SENTIMENT ANALYSIS FOR MARKETING USING MACHINE LEARNING

TEAM MEMBERS

310821104003-ABINAYA TS
310821104034-HARINI M
310821104042-JOTHIKA K
310821104043-JULIYA A

Phase-4 DEVELOPMENT PART-2

Project:Sentiment Analysis for Marketing:

INTRODUCTION:

- In Our days, people use social media networks with a unbelievable frequency, writing posts, sharing photos and videos and sending private or public messages. One of th most used social network is Tweeter. Twitter is one of the most popular social media platforms in the world, with 330 million monthly active users and 500 million tweets sent each day. That's why analyzing tweets is very important to understand how people deal with a given subject.Understanding the sentiment of tweets is important for a variety of reasons: business marketing, politics, public behavior analysis, and information gathering are just a few examples. Sentiment analysis of Twitter data can help marketers understand the customer response to product launches and marketing campaigns, and it can also help political parties understand the public response to policy changes or announcements. Since Tweeter generate a huge amount of data (6000 tweets per second).

- Sentiment analysis refers to identifying as well as classifying the sentiments that are expressed in the text source. Tweets are often useful in generating a vast amount of sentiment data upon analysis. These data are useful in understanding the opinion of the people about a variety of topics.

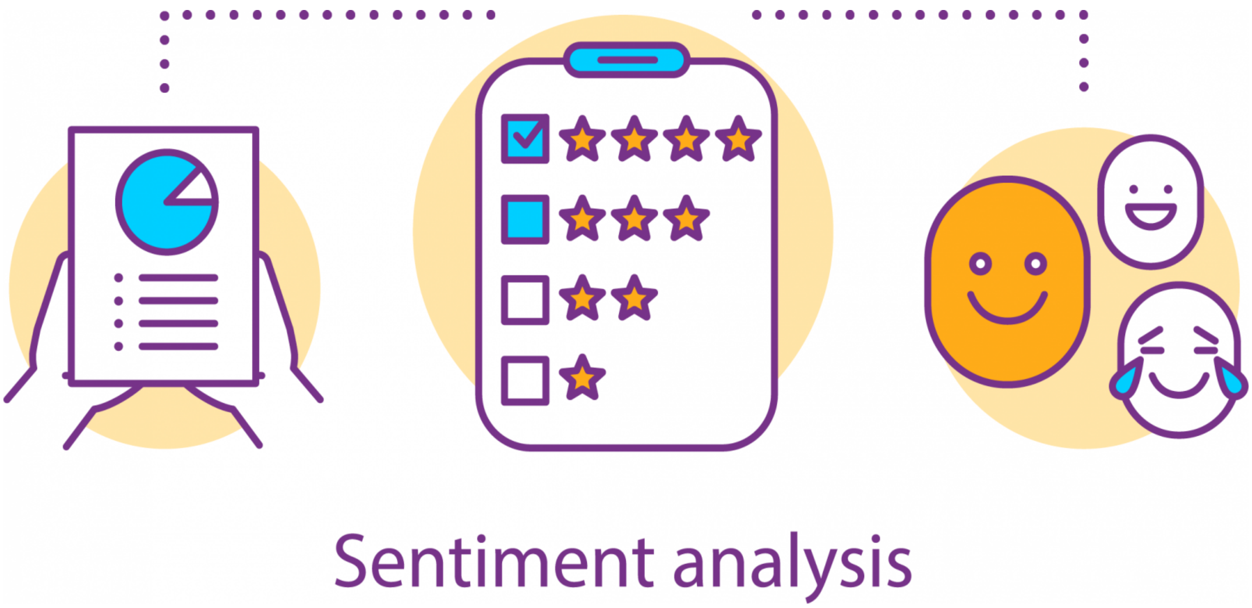
- Therefore we need to develop an Automated Machine Learning Sentiment Analysis Model in order to compute the customer perception. Due to the presence of non-useful characters (collectively termed as the noise) along with useful data, it becomes difficult to implement models on them.

🎯OBJECTIVE :

In this project, we are trying to implement a Twitter sentiment analysis model that helps to overcome the challenges of identifying the sentiments of the tweets.We aim to analyze the sentiment of the tweets provided from the Sentiment140 dataset by developing a machine learning pipeline involving the use of three classifiers:

- Logistic Regression.
- Bernoulli Naive Bayes.
- Decision Tree.
- K-nearest neighbors.
- Support Vector Machine.

Along with using Term Frequency- Inverse Document Frequency (TF-IDF).
The performance of these classifiers is then evaluated using accuracy, ROC-AUC Curve and F1 Scores.



1 Data Visualization after preprocessing:

Before performing the machine learning, **let's have a general idea of the accuracy of our data**, to do this we will use a **word cloud** which is a collection, or group, of words represented in different sizes. The bigger and bolder the word appears, the more often it is mentioned in a given text and the more important it is.

Which means that in our case we expect a **word cloud** to contain a sample of words representing the category we are plotting

we'll generate a **word cloud** for **positive tweets** and another for **negative tweets** to see which are the most commonly used words for each tweet category.

In[1]:

```
df.head(2)
```

Out[1]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized
0	0	awww thats bummer shoulda got david carr day d	[awww, thats, bummer, shoulda, got, david, car...	awww that bummer shoulda got david carr day d	awww that bummer shoulda got david carr day d
1	0	upset update facebook texting result school to...	[upset, update, facebook, texting, result, sch...	upset updat facebook text result school today ...	upset updat facebook text result school today ...

