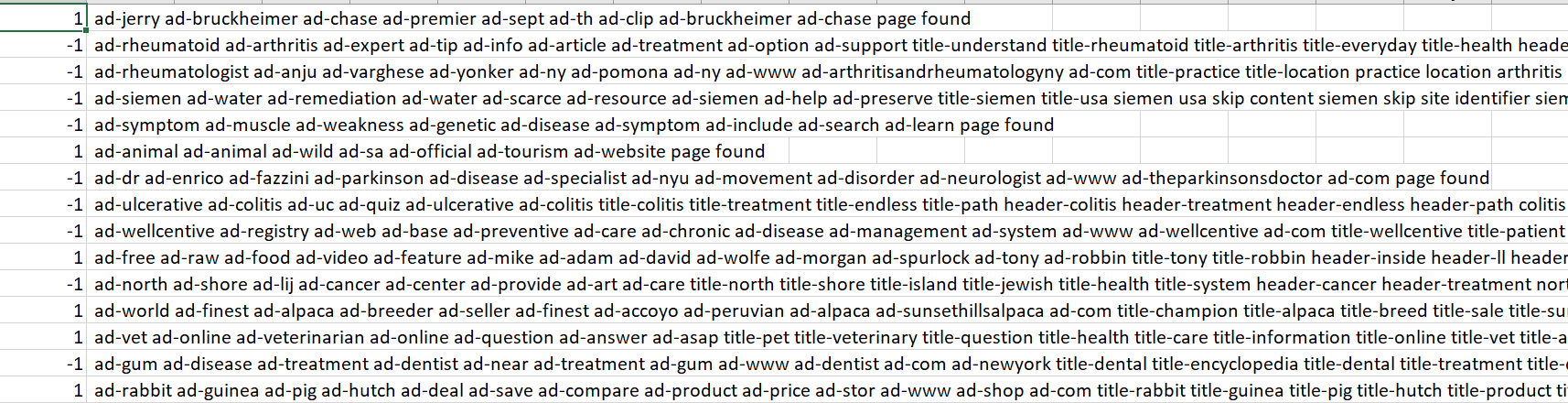
**IoT Management (ISTM\_6290\_80)**

**Text Mining with PySpark**

**Classifying Ads Submitted Online**

**• Open the file *farm-ads.csv*, and briefly review some of the relevant and non-relevant ads to get a flavor for their contents.**

