**IR Competition Project Documentation**

Cody Webster

Contents

[**Overview** 3](#_Toc58705428)

[**Project Results** 3](#_Toc58705429)

[**DirichletPrior** 3](#_Toc58705430)

[**JelinekMercer** 3](#_Toc58705431)

[**OkapiBM25 (variation in weighting parameters not captured)** 3](#_Toc58705432)

[**BM25+ (variations in parameters for each elaborated)** 4](#_Toc58705433)

[**BERT** 4](#_Toc58705434)

[**SciBERT \*difficulty running this model limited options** 4](#_Toc58705435)

[**Tensorflow Ranking custom model** 5](#_Toc58705436)

[**Assumptions** 5](#_Toc58705437)

[**Installation Instructions** 5](#_Toc58705438)

[**What to install if running locally:** 6](#_Toc58705439)

[**What python packages to install** 6](#_Toc58705440)

[**How to Run** 6](#_Toc58705441)

[**Prepare Data (Required if local):** 6](#_Toc58705442)

[**(Method 1) Run BM25+ or another ranker:** 6](#_Toc58705443)

[Option 1 (Google Colab) 6](#_Toc58705444)

[Option 2 (run locally) 6](#_Toc58705445)

[**(Method 2) Run Tensorflow Ranking** 6](#_Toc58705446)

[Option 1 (Google Colab) 6](#_Toc58705447)

[Option 2 (run locally) 6](#_Toc58705448)

[**(Method 3) Run a BERT model** 6](#_Toc58705449)

[Option 1 (Google Colab) 6](#_Toc58705450)

[Option 2 (run locally) 6](#_Toc58705451)

[**Script API Documentation** 6](#_Toc58705452)

[**General** 6](#_Toc58705453)

[**Metapy (BM25, …)** 6](#_Toc58705454)

[**Tensorflow Ranking** 6](#_Toc58705455)

[**BERT** 6](#_Toc58705456)

[**Resources** 6](#_Toc58705457)

[**Citations** 6](#_Toc58705458)

[**Installation Guidance** 6](#_Toc58705459)

[**Stopwords** 6](#_Toc58705460)

[**BM25** 6](#_Toc58705461)

[**Tensorflow Ranking** 6](#_Toc58705462)

[**Tensorflow Ranking (how to serve a model)** 6](#_Toc58705463)

[**Tensorflow Ranking (how to predict)** 6](#_Toc58705464)

[**BERT Checkpoint Conversion** 6](#_Toc58705465)

[**BERT** 6](#_Toc58705466)

[**SciBERT** 6](#_Toc58705467)

[**Docker** 6](#_Toc58705468)

# **Overview**

The purpose of this project is to participate in the IR Competition. For this project I developed and extensively tested three main ways to rank documents. The first is ranking documents using a ranking function either packaged with metapy or through a custom definition implemented through metapy. The best results that I achieved with this method were through a custom implementation of BM25+. The second is ranking documents using a custom model that is trained using the python library tensorflow\_ranking. The third option is to use a pretrained model and it’s associated vocabulary and then finetune it using tensorflow\_ranking. Overall the best results that I have achieved have come from first running the BM25+ model and then reranking a subset of those results with a BERT model to define the final predictions.

# **Project Results**

Unfortunately, despite my best attempt I was not able to beat the baseline performance score as defined by the class professor or TAs. Nobody else was able to beat the baseline either so it is likely that the baseline is an unreasonable standard for us to achieve with our limited knowledge and experience. In this section I will briefly detail some of the variations and experiments that I attempted. It is not possible for me capture every variation that I tried but I will cover as best as I can. I do not have a consistent record of the score for every attempt so I will not be including those scores.

## **DirichletPrior**

This algorithm was implemented using metapy. The value of the parameter was optimized on the train dataset using a brute forcing approach of testing every reasonable parameter and selecting the parameter that achieved the best NCDG@20 score on the training dataset. None of the attempts for this algorithm achieved results that rivaled the BM25 algorithm so it was quickly determined to be a non-ideal approach.

Initial attempts with this algorithm were made using the body text only. Later attempts also used the title and abstract/intro. Further experimentation with the KLDivergencePRF implementation in metapy and variations on its parameters was also performed but performance lagged behind other approaches.

