# Jeremey_code

July 5, 2021

## 1 DS 5110 Group Project

Team: Alexandra Cathcart (adc6fs), Benjamin Feciura (bmf3bw), Jeremey Donovan (jdd5dw), Jordan Hiatt (jdh2e)

Original data: https://www.kaggle.com/reddit/reddit-comments-may-2015

### 1.1 Includes & Spark Setup

[36]:
```
import pandas as pd

from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import *

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, countDistinct, udf
from pyspark.sql.types import ArrayType, IntegerType,  StringType

spark = SparkSession.builder.getOrCreate()
```

### 1.2 Read & Prepare Data

[37]:
```
full_path = '/project/ds5559/r-slash-group8/sample.csv'

df = spark.read.csv(full_path,  inferSchema=True, header = True)
```

[38]:
```
# Drop unneeded cols from dataframe
df=df.
 ↪drop('_c0','created_utc','subreddit_id','link_id','name','score_hidden','author_flair_css_c
 ↪\
        'author_flair_text','id','archived','retrieved_on',␣
 ↪'edited','controversiality','parent_id','score')

# convert integer cols (ups, downs, and gilded) to integers
# Note: we could have done this by defining a schema before the csv read
df=df.withColumn("ups",df.ups.cast(IntegerType()))
df=df.withColumn("downs",df.downs.cast(IntegerType()))
```

```
df=df.withColumn("gilded",df.gilded.cast(IntegerType()))

# Confirm new schema
df.printSchema()
df.show(5)
```

```
root
 |-- ups: integer (nullable = true)
 |-- subreddit: string (nullable = true)
 |-- removal_reason: string (nullable = true)
 |-- gilded: integer (nullable = true)
 |-- downs: integer (nullable = true)
 |-- author: string (nullable = true)
 |-- body: string (nullable = true)
 |-- distinguished: string (nullable = true)

+----+---------+--------------+------+-----+------------+--------------------+--
-----------+
| ups|subreddit|removal_reason|gilded|downs|      author|
body|distinguished|
+----+---------+--------------+------+-----+------------+--------------------+--
-----------+
|   4|soccer_jp|            NA|     0|    0|       rx109|                  |
null|
|null|     null|          null|  null| null|        null|                null|
null|
|   0|     null|          null|  null| null|        null|                null|
null|
|   4|      nba|            NA|     0|    0|   WyaOfWade|gg this one's ove…|
NA|
|   0| politics|            NA|     0|    0|Wicked_Truth|Are you really im…|
NA|
+----+---------+--------------+------+-----+------------+--------------------+--
-----------+
only showing top 5 rows
```

[39]:
```
# Count the number of rows before removing NA
df.count()
# There are 15,317,725 rows
```

[39]:
```
15317725
```

[40]:
```
# Remove rows where up, down, or body is null. We  do this since inference of␣
↪these values is not applicable
df=df.filter(df['ups'].isNotNull())
df=df.filter(df['downs'].isNotNull())
```

```
df=df.filter(df['body'].isNotNull())

df.show(5)
```

```
+---+---------+--------------+------+-----+--------------+--------------------+-
------------+
|ups|subreddit|removal_reason|gilded|downs|        author|
body|distinguished|
+---+---------+--------------+------+-----+--------------+--------------------+-
------------+
|  4|soccer_jp|            NA|     0|    0|         rx109|                  |
null|
|  4|      nba|            NA|     0|    0|     WyaOfWade|gg this one's ove…|
NA|
|  0| politics|            NA|     0|    0|   Wicked_Truth|Are you really im…|
NA|
|  3|AskReddit|            NA|     0|    0|      jesse9o3|No one has a Euro…|
NA|
|  3|AskReddit|            NA|     0|    0|beltfedshooter|"That the kid ""…|
NA|
+---+---------+--------------+------+-----+--------------+--------------------+-
------------+
only showing top 5 rows
```

[41]:
```
# Remove rows where the author was '[deleted]'
df=df.filter(df['author']!='[deleted]')

# Remove author "0"
df=df.filter(df['author']!='0')



# Remove rows where the author was 'AutoModerator'
# see https://www.reddit.com/wiki/automoderator
df=df.filter(df['author']!='AutoModerator')
```

[42]:
```
# Count the number of rows AFTER removing NA
df.count()
# There now 9,229,025 rows
```

