Project B

Sentimental Analysis

Supervisor:
Dr. Anwar Hossain

No	Members	ID
1	Eman Elrefai	20401066
2	Esraa Ahmed Fouad Omar	20399123
3	Lomai Mohamed saadeldin	20398049
4	Mariam Reziq	20401740

Contents

List of Figures

1 Data Collection

We collected data from different sources.Then we merged them.

1. Get from third party (Kaggle) : This data consists of 2 columns :
   (a) tweet: content of tweet (contain text)
   (b) label : positive(1) or Negative(0)
2. Scrape data manually from the Twitter website: This data consists of one column :
   (a) tweet: content of tweet (contain text)

1.1 Scraping

we scraped data using a few lines of code as shown in figure 1 . we used sntwitter.TwitterSearchScraper python library to scrape a query that we are interested in. In our project, we scraped data about AI and neuroscience and saved it as CSV file.

```
# Scrape data about ai and neuroscience
query = "ai and neuroscience"
tweets = []
limit = 50000


for tweet in sntwitter.TwitterSearchScraper(query).get_items():

    if len(tweets) == limit:
        break
    else:
        tweets.append([tweet.content])

df = pd.DataFrame(tweets, columns=['tweet'])
print(df)

# to save to csv
df.to_csv('ai_neuroscience_tweets.csv')
```

Figure 1: Code of Scraping data

Establish spark session and Read scraped data

```
df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('/content/ai_neuroscience_tweets.csv')
print(type(df))
print("data count " ,df.count())
print(df.show())
```

```
<ipython-input-48-630ff8b76d87>:7: UserWarning: SparkContext already exists in this scope
  warnings.warn("SparkContext already exists in this scope")
local[4]
<class 'pyspark.sql.dataframe.DataFrame'>
data count  36106
+--------------------+--------------------+
|                 _c0|               tweet|
+--------------------+--------------------+
|                   0|@cryptoworld202 Y...|
|After years of bu...| #Matrix is initi...|
|Data from the bra...| stored and proce...|
|                   1|@MatrixAINetwork ...|
|#AI #metaverse #w...|                null|
|                   2|@Metathea11 Yeah ...|
|I studied neurosc...|   I love philosophy|
|Which is why it’s...|                null|
|                   3|✨Safeguarding mem...|
|#OasisNetwork and...|                null|
|Privacy and confi...|                null|
|Control your data...|                null|
|                   4|Interested in lea...|
|https://t.co/EkIC...|                null|
|                   5|@PessoaBrain Lear...|
|                   6|Stanford's @stanf...|
|Information and a...|                null|
|                   7|@WhaleEverything ...|
|After years of bu...| #Matrix is blend...|
|                   8|@CryptoThro You c...|
+--------------------+--------------------+
only showing top 20 rows
```

Figure 2: Sample of our Scraped data

We applied some preprocessing methods to this data before merging it.

1. **Removing null and duplicated values :**

```
df = df.select(df["tweet"])
print("Count Null values in data " , df.filter(df.tweet.isNull()).count())
data_withoutnull = df.filter(df.tweet.isNotNull()).dropna()
print("Count Null values in data after removing null  " , data_withoutnull.filter(data_withoutnull.tweet.isNull()).count())
print("Count data after removing null  " , data_withoutnull.count())
```

```
Count Null values in data  11530
Count Null values in data after removing null   0
Count data after removing null   24576
```

Figure 3: Removing Null values

```
print("count of data before removing duplications :" , data_withoutnull.count())
print("Distinct count: "+str(data_withoutnull.distinct().count()))
#Remove Duplicates
tweets_distinct = data_withoutnull.dropDuplicates()
print("count of data after removing duplications : "+str(tweets_distinct.count()))
```

```
count of data before removing duplications : 24576
Distinct count: 23003
count of data after removing duplications : 23003
```

Figure 4: Removing duplicated values

2. **Labeling data :** Our data has no label. So, we labelled it manually using textblob python library. As you know, we use pyspark.So, we accessed this library using UDF and apply this function on the tweet column as shown in figure 5

```
# Function to get sentiment
def apply_blob(sentence):
    temp = TextBlob(sentence).sentiment[0]
    if temp == 0.0:
        return 0.0 # Neutral
    elif temp >= 0.0:
        return 1.0 # Positive
    else:
        return 2.0 # Negative

# UDF to write sentiment on DF
sentiment = udf(apply_blob, DoubleType())

data = tweets_distinct.select(lit( sentiment(tweets_distinct['tweet'])).alias("label"), "*")
data.show(5)
```

```
+-----+--------------------+
|label|               tweet|
+-----+--------------------+
|  1.0|@cz_binance 🔷 Ma...|
|  1.0|As a layman, the ...|
|  0.0|📣MAIN2022 call f...|
|  0.0|I think @DavidAMa...|
|  1.0|@gershbrain We ke...|
+-----+--------------------+
only showing top 5 rows
```

Figure 5: Labeling data

Then, we saved the data as a CSV file to merge it later.

1.2 Third party

We got data from Kaggle website. we read the data and prepared it by applying some preprocessing methods.

(a) **Removing null and duplicated values :**

```
twitter =  twitter.select('label' , 'tweet' )
#Dropping Rows With Empty Values
twitter_new = twitter.dropna()
print("count of data after dropping null" ,twitter_new.count()) #There is no null values
twitter_new.show(truncate = False,n=3)
```

```
count of data after dropping null 31962
```

Figure 6: Removing Null values

```
#Get Distinct Rows
print("count of data after dropping null" ,twitter_new.count())
distinctDF = twitter_new.distinct()
print("Distinct count: "+str(distinctDF.count()))
#removing duplicates
new_tweet =  twitter_new.dropDuplicates()
print("count of data after removing duplicated rows: "+str(new_tweet.count()))
distinctDF.show(truncate=False, n=3)
```

```
count of data after dropping null 31962
Distinct count: 29528
count of data after removing duplicated rows: 29528
```

Figure 7: Removing duplicated values

Then we saved it as a CSV file to merge it later

2 Data Preparation

2.1 Merging the datasets :

In the previous section, we saved 2 datasets in CSV format. Now , we read 2 datasets as spark sql data frame as shown in figure 8. Then, we merged them into one spark sql data frame as shown in figure 9.

```
df_twitter = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('/content/part-00000-d43eef32-dc2e-43f2-be6a-76fcb89370c4-c000.csv')
df_twitter_scrape = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('/content/part-00000-31cb3ff6-ed56-4ab9-8745-918bf546cde2-c000.csv')
df_twitter_scrape2 = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('/content/part-00001-31cb3ff6-ed56-4ab9-8745-918bf546cde2-c000.csv')
```

Figure 8: Read 2 datasets

```
cols = ['label', 'tweet']
df_twitter_ = df_twitter.select(cols)
print("Count of third_party data" , df_twitter_.count())
df_scrapping = df_twitter_scrape.union(df_twitter_scrape2)
print("Count of scraped data" , df_scrapping.count())
data_twitter_merge = df_scrapping.union(df_twitter_)
print("Count of data After merging 2 datasets " , data_twitter_merge.count())
data_twitter_merge.show(5)
```

```
Count of third_party data 29528
Count of scraped data 23003
Count of data After merging 2 datasets  52531
+-----+--------------------+
|label|               tweet|
+-----+--------------------+
|  1.0|@cz_binance 🔷 Ma...|
|  1.0|As a layman, the ...|
|  0.0|📣MAIN2022 call f...|
|  0.0|I think @DavidAMa...|
|  1.0|@gershbrain We ke...|
+-----+--------------------+
only showing top 5 rows
```

Figure 9: merge 2 datasets

2.2 Check null values :

We found one null value and removed it as shown in the figure 10

```
print("count of  merged data  "  ,  data_twitter_merge.count())
print("count of null values in merged data " , data_twitter_merge.filter(data_twitter_merge.tweet.isNull()).count())
data_twitter_merge = data_twitter_merge.dropna()
print("count of null values in merged data after droping null " , data_twitter_merge.count())
```

```
count of  merged data   52531
count of null values in merged data  1
count of null values in merged data after droping null  52530
```

