COMP 543 Assignment #5

Rung-De Chu (rc118)

Task 1

Note : I used SMALL DATA SET for this assignment.

In [ ]:
```
import re
import numpy as np


# Data Loading
corpus = sc.textFile("s3://chrisjermainebucket/comp330_A5/TestingDataOneLinePerDoc.txt")
#corpus = sc.textFile("s3://chrisjermainebucket/comp330_A5/SmallTrainingDataOneLinePerDoc.txt")

# each entry in validLines will be a line from the text file
validLines = corpus.filter(lambda x : 'id' in x)
# now we transform it into a bunch of (docID, text) pairs
keyAndText = validLines.map(lambda x :(x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">') + 2:]))
# now we split the text in each (docID, text) pair into a list of words
# after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
# we have a bit of fancy regular expression stuff here to make sure that we do not
# die on some of the documents
regex = re.compile('[^a-zA-Z]')
keyAndListOfWords = keyAndText.map(lambda x : (str(x[0]), regex.sub(' ',x[1]).lower().split()))
# now get the top 20,000 words... first change (docID, ["word1", "word2", "word3", ...])
# to ("word1", 1) ("word2", 1)...
allWords = keyAndListOfWords.flatMap(lambda x: ((j, 1) for j in x[1]))
# now, count all of the words, giving us ("word1", 1433), ("word2", 3423423), etc.
allCounts = allWords.reduceByKey (lambda a, b: a + b)
# and get the top 20,000 words in a local array
# each entry is a ("word1", count) pair
topWords = allCounts.top (20000, lambda x : x[1])
twentyK = sc.parallelize(range(20000))
dictionary = twentyK.map (lambda x : (topWords[x][0], x))
```

In [ ]:
```
word_to_find = ["applicant", "and", "attack", "protein", "car"]
positions = {word: dictionary.lookup(word)[0] if dictionary.lookup(word) else -1 for word in word_to_find}
print(positions)
```

Result of task 1 : using small training dataset :

```
plaintext
print(positions)
{'applicant': 347, 'and': 2, 'attack': 504, 'protein': 3018, 'car': 612}
```

using testing dataset :

```
plaintext
print(positions)
{'applicant': 604, 'and': 2, 'attack': 515, 'protein': 3681, 'car': 635}
```

Task 2

$LLH = \sum_i (y_i \theta^T x_i - \log(1 + e^{\theta^T x_i})), where y_i \in \{0, 1\}, x \in R^{20000}, \theta \in R^{20000}$

After regularization: $LLH = \sum_i (y_i \theta^T x_i - \log(1 + e^{\theta^T x_i})) + \frac{\beta}{2} ||\theta||_2^2$

$\frac{\partial LLH}{\partial \theta} = \sum_i (y_i x_i - \frac{e^{\theta^T x_i}}{1 + e^{\theta^T x_i}} x_i) + 2\beta ||\theta||_2$