In[2]:

```
# Let's create a function which creates a wordcloud of a given pandas Series object :
def wordCloud(data_pos, max_words):
    # call the wordcloud function to show the most top 1000 used words:
    cloud = WordCloud(max_words=max_words, background_color="white", width=1600, height=800,
                      collocations=False).generate(" ".join(data_pos))
    plt.figure(figsize=(20, 20))
    plt.imshow(cloud)
```

Generating a `word cloud` for positive tweets :

In[3]:

wordCloud(df.loc[df["target"] == 1, "text"],2000)

Out[3]:

- As the picture shows, a lot of **positive words** appear: love, thank , haha, new, lol, great, nice, excited, happy, ready...

Generating a `word cloud` for negative tweets :

In[4]:

```
wordCloud(df.loc[df["target"] == 0, "text"], 2000)
```

Out[4]:

- As the picture shows, a lot of **negative words** appear: bad, sad, wish, need, sorry...

seeking to gather more information about our data

Now, let's compare the length of tweets from each sentiment category and see if there is a relationship between tweet sentiment and tweet length.

In[5]:

```
# Calculating tweet's lenght :


df["text_length"] = df["text"].apply(len)


# let's show the mean word count of each sentiment :
```

```
round(pd.DataFrame(df.groupby("target").text_length.mean()),2)
```

Out[5]:

	text_length
target	
0	41.25
1	40.92

We can see that positive and negative sentiment have the same average text length, which means the **sentiment** and tweet **length** are **independent** variables.

2Splitting our data into Train & Test Subset:

⚠️⚠️⚠️ For performance reasons, for **some models** we will use the **full dataset**, for other **computationally heavy models** we will use **10% of the original dataset.**⚠️⚠️⚠️.

Creating a new variable `df_reduced` that contains a shuffled sample of the dataset :

In[6]:

```
# Generating one row :

df_reduced = df.sample(frac =.10)

 # Displaying the reduced dataset :

df_reduced
```

Out[6]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized	text_length
839773	1	bathroom series ellen tooo funny day	[bathroom, series, ellen, tooo, funny, day]	bathroom seri ellen tooo funni day	bathroom seri ellen tooo funni day	36
643717	0	feeling tired anxious today upsets home younge...	[feeling, tired, anxious, today, upsets, home,...	feel tire anxiou today upset home youngest son...	feel tire anxiou today upset home youngest son...	72
256786	0	online soon girl ill diie dont want dying love x	[online, soon, girl, ill, diie, dont, want, dy...	onlin soon girl ill diie dont want die love x	onlin soon girl ill diie dont want die love x	48
208926	0	lost send game	[lost, send, game]	lost send game	lost send game	14
1216154	1	ecstasy key treating ptsd like ptsd ecstasy tr...	[ecstasy, key, treating, ptsd, like, ptsd, ecs...	ecstasi key treat ptsd like ptsd ecstasi treat...	ecstasi key treat ptsd like ptsd ecstasi treat...	73
...	...	...	...	...	...	...
583215	0	rainy day catching episodes ncis	[rainy, day, catching, episodes, ncis]	raini day catch episod nci	raini day catch episod nci	32
900488	1	eighty min sec new record	[eighty, min, sec, new, record]	eighti min sec new record	eighti min sec new record	25
369744	0	till sale spent money like	[till, sale, spent, money, like]	till sale spent money like	till sale spent money like	26
1248856	1	got ma pink bikini b like twins	[got, ma, pink, bikini, b, like, twins]	got ma pink bikini b like twin	got ma pink bikini b like twin	31
1589701	1	working fun today loads hot guys	[working, fun, today, loads, hot, guys]	work fun today load hot guy	work fun today load hot guy	32

146148 rows × 6 columns

In[7]:

```
print( "The shape of the original dataset: " + str(df.shape))
print( "The shape of the reduced dataset: " + str(df_reduced.shape))
```

Out[7]:

```
The shape of the original dataset: (1461480, 6)
The shape of the reduced dataset: (146148, 6)
```

🙌 You can see here **reduced dataset** equals 10% of the **original dataset**

In[8]:

```
# Separating input feature and label :
X = df["tokenized_tweets_stemmed_lemmatized"]
y = df["target"]

X_reduced = df_reduced["tokenized_tweets_stemmed_lemmatized"]
y_reduced = df_reduced["target"]
```

In[9]:

```
# Separating the 85% data for training data and 15% for testing data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.15, random_state=100)
X_train_reduced, X_test_reduced, y_train_reduced, y_test_reduced =
train_test_split(X_reduced, y_reduced,
test_size=0.15, random_state=100)
```

- `random_state` is basically used for reproducing your problem the same every time it is run. If we do not use a **random_state** in **train_test_split**, every time you make the split we might get a different set of train and test data points and will not help in debugging in case we get an issue.
- `X` contains `df["tokenized_tweets_stemmed_lemmatized"]`
- `y` contains = `df["target"]`
- `X_train` contains **85%** of `df["tokenized_tweets_stemmed_lemmatized"]`
- `X_test` contains **15%** of `df["tokenized_tweets_stemmed_lemmatized"]`
- `y_train` contains **85%** of `df["target"]`
- `y_test` contains **15%** of `df["target"]`

⚠️ The same goes for the reduced variables !

3️⃣ Word Embedding and Transforming Dataset using TF-IDF Vectorizer :

NLP experts developed a technique called **word embeddings** that convert words into their numerical representations. Once converted, **NLP algorithms can easily digest these learned representations to process textual information.** Word embeddings map the words as real-valued numerical vectors. It does so by tokenizing each word in a sequence (or sentence) and converting them into a vector space. Word embeddings aim to capture the semantic meaning of words in a sequence of text. It assigns similar numerical representations to words that have similar meanings.