Figure 10: Removing Null values from merged dataset

2.3 Remove duplication :

```
# To remove the duplicates:
print("count of  merged data  "  ,  data_twitter_merge.count())
print("count of Distinct values in merged data " , data_twitter_merge.distinct().count())
data_twitter_merge = data_twitter_merge.dropDuplicates()
print("count of merged data after drop duplications " , data_twitter_merge.count())# there is no duplicated
```

```
count of  merged data   52450
count of Distinct values in merged data  52450
count of merged data after drop duplications  52450
```

Figure 11: Removing duplicated values from merged dataset

2.4 Check imbalance :

we found a large imbalance in our data. Figure 12 shows this imbalance. The negative class (0) has many instances in our merged data compared to others. The imbalance in classes has a bad impact on the performance of the model. So, we solve this problem using oversampling. we dummy data in the minority classes (class (1) and class (2)). Figure 13 shows that all classes have approximately the same number.

```
The Number of instances in class 0 (Neg_class)  38255
The Number of instances in class 1 (Pos_class)  10946
The Number of instances in class 2 (Neutral_class)  3249
<BarContainer object of 3 artists>
```

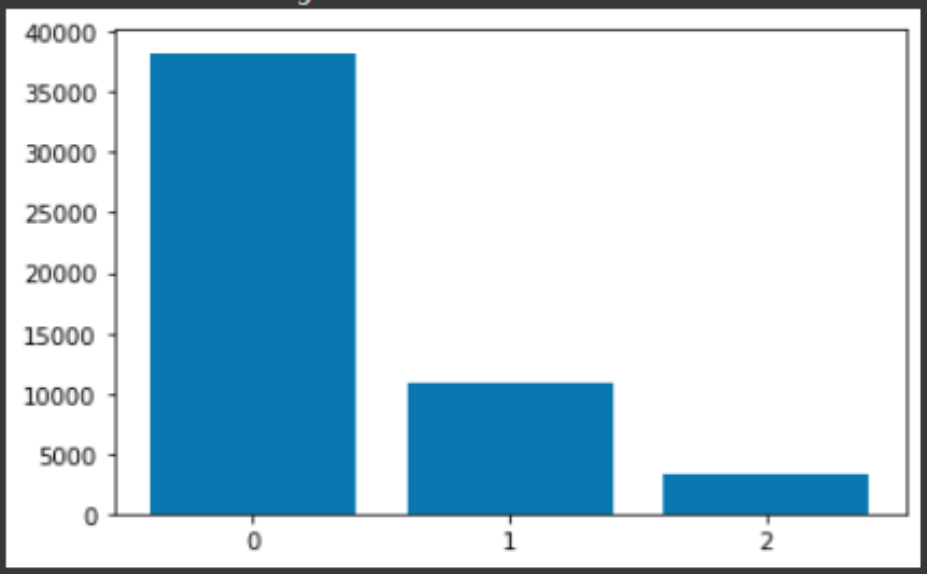

Figure 12: Number of instances in each class

```
The Number of instances in class 0 (Neg_class) After applying oversampling  38255
The Number of instances in class 1 (Pos_class) After applying oversampling  32838
The Number of instances in class 2 (Neutral_class) After applying oversampling 35739
<BarContainer object of 3 artists>
```

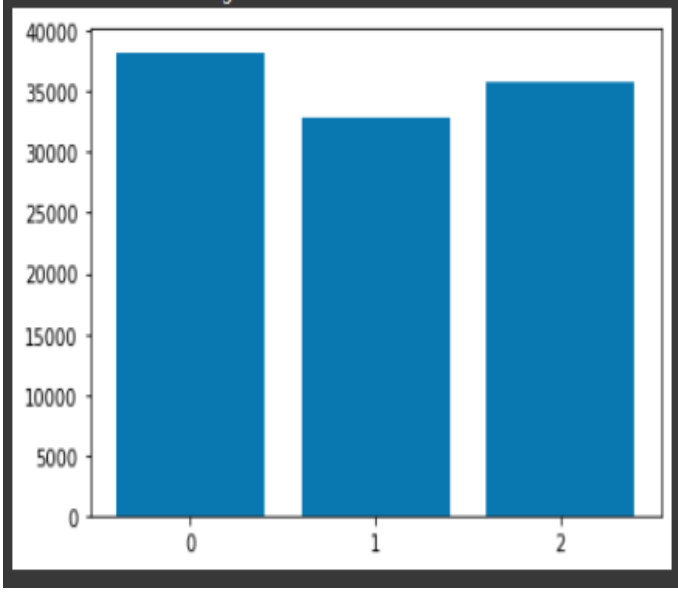

Figure 13: Number of instances in each class After oversampling

2.5 Remove(url, numbers ,special characters) and Tokenize tweet :

As shown in figure 14, our data has numbers and special characters. We removed them. first, we replace the URL and number with space. Then, we removed regex and tokenized sentences using regextokenizer spark library as shown in figure15 As you can see in figure 15, the data isn't still clean. we need more preprocessing.

