Nikhil Raj

Summary

The provided content is a comprehensive guide on implementing

NLP — Getting started with Sentiment Analysis

As the name suggests, it means to identify the view or emotion behind a situation. It basically means to analyze and find the emotion or intent behind a piece of text or speech or any mode of communication.

In this article, we will focus on the sentiment analysis of text data.

We, humans, communicate with each other in a variety of languages, and any language is just a mediator or a way in which we try to express ourselves. And, whatever we speak or write, has a sentiment associated with it. It might be positive or negative or it might be neutral as well.

Let's take an example —

Suppose, there is a fast-food chain company and they sell a variety of different food items like burgers, pizza, sandwiches, milkshakes, etc. They have created a website to sell their food items and now the customers can order any food item from their website. There is an option on the website, for the customers to provide feedback or reviews as well, like whether they liked the food or not.

User Review 1: I love this cheese sandwich, it's so delicious. User Review 2: This chicken burger has a very bad taste. User Review 3: I ordered this pizza today.

So, as we can see that out of these above 3 reviews,

- The second review is negative, and hence the company needs to look into their burger department.
- And, the third one doesn't signify whether that customer is happy or not, and hence we can consider this as a neutral statement.

By looking at the above reviews, the company can now conclude, that it needs to focus more on the production and promotion of their sandwiches as well as improve the quality of their burgers if they want to increase their overall sales.

But, now a problem arises, that, there will be hundreds and thousands of user reviews for their products and after a point of time, it will become nearly impossible to scan through each and every user review and come to a conclusion.

Neither can they just come up with a conclusion, by taking just 100 reviews or so, because maybe the first 100–200 customers were having similar taste and hence, they liked the sandwiches.

But over time when the no. of reviews increases, there might be a situation where the positive reviews are overtaken by more no. of negative reviews.

Therefore, this is where Sentiment Analysis and Machine Learning comes into play, which makes the whole process seamless. The ML model for

rather than assumptions made on a small sample of data.

Now, we know that we have 3 principal sentiments associated with every sentence. But, we can even break these principal sentiments(positive, negative and neutral) into smaller sub sentiments such as “Happy”, “Love”, ”Surprise”, “Sad”, “Fear”, “Angry” etc. as per the needs or business requirement.

Let’s take a real-world example –

- There was a time when the social media services like Facebook used to just have two emotions associated with each post, i.e. You can like a post or you can leave the post without any reaction and that signifies that you didn’t like it.
- But, over time these reactions to post have changed and grew into more granular sentiments which we see as of now, such as “like”, “love”, “sad”, “angry” etc.

And, because of this upgrade, when any company promotes their products on Facebook, they receive more specific reviews which in turn helps them to enhance the customer experience.

And because of that, they now have more granular control on how to handle their consumers, i.e. they can target the customers who are just “sad”, in a different way as compared to customers who are “angry”, and come up with a business plan accordingly, because nowadays, just doing the **bare minimum is not enough**.

Now, we will create a Sentiment Analysis Model, but it's easier said than done.

As we humans communicate with each other in a Natural Language, which is easy for us to interpret but it's much more complicated and messy if we really look into it.

Because, there are billions of people and they have their own style of communicating, i.e. a lot of tiny variations are added to the language and a lot of sentiments are attached to it which is easy for us to interpret but it becomes a challenge for the machines. For example, most of us use sarcasm in our sentences, which is just saying the opposite of what is really true.

This is why we need a process that makes the computers understand the Natural Language as we humans do, and this is what we call Natural Language Processing(NLP).

Sentiment Analysis is a sub-field of NLP and together with the help of machine learning techniques, it tries to identify and extract the insights from the data.

will predict the sentiment of a given statement.

First, let's import all the python libraries that we will use throughout the program.

