Big Data Project (21L-5625,L21-5626,L21-5632)

Movie and Political Sentiment Analysis with PySpark

```
%%bash
#Installing Pyspark
pip install pyspark
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.4.0.tar.gz (310.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.8/310.8 MB 2.9 MB/s
eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: py4j==0.10.9.7 in
/usr/local/lib/python3.9/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py): started
  Building wheel for pyspark (setup.py): finished with status 'done'
  Created wheel for pyspark: filename=pyspark-3.4.0-py2.py3-none-
any.whl size=311317145
sha256=83ae4052984f025147841752d80310a153888e6e34a3a4e1540182459372538
7
  Stored in directory:
/root/.cache/pip/wheels/9f/34/a4/159aa12d0a510d5ff7c8f0220abbea42e5d81
ecf588c4fd884
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.0
```

```
%%bash

# Download the data files from github
#or one can just run this phython notebook, since I am just linking
the sources
data_file_1=imdb_reviews_preprocessed.parquet
data_file_2=sentiments.parquet
data_file_3=tweets.parquet

# If data_file_1 file does not exist in the colab environment
if [[ ! -f ${data_file_1} ]]; then
   # download the data file from github and save it in this colab
environment instance
   wget
https://raw.githubusercontent.com/wewilli1/ist718_data/master/$
{data_file_1}
```

```
fi

# If data_file_1 file does not exist in the colab environment
if [[ ! -f ${data_file_2} ]]; then
   # download the data file from github and save it in this colab
environment instance
   wget
https://raw.githubusercontent.com/wewilli1/ist718_data/master/$
{data_file_2}
fi

# If data_file_1 file does not exist in the colab environment
if [[ ! -f ${data_file_3} ]]; then
   # download the data file from github and save it in this colab
environment instance
   wget
https://raw.githubusercontent.com/wewilli1/ist718_data/master/$
{data_file_3}
fi
```

```
--2023-04-24 19:19:31--
https://raw.githubusercontent.com/wewilli1/ist718_data/master/imdb_rev
iews_preprocessed.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 21597134 (21M) [application/octet-stream]
Saving to: ‘imdb_reviews_preprocessed.parquet’

     0K .......... .......... .......... .......... ..........  0%
8.57M 2s
    50K .......... .......... .......... .......... ..........  0%
8.00M 2s
   100K .......... .......... .......... .......... ..........  0%
42.3M 2s
   150K .......... .......... .......... .......... ..........  0%
45.5M 1s
   200K .......... .......... .......... .......... ..........  1%
12.5M 1s
   250K .......... .......... .......... .......... ..........  1%
52.1M 1s
   300K .......... .......... .......... .......... ..........  1%
47.8M 1s
   350K .......... .......... .......... .......... ..........  1%
89.3M 1s
   400K .......... .......... .......... .......... ..........  2%
233M 1s
   450K .......... .......... .......... .......... ..........  2%
```

```
14.3M 1s
   500K .......... .......... .......... .......... ..........  2%
50.0M 1s
   550K .......... .......... .......... .......... ..........  2%
62.2M 1s
   600K .......... .......... .......... .......... ..........  3%
199M 1s
   650K .......... .......... .......... .......... ..........  3%
204M 1s
   700K .......... .......... .......... .......... ..........  3%
235M 1s
   750K .......... .......... .......... .......... ..........  3%
220M 1s
   800K .......... .......... .......... .......... ..........  4%
231M 1s
   850K .......... .......... .......... .......... ..........  4%
191M 1s
   900K .......... .......... .......... .......... ..........  4%
21.4M 1s
   950K .......... .......... .......... .......... ..........  4%
47.6M 1s
  1000K .......... .......... .......... .......... ..........  4%
38.9M 1s
  1050K .......... .......... .......... .......... ..........  5%
41.5M 1s
  1100K .......... .......... .......... .......... ..........  5%
18.2M 1s
  1150K .......... .......... .......... .......... ..........  5%
40.4M 1s
  1200K .......... .......... .......... .......... ..........  5%
51.0M 1s
  1250K .......... .......... .......... .......... ..........  6%
33.3M 1s
  1300K .......... .......... .......... .......... ..........  6%
47.4M 1s
  1350K .......... .......... .......... .......... ..........  6%
42.8M 1s
  1400K .......... .......... .......... .......... ..........  6%
37.6M 1s
  1450K .......... .......... .......... .......... ..........  7%
46.8M 1s
  1500K .......... .......... .......... .......... ..........  7%
138M 1s
  1550K .......... .......... .......... .......... ..........  7%
163M 1s
  1600K .......... .......... .......... .......... ..........  7%
214M 1s
  1650K .......... .......... .......... .......... ..........  8%
184M 1s
```

```
  1700K .......... .......... .......... .......... ..........  8%
84.1M 1s
  1750K .......... .......... .......... .......... ..........  8%
44.6M 0s
  1800K .......... .......... .......... .......... ..........  8%
50.9M 0s
  1850K .......... .......... .......... .......... ..........  9%
36.7M 0s
  1900K .......... .......... .......... .......... ..........  9%
59.6M 0s
  1950K .......... .......... .......... .......... ..........  9%
42.2M 0s
  2000K .......... .......... .......... .......... ..........  9%
53.8M 0s
  2050K .......... .......... .......... .......... ..........  9%
53.7M 0s
  2100K .......... .......... .......... .......... .......... 10%
50.0M 0s
  2150K .......... .......... .......... .......... .......... 10%
48.4M 0s
  2200K .......... .......... .......... .......... .......... 10%
219M 0s
  2250K .......... .......... .......... .......... .......... 10%
198M 0s
  2300K .......... .......... .......... .......... .......... 11%
191M 0s
  2350K .......... .......... .......... .......... .......... 11%
245M 0s
  2400K .......... .......... .......... .......... .......... 11%
248M 0s
  2450K .......... .......... .......... .......... .......... 11%
194M 0s
  2500K .......... .......... .......... .......... .......... 12%
179M 0s
  2550K .......... .......... .......... .......... .......... 12%
73.6M 0s
  2600K .......... .......... .......... .......... .......... 12%
47.3M 0s
  2650K .......... .......... .......... .......... .......... 12%
52.5M 0s
  2700K .......... .......... .......... .......... .......... 13%
45.1M 0s
  2750K .......... .......... .......... .......... .......... 13%
52.4M 0s
  2800K .......... .......... .......... .......... .......... 13%
51.9M 0s
  2850K .......... .......... .......... .......... .......... 13%
47.3M 0s
  2900K .......... .......... .......... .......... .......... 13%
```

```
51.2M 0s
  2950K .......... .......... .......... .......... .......... 14%
46.3M 0s
  3000K .......... .......... .......... .......... .......... 14%
52.6M 0s
  3050K .......... .......... .......... .......... .......... 14%
65.9M 0s
  3100K .......... .......... .......... .......... .......... 14%
239M 0s
  3150K .......... .......... .......... .......... .......... 15%
214M 0s
  3200K .......... .......... .......... .......... .......... 15%
184M 0s
  3250K .......... .......... .......... .......... .......... 15%
249M 0s
  3300K .......... .......... .......... .......... .......... 15%
240M 0s
  3350K .......... .......... .......... .......... .......... 16%
209M 0s
  3400K .......... .......... .......... .......... .......... 16%
177M 0s
  3450K .......... .......... .......... .......... .......... 16%
244M 0s
  3500K .......... .......... .......... .......... .......... 16%
243M 0s
  3550K .......... .......... .......... .......... .......... 17%
197M 0s
  3600K .......... .......... .......... .......... .......... 17%
262M 0s
  3650K .......... .......... .......... .......... .......... 17%
147M 0s
  3700K .......... .......... .......... .......... .......... 17%
239M 0s
  3750K .......... .......... .......... .......... .......... 18%
248M 0s
  3800K .......... .......... .......... .......... .......... 18%
242M 0s
  3850K .......... .......... .......... .......... .......... 18%
177M 0s
  3900K .......... .......... .......... .......... .......... 18%
203M 0s
  3950K .......... .......... .......... .......... .......... 18%
247M 0s
  4000K .......... .......... .......... .......... .......... 19%
248M 0s
  4050K .......... .......... .......... .......... .......... 19%
91.3M 0s
  4100K .......... .......... .......... .......... .......... 19%
49.0M 0s
```

```
  4150K .......... .......... .......... .......... .......... 19%
55.6M 0s
  4200K .......... .......... .......... .......... .......... 20%
46.8M 0s
  4250K .......... .......... .......... .......... .......... 20%
48.6M 0s
  4300K .......... .......... .......... .......... .......... 20%
56.5M 0s
  4350K .......... .......... .......... .......... .......... 20%
43.8M 0s
  4400K .......... .......... .......... .......... .......... 21%
51.1M 0s
  4450K .......... .......... .......... .......... .......... 21%
52.3M 0s
  4500K .......... .......... .......... .......... .......... 21%
55.3M 0s
  4550K .......... .......... .......... .......... .......... 21%
52.3M 0s
  4600K .......... .......... .......... .......... .......... 22%
43.3M 0s
  4650K .......... .......... .......... .......... .......... 22%
52.5M 0s
  4700K .......... .......... .......... .......... .......... 22%
50.4M 0s
  4750K .......... .......... .......... .......... .......... 22%
52.8M 0s
  4800K .......... .......... .......... .......... .......... 22%
53.6M 0s
  4850K .......... .......... .......... .......... .......... 23%
47.3M 0s
  4900K .......... .......... .......... .......... .......... 23%
51.4M 0s
  4950K .......... .......... .......... .......... .......... 23%
49.4M 0s
  5000K .......... .......... .......... .......... .......... 23%
52.6M 0s
  5050K .......... .......... .......... .......... .......... 24%
55.3M 0s
  5100K .......... .......... .......... .......... .......... 24%
47.6M 0s
  5150K .......... .......... .......... .......... .......... 24%
57.6M 0s
  5200K .......... .......... .......... .......... .......... 24%
128M 0s
  5250K .......... .......... .......... .......... .......... 25%
188M 0s
  5300K .......... .......... .......... .......... .......... 25%
250M 0s
  5350K .......... .......... .......... .......... .......... 25%
```

```
223M 0s
  5400K .......... .......... .......... .......... .......... 25%
243M 0s
  5450K .......... .......... .......... .......... .......... 26%
233M 0s
  5500K .......... .......... .......... .......... .......... 26%
170M 0s
  5550K .......... .......... .......... .......... .......... 26%
224M 0s
  5600K .......... .......... .......... .......... .......... 26%
253M 0s
  5650K .......... .......... .......... .......... .......... 27%
223M 0s
  5700K .......... .......... .......... .......... .......... 27%
215M 0s
  5750K .......... .......... .......... .......... .......... 27%
203M 0s
  5800K .......... .......... .......... .......... .......... 27%
190M 0s
  5850K .......... .......... .......... .......... .......... 27%
250M 0s
  5900K .......... .......... .......... .......... .......... 28%
222M 0s
  5950K .......... .......... .......... .......... .......... 28%
239M 0s
  6000K .......... .......... .......... .......... .......... 28%
201M 0s
  6050K .......... .......... .......... .......... .......... 28%
253M 0s
  6100K .......... .......... .......... .......... .......... 29%
245M 0s
  6150K .......... .......... .......... .......... .......... 29%
102M 0s
  6200K .......... .......... .......... .......... .......... 29%
34.4M 0s
  6250K .......... .......... .......... .......... .......... 29%
51.1M 0s
  6300K .......... .......... .......... .......... .......... 30%
