Programming Project 4

Tanmayi Balla

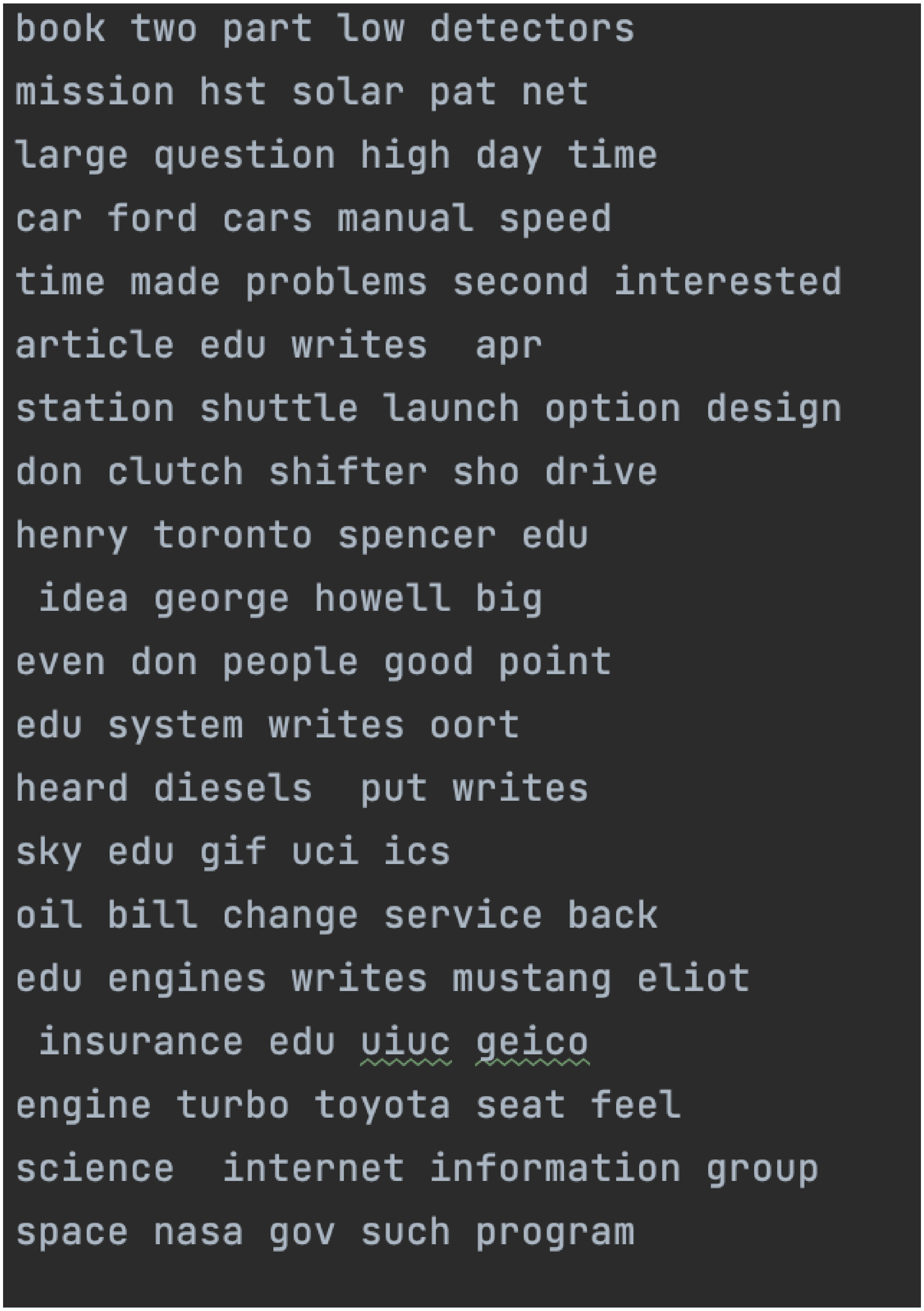
December 10, 2022

Task-1

Implementation Overview

- The main objective of this task is to implement the collapsed Gibbs sampler using Latent Dirichlet Allocation (LDA), i.e., to classify the words in the documents into K topics and create a feature vector with reduced dimensions, to perform classification in Task-2.
- This task also outputs the most frequent words generated from the sampler, for each topic.
- The main function in Task1 is the **GibbsSampler()** with input parameters: **K, alpha, beta, N, words, documents, topics, D, dataset**. Here, K is the number of topics, alpha beta are the Dirichlet parameters, N is 500 (Iterations), words is a list of all the words present in the corpus, and documents is a list containing the document to which words[i] belong. Topics is a randomly generated list, where we allocate random topics for each word in words list. D is a list of the document numbers in the dataset, while dataset is the name of the dataset (can be either 'artificial' or 'newsgroups'.
- There is a helper function for task1, that generates all the required lists, that needs to be fed into the GibbsSampler.
- Task-1 has a runtime of around 30 minutes.
- I have run Task-1 many times, and I am presenting the topics.csv file that I felt is more meaningful.
- All the helper functions are placed in utils.py.