## **JelinekMercer**

This algorithm was implemented using metapy. The value of the parameter was optimized on the train dataset using a brute forcing approach of testing every reasonable parameter and selecting the parameter that achieved the best NCDG@20 score on the training dataset. None of the attempts for this algorithm achieved results that rivaled the BM25 algorithm so it was quickly determined to be a non-ideal approach.

This algorithm was only ever attempted with the body text and results were deemed not good enough to justify further experimentation.

## **OkapiBM25 (variation in weighting parameters not captured)**

This algorithm was implemented using metapy. The value of the three parameters was optimized on the training dataset using a brute force approach of testing every reasonable combination of parameters and selecting the set that achieved the greatest. Due to the number of variations that I tried with this algorithm, I had to optimize the parameters a number of times and I developed a multi-processing script that was capable of doing this at a much quicker pace. Initially I only attempted to run the algorithm on the body text for each paper but the performance for that was poor. With the body text I attempted to optimize a different NDCG values, 10, 20, and 50 respectively. None of them produced significant improvements in the overall performance.

Next, I attempted to use the title, abstract, and introduction along with the body text and that produce marginal improvements in the score. Another boost came once I removed the body text from the dataset and only trained on the title, abstract, and introduction. I also attempted to use metapy’s implementation of Rocchio feedback. I optimized the parameters for Rocchio in a similar manner to the normal OkapiBM25 algorithm. Ultimately the Rocchio feedback produced worse results then the standalone ranker so I excluded it from further test with this algorithm. The performance was still inadequate so I pursued further methods.

## **BM25+ (variations in parameters for each elaborated)**

This algorithm was implemented with metapy’s api using the definition of BM25+ from (Trotman et al, 2014). Since it was implemented in Python and C++ like the native metapy implementations, it ran slower. I optimized it in the same way that I optimized the OkapiBM25 algorithm. This implementation ultimately outperformed the native OkapiBM25 algorithm, but it did not beat the baseline. Better results were obtained when using this in conjunction with the BERT implementation.

I tried numerous variations on this algorithm in an attempt to increase my score on the test dataset. Similar to the other algorithms, I initially tried to only use the body text for my analysis. This did not produce the results that I desired. I have briefly listed some of the variations that I tried:

* attempted with body text only
* attempted with title, abstract, intro, and body text
* attempted with title, abstract, and intro
* attempted with title only
* attempted with Rocchio pseudo feedback, varying the parameters to Rocchio
* attempted to remove all urls in the corpus
* attempted to replace all variants of the words coronavirus, covid-19, 2019-ncov, and sars-ncov-2 with coronavirus
* attempted to run with query text, question text, or narrative text
* attempted to run with all variations of query combined
* attempted different variations of metapy analyzer chains
* attempted to remove docs with duplicate s2\_id but different uids, docs were functionally duplicates
* attempted to use a date cutoff for document
* attempted to pre-tokenize the queries and documents using BERT's tokenizer
* attempted to use multiple datasets with various weightings and combine into one result
* used different variations of stopwords that were gathered online

Ultimately I was not able to beat the baseline in the

## **BERT**

* attempted with BERT-Mini, BERT-Small, and BERT-Base
* varied the list size
* varied the batch size
* updated vocabulary to include most common coronavirus variants
* attempted to run with query, question, or narrative
* attempted to vary the max token sequence size 256, 512, 1024
* relevance = score of first doc segment
* relevance = max score of all doc segments
* relevance = mean score of all doc segments
* ignore body text and only use title+abstract+intro
* varied the number of training steps
* used both NCDG approximated loss and softmax loss
* re-rank top 1000, 2000, 5000, and 10000 results of BM25

## **SciBERT \*difficulty running this model limited options**

This model was initially very promising but the issues with getting it to run ultimately made it insufficient for what I needed. This model was create by Iz Beltagy et al. for scientific research. This model is essentially just a BERT-Base model that was pretrained on a scientific corpus instead of the generic one. The vocabulary consists of more scientific terms as a result and should theoretically be able to perform better at ranking the scientific documents in the CORD-19 corpus. I updated a few of the unused vocabulary items to the command variants of coronavirus and included the drug remdesivir as well. Unfortunately, due to the size of the model I was very limited in the list size and batch size options that I could try. I firmly believe that with a larger list size I would have been able to adequately finetune this model and it would have performed better than the BERT model. My inability to get the model to run on the TPUs available through Google Colab prevented me from realizing this goal.