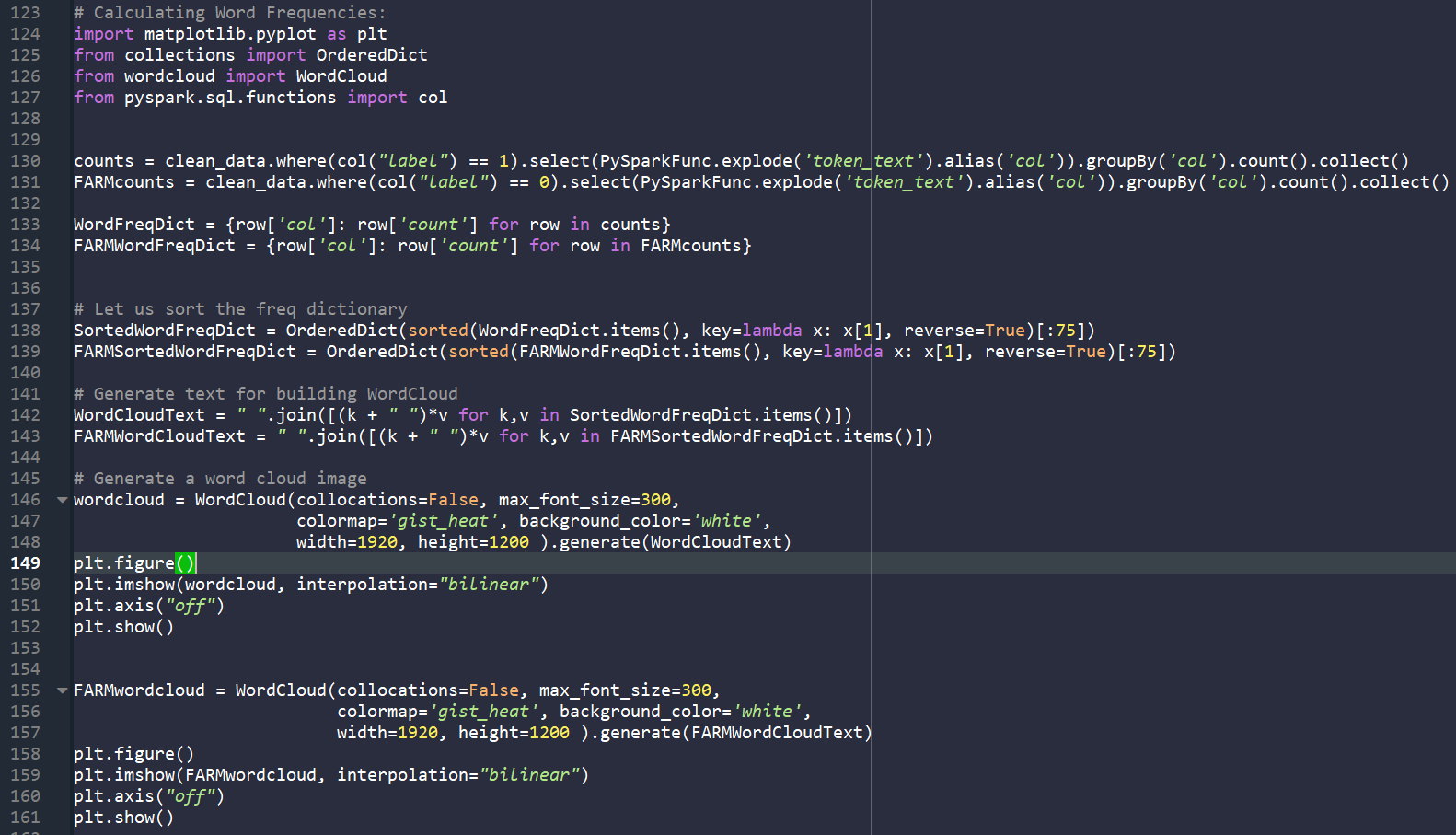


Looking at the data snippet above, we can see that some ads have been given (-1) and some have been given (1) as class, signifying the non-relevant and the relevant ads respectively.

If we see the data, we can see that the ads with words related to **animals, poultry, farming, vets, food items, produce**, etc. are all marked as **1**, meaning relevant ads and all the others are marked as -1, meaning the non- relevant ads.

In addition to the homework assignment, I wanted to see if just by looking at the data in a visual format would give some insight as to what really makes an AD relevant for the farming community. The question, I try to answer is, are there words which are more frequently used in farming ADs and not so much in spam ADs hence I tried calculating word occurrence frequency between farming and other ADs. Wrote some code and imported WordCloud library for making it visually appealing.

Here is the code snippet:



WordCloud image for the farming related Ads looks like this [NOTE: font size reflects frequency of occurrence and I took top 75 highest frequencies of occurrence]:

A screenshot of a cell phone

Description automatically generated

Clearly, prominent words are “product”, “animal”, “price”, “goat”, “rabbit”, “chicken”, “dog”, “horse”, “pig”, “veterinary”, “food”, “weight” etc etc.

WordCloud image for the other NON-farming related Ads looks like this [NOTE: font size reflects frequency of occurrence and I took top 75 highest frequencies of occurrence]:

A screenshot of a cell phone

Description automatically generated

Clearly, prominent words are “disease”, “dentist”, “treatment”, “home”, “arthiritis”, “health”, “home”, “symptom”, “pain”, “cause”, “free”, “service” etc etc.

