopping, overall nice experience having pay 40 parking night,  '
```

# Step 4: TF-IDF Transformation:

Now, we need to convert this text data into a numeric representation so that it can be ingested into the ML model. We will do this with the help of scikit-learn's TF-IDF Vectorizer package.

TF-IDF stands for "term frequency-inverse document frequency" – a statistical measure that tells us how relevant a word is to a document in a collection. In simpler terms, it converts words into a vector of numbers where each word has its own numeric representation.

TF-IDF is calculated based on 2 metrics:

Term frequency

Inverse document frequency

Let's look at each individually.

## Term Frequency:

It's really what it says on the tin – how many times a term is repeated in a single document. Words that appear more frequently in a piece of text are considered to have a lot of importance. For example, in this sentiment analysis tutorial, we repeat the words "sentiment" and "analysis" multiple times, therefore, an ML model will consider them highly relevant.

## Inverse Document Frequency:

Rather than focusing on individual pieces, inverse document frequency measures how many times a word is repeated across a set of documents. And opposite of the previous metric, here the higher frequency is – the lower the relevance. This helps the algorithm eliminate naturally occurring words such as "a", "the", "and", etc, as they will appear frequently across all documents in a corpus.

Now that you understand how TF-IDF works, let's use this algorithm to vectorize our data:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)
X = tfidf.fit_transform(df['Review'])
```

We have successfully transformed the reviews in our dataset into a vector that can be fed into a machine learning algorithm!

## Step 5: Building and Evaluating the Machine Learning Model:

We can now train our algorithm on the review data to classify its sentiment into 3 categories:

- Positive
- Negative
- Neutral

First, let's perform a train-test split:

```
from sklearn.model_selection import train_test_split
y = df['Sentiment'] # target variable
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

Now, fit a logistic regression classifier on the training dataset and use it to make predictions on the test data:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear')
lr.fit(X_train,y_train) # fit the model
preds = lr.predict(X_test) # make predictions
```

## Finally, evaluate the performance:

```
from sklearn.metrics import accuracy_score

accuracy_score(preds,y_test) # 0.86
```

Our model has an accuracy of approximately 0.86, which is quite good.

And that concludes our tutorial! For a better understanding of the concept, here is the complete sentiment analysis Python code I've used:

```
import pandas as pd

import numpy as np

from sklearn.feature_extraction.text import re

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

df = pd.read_csv('tripadvisor_hotel_reviews.csv')

def create_sentiment(rating):


    res = 0 # neutral sentiment


    if rating==1 or rating==2:

        res = -1 # negative sentiment

    elif rating==4 or rating==5:

        res = 1 # positive sentiment
```

```
    return res
df['Sentiment'] = df['Rating'].apply(create_sentiment)
def clean_data(review):

    no_punc = re.sub(r'[^\w\s]', '', review)
    no_digits = ''.join([i for i in no_punc if not i.isdigit()])

    return(no_digits)
df['Review'] = df['Review'].apply(clean_data)
tfidf = TfidfVectorizer(strip_accents=None,
                        lowercase=False,
                        preprocessor=None)
X = tfidf.fit_transform(df['Review'])
y = df['Sentiment']
X_train, X_test, y_train, y_test = train_test_split(X,y)
lr = LogisticRegression(solver='liblinear')
lr.fit(X_train,y_train)
preds = lr.predict(X_test)
accuracy_score(preds,y_test)
```

MAIN CONTENT :

Sentiment analysis is a natural language processing (NLP) technique that involves determining the sentiment or emotional tone of a piece of text, such as a customer review or a social media post. It's a valuable tool for marketing because it allows businesses to gain insights into customer opinions, track brand sentiment, and make data-driven decisions. In this response, I'll outline how to perform sentiment analysis

for marketing using a machine learning algorithm in Python.

1. Data Collection:

Gather the text data you want to analyze, such as customer reviews, tweets, or comments. You can collect this data from various sources, including social media platforms, review websites, or your own customer feedback surveys.

2. Data Preprocessing:

Clean and preprocess the text data to prepare it for analysis. Common preprocessing steps include:

- Removing special characters, numbers, and punctuation.
- Tokenization: Splitting the text into individual words or tokens.
- Lowercasing all text to ensure uniformity.
- Removing stop words (common words like "and," "the," "is" that don't carry sentiment information).

3. Labeling Data:

Assign labels to your data to indicate the sentiment, such as positive, negative, or neutral. You can either label data manually or use pre-labeled datasets.

4. Feature Extraction:

Convert the text data into numerical vectors that can be used as input for machine learning models. Common techniques include:

- Bag of Words (BoW): A simple method that counts the frequency of each word in the text.
- TF-IDF (Term Frequency-Inverse Document Frequency): Assigns weights to words based on their importance in the document and across the corpus.

Word Embeddings: Use pre-trained word embeddings like Word2Vec or GloVe to convert words into dense vector representations.

5. Split Data:

Split your labeled data into training and testing sets to evaluate the performance of your model.

6. Model Selection:

Choose a machine learning model for sentiment analysis. Common choices include:

Naive Bayes

Support Vector Machine (SVM)

Recurrent Neural Networks (RNNs)

Long Short-Term Memory (LSTM)

Transformer-based models like BERT or GPT

7. Model Training:

Train the selected model using your training data. Fine-tune hyperparameters for optimal performance.

8. Model Evaluation:

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

9. Inference:

Use the trained model to predict sentiment for new, unlabeled text data.

10. Post-processing:

Analyze the results and visualize the sentiment distribution to gain insights for

marketing strategies. You can create visualizations, reports, or dashboards to present the findings.

Here's an example of how to perform sentiment analysis using Python and Scikit-Learn with a simple Bag of Words model:

```
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


# Load and preprocess your text data

# Label your data


# Feature extraction

vectorizer = CountVectorizer()

X = vectorizer.fit_transform(text_data)


# Split the 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)


# Train a Naive Bayes model

clf = MultinomialNB()

clf.fit(X_train, y_train)


# Make predictions
```

```
y_pred = clf.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

****THE END****