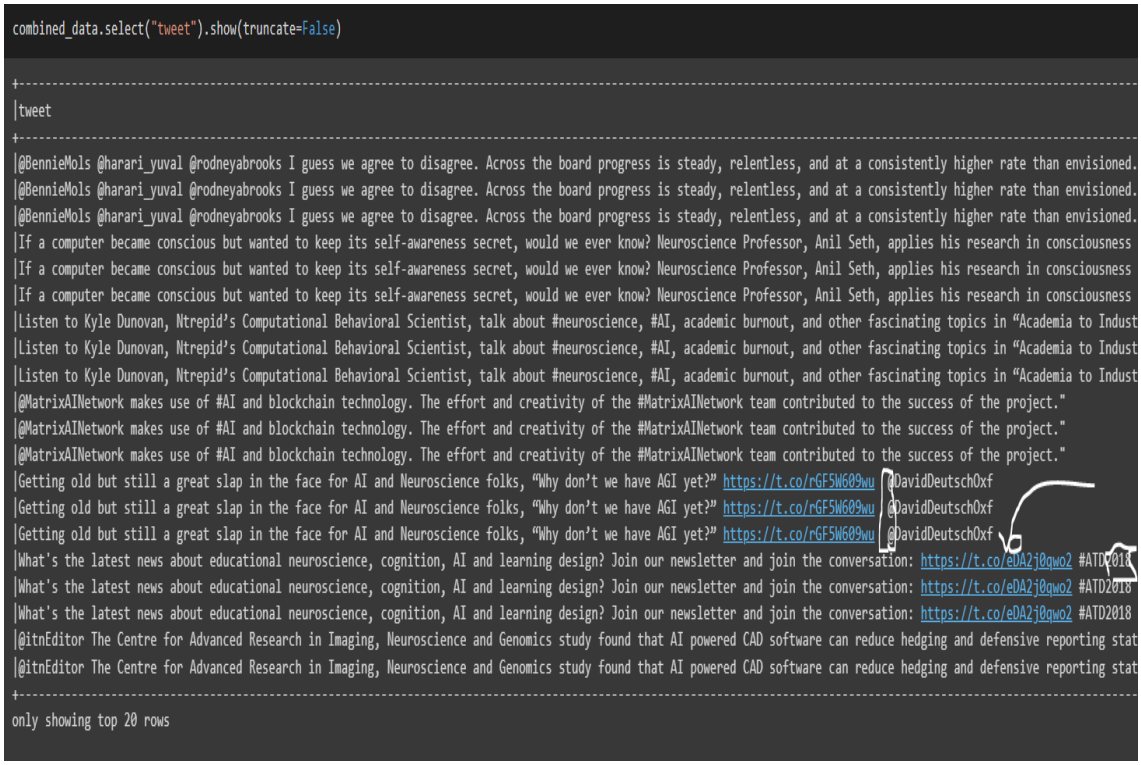

Figure 14: data before cleaning

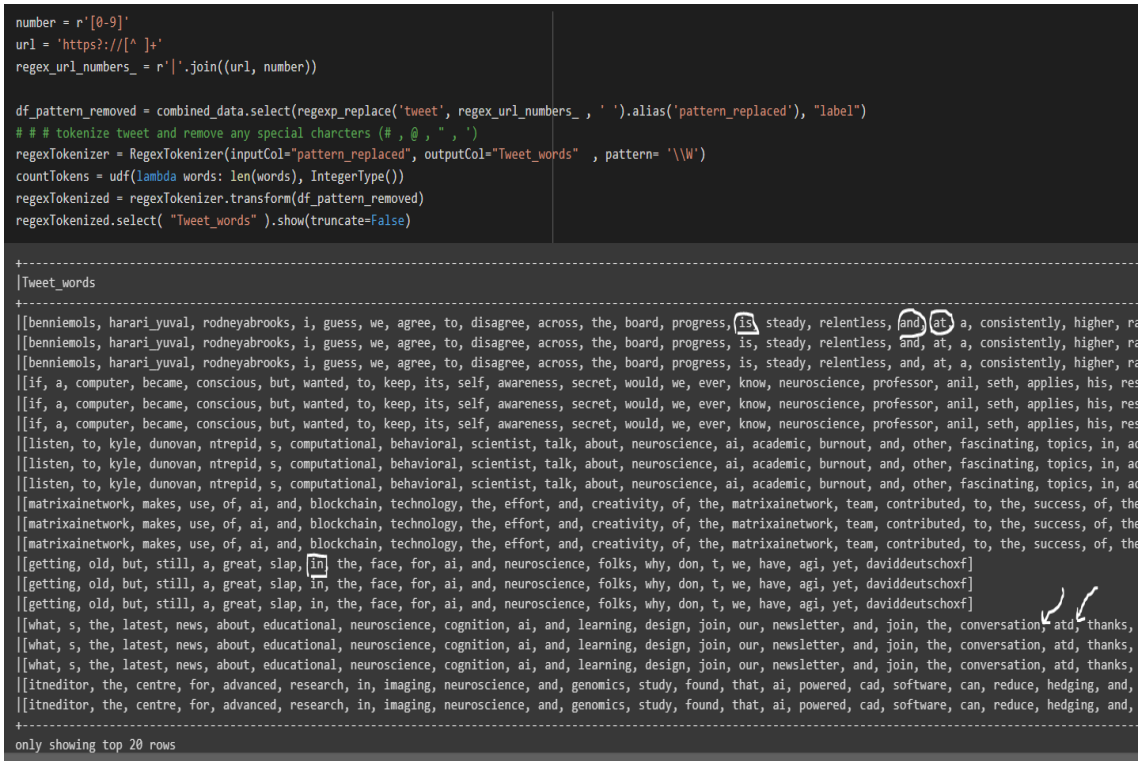

Figure 15: Removing regex and tokenize sentences

2.6 Remove stop words :

we used the StopWordsRemover spark ml feature library to remove stop words and lower words such as is, are, do, does ...etc. Figure 16 shows how we are successful in that.

```
remover = StopWordsRemover( inputCol="Tweet_words", outputCol="Tweet_without_stopword") # lower all charchter
tweets_distinct_remove_stopwords = remover.transform(regexTokenized)
tweets_distinct_remove_stopwords.select("Tweet_without_stopword").show(truncate=False)
```

```
+------------------------------------------------------------------------------------------------------------------------------------------------------
|Tweet_without_stopword
+------------------------------------------------------------------------------------------------------------------------------------------------------
|[benniemols, harari_yuval, rodneyabrooks, guess, agree, disagree, across, board, progress, steady, relentless, consistently, higher, rate, envisioned, given, progress,
|[benniemols, harari_yuval, rodneyabrooks, guess, agree, disagree, across, board, progress, steady, relentless, consistently, higher, rate, envisioned, given, progress,
|[benniemols, harari_yuval, rodneyabrooks, guess, agree, disagree, across, board, progress, steady, relentless, consistently, higher, rate, envisioned, given, progress,
|[computer, became, conscious, wanted, keep, self, awareness, secret, ever, know, neuroscience, professor, anil, seth, applies, research, consciousness, better, underst
|[computer, became, conscious, wanted, keep, self, awareness, secret, ever, know, neuroscience, professor, anil, seth, applies, research, consciousness, better, underst
|[computer, became, conscious, wanted, keep, self, awareness, secret, ever, know, neuroscience, professor, anil, seth, applies, research, consciousness, better, underst
|[listen, kyle, dunovan, ntrepid, computational, behavioral, scientist, talk, neuroscience, ai, academic, burnout, fascinating, topics, academia, industry, podcast, bra
|[listen, kyle, dunovan, ntrepid, computational, behavioral, scientist, talk, neuroscience, ai, academic, burnout, fascinating, topics, academia, industry, podcast, bra
|[listen, kyle, dunovan, ntrepid, computational, behavioral, scientist, talk, neuroscience, ai, academic, burnout, fascinating, topics, academia, industry, podcast, bra
|[matrixainetwork, makes, use, ai, blockchain, technology, effort, creativity, matrixainetwork, team, contributed, success, project]
|[matrixainetwork, makes, use, ai, blockchain, technology, effort, creativity, matrixainetwork, team, contributed, success, project]
|[matrixainetwork, makes, use, ai, blockchain, technology, effort, creativity, matrixainetwork, team, contributed, success, project]
|[getting, old, still, great, slap, face, ai, neuroscience, folks, agi, yet, daviddeutschoxf]
|[getting, old, still, great, slap, face, ai, neuroscience, folks, agi, yet, daviddeutschoxf]
|[getting, old, still, great, slap, face, ai, neuroscience, folks, agi, yet, daviddeutschoxf]
|[latest, news, educational, neuroscience, cognition, ai, learning, design, join, newsletter, join, conversation, atd, thanks, highvaluemembers, get, website, traffic]
|[latest, news, educational, neuroscience, cognition, ai, learning, design, join, newsletter, join, conversation, atd, thanks, highvaluemembers, get, website, traffic]
|[latest, news, educational, neuroscience, cognition, ai, learning, design, join, newsletter, join, conversation, atd, thanks, highvaluemembers, get, website, traffic]
|[itneditor, centre, advanced, research, imaging, neuroscience, genomics, study, found, ai, powered, cad, software, reduce, hedging, defensive, reporting, statements, r
|[itneditor, centre, advanced, research, imaging, neuroscience, genomics, study, found, ai, powered, cad, software, reduce, hedging, defensive, reporting, statements, r
+------------------------------------------------------------------------------------------------------------------------------------------------------
only showing top 20 rows
```

Figure 16: Removing stop words

2.7 Vectorizing features :

As we know machines understand only numbers. So, we need to convert text to numbers. we used Term frequency-inverse document frequency (TF-IDF) method is a feature vectorization method used to reflect the importance of a term to a document in the corpus.TF is used to generate the term frequency vectors.IDF is an Estimator which is fit on a dataset and produces an IDFModel. The IDFModel takes feature vectors created from HashingTF and scales each feature. Intuitively, it down-weights features which appear frequently in a corpus.As shown in figure 17, TF takes feature vector with size $2^{18} = 262144$. Then, pass its output to IDF.

```python
def Vectorize_features(name_input_col, name_out_col , train_data , test_data):
    hashtf = HashingTF(numFeatures=2**18, inputCol=name_input_col, outputCol='tf')
    idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
    selector = ChiSqSelector(numTopFeatures=2**10,featuresCol='features', outputCol="features_selected")
    label_stringIdx = StringIndexer(inputCol = name_out_col, outputCol = "sentiment")
    pipeline = Pipeline(stages=[ hashtf, idf, label_stringIdx  , selector])
    pipelineFit = pipeline.fit(train_data)
    train_new = pipelineFit.transform(train_data)
    test_new = pipelineFit.transform(test_data)
    return train_new , test_new
```

Figure 17: Functions used to Vectorize Features and select 1024 the most important features