Simply, these words **need to be made meaningful for machine learning or deep learning algorithms**. Therefore, **they must be expressed numerically**. Algorithms such as **One Hot Encoding**, **TF-IDF**, **Word2Vec**, **FastText** enable words to be expressed mathematically as word embedding techniques used to solve such problems.

`One-hot encoding` is an important step for preparing our dataset for use in machine learning. **One-hot encoding** `turns your categorical data into a binary vector representation`. Pandas get dummies makes this very easy!

- This means that for each unique value in a column, a new column is created. The values in this column are represented as 1s and 0s, depending on whether the value matches the column header.
- For example, with the help of the `get_dummies` function, we turn this table below :

Gender
Male
Female
Male
Male

  - To this :

Gender	Male	Female
Male	1	0
Female	0	1
Male	1	0
Male	1	0

- `Bag Of Words`:

The **bag-of-words** model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

Bag of Words (BOW) 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

In[10]:

```
# Quick overview of our dataset:
df.head()
```

Out[10]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized	text_length
0	0	awww thats bummer shoulda got david carr day d	[awww, thats, bummer, shoulda, got, david, car...	awww that bummer shoulda got david carr day d	awww that bummer shoulda got david carr day d	46
1	0	upset update facebook texting result school to...	[upset, update, facebook, texting, result, sch...	upset updat facebook text result school today ...	upset updat facebook text result school today ...	54
2	0	dived times ball managed save rest bounds	[dived, times, ball, managed, save, rest, bounds]	dive time ball manag save rest bound	dive time ball manag save rest bound	41
3	0	body feels itchy like	[body, feels, itchy, like]	bodi feel itchi like	bodi feel itchi like	21
4	0	behaving im mad	[behaving, im, mad]	behav im mad	behav im mad	15

In[11]:

```
# Quick overview of our reduced dataset:
df_reduced.head()
```

Out[11]:

	target	text	tokenized_tweets	tokenized_tweets_stemmed	tokenized_tweets_stemmed_lemmatized	text_length
839773	1	bathroom series ellen tooo funny day	[bathroom, series, ellen, tooo, funny, day]	bathroom seri ellen tooo funni day	bathroom seri ellen tooo funni day	36
643717	0	feeling tired anxious today upsets home younge...	[feeling, tired, anxious, today, upsets, home,...	feel tire anxiou today upset home youngest son...	feel tire anxiou today upset home youngest son...	72
256786	0	online soon girl ill diie dont want dying love x	[online, soon, girl, ill, diie, dont, want, dy...	onlin soon girl ill diie dont want die love x	onlin soon girl ill diie dont want die love x	48
208926	0	lost send game	[lost, send, game]	lost send game	lost send game	14
1216154	1	ecstasy key treating ptsd like ptsd ecstasy tr...	[ecstasy, key, treating, ptsd, like, ptsd, ecs...	ecstasi key treat ptsd like ptsd ecstasi treat...	ecstasi key treat ptsd like ptsd ecstasi treat...	73

In[12]:

```
# Fit the TF-IDF Vectorizer :
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=10000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

Out[12]:

```
No. of feature_words:  10000
```

In[13]:

```
# Fit the TF-IDF Vectorizer :
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=1000)
vectoriser.fit(X_train_reduced)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

Out[13]:

```
No. of feature_words:  1000
```

In[14]:

```
# Transform the data using TF-IDF Vectorizer :
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)

X_train_reduced = vectoriser.transform(X_train_reduced)
X_test_reduced  = vectoriser.transform(X_test_reduced)
```

4 Function for Model Evaluation :

After training the model we then apply the evaluation measures to check how the model is performing. Accordingly, we use the following evaluation parameters to check the performance of the models respectively :

- `Accuracy Score` : Typically, the accuracy of a predictive model is good (above 90% accuracy)
- `ROC-AUC Curve` : The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.
- `Confusion Matrix with Plot` : A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.
    - `Actual values` are the columns.
    - `Predicted values` are the lines.

	Positive	Negative
Positive	TP	TN
Negative	FP	TN

In[15]:

```
def model_Evaluate(model):

# Predict values for Test dataset

    y_pred = model.predict(X_test)

    # Print the evaluation metrics for the dataset.

    print(classification_report(y_test, y_pred))

    # Compute and plot the Confusion matrix

    cf_matrix = confusion_matrix(y_test, y_pred)

    categories = ['Negative','Positive']

    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten() / np.sum(cf_matrix)]

    labels = [f'{v1}n{v2}' for v1, v2 in
zip(group_names,group_percentages)]

    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',

    xticklabels = categories, yticklabels = categories)
```

```
    plt.xlabel("Predicted values", fontdict = {'size':14}, labelpad = 10)

    plt.ylabel("Actual values" , fontdict = {'size':14}, labelpad = 10)

    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)
```

- To avoid each time and for each model, drawing the confusion matrix, printing the precision, the f1-score... we just define the `model Evaluate()` function which will do the job each time.

5 Model Building:

In the problem statement we have used three different models respectively :

- **Model 1:** `Bernoulli Naive Bayes`.
- **Model 2:** `SVM (Support Vector Machine)`.
- **Model 3:** `Logistic Regression`.
- **Model 4:** `Decision Tree`.
- **Model 5:** `K-nearest neighbors`.

The idea behind choosing these models is that **we want to try all the classifiers on the dataset** ranging from simple models to complex models, and try to **find the one that performs the best**.