[42]:
```
9226090
```

[43]:
```
# Even though we dropped the column, adding score back into dataframe by␣
 ↪computing it
df=df.withColumn('score',df['ups']-df['downs'])
df=df.withColumn("score",df.score.cast(IntegerType()))
df.show(5)
```

```
+---+---------+--------------+------+-----+--------------+--------------------+-
------------+-----+
|ups|subreddit|removal_reason|gilded|downs|        author|
body|distinguished|score|
+---+---------+--------------+------+-----+--------------+--------------------+-
------------+-----+
|  4|soccer_jp|            NA|     0|    0|         rx109|                    |
null|    4|
|  4|      nba|            NA|     0|    0|     WyaOfWade|gg this one's ove…|
NA|    4|
|  0| politics|            NA|     0|    0|  Wicked_Truth|Are you really im…|
NA|    0|
|  3|AskReddit|            NA|     0|    0|      jesse9o3|No one has a Euro…|
NA|    3|
|  3|AskReddit|            NA|     0|    0|beltfedshooter|"That the kid ""…|
NA|    3|
+---+---------+--------------+------+-----+--------------+--------------------+-
------------+-----+
only showing top 5 rows
```

## 1.3 EDA

[44]:
```
# How many authors are there?
df.select(countDistinct('author')).show()
# There are 1,237,196 authors
```

```
+----------------------+
|count(DISTINCT author)|
+----------------------+
|               1234824|
+----------------------+
```

[45]:
```
# Show the top 10 authors with sum of ups and downs
df.groupby('author').agg({"author":"count","ups":"sum","downs":"sum","score":
↪"sum"}).sort(col('count(author)').desc()).show(10)
```

```
+-------------------+----------+----------+-------------+--------+
|             author|sum(score)|sum(downs)|count(author)|sum(ups)|
+-------------------+----------+----------+-------------+--------+
|       TheNitromeFan|     10445|         0|         3997|   10445|
|         TweetPoster|      7090|         0|         3589|    7090|
|         autowikibot|      6420|         0|         3210|    6420|
|          PoliticBot|      3159|         0|         3142|    3159|
|TweetsInCommentsBot|      9965|         0|         3130|    9965|
|      atomicimploder|      7363|         0|         2616|    7363|
|        Removedpixel|      5333|         0|         2265|    5333|
|          TrollaBot|      2640|         0|          2247|    2640|
|           havoc_bot|      2120|         0|          2102|    2120|
|      MTGCardFetcher|      3089|         0|          2084|    3089|
+--------------------+----------+----------+--------------+--------+
only showing top 10 rows
```

Odd that the preceding authors have no down but this is correct

[46]:
```
# Show authors with the lowest scores
df.groupby('author').agg({"score":"sum","ups":"sum","downs":"sum"}).
↪sort(col('sum(score)').asc()).show(10)
```

```
+----------------+----------+----------+--------+
|          author|sum(score)|sum(downs)|sum(ups)|
+----------------+----------+----------+--------+
|     ItWillBeMine|     -6839|         0|   -6839|
|         blaghart|     -4233|         0|   -4233|
|        Shanondoa|     -3555|         0|   -3555|
|     bad_driverman|     -3053|         0|   -3053|
|        RSneedsEoC|     -2192|         0|   -2192|
|     b00gymonster1|     -2050|         0|   -2050|
|        frankenham|     -2024|         0|   -2024|
|     SaddharKadham|     -1485|         0|   -1485|
|letters_numbers-|     -1412|         0|   -1412|
|      djroomba322|     -1392|         0|   -1392|
+----------------+----------+----------+--------+
only showing top 10 rows
```

### 1.4 Model: Predict Score Sentiment from body

[47]:
```
# determine a scoreSentiment as either postive, neutral, or negative.
# This will be our response variable

# Note: score_sentiment: 0=negative; 1=neutral; 2=positive
splits = [-float("inf"), -0.1,0.1, float("inf")]
bkt = Bucketizer(splits=splits, inputCol="score", outputCol="scoreSentiment")

# testing
tmpDF=bkt.transform(df)
tmpDF.show(2)
```

```
+---+---------+--------------+------+-----+---------+---------------------+------
--------+-----+--------------+
|ups|subreddit|removal_reason|gilded|downs|   author|
body|distinguished|score|scoreSentiment|
+---+---------+--------------+------+-----+---------+---------------------+------
```

```
-------+-----+--------------+
|  4|soccer_jp|            NA|     0|    0|    rx109|                 |
null|    4|           2.0|
|  4|      nba|            NA|     0|    0|WyaOfWade|gg this one's ove…|
NA|    4|           2.0|
+---+---------+--------------+------+-----+---------+--------------------+------
-------+-----+--------------+
only showing top 2 rows
```