2.8 Features Selection :

As we know increasing the number of features may cause overfitting.So, we used Chisquare to select only 1024 features which are the most important. As shown in figure 17, we passed the output of IDF to chisqSelector to reduce the number of features. Figure 18 shows the output of this step.

```
train_df , test_df = Vectorize_features("Tweet_without_stopword" , "label" ,train_set, test_set  )
train_df.show()
```

```
+----------------------+-----+--------------------+--------------------+---------+--------------------+
|Tweet_without_stopword|label|                  tf|            features|sentiment|   features_selected|
+----------------------+-----+--------------------+--------------------+---------+--------------------+
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|                    []|  1.0|      (262144,[],[])|      (262144,[],[])|      2.0|        (1024,[],[])|
|  [__rej__, think, ...|  1.0|(262144,[25513,31...|(262144,[25513,31...|      2.0|(1024,[119,134,20...|
|  [__rej__, think, ...|  1.0|(262144,[25513,31...|(262144,[25513,31...|      2.0|(1024,[119,134,20...|
|  [__rej__, think, ...|  1.0|(262144,[25513,31...|(262144,[25513,31...|      2.0|(1024,[119,134,20...|
|  [_deame, kelly_mc...|  1.0|(262144,[10583,23...|(262144,[10583,23...|      2.0|(1024,[252,379,56...|
|  [_deame, kelly_mc...|  1.0|(262144,[10583,23...|(262144,[10583,23...|      2.0|(1024,[252,379,56...|
|  [_humblebeast_, e...|  1.0|(262144,[7453,640...|(262144,[7453,640...|      2.0|(1024,[32,452,631...|
|  [_humblebeast_, e...|  1.0|(262144,[7453,640...|(262144,[7453,640...|      2.0|(1024,[32,452,631...|
|  [_kriesz_, monstr...|  1.0|(262144,[3059,118...|(262144,[3059,118...|      2.0|(1024,[81,133,219...|
|  [_kriesz_, monstr...|  1.0|(262144,[3059,118...|(262144,[3059,118...|      2.0|(1024,[81,133,219...|
|  [_kriesz_, monstr...|  1.0|(262144,[3059,118...|(262144,[3059,118...|      2.0|(1024,[81,133,219...|
|  [_lost_letters_, ...|  1.0|(262144,[54961,10...|(262144,[54961,10...|      2.0|(1024,[526,631,82...|
+----------------------+-----+--------------------+--------------------+---------+--------------------+
only showing top 20 rows
```

Figure 18: data after vectorization

3 Insights

we implemented some visualization in the data to Know more about common topic or words.From Figure 19 and Figure19, we noticed that the behaviour of user is towards neuroscience and AI. Figure 20 is another nice visualization for first 200 rows in the data to reflect high frequencies words in these rows.Big word has high frequency.

```
word_count = clean_data.withColumn('word', explode( clean_data.Tweet_without_stopword)).groupby('word').count()
sqlContext.registerDataFrameAsTable(word_count, "word_count")
df = sqlContext.sql("Select * from word_count ORDER BY word_count.count DESC limit 5")
df = df.toPandas()
df.set_index('word').plot.bar()
```

```
<AxesSubplot:xlabel='word'>
```

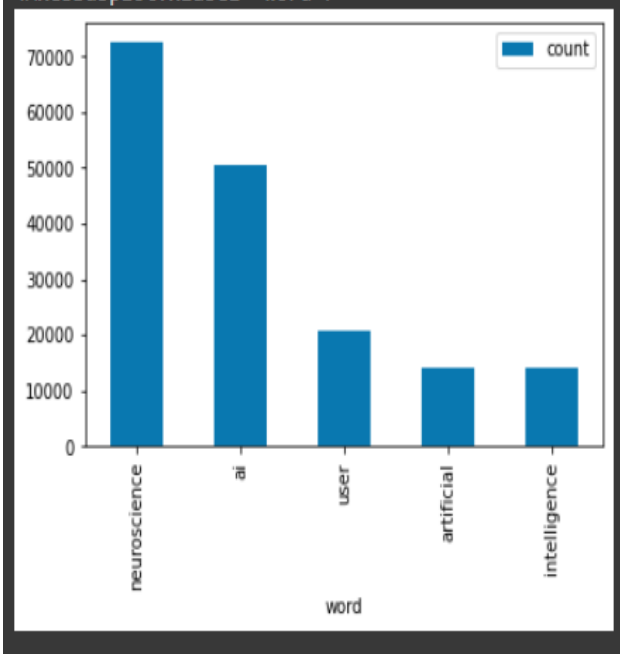

Figure 19: 5 words with the highest frequency in the data

```
most_freq_tweet = clean_data.groupBy("Tweet_without_stopword").count().orderBy(col("count").desc())
sqlContext.registerDataFrameAsTable(most_freq_tweet, "most_freq_tweet")
df_most_freq_tweet = sqlContext.sql("Select * from most_freq_tweet limit 5")
df_most_freq_tweet  = df_most_freq_tweet.toPandas()
df_most_freq_tweet.set_index('Tweet_without_stopword').plot.bar()
```

```
<AxesSubplot:xlabel='Tweet_without_stopword'>
```

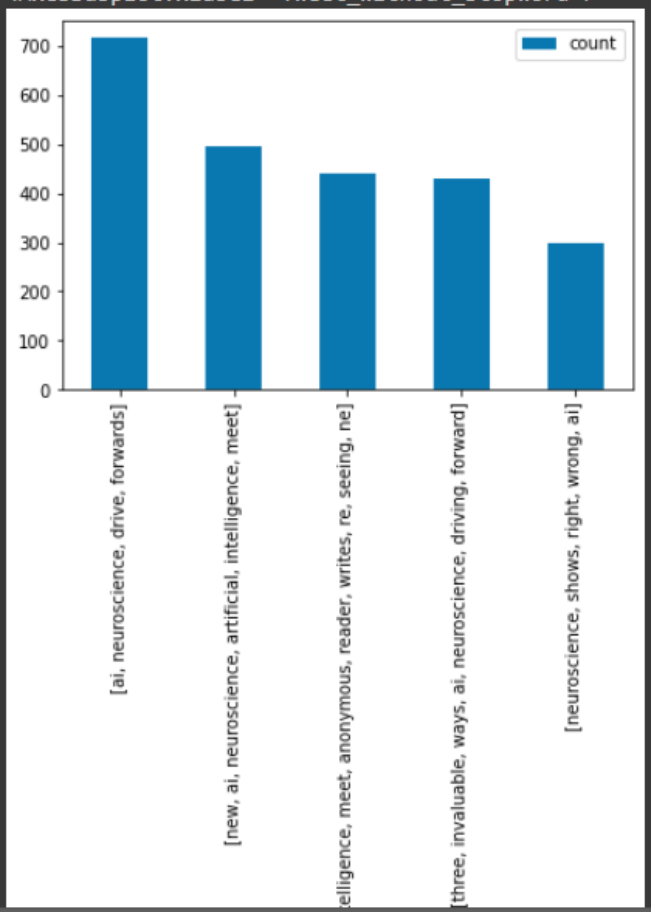

Figure 20: 5 tweets with the highest frequency in the data

```
from wordcloud import WordCloud, STOPWORDS
word_common_df =clean_data.withColumn('word', explode( clean_data.Tweet_without_stopword))
sqlContext.registerDataFrameAsTable(word_common_df, "wordcloud")
df_cloud = sqlContext.sql("Select * from wordcloud limit 200")
df_cloud =df_cloud.toPandas()
text = " ".join(w for w in df_cloud.word)
#  Generate word cloud
word_cloud = WordCloud(width=400,height=200,random_state=1,background_color="black",colormap="Pastel1",collocations=False).generate(text)
plt.imshow(word_cloud)
plt.axis("off")
plt.show()
```

Figure 21: visualize words in the first 200 rows

4 Split data

we spiltted data into 2 partitions: train and test with a ratio 90 to 10 respectively as shown in Figure 22.

```
(train_set, test_set) = clean_data.randomSplit([0.90, 0.1], seed = 2000)
```

Figure 22: Split data

5 Build Model

We built 3 different classifiers: Logistic Regression, Naive Bayes, and SVM. we tried to find the best model fit data. we used the ml spark library to import these models s shown in figure 23.
Notice that:
Logistic Regression and Naive Bayes class support multinomial logistic (softmax) and binomial logistic regression but don't support multinomial-only binary. So, we used one vs rest

```
lr = LogisticRegression(maxIter=10)
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
lsvc = LinearSVC(maxIter=20, regParam=0.01)
# instantiate the One Vs Rest Classifier.
ovr = OneVsRest(classifier=lsvc)
models_list = [lr , nb ,  ovr ]
models_name = ["Logistic Regression" , "Naive Bayes" , "SVM"]
```

