Homework 3 Written Answers

Data Engineering 300
Aden Benson
5/26/2025

Part 1

Task 1

To begin Task 1, I first created a "words" Pyspark dataframe that exploded all document columns and listed each individual word with its corresponding document number. This is shown in figure 1.

```
+---+---------+
|_c0|     word|
+---+---------+
|  0|     wall|
|  0|       st|
|  0|    bears|
|  0|     claw|
|  0|     back|
|  0|    black|
|  0|  reuters|
|  0|  reuters|
|  0|    short|
|  0|  sellers|
|  0|     wall|
|  0|   street|
|  0|dwindling|
|  0|     band|
|  0|    ultra|
|  0|   cynics|
|  0|   seeing|
|  0|    green|
|  1|  carlyle|
|  1|    looks|
+---+---------+
```

Fig. 1 - Exploded word dataframe

After this, I was able to design map/reduce functions to create dataframes for the following statements: For each *d*, the counts of *t,* for each *d* the counts of words, and for each *t*, the counts of *d* that contain *t*. The structure for the second and third dataframe are shown below in figures 2 and 3. The first dataframe was too large to use .collect().

```
[(0, 18.0), (9, 19.0), (18, 21.0), (27, 21.0), (36, 26.0)]
```

Fig. 2 - For each *d* the counts of words (Doc number, count of words)

```
[('nation', 1598.0),
 ('eggs', 38.0),
 ('software', 3798.0),
 ('different', 410.0),
 ('7', 2614.0)]
```

Fig. 3 - For each *t*, the counts of *d* that contain *t* (Word, number of occurrences in documents)

Task 2

In Task 2, we create functions that first calculate the TF term, then the IDF term, and combine these measures to compute the TF-IDF measure for each word in each document. In figure 4, we see an example of how this is displayed in our final dataframe.

```
+----------+-------+------------------+
|doc_Number|   word|     tf_idf_measure|
+----------+-------+------------------+
|        18|deficit|  0.540640233790213|
|     11970|deficit|0.24681401977379286|
|     20826|deficit|0.23653010228321816|
|     21078|deficit|0.23170295733866267|
|     37116|deficit|0.24681401977379286|
+----------+-------+------------------+
```

Fig. 4 - Final dataframe that contains a document number, a word, and its TF-IDF measure.

Task 3

In Task 3, we find the TF-IDF measure for each word in the first five documents. These had to be displayed separately since there are over 10 words in each document. In figures 5-6, we see these measures.

doc_Number	word	tf_idf_measure
0	seeing	0.37743394553516213
0	short	0.2773120373951269
0	claw	0.499114829314058
0	back	0.1892216338539946
0	green	0.2877107940095433
0	dwindling	0.4572386180709258
0	ultra	0.4125512394225831
0	st	0.2584728642725166
0	sellers	0.4468379768438066
0	street	0.24678348986493034
0	cynics	0.563734318747707
0	bears	0.3372044607529448
0	wall	0.5115985326511431
0	reuters	0.24754017186645658
0	black	0.2953171727366614
0	band	0.3643421454792778

doc_Number	word	tf_idf_measure
1	placed	0.2284965552404658
1	industry	0.15043731768548949
1	aerospace	0.2581171817448437
1	quietly	0.25188254045524316
1	looks	0.1973537176743789
1	bets	0.27861293130724324
1	toward	0.1898997183872362
1	carlyle	0.7168306746824437
1	firm	0.15969712503706046
1	defense	0.1751279339938823
1	plays	0.22418048797172685
1	investment	0.1890771769001148
1	commercial	0.2057832028092643
1	market	0.13394932212703356
1	private	0.1929050573011279
1	reputation	0.2578098186776328
1	timed	0.324478643568105
1	making	0.1698717076460444
1	group	0.12468100563149095
1	part	0.16022031730914288

doc_Number	word	tf_idf_measure
2	expected	0.16094627131903613
2	earnings	0.1792714404894228
2	stock	0.17879168082328206
2	prices	0.14472559202114177
2	hang	0.30475018305843793
2	worries	0.23009353850726894
2	stocks	0.14976769101715193
2	depth	0.31343954772064864
2	outlook	0.4265073217271922
2	oil	0.13908157105107033
2	summer	0.22694739048609625
2	market	0.15069298739291276
2	next	0.14062721303262238
2	soaring	0.2596334462817101
2	crude	0.197241148492091
2	economy	0.3721400726458204
2	week	0.13121900794126834
2	plus	0.24449073714833106
2	cloud	0.295159450642955
2	reuters	0.18565512889984243

