%pyspark

from pyspark.sql.functions import col, lower, regexp\_replace, split

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 nltk.stem.porter import \*

from pyspark.sql.functions import col, lower, regexp\_replace, split

from pyspark.sql.types import \*

from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel

from pyspark.mllib.regression import LabeledPoint

def clean\_text\_words(c):

c = lower(c)

c = regexp\_replace(c, "[^a-zA-Z0-9\\s]", " ")

return c

df = spark.read.csv("senticnet4-comma.csv").select(clean\_text\_words(col("\_c0")).alias("words"),col("\_c2").alias("rank"))

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

tokenized\_df = tokenizer\_words.transform(df).select("tokens", "rank")

remover\_words = StopWordsRemover()

remover\_words.setInputCol("tokens")

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

removed\_df = remover\_words.transform(tokenized\_df).select("vector\_no\_stopw", "rank")

%pyspark

stemmer = PorterStemmer()

def stem\_words(in\_vec):

out\_vec = []

for t in in\_vec:

t\_stem = stemmer.stem(t)

if len(t\_stem) > 2:

out\_vec.append(t\_stem)

return out\_vec

stemmer\_udf\_words = udf(lambda x: stem\_words(x), ArrayType(StringType()))

# Create new df with vectors containing the stemmed tokens

stemmed\_df = (

removed\_df

.withColumn("vector\_stemmed", stemmer\_udf\_words("vector\_no\_stopw"))

.withColumn("rank", removed\_df['rank'])

.select("vector\_stemmed", "rank")

)

stemmed\_df = stemmed\_df.where(size(col("vector\_stemmed")) > 1)

stemmed\_df.show()

%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 nltk.stem.porter import \*

from pyspark.sql.functions import col, lower, regexp\_replace, split

from pyspark.sql.types import \*

from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel

from pyspark.mllib.regression import LabeledPoint

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

df.na.drop()

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

def clean\_text(c):

c = lower(c)

c = regexp\_replace(c, "^rt ", "")

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

c = regexp\_replace(c, "[^a-zA-Z0-9\\s]", "")

return c

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

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

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

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")

stemmer = PorterStemmer()

def stem(in\_vec):

out\_vec = []

for t in in\_vec:

t\_stem = stemmer.stem(t)

if len(t\_stem) > 2:

out\_vec.append(t\_stem)

return out\_vec

stemmer\_udf = udf(lambda x: stem(x), ArrayType(StringType()))

# Create new df with vectors containing the stemmed tokens

vector\_stemmed\_df = (

vector\_no\_stopw\_df

.withColumn("vector\_stemmed", stemmer\_udf("vector\_no\_stopw"))

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

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

)

# Rename df and column for clarity

production\_df1 = vector\_stemmed\_df.select(col("vector\_stemmed").alias("unigrams"), col("mood"))

production\_df1.show()

%pyspark

stemmed\_df = stemmed\_df

joined = production\_df1.crossJoin(stemmed\_df)

joined = joined.na.drop()

joined\_select = joined.select("unigrams", "vector\_stemmed", "rank")

def calculate\_rank(x,y,z):

if len(list(set(y)-set(x))) == 0:

return float(z)

else:

return 0.0

differencer=udf(lambda x,y,z: calculate\_rank(x,y,z), FloatType())

diff\_table = joined\_select.limit(1000000).withColumn('difference', differencer('unigrams', 'vector\_stemmed', 'rank'))

diff\_table = diff\_table.na.drop()

diff\_table = diff\_table.filter(diff\_table['difference'] != 0.0)

diff\_table.show()

%pyspark

group\_table = diff\_table.select("unigrams","difference").groupBy("unigrams").sum("difference")

group\_table.show()

%pyspark

complete\_table = production\_df1.join(group\_table,"unigrams")

complete\_table = complete\_table.select(col("sum(difference)").alias("sumDiff"), col("mood"))

#complete\_table = complete\_table.filter(complete\_table['sumDiff'] > 1)

complete\_table.show()

%pyspark

from pyspark.ml.clustering import KMeans

from pyspark.ml.feature import VectorAssembler

vecAssembler = VectorAssembler(inputCols= ["sumDiff"], outputCol="features")

vector\_df = vecAssembler.transform(complete\_table)

#kmeans clustering

kmeans=KMeans(k=2, seed=1)

model=kmeans.fit(vector\_df)

predictions=model.transform(vector\_df)

predictions.show()

%pyspark

from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol="mood", outputCol="mood\_binary")

indexed = indexer.fit(predictions).transform(predictions)

indexed = indexed.select(col("sumDiff"), col("mood"), col("features"),col("prediction").alias("predict").cast(DoubleType()), col("mood\_binary").cast(IntegerType()))

indexed.show()

%pyspark

correct\_rows = indexed.filter(indexed['predict'] == indexed['mood\_binary'])

correct = correct\_rows.count()

print(correct)

total = indexed.count()

print(total)

(trainingData, testData) = indexed.randomSplit([0.7, 0.3])

svm = LinearSVC(maxIter=10, regParam=0.01, featuresCol="features", labelCol="mood\_binary")

svm\_model = svm.fit(trainingData)

svm\_predictions = svm\_model.transform(testData)

evaluator = MulticlassClassificationEvaluator(

labelCol="mood\_binary", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(svm\_predictions)

print(accuracy)