## **Tensorflow Ranking custom model**

This model was created by basing it off of the example in the tensorflow ranking repository. I left the structure of the context features and the example features the same. The context features contain the query tokens and the example features contain the document tokens and when training, the relevance judgement of the file. I attempted to change the script to use the argparse module instead of the flags from the absl module but when that was attempted the script stopped producing the results of the training data to stdout. I determine that there are likely scripts deeper within the tensorflow ranking module that are using these flags to determine various facets of the training process, so I restore the flags.

The model consists of 3 hidden layers at a size of 64, 32, and 16 respectively. During testing the dropout rate of the model was adjust to 0.65 from the default of 0.80. This adjustment produced better results but further drops in this dropout rate risked overfitting the model to the training data. The batch size was set to 1 and the list size maximized so that it would train on more docs for a single query at once. Based on empirical results that I observed during experimentation, this produced better results than increasing batch size and lowering the list size. The maximum limit for the list\_size was 100 depending on the hardware available in Colab and if it exceeded that value then it would run out of memory when training.

One other significant variable was the max sequence length to use for each example. I tested with both 512 and 1024. The results did not appear to differ significantly between the two but this variable is important because it defines how much of the document can be captured per example. For this model I only used the title, abstract, and introduction because previous experimentation on other models and rankers showed that the body text was not helpful. Ultimately the results produced with this method were worse the results obtained from the BERT models.

# **Assumptions**

* If you are running this locally then you are running this code on Windows 10 machine that has ample RAM and a CUDA capable gpu. Every script can run on Linux but I have not generated scripts within this repo to accommodate that.
* All file paths are relative to the base directory of the repository

# **Installation Instructions**

## **What to install if running locally**

Acquire the datasets:

Files should be downloaded and unzipped here: .\competition\datasets

The test files can be obtained from <https://drive.google.com/file/d/1FCW8fmcneow5yyDgApkPIGM-r2x6OFkm/view?usp=sharing>

The train files can be obtained from <https://drive.google.com/file/d/1E_Y-MkNvoOYoCZUZZa8JJ3ExiHYTfKTo/view?usp=sharing>

Python

Instructions: https://docs.conda.io/projects/conda/en/latest/user-guide/install/

Latest Nvidia Drivers (required to run BERT or Tensorflow Ranking on GPU)

Instructions: https://docs.nvidia.com/cuda/wsl-user-guide/index.html

Instructions require installation of other software and packages you must follow all of them

BERT Model (required to run BERT)

Due to the size of the models, they are not directly included in the repository

Google BERT Models (recommended):

Google offers a variety of different sizes for the models to run. I ran using a BERT Mini but could possibly use a BERT Small as well. Anything larger likely requires a TPU through Google Colab to run.

https://github.com/google-research/bert

SciBERT Model (not recommended, requires TPU):

https://github.com/allenai/scibert/

## **What python packages to install**

To run the full capability of all scripts defined in this project it is required that you setup two environments. All but one script can run in the main environment but the BERT checkpoint convertor relies on the dev version of a module that conflicts with requirements of the other modules:

Main Python Environment:

This is the main environment and should be used when running most of the scripts. You can install the latest stable versions of these modules to run the project.

Python Version: 3.7

Pip Packages:

tensorflow\_ranking

metapy

pytoml

BERT Conversion Environment:

This is required because there are conflicts in the dependencies of the modules required to convert a model’s checkpoints and the modules used to run it.

Python Version: 3.7

Pip Packages:

tf-models-nightly

# **How to Run**

\*\*WARNING: running some of these scripts will require a large amount of RAM if you use the full dataset\*\*

\* Your python environment should be activated before running any scripts

## **Prepare Data (Required if local)**

Generate the json file that contains the information for each dataset. This is required in order to run any of the different methodologies.

File Path: .\competition\create\_bert\_data.bat

## **(Method 1) Run BM25+ or another ranker**

### Option 1 (Google Colab)

Run each cell individually from top to bottom to fully. If you view the file within Github there is a link at the top of the file to open in Google Colab. This is the only Google Colab file that does not require a Google Colab Pro account. The free account should be capable of running this notebook.

