Hello everyone, so let’s start right where we left off in Part — I.

In this post we will discuss how we can extract features from our textual dataset by using Bag-of-Words and TF-IDF. Then we will see how we can apply Machine Learning models using these features to predict whether a tweet falls into the Positive: ‘0’ or Negative: ‘1’ sentiment.
Note: In case you haven't read the Part — I of this series do give it a read to get a better understanding of Part — II.

Social Media Sentiment Analysis using Machine Learning: Part — I

Social media has opened a whole new world for people around the globe. People are just a click...

medium.com

So, let’s start shall we?

Text Processing

Extracting Features from cleaned Tweets

Bag-of-Words Features
Bag of Words is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set.

Consider a corpus (a collection of texts) called C of D documents \{d1,d2……dD\} and N unique tokens extracted out of the corpus C. The N tokens (words) will form a list, and the size of the bag-of-words matrix M will be given by D X N. Each row in the matrix M contains the frequency of tokens in document D(i).

For example, if you have 2 documents-

- **D1**: *He is a lazy boy. She is also lazy.*
- **D2**: *Smith is a lazy person.*

First, it creates a vocabulary using unique words from all the documents.

\[\text{['He', 'She', 'lazy', 'boy', 'Smith', 'person']}\]

As we can see in the above list we don’t consider “is”, “a”, “also” in this set because they don’t convey the necessary information required for the model.

- Here, \(D=2, N=6\)
- The matrix M of size 2 X 6 will be represented as:

```
<table>
<thead>
<tr>
<th></th>
<th>He</th>
<th>She</th>
<th>lazy</th>
<th>boy</th>
<th>Smith</th>
<th>person</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

The above table depicts the training features containing term frequencies of each word in each document. This is called bag-of-words approach since the number of occurrence and not sequence or order of words matters in this approach.

So, let’s apply this word embedding technique to our available dataset.

We have a package called **CountVectorizer** to perform this task.

```python
from sklearn.feature_extraction.text import CountVectorizer
bow_vectorizer = CountVectorizer(max_df=0.9, min_df=2, max_features=1000, stop_words='english')
```
```python
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')

# bag-of-words feature matrix
bow = bow_vectorizer.fit_transform(combine['Tidy_Tweets'])

df_bow = pd.DataFrame(bow.todense())

df_bow
```

**OUTPUT :-**

```
Out[42]:

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 990 | 991 | 992 | 993 | 994 | 995 | 996 | 997 | 998 | 999 |
|---|---|---|---|---|---|---|---|---|---|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 10| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 11| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 12| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 13| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 14| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 15| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 16| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 17| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 18| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 19| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| 20| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
```

```
TF-IDF Features

TF-IDF stands for **Term Frequency-Inverse Document Frequency**, and the TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

Typically, the TF-IDF weight is composed by two terms:

**Term Frequency (TF)**:
The first computes the normalized **Term Frequency** (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document.
Example :-

Consider a document containing 100 words wherein the word cat appears 3 times.

The *Term Frequency (TF)* for cat is then \( \frac{3}{100} = 0.03 \)

**Inverse Document Frequency (IDF) :**

The second term is the *Inverse Document Frequency (IDF)*, computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

\[
IDF(term) = \log \left( \frac{\text{Total number of documents}}{\text{Number of documents with } term \text{ in it}} \right)
\]

Example :-

Assume we have 10 million documents and the word cat appears in one thousand of these.

Then, the *Inverse Document Frequency (IDF)* is calculated as

\[
\log(10,000,000 / 1,000) = 4.
\]

**TF-IDF Example :**

Formula for finding the TF-IDF weight :-

\[
TFIDF(term) = TF(term) \times IDF(term)
\]

\[
w_{i,j} = tf_{i,j} \times \log \left( \frac{N}{df_i} \right)
\]

*tf_{i,j} = number of occurrences of i in j*

*df_i = number of documents containing i*

*N = total number of documents*
From the above examples the **Term Frequency** is 0.03 and **Inverse Document Frequency** is 4.

