COMPSCI 546: Applied Information Retrieval - Spring 2023

Assignment 7: Learning to Rank & Neural Ranking Model (Total : 100 points)

Description

This assignment consists of programming and analytical questions on Learning to Rank models and neural ranking models. Basic proficiency in Python is recommended.

Instructions

- To start working on the assignment, you would first need to save the notebook to your local Google Drive. For this purpose, you can click on *Copy to Drive* button. You can alternatively click the *Share* button located at the top right corner and click on *Copy Link* under *Get Link* to get a link and copy this notebook to your Google Drive.
- For questions with descriptive answers, please replace the text in the cell which states "Enter your answer here!" with your answer. If you are using mathematical notation in your answers, please define the variables.
- You should implement all the functions yourself and should not use a library or tool for the computation.
- For coding questions, you can add code where it says "enter code here" and execute the cell to print the output.
- To create the final pdf submission file, execute *Runtime->RunAll* from the menu to re-execute all the cells and then generate a PDF using *File->Print->Save as PDF*. Make sure that the generated PDF contains all the codes and printed outputs before submission. To create the final python submission file, click on File->Download .py.

Submission Details

- Due data: May 19, 2023 at 9:00 PM (ET).
- The final PDF and python file must be uploaded on Gradescope.
- Please make sure you submit both PDF and python file ***You will not recieve any credit if you don't submit your code!*** Make sure we can access your code.

Academic Honesty

Please follow the guidelines under the *Collaboration and Help* section of the course website.

Download input files

Please execute the cell below to download the input files.

```
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
```

```
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


import os
import zipfile
import numpy as np

download = drive.CreateFile({'id': '1Se44iDgNbvnuXVoB3i3Fp8rltw8nKEqw'})
download.GetContentFile('HW07.zip')

with zipfile.ZipFile('HW07.zip', 'r') as zip_file:
    zip_file.extractall('./')

os.remove('HW07.zip')
# We will use hw07 as our working directory
os.chdir('HW07')

#Setting the input files
passage_file = "psgs.txt"
test_queries_file = "test-queries.txt"
train_queries_file = "train-queries.txt"
val_queries_file = "val-queries.txt"
stopwords_file = "stopword.txt"
val_baseline_features_file = "val-features.txt"
test_baseline_features_file = "test-features.txt"
train_baseline_features_file = "train-features.txt"
```

1 : Initial Data Setup (20 points)

We use the MS MARCO dataset for this assignment. As described in the previous assignments, this is a passage retrieval dataset.

The description of the input files provided for this assignment is given below.

Collection file

Each row of the file consists of the following information:

passage_id passage_text

The id and text information is tab separated. The passage text has been pre-processed to remove punctutation, tokenized and stemmed using the Krovetz stemmer. The terms in the passage text can be accessed by splitting the text based on space.

Query files

You are provided with train, validation, and test query files. Each row of the file consists of the following information:

query_id query_text

The id and text information is tab separated. The query text has been pre-processed to remove punctutation, tokenised and stemmed using the Krovetz stemmer. The terms in the text can be accessed by splitting the

text based on space.

Feature files

You are provided with train, validation, and test feature files. Each row of the file consists of the following information:

query_id passage_id relevance_label vsm_score bm25_score

Each row contains features for a (query, passage) pair and is space separated. The relevance_label is the human annotated relevance judgment. vsm_score and bm25 scores are the relevance features for the pair corresponding to the two different scoring methods.

Stopwords file

The stopword file contains the list of stopwords. This file has a stopword per line.

Sample file

For this assignment, we use the pyltr Learning to Rank framework. The input file to the framework has to be set similar to the sample.txt in the following format:

relevance_label qid:query_id 1:feature1 2:feature2 #docid = passage_id

Each entry is space separated. This file has been provided only for reference to create files of the same format.

In the cell below, you have to implement the following:

- Load the query files
- Load the collection
- Load the stopwords

```
'''
In this function, load the query files into dict
Return Variables:
queries - dict with qid as key and querytext as value
'''
def loadQueryFile(query_file):
    #enter your code here
    queries={}
    for line in open(query_file):
      row=line.strip().split("\t")
      queries[row[0]]=row[1]
    return queries



'''
In this function, load the collection into dict
Return Variables:
coll - dict with passage id as key and passage text as value
'''
def loadCollection(passage_file):
    #enter your code here
    coll={}
    for line in open(passage_file):
      row=line.strip().split("\t")
      coll[row[0]]=row[1]
    return coll
```

```
'''
In this function, load the stopwords into set
Return Variables:
stopwords - set of stopwords
'''
def loadStopWords(stopwords_file):
    # enter your code here
    stopwords=set()
    for line in open(stopwords_file):
      row=line.strip()
      stopwords.add(row)
    return stopwords


train_queries = loadQueryFile(train_queries_file)
val_queries = loadQueryFile(val_queries_file)
test_queries = loadQueryFile(test_queries_file)
coll = loadCollection(passage_file)
stopwords = loadStopWords(stopwords_file)

print('Total Number of train queries: {0}'.format(len(train_queries)))
print('Total Number of validation queries: {0}'.format(len(val_queries)))
print('Total Number of test queries: {0}'.format(len(test_queries)))
print('Total Number of passages in the collection: {0}'.format(len(coll)))
print('Total Number of stopwords: {0}'.format(len(stopwords)))
```

```
Total Number of train queries: 130
Total Number of validation queries: 10
Total Number of test queries: 10
Total Number of passages in the collection: 31173
Total Number of stopwords: 418
```