Figure 23: Build models

6 Train Model

We implemented a function to train, test and evaluate the model. we used accuracy ,precision ,recall and f1score to evaluate models as shown in figure 25. Then, use for loop through each model and each time call this function as shown in figure 24

```
for i in range(len(models_list)):
  predictions, f1score_ , accuracy , precision , recall = train_Evaluate_model(train_df,test_df ,models_list[i] , models_name[i])
```

Figure 24: Train Models

```python
def train_Evaluate_model(train_data,test_data ,model , model_name):
    #----------------Train model ^_^--------------------
    Model = model.fit(train_data)
    #-----------------Test Model -------------------
    predictions = Model.transform(test_data)
    train_pre = Model.transform(train_data)
    print(model_name)
    print("sample of our prediction" )
    print(predictions.show(5))
    print("__"*30)
    #-----------------Evaluate Model -------------------
    evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
    f1score_ = evaluator.evaluate(predictions)
    print('evaluator.getMetricName(): ' , evaluator.getMetricName() ," = ", f1score_)
    print("__"*30)
    train_accuracy = evaluator.evaluate( train_pre , {evaluator.metricName: "accuracy"})
    print('train accuracy = ' ,train_accuracy)
    accuracy = evaluator.evaluate(predictions , {evaluator.metricName: "accuracy"})
    print('test accuracy = ' ,accuracy)
    print("__"*30)
    precision = evaluator.evaluate(predictions , {evaluator.metricName: "weightedPrecision"})
    print('precision = ' ,precision)
    print("__"*30)
    recall = evaluator.evaluate(predictions , {evaluator.metricName: "weightedRecall"})
    print('recall = ' ,recall)
    print("__"*30)
    return predictions, f1score_ , accuracy , precision , recall
```

Figure 25: Functions for Train , test and evaluate models

7 visualization

This section shows the results of the three selected models (Logistic Regression, Naive Bayes, and Support Vector Machine) according to

1. Performance Metrics
2. ROC/ AUC curve
3. Precision vs Recall curve
4. Confusion Matrix

After that, we only compare these selected models by performance metrics and choose the best model.

We built a function to visualize these evaluation matrices as shown in figure 26. we call it in the same for loop that is shown in figure 24 and passes the predictions of each model to this function to plot ROC Curve and precision vs recall curve.

```
def Visualize_evalution_matrices(predictions ,model , model_name):
    if (model_name == "SVM"):
      y_score_prob = predictions.select(vector_to_array("rawPrediction")[1]).rdd.keys().collect()
    else:
      y_score_prob = predictions.select(vector_to_array("probability")[1]).rdd.keys().collect()
    y_true = predictions.select("label").rdd.keys().collect()


    #-------------------------------Roc Curve  ^_^ ----------------------
    fpr, tpr, thresholds = roc_curve(y_true, y_score_prob,pos_label=1)
    roc_auc = auc(fpr, tpr)
    RocCurveDisplay(fpr = fpr, tpr = tpr, roc_auc = roc_auc, estimator_name = model.__class__.__name__).plot()
    plt.title('ROC/AUC')
    plt.show()

    #-------------------------------recall_percision ^_^ ----------------------
    precision, recall, _ = precision_recall_curve(y_true ,y_score_prob,pos_label=1)
    PrecisionRecallDisplay(precision = precision, recall = recall).plot()
    plt.title('Precision Vs Recall')
    plt.show()
```

Figure 26: Function to plot ROC and precision vs recall Curve

Also, we built 2 functions to calculate the confusion matrix as shown in figure 27. we also call these functions in the same for loop and pass the predictions of each model to this function.

```
def Confusion_matrix(predictions):
    predictionAndLabels = predictions.select(col("label"), col("prediction"))
    metrics = MulticlassMetrics(predictionAndLabels.rdd.map(lambda x: tuple(map(float, x))))
    confusion_matrix = metrics.confusionMatrix().toArray()
    labels = [int(l) for l in metrics.call('labels')]
    confusion_matrix_df = pd.DataFrame(confusion_matrix , index=labels, columns=labels)
    return confusion_matrix_df , confusion_matrix

def plot_confusionMatrix(Matrix):
    fig, ax = plt.subplots(figsize=(7.5, 7.5))
    ax.matshow(Matrix, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(Matrix.shape[0]):
        for j in range(Matrix.shape[1]):
            ax.text(x=j, y=i,s=Matrix[i, j], va='center', ha='center', size='xx-large')

    plt.xlabel('Predictions', fontsize=18)
    plt.ylabel('Actuals', fontsize=18)
    plt.title('Confusion Matrix', fontsize=18)
    plt.show()
```

Figure 27: Function to calculate and plot confusion matrix

Let's see the output of each model.

7.1 Logistic Regression

```
_____________________________________________________________________
evaluator.getMetricName():  f1  =  0.9491460129539462
_____________________________________________________________________
train accuracy =  0.9848813890418645
test accuracy =  0.9494005475313887
_____________________________________________________________________
precision =  0.950241750748221
_____________________________________________________________________
recall =  0.9494005475313887
```