Results of Task-1


```
book two part low detectors
mission hst solar pat net
large question high day time
car ford cars manual speed
time made problems second interested
article edu writes  apr
station shuttle launch option design
don clutch shifter sho drive
henry toronto spencer edu
 idea george howell big
even don people good point
edu system writes oort
heard diesels  put writes
sky edu gif uci ics
oil bill change service back
edu engines writes mustang eliot
 insurance edu uiuc geico
engine turbo toyota seat feel
science  internet information group
space nasa gov such program
```

Inference

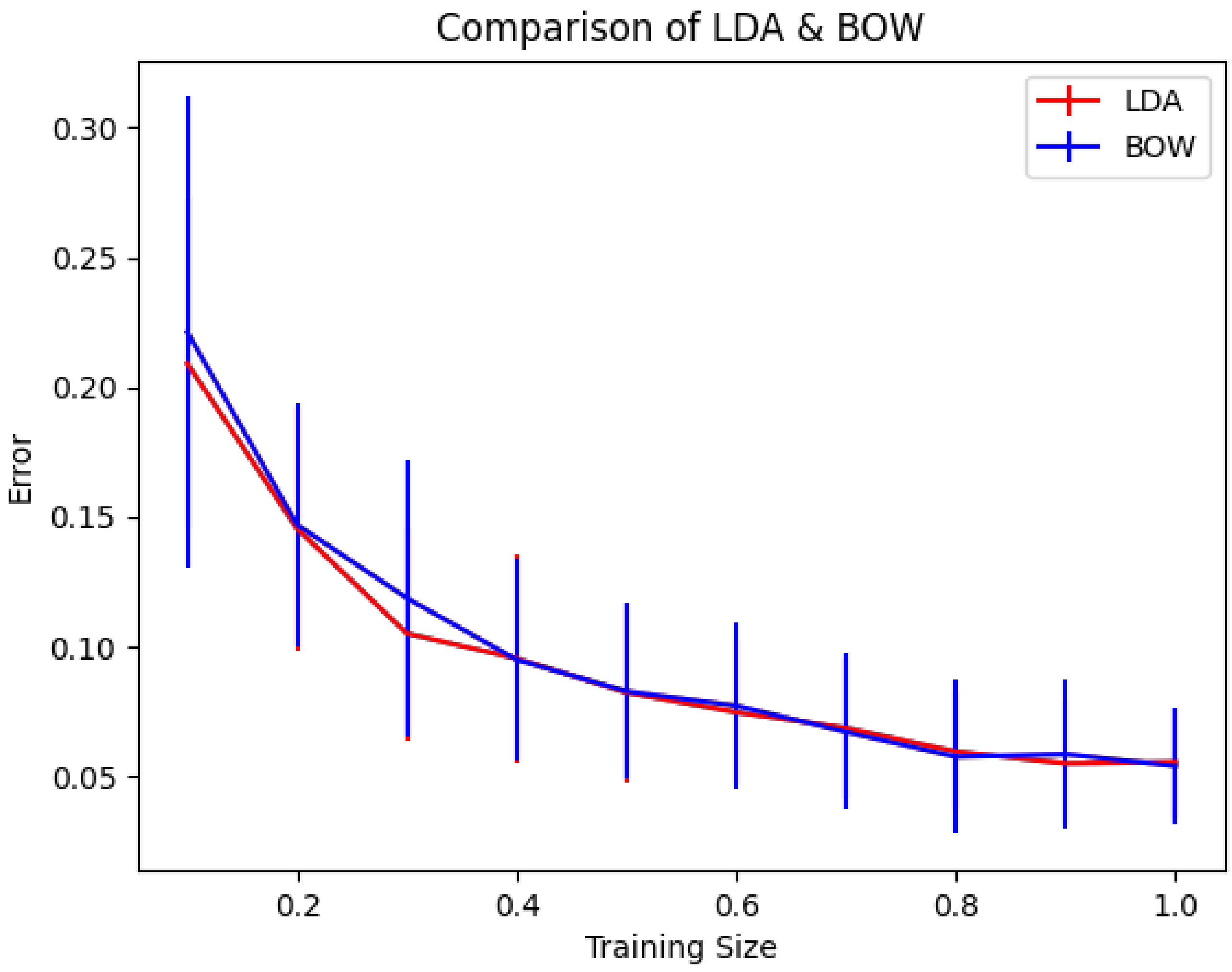
1. The top 5 frequent words of 20 topics are given in the above image. If we closely observe the words for each topics, the combination of words is meaningful (related to news articles). For Example, the last topic is above a space program by nasa; we have a row corresponding to vehicles (engines, toyota, seat, etc.). Though all the rows cannot be considered accurate, the algorithm has managed to give some useful features for classification. Since, the number of features now is reduced to K (no.of topics), the execution would be much faster, while predicting the label in Task-2.