48.2M 0s
  6350K .......... .......... .......... .......... .......... 30%
90.8M 0s
  6400K .......... .......... .......... .......... .......... 30%
131M 0s
  6450K .......... .......... .......... .......... .......... 30%
67.4M 0s
  6500K .......... .......... .......... .......... .......... 31%
49.1M 0s
  6550K .......... .......... .......... .......... .......... 31%
53.3M 0s
```

```
  6600K .......... .......... .......... .......... .......... 31%
44.3M 0s
  6650K .......... .......... .......... .......... .......... 31%
68.3M 0s
  6700K .......... .......... .......... .......... .......... 32%
177M 0s
  6750K .......... .......... .......... .......... .......... 32%
165M 0s
  6800K .......... .......... .......... .......... .......... 32%
197M 0s
  6850K .......... .......... .......... .......... .......... 32%
50.8M 0s
  6900K .......... .......... .......... .......... .......... 32%
55.7M 0s
  6950K .......... .......... .......... .......... .......... 33%
54.3M 0s
  7000K .......... .......... .......... .......... .......... 33%
40.8M 0s
  7050K .......... .......... .......... .......... .......... 33%
93.2M 0s
  7100K .......... .......... .......... .......... .......... 33%
93.3M 0s
  7150K .......... .......... .......... .......... .......... 34%
55.1M 0s
  7200K .......... .......... .......... .......... .......... 34%
43.8M 0s
  7250K .......... .......... .......... .......... .......... 34%
56.3M 0s
  7300K .......... .......... .......... .......... .......... 34%
56.9M 0s
  7350K .......... .......... .......... .......... .......... 35%
59.6M 0s
  7400K .......... .......... .......... .......... .......... 35%
65.3M 0s
  7450K .......... .......... .......... .......... .......... 35%
82.9M 0s
  7500K .......... .......... .......... .......... .......... 35%
75.9M 0s
  7550K .......... .......... .......... .......... .......... 36%
73.7M 0s
  7600K .......... .......... .......... .......... .......... 36%
60.7M 0s
  7650K .......... .......... .......... .......... .......... 36%
5.65M 0s
  7700K .......... .......... .......... .......... .......... 36%
60.9M 0s
  7750K .......... .......... .......... .......... .......... 36%
59.6M 0s
  7800K .......... .......... .......... .......... .......... 37%
```

```
54.2M 0s
  7850K .......... .......... .......... .......... .......... 37%
46.9M 0s
  7900K .......... .......... .......... .......... .......... 37%
58.7M 0s
  7950K .......... .......... .......... .......... .......... 37%
23.8M 0s
  8000K .......... .......... .......... .......... .......... 38%
52.2M 0s
  8050K .......... .......... .......... .......... .......... 38%
50.8M 0s
  8100K .......... .......... .......... .......... .......... 38%
119M 0s
  8150K .......... .......... .......... .......... .......... 38%
179M 0s
  8200K .......... .......... .......... .......... .......... 39%
216M 0s
  8250K .......... .......... .......... .......... .......... 39%
203M 0s
  8300K .......... .......... .......... .......... .......... 39%
199M 0s
  8350K .......... .......... .......... .......... .......... 39%
192M 0s
  8400K .......... .......... .......... .......... .......... 40%
224M 0s
  8450K .......... .......... .......... .......... .......... 40%
218M 0s
  8500K .......... .......... .......... .......... .......... 40%
130M 0s
  8550K .......... .......... .......... .......... .......... 40%
119M 0s
  8600K .......... .......... .......... .......... .......... 41%
137M 0s
  8650K .......... .......... .......... .......... .......... 41%
252M 0s
  8700K .......... .......... .......... .......... .......... 41%
256M 0s
  8750K .......... .......... .......... .......... .......... 41%
253M 0s
  8800K .......... .......... .......... .......... .......... 41%
216M 0s
  8850K .......... .......... .......... .......... .......... 42%
227M 0s
  8900K .......... .......... .......... .......... .......... 42%
269M 0s
  8950K .......... .......... .......... .......... .......... 42%
233M 0s
  9000K .......... .......... .......... .......... .......... 42%
213M 0s
```

```
  9050K .......... .......... .......... .......... .......... 43%
244M 0s
  9100K .......... .......... .......... .......... .......... 43%
195M 0s
  9150K .......... .......... .......... .......... .......... 43%
259M 0s
  9200K .......... .......... .......... .......... .......... 43%
246M 0s
  9250K .......... .......... .......... .......... .......... 44%
237M 0s
  9300K .......... .......... .......... .......... .......... 44%
182M 0s
  9350K .......... .......... .......... .......... .......... 44%
231M 0s
  9400K .......... .......... .......... .......... .......... 44%
252M 0s
  9450K .......... .......... .......... .......... .......... 45%
251M 0s
  9500K .......... .......... .......... .......... .......... 45%
245M 0s
  9550K .......... .......... .......... .......... .......... 45%
72.9M 0s
  9600K .......... .......... .......... .......... .......... 45%
55.2M 0s
  9650K .......... .......... .......... .......... .......... 45%
40.3M 0s
  9700K .......... .......... .......... .......... .......... 46%
52.9M 0s
  9750K .......... .......... .......... .......... .......... 46%
37.2M 0s
  9800K .......... .......... .......... .......... .......... 46%
50.8M 0s
  9850K .......... .......... .......... .......... .......... 46%
49.1M 0s
  9900K .......... .......... .......... .......... .......... 47%
9.68M 0s
  9950K .......... .......... .......... .......... .......... 47%
53.8M 0s
 10000K .......... .......... .......... .......... .......... 47%
47.8M 0s
 10050K .......... .......... .......... .......... .......... 47%
54.7M 0s
 10100K .......... .......... .......... .......... .......... 48%
44.7M 0s
 10150K .......... .......... .......... .......... .......... 48%
51.5M 0s
 10200K .......... .......... .......... .......... .......... 48%
39.9M 0s
 10250K .......... .......... .......... .......... .......... 48%
```

```
55.6M 0s
 10300K .......... .......... .......... .......... .......... 49%
55.1M 0s
 10350K .......... .......... .......... .......... .......... 49%
5.45M 0s
 10400K .......... .......... .......... .......... .......... 49%
47.6M 0s
 10450K .......... .......... .......... .......... .......... 49%
57.1M 0s
 10500K .......... .......... .......... .......... .......... 50%
55.9M 0s
 10550K .......... .......... .......... .......... .......... 50%
55.7M 0s
 10600K .......... .......... .......... .......... .......... 50%
49.5M 0s
 10650K .......... .......... .......... .......... .......... 50%
65.3M 0s
 10700K .......... .......... .......... .......... .......... 50%
65.5M 0s
 10750K .......... .......... .......... .......... .......... 51%
50.9M 0s
 10800K .......... .......... .......... .......... .......... 51%
61.3M 0s
 10850K .......... .......... .......... .......... .......... 51%
80.1M 0s
 10900K .......... .......... .......... .......... .......... 51%
93.1M 0s
 10950K .......... .......... .......... .......... .......... 52%
237M 0s
 11000K .......... .......... .......... .......... .......... 52%
197M 0s
 11050K .......... .......... .......... .......... .......... 52%
250M 0s
 11100K .......... .......... .......... .......... .......... 52%
215M 0s
 11150K .......... .......... .......... .......... .......... 53%
253M 0s
 11200K .......... .......... .......... .......... .......... 53%
249M 0s
 11250K .......... .......... .......... .......... .......... 53%
77.8M 0s
 11300K .......... .......... .......... .......... .......... 53%
47.5M 0s
 11350K .......... .......... .......... .......... .......... 54%
45.0M 0s
 11400K .......... .......... .......... .......... .......... 54%
54.7M 0s
 11450K .......... .......... .......... .......... .......... 54%
54.8M 0s
```

```
 11500K .......... .......... .......... .......... .......... 54%
54.5M 0s
 11550K .......... .......... .......... .......... .......... 54%
48.5M 0s
 11600K .......... .......... .......... .......... .......... 55%
43.1M 0s
 11650K .......... .......... .......... .......... .......... 55%
55.7M 0s
 11700K .......... .......... .......... .......... .......... 55%
152M 0s
 11750K .......... .......... .......... .......... .......... 55%
224M 0s
 11800K .......... .......... .......... .......... .......... 56%
193M 0s
 11850K .......... .......... .......... .......... .......... 56%
263M 0s
 11900K .......... .......... .......... .......... .......... 56%
271M 0s
 11950K .......... .......... .......... .......... .......... 56%
213M 0s
 12000K .......... .......... .......... .......... .......... 57%
246M 0s
 12050K .......... .......... .......... .......... .......... 57%
230M 0s
 12100K .......... .......... .......... .......... .......... 57%
264M 0s
 12150K .......... .......... .......... .......... .......... 57%
227M 0s
 12200K .......... .......... .......... .......... .......... 58%
255M 0s
 12250K .......... .......... .......... .......... .......... 58%
241M 0s
 12300K .......... .......... .......... .......... .......... 58%
216M 0s
 12350K .......... .......... .......... .......... .......... 58%
216M 0s
 12400K .......... .......... .......... .......... .......... 59%
262M 0s
 12450K .......... .......... .......... .......... .......... 59%
257M 0s
 12500K .......... .......... .......... .......... .......... 59%
248M 0s
 12550K .......... .......... .......... .......... .......... 59%
200M 0s
 12600K .......... .......... .......... .......... .......... 59%
261M 0s
 12650K .......... .......... .......... .......... .......... 60%
252M 0s
 12700K .......... .......... .......... .......... .......... 60%
261M 0s
```

```
 12750K .......... .......... .......... .......... .......... 60%
86.0M 0s
 12800K .......... .......... .......... .......... .......... 60%
41.6M 0s
 12850K .......... .......... .......... .......... .......... 61%
53.6M 0s
 12900K .......... .......... .......... .......... .......... 61%
75.1M 0s
 12950K .......... .......... .......... .......... .......... 61%
209M 0s
 13000K .......... .......... .......... .......... .......... 61%
169M 0s
 13050K .......... .......... .......... .......... .......... 62%
184M 0s
 13100K .......... .......... .......... .......... .......... 62%
195M 0s
 13150K .......... .......... .......... .......... .......... 62%
222M 0s
 13200K .......... .......... .......... .......... .......... 62%
219M 0s
 13250K .......... .......... .......... .......... .......... 63%
187M 0s
 13300K .......... .......... .......... .......... .......... 63%
66.6M 0s
 13350K .......... .......... .......... .......... .......... 63%
35.4M 0s
 13400K .......... .......... .......... .......... .......... 63%
47.8M 0s
 13450K .......... .......... .......... .......... .......... 64%
52.4M 0s
 13500K .......... .......... .......... .......... .......... 64%
52.7M 0s
 13550K .......... .......... .......... .......... .......... 64%
56.0M 0s
 13600K .......... .......... .......... .......... .......... 64%
42.5M 0s
 13650K .......... .......... .......... .......... .......... 64%
47.0M 0s
 13700K .......... .......... .......... .......... .......... 65%
21.0M 0s
 13750K .......... .......... .......... .......... .......... 65%
55.6M 0s
 13800K .......... .......... .......... .......... .......... 65%
57.7M 0s
 13850K .......... .......... .......... .......... .......... 65%
38.8M 0s
 13900K .......... .......... .......... .......... .......... 66%
53.6M 0s
 13950K .......... .......... .......... .......... .......... 66%
```

```
55.3M 0s
 14000K .......... .......... .......... .......... .......... 66%
50.2M 0s
 14050K .......... .......... .......... .......... .......... 66%
66.6M 0s