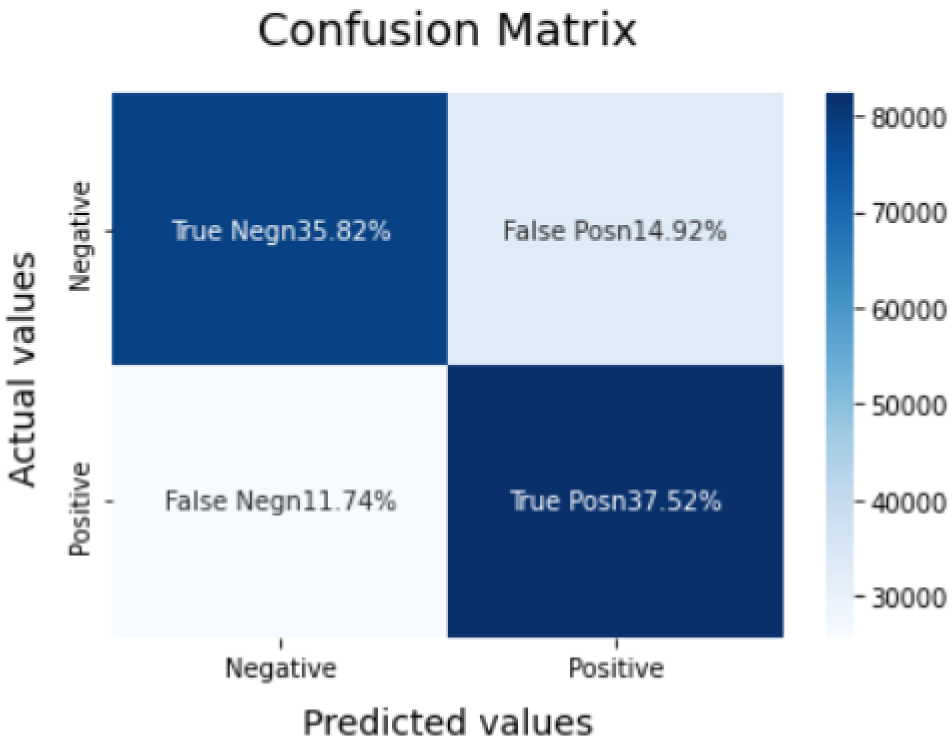
In[16]:

```
# Model-1 : Bernoulli Naive Bayes.

BNBmodel = BernoulliNB()

start1 = time.time()

BNBmodel.fit(X_train, y_train)

end1 = time.time()
```

```
print("\t\t⚠️⚠️⚠️ The training execution time of this model is {:.2f} seconds ⚠️⚠️⚠️\n".format(end1-start1))

start2 = time.time()

model_Evaluate(BNBmodel)

y_pred1 = BNBmodel.predict(X_test)

end2 = time.time()

print("\t\t⚠️⚠️⚠️ The test execution time of this model is {:.2f} seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

Out[16]:

```
⚠️⚠️⚠️ The training execution time of this model is 0.39 seconds ⚠️⚠️⚠️

              precision    recall  f1-score   support

           0       0.75      0.71      0.73    111228

           1       0.72      0.76      0.74    107994

    accuracy                           0.73    219222

   macro avg       0.73      0.73      0.73    219222

weighted avg       0.73      0.73      0.73    219222


           ⚠️⚠️⚠️ The test execution time of this model is 0.61 seconds ⚠️⚠️⚠️
```

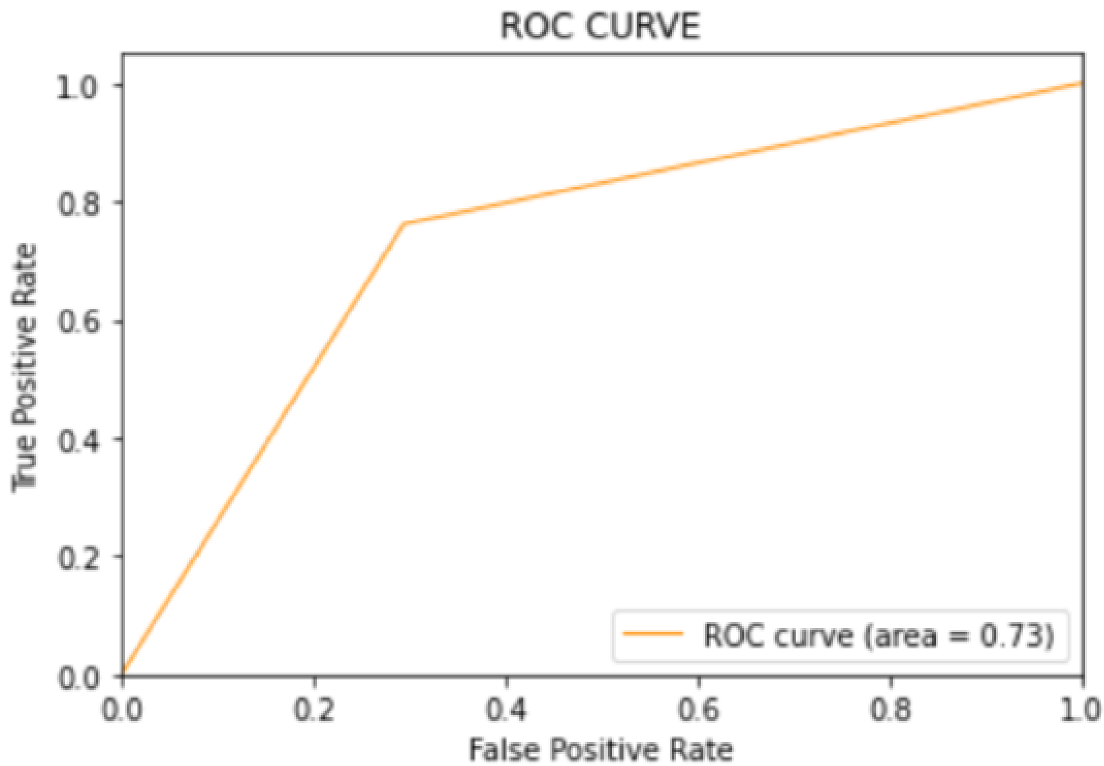


In[17]:

```
# Plot the ROC-AUC Curve for model-1 :

fpr, tpr, thresholds = roc_curve(y_test, y_pred1)

roc_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC CURVE')

plt.legend(loc="lower right")

plt.show()
```

Out[17]:

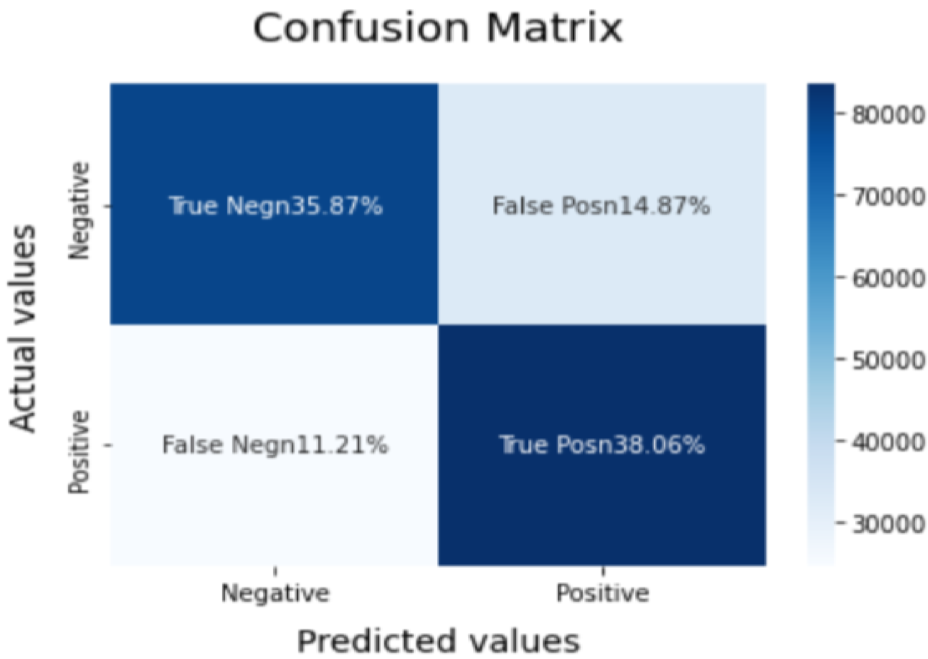


In[18]:

```
# Model-2 : SVM (Support Vector Machine).
SVCmodel = LinearSVC()
start1 = time.time()
SVCmodel.fit(X_train, y_train)
end1 = time.time()
print("\t\t⚠️⚠️⚠️ The training execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end1-start1))
start2 = time.time()
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test)
end2 = time.time()
print("\t\t⚠️⚠️⚠️ The test execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

Out[18]:

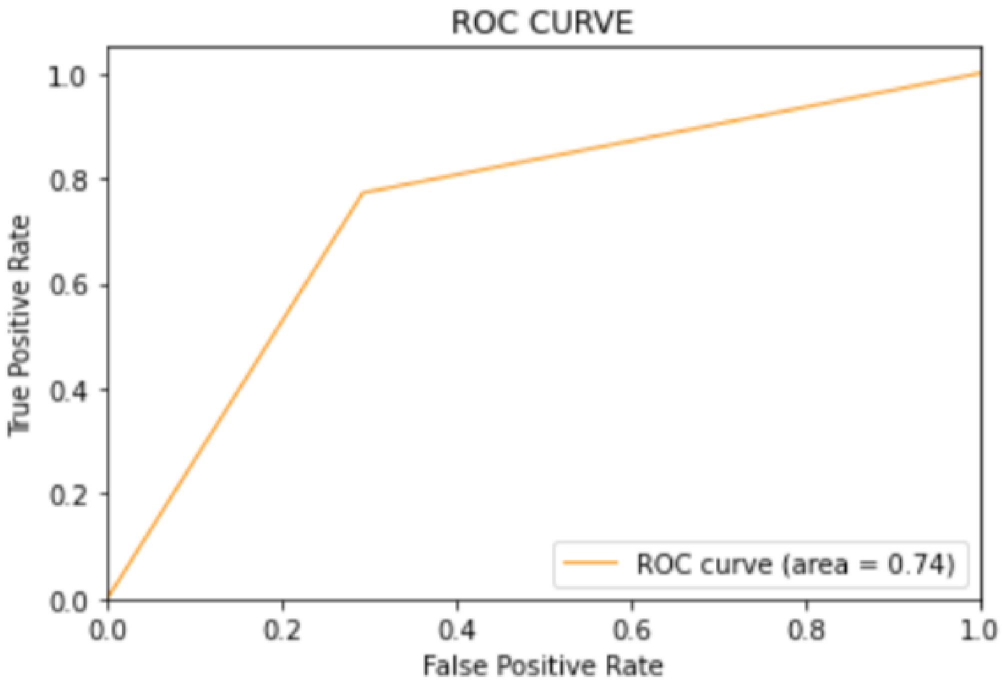
```
⚠️⚠️⚠️ The training execution time of this model is 24.07 seconds
⚠️⚠️⚠️

              precision    recall  f1-score   support

           0       0.76      0.71      0.73    111228
           1       0.72      0.77      0.74    107994

    accuracy                           0.74    219222
   macro avg       0.74      0.74      0.74    219222
   weighted avg       0.74      0.74      0.74    219222

            ⚠️⚠️⚠️ The test execution time of this model is 0.53
seconds ⚠️⚠️⚠️
```