Figure 28: Evaluation Matrices for Logistic Regression

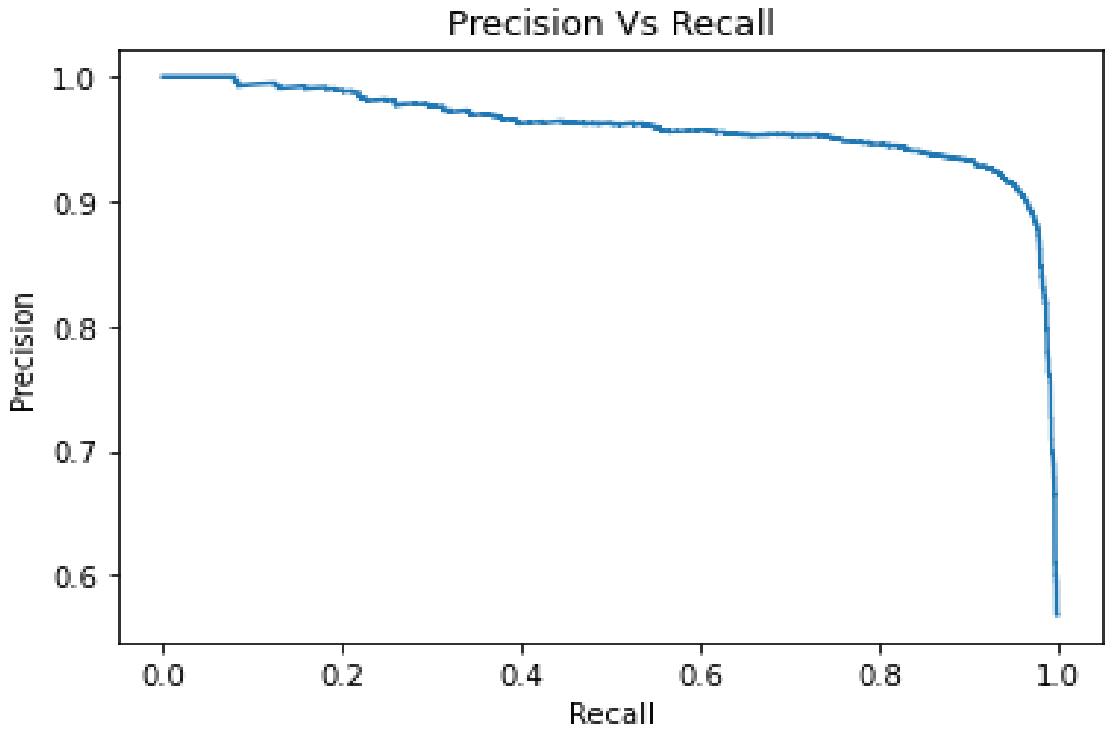

Figure 29: Precision vs Recall Curve for Logistic Regression

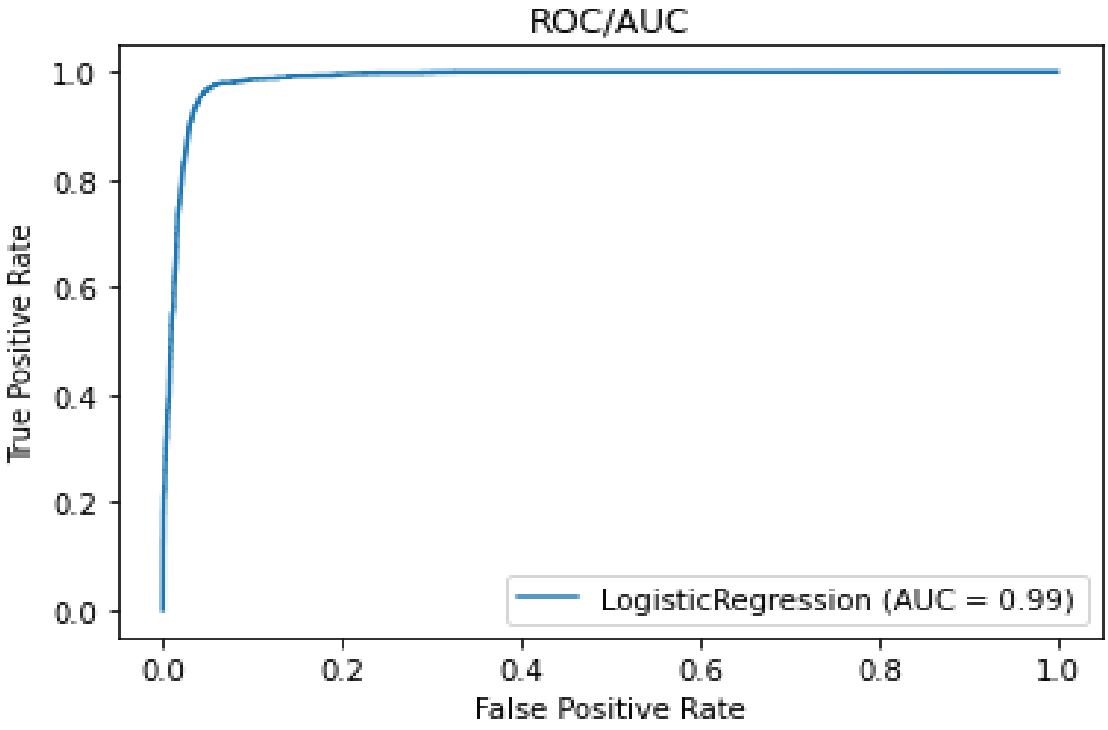

Figure 30: ROC Curve for Logistic Regression

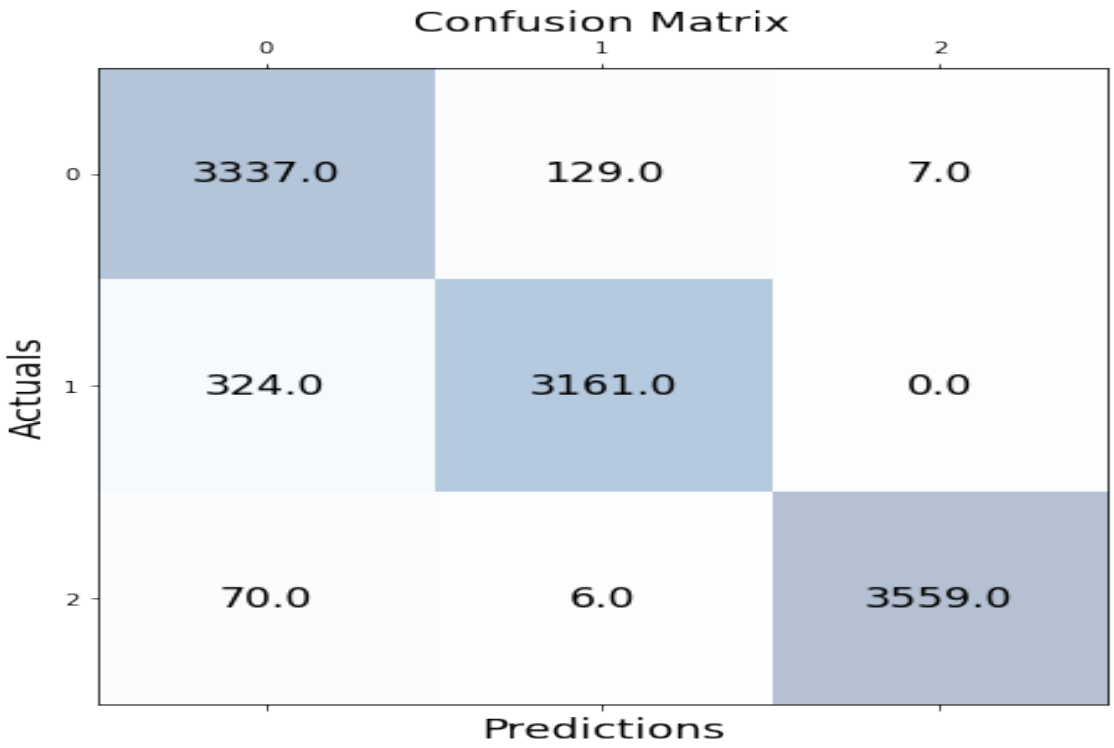

Figure 31: Confusion Matrix For Logistic Regression

7.2 Naive Bayes

```
evaluator.getMetricName():  f1  =  0.8497350175904601
train accuracy =  0.8770664699342262
test accuracy =  0.8514113093552346
precision =  0.8540704607205339
recall =  0.8514113093552347
```