[73]:
```
# Get a summary of score sentiment by label
tmpDF.groupby('scoreSentiment').agg({"scoreSentiment":"count"}).show()
```

```
+--------------+---------------------+
|scoreSentiment|count(scoreSentiment)|
+--------------+---------------------+
|           0.0|               394008|
|           1.0|               400062|
|           2.0|              8432020|
+--------------+---------------------+
```

[48]:
```
# Create TF (Term Frequency) feature
tok = Tokenizer(inputCol="body", outputCol="words")
htf = HashingTF(inputCol="words", outputCol="tf", numFeatures=200)

#testing
tmpDF=tok.transform(tmpDF)
tmpDF=htf.transform(tmpDF)
tmpDF.select('words','tf').show(2)
```

```
+--------------------+--------------------+
|               words|                  tf|
+--------------------+--------------------+
|                [ ]|   (200,[147],[1.0])|
|[gg, this, one's,…|(200,[2,17,24,35,…|
+--------------------+--------------------+
only showing top 2 rows
```

[50]:
```
# Create w2v (word to vec) feature

# the comment string needs to be turned into a vector for w2v to work
# unfortunately, VectorAssember does not work on string so we need a UDF

# Create UDF (note: split(anything,0) simply means don't split)
str_to_vec=spark.udf.register("str_to_vec",
                             lambda row:row.split("#",0),
                                    ArrayType(StringType()))

# set up the tranformation
rva=SQLTransformer(statement="SELECT *, str_to_vec(body) bodyVec FROM __THIS__")

w2v = Word2Vec(inputCol='bodyVec', outputCol='w2v')  # not setting minCount

# testing
tmpDF=rva.transform(tmpDF)
model=w2v.fit(tmpDF)
tmpDF=model.transform(tmpDF)
tmpDF.show(2)
```

```
+---+---------+--------------+------+-----+---------+--------------------+------
-------+-----+--------------+--------------------+--------------------+---------
-----------+--------------------+
|ups|subreddit|removal_reason|gilded|downs|   author|
body|distinguished|score|scoreSentiment|                words|
tf|             bodyVec|                 w2v|
+---+---------+--------------+------+-----+---------+--------------------+------
-------+-----+--------------+--------------------+--------------------+---------
-----------+--------------------+
|  4|soccer_jp|            NA|     0|    0|    rx109|                    |
null|    4|           2.0|                [ ]|   (200,[147],[1.0])|
[ ]|[0.0,0.0,0.0,0.0,…|
|  4|      nba|            NA|     0|    0|WyaOfWade|gg this one's ove…|
NA|    4|           2.0|[gg, this, one's,…|(200,[2,17,24,35,…|[gg this one's
ov…|[0.0,0.0,0.0,0.0,…|
+---+---------+--------------+------+-----+---------+--------------------+------
-------+-----+--------------+--------------------+--------------------+---------
-----------+--------------------+
only showing top 2 rows
```

[51]:
```
# Assemble predictors
va=VectorAssembler(inputCols=['tf','w2v'],outputCol='features')
```

[55]:
```
# Set up the regression model
lr = LogisticRegression(labelCol='scoreSentiment',maxIter=10, regParam=0.3,␣
 ↪elasticNetParam=0.8)
```

[56]:
```
# Build the pipeline
pipeline=Pipeline(stages=[bkt,tok,htf,rva,w2v,va,lr])
```

[59]:
```
# Split into train and test
seed=314
trainDF,testDF=df.randomSplit([0.8,0.2],seed)
```

[60]:
```
# Fit the multinomial logistic regression model
mlrModel=pipeline.fit(trainDF)
```