AND, clearly words like “product”, “list”, “health” and “price” occur in both categories (farming and non-farming) and with similar frequencies, which opens the possibility for an early interpretation that any machine model, trained purely on this dataset without special provisions won’t be very accurate (maybe less than 90% accurate).

AND, from the predominant words (“treatment”, “arthritis”, “pain” etc) in the non-farming ADs, it looks like medical ADs are the ones creating more spam.

**• Following the class example, please create the TF-IDF feature matrix.**

# In[1]:

# Read Text Data

from pyspark.sql import SparkSession

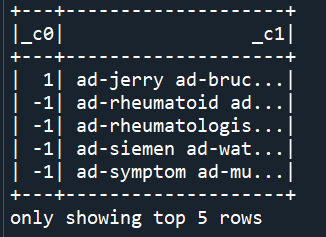
import pyspark.sql.functions as PySparkFunc

spark = SparkSession.builder.appName('text mining').getOrCreate()

data = spark.read.csv("C:/Users/anuju/desktop/Internet\_Of\_Things/Assignments/Assignment3/farm-ads.csv", inferSchema=True, sep=',')

data.show(5)

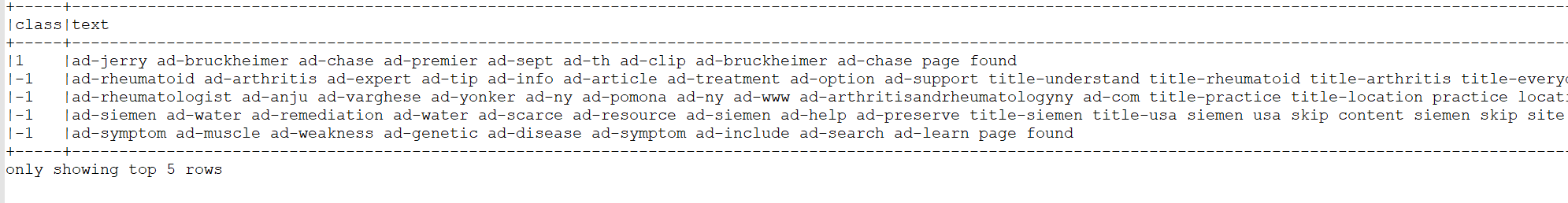
data = data.withColumnRenamed('\_c0','class').withColumnRenamed('\_c1','text')



# Removing the extra space in front of every ad

data = data.withColumn('text', (PySparkFunc.trim (PySparkFunc.col("text"))))

data.show(5,truncate=False)



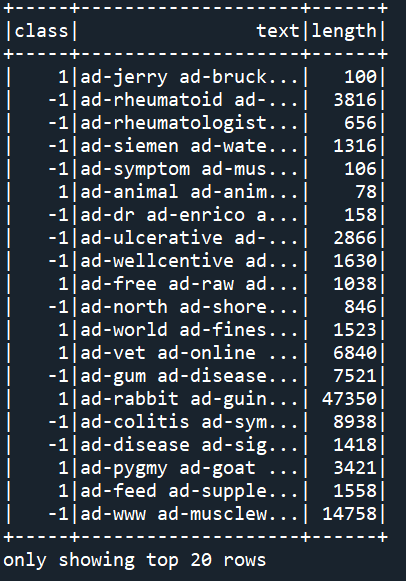
# In[2]:

# Count number of Words in each Text

from pyspark.sql.functions import length

data = data.withColumn('length', length(data['text']))

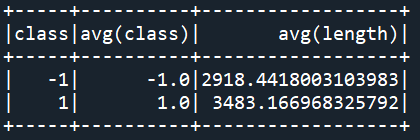
data.show()



# In[3]:

# Compare the length difference between -1(not relevant) and 1(relevant)

data.groupby('class').mean().show()