Figure 32: Evaluation Matrices for Naive Bayes

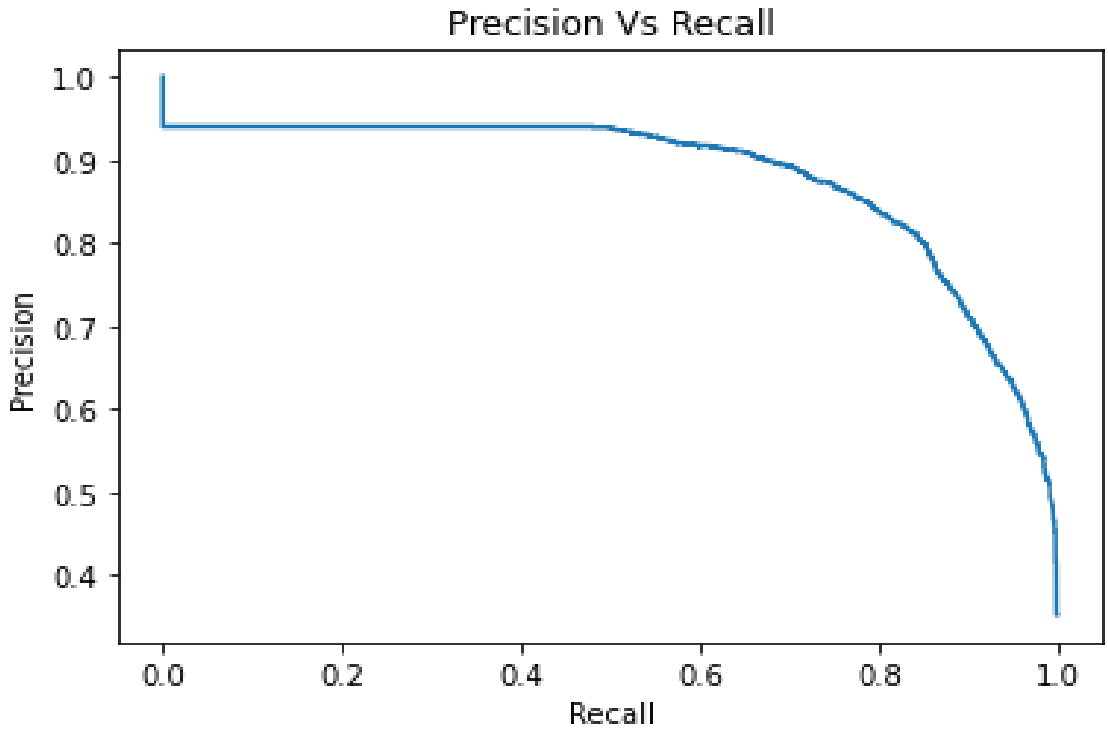

Figure 33: Precision vs Recall Curve for Naive Bayes

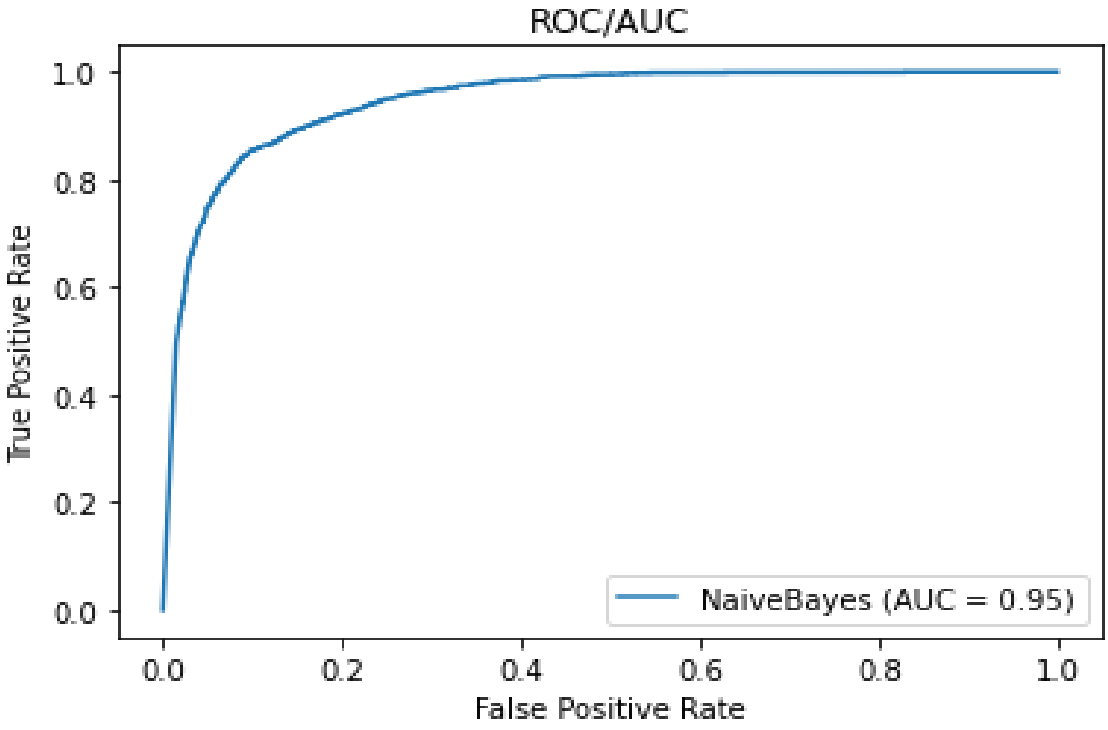

Figure 34: ROC Curve for Naive Bayes

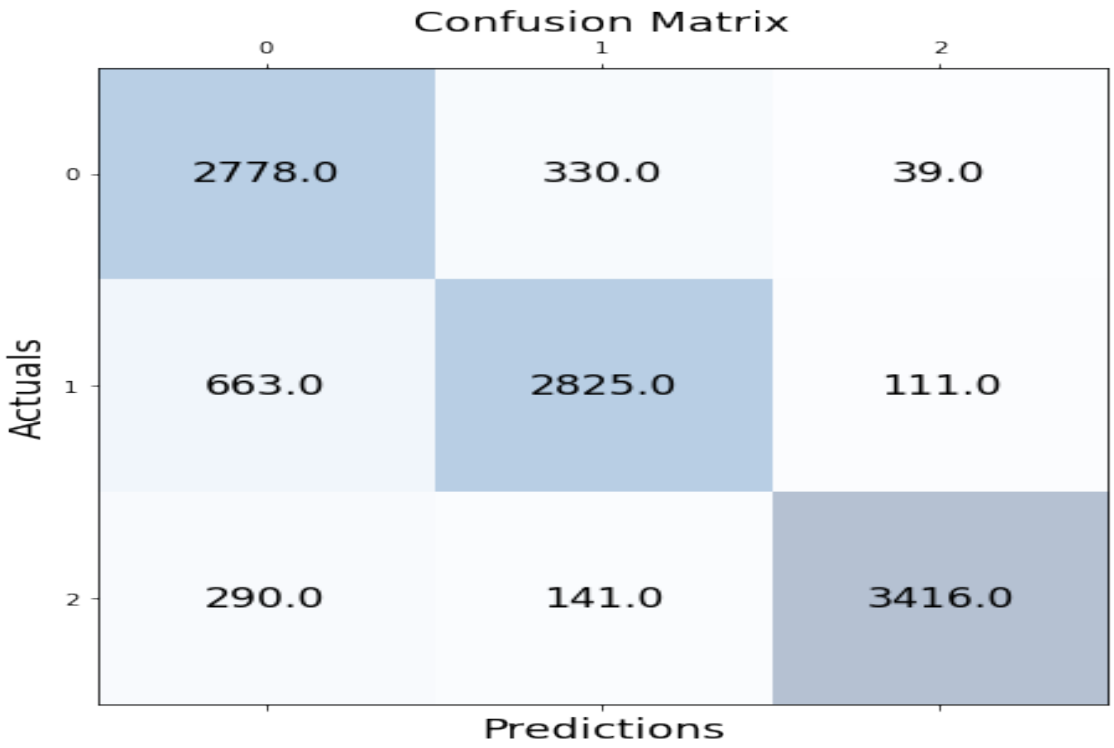

Figure 35: Confusion Matrix For Naive Bayes

7.3 SVM

```
evaluator.getMetricName():  f1  =  0.9387009751467303
train accuracy =  0.9657831024844398
test accuracy =  0.9389219295761352
precision =  0.9386161173167471
recall =  0.9389219295761353
```