Basic Python Libraries

- **Pandas —** For data analysis and data manipulation.
- **Matplotlib —** For data visualization.
- **Seaborn —** It's based on matplotlib and provides a high-level interface for data visualization.
- **WordCloud —** For visualizing text data in the form of clouds.
- **re —** It provides functions to pre-process the strings as per the given regular expression.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
```

Natural Language Processing

- **stopwords —** A collection of words that don't provide any meaning to a sentence.
- **WordNetLemmatizer —** It is used to convert different forms of words into a single item but still keeping the context intact.

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

Scikit-Learn (Machine Learning Library for Python)

- **CountVectorizer —** For transforming text into vectors.
- **GridSearchCV —** For hyperparameter tuning of ML models.
- **RandomForestClassifier —** Machine learning algorithm for classification.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

- **Accuracy Score —** It is no. of correctly classified instances/total no. of instances.
- **Precision Score —**It is the ratio of correctly predicted instances over total positive instances.
- **Recall Score —** It is the ratio of correctly predicted instances over total instances in that class.
- **Roc Curve —** A plot of true positive rate against false positive rate.
- **Classification Report —** Report of precision, recall and f1 score.
- **Confusion Matrix —** A table used to describe the classification models.

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,confu
from scikitplot.metrics import plot_confusion_matrix
```

We will use this dataset, which is available on Kaggle for sentiment analysis, which consists of sentences and their respective sentiment as a target variable. This dataset contains 3 separate files named train.txt, test.txt and val.txt.

the data frame with read_csv() and parameters as “delimiter” and “names” respectively.

```
df_train = pd.read_csv("train.txt", delimiter=';',names=['text','label'])
df_val = pd.read_csv("val.txt", delimiter=';',names=['text','label'])
```

As we will be using cross-validation and we have a separate test dataset as well, so we don’t need a separate validation set of data. So, we will concatenate these two Data Frames, and then we will reset the index to avoid duplicate indexes.

```
df = pd.concat([df_train,df_val])
df.reset_index(inplace=True, drop=True)
```

We can view a sample of the contents of the dataset using the “sample” method of pandas, and check the dimensions using the “shape” method.

```
print("Shape of the DataFrame:", df.shape)
df.sample(5)
```

Now, we will check for the various target labels in our dataset using seaborn.

```
sns.countplot(df.label)
```

multi-class classifier for Sentiment Analysis. But, for the sake of simplicity, we will merge these labels into two classes, i.e. Positive and Negative sentiment.

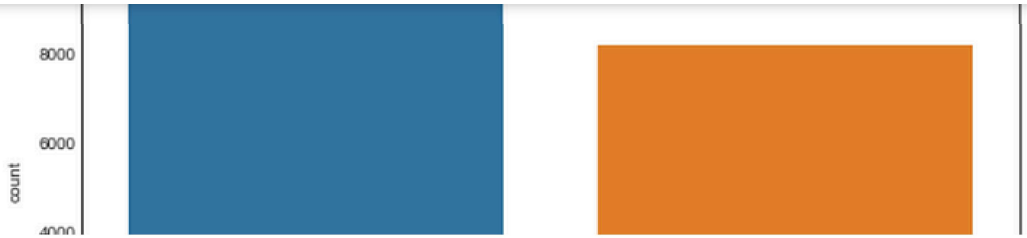
- *Positive Sentiment — "joy", "love", "surprise"*
- *Negative Sentiment — "anger", "sadness", "fear"*

Now, we will create a custom encoder to convert categorical target labels to numerical form, i.e. (0 and 1).

```
def custom_encoder(df):
    df.replace(to_replace ="surprise", value =1, inplace=True)
    df.replace(to_replace ="love", value =1, inplace=True)
    df.replace(to_replace ="joy", value =1, inplace=True)
    df.replace(to_replace ="fear", value =0, inplace=True)
    df.replace(to_replace ="anger", value =0, inplace=True)
    df.replace(to_replace ="sadness", value =0, inplace=True)
```

```
custom_encoder(df['label'])
```

```
sns.countplot(df.label)
```



Now, we can see that our target has changed to 0 and 1,i.e. 0 for Negative and 1 for Positive, and the data is more or less in a balanced state.

Data Pre-processing

Now, we will perform some pre-processing on the data before converting it into vectors and passing it to the machine learning model.