[64]:
```
# Training Summary
# source: https://spark.apache.org/docs/latest/ml-classification-regression.html

# Fix source: https://stackoverflow.com/questions/37278999/
 ↪logistic-regression-with-spark-ml-data-frames
lrm=mlrModel.stages[-1]

# Print the coefficients and intercept for multinomial logistic regression
print("Coefficients: \n" + str(lrm.coefficientMatrix))
print("Intercept: " + str(lrm.interceptVector))

trainingSummary = lrm.summary

# Obtain the objective per iteration
objectiveHistory = trainingSummary.objectiveHistory
print("objectiveHistory:")
for objective in objectiveHistory:
    print(objective)

# for multiclass, we can inspect metrics on a per-label basis
print("False positive rate by label:")
for i, rate in enumerate(trainingSummary.falsePositiveRateByLabel):
    print("label %d: %s" % (i, rate))

print("True positive rate by label:")
for i, rate in enumerate(trainingSummary.truePositiveRateByLabel):
    print("label %d: %s" % (i, rate))

print("Precision by label:")
for i, prec in enumerate(trainingSummary.precisionByLabel):
    print("label %d: %s" % (i, prec))

print("Recall by label:")
for i, rec in enumerate(trainingSummary.recallByLabel):
    print("label %d: %s" % (i, rec))

print("F-measure by label:")
for i, f in enumerate(trainingSummary.fMeasureByLabel()):
    print("label %d: %s" % (i, f))

accuracy = trainingSummary.accuracy
falsePositiveRate = trainingSummary.weightedFalsePositiveRate
truePositiveRate = trainingSummary.weightedTruePositiveRate
fMeasure = trainingSummary.weightedFMeasure()
```

```
precision = trainingSummary.weightedPrecision
recall = trainingSummary.weightedRecall
print("Accuracy: %s\nFPR: %s\nTPR: %s\nF-measure: %s\nPrecision: %s\nRecall: %s"
      % (accuracy, falsePositiveRate, truePositiveRate, fMeasure, precision,␣
 ↪recall))
```

```
Coefficients:
3 X 300 CSRMatrix

Intercept: [-1.0264586714522934,-1.0107704204551655,2.037229091907459]
objectiveHistory:
0.35298803317465105
False positive rate by label:
label 0: 0.0
label 1: 0.0
label 2: 1.0
True positive rate by label:
label 0: 0.0
label 1: 0.0
label 2: 1.0
Precision by label:
label 0: 0.0
label 1: 0.0
label 2: 0.9139359489937812
Recall by label:
label 0: 0.0
label 1: 0.0
label 2: 1.0
F-measure by label:
label 0: 0.0
label 1: 0.0
label 2: 0.9550329513109017
Accuracy: 0.9139359489937812
FPR: 0.9139359489937812
TPR: 0.9139359489937812
F-measure: 0.8728389466766606
Precision: 0.8352789188631633
Recall: 0.9139359489937812
```

[66]:
```
# Make preductions on the test data
mlrPrediction=mlrModel.transform(testDF)
```

[74]:
```
mlrPrediction.select('scoreSentiment','prediction').show(3)
```

```
+--------------+----------+
|scoreSentiment|prediction|
+--------------+----------+
|           0.0|       2.0|
|           0.0|       2.0|
|           0.0|       2.0|
+--------------+----------+
only showing top 3 rows
```

#### 1.4.1 TBD: Evaluate the predictions. Judging from the training though, it seems to over-predict category 2 "positive" — which is the most prevalent

[25]:
```
# Stuff with ngrams not currently used

#May need to drop col when rerunning
#df=df.drop('body2grams')
#df=df.drop('body3grams')

# Create 2grams
#ngram = NGram(n=2, inputCol="words", outputCol="body2grams")
#df = ngram.transform(df)

# Create 3grams
#ngram = NGram(n=3, inputCol="words", outputCol="body3grams")
#df = ngram.transform(df)
```

[26]:
```
# NOT USED since scoreSentiment is multinomial response not predictor
# OneHotEncoding of Score_sentiment
# since it is already numeric, no need for StringIndexer
#encoder = OneHotEncoder(inputCol="score_sentiment",␣
↪outputCol="scoreSentimentVec")
#model = encoder.fit(df)
#df = model.transform(df)
```

[ ]:
```

```

### 1.5 Save notebook as PDF document

[ ]:
```
# Save notebook as PDF document
!jupyter nbconvert --to pdf `pwd`/*.ipynb
```