# In[4]:

# Treat TF-IDF features for each text

# TF: Term Frequency

# IDF: Inverse Document Frequency

from pyspark.ml.feature import Tokenizer, StopWordsRemover, CountVectorizer, StringIndexer, IDF, VectorAssembler

from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="text", outputCol="token\_text")

stopremove = StopWordsRemover(inputCol='token\_text',outputCol='stop\_tokens')

count\_vec = CountVectorizer(inputCol='stop\_tokens',outputCol='c\_vec')

idf = IDF(inputCol="c\_vec", outputCol="tf\_idf")

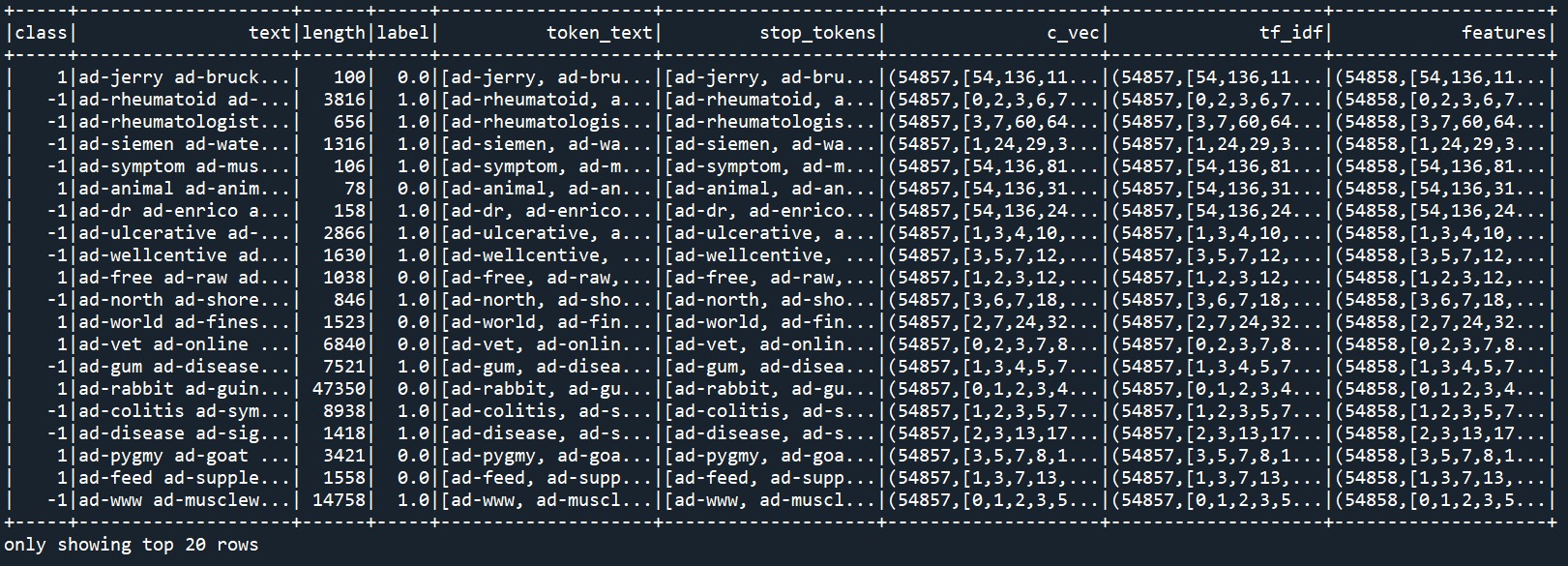
String\_Coded = StringIndexer(inputCol='class',outputCol='label')

final\_feature = VectorAssembler(inputCols=['tf\_idf', 'length'],outputCol='features')

data\_prep\_pipe = Pipeline(stages=[String\_Coded, tokenizer,stopremove,count\_vec,idf,final\_feature])

