SI 650 / EECS 549 F24 Homework 2
Learning to Rank

Due: See Canvas for official deadline

1 Homework Overview

Document relevance for a query is a function of many factors. Relevance scoring techniques like BM25 capture only a part of what might make a document relevant. Factors such as the meta-data of a page, how and where query text occurs, or even where the page is located within a larger network of pages all might meaningfully contribute to a better estimate of a page's relevance to a query. But how to merge all these different signals? Commonly, search engines use *Learning to Rank* to combine different signals of relevance to produce a single score; here, a machine learning model is trained to predict pages' relevance scores from a variety of features—directly learning how to weight various signals.

Homework 2 will have you *extending* the code from Homework 1 to integrate a Learning to Rank model in your Ranker. To accomplish this goal, you will implement three new pieces of an Information Retrieval system:

1. **Relevance Feature Engineering**: Calculate different features that may make a document more or less relevant for a particular query
2. **Network Structure**: Calculate structural properties of a network of linked documents to estimate a document's quality and importance
3. **Learning to Rank**: Use learning to rank models to estimate the relevance of a document for a query on the basis of features

To implement these three pieces, we have provided updated python files with stub methods that need to be implemented and descriptions of what is to be done. Like in the first homework, this document serves as high-level guidance on the tasks to be done and additional specifics of the implementations are provided in the code itself, which is probably where you'll be looking when you actually go to implement things. This document contains the list of tasks you'll need to complete and the point values of completing each task.

This assignment has the following learning goals:

1. Learn which features might be useful for scoring relevance
2. Create a network from an edge list
3. Calculate simple network properties like PageRank on a network
4. Create features for training a L2R model
5. Learn how to train a L2R model
6. Learn where and how a L2R model is used in re-ranking documents
7. Gain new software development and debugging skills
8. Build more intuition on how IR systems are designed and how the different pieces fit together
9. Improve the ability to read software library documentation

2 Important Notes

Homework 2 includes a few changes to the core search engine implementation and design. The following is a list of several notable changes.

1. You will use the `RegexTokenizer` for all preprocessing. You still do not need to deal with punctuation.
2. No multi-word expression processing is to be used.
3. You should lower-case all tokens for this homework.
4. The method signatures for the scorers have changed slightly to improve your implementation's performance. We have included notes in the stub code and a README to help you see how to quickly convert your HW1 solution's code.
5. In HW1, minimum word frequency was calculated with respect to each document; this was incorrect. You should calculate a token's frequency across the entire collection and filter based on that value.
6. Set minimum word frequency for document text to 50. Do not use a minimum word frequency for titles.
7. All solutions should use stop word filtering. A new stopword list is provided as well.
8. Use the BasicInvertedIndex for all solutions.

Things to keep in mind when working on this assignment:

1. The public test cases that we released should cover some of the basic tasks you need to do. Feel free to add more cases to test your own code.
2. The toy input file is useful for testing feature extraction with queries. We recommend doing manual tests to verify that the implementation produces what you expect based on the specifications. You are welcomed/encouraged to try adding your own documents to this file if you want to try out specific edge cases.
3. You do not need to implement all of the features to begin implementing/testing the L2R code. You can design your code to work with whatever subset of the features you have implemented. Since three of the features were in homework 1, you can try starting with just those three if you want to get the L2R model working first.
4. We have provided a few small network files for you to test your code with. We recommend looking at the outputs manually. You can also use the visualization code for `scikit-network` to try showing your networks too if you want.
5. Try testing your implementation locally using Jupyter to import functions and passing input test you expect. We have provided a starter notebook to help outline the main steps for getting this running.
6. The network part is separate and should be pre-computed. You do not need to re-run PageRank every time you start the search engine

To reduce ambiguity in this assignment, the following notation is used here and through all future assignments.

1. an arbitrary word is w_i where i refers to which word this is within the vocabulary V. A word can be a single token or a multi-word expression.[1]

[1] Note that for Homework 2, there are no multiword expressions.

2. an individual document is d, which may be denoted as d_i to indicate an arbitrary document in the collection. The length of the document in words is denoted as $|d|$, which includes stop words (before they are filtered).
3. the set of all documents in the collection is D and $|D|$ denotes the number of documents in the collection
4. the set of all documents' titles is T and $|T|$ denotes the number of documents' titles in the collection. In practice, this is the same as the number of documents but we use T to make the distinction clearer in the equations of whether we're referring to the document's main text (D) or its title (T).
5. $c_d(w_i)$ is the count of how many times w_i appears in the main text of document d (not including any other metadata/title associated with the document)
6. $c_t(w_i)$ is the count of how many times w_i appears in the title t (this assumes that all documents have a title)
7. $df_{w_i}^D$ is the number of documents in D that contain w_i in the body text (not title or metadata). $df_{w_i}^T$ is the analogous value for the number of titles that a word occurs in.
8. $avdl$ is the average document length, i.e., $\frac{1}{|D|}\sum_i |d_i|$

3 Feature Indexing

What features are important to making a web page relevant? There are a surprising number of options! Broadly, features are grouped into two categories: (1) *static* features that are independent of the query, such as the domain of the webpage itself, and (2) dynamic features that are calculated on the basis of the query. We'll be implementing both types of features in this part and adding a few more features in the next part.

You have already seen some of the features from Homework 1. In Homework 2, we'll re-use most of that code to add the various relevance scorers as features. Ideally, our L2R model will learn how to weight these different scores to get an even better ranking. The following is the list of features you need to implement.

1. Query length: $|q|$
2. Article length: $|d|$
3. Title length: $|t|$
4. TF (document): $\sum_{w_i \in q \cap d} log(c_d(w_i) + 1)$
5. TF (title): $\sum_{w_i \in q \cap t} log(c_t(w_i) + 1)$
6. TF-IDF (document): $\sum_{w_i \in q \cap d} \left[log(c_d(w_i) + 1) \cdot \left(log\left(\frac{|D|}{df_{w_i}^D}\right) + 1 \right) \right]$
7. TF-IDF (title): $\sum_{w_i \in q \cap t} \left[log(c_t(w_i) + 1) \cdot \left(log\left(\frac{|D|}{df_{w_i}^T}\right) + 1 \right) \right]$
8. BM25 (document): $\sum_{w_i \in q \cap d} log(\frac{N - df(w_i) + 0.5}{df(w_i) + 0.5}) \cdot \frac{(k_1+1) \cdot c_d(w_i)}{k_1(1-b+b\frac{|d|}{avdl}) + c_d(w_i)} \cdot \frac{(k_3+1) \cdot c_q(w_i)}{k_3 + c_q(w_i}$
9. Pivoted Normalization (document): $\sum_{w_i \in q \cap d} c_q(w_i) \frac{1 + log(1 + log(c_d(w_i))}{1 - b + b\frac{|d|}{avdl}} log(\frac{|D|+1}{df(w_i}))$
10. Document Categories: A binary feature for each of the categories appearing in at least 1000 pages, indicating whether the current page also has that category. (There should be 118 binary features)

■ **Problem 1.** (5 points) Calculate the set of categories that occur in at least 1000 articles (these are in the `category` field of each json object). Save these categories to a file.
■ **Problem 2.** (5 points) Add at least one new feature to be used with your L2R model. The features could be dynamic or static. The feature should not be something you already implemented as a scorer in Homework 1. For some inspiration, think of all the query-document relevance ratings you did for Homework 1. What kinds of documents would you want to retrieve or what made them seem more relevant? In a few sentences, describe what your feature is and what your motivation/inspiration was for creating it.
■ ***Optional* Problem 1.** (0 points) Some features require knowing where the query's words occur in a document. For example, one common feature is the query-span length, which is the smallest number of sequential tokens in the document that contain all the query's terms (in any order). Supporting these types of features requires using positional indexing. Try including a PositionalInvertedIndex and adding these new features to the model.

4 Network Analysis

The second piece will have you working with the Wikipedia hyperlink network to add more static features about each document. Here, we'll use two methods that we discussed in class for scoring features of nodes in a graph: PageRank and HITS. Both have been shown to be useful for ranking web pages—Google was based on PageRank initially, even before L2R was widely used! For calculating both, you'll be using standard libraries, rather than implementing these methods yourself. You should implement this code in `scikit-network` and save the network features to a file separately since they won't change. Please install scikit-network==0.31.0 to avoid errors.

To summarize (more details in code), you will implement:

1. Load in the Wikipedia link network using the `scikit-network` library
2. Calculate the PageRank value for each article using `scikit-network`
3. Calculate the HITS hubs and authorities values for each article using `scikit-network`
4. Save the docid, PageRank, and HITS results to a file, for use later
5. (later) Include the PageRank, hub, and authority values as features in the L2R calculation.

To get a sense of what these static measures are doing, let's take a look at which pages have high values and how PageRank relates to other features.
■ **Problem 3.** (5 points) What are the 10 articles with the highest values for PageRank, HITS hub, and HITS authority? Report all three top-10 lists in a table. In a few sentences describe what you see and whether you are surprised to see any articles in these top 10 lists. In another sentence, describe which of the three scores you think would be most useful as a feature for prioritizing "good" documents.
■ **Problem 4.** (5 points) Plot article length (in terms of tokens as measured by split()) versus PageRank using the values for the first 10K documents (calculate PageRank for the whole network first). In a few sentences, describe what you see and how you interpret the relationship between the two measures.

5 Learning to Rank

In the last part of the assignment, we'll put all these features together and learn how to weigh them with our L2R system. For this assignment, we'll use the LightGBM to train the ranker—i.e., you're not implementing any of the methods we saw in class!

The common way IR systems use a L2R model is for ranking a *subset* of the documents that are potentially relevant. Because our collection is huge, we almost never run our L2R model to rank everything! Instead, we'll use a simple retrieval scoring heuristic like BM25 to initially identify potentially-relevant documents and then apply our L2R model to re-rank those documents using all the features. In practice, we refer to this as *re-ranking*, since some scoring metric like BM25 has already ranked the documents and then we'll just re-order a part of its results. This has huge computational advantages by only using the L2R model on what is a "good set" to start from while avoiding wasting our time re-ranking documents that are likely not good matches.

You'll use LightGBM's ranking implementation, which is a pair-wise ranker. This library is very similar to scikit-learn. We *fit* the model's parameters to train it, and then we *predict* a rank ordering from a list of new query-document features. There are many different hyperparameter options for the model. You can use the default options for this assignment.

The basic workflow should look something like this:

1. Initialize the LightGBM when you initialize your `L2RRanker`
2. Call `train` on your `L2RRanker` passing in the training data file, which will...
   (a) Load the relevance scores for query-document pairs
   (b) Convert the query and document pairs to features
   (c) Train the `LightGBM` using the features

   train your L2R model
3. Then, for a new query...
   (a) Retrieve *potentially-relevant documents* using a simplified scoring function (e.g., BM25) and a `Ranker` (much like Homework 1!)
   (b) Separate out the top 100 most potentially-relevant documents using the simplified scoring function and the remaining documents, if any
   (c) Construct the feature vectors for each query-document pair in the top 100
   (d) