Sentiment Analysis of IMDB Movie Reviews

Submitted By :

Beeram Sai Sarvagna (AM.EN.U4CSE20316)

Krishnapriya Dinesan (AM.EN.U4CSE20339)

Nayan Thara M (AM.EN.U4CSE20347)

T S N Manikanta (AM.EN.U4CSE20370)

Yeganathan S (AM.EN.U4CSE20376)

Problem Definition

Sentiment analysis is the interpretation and classification of emotions within text data using text analysis techniques. Sentiment analysis allows businesses to identify customer sentiment toward products, brands or services in online conversations and feedback. Sentiment analysis models focus on polarity (positive, negative, neutral) feelings, emotions (angry, happy, sad, etc), and even intentions (e.g., interested vs. not interested).

Despite the inherent difficulty, it is still worth investigating whether it is possible to assign a Review to a movie starting from some objective and measurable features.But there isn't any method that can provide the prediction based on movie reviews so, to determine the success of the movie based on reviews, *Sentiment Analysis* plays an important role.

Datasets: Link to Datasets

Dataset 1:

1. What is the data about?

We have gathered data from Kaggle which includes a dataset that has 50000 reviews from IMDB which is equally divided into 25000 for training and testing. The data have already been divided into 25k reviews for training purposes, while the other 25k is intended for testing the classifier. In addition, both sets contain 12.5k positive and negative reviews. The reviews are classified into positive and negative in reference to the IMDB rating system.

There are only 30 reviews per movie as reviews for the same movie tend to have correlated ratings. Furthermore, the train and test sets contain a disjoint set of movies, so memorising a particular movie term and its associated labels would have no significance. A negative review is given a score of <=4 out of 10 while a positive review holds a score of >=7 and a neutral review has a score between 4 and 7.

2. What are the number of features, describe each of the features and explain the importance.

The features present are review and sentiment. The review will have different reviews from the user of the movie and for the Sentiment part, we have the Positive or Negative value. In our experiment, we have made use of Naive Bayes, Logistic Regression, and Support Vector Machine.

We have to train our model on the above classifiers to predict the movie polarity as positive or negative. To improve the performance of our model, we have done some operations on the data that we have collected.

3. In what applications the dataset has been used previously, if any?

User-generated reviews are often considered a source of truth when we consider watching a movie or a TV show. However, beyond telling us the qualitative aspects of the item we want to consume, reviews may inevitably contain undesired revelatory information (i.e., 'spoilers') such as the surprising fate of a character in a movie, or the identity of a murderer in a crime-suspense movie etc.

For users who are interested in consuming the item but are unaware of the critical plot twists, spoilers may decrease the excitement regarding the pleasurable uncertainty and curiosity of media consumption. Therefore, a natural question is how to identify these spoilers in entertainment reviews, so that users can more effectively navigate review platforms. So, this is one of the applications the dataset has been used previously.

Dataset 2:

1. What is the data about?

Internet Movie Database users are invited to participate in the site's ever-growing wealth of information by rating movies on a rating scale. The labelled dataset consists of 25,000 IMDB movie reviews. No individual movie has more than 30 reviews. The 25,000 reviews are labelled set. And for the Sentiment part, we have the Positive or Negative.

2. What are the number of features, describe each of the features and explain the importance.

There are 3 features in this dataset. They are ID, Review, and Sentiment. The ID is to uniquely identify the movies in the dataset. And the Positive is given as 1 and the Negative is given as 0. The review feature will have different reviews from the user of the movie. *There are two classes in the movie review dataset:* 1: >= 7 rating and 0: < 5 rating, they are given accordingly.

3. In what applications the dataset has been used previously, if any?

User reviews are our single source of truth when we are searching for a movie or a TV show to watch. Therefore, a common quest is to identify these spoilers among good reviews. So, this is one of the applications the dataset has been used previously.

Dataset 3:

1. What is the data about?

We have gathered data from Kaggle which includes a dataset that has 50000 reviews from IMDB which is equally divided into 25k for training and testing. The data have already been divided into 25k reviews for training purposes, while the other 25k is intended for testing the classifier. Furthermore, the train and test sets contain a disjoint set of movies, so memorising a particular movie term and its associated labels would have no significance. A negative review is given a score of <=4 out of 10 while a positive review holds a score of >=7 and neutral review has a score between 4 and 7.

2. What are the number of features, describe each of the features and explain the importance.

The features present are review and sentiment. The review will have different reviews from the user of the movie and for the Sentiment part, we have the Positive or Negative value. In our experiment, we have made use of Naive Bayes, Logistic Regression, and Support Vector Machine.

3. In what applications the dataset has been used previously, if any?

User reviews are our single source of truth when we are searching for a movie or a TV show to watch. However, some of the reviews may contain undesired facts such as the spoiler alerts of the movie, which causes high disappointment to users. Therefore, a common quest is to identify these spoilers among good reviews. So, this is one of the applications the dataset has been used previously.

Prepare Data:

Data preprocessing is required for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves the following steps: Getting the dataset, Importing libraries, Importing datasets, Finding Missing Data, Encoding Categorical Data, Splitting dataset into training and test set, Feature scaling.

Summarization:

```
In [12]: df.describe()
```

Out[12]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

Inference: describes gives us the count,unique values and frequency in a dataset.

```
In [13]: df['sentiment'].value_counts()
Out[13]: positive    25000
         negative    25000
         Name: sentiment, dtype: int64
```

Inference: we are getting the Frequency of each word.

```
In [15]: print(df.isnull().sum())
         review       0
         sentiment    0
         dtype: int64
```

Inference: To check the number of null values present in each column.

Dimensions of the dataset:

```
In [7]: df.shape
Out[7]: (50000, 2)

In [6]: df.size
Out[6]: 100000
```

Inference: To check the number of columns and rows in a dataset.

Breakdown of the data by the class variable:

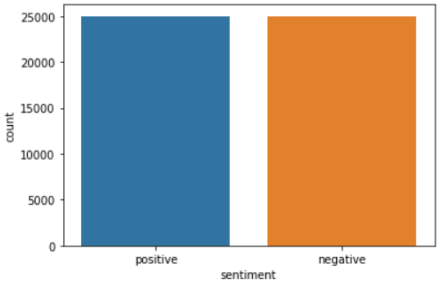
```
In [14]: df.info()
         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 50000 entries, 0 to 49999
         Data columns (total 2 columns):
          #   Column     Non-Null Count  Dtype
         ---  ------     --------------  -----
          0   review     50000 non-null  object
          1   sentiment  50000 non-null  object
         dtypes: object(2)
         memory usage: 1.1+ MB
```

Inference: info we are using to get a summary of the dataframe.

Data Visualization:

- Show the counts of observations in each categorical bin using bars

```
In [16]: sns.countplot(x='sentiment',data = df)
         plt.show()
```

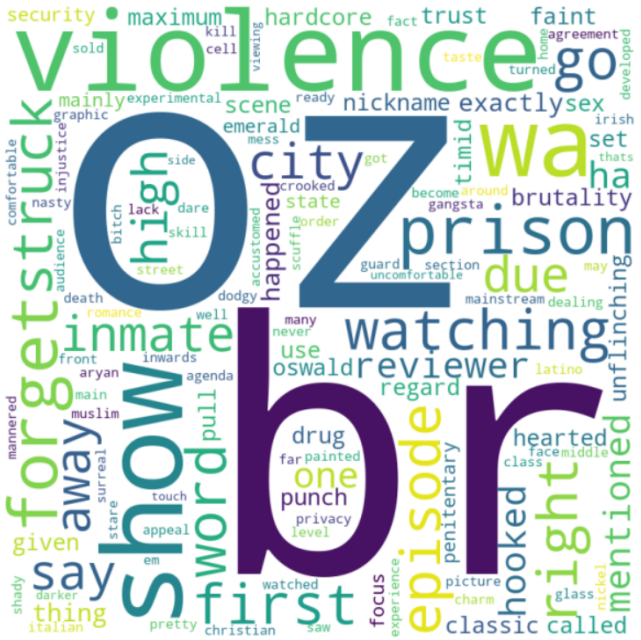


Inference: Here we have used countplot() to show the counts of observations in each categorical bin using bars.

- Show words that doesn't include stop words in tight layout of words

```
In [17]: w = [' '.join(text) for text in without_stopwords]
         wordcloud = WordCloud(width = 800, height = 800,
                               background_color ='white',
                               min_font_size = 10).generate(w[0])

         plt.figure(figsize = (8, 8), facecolor = None)
         plt.imshow(wordcloud)
         plt.axis("off")
         plt.tight_layout(pad = 0)
         plt.show()
```

Output:



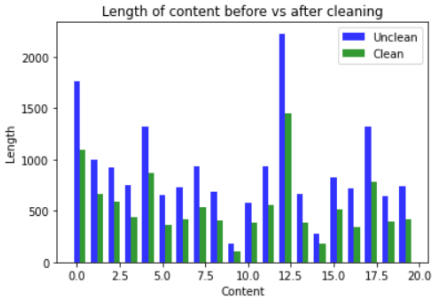
Inference: Word Cloud is a data visualisation technique used for representing text data in which the size of each word indicates its frequency or importance.

- Show clean and unclean data in the bar graph from dataset

In [18]:
```
unclean = [len(x) for x in df["review"]]
clean = [len(y) for y in w]

n_groups = 20
a = unclean[:20]
b = clean[:20]

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, a, bar_width,
                 alpha=opacity,
                 color='b',
                 label='Unclean')

rects2 = plt.bar(index + bar_width, b, bar_width,
                 alpha=opacity,
                 color='g',
                 label='Clean')

plt.xlabel('Content')
plt.ylabel('Length')
plt.title('Length of content before vs after cleaning')
plt.legend()
```

Output:

Out[18]:
```
<matplotlib.legend.Legend at 0x1da008ec2b0>
```

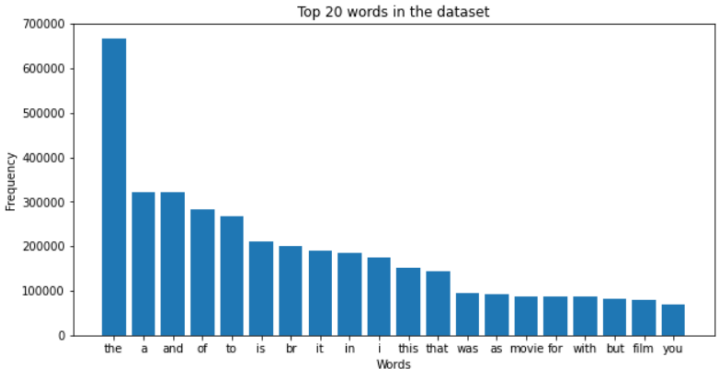


Inference: The following graph depicts the comparison between unclean and cleaned data length of content.

- Show top 20 words with the highest frequency in the plot

In [18]:
```
def top_words(cleaned_df):
    # create a dictionary to store the words and their frequency
    word_dict = {}
    for text in cleaned_df:
        for word in text.split():
            if word in word_dict:
                word_dict[word] += 1
            else:
                word_dict[word] = 1
    # sort the dictionary by values
    sorted_dict = sorted(word_dict.items(), key=lambda x: x[1], reverse=True)
    # get the top 20 words
    top_words = sorted_dict[:20]
    # get the top 20 words and their frequency
    top_words, top_freq = zip(*top_words)
    # plot the top 20 words and their frequency
    plt.figure(figsize=(10, 5))
    plt.bar(top_words, top_freq)
    plt.title('Top 20 words in the dataset')
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.show()

top_words(cleaned_df)
```

Output:



Inference: The above graph shows us the top 20 words according to their frequencies in decreasing order.

Python packages:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import spacy
import re,string,unicodedata
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from textblob import TextBlob
from textblob import Word
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

- ***Numpy***: NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.
- ***Pandas***: pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labelled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

- ***Seaborn***: Seaborn is a Python data visualisation library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- ***matplotlib.pyplot***: It is a collection of functions that make matplotlib work like MATLAB. Each py plot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
- ***Nltk***: NLTK, or Natural Language Toolkit, is a Python package that you can use for NLP. used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.
- ***Sklearn.feature_extraction***: The sklearn.feature_extraction module can be used to extract features in a format supported by machine learning algorithms from datasets consisting of formats such as text and image.
- ***CountVectorizer***: It is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.
- ***TfidfVectorizer***: It converts a collection of raw documents into a matrix of TF-IDF features.
- ***sklearn.preprocessing package***: It provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators. In general, learning algorithms benefit from standardisation of the data set.
- ***sklearn.preprocessing.Binarizer()***: It is a method which belongs to the preprocessing module. It plays a key role in the discretization of continuous feature values.
- ***.nltk.corpus import stopwords:*** By default, NLTK (Natural Language Toolkit) includes a list of 40 stop words, including: “a”, “an”, “the”, “of”, “in”, etc. The stopwords in nltk are the most common words in data. They are words that you do not want to use to describe the topic of your content. They are pre-defined and cannot be removed.