Thus, the **TF-IDF weight** is the product of these quantities: \(0.03 \times 4 = 0.12\).

**CODE :-**

Let us apply this technique to our dataset using Python.

We have a package available for this in **Scikit-Learn** known as **TfidfVectorizer**.

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

tfidf=TfidfVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')

tfidf_matrix=tfidf.fit_transform(combine['Tidy_Tweets'])
df_tfidf = pd.DataFrame(tfidf_matrix.todense())
df_tfidf
```

**OUTPUT :-**

```
|   | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | ... | 990  | 991  | 992  | 993  | 994  | 995  | 996  | 997  | 998  | 999 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|------|------|------|------|
|0  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|1  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|2  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|3  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|4  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|5  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|6  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|7  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.403826 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|8  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|9  | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|10 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|11 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|12 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|13 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|14 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|15 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|16 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|17 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|18 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
|19 | 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| 0.00| ... | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
```
These are the **Word Embedding** techniques which we have used on our dataset for feature extraction.

Let's move on to the next step.

**Splitting of our dataset into training and validation set.**

**Splitting our dataset into Training and Validation Set**

From the above two techniques that is **Bag-of-Words** and **TF-IDF** we have extracted features from the tweets present in our dataset.

Now, we have one dataset with features from the **Bag-of-Words** model and another dataset with features from **TF-IDF** model.

First task is to split the dataset into training and validation set so that we can train and test our model before applying it to predict for unseen and unlabeled test data.

**Using the features from Bag-of-Words for training set**
train_bow = bow[:31962]
train_bow.todense()

OUTPUT :-

Out[44]:
```
[[0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 ..., 
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Using features from TF-IDF for training set

train_tfidf_matrix = tfidf_matrix[:31962]
train_tfidf_matrix.todense()

OUTPUT :-

Out[45]:
```
[[0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 ..., 
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.]]
```

Splitting the data into training and validation set

from sklearn.model_selection import train_test_split

Bag-of-Words Features

x_train_bow, x_valid_bow, y_train_bow, y_valid_bow =
train_test_split(train_bow, train['label'], test_size=0.3, random_state =2)
TF-IDF features

```python
x_train_tfidf, x_valid_tfidf, y_train_tfidf, y_valid_tfidf = train_test_split(train_tfidf_matrix, train['label'], test_size=0.3, random_state=17)
```

We are done with splitting our dataset into train and validation set.

Want to know more about the Bag-of-Words and TF-IDF models used for feature extraction. Do give a read to the following blog post for an in depth discussion.

![Introduction to Natural Language Processing for Text](towardsdatascience.com)

Finally, we are here for the most awaited part of this series that is applying Machine Learning Models on our dataset.

**Applying Machine Learning Models**
The underlying problem we are going to solve comes under the **Supervised Machine Learning** category. So, let us have a brief discussion about this topic before moving on to apply different Machine Learning models on our dataset.

**Supervised Machine Learning :-**

The majority of practical machine learning uses **supervised learning**.

Supervised learning is where you have input variables \( (x) \) and an output variable \( (Y) \) and you use an algorithm to learn the mapping function from the input to the output.

\[ Y = f(X) \]

The goal is to approximate the mapping function so well that when you have new **input data** \( (x) \) that you can predict the **output variables** \( (Y) \) for that data.

It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

Supervised learning problems can be further grouped into regression and classification problems.

- **Classification**: A classification problem is when the output variable is a category, such as “**red**” or “**blue**” or “**disease**” and “**no disease**” or in our case “**Positive**” or “**Negative**”
Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Our problem comes under the classification category because we have to classify our results into either Positive or Negative class.

There is another category of Machine Learning algorithm called Unsupervised Machine Learning where you have an input data but no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. But that is of no concern for us for this problem statement.

Moving on :-
So from the above splitting of dataset we see that we will use features from the Bag-of-Words and TF-IDF for our Machine Learning Models.

We generally use different models to see which best fits our dataset and then we use that model for predicting results on the test data.

Here we will use 3 different models

- Logistic Regression
- XGBoost
- Decision Trees

and then we will compare their performance and choose the best possible model with the best possible feature extraction technique for predicting results on our test data.