In[19]:

```
# Plot the ROC-AUC Curve for model-2 :
fpr, tpr, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

Out[19]:



In[20]:

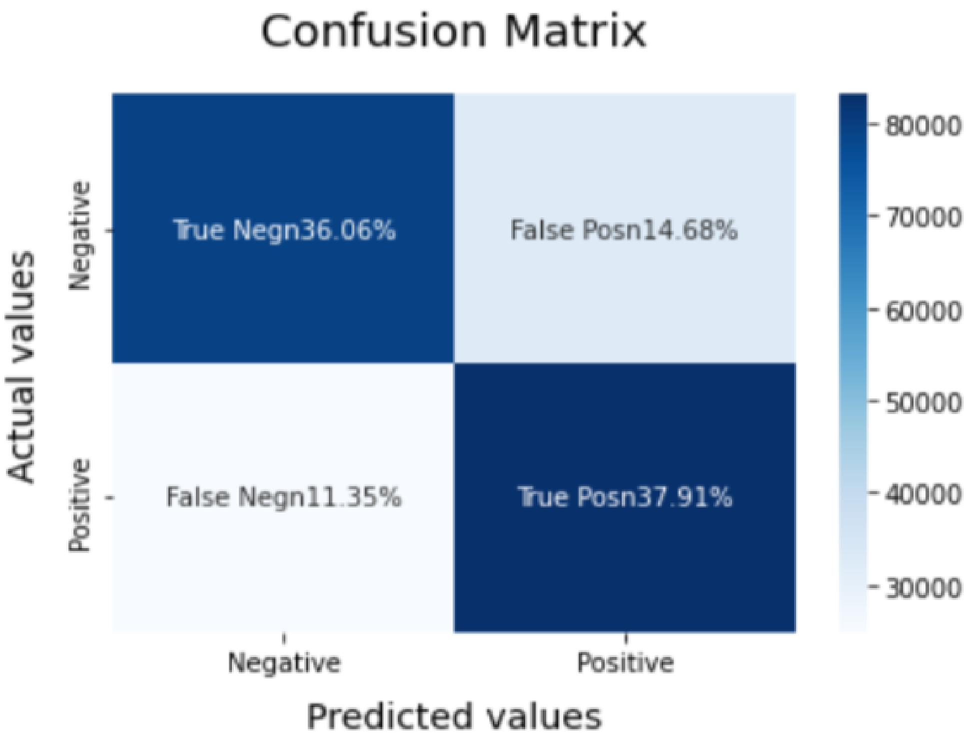
```
# Model-3 : Logistic Regression.
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
start1 = time.time()
LRmodel.fit(X_train, y_train)
end1 = time.time()
print("\t\t⚠️⚠️⚠️ The training execution time of this model is
{:.2f} seconds ⚠️⚠️⚠️\n".format(end1-start1))
start2 = time.time()
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)
end2 = time.time()
print("\t\t⚠️⚠️⚠️ The test execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

Out[20]:

⚠️⚠️⚠️ The training execution time of this model is 29.52 seconds ⚠️⚠️⚠️

```
              precision    recall  f1-score   support

           0       0.76      0.71      0.73    111228
           1       0.72      0.77      0.74    107994

    accuracy                           0.74    219222
   macro avg       0.74      0.74      0.74    219222
weighted avg       0.74      0.74      0.74    219222
```

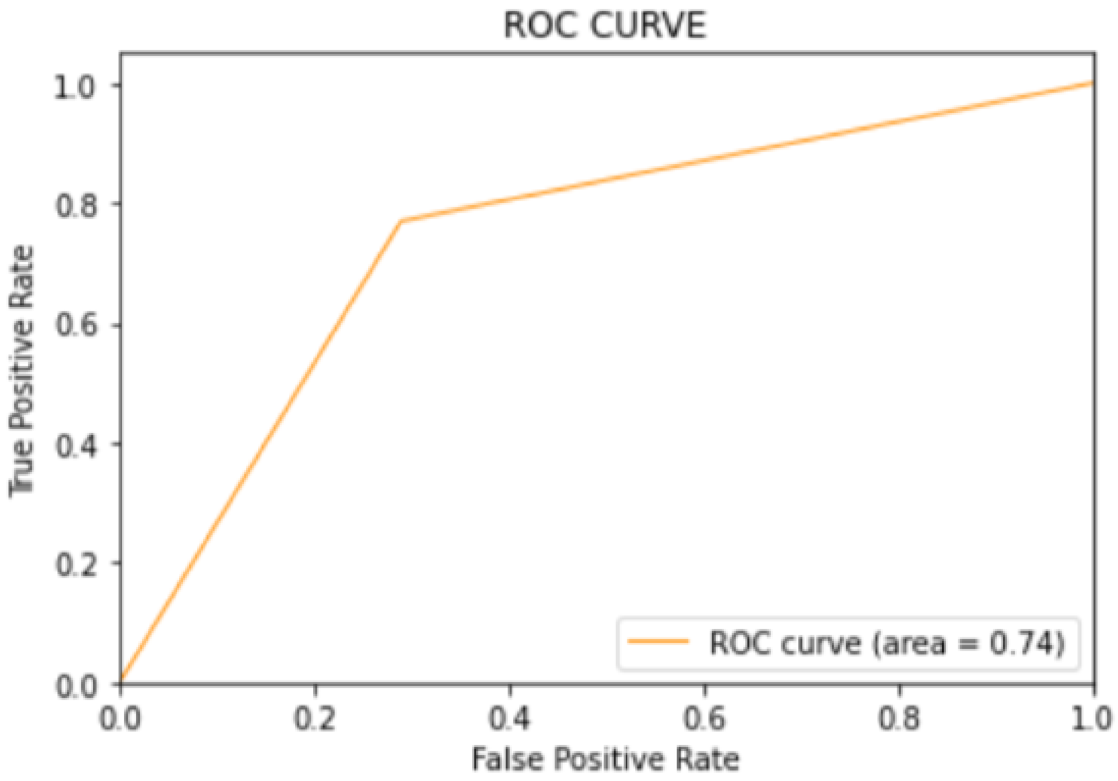
⚠️⚠️⚠️ The test execution time of this model is 0.54 seconds ⚠️⚠️⚠️



