Comp 543 Assignment 5

Kai-Po Lin

Task 1:

【Result】

Train on Large Dataset:

applicant: 448

and: 2

attack: 514

protein: 3167

car: 652

[4]:
```python
# and we'll create a RDD that has a bunch of (word, dictNum) pairs
# start by creating an RDD that has the number 0 thru 20000
# 20000 is the number of words that will be in our dictionary
twentyK = sc.parallelize(range(20000))

# now, we transform (0), (1), (2), ... to ("mostcommonword", 0) ("nextmostcommon", 1), ...
# the number will be the spot in the dictionary used to tell us where the word is located
# A bunch of (word, posInDictionary) pairs
dictionary = twentyK.map(lambda x: (topWords[x][0], x))

# Collect the Rdd to a Dict
localDict = dictionary.collectAsMap()
for inputWord in ["applicant", "and", "attack", "protein", "car"]:
    if inputWord in localDict:
        print(f'{inputWord}: {localDict[inputWord]}')
    else:
        print(f'{inputWord}: -1')
```

▸ Spark Job Progress

```
applicant: 448
and: 2
attack: 514
protein: 3167
car: 652
```

Train on Medium Dataset:

applicant: 604

and: 2

attack: 515

protein: 3681

car: 635

```python
[4]: # and we'll create a RDD that has a bunch of (word, dictNum) pairs
# start by creating an RDD that has the number 0 thru 20000
# 20000 is the number of words that will be in our dictionary
twentyK = sc.parallelize(range(20000))

# now, we transform (0), (1), (2), ... to ("mostcommonword", 0) ("nextmostcommon", 1), ...
# the number will be the spot in the dictionary used to tell us where the word is located
# A bunch of (word, posInDictionary) pairs
dictionary = twentyK.map(lambda x: (topWords[x][0], x))

# Collect the Rdd to a Dict
localDict = dictionary.collectAsMap()
for inputWord in ["applicant", "and", "attack", "protein", "car"]:
    if inputWord in localDict:
        print(f'{inputWord}: {localDict[inputWord]}')
    else:
        print(f'{inputWord}: -1')
```

▸ Spark Job Progress

```
applicant: 604
and: 2
attack: 515
protein: 3681
car: 635
```

Task 2:

(a) Writing up your gradient update formula

【Result】

Gradient Function: $-\left(\sum_i y_i \cdot x_i - \frac{x_i \cdot e^{\sum_j x_{ij} \cdot w_j}}{1+e^{\sum_j x_{ij} \cdot w_j}} - \lambda \times 2 \times W\right)$

(W: weight, x: input data,
y: label)

(b) Giving us the fifty words with the largest regression coefficients

【Result】

Train on Large Dataset:

['that', 'not', 'any', 'court', 'mr', 'act', 'evidence', 'decision', 'whether', 'applicant', 'application', 'tribunal', 'claim', 'costs', 'matter', 'reasons', 'ltd', 'appeal', 'respondent', 'orders', 'respect', 'relation', 'relevant', 'appellant', 'sought', 'notice', 'circumstances', 'hearing', 'proceedings', 'matters', 'consider', 'pty', 'respondents', 'proceeding', 'regard', 'judgment', 'satisfied', 'submissions', 'affidavit', 'pursuant', 'fca', 'clr', 'relied', 'hca', 'discretion', 'fcr', 'alr', 'fcafc', 'relevantly', 'gummow']

```
[13]: idx = np.argpartition(w, -50)[-50:]
output = list()

for key, value in localDict.items():
    if value in idx:
        output.append(key)
print(output)

['that', 'not', 'any', 'court', 'mr', 'act', 'evidence', 'decision', 'whether', 'applicant', 'application', 'tribunal', 'claim', 'costs', 'matter', 'reasons', 'ltd', 'appeal', 'respondent', 'orders', 'respect', 'relation', 're
levant', 'appellant', 'sought', 'notice', 'circumstances', 'hearing', 'proceedings', 'matters', 'consider', 'pty', 'respondents', 'proceeding', 'regard', 'judgment', 'satisfied', 'submissions', 'affidavit', 'pursuant', 'fca',
'clr', 'relied', 'hca', 'discretion', 'fcr', 'alr', 'fcafc', 'relevantly', 'gummow']
```