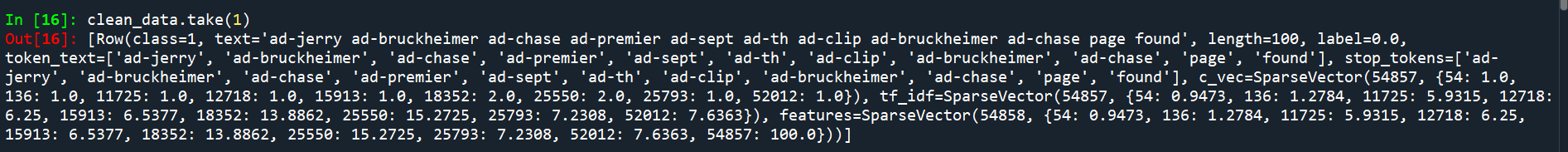
clean\_data = data\_prep\_pipe.fit(data).transform(data)

clean\_data.show()



# Selecting the first row of the data to check the elements

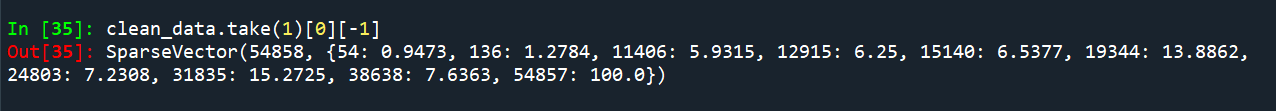
clean\_data.take(1)



**a. Examine the TF-IDF matrix.**

# Selecting the last column i.e. the Features column from the first row to examine the Tf-IDf Matrix

clean\_data.take(1)[0][-1]



1. **Is it sparse matrix or dense matrix? (Or answer How much percentage of the entries in the**

**matrix is zero?)**

**Sparse Matrices** are the matrices that contain mostly zero values, distinct from the matrices where most of the values are non-zero, called **Dense.**

The sparsity of a matrix can be quantified with a score, which is the number of zero values in the matrix divided by the total number of elements in the matrix.

**Sparsity = (1 – ) \* 100**

**Density = ( ) \* 100**

**Where, x is the matrix**

**The Tf-IDf Feature matrix created for the given dataset is a Sparse Matrix.**

If we look at the data sample above, out of 54858 features, only 10 have a certain value (non-zero), which means all the others are 0.

We can also calculate the Density and the Sparsity of the Matrix using NumPy library of Python.

**## Calculating the Density and Sparsity of the Tf-IDf Feature Matrix**

**from numpy import count\_nonzero**

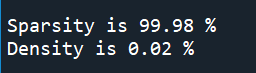
**sparsity = (1.0 - count\_nonzero(clean\_data.take(1)[0][-1]) / clean\_data.take(1)[0][-1].size)\*100**

**print ("Sparsity is" ,round( sparsity,2),"%")**

**density = (count\_nonzero(clean\_data.take(1)[0][-1]) / clean\_data.take(1)[0][-1].size)\*100**

**print ("Density is" , round(density,2),"%")**

**Results:**



**The results above clearly show that the TF-IDF Matrix is a Sparse Matrix.**

1. **Find two non-zero entries and briefly interpret their meaning, in words. (you do not need to derive their calculation)**

Mathematically, TF-IDF is expressed as:

**Wx,y = tfx,y \* log ()**

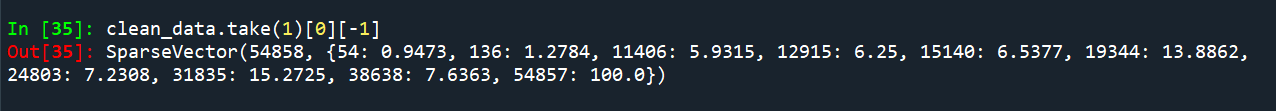
|  |  |
| --- | --- |
| **TF-IDF**  **Term x within Document y** | **tfx,y = Frequency of x in y**  **dfx = number of documents containing x**  **N = Total number of documents** |

TF-IDF therefore ensures that terms with high frequency in the document will have high TF but if a term has high frequency across the corpus then its importance is reduced by IDF.