 14100K .......... .......... .......... .......... .......... 67%
55.7M 0s
 14150K .......... .......... .......... .......... .......... 67%
83.5M 0s
 14200K .......... .......... .......... .......... .......... 67%
130M 0s
 14250K .......... .......... .......... .......... .......... 67%
205M 0s
 14300K .......... .......... .......... .......... .......... 68%
244M 0s
 14350K .......... .......... .......... .......... .......... 68%
244M 0s
 14400K .......... .......... .......... .......... .......... 68%
264M 0s
 14450K .......... .......... .......... .......... .......... 68%
218M 0s
 14500K .......... .......... .......... .......... .......... 68%
237M 0s
 14550K .......... .......... .......... .......... .......... 69%
242M 0s
 14600K .......... .......... .......... .......... .......... 69%
224M 0s
 14650K .......... .......... .......... .......... .......... 69%
264M 0s
 14700K .......... .......... .......... .......... .......... 69%
216M 0s
 14750K .......... .......... .......... .......... .......... 70%
258M 0s
 14800K .......... .......... .......... .......... .......... 70%
194M 0s
 14850K .......... .......... .......... .......... .......... 70%
234M 0s
 14900K .......... .......... .......... .......... .......... 70%
164M 0s
 14950K .......... .......... .......... .......... .......... 71%
256M 0s
 15000K .......... .......... .......... .......... .......... 71%
252M 0s
 15050K .......... .......... .......... .......... .......... 71%
191M 0s
 15100K .......... .......... .......... .......... .......... 71%
249M 0s
 15150K .......... .......... .......... .......... .......... 72%
233M 0s
```

```
 15200K .......... .......... .......... .......... .......... 72%
255M 0s
 15250K .......... .......... .......... .......... .......... 72%
254M 0s
 15300K .......... .......... .......... .......... .......... 72%
48.5M 0s
 15350K .......... .......... .......... .......... .......... 73%
44.1M 0s
 15400K .......... .......... .......... .......... .......... 73%
49.0M 0s
 15450K .......... .......... .......... .......... .......... 73%
50.2M 0s
 15500K .......... .......... .......... .......... .......... 73%
48.4M 0s
 15550K .......... .......... .......... .......... .......... 73%
55.1M 0s
 15600K .......... .......... .......... .......... .......... 74%
55.0M 0s
 15650K .......... .......... .......... .......... .......... 74%
39.8M 0s
 15700K .......... .......... .......... .......... .......... 74%
59.1M 0s
 15750K .......... .......... .......... .......... .......... 74%
52.5M 0s
 15800K .......... .......... .......... .......... .......... 75%
45.2M 0s
 15850K .......... .......... .......... .......... .......... 75%
54.6M 0s
 15900K .......... .......... .......... .......... .......... 75%
49.4M 0s
 15950K .......... .......... .......... .......... .......... 75%
47.4M 0s
 16000K .......... .......... .......... .......... .......... 76%
55.5M 0s
 16050K .......... .......... .......... .......... .......... 76%
54.0M 0s
 16100K .......... .......... .......... .......... .......... 76%
41.9M 0s
 16150K .......... .......... .......... .......... .......... 76%
53.3M 0s
 16200K .......... .......... .......... .......... .......... 77%
55.1M 0s
 16250K .......... .......... .......... .......... .......... 77%
50.7M 0s
 16300K .......... .......... .......... .......... .......... 77%
49.6M 0s
 16350K .......... .......... .......... .......... .......... 77%
54.6M 0s
 16400K .......... .......... .......... .......... .......... 77%
```

```
56.8M 0s
 16450K .......... .......... .......... .......... .......... 78%
78.1M 0s
 16500K .......... .......... .......... .......... .......... 78%
182M 0s
 16550K .......... .......... .......... .......... .......... 78%
249M 0s
 16600K .......... .......... .......... .......... .......... 78%
240M 0s
 16650K .......... .......... .......... .......... .......... 79%
249M 0s
 16700K .......... .......... .......... .......... .......... 79%
224M 0s
 16750K .......... .......... .......... .......... .......... 79%
226M 0s
 16800K .......... .......... .......... .......... .......... 79%
211M 0s
 16850K .......... .......... .......... .......... .......... 80%
243M 0s
 16900K .......... .......... .......... .......... .......... 80%
246M 0s
 16950K .......... .......... .......... .......... .......... 80%
214M 0s
 17000K .......... .......... .......... .......... .......... 80%
220M 0s
 17050K .......... .......... .......... .......... .......... 81%
250M 0s
 17100K .......... .......... .......... .......... .......... 81%
215M 0s
 17150K .......... .......... .......... .......... .......... 81%
238M 0s
 17200K .......... .......... .......... .......... .......... 81%
179M 0s
 17250K .......... .......... .......... .......... .......... 82%
260M 0s
 17300K .......... .......... .......... .......... .......... 82%
210M 0s
 17350K .......... .......... .......... .......... .......... 82%
243M 0s
 17400K .......... .......... .......... .......... .......... 82%
192M 0s
 17450K .......... .......... .......... .......... .......... 82%
55.3M 0s
 17500K .......... .......... .......... .......... .......... 83%
56.7M 0s
 17550K .......... .......... .......... .......... .......... 83%
49.3M 0s
 17600K .......... .......... .......... .......... .......... 83%
56.0M 0s
```

```
 17650K .......... .......... .......... .......... .......... 83%
56.9M 0s
 17700K .......... .......... .......... .......... .......... 84%
56.7M 0s
 17750K .......... .......... .......... .......... .......... 84%
54.2M 0s
 17800K .......... .......... .......... .......... .......... 84%
49.5M 0s
 17850K .......... .......... .......... .......... .......... 84%
55.4M 0s
 17900K .......... .......... .......... .......... .......... 85%
56.0M 0s
 17950K .......... .......... .......... .......... .......... 85%
55.2M 0s
 18000K .......... .......... .......... .......... .......... 85%
55.9M 0s
 18050K .......... .......... .......... .......... .......... 85%
48.8M 0s
 18100K .......... .......... .......... .......... .......... 86%
56.5M 0s
 18150K .......... .......... .......... .......... .......... 86%
52.1M 0s
 18200K .......... .......... .......... .......... .......... 86%
54.5M 0s
 18250K .......... .......... .......... .......... .......... 86%
56.8M 0s
 18300K .......... .......... .......... .......... .......... 87%
48.7M 0s
 18350K .......... .......... .......... .......... .......... 87%
54.9M 0s
 18400K .......... .......... .......... .......... .......... 87%
79.1M 0s
 18450K .......... .......... .......... .......... .......... 87%
208M 0s
 18500K .......... .......... .......... .......... .......... 87%
211M 0s
 18550K .......... .......... .......... .......... .......... 88%
245M 0s
 18600K .......... .......... .......... .......... .......... 88%
238M 0s
 18650K .......... .......... .......... .......... .......... 88%
241M 0s
 18700K .......... .......... .......... .......... .......... 88%
180M 0s
 18750K .......... .......... .......... .......... .......... 89%
250M 0s
 18800K .......... .......... .......... .......... .......... 89%
230M 0s
 18850K .......... .......... .......... .......... .......... 89%
```

```
245M 0s
 18900K .......... .......... .......... .......... .......... 89%
236M 0s
 18950K .......... .......... .......... .......... .......... 90%
160M 0s
 19000K .......... .......... .......... .......... .......... 90%
249M 0s
 19050K .......... .......... .......... .......... .......... 90%
246M 0s
 19100K .......... .......... .......... .......... .......... 90%
240M 0s
 19150K .......... .......... .......... .......... .......... 91%
199M 0s
 19200K .......... .......... .......... .......... .......... 91%
239M 0s
 19250K .......... .......... .......... .......... .......... 91%
254M 0s
 19300K .......... .......... .......... .......... .......... 91%
250M 0s
 19350K .......... .......... .......... .......... .......... 91%
204M 0s
 19400K .......... .......... .......... .......... .......... 92%
54.0M 0s
 19450K .......... .......... .......... .......... .......... 92%
56.4M 0s
 19500K .......... .......... .......... .......... .......... 92%
56.2M 0s
 19550K .......... .......... .......... .......... .......... 92%
49.5M 0s
 19600K .......... .......... .......... .......... .......... 93%
56.6M 0s
 19650K .......... .......... .......... .......... .......... 93%
56.3M 0s
 19700K .......... .......... .......... .......... .......... 93%
49.1M 0s
 19750K .......... .......... .......... .......... .......... 93%
53.6M 0s
 19800K .......... .......... .......... .......... .......... 94%
57.0M 0s
 19850K .......... .......... .......... .......... .......... 94%
55.3M 0s
 19900K .......... .......... .......... .......... .......... 94%
49.1M 0s
 19950K .......... .......... .......... .......... .......... 94%
56.1M 0s
 20000K .......... .......... .......... .......... .......... 95%
55.7M 0s
 20050K .......... .......... .......... .......... .......... 95%
56.8M 0s
```

```
 20100K .......... .......... .......... .......... .......... 95%
48.8M 0s
 20150K .......... .......... .......... .......... .......... 95%
56.9M 0s
 20200K .......... .......... .......... .......... .......... 96%
56.0M 0s
 20250K .......... .......... .......... .......... .......... 96%
52.3M 0s
 20300K .......... .......... .......... .......... .......... 96%
49.4M 0s
 20350K .......... .......... .......... .......... .......... 96%
75.6M 0s
 20400K .......... .......... .......... .......... .......... 96%
221M 0s
 20450K .......... .......... .......... .......... .......... 97%
251M 0s
 20500K .......... .......... .......... .......... .......... 97%
202M 0s
 20550K .......... .......... .......... .......... .......... 97%
251M 0s
 20600K .......... .......... .......... .......... .......... 97%
248M 0s
 20650K .......... .......... .......... .......... .......... 98%
196M 0s
 20700K .......... .......... .......... .......... .......... 98%
243M 0s
 20750K .......... .......... .......... .......... .......... 98%
240M 0s
 20800K .......... .......... .......... .......... .......... 98%
252M 0s
 20850K .......... .......... .......... .......... .......... 99%
216M 0s
 20900K .......... .......... .......... .......... .......... 99%
217M 0s
 20950K .......... .......... .......... .......... .......... 99%
248M 0s
 21000K .......... .......... .......... .......... .......... 99%
242M 0s
 21050K .......... .......... .......... ..........           100%
252M=0.3s

2023-04-24 19:19:33 (70.1 MB/s) - ‘imdb_reviews_preprocessed.parquet’
saved [21597134/21597134]

--2023-04-24 19:19:33--
https://raw.githubusercontent.com/wewilli1/ist718_data/master/sentimen
ts.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
```

```
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 56439 (55K) [application/octet-stream]
Saving to: ‘sentiments.parquet’

     0K .......... .......... .......... .......... .......... 90%
5.71M 0s
    50K .....                                                 100%
5.29M=0.009s

2023-04-24 19:19:33 (5.67 MB/s) - ‘sentiments.parquet’ saved
[56439/56439]

--2023-04-24 19:19:33--
https://raw.githubusercontent.com/wewilli1/ist718_data/master/tweets.p
arquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 154653 (151K) [application/octet-stream]
Saving to: ‘tweets.parquet’

     0K .......... .......... .......... .......... .......... 33%
6.55M 0s
    50K .......... .......... .......... .......... .......... 66%
6.82M 0s
   100K .......... .......... .......... .......... .......... 99%
40.0M 0s
   150K .                                                     100%
1961G=0.02s

2023-04-24 19:19:33 (9.31 MB/s) - ‘tweets.parquet’ saved
[154653/154653]
```

Loading packages and connecting to Spark cluster

```
from __future__ import division
from pyspark.sql import SparkSession
from pyspark.ml import feature, regression, evaluation, Pipeline
from pyspark.sql import functions as fn, Row
import matplotlib.pyplot as plt
import glob
import subprocess
import numpy as np
import pandas as pd
import os
```

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

Load sentiment data

```
sentiments_df =  spark.read.parquet('sentiments.parquet')

sentiments_df.printSchema()

root
 |-- word: string (nullable = true)
 |-- sentiment: long (nullable = true)


sentiments_df.show(5)

+------------+---------+
|        word|sentiment|
+------------+---------+
|  gratefully|        1|
|gratification|        1|
|   gratified|        1|
|   gratifies|        1|
|     gratify|        1|
+------------+---------+
only showing top 5 rows
```

The schema is very simple: for each word, we have whether it is positive (+1) or negative (-1)

```
# a sample of positive words
sentiments_df.where(fn.col('sentiment') == 1).show(5)

+------------+---------+
|        word|sentiment|
+------------+---------+
|  gratefully|        1|
|gratification|        1|
|   gratified|        1|
|   gratifies|        1|
|     gratify|        1|
+------------+---------+
only showing top 5 rows


# a sample of negative words
sentiments_df.where(fn.col('sentiment') == -1).show(5)

+----------+---------+
|      word|sentiment|
```

```
+----------+---------+
|    2-faced|       -1|
|    2-faces|       -1|
|   abnormal|       -1|
|    abolish|       -1|
|abominable|       -1|
+----------+---------+
only showing top 5 rows
```

#Lets see how many of each category we have

```
sentiments_df.groupBy('sentiment').agg(fn.count('*')).show()
```

```
+---------+--------+
|sentiment|count(1)|
+---------+--------+
|        1|    2006|
|       -1|    4783|
+---------+--------+
```

We have almost two times the number of negative words!

To test our approach, we will use a sample of IMDB reviews that were tagged as positive and negative.

Let's load them:

```
imdb_reviews_df =
spark.read.parquet('imdb_reviews_preprocessed.parquet')
imdb_reviews_df.show(10)
```

```
+---------+--------------------+-----+
|       id|              review|score|
+---------+--------------------+-----+
|pos_10006|In this "critical...|  1.0|
|pos_10013|Like one of the p...|  1.0|
|pos_10022|Aro Tolbukhin bur...|  1.0|
|pos_10033|The movie Titanic...|  1.0|
| pos_1003|Another Aussie ma...|  1.0|
| pos_1004|After a brief pro...|  1.0|
|pos_10053|I must admit, whe...|  1.0|
|pos_10062|Wow. What a wonde...|  1.0|
|pos_10074|quote by Nicolas ...|  1.0|
|pos_10083|The fact that thi...|  1.0|
+---------+--------------------+-----+
only showing top 10 rows
```

The schema is very simple: for each word, we have whether it is positive (+1) or negative (-1)

Print the unique scores in the imdb_reviews_df. A positive review has a score of 1, and a negative review has a score of 0.

```
imdb_reviews_df.toPandas()['score'].unique()
```

```
array([1., 0.])
```

Let's take a look at a positive review

```
imdb_reviews_df.where(fn.col('score') == 1).first()
```

```
Row(id='pos_10006', review='In this "critically acclaimed
psychological thriller based on true events, Gabriel (Robin Williams),
a celebrated writer and late-night talk show host, becomes captivated
by the harrowing story of a young listener and his adoptive mother
(Toni Collette). When troubling questions arise about this boy\'s
(story), however, Gabriel finds himself drawn into a widening mystery
that hides a deadly secret\x85" according to film\'s official
synopsis.<br /><br />You really should STOP reading these comments,
and watch the film NOW...<br /><br />The "How did he lose his leg?"
ending, with Ms. Collette planning her new life, should be chopped
off, and sent to "deleted scenes" land. It\'s overkill. The true
nature of her physical and mental ailments should be obvious, by the
time Mr. Williams returns to New York. Possibly, her blindness could
be in question - but a revelation could have be made certain in either
the "highway" or "video tape" scenes. The film would benefit from a
re-editing - how about a "director\'s cut"? <br /><br />Williams and
Bobby Cannavale (as Jess) don\'t seem, initially, believable as a
couple. A scene or two establishing their relationship might have
helped set the stage. Otherwise, the cast is exemplary. Williams
offers an exceptionally strong characterization, and not a "gay
impersonation". Sandra Oh (as Anna), Joe Morton (as Ashe), and Rory
Culkin (Pete Logand) are all perfect.<br /><br />Best of all,
Collette\'s "Donna" belongs in the creepy hall of fame. Ms. Oh is
correct in saying Collette might be, "you know, like that guy
from \'Psycho\'." There have been several years when organizations
giving acting awards seemed to reach for women, due to a slighter
dispersion of roles; certainly, they could have noticed Collette with
some award consideration. She is that good. And, director Patrick
Stettner definitely evokes Hitchcock - he even makes getting a
sandwich from a vending machine suspenseful.<br /><br />Finally,
writers Stettner, Armistead Maupin, and Terry Anderson deserve
gratitude from flight attendants everywhere.<br /><br />******* The
Night Listener (1/21/06) Patrick Stettner ~ Robin Williams, Toni
Collette, Sandra Oh, Rory Culkin', score=1.0)
```