Train on Medium Dataset:

['that', 'not', 'any', 'court', 'act', 'mr', 'evidence', 'decision', 'whether', 'tribunal', 'application', 'applicant', 'claim', 'matter', 'reasons', 'appeal', 'appellant', 'orders', 'relevant', 'ltd', 'sought', 'notice', 'circumstances', 'relation', 'hearing', 'proceedings', 'respondent', 'consider', 'matters', 'regard', 'proceeding', 'respondents', 'pty', 'judgment', 'satisfied', 'submissions', 'affidavit', 'magistrate', 'pursuant', 'fca', 'clr', 'hca', 'amp', 'discretion', 'fcr', 'alr', 'jurisdictional', 'relevantly', 'fcafc', 'gummow']

```
[13]: idx = np.argpartition(w, -50)[-50:]
output = list()

for key, value in localDict.items():
    if value in idx:
        output.append(key)
print(output)

['that', 'not', 'any', 'court', 'act', 'mr', 'evidence', 'decision', 'whether', 'tribunal', 'application', 'applicant', 'claim', 'matter', 'reasons', 'appeal', 'appellant', 'orders', 'relevant', 'ltd', 'sought', 'no
tice', 'circumstances', 'relation', 'hearing', 'proceedings', 'respondent', 'consider', 'matters', 'regard', 'proceeding', 'respondents', 'pty', 'judgment', 'satisfied', 'submissions', 'affidavit', 'magistrate', 'pu
rsuant', 'fca', 'clr', 'hca', 'amp', 'discretion', 'fcr', 'alr', 'jurisdictional', 'relevantly', 'fcafc', 'gummow']
```

Task 3:

【Result】

(a.) Test on Medium Dataset

TP: 328

TN: 18340

FP: 7

FN: 49

18668 out of 18724 correct.

Precision: 0.9791044776119403

Recall: 0.870026525198939

F1 Score: 0.9213483146067415

```
[37]: print(f'TP: {tp}')
      print(f'TN: {tn}')
      print(f'FP: {fp}')
      print(f'FN: {fn}')
```

```
TP: 328
TN: 18340
FP: 7
FN: 49
```

```
[38]: precision = tp / (tp + fp)
      recall = tp / (tp + fn)
      f1score = 2 * precision * recall / (precision + recall)
      print("%d out of %d correct." % (tp + tn, len(prediction)))
      print(f"Precision: {precision}")
      print(f"Recall: {recall}")
      print(f"F1 Score: {f1score}\n")
```

```
18668 out of 18724 correct.
Precision: 0.9791044776119403
Recall: 0.870026525198939
F1 Score: 0.9213483146067415
```

(b.) Test on Small Dataset:

TP: 67

TN: 3364

FP: 4

FN: 7

3431 out of 3442 correct.

Precision: 0.9436619718309859

Recall: 0.9054054054054054

F1 Score: 0.9241379310344827

```
[17]: print(f'TP: {tp}')
      print(f'TN: {tn}')
      print(f'FP: {fp}')
      print(f'FN: {fn}')
      print("%d out of %d correct." % (tp + tn, len(prediction)))
      print(f"Precision: {precision}")
      print(f"Recall: {recall}")
      print(f"F1 Score: {f1score}\n")
```


```
TP: 67
TN: 3364
FP: 4
FN: 7
3431 out of 3442 correct.
Precision: 0.9436619718309859
Recall: 0.9054054054054054
F1 Score: 0.9241379310344827
```

(c.) **3 examples of FP** => All False Positive in Large Dataset: Index = [1617, 3342, 12672, 13317, 14579, 14610, 17997].

I pick index 1617, 12672, and 13317as examples. Index 1617 is an article talking about “Removal jurisdiction”, index 12672 is an article talking about “Smit v Abrahams” (an important case in South African law), and index 13317 is an article talking about “Court of Appeal of Singapore”. I consider that **the words used in these articles will somewhat appear in the words used in Australian court cases**, and that may be the reason why the model will predict it as positive.