A term present in all documents in the corpus will have TF-IDF equal to 0.

The higher the numerical weight value, the rarer the term. The smaller the weight, the more common the term.

So, in our example i.e. the TF-IDF Feature of Row 1 of our dataset,



Non-Zero entries are:

1. 54: 0.9473
2. 31835: 15.725

This means the word with feature 54 has more occurrence in our dataset than the word with feature 31835, as higher the TF-IDF score, the lower is the occurrence, and vice versa.

**Occurrence (High)**

**Occurrence (Low)**

**Rare Words**

**Frequent Words**

**Stop Words**

**TF-IDF Value (Low)**

**TF-IDF Value (High)**

**b. Using logistic regression, partition the data (60% training, 40% validation), and develop a model to classify the documents as ‘relevant’ or ‘non-relevant.’ Comment on your model’s accuracy.**

# In[5]:

# ## Split data into training and test datasets

training, test = clean\_data.randomSplit([0.6, 0.4], seed=12345)

# Build Logistic Regression Model

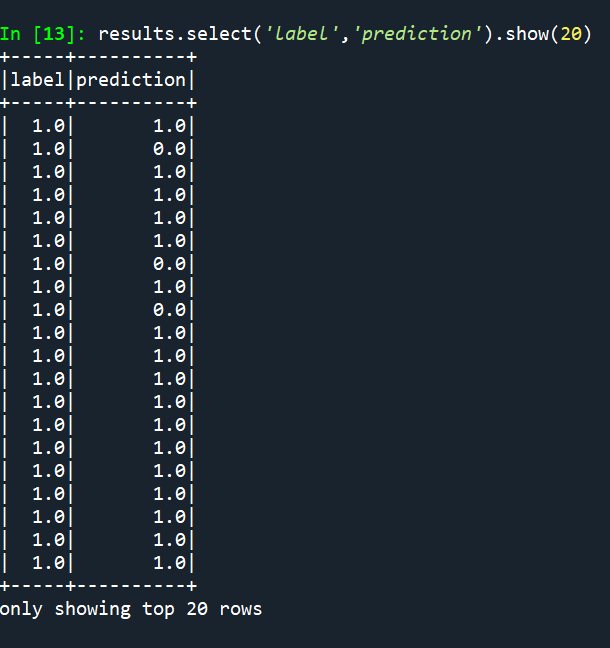
from pyspark.ml.classification import LogisticRegression

log\_reg = LogisticRegression(featuresCol='features', labelCol='label')

model = log\_reg.fit(training)

results = model.transform(test)

results.select('label','prediction').show(20)



# In[10]:

# #### CONFUSION MATRIX

from sklearn.metrics import confusion\_matrix, classification\_report

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

y\_true = results.select("label")

y\_true = y\_true.toPandas()

y\_pred = results.select("prediction")

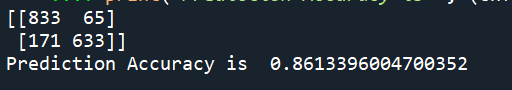
y\_pred = y\_pred.toPandas()

cnf\_matrix = confusion\_matrix(y\_true, y\_pred)

print(cnf\_matrix)

print("Prediction Accuracy is ", (cnf\_matrix[0,0]+cnf\_matrix[1,1])/sum(sum(cnf\_matrix)) )

**RESULTS:**



# PLOT CONFUSION MATRIX

fig, ax = plt.subplots(1)

ax.xaxis.set\_label\_position('top')

ax.xaxis.tick\_top()

sns.heatmap(cnf\_matrix, annot=True, fmt='d', ax=ax)

plt.xlabel("PREDICTED")

plt.ylabel("ACTUAL")