In[21]:

```
# Plot the ROC-AUC Curve for model-3 :
fpr, tpr, thresholds = roc_curve(y_test, y_pred3)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

Out[21]:



How do you choose the `value of k ( n_neighbors )` in KNN algorithm ?

In **KNN**, finding the value of k is not easy. A small value of k means that noise will have a higher influence on the result and a large value make it computationally expensive. Data scientists usually choose as an odd number if the number of classes is 2 and another simple approach to select k is set **K=sqrt(n).**

.

In[22]:

```
int(sqrt(len(df))) # k = sqrt(len(df) = sqrt(n) = sqrt(len(df)
```

Out[22]:

1208

In[23]:

```
# Model-4 : k-nearest neighbors.
knn = KNeighborsClassifier(n_neighbors=int(sqrt(len(df)))) #
sqrt(len(df)) = 1208
start1 = time.time()
knn.fit(X_train, y_train)
end1 = time.time()
print("⚠️⚠️⚠️ The training execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end1-start1))
start2 = time.time()
y_pred4 = knn.predict(X_test_reduced)
print("The accuracy of the model is : " +
str(knn.score(X_test_reduced, y_test_reduced))) # Calculate the
accuracy of the model
end_2 = time.time()
print("⚠️⚠️⚠️ The test execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

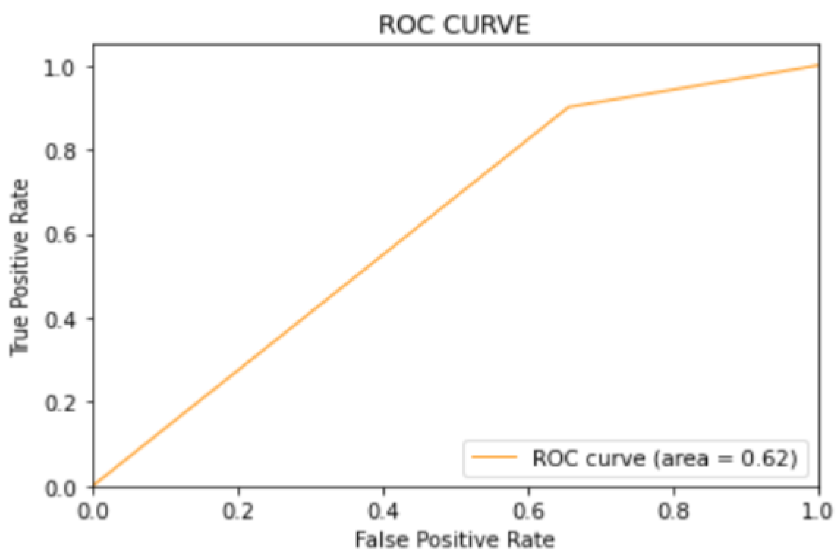
Out[23]:

```
⚠️⚠️⚠️ The training execution time of this model is 0.12 seconds ⚠️⚠️⚠️

The accuracy of the model is : 0.6163390046982621
⚠️⚠️⚠️ The test execution time of this model is -2.93 seconds ⚠️⚠️⚠️
```

In[24]:

```
# Plot the ROC-AUC Curve for model-4 :
fpr, tpr, thresholds = roc_curve(y_test_reduced, y_pred4)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

Out[24]:

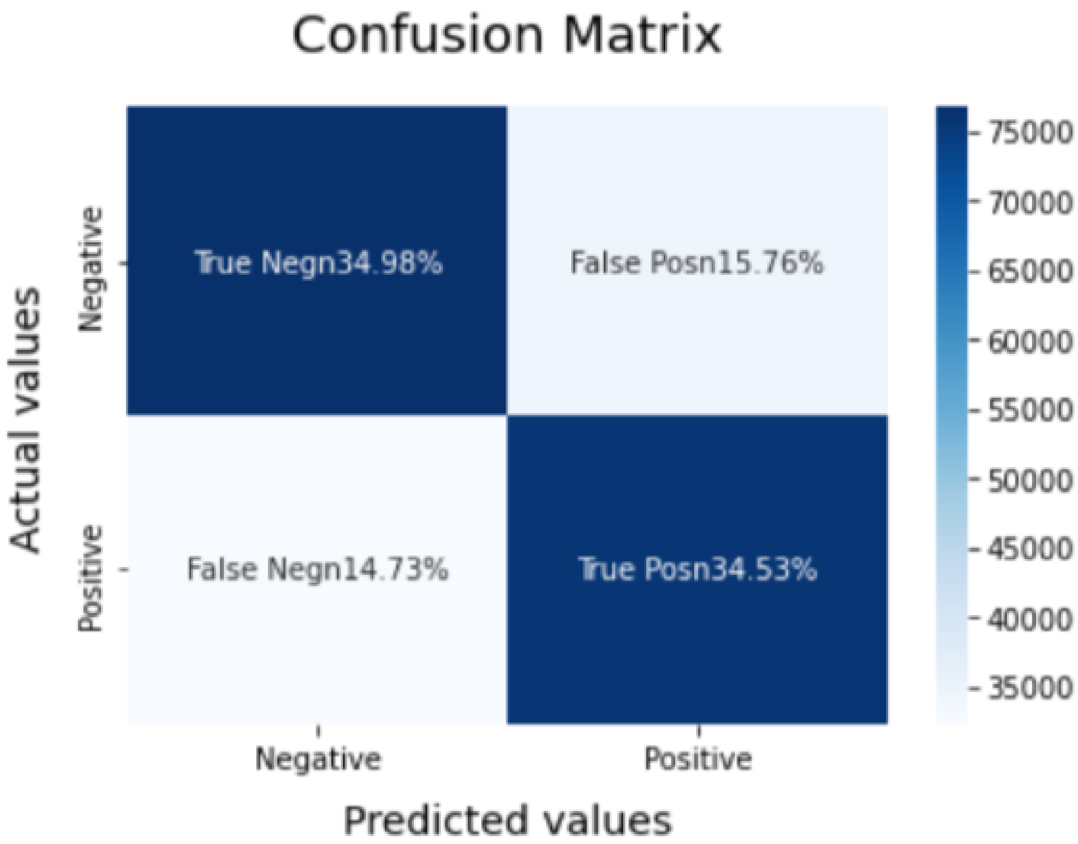


In[25]:

```
# Model-5 : Decision Tree
clf = DecisionTreeClassifier() ## Create Decision Tree classifer object
start1 = time.time()
clf = clf.fit(X_train_reduced, y_train_reduced) # Training Decision Tree
Classifer
LRmodel.fit(X_train_reduced, y_train_reduced)
end1 = time.time()
print("\t\t⚠️⚠️⚠️ The training execution time of this model is {:.2f}
seconds ⚠️⚠️⚠️\n".format(end1-start1))
start2 = time.time()
model_Evaluate(clf) ## Predict the response for test dataset
y_pred5 = clf.predict(X_test)
end2 = time.time()
print("\t\t⚠️⚠️⚠️ The test execution time of this model is {:.2f}

seconds ⚠️⚠️⚠️\n".format(end2-start2))
```

Out[25]:

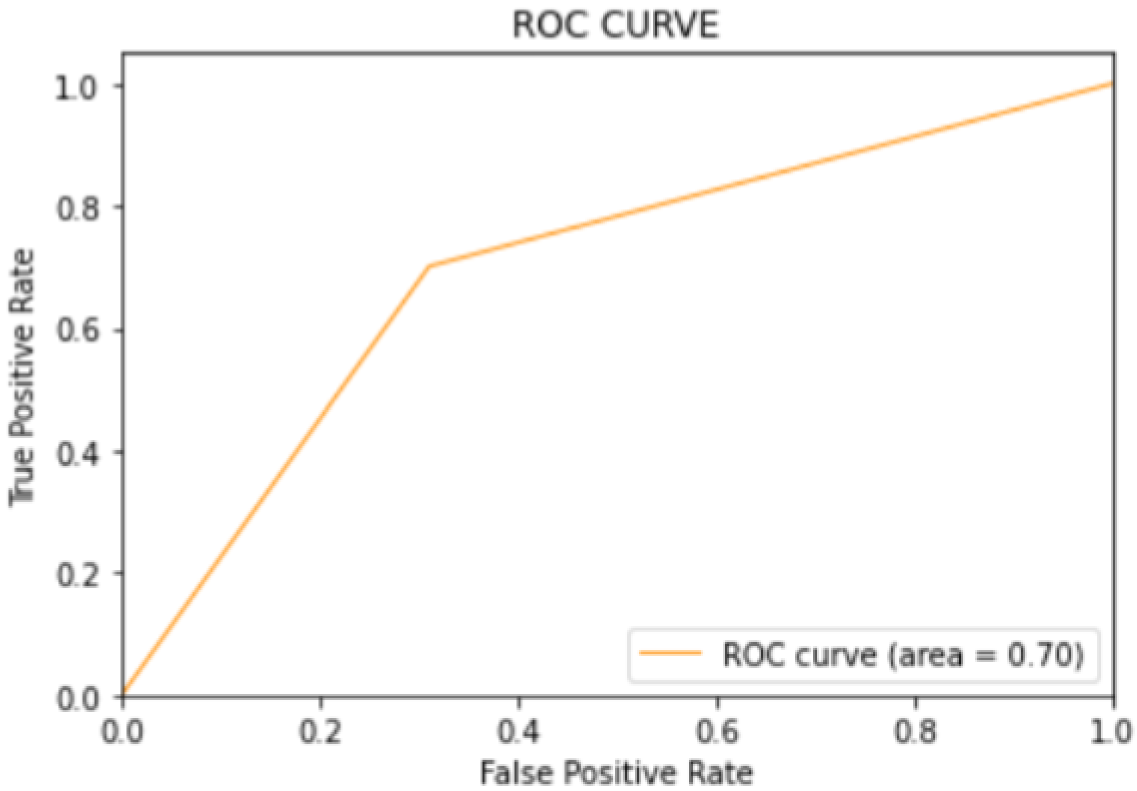
```
⚠️⚠️⚠️ The training execution time of this model is 43.47 seconds
⚠️⚠️⚠️

              precision    recall  f1-score   support

           0       0.70      0.69      0.70    111228
           1       0.69      0.70      0.69    107994

    accuracy                           0.70    219222
   macro avg       0.70      0.70      0.70    219222
weighted avg       0.70      0.70      0.70    219222

           ⚠️⚠️⚠️ The test execution time of this model is 1.06

seconds ⚠️⚠️⚠️
```



In[26]:

```
# Plot the ROC-AUC Curve for model-5 :
fpr, tpr, thresholds = roc_curve(y_test, y_pred5)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

Out[26]:


ROC CURVE
True Positive Rate
False Positive Rate
ROC curve (area = 0.70)
0.0
0.2
0.4
0.6
0.8
1.0