And a negative one

```
imdb_reviews_df.where(fn.col('score') == 0).first()
```

```
Row(id='neg_10006', review="I don't know who to blame, the timid
writers or the clueless director. It seemed to be one of those movies
where so much was paid to the stars (Angie, Charlie, Denise, Rosanna
and Jon) that there wasn't enough left to really make a movie. This
could have been very entertaining, but there was a veil of timidity,
even cowardice, that hung over each scene. Since it got an R rating
anyway why was the ubiquitous bubble bath scene shot with a 70-year-
old woman and not Angie Harmon? Why does Sheen sleepwalk through
potentially hot relationships WITH TWO OF THE MOST BEAUTIFUL AND SEXY
ACTRESSES in the world? If they were only looking for laughs why not
cast Whoopi Goldberg and Judy Tenuta instead? This was so predictable
I was surprised to find that the director wasn't a five year old. What
a waste, not just for the viewers but for the actors as well.",
score=0.0)
```

The first problem that we encounter is that the reviews are in plain text. We need to split the words and then match them to `sentiment_df`. To do this, we will use a transformation that takes raw text and outputs a list of words

```
from pyspark.ml.feature import RegexTokenizer
```

`RegexTokenizer` extracts a sequence of matches from the input text. Regular expressions are a powerful tool to extract strings with certain characteristics. The pattern `\p{L}+` means that it will extract letters without accents (e.g., it will extract "Acuna" from "Acuña"). `setGaps=False` means that it will keep applying the rule until it can't extract new words. You have to set the input column from the incoming dataframe (in our case the `review` column) and the new column that will be added (e.g., `words`).

```
tokenizer = RegexTokenizer().setGaps(False)\
  .setPattern("\\p{L}+")\
  .setInputCol("review")\
  .setOutputCol("words")

review_words_df = tokenizer.transform(imdb_reviews_df)
print(review_words_df)
```

```
DataFrame[id: string, review: string, score: double, words:
array<string>]
```

Applying the transformation doesn't actually do anything until you apply an action. But as you can see, a new column `words` of type `array` of `string` was added by the transformation. We can see how it looks:

```
review_words_df.show(5)
```

```
+---------+--------------------+-----+--------------------+
|       id|              review|score|               words|
```

```
+---------+-------------------+-----+-------------------+
|pos_10006|In this "critical...|  1.0|[in, this, critic...|
|pos_10013|Like one of the p...|  1.0|[like, one, of, t...|
|pos_10022|Aro Tolbukhin bur...|  1.0|[aro, tolbukhin, ...|
|pos_10033|The movie Titanic...|  1.0|[the, movie, tita...|
| pos_1003|Another Aussie ma...|  1.0|[another, aussie,...|
+---------+-------------------+-----+-------------------+
only showing top 5 rows
```

Now, we want to match every word from `sentiment_df` in the array `words` shown before. One way of doing this is to *explode* the column `words` to create a row for each element in that list. Then, we would join that result with the dataframe `sentiment` to continue further.

```
review_words_df.select('id',
fn.explode('words').alias('word1')).show(5)

+---------+-------------+
|       id|        word1|
+---------+-------------+
|pos_10006|           in|
|pos_10006|         this|
|pos_10006|   critically|
|pos_10006|    acclaimed|
|pos_10006|psychological|
+---------+-------------+
only showing top 5 rows
```

Now if we join that with sentiment, we can see if there are positive and negative words in each review:

```
review_word_sentiment_df = review_words_df.\
    select('id', fn.explode('words').alias('word')).\
    join(sentiments_df, 'word')
review_word_sentiment_df.show(5)

+----------+---------+---------+
|      word|       id|sentiment|
+----------+---------+---------+
| acclaimed|pos_10006|        1|
|celebrated|pos_10006|        1|
| troubling|pos_10006|       -1|
|   mystery|pos_10006|       -1|
|    deadly|pos_10006|       -1|
+----------+---------+---------+
only showing top 5 rows
```

Check the unique sentiment column values. Sentiments should be +1 and -1 for positive and negative word sentiments respectively.

```
review_word_sentiment_df.groupBy("sentiment").count().show()
```

```
+---------+------+
|sentiment| count|
+---------+------+
|        1|257274|
|       -1|241972|
+---------+------+
```

Now we can simply average the sentiment per review id and, say, pick positive when the average is above 0, and negative otherwise.

```
simple_sentiment_prediction_df = review_word_sentiment_df.\
    groupBy('id').\
    agg(fn.avg('sentiment').alias('avg_sentiment')).\
    withColumn('predicted', fn.when(fn.col('avg_sentiment') > 0, 
1.0).otherwise(0.))
simple_sentiment_prediction_df.show(5)
```

```
+---------+-------------------+---------+
|       id|      avg_sentiment|predicted|
+---------+-------------------+---------+
|pos_10149| 0.4285714285714285|      1.0|
|pos_10377|  0.5384615384615384|     1.0|
| pos_1299| 0.0909090909090909|      1.0|
| pos_2228|-0.14285714285714285|     0.0|
| pos_5052|  0.7777777777777778|     1.0|
+---------+-------------------+---------+
only showing top 5 rows
```

Now, lets compute the accuracy of our prediction

```
imdb_reviews_df.\
    join(simple_sentiment_prediction_df, 'id').\
    select(fn.expr('float(score == predicted)').alias('correct')).\
    select(fn.avg('correct')).\
    show()
```

```
+-----------------+
|     avg(correct)|
+-----------------+
|0.732231471106131|
+-----------------+
```

A data-driven sentiment prediction

First, we need to create a sequence to take from raw text to term frequency. This is necessary because we don't know the number of tokens in the text and therefore we need to *estimate* such quantity from the data.

```
# we obtain the stop words from a website
import requests
stop_words =
requests.get('http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_
words').text.split()
stop_words[0:10]
```

```
['a',
 'about',
 'above',
 'across',
 'after',
 'afterwards',
 'again',
 'against',
 'all',
 'almost']
```

```
from pyspark.ml.feature import StopWordsRemover
sw_filter = StopWordsRemover()\
  .setStopWords(stop_words)\
  .setCaseSensitive(False)\
  .setInputCol("words")\
  .setOutputCol("filtered")
```

Count Vectorizer

```
from pyspark.ml.feature import CountVectorizer

# we will remove words that appear in 5 docs or less
cv = CountVectorizer(minTF=1., minDF=5., vocabSize=2**17)\
  .setInputCol("filtered")\
  .setOutputCol("tf")

# we now create a pipelined transformer
cv_pipeline = Pipeline(stages=[tokenizer, sw_filter,
cv]).fit(imdb_reviews_df)

# now we can make the transformation between the raw text and the
counts
cv_pipeline.transform(imdb_reviews_df).show(5)
```

```
+---------+--------------------+-----+--------------------
+--------------------+--------------------+
```

```
|       id|              review|score|               words|
filtered|                  tf|
+---------+--------------------+-----+--------------------
+--------------------+--------------------+
|pos_10006|In this "critical...|  1.0|[in, this, critic...|
[critically, accl...|(26677,[0,1,3,4,5...|
|pos_10013|Like one of the p...|  1.0|[like, one, of, t...|[like,
previous, ...|(26677,[1,2,3,4,5...|
|pos_10022|Aro Tolbukhin bur...|  1.0|[aro, tolbukhin, ...|[aro,
tolbukhin, ...|(26677,[0,1,2,12,...|
|pos_10033|The movie Titanic...|  1.0|[the, movie, tita...|[movie,
titanic, ...|(26677,[0,1,2,3,4...|
| pos_1003|Another Aussie ma...|  1.0|[another, aussie,...|[aussie,
masterpi...|(26677,[4,5,9,24,...|
+---------+--------------------+-----+--------------------
+--------------------+--------------------+
only showing top 5 rows
```

The term frequency vector is represented with a sparse vector. We have 26,677 terms.

```
len(cv_pipeline.stages[-1].vocabulary)

26677
```