Importing f1_score from sklearn

We will use F1 Score throughout to asses our model’s performance instead of accuracy. You will get to know why at the end of this article.

CODE :-

```python
from sklearn.metrics import f1_score
```
Now, let’s move on to applying different models on our dataset from the features extracted by using Bag-of-Words and TF-IDF.

**Logistic Regression**

*The first model we are going to use is Logistic Regression.*

```python
from sklearn.linear_model import LogisticRegression
Log_Reg = LogisticRegression(random_state=0, solver='lbfgs')
```

**Bag-of-Words Features**

*Fitting the Logistic Regression Model.*

```python
Log_Reg.fit(x_train_bow, y_train_bow)
```

**Predicting the probabilities.**

```python
prediction_bow = Log_Reg.predict_proba(x_valid_bow)
prediction_bow
```

**OUTPUT :-**

```
Out[52]: array([[9.86501156e-01, 1.34988440e-02],
                 [9.99590966e-01, 4.00041441e-04],
                 [9.13577383e-01, 8.64225167e-02],
                 ...
                 [8.95457155e-01, 1.04542845e-01],
                 [9.59736065e-01, 4.02639345e-02],
                 [9.67541420e-01, 3.24585797e-02]])
```

Predicting the probabilities for a tweet falling into either Positive or Negative class.

If you are confused about the above output, read this stack overflow answer and you will have a clear idea about it.
I've declared a x variable and filled it with the stackoverflow.com

The output basically provides us with the probabilities of the tweet falling into either of the classes that is Negative or Positive.

**Calculating the F1 score**

```python
# if prediction is greater than or equal to 0.3 than 1 else 0
# Where 0 is for positive sentiment tweets and 1 for negative sentiment tweets
prediction_int = prediction_bow[:,1]>=0.3

# converting the results to integer type
prediction_int = prediction_int.astype(np.int)

# calculating f1 score
log_bow = f1_score(y_valid_bow, prediction_int)
```

```
Out[54]: 0.5721352019785655
```

**TF-IDF Features**

**Fitting the Logistic Regression Model.**

```
Log_Reg.fit(x_train_tfidf,y_train_tfidf)
```

**Predicting the probabilities.**

```
prediction_tfidf = Log_Reg.predict_proba(x_valid_tfidf)
prediction_tfidf
```
OUTPUT:

Out[56]: array([[0.98487997, 0.01512003],
            [0.97949889, 0.02050111],
            [0.919737 , 0.080263 ],
            ...
            [0.98630906, 0.01369094],
            [0.96746188, 0.03253812],
            [0.99055287, 0.00944713]]

Predicting the probabilities for a tweet falling into either Positive or Negative class

**Calculating the F1 Score**

```python
# if prediction is greater than or equal to 0.3 than 1 else 0
# Where 0 is for positive sentiment tweets and 1 for negative sentiment tweets
prediction_int = prediction_tfidf[:,1]>=0.3

prediction_int = prediction_int.astype(np.int)
prediction_int

# calculating f1 score
log_tfidf = f1_score(y_valid_tfidf, prediction_int)

log_tfidf
```

Out[57]: 0.5862068965517241

F1 Score

Note: In the nested list the element[0][0] is for label: 0 or Positive tweets and element [0][1] is for label: 1 or Negative Tweets.

For an in depth analysis of Logistic Regression do read the following article.

**Logistic Regression for Machine Learning**

Logistic regression is another technique borrowed by machine learning from the field of statistics. I...

machinelearningmastery.com
The next model we use is XGBoost.

```
from xgboost import XGBClassifier
```

**Bag-of-Words Features**

```
model_bow = XGBClassifier(random_state=22, learning_rate=0.9)
```

**Fitting the XGBoost Model**

```
model_bow.fit(x_train_bow, y_train_bow)
```

**Predicting the probabilities.**

```
xgb = model_bow.predict_proba(x_valid_bow)
xgb
```

```
Out[61]: array([[0.9717447 , 0.02825526],
               [0.99767685, 0.00232312],
               [0.9436968 , 0.05630319],
               ...
               [0.9660848 , 0.03391522],
               [0.9436968 , 0.05630319],
               [0.9436968 , 0.05630319]], dtype=float32)
```

Predicting the probability of a tweet falling into either Positive or Negative class.

