Sentiment Analysis with Psychiatric Drug WebMD Reviews

Shruthi Narayanan
Self Learning Tutorial Assignment

Sentiment Analysis

Sentiment Analysis is a natural language processing (NLP) method that is used to determine sentiment or opinion in a sentence or text. It analyzes and categorizes a piece of text and determines whether that text has a positive, negative, or neutral connotation associated with it. Below are some typical steps used to complete a sentiment analysis.

1. Import and explore data
2. Preprocess the data
3. Assign labels to the data
4. Feature Extraction
5. Select and train the model
6. Evaluate the model
7. Testing the model and analysing results

Use Case in Healthcare

Sentiment analysis can bring immense benefits in the healthcare field. It allows hospitals/nursing homes, pharma companies, doctors/nurses, and even the common man or woman to gain insights and evaluate various entities related to healthcare.

Some use cases where sentiment analysis can be applied in healthcare include: patient feedback (from analyzing reviews via online surveys, website reviews, social media, etc.), healthcare facility reviews (used by hospitals to gather feedback to see where to improve, how to address patient concerns, etc.), and public health monitoring (to analyze perceptions of different diseases, vaccination campaigns, etc. across the world).

Psychiatric Drug Reviews Use Case

For this tutorial, I am going to be using sentiment analysis on psychiatric drug reviews.

Sentiment analysis can be used on drug reviews to benefit healthcare in many ways such as enabling healthcare providers to gauge satisfaction and better monitor patients who take certain drugs or helping determine harmful side effects (if a drug has all negative sentiment associated, that may be a sign to dig deeper).

For this analysis I used a dataset from Kaggle that had 60k+ records of psychiatric drug reviews that were pulled from WebMD. See the following slide for more details on how to access the data.

Data Source and Code

The dataset selected for this self-learning tutorial is from Kaggle and can be found here: https://www.kaggle.com/datasets/sepidehparhami/psychiatric-drug-webmd-reviews/data. Click on the black download button at the top right of the screen.

Data context from Kaggle website: "This dataset consists of unstructured text reviews, categorical ratings, and demographics from patients and caregivers of patients on various psychiatric drugs. The current version of the dataset contains over 62,000 reviews for hundreds of medications used to treat psychiatric disorders. The list of medications to include was compiled using the search by illness function for WebMD's drug database, accessible at https://www.webmd.com/drugs/2/conditions/index."

The workbook to access the Python code can be found here: https://colab.research.google.com/drive/1C0JuDpExvwo05on6Mdv_AGKSJgvy89Q8?usp=share_link

Please do not change the code in the notebook - make a copy of it onto your local drive and then alter it if needed.

Now that the data and code is provided, I will jump right in with the seven steps used in sentiment analysis (refer to Slide 2 to refresh).

Step 1: Import and Explore Data

```
#Step 1: Explore and Import Data
```

```
#import pandas package
import pandas as pd
import string
```

```
#import drive access
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Pulling the data files that I saved to my Google drive
column_data = '/content/drive/MyDrive/column_descriptions.csv'
drug_data = '/content/drive/MyDrive/drug_list.csv'
patients_data = '/content/drive/MyDrive/psychiatric_drug_webmd_reviews.csv'
```

```
#reading the data files in
column_df = pd.read_csv(column_data)
drug_df = pd.read_csv(drug_data)
patients_df = pd.read_csv(patients_data)
```

First step is to import pandas package and import the datafiles (which I previously uploaded into my google drive) and read them in.

