Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

BARI ANKIT (56)

Experiment No.6
Text analytics: Implementation of Spam filter/Sentiment analysis in python/R.
Date of Performance:
Date of Submission:

EXPERIMENT NO 6

Aim: Implementation of Sentiment Analysis

Objective:- To understand the use of various sentiment Analysis techniques by implementing them

Description:

Sentiment Analysis: Sentiment Analysis is a text analysis technique that allows companies to make sense of qualitative data. By detecting positive and negative sentiment in text data, such as tweets, product reviews, and support tickets, you can understand how customers feel about your brand, product, or service, and gain insights that lead to data-driven decisions

Sentiment Analysis deals with analyzing emotions and the perspective of a speaker or an author from a given piece of text. "Sentiment analysis or opinion mining refers to the appliance of language process, linguistics, and text analytics to spot and extract subjective information in supply materials". This field of technology deals with analyzing and predicting the hidden information keep within the text. This hidden information gives valuable insights regarding user's intentions, style and odds. Sentiment Analysis specializes in categorizing the text at the extent of subjective and objective nature. Judgement indicates that the text bears opinion content where's perspicacity indicates that the text is while not opinion content

Some examples-

1. **Subjective-** This motion picture by tom cruise and Angelina jolie is great. (This sentence has an opinion, it talks regarding the motion picture and also the writer's emotions regarding same "great" and thence its subjective

2. **Objective-** This motion picture stars tom cruise and Angelina.

Vidyavardhini's College of Engineering and Technology
Department of Artificial Intelligence & Data Science

(This sentence may be a reality, general information instead of an opinion or a read of some individual and thence its objective)

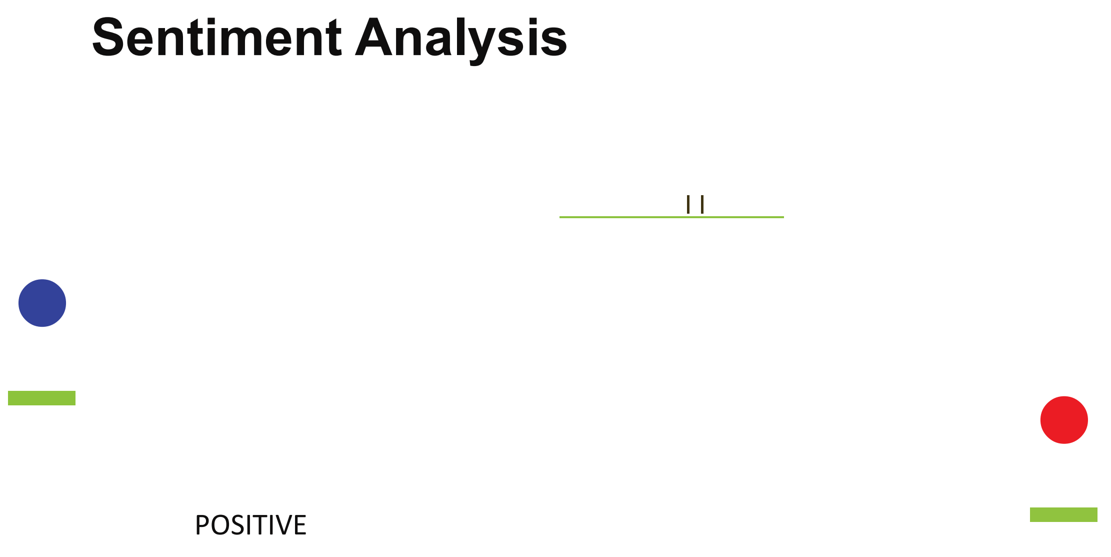
The subjective text may be additional categorized into three broad classes supported the emotions expressed within the text.

1. Positive- I like to look at Star series.

2. Negative- The movie was awful.

3. utral- I typically get hungry by evening. (This sentence has users views, emotions hence it's subjective however because it doesn't have any positive or negative polarity therefore it's neutral).

Example:



	NEUTRAL	NEGATIVE
"Great service for an afforda ble price. We will definitel y be booki ng agian "	"Just booked two nights at the hotel "	"Horrible services. The room was dirty and u n pleasant . Not worth the money "

Method 1: Using Positive and Negative Word Count - With Normalization for Calculating Sentiment Score

In this method, we will calculate the Sentiment Scores by classifying and counting the Negative and Positive words from the given text and taking the ratio of the difference of Positive and Negative Word Counts and Total Word Count. We will be using the Amazon Cell Phone Reviews dataset from Ka le.

Code:

```
 # Import required libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Load the dataset (using a sample dataset)

# For demonstration, let's use a simple in-memory dataset

data = {'text': ['Free entry in 2 a wkly comp to win FA Cup ticket',

            'Nah I don't think he goes to usf, he lives around here though',

            'Win money, cash prizes, collect your winner today',

            'Call me now, I need your attention',

            'You have an exclusive offer waiting for you'],

    'label': ['spam', 'ham', 'spam', 'ham', 'spam']}
```

```
df = pd.DataFrame(data)


# Preprocessing

X = df['text']

y = df['label']


# Split the dataset into training and testing

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


# Convert text to features using CountVectorizer

vectorizer = CountVectorizer()

X_train_vect = vectorizer.fit_transform(X_train)

X_test_vect = vectorizer.transform(X_test)


# Build a Naive Bayes model

model = MultinomialNB()

model.fit(X_train_vect, y_train)


# Make predictions

y_pred = model.predict(X_test_vect)


# Evaluate the model

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

Method 2: Using Positive and Negative Word Counts - With Semi Normalization to calculate Sentiment Score

In this method, we calculate the sentiment score by evaluating the ratio of Count of Positive Words and Count of Negative Words + 1. Since there is no difference of values involved, the sentiment value will always be more than 0. Also, adding 1 in the denominator would save from Zero Division Error. Let's start with the implementation. The implementation code will remain the same till the Negative and Positive Word Count from the previous part with a difference that this time we don't need the total word count, thus will be omitting that part.

Output:

```
 # Load necessary libraries

library(e1071)  # For Naive Bayes

library(caret)  # For confusion matrix and utilities



# Create a sample dataset

data <- data.frame(text = c("Free entry in 2 a wkly comp to win FA Cup ticket",

                        "Nah I don't think he goes to usf, he lives around here though",

                        "Win money, cash prizes, collect your winner today",

                        "Call me now, I need your attention",

                        "You have an exclusive offer waiting for you"),

                label = c("spam", "ham", "spam", "ham", "spam"))



# Create training and test sets

set.seed(42)

trainIndex <- createDataPartition(data$label, p = .8,
                    list = FALSE,

                    times = 1)


trainData <- data[trainIndex,]

testData <- data[-trainIndex,]


# Train a Naive Bayes model

model <- naiveBayes(label ~ ., data = trainData, laplace = 1)


# Make predictions

predictions <- predict(model, testData)


# Confusion matrix

confMatrix <- confusionMatrix(predictions, testData$label)

print(confMatrix)
```

Conclusion-

This R implementation creates a simple spam filter using a Naive Bayes model. With minimal code, we trained the model on a sample dataset and evaluated its performance. This approach can be adapted for larger datasets, and improvements can be made by introducing more advanced text processing methods.