2 : Feature Preparation (20 points)

The input feature file consists of two main features : VSM score and bm25 score of a query and passage pair. In this section, you will implement three additional features and use the information to create input feature file which contains the 5 features. The feature file must have the same format as sample.

In the cell below, implement the following features:

- Number of unique term overlap between query and passage after excluding stopwords and words with only one character from both.

  [Example = Query : why do a cat headbutt

  Passage : cat fight for attention and domination if you show a can of food to my cat he headbutt it.

  Number of Overlapped terms for the query/passage pair: 2]
- Number of terms in query
- Number of terms in passage

```
'''
In this function, create new feature file with additional features in the format required as
Return Variables:
```

```
There is no return variable. You would create a new feature file "final_features_file"
One example of the row of the newly created file is
"0 qid:3990512 1:3.5053628162466897 2:10.841493137122296 3:1 4:6 5:112 #docid = 882429_10"
The format of the file is:
"relevance_score qid:query_id 1:feature1 2:feature2 3:feature3 4:feature4 5:feature5 #docid =
You can read through the input baseline_features_file, create additional features and
add the updated entry into the new file.
'''
from collections import Counter

def featureCreation(baseline_features_file, stopwords, queries, coll, final_features_file):
    #enter your code here
    f = open(final_features_file, "w")
    a=""
    for line in open(baseline_features_file):
      row = line.strip().split(" ")
      queryCounter =Counter(queries[row[0]])
      docCounter = Counter(coll[row[1]])
      c=0
      qterms=0
      passterms=0
      for i in queryCounter.keys():
        if i in docCounter.keys() and i not in stopwords:
          c+=1
        if i not in stopwords:
          qterms+=1
      for i in docCounter.keys():
        if i not in stopwords:
          passterms+=1
      a+=row[2]+" "+ "qid:"+row[0] + " 1:" + row[3] + " 2:"+row[4] + " 3:"+str(c) + " 4:"+ st
    f.write(a)
featureCreation(train_baseline_features_file, stopwords, train_queries, coll, 'train-features
featureCreation(val_baseline_features_file, stopwords, val_queries, coll, 'val-features-final
featureCreation(test_baseline_features_file, stopwords, test_queries, coll, 'test-features-fi
```

3 : Model Training and Evaluation (20 points)

