### 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 nltk.stem.porter import \*

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

from pyspark.sql.types import \*

from pyspark.ml import Pipeline

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

### Read CSV

%pyspark

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

df.na.drop()

df.count()

### Chop off Unimportant columns

%pyspark

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

df.printSchema()

### Clean Text

%pyspark

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"))

### Tokenize

%pyspark

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

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

### 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")

### Stemming

%pyspark

# Instantiate stemmer object

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"))

### N-Grams

%pyspark

# Define NGram transformer

ngram = NGram(n=2, inputCol="unigrams", outputCol="bigrams")

# Create bigram\_df as a transform of unigram\_df using NGram tranformer

production\_df2 = ngram.transform(production\_df1)

### Filter out empty/Len < 2 records

%pyspark

production\_df2 = production\_df2.where(size(col("bigrams")) >= 2)

production\_df2.na.drop()

production\_df2.count()

### Label mood column

%pyspark

#String Indexer

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

si\_model = si.fit(production\_df2)

production\_df3 = si\_model.transform(production\_df2)

### Count Vectorizer

%pyspark

cv\_1 = CountVectorizer(inputCol="unigrams", outputCol="feature\_1")

cv\_2 = CountVectorizer(inputCol="bigrams", outputCol="feature\_2")

### Vector Assembler

%pyspark

va = VectorAssembler(inputCols=["feature\_1", "feature\_2"], outputCol="features")

### TFIDF

%pyspark

hashingTF\_1 = HashingTF(inputCol="unigrams", outputCol="feature\_1", numFeatures=500)

#featurizedData\_1 = hashingTF.transform(production\_df3)

hashingTF\_2 = HashingTF(inputCol="bigrams", outputCol="feature\_2", numFeatures=500)

#featurizedData\_2 = hashingTF.transform(featurizedData\_1)

%pyspark

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

(trainingData, testData) = production\_df5.randomSplit([0.7, 0.3])

trainingData.show()

### Naive Bayes Pipeline

%pyspark

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

(trainingData, testData) = production\_df3.randomSplit([0.7, 0.3])

nb1 = NaiveBayes(smoothing=1.0, modelType="multinomial", featuresCol="feature\_1")

#Pipeline 1

pipeline\_cv\_nb\_f1 = Pipeline(stages=[cv\_1, cv\_2, nb1])

m1 = pipeline\_cv\_nb\_f1.fit(trainingData)

pr1 = m1.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr1)))

#Pipeline 2

pipeline\_tf\_nb\_f1 = Pipeline(stages=[hashingTF\_1, hashingTF\_2, nb1])

m2 = pipeline\_tf\_nb\_f1.fit(trainingData)

pr2 = m2.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr2)))

##############################################################################################################

nb2 = NaiveBayes(smoothing=1.0, modelType="multinomial", featuresCol="feature\_2")

#Pipeline 3

pipeline\_cv\_nb\_f2 = Pipeline(stages=[cv\_1, cv\_2, nb2])

m3 = pipeline\_cv\_nb\_f2.fit(trainingData)

pr3 = m3.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr3)))

#Pipeline 4

pipeline\_tf\_nb\_f2 = Pipeline(stages=[hashingTF\_1, hashingTF\_2, nb2])

m4 = pipeline\_tf\_nb\_f2.fit(trainingData)

pr4 = m4.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr4)))

##############################################################################################################

nb3 = NaiveBayes(smoothing=1.0, modelType="multinomial", featuresCol="features")

#Pipeline 5

pipeline\_cv\_nb\_f12 = Pipeline(stages=[cv\_1, cv\_2, va, nb3])

m5 = pipeline\_cv\_nb\_f12.fit(trainingData)

pr5 = m5.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr5)))

#Pipeline 6

pipeline\_tf\_nb\_f12 = Pipeline(stages=[hashingTF\_1, hashingTF\_2, va, nb3])

m6 = pipeline\_tf\_nb\_f12.fit(trainingData)

pr6 = m6.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr6)))

Test set accuracy = 0.498170843241

Test set accuracy = 0.482760197549

Test set accuracy = 0.419494238156

Test set accuracy = 0.476038046461

Test set accuracy = 0.454271081032

Test set accuracy = 0.477547100787

### SVM Pipeline

%pyspark

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

(trainingData, testData) = production\_df3.randomSplit([0.7, 0.3])

svm1 = LinearSVC(maxIter=10, regParam=0.01, featuresCol="feature\_1", labelCol="label")

