### Import

%pyspark

from pyspark.ml.recommendation import ALS

from pyspark.ml.feature import \*

from pyspark.ml.classification import \*

from pyspark.ml.evaluation import \*

from pyspark.sql.functions import \*

from pyspark.sql.types import \*

import nltk

from nltk.stem.wordnet import WordNetLemmatizer

from pyspark.ml import Pipeline

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

from nltk.sentiment.vader import SentimentIntensityAnalyzer

### Read CSV

%pyspark

df = spark.read.csv("train\_lyrics\_1000.csv", header='true', multiLine='true', escape="\"")

df.na.drop()

df.count()

### Selecting Required Columns

%pyspark

df = df['lyrics','mood']

df.printSchema()

### Clean Text

%pyspark

def clean\_text(c):

c = lower(c)

c = regexp\_replace(c, "(RT |rt )\S+", "")

c = regexp\_replace(c, "(https:\/\/|http:\/\/)\S+", "")

c = regexp\_replace(c, "[^a-zA-Z\\s]", "")

c = regexp\_replace(c, "[\\s+]", " ")

c = regexp\_replace(c, "[\\s][\\s]", " ")

return c

clean\_text\_df = df.select(clean\_text(col("lyrics")).alias("words"), col("mood"))

clean\_text\_df.show()

### Tokenize

%pyspark

tokenizer = Tokenizer(inputCol="words", outputCol="tokens")

vector\_df = tokenizer.transform(clean\_text\_df).select("tokens", "mood")

vector\_df.show()

### Stop Words Removal

%pyspark

# Define a list of stop words or use default list

remover = StopWordsRemover()

# Specify input/output columns

remover.setInputCol("tokens")

remover.setOutputCol("vector\_no\_stopw")

# Transform existing dataframe with the StopWordsRemover

vector\_no\_stopw\_df = remover.transform(vector\_df).select("vector\_no\_stopw", "mood")

vector\_no\_stopw\_df.printSchema()

### Lemmatization

%pyspark

#nltk.download('all')

#nltk.download('averaged\_perceptron\_tagger')

lemmatizer = WordNetLemmatizer()

def lemmatize(in\_vec):

out\_vec = []

for t in in\_vec:

t\_lemmatize = lemmatizer.lemmatize(t)

out\_vec.append(t\_lemmatize)

return out\_vec

lemmatizer\_udf = udf(lambda x: lemmatize(x), ArrayType(StringType()))

# Create new df with vectors containing the stemmed tokens

vector\_lemmatizer\_df = (

vector\_no\_stopw\_df

.withColumn("vector\_lemmatized", lemmatizer\_udf("vector\_no\_stopw"))

.withColumn("mood", vector\_no\_stopw\_df['mood'])

.select("vector\_lemmatized", "mood")

)

# Rename df and column for clarity

production\_df = vector\_lemmatizer\_df.select(col("vector\_lemmatized").alias("unigrams"), col("mood"))

production\_df.show()

### TF IDF

%pyspark

cvTF = CountVectorizer(inputCol="unigrams", outputCol="rawfeatures")

TFmodel = cvTF.fit(production\_df)

production\_df = TFmodel.transform(production\_df)

idf = IDF(inputCol="rawfeatures", outputCol="features1")

idfModel = idf.fit(production\_df)

production\_df = idfModel.transform(production\_df)

production\_df = production\_df.select(col("unigrams"),col("mood"),col("features1"))

production\_df.show()

### Polarity Assignment Of Lyrics

%pyspark

sid = SentimentIntensityAnalyzer()

def polarity\_assignment(in\_vec):

no\_of\_words = 0

polarity\_value = 0.0

for t in in\_vec:

ss = sid.polarity\_scores(t)

polarity\_value+= ss["compound"]

no\_of\_words +=1

return polarity\_value/no\_of\_words

polarity\_assignment\_udf = udf(lambda x: polarity\_assignment(x), DoubleType())

# Create new df with vectors containing the stemmed tokens

polarity\_assignment\_df = (

production\_df

.withColumn("polarity\_value", polarity\_assignment\_udf("unigrams"))

)

production\_df1 = polarity\_assignment\_df.select(col("unigrams"),col("features1"),col("polarity\_value"), col("mood"))

production\_df1.show()

### Label mood column

%pyspark

#String Indexer for Feature2

si = StringIndexer(inputCol="mood", outputCol="label")

si\_model = si.fit(production\_df1)

production\_df2 = si\_model.transform(production\_df1)

production\_df3 = production\_df2.select(col("unigrams"),col("features1"),col("polarity\_value"),col("mood"),col("label"))

production\_df3.show()