Step 1: Import and Explore Data (cont.)

```
#Exploring what is contained in the Columns file
print("Columns:")
column_df
```

```
Columns:
```

	column_name	description
0	drug_name	Name of medication being reviewed
1	date	Date of review
2	age	Age group of patient, selected from a dropdown...
3	gender	Gender of patient, selected from a dropdown list
4	time_on_drug	How long the medication has been taken, select...
5	reviewer_type	The type of person reviewing, patient or careg...
6	condition	The "main reason for taking this medication," ...
7	rating_overall	Overall rating for the drug, the average of ra...
8	rating_effectiveness	Agreement with the statement "This drug has wo...
9	rating_ease_of_use	Agreement with the statement "This medication ...
10	rating_satisfaction	Agreement with the statement "Overall, I have ...
11	text	Raw, unstructured text response to "Write a br...

```
#Exploring what is contained in the Drugs file

print("Count:")
print(len(drug_df))
print(drug_df.head())
```

```
Count:
47
           drug_name                                                url
0       Abilify Oral  https://reviews.webmd.com/drugs/drugreview-644...
1       Doxepin Oral  https://reviews.webmd.com/drugs/drugreview-864...
2  Clomipramine Oral  https://reviews.webmd.com/drugs/drugreview-130...
3      Aplenzin Oral  https://reviews.webmd.com/drugs/drugreview-151...
4       Asendin Oral  https://reviews.webmd.com/drugs/drugreview-617...
```

Next I print out the dataframes to see what I'm actually working with on this analysis - there were three datasets in Kaggle link that are downloaded: columns, drugs, and drug reviews (next slide). I'm printing out what the datasets contain and the count of values in each.

Step 1: Import and Explore Data (cont.)

```
##Exploring what is contained in the drug review file

print("Count:")
print(len(patients_df))
patients_df.head()
```

```
Count:
62530
```

	Unnamed: 0	drug_name	date	age	gender	time_on_drug	reviewer_type	condition	rating_overall	rating_effectiveness	rating_ease_of_use	rating_satisfaction	text
0	0	Sertraline Oral	2/25/2024	35-44	Female	NaN	Patient	Depressed Mood Disorder Occurring Every Year a...	1.0	1	1	1	i have depression thats a lot worse in winter ...
1	1	Sertraline Oral	2/8/2024	45-54	Female	1 to 6 months	Patient	Anxiousness associated with Depression	4.7	5	4	5	started sertraline after having the worst year...
2	2	Sertraline Oral	2/7/2024	45-54	Male	1 to 6 months	Patient	Depression	1.7	1	3	1	had a nervous breakdown along with diagnosed w...
3	3	Sertraline Oral	1/31/2024	25-34	Female	1 to less than 2 years	Patient	Anxiousness associated with Depression	3.7	4	3	4	i have been on sertraline for a year my life h...
4	4	Sertraline Oral	1/15/2024	35-44	Male	1 to 6 months	Patient	Depression	1.7	1	3	1	the use of this drug has now destroyed my life...

Next I print out the dataframes to see what I'm actually working with on this analysis - there were three datasets in Kaggle link that are downloaded: columns and drugs (on previous slide), and drug reviews. I'm printing out what the datasets contain and the count of values in each.

This dataset contains the most information and the actual text reviews that people have provided on the drugs so I will use this dataset only for the analysis.

Step 2: Preprocessing

```
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Using a lemmatizer to help standardize text to prep it for my ML model
lemmatizer = WordNetLemmatizer()

# Here I create a function for preprocessing the text

def preprocess_text(text):
    # This is to check to make sure the text is not NaN

    if isinstance(text, str):

        # Converts text to lowercase
        text = text.lower()

        # Tokenizes the text into words
        tokens = word_tokenize(text)

        # Removes punctuation
        tokens = [token for token in tokens if token not in string.punctuation]

        # Removes stop words
        stop_words = set(stopwords.words('english'))
        tokens = [token for token in tokens if token not in stop_words]

        # Lemmatizes the tokens
        tokens = [lemmatizer.lemmatize(token) for token in tokens]

        # Joins the tokens back into a string
        preprocessed_text = ' '.join(tokens)
        return preprocessed_text
    else:

        # This returns an empty string for NaN values
        return ''
```

```
# Apply and check the preprocess function did process the text
patients_df['text'] = patients_df['text'].apply(preprocess_text)
patients_df['text'].head()
```

```
0    depression 's lot worse winter manageable summ...
1    started sertraline worst year life feeling lik...
2    nervous breakdown along diagnosed pre-hyperten...
3    sertraline year life changed better anxiety mu...
4                   use drug destroyed life living pssd
Name: text, dtype: object
```