Finally, we build another pipeline that takes the output of the previous pipeline and *lowers* the terms of documents that are very common.

```
from pyspark.ml.feature import IDF
idf = IDF().\
    setInputCol('tf').\
    setOutputCol('tfidf')

idf_pipeline = Pipeline(stages=[cv_pipeline,
idf]).fit(imdb_reviews_df)

idf_pipeline.transform(imdb_reviews_df).select('tfidf').show(1, False)

+---------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
```

```
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------------------+
|tfidf
|
+-----------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
```

```
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
---------------------------------------------------+
|(26677,
[0,1,3,4,5,7,8,9,10,14,21,24,26,29,30,31,37,38,46,48,51,54,59,64,88,11
6,125,129,150,189,200,202,222,227,230,232,239,243,255,261,263,269,328,
347,356,364,368,377,392,432,451,454,472,486,514,517,521,540,567,570,58
3,628,630,631,633,647,650,651,673,705,747,749,952,1058,1083,1158,1254,
1303,1316,1320,1325,1399,1400,1464,1472,1474,1530,1574,1654,1693,1791,
```

```
1815,1817,1826,1845,1917,1954,2008,2009,2011,2022,2117,2188,2272,2453,
2464,2882,2946,3018,3109,3289,3309,3538,3622,3624,3642,3794,3922,4043,
4354,4513,4815,5050,5158,5186,5576,5745,5947,6089,6189,6263,6357,6360,
7205,7219,7508,7972,9998,10080,10103,10347,10535,10684,11840,12761,136
33,14591,15021,15624,16078,18058,20532,22354,24267,24420,25217],
[6.399789021643247,1.5355439272599614,1.751599547551507,0.502897453316
8351,0.7612090942068761,0.9527872680847915,1.051577880482202,2.3754916
62124171,1.1869590489601298,1.3491156498373038,1.5198200397035864,1.68
87375926197539,1.53920585864879,1.6292725389623013,1.6586681558248935,
1.6538478593700623,1.8765733504775144,3.705648340531702,1.924736751856
1371,3.905063519241778,4.097206442469843,1.9536597913253875,2.04457782
4871977,2.2186516584178886,2.4178115020031847,2.4609176711886604,2.553
139840304409,5.076694851430274,5.339278027855957,2.950416571033215,2.8
99008604321704,2.8228278777726548,2.8577509748566374,2.912166688899215
5,2.8732007203658316,3.0349289879552064,2.895384098722744,3.1456653623
286295,2.951946793213983,8.994525472110968,3.2555798084398138,2.949652
337140459,3.125425480980883,3.210947654419045,3.2059995988016756,3.257
6566523846524,3.216917821405549,3.2350452059981056,3.3365738675364547,
3.437071833871393,3.3980424899656576,3.360479388390009,3.4802805882026
293,3.522727278449887,3.5755907680069545,3.5336265689079225,3.44707191
72059763,3.6969516250112218,3.5784520002879865,3.5755907680069545,3.57
2737699024548,3.7987343193211642,3.6223829295137135,3.734753989657757,
3.62538143250997,4.135206555942377,3.7083061671141477,3.66520292669664
15,3.734753989657757,3.8462752640901643,3.916071026025706,4.2050926834
06543,4.1105139433520055,4.115403928646197,4.205092683406543,4.2240377
69648993,4.367769329173078,9.116653198578526,4.3306133522849874,4.3836
67915240877,23.37816324742329,4.4096434016441375,4.577595018155139,4.4
95459321228993,4.499049989359722,4.467188887290738,4.467188887290738,4
.506270237333209,4.546941277064137,4.662839298024749,5.206690177222234
,9.93523160767166,4.728508401532606,4.692949099496119,4.79879493426077
8,4.779563572332891,4.7421760402612705,10.128152140046785,5.0962331816
57924,4.756033074922696,4.89022914022041,5.129458829286244,4.933714252
160149,4.900924429337158,4.996772388127286,5.192197169919667,5.2668586
98688687,5.185028680441055,5.5833763207803555,5.251473779849207,5.3391
793602683135,5.330880557453618,5.863991226009044,5.63803473331822,5.70
7830495253761,5.472710752892835,5.5115505862091,5.501698289766088,12.1
67239670431618,12.510940184284935,5.672323806796851,6.10131941231521,5
.795937762764028,6.276523501340301,5.921978483659393,5.967788019690687
,6.06622809250394,12.64001722656008,6.083619835215809,6.06622809250394
,39.258912987565495,6.10131941231521,6.137687056486085,6.2980297065612
64,6.298029706561264,6.32000861328004,13.66166847409206,6.725473721388
203,6.794466592875155,13.98235377424242,14.071257299384087,6.830834237
04603,7.035628649692043,6.99117688712121,7.082148665326936,7.182232123
883919,7.293457758994143,7.354082380810578,7.4876137734351005,7.561721
745588822,7.824086010056313,8.047229561370523,8.047229561370523,8.1807
60953995046,8.334911633822305,8.334911633822305])|
+---------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
```

```
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
--------------------------------------------------------+
```

```
only showing top 1 row
```

Therefore, the `idf_pipeline` takes the raw text from the datafarme `imdb_reviews_df` and creates a feature vector called `tfidf`!

```
tfidf_df = idf_pipeline.transform(imdb_reviews_df)
print("tfidf_df shape: ", tfidf_df.count(), len(tfidf_df.columns))
```

```
tfidf_df shape:  25000 7
```

```
tfidf_df.show(5)
```

```
+---------+--------------------+-----+--------------------
+--------------------+--------------------+--------------------+
|       id|              review|score|               words|
filtered|                  tf|               tfidf|
+---------+--------------------+-----+--------------------
+--------------------+--------------------+--------------------+
|pos_10006|In this "critical...|  1.0|[in, this, critic...|
[critically, accl...|(26677,[0,1,3,4,5...|(26677,[0,1,3,4,5...|
|pos_10013|Like one of the p...|  1.0|[like, one, of, t...|[like,
previous, ...|(26677,[1,2,3,4,5...|(26677,[1,2,3,4,5...|
|pos_10022|Aro Tolbukhin bur...|  1.0|[aro, tolbukhin, ...|[aro,
tolbukhin, ...|(26677,[0,1,2,12,...|(26677,[0,1,2,12,...|
|pos_10033|The movie Titanic...|  1.0|[the, movie, tita...|[movie,
titanic, ...|(26677,[0,1,2,3,4...|(26677,[0,1,2,3,4...|
| pos_1003|Another Aussie ma...|  1.0|[another, aussie,...|[aussie,
masterpi...|(26677,[4,5,9,24,...|(26677,[4,5,9,24,...|
+---------+--------------------+-----+--------------------
+--------------------+--------------------+--------------------+
only showing top 5 rows
```

The cell below prints out the 'tf' and 'tfidf' columns for the first 10 rows of the tfidf_df. Note that the tfidf column is transformed to account for the frequency with which the word appears in the corpus. Words that appear more often are penalized more than words that do not appear as frequently in the corpus.

```
tfidf_df.limit(10).toPandas().loc[:10, ['tf', 'tfidf']]
```

```
                                                  tf  \
0  (12.0, 5.0, 0.0, 3.0, 1.0, 1.0, 0.0, 1.0, 1.0,...
1  (0.0, 2.0, 4.0, 1.0, 1.0, 3.0, 3.0, 0.0, 0.0, ...
2  (6.0, 2.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
3  (6.0, 1.0, 3.0, 7.0, 3.0, 2.0, 1.0, 0.0, 2.0, ...
4  (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...
5  (14.0, 7.0, 1.0, 6.0, 2.0, 0.0, 2.0, 3.0, 1.0,...
6  (6.0, 2.0, 3.0, 1.0, 2.0, 1.0, 0.0, 0.0, 1.0, ...
7  (6.0, 0.0, 0.0, 2.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...
```

```
8  (4.0, 0.0, 0.0, 4.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...
9  (6.0, 1.0, 0.0, 2.0, 1.0, 1.0, 1.0, 1.0, 0.0, ...

                                               tfidf
0  (6.399789021643247, 1.5355439272599614, 0.0, 1...
1  (0.0, 0.6142175709039845, 1.9611159574304948, ...
2  (3.1998945108216237, 0.6142175709039845, 0.490...
3  (3.1998945108216237, 0.30710878545199227, 1.47...
4  (0.0, 0.0, 0.0, 0.0, 0.5028974533168351, 0.761...
5  (7.466420525250455, 2.149761498163946, 0.49027...
6  (3.1998945108216237, 0.6142175709039845, 1.470...
7  (3.1998945108216237, 0.0, 0.0, 1.1677330317010...
8  (2.1332630072144156, 0.0, 0.0, 2.3354660634020...
9  (3.1998945108216237, 0.30710878545199227, 0.0,...
```

Data science pipeline for estimating sentiments

First, let's split the data into training, validation, and testing.

```
training_df, validation_df, testing_df =
imdb_reviews_df.randomSplit([0.6, 0.3, 0.1], seed=0)

training_df.limit(5).toPandas()
```

	id	review	score
0	neg_1	Robert DeNiro plays the most unbelievably inte...	0.0
1	neg_10	This film had a lot of promise, and the plot w...	0.0
2	neg_100	OK its not the best film I've ever seen but at...	0.0
3	neg_10000	Airport '77 starts as a brand new luxury 747 p...	0.0
4	neg_10001	This film lacked something I couldn't put my f...	0.0

```
[training_df.count(), validation_df.count(), testing_df.count()]

[14962, 7531, 2507]
```

Logistic Regression

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression().\
    setLabelCol('score').\
    setFeaturesCol('tfidf').\
    setRegParam(0.0).\
    setMaxIter(100).\
    setElasticNetParam(0.)
```

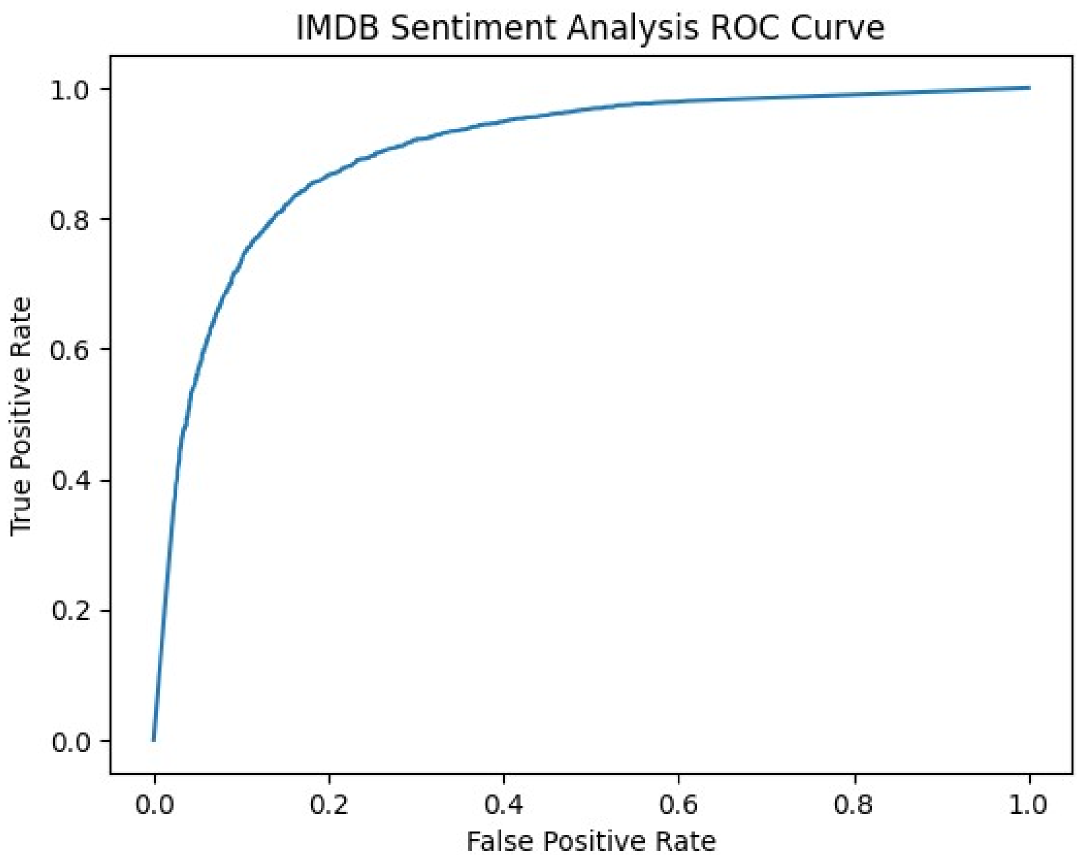
Lets create a pipeline transformation by chaining the `idf_pipeline` with the logistic regression step (`lr`)

```
lr_pipeline = Pipeline(stages=[idf_pipeline, lr]).fit(training_df)
```

The next cell defines a class capable of calculating ROC and PR curves. The following cell creates a ROC curve using transformed data from the pipeline above.