**Calculating the F1 Score**

```
# if prediction is greater than or equal to 0.3 than 1 else 0
# Where 0 is for positive sentiment tweets and 1 for negative sentiment tweets
xgb=xgb[:,1] >= 0.3

# converting the results to integer type
xgb_int=xgb.astype(np.int)
```
TF-IDF Features

model_tfidf = XGBClassifier(random_state=29, learning_rate=0.7)

Fitting the XGBoost model

model_tfidf.fit(x_train_tfidf, y_train_tfidf)

Predicting the probabilities.

xgb_tfidf = model_tfidf.predict_proba(x_valid_tfidf)

Predicting the probability of a tweet falling into either Positive or Negative class.
Calculating the F1 Score

```python
# if prediction is greater than or equal to 0.3 than 1 else 0
# Where 0 is for positive sentiment tweets and 1 for negative sentiment tweets
xgb_tfidf=xgb_tfidf[:, 1] >= 0.3

# converting the results to integer type
xgb_int_tfidf = xgb_tfidf.astype(np.int)

# calculating f1 score
score = f1_score(y_valid_tfidf, xgb_int_tfidf)
```

OUTPUT :-

```
Out[66]: 0.5657051282051281
```

F1 Score

For a more in depth analysis of the XGBoost model read the following article.

A Gentle Introduction to XGBoost for Applied Machine Learning

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggl...

machinelearningmastery.com

Decision Trees

The last model we use is Decision Trees.

```python
from sklearn.tree import DecisionTreeClassifier
dct = DecisionTreeClassifier(criterion='entropy', random_state=1)
```

Bag-of-Words Features
Fitting the Decision Tree model.

dct.fit(x_train_bow, y_train_bow)

Predicting the probabilities.

dct_bow = dct.predict_proba(x_valid_bow)

dct_bow

```
Out[70]: array([[1., 0.],
               [1., 0.],
               [1., 0.],
               ...
               [1., 0.],
               [1., 0.],
               [1., 0.]])
```

Predicting the probability of a tweet falling into either Positive or Negative class

Calculating the F1 Score

```
# if prediction is greater than or equal to 0.3 than 1 else 0
# Where 0 is for positive sentiment tweets and 1 for negative sentiment tweets
dct_bow = dct_bow[:, 1] >= 0.3

dct_int_bow = dct_bow.astype(np.int)

# converting the results to integer type

# calculating f1 score

dct_score_bow = f1_score(y_valid_bow, dct_int_bow)

dct_score_bow
```

```
Out[71]: 0.5141776937618148
```

F1 Score
**TF-IDF Features**

*Fitting the Decision Tree model*

```python
dct.fit(x_train_tfidf, y_train_tfidf)
```

*Predicting the probabilities.*

```python
dct_tfidf = dct.predict_proba(x_valid_tfidf)
dct_tfidf
```

Predicting the probability of a tweet falling into either Positive or Negative class

*Calculating the F1 Score*

```python
# if prediction is greater than or equal to 0.3 than 1 else 0
# Where 0 is for positive sentiment tweets and 1 for negative sentiment tweets
dct_tfidf = dct_tfidf[:, 1] >= 0.3

# converting the results to integer type
dct_int_tfidf = dct_tfidf.astype(np.int)

# calculating f1 score
dct_score_tfidf = f1_score(y_valid_tfidf, dct_int_tfidf)

dct_score_tfidf
```

Out[74]: 0.54988216810668342
For a more in depth analysis of Decision Trees model do give a read to the following article.

Intuitive Guide to Understanding Decision Trees
Understanding the underlying mechanics and parameters of decision trees
towardsdatascience.com

Model Comparison
Now, let us compare the different models we have applied on our dataset with different word embedding techniques.