Now that I have my dataset identified, I am ready for the preprocessing step. Preprocessing step is to make the data consistent and ready to use for the ML/DL models. Here I've written a method which takes a text and converts it to lowercase, tokenizes it (which means it splits the text into individual words or tokens), strips off the punctuation, applies a lemmatizer to it (which means it reduces a word to the base word or dictionary form), and joins the individual tokens back into a string. It finally checks to see if there is an NaN in which case it returns an empty string.I then apply that method to the dataframe text attribute and print it to ensure that the preprocessing worked.

Step 3: Label Assignment

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download the VADER lexicon  and initiatlize it
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

```
# Create a function to assign sentiment labels based on the VADER scores
def label_sentiment(text):


    # Analyze the sentiment of the text using VADER
    compound_score = sid.polarity_scores(text)['compound']
    if compound_score >= 0.05:
        return 'positive'
    elif compound_score <= -0.05:
        return 'negative'
    else:
        return 'neutral'
        #would be between -.5 to +.5

# Apply the function to the 'text' column to assign sentiment labels
patients_df['sentiment'] = patients_df['text'].apply(label_sentiment)

# View the first few rows of the DataFrame to verify the labeling
print(patients_df[text].head(), patients_df['sentiment'].head())
```

	Unnamed: 0	drug_name	date	age	gender	time_on_drug	reviewer_type	condition	rating_overall	rating_effectiveness	rating_ease_of_use	rating_satisfaction	text	sentiment
0	0	Sertraline Oral	2/25/2024	35-44	Female	NaN	Patient	Depressed Mood Disorder Occurring Every Year a...	1.0	1	1	1	depression 's lot worse winter manageable summ...	negative
1	1	Sertraline Oral	2/8/2024	45-54	Female	1 to 6 months	Patient	Anxiousness associated with Depression	4.7	5	4	5	started sertraline worst year life feeling lik...	negative
2	2	Sertraline Oral	2/7/2024	45-54	Male	1 to 6 months	Patient	Depression	1.7	1	3	1	nervous breakdown along diagnosed pre-hyperten...	negative
3	3	Sertraline Oral	1/31/2024	25-34	Female	1 to less than 2 years	Patient	Anxiousness associated with Depression	3.7	4	3	4	sertraline year life changed better anxiety mu...	positive
4	4	Sertraline Oral	1/15/2024	35-44	Male	1 to 6 months	Patient	Depression	1.7	1	3	1	use drug destroyed life living pssd	negative

Assigning labels to the text will help create the ‘sentiment’ that we are analyzing. There are several ways to approach labeling - some may do it manually by assigning a sentiment to a group of words (ex. If the word “effective” appears in a review, I can manually assign that to be a positive sentiment). However for this project, I went ahead and used the NLTK package and VADER lexicon that could do the labeling for me so I don’t have to manually do it. VADER stands for Valence Aware Dictionary and sEntiment Reasoner and it works by assigning polarity aka compound scores to words. The compound scores can range from -1 (most negative) to 1 (most positive).

Once I import / initialize NLTK and VADER, I create another function that will create a new column in the dataframe called ‘sentiment’ and based on the scores produced by VADER, assign a value ‘positive’, ‘negative’, or ‘neutral’ based on the review text.