Figure 36: Evaluation Matrices for SVM

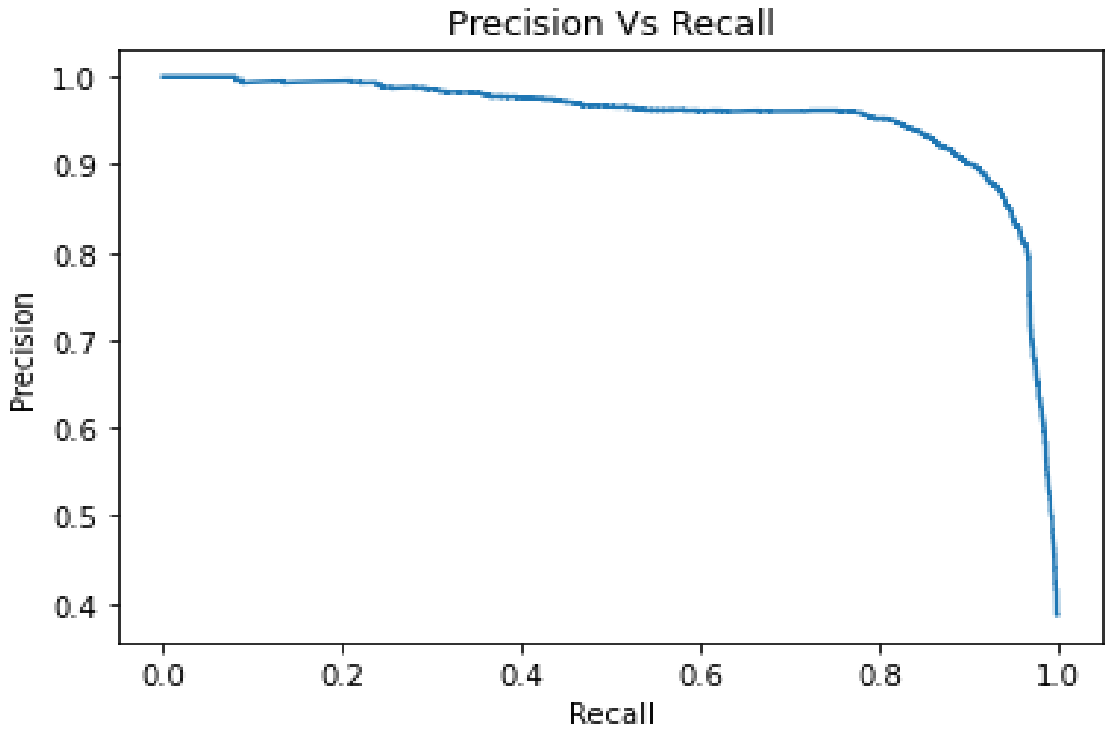

Figure 37: Precision vs Recall Curve for SVM

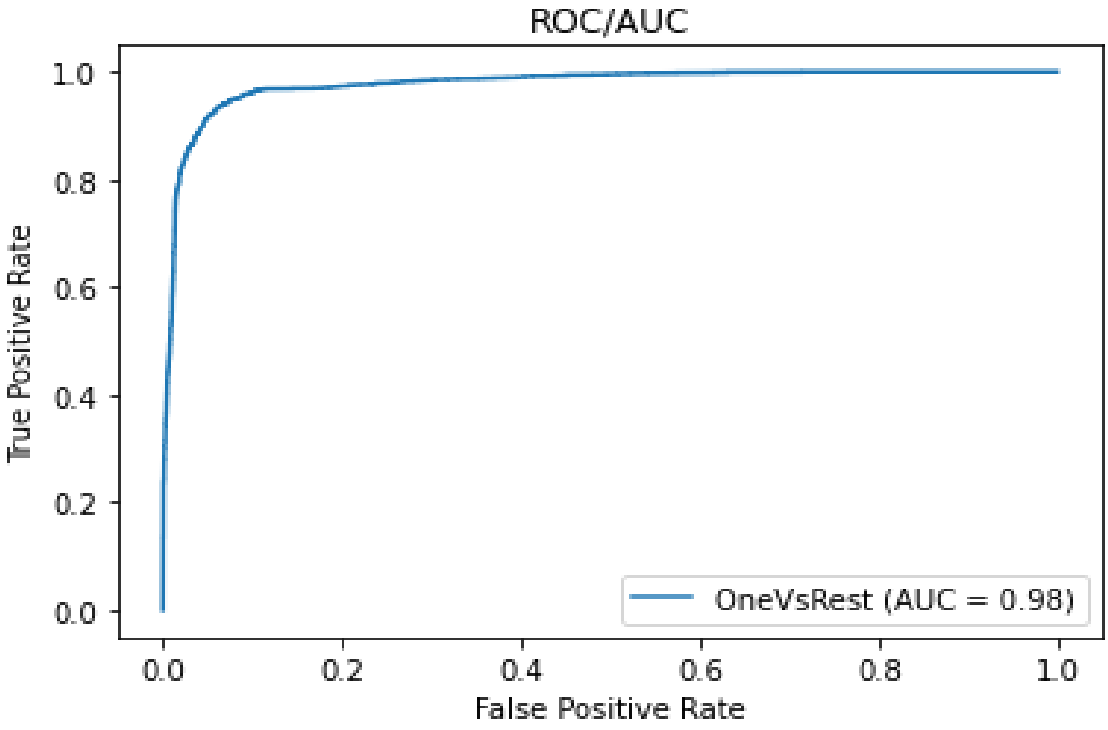

Figure 38: ROC Curve for SVM

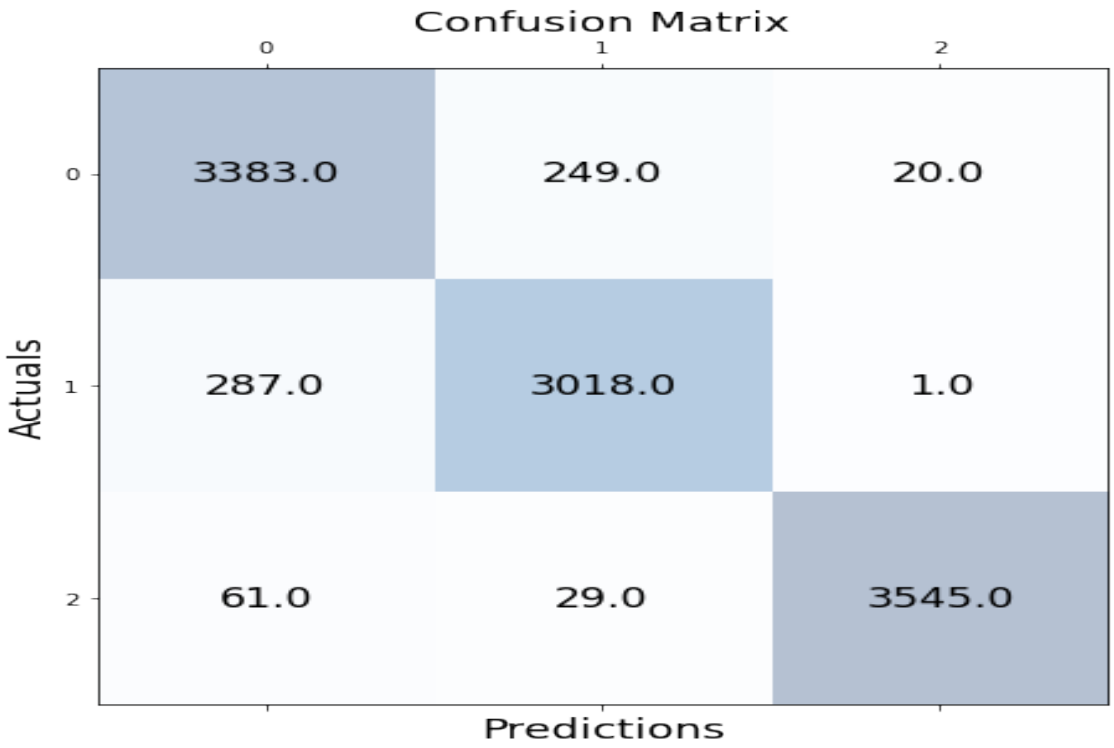

Figure 39: Confusion Matrix For SVM

8 Conclusion

The best model is Logistic regression for all performance metrics then SVM.

Classifiers	**Precision**	**Recall**	**F1score**	**Accuracy**	**AUC**
Logistic Regression	0.9502	0.9494	0.94915	0.9494	0.99
Naive Bayes	0.8541	0.8514	0.8497	0.8514	0.95
SVM	0.9386	0.9389	0.9387	0.93892	0.98

Table 1: Comparison between the selected models

9 Contribution

1. For the first part (Data Collection), we collected data from various sources

   (a) Scraping first data and preparing the data for analysis by cleaning the data, removing duplicates and labeling it.Then, Saving it in the csv file. This step is **Done By Esraa Ahmed Fouad and Mariam Reziq** .

   (b) Geting second data from third party (kaggle) and preparing the data for analysis by cleaning the data, removing duplicates. This step is **Done By Mariam Reziq and Lomai Mohamed saadeldin** .

2. For the second part (Data Preparation):
   Merging 2 data sets and preparing the data for analysis by cleaning the data, removing duplicates, Checking imbalance ,solving imbalance ,removing special characters, tokenizing , vectorizing and selecting features using Spark's built-in data processing functions, such as filtering, mapping, and aggregation. This step is **Done By Esraa Ahmed Fouad and Lomai Mohamed saadeldin** .

3. For the third part (Data Analysis)
   we used Spark's machine learning libraries (MLlib) to analyze the data and performed sentiment analysis to classify posts as positive, negative, or neutral. We built and evaluated three different models (Logistic Regression, Naive Bayes and Support Vector Machine). This step is **Done By Esraa Ahmed Fouad and Eman Elrefai** .

4. For the fourth part (Data Visualization) we used Spark's data visualization (SparkSQL) to visualize the results of your analysis. We created a histogram to visualize the distribution of sentiment before and after splitting the data and also after the oversampling. We showed the result for each model (Logistic Regression, Naive Bayes and Support Vector Machine) according to Performance Metrics, ROC/ AUC curve, Precision vs Recall curve, and Confusion Matrix. We compared them by Performance Metrics and showed the best model.This step is **Done By Eman Elrefai** .

5. For the Fifth part (Insights): we implemented some visualization in the data to Know more about the behaviour of users and what they are interested about.This step is **Done By Esraa Ahmed Fouad** .