Bag-of-Words

```
Algo_1 = ['LogisticRegression(Bag-of-Words)','XGBoost(Bag-of-Words)','DecisionTree(Bag-of-Words)'

score_1 = [log_bow,xgb_bow,dct_score_bow]

compare_1 = pd.DataFrame({'Model':Algo_1,'F1_Score':score_1},index=[i for i in range(1,4)])

compare_1.T
```

Out[76]:

<table>
<thead>
<tr>
<th>Model</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogisticRegression(Bag-of-Words)</td>
<td>0.572135</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XGBoost(Bag-of-Words)</td>
<td>0.571201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DecisionTree(Bag-of-Words)</td>
<td>0.514178</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F1 Score of different models using features from Bag-of-Words

Comparison Graph

```
plt.figure(figsize=(18,5))
sns.pointplot(x='Model',y='F1_Score',data=compare_1)
```
Algo_2 = ['LogisticRegression(TF-IDF)','XGBoost(TF-IDF)','DecisionTree(TF-IDF)']
score_2 = [log_tfidf,score,dct_score_tfidf]
compare_2 = pd.DataFrame({'Model':Algo_2,'F1_Score':score_2},index=[i for i in range(1,4)])

Out[79]:

<table>
<thead>
<tr>
<th>Model</th>
<th>LogisticRegression(TF-IDF)</th>
<th>XGBoost(TF-IDF)</th>
<th>DecisionTree(TF-IDF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1_Score</td>
<td>0.586207</td>
<td>0.565705</td>
<td>0.549882</td>
</tr>
</tbody>
</table>

Comparison Graph
As we can see the best possible model from both Bag-of-Words and TF-IDF is Logistic Regression.

Now, let us compare the score of the Logistic Regression model with both the feature extraction techniques that is Bag-of-Words and TF-IDF.

```
Algo_best = ['LogisticRegression(Bag-of-Words)','LogisticRegression(TF-IDF)']

score_best = [log_bow,log_tfidf]

compare_best = pd.DataFrame({'Model':Algo_best,'F1_Score':score_best},index=[i for i in range(1, len(compare_best)+1)])

compare_best.T
```

<table>
<thead>
<tr>
<th>Model</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogisticRegression(Bag-of-Words)</td>
<td>0.572135</td>
<td>0.586207</td>
</tr>
<tr>
<td>LogisticRegression(TF-IDF)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparison Graph
Predicting the results for our test data

From the above comparison graph we can clearly see that the best possible F1 Score is obtained by the Logistic Regression Model using TF-IDF features.

Code :-

```python
test_tfidf = tfidf_matrix[31962:]
test_pred = Log_Reg.predict_proba(test_tfidf)

test_pred_int = test_pred[:,1] >= 0.3
test_pred_int = test_pred_int.astype(np.int)

test['label'] = test_pred_int

submission = test[['id','label']]
submission.to_csv('result.csv', index=False)
```

Results after Prediction
res = pd.read_csv('result.csv')
res

Out[82]:

<table>
<thead>
<tr>
<th>id</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>31963</td>
</tr>
<tr>
<td>1</td>
<td>31064</td>
</tr>
<tr>
<td>2</td>
<td>31965</td>
</tr>
<tr>
<td>3</td>
<td>31966</td>
</tr>
<tr>
<td>4</td>
<td>31967</td>
</tr>
<tr>
<td>5</td>
<td>31968</td>
</tr>
<tr>
<td>6</td>
<td>31969</td>
</tr>
<tr>
<td>7</td>
<td>31970</td>
</tr>
<tr>
<td>8</td>
<td>31971</td>
</tr>
<tr>
<td>9</td>
<td>31972</td>
</tr>
<tr>
<td>10</td>
<td>31973</td>
</tr>
<tr>
<td>11</td>
<td>31974</td>
</tr>
<tr>
<td>12</td>
<td>31975</td>
</tr>
<tr>
<td>13</td>
<td>31976</td>
</tr>
<tr>
<td>14</td>
<td>31977</td>
</tr>
<tr>
<td>15</td>
<td>31978</td>
</tr>
<tr>
<td>16</td>
<td>31979</td>
</tr>
<tr>
<td>17</td>
<td>31980</td>
</tr>
<tr>
<td>18</td>
<td>31981</td>
</tr>
<tr>
<td>17174</td>
<td>49137</td>
</tr>
<tr>
<td>17175</td>
<td>49138</td>
</tr>
<tr>
<td>17176</td>
<td>49139</td>
</tr>
<tr>
<td>17177</td>
<td>49140</td>
</tr>
<tr>
<td>17178</td>
<td>49141</td>
</tr>
<tr>
<td>17179</td>
<td>49142</td>
</tr>
<tr>
<td>17180</td>
<td>49143</td>
</tr>
<tr>
<td>17181</td>
<td>49144</td>
</tr>
<tr>
<td>17182</td>
<td>49145</td>
</tr>
<tr>
<td>17183</td>
<td>49146</td>
</tr>
<tr>
<td>17184</td>
<td>49147</td>
</tr>
<tr>
<td>17185</td>
<td>49148</td>
</tr>
<tr>
<td>17186</td>
<td>49149</td>
</tr>
</tbody>
</table>
From the above output we can see that our Logistic Regression model with TF-IDF features predicts whether a tweet falls into the category of Positive — label : 0 or Negative — label : 1 sentiment.

**Summary :-**

5. Feature extraction techniques used
   - Bag-of-Words
   - TF-IDF

6. Machine Learning Models used
   - Logistic Regression
   - XGBoost
   - Decision Trees

7. Evaluation Metrics used

So, finally we have reached the end of our journey. We completed the tasks which were required to predict the sentiment of a particular tweet using Machine Learning.

**Tying up some Loose ends.**
The questions that arise are “What is F1 Score?” and “Why F1 Score instead of accuracy?”. 

So, before we proceed you need to have a basic idea about the terminologies like Confusion Matrix and its contents for example.

So, refer to this article for a basic understanding of the terminologies associated with Confusion Matrix.

**Understanding Confusion Matrix**

When we get the data, after data cleaning, pre-processing and wrangling, the first step we do is ...

towardsdatascience.com

OK, let us answer the above queries.

**Why F1 Score instead of Accuracy?**

Let us generate a countplot for our training dataset labels i.e. ‘0’ or ‘1’.

```
sns.countplot(train_original['label'])
sns.despine()
```
From the above countplot generated above we see how imbalanced our dataset is. We can see that the values with Positive — label : 0 sentiments are quite high in number as compared to the values with Negative — label : 1 sentiments.

So when we keep Accuracy as our evaluation metric there may be cases where we may encounter high number of false positives. So that is why we use F1 Score as our evaluation metric instead of Accuracy.

What is F1 Score?

To know about F1 Score we first have to know about Precision and Recall.

- Precision means the percentage of your results which are relevant.

- Recall refers to the percentage of total relevant results correctly classified by your algorithm.

We always face a trade-off situation between Precision and Recall i.e. High Precision gives low Recall and vice versa.
In most problems, you could either give a higher priority to maximising precision, or recall, depending upon the problem you are trying to solve. But in general, there is a simpler metric which takes into account both precision and recall, and therefore, you can aim to maximise this number to make your model better. This metric is known as **F1-score**, which is simply the harmonic mean of **Precision** and **Recall**.

So this metric seems much more easier and convenient to work with, as you only have to maximise one score, rather than balancing two separate scores.

```
  . . .
```

Finally, we have reached the end of the two part series of this article. I hope after this post you get a basic understanding of how you can start off with **text processing** and apply **Machine Learning** models to text data and extract information out of it.

After this we can try to deploy our Machine Learning Models over a website using the available web frameworks such as **Flask**, **FastAPI** etc. to production. But that’s a story for another blog post.

You can reach me at:

**LinkedIn**: [https://www.linkedin.com/in/deepak-das-profile/](https://www.linkedin.com/in/deepak-das-profile/)

**GitHub**: [https://github.com/dD2405](https://github.com/dD2405)

**Happy Reading !!!**

)