Task-2

Implementation Overview

- The main objective of this task is to compare topic classification for features generated from LDA (Task-1), and Bag-of-words (BOW).
- The features generated from task-1 are stored in f1.npy file. This same file is imported in Task-2 to compare the performace.
- Task-2 is basically a repetition of PP3 Task. It contains all the functions required to generate weights using Newton's method and make predictions using Logistic Regression.
- Task2Helper is used to randomly select the training set and convert it into the respective 10 splits ([0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]). Predictions are generated for all the 10 train splits, and the process is repeated for 30 iterations.
- The mean of the errors for all the training sizes and standard deviation are used to plot the learning curves.
- The standard deviation for both LDA and BOW is very negligible. Hence, the error bars are not properly visible in the plot.

Results



It is evident from the graph that, even with reduced dimensionality, i.e., less number of features, we are able to classify the documents, with good accuracy.

Also, while execution, Bag of words data took higher time as compared to LDA. And, LDA is able to match the accuracy of BOW for all the training sizes. Time taken for LDA is 2.39 seconds, while the runtime of BOW is 77 seconds.

We can observe that the learning curves for both LDA and BOW are similar, and they are reduced with increased training size. This is apparent since a larger dataset provides better learning and therefore better results.

These observations conclude that LDA is a better and faster alternative for document classification.

```python
import random

import numpy as np
import pandas as pd

artificial = './pp4data/artificial/'
newsgroups = './pp4data/20newsgroups/'
task_1_fm = []
art_D = 10
news_D = 200

def Gibbs_Sampler(K,alpha,beta,N,words,documents,topics,D,dataset):
    ## Construct Cd matrix D*K (Documents * Topics)
    CD = [[0 for i in range(K)] for j in range(len(D))]
    CT = [[0 for i in range(K)] for j in range(len(words))]
    ind = 0
    prob = np.array([0.0 for i in range(K)])
    for i in D:
        with open(dataset + str(i)) as doc:
            text = doc.readlines()[0]
            for word in text.split(' '):
                CD[i-1][topics[ind]]+=1
                ind+=1
    #print(CD)
    word_topic_dict = {}
    #beta = {}
    for i in range(len(words)):
        word = words[i]
        topic = topics[i]
        #print(word)
        if word in word_topic_dict.keys():
            word_topic_dict[word][topic]+=1
        else:
            word_topic_dict[word] = [0 for i in range(K)]
            word_topic_dict[word][topic] += 1
    CT = word_topic_dict.copy()
    topic_choices = list(range(0,K))

    for i in range(N):
        print("Iteration: ",i)
        phi_n = np.random.permutation(len(words))
        for l in range(len(words)):
            j = phi_n[l]
            #print(j)
            word = words[j]
            topic = topics[j]
            document = documents[j]-1
            CD[document][topic]-=1
            CT[word][topic]-=1
            V = len(CT)

            for k in range(K):
                sum_ct = 0
                for key, val in CT.items():
                    sum_ct += val[k]
                sum_cd = CD[document][0] + CD[document][1]
                prob[k] = ((CT[word][k] + beta)*(CD[document][k] + alpha))/((V*beta + sum_ct) * (K*alpha + sum_cd))
            prob_nrm = prob/np.linalg.norm(prob)

            new_topic = random.choices(topic_choices,prob_nrm)[0]
            #print(new_topic)
            CD[document][new_topic]+=1
            CT[word][new_topic]+=1
            topics[j] = new_topic
            #print(CT)

    # Output 5 most frequent words for a topic:
    topic_dict = {}
    for i in range(K):
        topic_dict[i] = []
    for i in CT.keys():
        for j in range(K):
            topic_dict[j].append([CT[i][j],i])
    res = []
    #print("Topics dict")
    #print(topic_dict)
    for i in topic_dict.keys():
        words_topic = topic_dict[i]
        #print(words_topic)
        words_topic.sort(reverse = True)
        #print(words_topic)
        line = ""
        #print(words_topic)
        for k in range(5):
            line=line+words_topic[k][1]+" "
        res.append(line)
    df = pd.DataFrame(res)
    df.to_csv('topicwords.csv',index = False, header = False)

    ## Creating the feature vector
    D_feature = []
    for i in D:
        doc_fm = []
        for j in range(K):
            #print(CD[i-1],"cd")
            doc_fm.append((CD[i-1][j] + alpha)/(K*alpha + sum(CD[i-1])))
            #print(sum(CD[i-1]),"cdsum")
        D_feature.append(doc_fm)
    task_1_fm = D_feature
    nparr = np.array(task_1_fm)
    np.save('f1.npy', nparr)
    #print(nparr)
    return