### Creating Feature Vector

%pyspark

assembler = VectorAssembler(

inputCols=["polarity\_value","features1"],

outputCol="features")

production\_df4 = assembler.transform(production\_df3).select(col("unigrams"),col("polarity\_value"), col("features"), col("mood"), col("label"))

production\_df4.show()

### Train Test Data Split

%pyspark

# Split the data into training and test sets (30% held out for testing)

(trainingData1, testData1) = production\_df4.randomSplit([0.7, 0.3])

trainingData1.show()

### Feature and Label Extraction from Training and Test Dataset

%pyspark

trainingData = trainingData1.select(col("features"),col("label"))

testData = testData1.select(col("features"),col("label"))

### Gradient Boosted Tree(MulticlassClassificationEvaluator)

%pyspark

gbt = GBTClassifier(maxIter=10, labelCol="label", featuresCol="features")

pipeline = Pipeline(stages = [gbt])

paramGrid = ParamGridBuilder().addGrid(gbt.maxIter, [10]).build()

evaluator = MulticlassClassificationEvaluator(

labelCol="label", predictionCol="prediction", metricName="accuracy")

crossval = CrossValidator(estimator = pipeline,

estimatorParamMaps = paramGrid,

evaluator = evaluator,

numFolds = 10)

%pyspark

cvModel = crossval.fit(trainingData)

%pyspark

predictions = cvModel.transform(testData)

%pyspark

evaluator = MulticlassClassificationEvaluator(

labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print(accuracy)

### SVM(MulticlassClassificationEvaluator)

%pyspark

svm = LinearSVC(maxIter=10, regParam=0.01, featuresCol="features", labelCol="label")

pipeline = Pipeline(stages = [svm])

paramGrid = ParamGridBuilder().addGrid(svm.maxIter, [10]).build()

evaluator = MulticlassClassificationEvaluator(

labelCol="label", predictionCol="prediction", metricName="accuracy")

crossval1 = CrossValidator(estimator = pipeline,

estimatorParamMaps = paramGrid,

evaluator = evaluator,

numFolds = 10)

%pyspark

cvmodel1 = crossval1.fit(trainingData)

%pyspark

predictions1 = cvmodel1.transform(testData)

%pyspark

evaluator = MulticlassClassificationEvaluator(

labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions1)

print(accuracy)

### Saving SVM(MulticlassClassificationEvaluator)

%pyspark

cvmodel1.bestModel.stages[0].write().overwrite().save("SVM\_model")

### Gradient Boosted Tree(BinaryClassificationEvaluator)

%pyspark

gbt = GBTClassifier(maxIter=10, labelCol="label", featuresCol="features")

pipeline = Pipeline(stages = [gbt])

paramGrid = ParamGridBuilder().addGrid(gbt.maxIter, [10]).build()

crossval2 = CrossValidator(estimator = pipeline,

estimatorParamMaps = paramGrid,

evaluator = BinaryClassificationEvaluator(),

numFolds = 10)

%pyspark

cvModel2 = crossval2.fit(trainingData)

%pyspark

predictions2 = cvModel2.transform(testData)

### Extracting Predicted Dataset for Visualisation

%pyspark

visualisation\_data = predictions2.join(testData1,predictions2.features==testData1.features)

visualisation\_data = visualisation\_data.select(col("unigrams"),col("polarity\_value"),col("mood"),col("prediction"))

visualisation\_data.coalesce(1).write.format('json').save('visualisation\_data1.json')

%pyspark

evaluator = BinaryClassificationEvaluator()

accuracy = evaluator.evaluate(predictions2)

print(accuracy)

### SVM(BinaryClassificationEvaluator)

%pyspark

svm = LinearSVC(maxIter=10, regParam=0.01, featuresCol="features", labelCol="label")

pipeline = Pipeline(stages = [svm])

paramGrid = ParamGridBuilder().addGrid(svm.maxIter, [10]).build()

crossval3 = CrossValidator(estimator = pipeline,

estimatorParamMaps = paramGrid,

evaluator = BinaryClassificationEvaluator(),

numFolds = 10)

%pyspark

cvmodel3 = crossval3.fit(trainingData)

%pyspark

predictions3 = cvmodel3.transform(testData)

%pyspark

evaluator = BinaryClassificationEvaluator()

accuracy = evaluator.evaluate(predictions3)

print(accuracy)