We will create a function for pre-processing of data.

- First, we will iterate through each record, and by using **regular expression**, we will get rid of any characters apart from alphabets.
- Then, we will convert the string to **lowercase** as the word **“Good”** is different from the word **“good”**.

will be created for the same word, which we don't want to.

3. Then we will check for stopwords in the data and get rid of them.

Terminology Alert — *Stopwords* *are commonly used words in a sentence such as "the", "an", "to" etc. which do not add much value.*

4. Then, we will perform **lemmatization** on each word, i.e. change the different forms of a word into a single item called a lemma.

Terminology Alert — *A **lemma** is a base form of a word. For example, "run", "running" and "runs" are all forms of the same lexeme, where the "run" is the lemma. Hence, we are converting all occurrences of the same lexeme to their respective lemma.*

5. And, then we will return a corpus of processed data.

So, first, we will create an object of WordNetLemmatizer and then we will perform the transformation.

```
#object of WordNetLemmatizer
lm = WordNetLemmatizer()
```

```
    corpus = []
    for item in df_col:
        new_item = re.sub('[^a-zA-Z]',' ',str(item))
        new_item = new_item.lower()
        new_item = new_item.split()
        new_item = [lm.lemmatize(word) for word in new_item if word not in se
        corpus.append(' '.join(str(x) for x in new_item))
    return corpus
```

```
corpus = text_transformation(df['text'])
```

Now, we will create a **Word Cloud**.

Terminology Alert — *WordCloud* *is a data visualization technique used to depict text in such a way that, the more frequent words appear enlarged as compared to less frequent words. This gives us a little insight into, how the data looks after being processed through all the steps until now.*

```
rcParams['figure.figsize'] = 20,8
word_cloud = ""
for row in corpus:
    for word in row:
        word_cloud+=" ".join(word)
```

Bag of Words

Now, we will use the Bag of Words Model(BOW), which is used to represent the text in the form of a bag of words, i.e. the grammar and the order of words in a sentence are not given any importance, instead, multiplicity, i.e. (the number of times a word occurs in a document) is the main point of concern.

Basically, it describes the total occurrence of words within a document.

using CountVectorizer.

So, we will convert the text data into vectors, by fitting and transforming the corpus that we have created.

```
cv = CountVectorizer(ngram_range=(1,2))
traindata = cv.fit_transform(corpus)
X = traindata
y = df.label
```

We have taken the **ngram_range** as (1,2) which signifies a bigram.

Terminology Alert — ***Ngram*** *is a sequence of 'n' of words in a row or sentence. 'ngram_range' is a parameter, which we use to give importance to the combination of words.*

*For example, the words "**social media**" together has a different meaning than the words "**social**" and "**media**" separately.*

We can experiment with the value of the **ngram_range** parameter and select the option which gives better results.

Model Creation

going to use Random Forest Classifier, and we will tune the hyperparameters using GridSearchCV.

GridSearchCV() is used to fit our estimators on the training data with all possible combinations of the predefined hyperparameters, which we will feed to it and provide us with the best model.

We will provide the following parameters to GridSearchCV,

- **Estimator or model —** RandomForestClassifier in our case.
- **parameters —** A dictionary of hyperparameter names and their values.
- **cv —** It signifies cross-validation folds.
- **return_train_score —** It returns the training scores of the various models.
- **n_jobs —** It signifies the no. of jobs to run parallelly ("-1" signifies that all CPU cores will be used which reduces the training time drastically)

First, We will create a dictionary, "parameters" which will contain the values of different hyperparameters.