Step 4: Feature Extraction

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Using a TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=10000)

# This is to fit and transform the text data to obtain TF-IDF features
X_tfidf = tfidf_vectorizer.fit_transform(patients_df['text'])

# Prints the shape of the matrix
print("Shape of TF-IDF matrix:", X_tfidf.shape)
```

```
Shape of TF-IDF matrix: (62530, 8637)
```

The next step, feature extraction, is used for text analysis. This is where data is transformed from raw text into numerical feature vectors that can then be used for ML algorithms. I used TF-IDF to do this - TF-IDF stands for Term Frequency-Inverse Document Frequency and it determines how important a select word is by comparing it to how frequent that word in a collection of documents and gives it a weightage. The matrix will show how each row corresponds to a document and each column corresponds to a unique term in the corpus - the values in that matrix represent the TF-IDF weightages for the terms in the respective files.

Step 5: Select and Train the Model

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Here I am splitting the dataset into featurs and labels
X = X_tfidf  #features
y = patients_df['sentiment']  #labels - that were created with the new sentiment column

# Split the dataset into training and testing using a 80% training, 20% testing split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Using the SVM classifier
model = SVC(kernel='linear')

# Train the model on the training data
model.fit(X_train, y_train)
```

```
SVC
SVC(kernel='linear')
```

Once feature extraction is complete, then I choose a model to use and train it. For my sentiment analysis, I chose to use the SVC model because I learned it can be used for sentiment analysis and dealing with text data, its good for binary classification (which just means it can categorize things into buckets - like bucketing a text into a positive, neutral, or negative sentiment), and is good with big amounts of data (and in this case, I had a dataset of 60K+ records).

To apply the model, I first have to split the data into labels and features. The labels are the sentiment buckets and the features are generated from the previous step using TF-IDF vectorization. I then split the dataset into training and test data and used a 80/20 split because that seems to be common practice - having 80% training data gives the model a good amount of data to train on.

Lastly, I applied the SVM classifier and trained the model. The next slide will show the results / accuracy.

Step 6: Evaluate Model

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Make predictions on the testing data
y_pred = model.predict(X_test)

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

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.995921957460419
Classification Report:
              precision    recall  f1-score   support

    negative       1.00      1.00      1.00      7233
     neutral       0.98      0.99      0.99       828
    positive       1.00      1.00      1.00      4445

    accuracy                           1.00     12506
   macro avg       0.99      1.00      0.99     12506
weighted avg       1.00      1.00      1.00     12506

Confusion Matrix:
[[7207   11   15]
 [   5  823    0]
 [  15    5 4425]]
```

Now that the model has used the training data to learn how sentiment is applied, its time to test it. I have the model predict what the sentiments would be for the test data and calculate the accuracy. For the accuracy calculation, it takes the actual sentiment and compares it to what the model said the sentiment would be and sees over the course of the model's predictions, how close to correct or accurate it is. I then print the classification report which shows various metrics including precision, recall, F-1 score, and support for each sentiment bucket.

Precision is calculated as True Positives / (True Positives + False Positives)
Recall is calculated as True Positives / (True Positives + False Positives)
F1 score is a balance between the two metrics above and support is the number of actual instances in the dataset

I also printed a confusion matrix which shows the model's performance by comparing the predicted labels with the true labels. Each number in the matrix represents the number of instances that were predicted to belong to a particular sentiment (rows) and actually belonged to that sentiment (columns).

Overall, looking at the metrics - it can be said that this model did a pretty stellar job seeing as the accuracy is 0.99 and precision/recall/f1 scores are 1.0 or near to it.

Step 7: Testing model

```python
# Generated some new text samples
new_reviews = [
    "This medication worked wonders for me! Highly recommended.",
    "I experienced some side effects with this drug, so I wouldn't recommend it.",
    "Neutral review: The medication was neither good nor bad.",
    "I'm not sure if this drug is effective. I might need to try something else.",
    "I'm completely satisfied with this medication. It has greatly improved my quality of life.",
    "I wouldn't recommend this drug to anyone. It caused severe side effects for me.",
    "The medication worked okay for me, but I had to stop taking it due to the side effects.",
    "I'm not sure if this drug is effective. I haven't noticed any changes since starting it.",
    "Neutral review: The medication had some positive effects, but also some negative side effects.",
    "This drug is a lifesaver! I don't know what I would do without it.",
    "I had high hopes for this medication, but unfortunately, it didn't work for me.",
    "Overall, I'm satisfied with the results of this medication. It's been helpful for my condition.",
    "I experienced some initial discomfort when starting this drug, but it eventually subsided.",
    "I'm very disappointed with this medication. It didn't work as expected and caused me a lot of discomfort.",
]