A screenshot of a cell phone

Description automatically generated

In the above figure,

**Class 0- Relevant Ads**

**Class 1- Non-Relevant Ads**

**True Positives(TP)**, Instances which are positive and are predicted positive are 833

**True Negatives(TN),** Instances which are negative and are predicted as negative are 633

**False Positives (FP),** Instances which are negative, but predicted as positive are 171

**False Negatives (FN),** Instances which are positive, but predicted as negative are 65

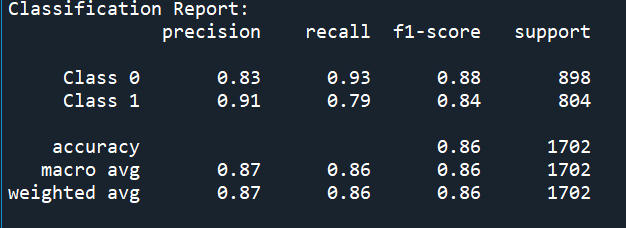
# In[11]:

# GETTING THE CLASSIFICATION REPORT

target\_names = ['Class 0' , 'Class 1']

print ('Classification Report:')

print(classification\_report(y\_true, y\_pred, target\_names = target\_names))



**Precision** is the ability of a classifier not to label an instance positive that is actually negative. For each class it is defined as the ratio of true positives to the sum of true and false positives.

**Precision = TP/ (TP+FP)**

**Recall** is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

**Recall = TP/(TP+FN)**

**F1 score** is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.

**F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)**

If we look at the Recall values, We can see that the ability of our classifier to find all the Relevant (Class 0) ads is 93%, where as the ability to find the Non-Relevant (Class 1) ads is 79%.

Precision values also give an insight into the classifier’s evaluation metrics, as it is 83% for the relevant ads and 91% for the non-relevant ads, which means our model can more precisely label the non-relevant ads in comparison to the relevant ads.

# In[12]:

# Model Evaluation

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Select (prediction, true label) and compute test error

evaluator = MulticlassClassificationEvaluator (labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(results)

print ("Test Accuracy = %g" %accuracy)

print("Test Error = %g" % (1.0 - accuracy))

**RESULTS:**



The model is 86.13% Accurate on testing data.

As it is believed that more the training data set, better the model learns. Hence better the accuracy.

I tried to partition the data in different ways and check the accuracy of the model.

Let us have a look.

1. 70% Training and 30% Test data.
2. 80% Training and 20% Test data
3. 90% Training and 10% Test data
4. 95% Training and 5% Test data

Following were the Accuracy Results:

|  |  |  |
| --- | --- | --- |
| DATA PARTITIONING | |  |
| TRAINING | **TEST** | **ACCURACY** |
| 60% | 40% | 86.13% |
| 70% | 30% | 87.04% |
| 80% | 20% | 87.85% |
| 90% | 10% | 85.39% |
| 95% | 5% | 81.11% |

The Test Accuracy did not improve considerably, rather decreased after a substantial amount of data was put into training data.

There can be many reasons for the less accuracy of the model:

1. IDF Is a kind of weighting that tries to suppress noise, and it tends to be a word with a relatively small frequency in the text, which makes IDF. So, the accuracy is not high.
2. TF-IDF can “stretch” the word count as well as “compress” it. In other words, it makes some counts bigger, and others close to zero. Therefore, TF-IDF could altogether eliminate uninformative words.
3. Right Feature Scaling can be helpful for an accurate model, as that accentuates the informative words and down weights the common words. But right scaling is not necessarily uniform column scaling.
4. TF-IDF is severely dependent on the corpus (especially when training similar corpus, it often conceals some keywords of the same type. For Example, In the above TF-IDF, during training, there are more doctor and diseases advertisements(non-relevant) in the corpus, and the weight of keywords related to doctor will be lower. Therefore, it is necessary to select a high-quality corpus for training, as Veterinary doctor, and animal diseases ads may be removed and marked as non-relevant.