File Path: .\colab\_cranfield\_metapy.ipynb

### Option 2 (run locally)

Generate the Cranfield Datasets that will be required to run the ranker

File Path: .\competition\cranfield\_metapy\cm\_create\_data.bat

Run the ranker of your choice (default arguments already in the file)

File Path: .\competition\cranfield\_metapy\cm\_rank\_docs.bat

## **(Method 2) Run Tensorflow Ranking**

### Option 1 (Google Colab)

Run each cell individually from top to bottom to fully. Training data for a fine-tuned model is provided so there is not need to run training but the capability is provided. If you view the file within Github there is a link at the top of the file to open in Google Colab. T This file will likely require a Google Colab Pro account for the script to work. If not using a Colab Pro account it will run out of memory.

File Path: .\colab\_tfr.ipynb

### Option 2 (run locally)

Convert the train data into two tensorflow example list with context (elwc) files

File Path: .\competition\tfr\_custom\tfr\_create\_train\_elwc.bat

Train the model on the elwc files

File Path: .\competition\tfr\_custom\tfr\_train\_model.bat

Re-rank the output of a previous run (requires a file in the format of a predictions file, no limit on docs per query but will truncate output to 1000

File Path: .\competition\tfr\_custom\tfr\_predict.bat

## **(Method 3) Run a BERT model**

### Option 1 (Google Colab)

Run each cell individually from top to bottom to fully. Training data for a fine-tuned model is provided so there is not need to run training but the capability is provided. If you view the file within Github there is a link at the top of the file to open in Google Colab. This file will likely require a Google Colab Pro account for the script to work. If not using a Colab Pro account it will run out of memory.

File Path: .\colab\_bert.ipynb

### Option 2 (run locally)

Convert the train data into two tensorflow example list with context (elwc) files

File Path: .\competition\bert\bert\_create\_train\_elwc.bat

Train the model on the elwc files

File Path: .\competition\bert\bert\_train\_model.bat

Re-rank the output of a previous run (requires a file in the format of a predictions file, no limit on docs per query but will truncate output to 1000

File Path: .\competition\bert\bert\_predict.bat

# **Script API Documentation**

\*\*Unless otherwise specified paths are relative to the root of the repository

## **General**

File: ./competition/check\_covid\_variants.py

Purpose: Check for coronavirus variants in the corpus

Source: developed by project team

API:

variant\_file:

the file that the coronavirus variants will be output to. File can already exist. Existing variants will be loaded.

known\_variants:

known variants of the coronavirus that exist in the corpus and will be used to determine other variants

doc\_keys:

the keys to use when search the document dictionary for variants

run\_type:

the dataset that will be searched for variants

input\_dir:

the directory that contains the json representation of the datasets to be processed

Detailed Description:

This is a python script that was implemented in python 3.7. This script relies on the output of create\_bert\_data.py in order to run. It loads the json for one or both datasets and then processes the text to find variants of some predefined words in the corpus and queries. The text is processed using python multiprocessing module in order to speed up execution of the script.

File: ./competition/checkpoint\_converter.py

Purpose: Converter BERT checkpoint files from Tensorflow v1 to v2+

Source: Tensorflow Model Garden Repository

API:

bert\_config\_file:

Bert configuration file to define core bert layers.

checkpoint\_to\_convert:

Initial checkpoint from a pretrained BERT model core (that is, only the BertModel, with no task heads.)

converted\_checkpoint\_path:

Name for the created object-based V2 checkpoint.

checkpoint\_model\_name:

The name of the model when saving the checkpoint, i.e., the checkpoint will be saved using: tf.train.Checkpoint(FLAGS.checkpoint\_model\_name=model).

converted\_model:

Whether to convert the checkpoint to a `BertEncoder` model or a `BertPretrainerV2` model (with mlm but without classification heads).

Detailed Description:

This script was not written by me but was lightly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script is meant to convert a BERT module that you downloaded into a version that is capable of being ran using tensorflow version 2.0+.