#Pipeline 1

pipeline\_cv\_svm\_f1 = Pipeline(stages=[cv\_1, cv\_2, svm1])

m1 = pipeline\_cv\_svm\_f1.fit(trainingData)

pr1 = m1.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr1)))

#Pipeline 2

pipeline\_tf\_svm\_f1 = Pipeline(stages=[hashingTF\_1, hashingTF\_2, svm1])

m2 = pipeline\_tf\_svm\_f1.fit(trainingData)

pr2 = m2.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr2)))

##############################################################################################################

svm2 = LinearSVC(maxIter=10, regParam=0.01, featuresCol="feature\_2", labelCol="label")

#Pipeline 3

pipeline\_cv\_svm\_f2 = Pipeline(stages=[cv\_1, cv\_2, svm2])

m3 = pipeline\_cv\_svm\_f2.fit(trainingData)

pr3 = m3.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr3)))

#Pipeline 4

pipeline\_tf\_svm\_f2 = Pipeline(stages=[hashingTF\_1, hashingTF\_2, svm2])

m4 = pipeline\_tf\_svm\_f2.fit(trainingData)

pr4 = m4.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr4)))

##############################################################################################################

svm3 = LinearSVC(maxIter=10, regParam=0.01, featuresCol="features", labelCol="label")

#Pipeline 5

pipeline\_cv\_svm\_f12 = Pipeline(stages=[cv\_1, cv\_2, va, svm3])

m5 = pipeline\_cv\_svm\_f12.fit(trainingData)

pr5 = m5.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr5)))

#Pipeline 6

pipeline\_tf\_svm\_f12 = Pipeline(stages=[hashingTF\_1, hashingTF\_2, va, svm3])

m6 = pipeline\_tf\_svm\_f12.fit(trainingData)

pr6 = m6.transform(testData)

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr6)))

Test set accuracy = 0.650207390649

Test set accuracy = 0.634860482655

Test set accuracy = 0.56427224736

Test set accuracy = 0.452658371041

Test set accuracy = 0.630788084465

Test set accuracy = 0.59217571644

### SVM Hyperparameter Tuning

%pyspark

(trainingData, testData) = production\_df3.randomSplit([0.7, 0.3])

svm1 = LinearSVC(maxIter=10, regParam=0.01, featuresCol="feature\_1", labelCol="label")

#Pipeline 1

pipeline\_cv\_svm\_f1 = Pipeline(stages=[cv\_1, cv\_2, svm1])

m1 = pipeline\_cv\_svm\_f1.fit(trainingData)

testData = testData.drop("label")

pr1 = m1.transform(testData)

pr1.show()

evaluator = BinaryClassificationEvaluator()

print("Test set accuracy = " + str(evaluator.evaluate(pr1)))

### Cross Validator

%pyspark

crossval = CrossValidator(estimator=pipeline\_cv\_svm\_f1,

estimatorParamMaps=paramGrid1,

evaluator=BinaryClassificationEvaluator(),

numFolds=8)

cvModel = crossval.fit(trainingData)

prediction = cvModel.transform(test)

%pyspark

(trainingData, testData) = production\_df3.randomSplit([0.7, 0.3])

svm1 = LinearSVC(maxIter=10, regParam=0.01, featuresCol="feature\_1", labelCol="label")

#Pipeline 1

pipeline\_cv\_svm\_f1 = Pipeline(stages=[cv\_1, cv\_2, svm1])

paramGrid1 = ParamGridBuilder().addGrid(svm1.maxIter, [10]).build()

crossval = CrossValidator(estimator=pipeline\_cv\_svm\_f1,

estimatorParamMaps=paramGrid1,

evaluator=BinaryClassificationEvaluator(),

numFolds=8)

cvModel = crossval.fit(trainingData)

prediction = cvModel.transform(testData)

print(evaluator.evaluate(prediction))

### SVMwithSGD

%pyspark

from pyspark import SparkContext

from pyspark.mllib.classification import SVMWithSGD, SVMModel

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib import linalg as mllib\_linalg

from pyspark.ml import linalg as ml\_linalg

def as\_old(v):

if isinstance(v, ml\_linalg.SparseVector):

return mllib\_linalg.SparseVector(v.size, v.indices, v.values)

if isinstance(v, ml\_linalg.DenseVector):

return mllib\_linalg.DenseVector(v.values)

raise ValueError("Unsupported type {0}".format(type(v)))

#df3 = production\_df3.rdd.map(tuple)

def parsePoint(line):

values = [float(x) for x in line.split(' ')]

return LabeledPoint(values[0], values[1:])

parsedData = production\_df3.rdd.map(parsePoint)

model = SVMWithSGD.train(parsedData, iterations=100)