In the cell below, the pyl2r library is used to train and evaluate a ranking model on MS MARCO data using LambdaMART model. This should take a couple of minutes to execute.

```
import warnings
warnings.filterwarnings('ignore')
import pyltr

with open('train-features-final.txt') as trainfile, open('val-features-final.txt') as valifil
    TX, Ty, Tqids, _ = pyltr.data.letor.read_dataset(trainfile)
    VX, Vy, Vqids, _ = pyltr.data.letor.read_dataset(valifile)
    EX, Ey, Eqids, _ = pyltr.data.letor.read_dataset(evalfile)


metric = pyltr.metrics.NDCG(k=10)
monitor = pyltr.models.monitors.ValidationMonitor(VX, Vy, Vqids, metric=metric, stop_after=25

model = pyltr.models.LambdaMART(
    metric=metric,
    n_estimators=100,
```

```
    learning_rate=0.02,
    max_features=0.5,
    query_subsample=0.5,
    max_leaf_nodes=10,
    min_samples_leaf=64,
    verbose=1,
)
model.fit(TX, Ty, Tqids,monitor=monitor)
Epred = model.predict(EX)
print ('LambdaMART model test score :'+ str(metric.calc_mean(Eqids, Ey, Epred)))
```

```
      Iter  Train score  OOB Improve    Remaining                          Monitor Output
         1       0.1189       0.0895        1.51m      C:      0.0855 B:      0.0855 S:  0
         2       0.1019       0.0039        1.65m      C:      0.0980 B:      0.0980 S:  0
         3       0.1433       0.0135        1.67m      C:      0.0934 B:      0.0980 S:  1
         4       0.1805       0.0106        1.69m      C:      0.0943 B:      0.0980 S:  2
         5       0.1613       0.0037        2.01m      C:      0.1175 B:      0.1175 S:  0
         6       0.1740       0.0139        2.08m      C:      0.1187 B:      0.1187 S:  0
         7       0.1718       0.0124        2.02m      C:      0.1106 B:      0.1187 S:  1
         8       0.1799       0.0010        1.98m      C:      0.1099 B:      0.1187 S:  2
         9       0.1947       0.0127        1.94m      C:      0.1099 B:      0.1187 S:  3
        10       0.1972       0.0051        1.90m      C:      0.1101 B:      0.1187 S:  4
        15       0.2520       0.0033        1.94m      C:      0.2178 B:      0.2178 S:  0
        20       0.3440       0.0023        1.81m      C:      0.2134 B:      0.2178 S:  5
        25       0.3146       0.0026        1.78m      C:      0.2146 B:      0.2178 S: 10
        30       0.3255       0.0018        1.66m      C:      0.2219 B:      0.2219 S:  0
        35       0.3356      -0.0003        1.60m      C:      0.2357 B:      0.2357 S:  0
        40       0.3339       0.0016        1.51m      C:      0.2219 B:      0.2357 S:  5
        45       0.3688       0.0001        1.38m      C:      0.2264 B:      0.2357 S: 10
        50       0.3827       0.0001        1.27m      C:      0.2385 B:      0.2385 S:  3
        60       0.3243       0.0062        1.02m      C:      0.2786 B:      0.2786 S:  0
        70       0.4060      -0.0001       46.85s      C:      0.2959 B:      0.2959 S:  1
        80       0.3260      -0.0010       31.48s      C:      0.2966 B:      0.3017 S:  3
        90       0.3395      -0.0000       15.79s      C:      0.2995 B:      0.3017 S: 13
    Early termination at iteration  99
    LambdaMART model test score :0.2677084864774112
```

3.1 : Briefly describe how listwise loss functions work. (10 points)

Enter your answer here!

The listwise loss function is defined for a query and a list of documents.It take into account the entire list and its ranking order rather than focusing on individual pairwise comparisons. They measure the discrepancy between the predicted ranking and the ground truth ranking by considering the entire list as a whole.

3.2 : Can we directly optimize **typical** ranking metrics, such as NDCG or MAP? If yes, how? If no, why? (10 points)

Enter your answer here!

Yes, it is possible to directly optimize typical ranking metrics such as NDCG (Normalized Discounted Cumulative Gain) or MAP (Mean Average Precision), but there are considerations and challenges involved in doing so. It often requires approximations, surrogate objectives, or specific learning to rank algorithms.

4 : Re-Ranking with BERT (40 points)

Assume that instead of manual feature engineering and using LambdaMART, you are asked to design a re-ranking model using BERT. Unfortunately, successful training of most neural ranking models for ranking would take multiple hours. Therefore, this part of the assignment does not contain coding questions. You are highly encouraged to implement a BERT Re-Ranking model for the MS MARCO dataset, if you are interested in learning more about neural ranking models. HuggingFace provides nice tools for implementing Transformer-based models.

Answer the following questions.

4.1 : Describe your model architecture with all the details, including the model's input and the network architecture. You can describe the architecture in words, or insert an image containing the model architecture. (15 points)

Enter your answer here!

Input: query-document pair

query and document tokenized using BERT tokenizer

CLS and SEP added to mark beginning and seperation of query and document

input passed throught BERT model

BERT encoder captures contextualized representations of the tokens.

Output pooled to obtain fixed size representation of query and coument using mean or max pooling

Query and document is compared using similarity function like cosine similarity

Similarity scores used to rerank based on relevance

4.2 : Write down a pointwise and a pairwise loss function of your choice for training the model. Which one do you expect perform better and why? (15 points)

Enter your answer here!

Pairwise loss function: hinge loss -> encourage relevant documents have higer similarity score than non relevant documents. Penalizes pairs where the similarity score violates the margin between relevant and non-relevant pairs.

Pointwise loss function: Means quared error loss -> Measures difference between predicted scores and relevance lebels ground truth

4.3 : If you want to use your model for a retrieval setting (instead of re-ranking a small number of documents), how do you design the model? Just a high-level description (a few sentences) would be sufficient. (10 points)

Enter your answer here!

Instead of individual query-document pairs, we can use batch queries and a set of documents

Padding can be used to ensure uniform input lengths within a batch.

Output pooled to obtain fixed-size representations for each query and document in the batch.

Pairwise similarity scores computed between each query and document pair in the batch

✓ 2m 40s completed at 19:05