File: ./competition/create\_bert\_data.py

Purpose: Combine and compile the information from the dataset into and easily loadable json for use by other scripts

Source: developed by project team

API:

vocab\_file:

The file containing the vocabulary to be used when tokenizing a text

variant\_file:

A file containing variants of the word coronavirus or its equivalents

variant\_default:

The value that will be used to replace all variants in the corpus

run\_type:

The dataset that will be searched for variants

tokenize:

If use, this will store a tokenized representation of the script in the output json. Must be used with `vocab\_file`

input\_dir:

The directory that contains all of the source files for the dataset

output\_dir:

The directory where the json representations of the documents will be stored

Detailed Description:

This is a python script that was tested in python 3.7. The inputs to the script will determine the exact behavior at run time but lets discuss the more complete case of the training dataset. When the training dataset is specified it will start by loading the queries from there original xml format into a python dictionary. If the script is ran with the tokenize option then each of the three variants of the query (query, question, and narrative) will be tokenized according to the vocabulary defined in the specified vocab file. This json is then dumped into a target directory. For the training dataset it will also load the query relevance judgements. A list of all of the documents and their associated uid, title, abstract, publication date, and list of files containing the text representation is pulled from the metadata.csv file. The list of documents to further process is pruned done to the same as the list of relevance judgements. For each document to process the script iterates through the list of files that contains the text representation until it finds a suitable candidate or exhausts all options. The text from the representation file is loaded in the document dictionary object. If the tokenize option is used, the documents will then be tokenized through the use of a multi-processed pool. The resulting output is then written to the disk.

## **Metapy (BM25, …)**

File: ./competition/cranfield\_metapy/create\_cranfield.py

Purpose: Use the json representation of the dataset to create a cranfield dataset for use with metapy

Source: developed by project team

API:

variant\_file:

A file containing variants of the word coronavirus or its equivalents

variant\_default:

The value that will be used to replace all variants in the corpus

run\_type:

The dataset that will be searched for variants

query\_keys:

The keys that will be used when creating the cranfield query file. If multiple keys are specified, text will be combined.

doc\_keys:

The keys that will be used when creating the cranfield document files. Multiple keys can be specified and multiple keys can be combined into a single document. If creating separate datasets, use a ';' to separate keys. If combining keys for a dataset use a ':' to separate keys:

cranfield\_dir:

The directory to use as a base for generating the cranfield data.

input\_dir:

The directory where the json representations of the documents are stored

Detailed Description:

This is a python script that was implemented in python 3.7. This script relies on the output of create\_bert\_data.py in order to run. This uses the loaded json representation of the dataset to produce a use specified cranfield dataset for use with metapy. Each cranfield dataset can be uniquely defined by the user to consist of a one-to-one relationship with keys in the document dictionary or it can consist of multiple keys combined. This allows for flexibility in how the expansive each iteration of ranking testing is.

File: ./competition/cranfield\_metapy/search\_eval.py

Purpose: Rank the documents in the corpus and create the predictions file

Source: original version from MP 2.2 and heavily modified to fit use case

API:

config\_template:

The template file that will be used for creating the configs for each run of the ranker

run\_type:

The dataset that will be searched for variants

dat\_keys:

The keys to use that indicate the name of the cranfield dataset(s). This corresponds to the first key used for every section of 'doc\_keys' parameter and the 'create\_cranfield.py' script.

doc\_weights:

The weights to use when combining the rankings of multiple datasets.

ranker:

The ranker to use for ranking the documents. Valid rankers can be found in the script.

params:

The value(s) for the ranker parameters. Multiple values should be separated by `;`.

cranfield\_dir:

The directory that is the base for the cranfield dataset(s)

predict\_dir:

The directory to contain the predictions file.

remove\_idx:

Delete an existing inverted index and create a new one. If no index exists it will not fail

Detailed Description:

This is a python script that was implemented in python 3.7. This script relies on the output of create\_cranfield.py in order to run. This script allows the user to specify what ranker to run from a predefined list of rankers. The user can also run rankings over multiple datasets and specify the weight that each ranking will contribute to the final prediction rankings.

File: ./competition/cranfield\_metapy/search\_eval\_pool.py

Purpose: Used to find the optimal parameters for ranking algorithms. Not for production use.

Source: original version from MP 2.2 and heavily modified to fit use case

API: N/A

Detailed Description:

This is a python script that was implemented in python 3.7. This script relies on the output of create\_cranfield.py in order to run. This script allows the user to attempt to optimize the parameters for a predefined list of rankers. The optimization takes place using a multi-processed pool so that it can run numerous iterations over a predefined range of values. Each iteration of the optimization loop is evaulated based on normalized cumulative gain at 20 documents.

