Documentation

Hamdi Rezgui

March 2021

Contents

1 Introduction

In this document, i will present a brief description of the code files that i used to reproduce the results of chapter 7 of Jibril FREJ's thesis about efficient IR models.

2 Brief description of the code files

2.1 std_tokenizer.py

This file contains some useful functions for building vocabulary, creating indexs and indexing documents or queries that are used in the collection classes methods. "std_tokenizer_build_standard_vocabulary" will build vocabulary from queries and documents. "std_tokenizer_index" creates index and indexes a dataframe which can be documents or queries for example "std_tokenizer_preprocess" will use the two previous functions and returns a vocabulary , a query index, document index ,indexed queries and indexed documents.

2.2 baseline_models_and_tdv_implementation.py

This file contains a definition of non differentiable baseline IR models for exact matching like TF, TF-IDF, DIR, BM25 and JM. It also has a definition of weighted versions of those models where they multiply by the TDV weight in the computation the relavance score for each token. The weighted versions were made as an experiment. They weren't used in the end in the code.

2.3 differentiable_models.py

This file contains the definition of the differentiable version of models used in chapter 7 of Jibril FREJ's thesis which are the simple tf model, idf-tf model, Dirichlet model, BM25 model and Jelineck-Mercer model. Each model,is initialized by an embedding layer, a dense hidden layer and a dropout rate. For the initialization an embedding matrix and dropout rate are needed. The embedding matrix given is calculated in the methods of the class of the collection to be used either TREC or Wikir collection.
The model has also "make_Bow" method which creates a bag of word representation out of a sequence, an index and a sparse index. This function is used in the "call" function of the class of model to have a bag of words representation (dense tensor after processing) of the query and the documents (after dropout has been applied) and calculate the relevance measure (depending on the model used). Finally, the modal has also a "compute_index" method that will compute the TDV weights of tokens in the vocabulary

2.4 Trec_Collection.py

This file contains global methods and a trec collection class definitions with its own internal methods. The most important global method is the "read_collection" global method.
It reads queries and their corresponding qrel file and builds folds of queries and qrel for kfold cross validation. It also saves queries and documents in csv format which is the format needed in the definition of the trec collection class. This method is a preprocessing step that should be run only once to get appropriate format of your trec collection. The class definition have several important methods like "load_collection" that loads the collection after it is been preprocessed by the global method "read_collection". It loads documents, queries for each fold, qrel for each fold and training qrel for each fold. The class also has preprocessing methods which are grouped in the function "standard_preprocess" that builds vocabulary of the collection (method " build_standard_vocabulary") ,building queries and documents index and indexing them...
It also has processing methods likebuilding inverted index of documents and compute important parameters like idf and collection frenquencies... We can get all of this by the use of the method "compute_info_retrieval" Moreover, it computes the fasttext embedding of the vocabulary through the method "compute_fasttext_embedding" and saves the indexed collection into a pickle file by the method "pickle_indexed_collection" Finally, it has the method "generate_training_batches" that generates batches for training from outside the fold in question (query batches ,postive documents batches, negative documents batches) (the concatination of

queries,positive and negative documents in the other folds that we are going to train on for the particular fold)

2.5 wikIR_Collection.py

This doesn't have global functions but it has the same internal methods that have the same objective in the definition of the class of the collection as the class of TREC collection. The only difference is since in this collection we don't use kfold cross validation but we use training , validation and test partitions.This file has a method for generating training batches and another one for the test batches.