def task1_helper(dataset,K,d):
    D = list(range(1,d+1))
    topic_choices = list(range(0,K))

    words = []
    topics = []
    documents = []
    for i in D:
        with open(dataset + str(i)) as doc:
            text = doc.readlines()[0]
            for word in text.split(' '):
                words.append(word)
                t = np.random.choice(topic_choices)
                topics.append(t)
                documents.append(i)
    Gibbs_Sampler(K,5/K,0.01,500,words,documents,topics,D,dataset)


if __name__ == '__main__':
    task1_helper(newsgroups, 20, 200)
```

```python
import random

import numpy as np
import pandas as pd
import datetime

from utils import train_test_split_random as ttsr
from utils import Newtons_Update as NU
from utils import BLR as blr
from utils import sample_trainR as train_sample
from utils import plotresults as plt_res
from utils import get_words

artificial = './pp4data/artificial/'
newsgroups = './pp4data/20newsgroups/'


def task2_helper(X,Y):

    #print(train_x,train_y)
    N = 30
    err_iter = []
    err_map = {}
    for i in range(0,10):
        err_map[i] = []
    while(N>0):
        train_x, train_y, test_x, test_y = ttsr(X, Y, 2 / 3)
        train_set = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
        err_split = []
        ind = 0
        for split in train_set:
            x_train,y_train = train_sample(train_x,train_y,split)
            weights = NU(x_train,y_train,0.01)
            err_map[ind].append(blr(x_train,y_train,test_x,test_y,weights))
            ind+=1
        #err_iter.append(err_split)
        print("Processing Iteration: ",N)
        N-=1
    return err_map

def task2(dataset,K,d):
    words = get_words(dataset,d)
    D = list(range(1,d+1))

    ## Generating features using Bag of Words representation
    # vocabulary
    vocab = {}
    ind = 0
    for i in words:
        if i not in vocab:
            vocab[i] = ind
            ind+=1
    #print(vocab)
    BOW = []
    for i in D:
        bow = [0 for i in range(len(vocab))]
        with open(dataset + str(i)) as doc:
            text = doc.readlines()[0]
            for word in text.split(' '):
                bow[vocab[word]]+=1
        BOW.append(bow)

    ## Topic features generated by Task-1

    l_np = np.load('f1.npy')
    new_list = []
    for i in l_np:
        new_list.append(list(i))
    Y = pd.read_csv(dataset+'index.csv',header = None,index_col=False)
    Y = Y[1]

    # Implementing Newton's method for both the representations
    #print(new_list)
    '''
    ##################

    # This is a list of features generated from Task-1. We can uncomment this line to directly run task-2, without running Task-1.


    ##################
    '''
    print("-------------LDA----------------")
    time_start_lda = datetime.datetime.now()
    err_LDA = task2_helper(new_list,Y)
    time_end_lda = datetime.datetime.now()

    print("-------------BOW----------------")
    time_start_bow = datetime.datetime.now()
    err_bow = task2_helper(BOW,Y)
    time_end_bow = datetime.datetime.now()

    meanLDA = []
    meanBOW = []
    stdLDA = []
    stdBOW = []

    for i in range(0, 10):
        meanLDA.append(np.mean((err_LDA[i])))
        meanBOW.append(np.mean((err_bow[i])))

        stdLDA.append(np.std(err_LDA[i]))
        stdBOW.append(np.std(err_bow[i]))

    training_size = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    plt_res(training_size, meanLDA, stdLDA, meanBOW, stdBOW)

    print("Time taken for LDA: ",time_end_lda - time_start_lda)
    print("Time taken for BOW:",time_end_bow-time_start_bow)
    return

if __name__ == '__main__':
    task2(newsgroups, 20, 200)
```

Programming Project - 4

Author: Tanmayi Balla

Installing the required packages

```
 # Install pandas

pip install pandas

# Install matplotlib

pip install matplotlib

# Install numpy

pip install numpy

# Install datetime

pip install datetime

It is assumed that the packages: math, random, and warnings are generally pre-installed. In case they
```

Task-1:

```
python task1.py
```

The required output is printed in the terminal. Top 5 words for each topic are saved in the same directory, with name 'topicwords.csv'. The feature list is stored as 'f1.npy', which is later used by Task-2 for classification.

Task-2:

```
python task2.py
```

The required output is printed in the terminal. The plots for error vs training size, for LDA and Bag-of-words, for the newsgroups dataset, are saved in the same directory, with name "Task2.png". This imports the 'f1.npy' file stored from the previous task.

Note:

These Python files were tested in PyCharm in MAC. Tabulate package is used to display the required output in the terminal. It is assumed that the packages: math, random, and warnings are generally pre-installed. In case they are not, kindly install them before testing the code.