## **Tensorflow Ranking**

File: ./competition/tfr\_custom/tfr\_convert\_json\_to\_elwc.py

Source: modified version of file located here https://github.com/cognitiveailab/ranking

Purpose: To convert json data to an elwc formatted tfrecord file for use with model training

API:

vocab\_file:

The file containing the vocabulary to be used when tokenizing a text

sequence\_length:

The max length of any individual query or document

query\_file:

The json file that contains all of the queries

qrel\_files:

The file containing the training relevance judgements

query\_key:

The type of query that will be used as context for ranking, i.e. (query, question, narrative)

doc\_file:

The json file that contains all of the documents

output\_train\_file:

The tfrecord file to be used for training

output\_eval\_file:

The tfrecord file to be used for evaluation

list\_size:

The maximum number of documents to score per query

do\_lower\_case:

ensure all query and document strings are lowercase

Detailed Description:

This script was not originally written by me but has been highly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script relies on the output of create\_bert\_data.py in order to run. This script loads in the queries, documents, and relevance judgments for the train dataset and converts them to an example list with context files for use in training the model. The maximum number of documents associated with each query is defined by the list size parameter. If the number of documents for a query exceeds this parameter value then the document list is chunked into multiple elwc representations before being output to the files. The elwc objects are formatted for the model's specific requirements. Each query and document is limited to a maxim number of tokens as defined by sequence\_length and if a document or query is longer than this value then it is truncated.

File: ./competition/tfr\_custom/tfr\_train.py

Source: lightly version of file located here https://github.com/tensorflow/ranking/blob/master/tensorflow\_ranking/examples/tf\_ranking\_tfrecord.py

Purpose: To train a tensorflow model on a dataset with predefined relevance judgements.

API:

data\_format:

Data format defined in data.py.

train\_path:

Input file path used for training.

eval\_path:

Input file path used for eval.

vocab\_path:

Vocabulary path for query and document tokens.

model\_dir:

Output directory for models.

batch\_size:

The batch size for train.

num\_train\_steps:

Number of steps for train.

learning\_rate:

Learning rate for optimizer.

dropout\_rate:

The dropout rate before output layer.

hidden\_layer\_dims:

Sizes for hidden layers.

list\_size:

List size used for training. Use None for dynamic list size.

group\_size:

Group size used in score function.

loss:

The RankingLossKey for the loss function.

weights\_feature\_name:

The name of the feature where unbiased learning-to-rank weights are stored.

listwise\_inference:

If true, exports accept `data\_format` while serving.

use\_document\_interactions:

If true, uses cross-document interactions to generate scores.

embedding\_dim:

max size of any query or document

Detailed Description:

This script was not originally written by me but has been lightly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script relies on the output of tfr\_convert\_json\_to\_elwc.py in order to run. This script loads in the elwc files generated by tfr\_convert\_json\_to\_elwc.py and trains the model on the data provided in the elwc files. The training will run for the "num\_train\_steps" defined by the user. The outputs of this script are stored in "model\_dir" and you can load the output model for use in prediction of a documents relevance.

File: ./competition/tfr\_custom/tfr\_predict.py

Source: modified version of file located here https://github.com/cognitiveailab/ranking

Purpose: To score the documents in the test dataset against the queries in the test dataset and output a predictions file

API:

vocab\_file:

The file containing the vocabulary to be used when tokenizing a text

sequence\_length:

The max length of any individual query or document

query\_file:

The json file that contains all of the queries

qrel\_files:

The file containing the training relevance judgements

query\_key:

The type of query that will be used as context for ranking, i.e. (query, question, narrative)

doc\_file:

The json file that contains all of the documents

output\_file:

The file that will contain the scores

model\_path:

The path to the saved model for use in predictions

docs\_at\_once:

The maximum number of documents to score at once

rerank\_file:

The input file consisting of previous rankings that will be reranked with the tensorflow model

do\_lower\_case:

ensure all query and document strings are lowercase

Detailed Description:

This script was not originally written by me but has been highly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script relies on the output of create\_bert\_data.py in order to run. This script loads in the queries and documents for the specified dataset. It also loads in the relevance judgements of a previous ranking and a trained model. The list of documents to rank is limited to only the files specified in the previous ranking. Each document query combination is converted into an example list with context(elwc) object and then it is passed to the loaded model. All of the documents ranked for each query is combined into a singular list that is sorted by the score and only the top 1000 documents are output to a predictions file. The predictions file output is the same as the file of the relevance judgements.