2.6 utils.py

This file contains useful functions that are used in most of the other files. There are two main catergories of methods for :

2.6.1 reading query relavance files of collection

For this goal, there are two functions. The function "read_qrels" which reads a query relavance file in a collection (wikIR or TREC collections) under a certain format (after running read_collection for TREC collection or directly used for wikIR collection) and extract relavance measure. The function "read_trec_train_qrels" that builds a dictionary of positive and negative qrels for training. Remember that our loss function is pairwise (positive and negative documents)

2.6.2 preprocessing collection using the TDV weights

It has a version of functions of preprocessing that include TDV weights like updating inverted index with weights, compute idf with weights, compute document length with weights ... It has also a definition of functions like "eval_baseline_index_trec" which evaluates the performance of baseline models on TREC collection before learning TDV weights and "eval_learned_index_trec" which evaluates baseline models after getting TDV weights and using them to update the inverted index for example, compute idf with weights ... Same thing for wikIR collection it has two functions who do the same as described before. The functions described in this part use a method that is also in the "utils.py" file which is "compute_metrics" which can save topk results for a particular model and evaluates it through multiple metrics like recall, ndcg_cut, p, map using the pytrec package.

2.7 build_Trec_Collection.py

This files parses different arguments like the collection path, the index path, the fasttext model for embedding... Then ,if you wish to build folds for cross validation , it uses the read_collection method to do that. It creates an instance of class Trec_Collection and it loads the collection, does the preprocessing, builds the inverted index, compute the embedding of the tokens in the vocabulary and it saves the indexed collection in a pickle file.

2.8 building_Trec_Collection.sh

It is a shell script file that takes the arguments needed for build_Trec_Collection.py , runs it for every Trec_Collection that we want to index and saves errors and output to console .

2.9 build_wikir_Collection.py

This file parses different arguments like the collection path, the index path, the fasttext model for embedding... Then ,if does all the steps as described in build_Trec_Collection.py other than building folds.

2.10 build_wikir_Collection.sh/build_wikirS_Collection.sh

It does the same steps as building_Trec_Collection.sh but for the wikIR_78 and wikIRS_78 databases.

2.11 training_on_trec_collection.py

This file parses different arguments like the collection path, the indexed collection file path ... ,the path to save weights , the path for results(where you store results for each epoch for each experiment for each model and for each fold) and plot values path (where you store for each fold the dictionary that for a certain experiment keeps the metrics of every model to evaluate (tf-idf, BM25, DIR,JM)).
It also parses training parameters like learning rate and Droupout rate as well as the experiment name and the IR model to use. There are several steps done in this file :

- Loading the indexed TREC collection
- Loading the relevance judgments (qrel) of the collection

  For each fold the following steps are taken:

- Creating a directory for the fold and for the experiment to store results , weights and plot values
- Evaluating baseline models and updating plot values dictionary created for the fold and saving them as pickle file
- Intializing batch size, the optimizer with the learning rate ,the loss function and loading the differentiable model (chosen in the IR_model argument)

  While the number of epochs is less than the max number of epochs do:

- Generating the training batch (queries,positive documents and negative documents)

  For each training entity do:

- Reshaping queries,positive and negative documents using padding and transforming it to a numpy ndarray
- Creating sparse query,positive and negative documents indexes
- Computing relevance and dense documents for positive and negative documents in the batch by using the call function of the differentiable model initialized before
- Computing the hinge loss, the regularization loss and the total loss
- Computing gradients and back propagating them
- Compute TDV weights for tokens in the vocabulary and saving them as a pickle file
- Building inverted index and calculating idf,collection frequencies and document length taking into account the TDV weights.

- Evaluating baseline models with the new inverted index and the new parameters calculated in the previous step

2.12 training_wikir_collection.py

It has the same steps as in the file training_on_trec_collection.py. The only difference is instead of having folds we have training,validation and test partitions.

2.13 training_trec.sh/training_wikIR.sh / training_wikIRS.sh

They are shell scripts that takes arguments like collection path, indexed collection path ... and training parameters like learning rate, number of folds , dropoutrate ...

3 Steps to follow to get the results of chapter 7

1. Launching building_Trec_Collection.sh if the collections have not been indexed and choose the argument -b as True if the folds weren't built before.
2. Launching build_wikir_Collection.sh and build_wikirS_Collection.sh if the collections haven't been indexed.
3. Launching train_trec.sh to train and evaluate the models on the TREC Collection
4. Launching training_wikIR.sh and training_wikIRS to train and evaluate the models on the wikIR and wikIRS collections