In [ ]:
```
allWords = keyAndListOfWords.flatMap(lambda x: ((j, x[0]) for j in x[1]))
allDictionaryWords = dictionary.join(allWords)
doc_pos_pair = allDictionaryWords.map(lambda x: (x[1][1], x[1][0])).groupByKey()
docTerms = doc_pos_pair.map(lambda x: (x[0], list(x[1])))

def calculate_tf(word_indices):
    tf_vector = np.zeros(20000)
    for index in word_indices:
        tf_vector[index] += 1
    sum_tf = np.sum(tf_vector)
    return tf_vector / sum_tf if sum_tf > 0 else tf_vector

def calculate_df(docs):
```

```
    df_vector = np.zeros(20000)
    for word_indices in docs:
        unique_indices = set(word_indices)
        for index in unique_indices:
            df_vector[index] += 1
    return df_vector

def calculate_idf(df_vector, n_docs):
    return np.log(n_docs / (df_vector + 1))

n_docs = docTerms.count()
word_indices_docs = docTerms.map(lambda x: x[1]).collect()
df_vector = calculate_df(word_indices_docs)
idf_vector = calculate_idf(df_vector, n_docs)
tf_idf = docTerms.map(lambda x: (x[0], calculate_tf(x[1]) * idf_vector))

# Normalize
def normalize(data):
    global mean, std
    mean = data.map(lambda x: x[1]).mean()
    std = data.map(lambda x: x[1]).stdev()
    std = np.where(std == 0, 1, std)
    return data.map(lambda a: (a[0], (a[1] - mean) /std))

n = tf_idf.count()
allDocsAsNumpyArrays = normalize(tf_idf)

def cal_grad(x, r, beta):
    y_i = 1 if "AU" in str(x[0]) else 0
    net_i = np.dot(r, x[1])
    grad = -x[1] * y_i + x[1] * (np.exp(net_i) / (1 + np.exp(net_i))) + 2 * beta * r
    return grad

def cal_nllh(x, r):
    y_i = 1 if "AU" in str(x[0]) else 0
    net_i = np.dot(r, x[1])
    nll = -y_i * net_i + np.log(1 + np.exp(net_i))
    return nll

def gd_optimize(tf_idf, beta=0.0001, max_iter=100):
    r = np.zeros(20000)
    delta = 1
    lr = 1
    loss_now = tf_idf.map(lambda x: cal_nllh(x, r)).reduce(lambda a, b: a + b) + beta * np.linalg.norm(r) ** 2
```

```
    num_epoch = 0
    while delta > 0.0001 and num_epoch < max_iter:
        num_epoch += 1
        grad = tf_idf.map(lambda x: cal_grad(x, r, beta)).reduce(lambda a, b: a + b) / tf_idf.count()
        r -= lr * grad
        loss_next = tf_idf.map(lambda x: cal_nllh(x, r)).reduce(lambda a, b: a + b) + beta * np.linalg.norm(r) ** 2
        delta = abs(loss_next - loss_now)
        print(f"Epoch: {num_epoch}, Negative Log Likelihood: {loss_next}")
        lr = lr/ 2 if loss_next > loss_now else lr * 1.1
        loss_now = loss_next
    return r


w = np.zeros(20000)
w = gd_optimize(allDocsAsNumpyArrays, 0.01)
```

In [ ]:
```
w_dict_reverse = dictionary.map(lambda x: (x[1], x[0])).sortByKey()
top_50_indices_desc = np.argsort(w)[-50:][::-1]
for idx in top_50_indices_desc:
    print(w_dict_reverse.lookup(idx)[0], "'s coefficient:", w[idx])
```

Term	Coefficient
applicant	0.4313592677577349
clr	0.4238193258068558
pty	0.4194318653850585
fcr	0.41223204589155593
hca	0.39654863901192233
alr	0.3823961644927237
respondent	0.36602988041261597
relevantly	0.34917705762561213
fca	0.3461214447536104
tribunal	0.3390207563746712
submissions	0.3386263435487898
fcafc	0.32447521140336827

Term	Coefficient
gummow	0.31547952719345107
affidavit	0.3146678362297677
pursuant	0.3071781278022647
proceeding	0.30181605547601065
application	0.2980407215956911
satisfied	0.29590223530279197
appellant	0.2796192834484266
respondents	0.24148047156089478
submits	0.2362140328839448
gaudron	0.22605320868349993
amp	0.22412108459502442
costs	0.22284432651921957
proceedings	0.22184471626512264
reasons	0.22067288656993814
jurisdictional	0.2128643122123014
relevant	0.20574439554196747
nswlr	0.20459562853812024
mr	0.2035292721402673
magistrate	0.20284850617512765
affidavits	0.20118799525813258
interlocutory	0.1973322594282697
notice	0.19438038891134007
hearing	0.1925708019227608
circumstances	0.19208673050126412
relation	0.19149283945333015