## **BERT**

File: ./competition/bert/bert\_predict.py

Source: modified version of file located here https://github.com/cognitiveailab/ranking

Purpose: To convert json data to an elwc formatted tfrecord file for use with model training

API:

vocab\_file:

The file containing the vocabulary to be used when tokenizing a text

sequence\_length:

The max length of any individual query or document

query\_file:

The json file that contains all of the queries

qrel\_files:

The file containing the training relevance judgements

query\_key:

The type of query that will be used as context for ranking, i.e. (query, question, narrative)

doc\_file:

The json file that contains all of the documents

output\_train\_file:

The tfrecord file to be used for training

output\_eval\_file:

The tfrecord file to be used for evaluation

list\_size:

The maximum number of documents to score per query

do\_lower\_case:

ensure all query and document strings are lowercase

Detailed Description:

This script was not originally written by me but has been highly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script relies on the output of create\_bert\_data.py in order to run. This script loads in the queries, documents, and relevance judgments for the train dataset and converts them to an example list with context files for use in training the model. The maximum number of documents associated with each query is defined by the list size parameter. If the number of documents for a query exceeds this parameter value then the document list is chunked into multiple elwc representations before being output to the files. The elwc objects are formatted for the model's specific requirements. Each query and document is limited to a maxim number of tokens as defined by sequence\_length and if a document or query is longer than this value then it is truncated.

File: ./competition/bert/bert\_train.py

Source: lightly version of file located here https://github.com/tensorflow/ranking/blob/master/tensorflow\_ranking/extension/examples/tfrbert\_example.py

Purpose: To train a tensorflow model on a dataset with predefined relevance judgements.

API:

local\_training:

If true, run training locally.

train\_input\_pattern:

Input file path pattern used for training.

eval\_input\_pattern:

Input file path pattern used for eval.

learning\_rate:

Learning rate for the optimizer.

train\_batch\_size:

Number of input records used per batch for training.

eval\_batch\_size:

Number of input records used per batch for eval.

checkpoint\_secs:

Saves a model checkpoint every checkpoint\_secs seconds.

num\_checkpoints:

Saves at most num\_checkpoints checkpoints in workspace.

num\_train\_steps:

Number of training iterations. Default means continuous training.

num\_eval\_steps:

Number of evaluation iterations.

loss:

The RankingLossKey deciding the loss function used in training.

list\_size:

List size used for training.

convert\_labels\_to\_binary:

If true, relevance labels are set to either 0 or 1.

model\_dir:

Output directory for models.

dropout\_rate:

The dropout rate.

bert\_config\_file:

The config json file corresponding to the pre-trained BERT model. This specifies the model architecture. Please download the model from the link: https://github.com/google-research/bert

bert\_init\_ckpt:

Initial checkpoint from a pre-trained BERT model. Please download from the link: https://github.com/google-research/bert

bert\_max\_seq\_length:

The maximum input sequence length (#words) after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter than this will be padded.

bert\_num\_warmup\_steps:

This is used for adjust learning rate. If global\_step < num\_warmup\_steps, the learning rate will be `global\_step/num\_warmup\_steps \* init\_lr`. This is implemented in the bert/optimization.py file.

Detailed Description:

This script was not originally written by me but has been lightly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script relies on the output of bert\_convert\_json\_to\_elwc.py in order to run. This script loads in the elwc files generated by bert\_convert\_json\_to\_elwc.py and trains the model on the data provided in the elwc files. The training will run for the "num\_train\_steps" defined by the user. The outputs of this script are stored in "model\_dir" and you can load the output model for use in prediction of a documents relevance.

File: ./competition/bert/bert\_convert\_json\_to\_elwc.py

Source: modified version of file located here https://github.com/cognitiveailab/ranking

Purpose: To score the documents in the test dataset against the queries in the test dataset and output a predictions file

API:

vocab\_file:

The file containing the vocabulary to be used when tokenizing a text

sequence\_length:

The max length of any individual query or document

query\_file:

The json file that contains all of the queries

qrel\_files:

The file containing the training relevance judgements

query\_key:

The type of query that will be used as context for ranking, i.e. (query, question, narrative)

doc\_file:

The json file that contains all of the documents

output\_file:

The file that will contain the scores

model\_path:

The path to the saved model for use in predictions

docs\_at\_once:

The maximum number of documents to score at once

rerank\_file:

The input file consisting of previous rankings that will be reranked with the tensorflow model

do\_lower\_case:

ensure all query and document strings are lowercase

Detailed Description:

This script was not originally written by me but has been highly modified for the purpose of this project. This is a python script that was tested in python 3.7. This script relies on the output of create\_bert\_data.py in order to run. This script loads in the queries and documents for the specified dataset. It also loads in the relevance judgements of a previous ranking and a trained model. The list of documents to rank is limited to only the files specified in the previous ranking. Each document query combination is converted into an example list with context(elwc) object and then it is passed to the loaded model. All of the documents ranked for each query is combined into a singular list that is sorted by the score and only the top 1000 documents are output to a predictions file. The predictions file output is the same as the file of the relevance judgements.

# **Resources**

## **Citations**

Rama Kumar Pasumarthi and Sebastian Bruch and Xuanhui Wang and Cheng Li and Michael Bendersky and Marc Najork and Jan Pfeifer and Nadav Golbandi and Rohan Anil and Stephan Wolf. 2019. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 2970-2978

Turc, Iulia and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. 2019. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. arXiv preprint arXiv:1908.08962v2

G. V. Cormack, C. L. A. Clarke, and Stefan Buttcher. 2009. Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods

Andrew Trotman, Antti Puurula, Blake Burgess. 2014. Improvements to BM25 and Language Models Examined

Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’19), July 21–25, 2019, Paris, France. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3331184.3331303>

Iz Beltagy, Kyle Lo, Arman Cohan. 2019. SCIBERT: A Pretrained Language Model for Scientific Text. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3615-3620

## **Installation Guidance**

tensorflow gpu requirements:

<https://www.tensorflow.org/install/gpu>

CUDA:

<https://developer.nvidia.com/cuda-10.1-download-archive-update2?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1804&target_type=deblocal>

cuDNN:

<https://developer.nvidia.com/rdp/cudnn-archive>

install .deb:

<https://www.quora.com/Is-it-possible-to-install-a-deb-package-in-Windows#:~:text=Potentially%20yes%2C%20as%20long%20as,deb>

## **Stopwords**

<https://www.ranks.nl/stopwords>

<https://countwordsfree.com/stopwords>

## **BM25**

<https://github.com/vespa-engine/cord-19/blob/master/cord-19-queries.md>

<https://docs.vespa.ai/documentation/reference/bm25.html>

## **Tensorflow Ranking**

<http://cognitiveai.org/2020/09/08/using-tensorflow-ranking-bert-tfr-bert-an-end-to-end-example/>

<https://github.com/cognitiveailab/ranking>

<https://github.com/tensorflow/ranking>

<https://colab.research.google.com/github/tensorflow/ranking/blob/master/tensorflow_ranking/examples/handling_sparse_features.ipynb>

## **Tensorflow Ranking (how to serve a model)**

<https://www.tensorflow.org/tfx/serving/serving_basic>

<https://www.tensorflow.org/tfx/tutorials/serving/rest_simple#serve_your_model_with_tensorflow_serving>

<https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/saved_model.md#cli-to-inspect-and-execute-savedmodel>

<https://github.com/cognitiveailab/ranking/blob/master/tensorflow_ranking/extension/examples/tfrbert_client_predict_from_json.py>

## **Tensorflow Ranking (how to predict)**

<https://github.com/tensorflow/ranking/issues/48>

<https://stackoverflow.com/questions/59528975/tf-estimator-predict-slow-with-tensorflow-ranking-module>

## **BERT Checkpoint Conversion**

<https://github.com/tensorflow/models/tree/master/official/nlp/bert>

## **BERT**

<https://github.com/google-research/bert>

<https://ai.googleblog.com/2018/12/tf-ranking-scalable-tensorflow-library.html>

<https://arxiv.org/pdf/1812.00073.pdf>

## **SciBERT**

<https://www.aclweb.org/anthology/D19-1371/>

<https://huggingface.co/gsarti/scibert-nli>

<https://github.com/allenai/scibert/>

## **Docker**

<https://superuser.com/questions/1382472/how-do-i-find-and-enable-the-virtualization-setting-on-windows-10>

<https://docs.docker.com/docker-for-windows/troubleshoot/#virtualization-must-be-enabled>

<https://docs.docker.com/docker-for-windows/install/>