# Manually created labels to compare results
actual_labels = ['positive', 'negative', 'neutral', 'neutral', 'positive', 'negative', 'negative', 'neutral',
                 'neutral', 'positive', 'negative', 'positive', 'neutral', 'negative']

# Preprocess the new text samples
preprocessed_data = [preprocess_text(text) for text in new_reviews]
X_new = tfidf_vectorizer.transform(preprocessed_data)

# Make predictions using the trained model
predictions = model.predict(X_new)

# Compare predictions to the actual labels
for i in range(len(new_reviews)):
    print("Sample:", new_reviews[i])
    #print("Sample:", preprocessed_data[i])
    print("Predicted Label:", predictions[i])
    print("Actual Label:", actual_labels[i])
    print()
```

```
Sample: This medication worked wonders for me! Highly recommended.
Predicted Label: neutral
Actual Label: positive

Sample: I experienced some side effects with this drug, so I wouldn't recommend it.
Predicted Label: negative
Actual Label: negative

Sample: Neutral review: The medication was neither good nor bad.
Predicted Label: positive
Actual Label: neutral

Sample: I'm not sure if this drug is effective. I might need to try something else.
Predicted Label: positive
Actual Label: neutral

Sample: I'm completely satisfied with this medication. It has greatly improved my quality of life.
Predicted Label: positive
Actual Label: positive

Sample: I wouldn't recommend this drug to anyone. It caused severe side effects for me.
Predicted Label: negative
Actual Label: negative

Sample: The medication worked okay for me, but I had to stop taking it due to the side effects.
Predicted Label: negative
Actual Label: negative

Sample: I'm not sure if this drug is effective. I haven't noticed any changes since starting it.
Predicted Label: positive
Actual Label: neutral

Sample: Neutral review: The medication had some positive effects, but also some negative side effects.
Predicted Label: positive
Actual Label: neutral

Sample: This drug is a lifesaver! I don't know what I would do without it.
Predicted Label: positive
Actual Label: positive

Sample: I had high hopes for this medication, but unfortunately, it didn't work for me.
Predicted Label: positive
Actual Label: negative

Sample: Overall, I'm satisfied with the results of this medication. It's been helpful for my condition.
Predicted Label: positive
```

The last step I took for this project was to test new sets of review texts. I used a GenAI tool to produce some sample psychiatric drug reviews and applied the model to that new set of texts. I printed out the sample text, the predicted label, and the actual label (that I manually created by applying analyzing the sentiment of the sentence). The next slide provides metrics on how well the model faired on this new sample data.

Step 7: Testing model (cont.)

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Calculate accuracy
accuracy = accuracy_score(actual_labels, predictions)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(actual_labels, predictions))

# Calculate confusion matrix
conf_matrix = confusion_matrix(actual_labels, predictions)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.5
Classification Report:
              precision    recall  f1-score   support

    negative       1.00      0.60      0.75         5
     neutral       0.33      0.20      0.25         5
    positive       0.38      0.75      0.50         4

    accuracy                           0.50        14
   macro avg       0.57      0.52      0.50        14
weighted avg       0.58      0.50      0.50        14

Confusion Matrix:
[[3 1 1]
 [0 1 4]
 [0 1 3]]
```

I repeated the evaluation / analysis step to see how the model faired using the generated sample review text. As seen from the metrics, the model actually did not do as well as it did with the original test data. The accuracy score is 0.5 and the metrics especially for the neutral sentiment label are quite low. There could be different reasons for this. One - I only have a few lines of sample data - I wonder if I had a few thousand records if those metrics would go up and not be so heavily skewed. Another is that the sample data produced could be quite different in terms of content or how its phrased. Perhaps if I were to scrape WebMD for more recent reviews, perhaps the model would better predict the sentiment.