- ***nltk.stem.porter import PorterStemmer:*** NLTK Stemmers. Interfaces used to remove morphological affixes from words, leaving only the word stem. Stemming algorithms aim to remove those affixes required for eg. grammatical role, tense, derivational morphology leaving only the stem of the word.
- ***Wordcloud import WordCloud,STOPWORDS:*** From the word cloud documentation: stopwords : set of strings or None. The words that will be eliminated. If None, the build-in STOPWORDS list will be used.
- ***nltk.stem import WordNetLemmatizer:*** NLTK Lemmatization is the process of grouping the inflected forms of a word in order to analyse them as a single word in linguistics. NLTK has different lemmatization algorithms and functions for using different lemma determinations.
- ***nltk .tokenize import word_tokenize, sent_tokenize***: NLTK contains a module called tokenize() which further classified into two sub-categories: Word tokenize: We use the word_tokenize() method to split a sentence into tokens or words. Sentence tokenize: We use the sent_tokenize() method to split a document or paragraph into sentences.
- ***bs4 import BeautifulSoup:*** Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favourite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work
- ***Import spacy:*** spaCy is a free, open-source library for NLP in Python. It's written in Cython and is designed to build information extraction or natural language understanding systems. It's built for production use and provides a concise and user-friendly API.
- ***Import re,string,unicodedata:*** unicode module provides access to the Unicode Character Database (UCD) which defines character properties for all Unicode characters.
- ***nltk.tokenize.toktok.ToktokTokenizer:*** The tok-tok tokenizer is a simple, general tokenizer, where the input has one sentence per line; thus only the final period is tokenized.
- ***nltk.stem.WordNetLemmatizer:*** Lemmatize using WordNet's built-in morphy function. Returns the input word unchanged if it cannot be found in WordNet.
- ***nltk.stem.LancasterStemmer:*** A word stemmer based on the Lancaster (Paice/Husk) stemming algorithm.

- ***sklearn.linear_model.SGDClassifier:*** This estimator implements regularised linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning via the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.
- ***sklearn.linear_model.LogisticRegression:*** This class implements regularised logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. Regularisation is applied by default. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).
- ***sklearn.naive_bayes.MultinomialNB:*** The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.
- ***sklearn.svm.SVC:*** Used for C-Support Vector Classification.
- ***textblob.TextBolob:*** TextBlob aims to provide access to common text-processing operations through a familiar interface. You can treat TextBlob objects as if they were Python strings that learned how to do Natural Language Processing.
- ***sklearn.metrics.classification_report:*** Build a text report showing the main classification metrics.
- ***sklearn.metrics.confusion_matrix:*** Compute confusion matrix to evaluate the accuracy of a classification.
- ***sklearn.metrics.accuracy_score:*** In multilabel classification, this function computes subset accuracy.

Learning Algorithms:

Here we are Splitting the data into the Test data and training data by using the Term frequency-inverse document frequency (TFIDF).

```
X_tfidf = tfidf.fit_transform(X)

# split the dataset in train and test
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.3, random_state = 101)
```

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

```
from sklearn.ensemble import RandomForestClassifier


rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
rfc_predictions = rfc.predict(X_test)


print('The accuracy score is:',accuracy_score(y_test,rfc_predictions))
print('The confusion matrix is:','\n',confusion_matrix(y_test,rfc_predictions))
print('The classification report is:','\n',classification_report(y_test,rfc_predictions))
```

```
The accuracy score is: 0.8515
The confusion matrix is:
 [[5202  854]
 [ 928 5016]]
The classification report is:
              precision    recall  f1-score   support

           0       0.85      0.86      0.85      6056
           1       0.85      0.84      0.85      5944

    accuracy                           0.85     12000
   macro avg       0.85      0.85      0.85     12000
weighted avg       0.85      0.85      0.85     12000
```

We can see that the Accuracy is 85% when we use the Random Forest Algorithm.

Logistic Regression

Logistic Regression is a supervised machine learning algorithm which is mostly used for predicting the probability of a target variable whose nature will be dependent and also categorical. Also, the dependent variable is in binary. Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems.