```
# see https://stackoverflow.com/questions/52847408/pyspark-extract-roc-curve
from pyspark.mllib.evaluation import BinaryClassificationMetrics

# Scala version implements .roc() and .pr()
# Python: https://spark.apache.org/docs/latest/api/python/_modules/pyspark/mllib/common.html
# Scala: https://spark.apache.org/docs/latest/api/java/org/apache/spark/mllib/evaluation/BinaryClassificationMetrics.html
class CurveMetrics(BinaryClassificationMetrics):
    def __init__(self, *args):
        super(CurveMetrics, self).__init__(*args)

    def _to_list(self, rdd):
        points = []

        for row in rdd.collect():

            points += [(float(row._1()), float(row._2()))]
        return points

    def get_curve(self, method):
        rdd = getattr(self._java_model, method)().toJavaRDD()
        return self._to_list(rdd)

import matplotlib.pyplot as plt

# Create a Pipeline estimator and fit on train DF, predict on test DF
predictions = lr_pipeline.transform(validation_df)

# Returns as a list (false positive rate, true positive rate)
preds = predictions.select('score','probability').rdd.map(lambda row: (float(row['probability'][1]), float(row['score'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('IMDB Sentiment Analysis ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(x_val, y_val)
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/context.py:157:
FutureWarning: Deprecated in 3.0.0. Use
SparkSession.builder.getOrCreate() instead.
  warnings.warn(

[<matplotlib.lines.Line2D at 0x7fed7c131610>]
```



Lets estimate the accuracy:

```
lr_pipeline.transform(validation_df).\
    select(fn.expr('float(prediction = score)').alias('correct')).\
    select(fn.avg('correct')).show()

+------------------+
|      avg(correct)|
+------------------+
|0.8360111538972248|
+------------------+
```

The performance is much better than before.

The problem however is that we are overfitting because we have many features compared to the training examples:

For example, if we look at the weights of the features, there is a lot of noise:

```
import pandas as pd
vocabulary = idf_pipeline.stages[0].stages[-1].vocabulary
weights = lr_pipeline.stages[-1].coefficients.toArray()
print("num weights:", len(weights))
print("num rows:", validation_df.count())

coeffs_df = pd.DataFrame({'word': vocabulary, 'weight': weights})
coeffs_df.head()
```

```
num weights: 26677
num rows: 7531
```

	word	weight
0	br	-0.105915
1	s	0.181536
2	movie	-0.336467
3	film	0.194032
4	t	-0.783280

The most negative words are:

```
coeffs_df.sort_values('weight').head(5)
```

	word	weight
23304	zoolander	-6.642286
21131	grossness	-6.096098
26293	residue	-5.677060
23883	publicize	-5.493201
23282	dodgeball	-5.350357

And the most positive:

```
coeffs_df.sort_values('weight', ascending=False).head(5)
```

	word	weight
20792	choco	7.881534
26406	silhouetted	7.406967
26318	eliciting	6.151619
21679	praiseworthy	6.142447
13496	shrewd	5.979843

But none of them make sense. What is happening? We are overfitting the data. Those words that don't make sense are capturing just noise in the reviews.

Spark allows us to fit elastic net regularization easily

```
lambda_par = 0.02
alpha_par = 0.3
en_lr = LogisticRegression().\
        setLabelCol('score').\
        setFeaturesCol('tfidf').\
        setRegParam(lambda_par).\
        setMaxIter(100).\
        setElasticNetParam(alpha_par)
```

And we define a new Pipeline with all steps combined

```
en_lr_estimator = Pipeline(stages=[tokenizer, sw_filter, cv, idf,
en_lr])

en_lr_pipeline = en_lr_estimator.fit(training_df)
```

Let's look at the performance

```
en_lr_pipeline.transform(validation_df).select(fn.avg(fn.expr('float(p
rediction = score)'))).show()
```

```
+------------------------+
|avg((prediction = score))|
+------------------------+
|      0.8688089231177798|
+------------------------+
```

We improve performance slightly, but whats more important is that we improve the understanding of the word sentiments. Lets look at the weights:

```
en_weights = en_lr_pipeline.stages[-1].coefficients.toArray()
en_coeffs_df = pd.DataFrame({'word':
en_lr_pipeline.stages[2].vocabulary, 'weight': en_weights})
```

The most negative words all make sense ("worst" is *actually* more negative than than "worse")!

```
en_coeffs_df.sort_values('weight').head(15)
```

	word	weight
105	worst	-0.369188
262	waste	-0.334131
190	awful	-0.259531
12	bad	-0.238970
606	poorly	-0.185563
522	dull	-0.185538
1109	disappointment	-0.181951

```
184           boring -0.180375
179             poor -0.177850
243            worse -0.173731
351         horrible -0.172044
1376       redeeming -0.163199
1180   disappointing -0.154673
579            avoid -0.152653
1032       laughable -0.151916
```

Same thing with positive words

```
en_coeffs_df.sort_values('weight', ascending=False).head(15)
```

```
             word    weight
13          great  0.283820
161     excellent  0.240269
221     wonderful  0.200133
26           best  0.168678
300      favorite  0.161667
227       perfect  0.157898
286       amazing  0.142479
889    incredible  0.139636
270         loved  0.133010
2047   refreshing  0.129359
1962     captures  0.127790
311       enjoyed  0.125049
700     perfectly  0.123770
309         today  0.122013
3047     flawless  0.120523
```

Are there words with *literarily* zero importance for predicting sentiment? Yes, and most of them!

```
en_coeffs_df.query('weight == 0.0').shape
```

```
(19609, 2)
```

In fact, approximately 95% of features are not needed to achieve a **better** performance than all previous models!

```
en_coeffs_df.query('weight == 0.0').shape[0]/en_coeffs_df.shape[0]
```

```
0.9502786527744124
```

Let's look at these *neutral* words

```
en_coeffs_df.query('weight == 0.0').head(15)
```

```
          word  weight
0           br     0.0
```

```
3         film     0.0
5         like     0.0
9        story     0.0
10      really     0.0
11      people     0.0
14         don     0.0
15         way     0.0
17      movies     0.0
18       think     0.0
19  characters     0.0
20   character     0.0
29      little     0.0
31        know     0.0
32         man     0.0
```

But, did we choose the right λ and α parameters? We should run an experiment where we try different combinations of them. Fortunately, Spark let us do this by using a grid - a method that generates combination of parameters.

```
from pyspark.ml.tuning import ParamGridBuilder
```

We need to build a new estimator pipeline

```
en_lr_estimator.getStages()

[RegexTokenizer_18e623850b1a,
 StopWordsRemover_49bcf0e3b650,
 CountVectorizer_789893e3001b,
 IDF_507b4bb23612,
 LogisticRegression_b427af0d8b7f]

grid = ParamGridBuilder().\
    addGrid(en_lr.regParam, [0., 0.01, 0.02]).\
    addGrid(en_lr.elasticNetParam, [0., 0.2, 0.4]).\
    build()
```

This is the list of parameters that we will try:

```
grid

[{Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.0,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.0},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.0,
  Param(parent='LogisticRegression_b427af0d8b7f',
```

```
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.2},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.0,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.4},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.01,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.0},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.01,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.2},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.01,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.4},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.02,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.0},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.02,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.2},
 {Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.02,
  Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.4}]

all_models = []
for j in range(len(grid)):
    print("Fitting model {}".format(j+1))
```

```
    model = en_lr_estimator.fit(training_df, grid[j])
    all_models.append(model)
```

```
Fitting model 1
Fitting model 2
Fitting model 3
Fitting model 4
Fitting model 5
Fitting model 6
Fitting model 7
Fitting model 8
Fitting model 9
```

```
# estimate the accuracy of each of them:
accuracies = [m.\
    transform(validation_df).\
    select(fn.avg(fn.expr('float(score =
prediction)')).alias('accuracy')).\
    first().\
    accuracy for m in all_models]
```

```
accuracies
```

```
[0.8389324126941973,
 0.8389324126941973,
 0.8389324126941973,
 0.8633647589961493,
 0.8794316823794981,
 0.8729252423316957,
 0.8678794316823795,
 0.8761120701102111,
 0.8600451467268623]
```

```
import numpy as np
```

```
best_model_idx = np.argmax(accuracies)
print("best model index =", best_model_idx)
```

```
best model index = 4
```

So the best model we found has the following parameters

```
grid[best_model_idx]
```

```
{Param(parent='LogisticRegression_b427af0d8b7f', name='regParam',
doc='regularization parameter (>= 0).'): 0.01,
 Param(parent='LogisticRegression_b427af0d8b7f',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in range
[0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it
is an L1 penalty.'): 0.2}
```

```
best_model = all_models[best_model_idx]
```

```
accuracies[best_model_idx]
```

```
0.8794316823794981
```

```
# estimate generalization performance
best_model.\
    transform(testing_df).\
    select(fn.avg(fn.expr('float(score =
prediction)')).alias('accuracy')).\
    show()
```

```
+------------------+
|          accuracy|
+------------------+
|0.8687674511368169|
+------------------+
```

Finally, predicting tweet sentiments

Now we can use this model to predict sentiments on Twitter

```
tweets_df = spark.read.parquet('tweets.parquet')
```

We have 1K tweets from each candidate

```
tweets_df.groupby('handle').agg(fn.count('*')).show()
```

```
+----------------+--------+
|          handle|count(1)|
+----------------+--------+
| @HillaryClinton|    1000|
|@realDonaldTrump|    1000|
+----------------+--------+
```

We can now predict the sentiment of the Tweet using our best model, we need to rename the column so that it matches our previous pipeline (`review` => ...)