We will pass this as a parameter to GridSearchCV to train our random forest classifier model using all possible combinations of these parameters to find the best model.

```
    'n_estimators': [500, 1000, 1500],
    'max_depth': [5, 10, None],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [1, 2, 5, 10],
    'bootstrap': [True, False]}
```

Now, we will fit the data into the grid search and view the best parameter using the "**best_params_**" attribute of GridSearchCV.

```
grid_search = GridSearchCV(RandomForestClassifier(),parameters,cv=5,return_tr
grid_search.fit(X,y)
grid_search.best_params_
```

```
{'bootstrap': True,
 'max_depth': None,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 500}
```

And then, we can view all the models and their respective parameters, mean test score and rank, as GridSearchCV stores all the intermediate results in the **cv_results_** attribute.

```
    print('Parameters: ',grid_search.cv_results_['params'][i])
    print('Mean Test Score: ',grid_search.cv_results_['mean_test_score'][i])
    print('Rank: ',grid_search.cv_results_['rank_test_score'][i])
```


A sample of the Output

Now, we will choose the best parameters obtained from GridSearchCV and create a final random forest classifier model and then train our new model.

```
rfc = RandomForestClassifier(max_features=grid_search.best_params_['max_featu
                                          max_depth=grid_search.best_params_['max
                                          n_estimators=grid_search.best_params_['
                                          min_samples_split=grid_search.best_para
                                          min_samples_leaf=grid_search.best_param
                                          bootstrap=grid_search.best_params_['boo
rfc.fit(X,y)
```

Test Data Transformation

on training data and finally evaluate the model on its predictions.

```
test_df = pd.read_csv('test.txt',delimiter=';',names=['text','label'])
```

```
X_test,y_test = test_df.text,test_df.label
#encode the labels into two classes , 0 and 1
test_df = custom_encoder(y_test)
#pre-processing of text
test_corpus = text_transformation(X_test)
#convert text data into vectors
testdata = cv.transform(test_corpus)
#predict the target
predictions = rfc.predict(testdata)
```

Model Evaluation

We will evaluate our model using various metrics such as Accuracy Score, Precision Score, Recall Score, Confusion Matrix and create a roc curve to visualize how our model performed.

```
rcParams['figure.figsize'] = 10,5
plot_confusion_matrix(y_test,predictions)
acc_score = accuracy_score(y_test,predictions)
pre_score = precision_score(y_test,predictions)
```

```
print('Recall_score: ',rec_score)
print("-"*50)
cr = classification_report(y_test,predictions)
print(cr)
```

Confusion Matrix:

Roc Curve:

We will find the probability of the class using the predict_proba() method of Random Forest Classifier and then we will plot the roc curve.

```
fpr,tpr,thresholds = roc_curve(y_test,predictions_probability[:,1])
plt.plot(fpr,tpr)
plt.plot([0,1])
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

As we can see that our model performed very well in classifying the sentiments, with an Accuracy score, Precision and Recall of approx. **96%**. And the roc curve and confusion matrix are great as well which means that our model can classify the labels accurately, with fewer chances of error.

Now, we will check for custom input as well and let our model identify the sentiment of the input statement.

Predict for Custom Input:

```
def expression_check(prediction_input):
    if prediction_input == 0:
        print("Input statement has Negative Sentiment.")
    elif prediction_input == 1:
        print("Input statement has Positive Sentiment.")
```

```
# function to take the input statement and perform the same transformations
def sentiment_predictor(input):
    input = text_transformation(input)
    transformed_input = cv.transform(input)
    prediction = rfc.predict(transformed_input)
    expression_check(prediction)
```

```
input1 = ["Sometimes I just want to punch someone in the face."]
input2 = ["I bought a new phone and it's so good."]
```

```
sentiment_predictor(input1)
sentiment_predictor(input2)
```

Hurray, As we can see that our model accurately classified the sentiments of the two sentences.

And, you can get the full code from here.

The END?

Data Science | Python | Machine Learning | NLP | Sentiment Analysis

---

Recommended from ReadMedium

Classification 2. Token Classification 3...

3 min read

Abdur Rahman

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

7 min read

Deepankar Singh

Building a Multi-Class Text Classifier with BERT: A Step-by-Step Guide with Code

Unlock the power of BERT for multi-class text classification! Dive into its architecture, fine-tuning, and practical code implementation.

9 min read

Austin Starks

I used OpenAI's o1 model to develop a trading strategy. It is DESTROYING the market

It literally took one try. I was shocked.

8 min read