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()

clf.fit(X_train, y_train)
```

```
LogisticRegression()
```

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

# Now apply those above metrics to evaluate your model
predictions = clf.predict(X_test)

print('The accuracy score is:',accuracy_score(y_test,predictions))
print('The confusion matrix is:','\n',confusion_matrix(y_test,predictions))
print('The classification report is:','\n',classification_report(y_test,predictions))
```

```
The accuracy score is: 0.8929166666666667
The confusion matrix is:
 [[5335  721]
 [ 564 5380]]
The classification report is:
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      6056
           1       0.88      0.91      0.89      5944

    accuracy                           0.89     12000
   macro avg       0.89      0.89      0.89     12000
weighted avg       0.89      0.89      0.89     12000
```

We can see that the Accuracy is 89% when we use the Logistic Regression Algorithm.

Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.The decisions or the test are performed on the basis of features of the given dataset.It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
dtc_predictions = dtc.predict(X_test)


print('The accuracy score is:',accuracy_score(y_test,dtc_predictions))
print('The confusion matrix is:','\n',confusion_matrix(y_test,dtc_predictions))
print('The classification report is:','\n',classification_report(y_test,dtc_predictions))
```

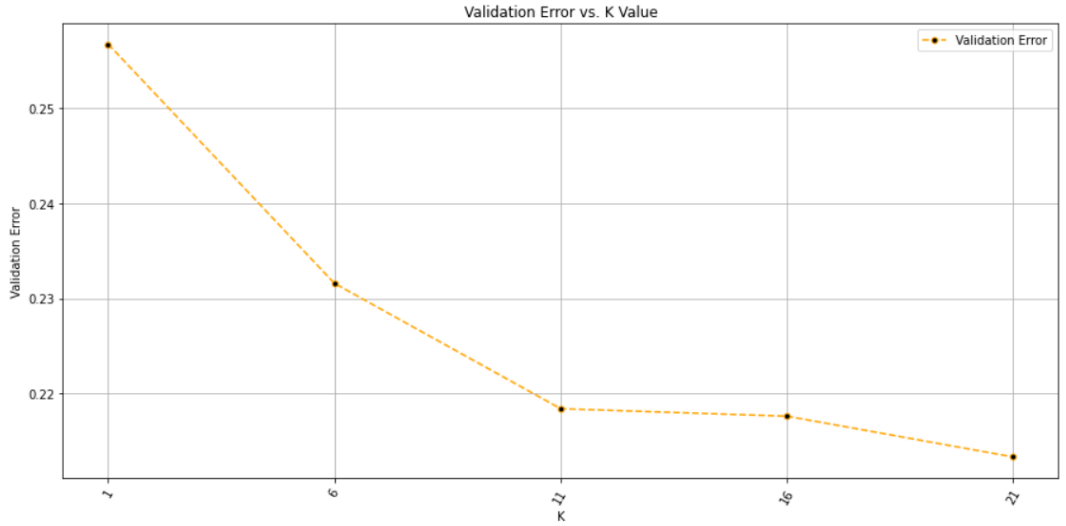
```
The accuracy score is: 0.7111666666666666
The confusion matrix is:
 [[4358 1698]
 [1768 4176]]
The classification report is:
              precision    recall  f1-score   support

           0       0.71      0.72      0.72      6056
           1       0.71      0.70      0.71      5944

    accuracy                           0.71     12000
   macro avg       0.71      0.71      0.71     12000
weighted avg       0.71      0.71      0.71     12000
```

We can see that the Accuracy is 71% when we use the Decision Tree Algorithm.

KNN

K – Nearest Neighbors is a supervised machine learning algorithm that can be used to solve both classification and regression problems. This algorithm can compete with most accurate models because it makes highly accurate predictions. So, we can believe that accuracy shown after using the KNN algorithm is very precise. We must always note that the quality of the predictions depends on the distance measured. The number of nearest neighbours to a new unknown variable that has to be predicted or classified is denoted by the symbol 'K'. KNN is a non – parametric algorithm. It is also called a lazy learner because it does not learn from the training part. Instead, it stores the dataset and during the classification, it performs its actions on the dataset.