```
best_model.transform(tweets_df.withColumnRenamed('text',
'review')).select('handle', 'review', 'prediction').show()
```

```
+---------------+--------------------+----------+
|         handle|              review|prediction|
+---------------+--------------------+----------+
|@HillaryClinton|RT @ZekeJMiller: ...|       1.0|
|@HillaryClinton|“She’s just out t...|       1.0|
```

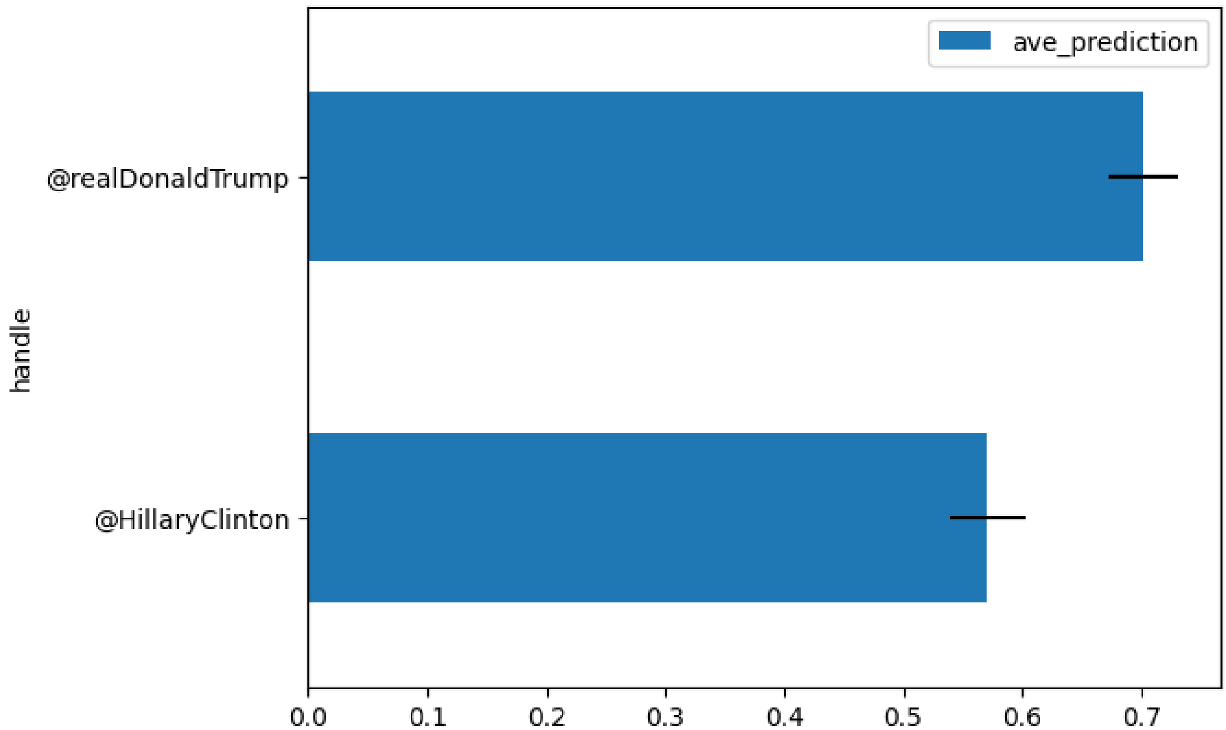
```
|@HillaryClinton|We're going to ma...|       0.0|
|@HillaryClinton|Don't boo. Vote! ...|       0.0|
|@HillaryClinton|This Republican d...|       0.0|
|@HillaryClinton|Hillary teamed up...|       1.0|
|@HillaryClinton|RT @mayaharris_: ...|       1.0|
|@HillaryClinton|"It was overwhelm...|       1.0|
|@HillaryClinton|Great step forwar...|       1.0|
|@HillaryClinton|"I feel like I'm ...|       1.0|
|@HillaryClinton|Nobody here was “...|       1.0|
|@HillaryClinton|For those few peo...|       0.0|
|@HillaryClinton|Remember, don't b...|       1.0|
|@HillaryClinton|Too many talented...|       1.0|
|@HillaryClinton|There are hundred...|       0.0|
|@HillaryClinton|It's 3:20am. As g...|       1.0|
|@HillaryClinton|Trump stood on a ...|       0.0|
|@HillaryClinton|Donald Trump said...|       1.0|
|@HillaryClinton|RT @timkaine: 39 ...|       1.0|
|@HillaryClinton|Trump wants to br...|       1.0|
+---------------+--------------------+----------+
only showing top 20 rows
```

Now, lets summarize our results in a graph!

```
%matplotlib inline

import seaborn

sentiment_pd = best_model.\
    transform(tweets_df.withColumnRenamed('text', 'review')).\
    groupby('handle').\
    agg(fn.avg('prediction').alias('ave_prediction'),
(2*fn.stddev('prediction')/fn.sqrt(fn.count('*'))).alias('std_err')).\
    toPandas()

sentiment_pd.head()
```

	handle	ave_prediction	std_err
0	@HillaryClinton	0.571	0.031318
1	@realDonaldTrump	0.702	0.028942

```
sentiment_pd.plot(x='handle', y='ave_prediction', xerr='std_err',
kind='barh');
```



But let's examine some "negative" tweets by Trump

```
best_model.\
    transform(tweets_df.withColumnRenamed('text', 'review')).\
    where(fn.col('handle') == '@realDonaldTrump').\
    where(fn.col('prediction') == 0).\
    select('review').\
    take(5)
```

```
[Row(review='Moderator: Hillary paid $225,000 by a Brazilian bank for
a speech that called for “open borders.” That’s a quote! #Debate
#BigLeagueTruth'),
 Row(review='TRUMP &amp; CLINTON ON IMMIGRATION\n#Debate
#BigLeagueTruth https://t.co/OP4c7Jc8Ad'),
 Row(review='Hillary is too weak to lead on border security-no
solutions, no ideas, no credibility.She supported NAFTA, worst deal in
US history. #Debate'),
 Row(review='One of my first acts as President will be to deport the
drug lords and then secure the border. #Debate #MAGA'),
 Row(review='Hillary Clinton will use American tax dollars to provide
amnesty for thousands of illegals. I will put…
https://t.co/ZpV33TfbR6')]
```

And Clinton

```
best_model.\
    transform(tweets_df.withColumnRenamed('text', 'review')).\
    where(fn.col('handle') == '@HillaryClinton').\
    where(fn.col('prediction') == 0).\
    select('review').\
    take(5)
```

```
[Row(review="We're going to make college debt-free for everyone in
America. See how much you could save with Hillary's plan at…
https://t.co/Fhzkubhpj7"),
 Row(review="Don't boo. Vote! https://t.co/tTgeqy51PU
https://t.co/9un3FUVxoG"),
 Row(review='This Republican dad is struggling with the idea of his
daughter growing up in a country led by Donald Trump.
https://t.co/Tn3rQqJJKp'),
 Row(review="For those few people knocking public service, hope you'll
reconsider answering the call to help others. Because we're stronger
together."),
 Row(review="There are hundreds of thousands more @AmeriCorps
applications than spots. Horrible! Let's expand it from 75,000 annual
members to 250,000.")]
```

```
from pyspark.ml import feature

from pyspark.sql import types

def probability_positive(probability_column):
    return float(probability_column[1])
func_probability_positive = fn.udf(probability_positive,
types.DoubleType())

prediction_probability_df = best_model.transform(validation_df).\
    withColumn('probability_positive',
func_probability_positive('probability')).\
    select('id', 'review', 'score', 'probability_positive')
prediction_probability_df.show()
```

```
+---------+--------------------+-----+--------------------+
|       id|              review|score|probability_positive|
+---------+--------------------+-----+--------------------+
|    neg_0|Story of a man wh...|  0.0|  0.4858256123045229|
| neg_1000|The plot for Desc...|  0.0| 0.23773092718405964|
|neg_10003|When I was little...|  0.0|6.864567527598009E-4|
|neg_10006|I don't know who ...|  0.0| 0.08939747035805656|
|neg_10012|This movie must b...|  0.0| 0.00807233188176526|
|neg_10014|I saw this movie ...|  0.0| 0.45210546440878807|
|neg_10019|Kareena Kapoor in...|  0.0|4.943247978200782E-4|
|neg_10022|Summer season is ...|  0.0|  0.6559859138052624|
|neg_10023|Shame on Yash Raj...|  0.0|2.732321286786909...|
|neg_10024|First lesson that...|  0.0| 0.03837454919442762|
```

```
|neg_10026|I had some expect...|  0.0|   0.567648231964684|
| neg_1003|OK, I am not Japa...|  0.0| 0.16001109982402284|
|neg_10033|I was very disple...|  0.0|  0.3506335602422507|
|neg_10034|If there is one f...|  0.0|2.447261518190302...|
|neg_10036|Sometime I fail t...|  0.0|  0.5942199783219011|
|neg_10041|The sight of Kare...|  0.0|4.983562194804669E-6|
|neg_10050|A huge disappoint...|  0.0|0.005242299674230...|
|neg_10052|Warner Bros. made...|  0.0| 0.20662075785815548|
|neg_10055|I grew up on the ...|  0.0|  0.3142833171137729|
|neg_10058|I was fascinated ...|  0.0| 0.12538714032358522|
+---------+--------------------+-----+--------------------+
only showing top 20 rows
```