Fig. 5 - TF-IDF measures for documents 1-3

doc_Number	word	tf_idf_measure
3	iraq	0.23809526243476142
3	showed	0.1743365558077232
3	intelligence	0.20782569445751425
3	strike	0.17411586950893898
3	official	0.15149485319300557
3	flows	0.2774168429760197
3	main	0.36492623402353547
3	said	0.06593367258642661
3	rebel	0.18209445014364567
3	militia	0.2252006141545402
3	oil	0.35763832555989516
3	pipeline	0.4720829409342409
3	infrastructure	0.22959926718225876
3	export	0.23862435123782139
3	authorities	0.18159366801541998
3	southern	0.336553609483104
3	saturday	0.12197305137253434
3	halts	0.27365396741681164
3	halted	0.2557691357056513
3	reuters	0.15913296762843637

doc_Number	word	tf_idf_measure
4	time	0.10623532598945136
4	us	0.1669859687392097
4	present	0.22209684830286883
4	record	0.1232987151692413
4	months	0.14002501854271598
4	soar	0.2306791247647116
4	prices	0.23156094723382684
4	barely	0.21935019724396657
4	wallets	0.2665151844733088
4	menace	0.5747440955975784
4	toppling	0.27964532733021175
4	oil	0.22253051368171256
4	tearaway	0.3918885216630942
4	new	0.1271397626254836
4	straining	0.2904044404056468
4	economy	0.14885602905832815
4	posing	0.2589223867776184
4	economic	0.14782686453681568
4	world	0.09332201126546583
4	elections	0.16009904796740967

Fig. 6 - TF-IDF measures for documents 4-5

Part 2

Task 1

In Task 1, I first designed MapReduce functions that would later be used in the loss_SVM() function to calculate a sum for the loss function. The summed total is shown below in figure 7.

```
# Design reduce portion, which finds the sum of all max(0, 1-y_i(w^T x_i +b)) values
from operator import add
summed_total=svm_ps.rdd.map(map_part2).reduceByKey(add).collect()[0][1]
summed_total
```

✓ 2.4s

```
399889.5049647091
```

Fig. 7 - Summed total used below in loss_SVM function

Task 2

Task 2 has no visible outputs, as it is simply defining a function to compute the loss objective function given weights, bias, X, y, and a lambda value.

Task 3

In Task 3, we compute the loss_SVM value given our data uploaded into Pyspark. We see that the objective value for the given data is 1.0029404.

```
# Create X and y dfs
X = svm_ps.select(svm_ps.columns[:-1])
y = svm_ps.select(svm_ps.columns[-1])

print('Objective value: ', loss_SVM(w_ps, bias_ps, X, y,1))
```

✓ 7.5s

```
[Stage 85:=====>                                          (1 + 9) / 10]
Objective value:  1.0029403834857487
```

Fig. 8 - Loss objective value from given data

Task 4

Finally, we design MapReduce functions to make predictions using the y_hat function provided in the assignment documentation. In figure 9, we see the structure of these predictions. In figure 10, we find that the loss objective value for our y_hat predictions is 0.9579.

```
[('Prediction', -1),
 ('Prediction', -1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', -1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', 1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', 1),
 ('Prediction', 1),
 ('Prediction', 1),
 ('Prediction', -1),
 ('Prediction', -1),
...
```

Fig. 9 - Structure of prediction dataframe.

```
# For comparison, find the loss of our predictions
preds_df=preds.toDF().select(col('_2').alias('_c64'))
print('Loss of Predictions: ', loss_SVM(w_ps, bias_ps, X, preds_df,1))
```

✓ 13.2s

```
[Stage 101:=====>                                              (1 + 9) / 10]
Loss of Predictions:  0.9579132416284583
```

Fig. 10 - Loss objective value for y_hat predictions.