Term	Coefficient
multicultural	0.18995626017913783
arguable	0.18827129515627228
contravention	0.18800941016214426
evidence	0.1855428995385597
honour	0.18255506852887857
magistrates	0.17449152918814753
acsr	0.17259076559798467
applicants	0.17176509004733795
orders	0.1680815733885168
paragraphs	0.16607556863724812
erred	0.16387846118789004
ltd	0.16140056274421033
contraventions	0.16098774942057018

Task 3

In [ ]:

```
# Data Loading
pcorpus = sc.textFile("s3://chrisjermainebucket/comp330_A5/SmallTraingDataOneLinePerDoc.txt")
# each entry in validLines will be a line from the text file
pvalidLines = pcorpus.filter(lambda x : 'id' in x)
# now we transform it into a bunch of (docID, text) pairs
pkeyAndText = validLines.map(lambda x :(x[x.index('id="') + 4 : x.index('" url=')], x[x.index('">') + 2:]))
# now we split the text in each (docID, text) pair into a list of words
# after this, we have a data set with (docID, ["word1", "word2", "word3", ...])
# we have a bit of fancy regular expression stuff here to make sure that we do not
# die on some of the documents
pkeyAndListOfWords = pkeyAndText.map(lambda x : (str(x[0]), regex.sub(' ',x[1]).lower().split()))
# now get the top 20,000 words... first change (docID, ["word1", "word2", "word3", ...])
# to ("word1", 1) ("word2", 1)...
pallDictionaryWords = dictionary.join (allWords)
pdoc_pos_pair = pallDictionaryWords.map (lambda x: (x[1][1], x[1][0])).groupByKey ()
pdocTerms = pdoc_pos_pair.map(lambda x: (1 if x[0][0] == "A" else 0, x[1]))
```

```
n_docs = pdocTerms.count()
p_word_indices_docs = pdocTerms.map(lambda x: x[1]).collect()
pdf_vector = calculate_df(p_word_indices_docs)
pidf_vector = calculate_idf(pdf_vector, n_docs)
p_tf_idf = pdocTerms.map(lambda x: (x[0], calculate_tf(x[1]) * pidf_vector))
pallDocsAsNumpyArrays = p_tf_idf.map(lambda a: (a[0], (a[1] - mean)/ std))
```

In [ ]:
```
def predict(test, r, cut =0):
    y_true_pred = test.map(lambda x: (x[0], 1 if r.dot(x[1]) > cut else 0))
    res = np.array(y_true_pred.collect())
    tp, tn, fp, fn = 0, 0, 0, 0
    for idx in range(res.shape[0]):
        if ((res[idx,0] == 1) and (res[idx,1] == 1)):
            tp += 1
        elif ((res[idx,0] == 0) and (res[idx,1] == 0)):
            tn += 1
        elif ((res[idx,0] == 1) and (res[idx,1] == 0)):
            fn += 1
        elif ((res[idx,0] == 0) and (res[idx,1] == 1)):
            fp += 1
            print("fp index:", idx)
    accuracy = (tp + tn)/(tp + tn + fp + fn) if (tp + tn + fp + fn) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    precision = tp/ (tp + fp) if    (tp + fp) > 0 else 0
    F1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0
    print("%d out of %d predictions correct." % (tp + tn, len(res)))
    print(f"test accuracy: {accuracy:.5f}")
    print(f"recall: {recall:.5f}")
    print(f"precision: {precision:.5f}")
    print(f"F1 score: {F1_score:.5f}")
```

In [ ]:
```
predict(pallDocsAsNumpyArrays, w, cut = 10)
```

predict(pallDocsAsNumpyArrays, w, cut = 10)

fp index: 96

fp index: 571

fp index: 2897

fp index: 4401

fp index: 6830

fp index: 7668

fp index: 9092

fp index: 12371

fp index: 13796

fp index: 17413

fp index: 17420

18713 out of 18724 predictions correct.

test accuracy: 0.99941

recall: 1.00000

precision: 0.97165

F1 score: 0.98562

I used a lookup tool to examine the index above. The Wikipedia documents contain terms related to labor and affairs, and they share some of the top 50 listed words.