```
from sklearn.neighbors import KNeighborsClassifier



### Tuning using K-fold Cross Validation

from sklearn.model_selection import cross_val_score

val_error_rate = []
neighbors_range = range(1,25,5)

for i in neighbors_range:

    knn = KNeighborsClassifier(n_neighbors=i)

    val_error = 1 - cross_val_score(knn, X_train, y_train,cv=5).mean()
    val_error_rate.append(val_error)

# Plot settings
plt.figure(figsize=(15,7))
plt.plot(neighbors_range, val_error_rate, color='orange', linestyle='dashed', marker='o',
         markerfacecolor='black', markersize=5, label='Validation Error')
plt.xticks(np.arange(neighbors_range.start, neighbors_range.stop, neighbors_range.step), rotation=60)
plt.grid()
plt.legend()
plt.title('Validation Error vs. K Value')
plt.xlabel('K')
plt.ylabel('Validation Error')
plt.show()
```



Here We are Plotting the Graph for the Different Values of the K vs the validation Error.

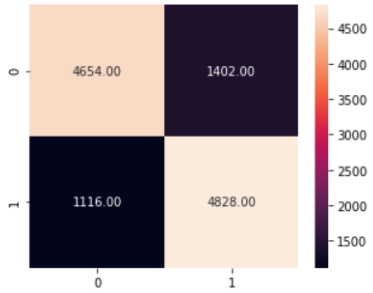
```
best_k = neighbors_range[val_error_rate.index(min(val_error_rate))]
best_k
```

```
21
```

Selecting the best value of the K from the Calculated Values.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
knn_predictions = knn.predict(X_test)

print('The accuracy score is:',accuracy_score(y_test,knn_predictions))
print('The classification report is:','\n',classification_report(y_test,knn_predictions))

matrix=confusion_matrix(y_test,knn_predictions)
plt.figure(figsize = (5,4))
sns.heatmap(matrix, annot=True, fmt = '.2f')
```

```
The accuracy score is: 0.7901666666666667
The classification report is:
              precision    recall  f1-score   support

           0       0.81      0.77      0.79      6056
           1       0.77      0.81      0.79      5944

    accuracy                           0.79     12000
   macro avg       0.79      0.79      0.79     12000
weighted avg       0.79      0.79      0.79     12000


<AxesSubplot:>
```



We can see that the Accuracy is 79% when we use the KNN Algorithm.

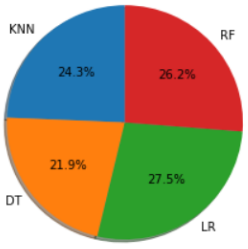
Comparisons:

```
print("In Percentages\n")
print("Accuracy of Logestic Regression: ",accuracy_lr*100)
print("Accuracy of Random Forest: ",accuracy_rf*100)
print("Accuracy of KNN: ",accuracy_knn*100)
print("Accuracy of Decision Trees: ",accuracy_dt*100)
```

```
In Percentages

Accuracy of Logestic Regression:  89.291
Accuracy of Random Forest:  85.15
Accuracy of KNN:  79.01599999999999
Accuracy of Decision Trees:  71.116
```

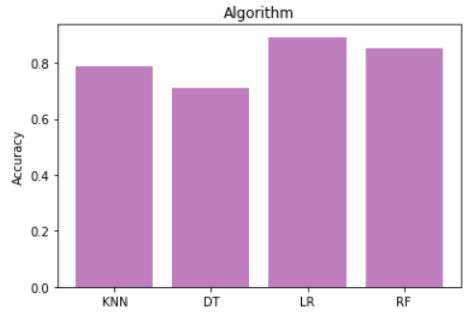
The Accuracy values of the Different Machine Learning Algorithms has been Shown.

```
labels =  'KNN','DT','LR','RF'
sizes = [accuracy_knn,accuracy_dt,accuracy_lr,accuracy_rf]
fig1, ax1 = plt.subplots()
ax1.pie(sizes,labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



By Using the Pie Chart we are Comparing the Different Machine Learning Algorithms for the 100 percentage of the pie chart. It shows that KNN Algorithm has a 24.3% , Random Forest has a 23.2% , Decision Trees has a 21.9% , Logistic Regression has a 27.5% for the overall 100 percent of the Pie Chart.

```
Algo = ('KNN','DT','LR','RF')
y_pos = np.arange(len(Algo))
performance = [accuracy_knn,accuracy_dt,accuracy_lr,accuracy_rf]

plt.bar(y_pos, performance, align='center', alpha=0.5,color='purple')
plt.xticks(y_pos, Algo)
plt.ylabel('Accuracy')
plt.title('Algorithm')

plt.show()
```



Comparing the Accuracies We can say that the Logistic Regression is having more accuracy compared to the other Algorithms we used.

— ---------- —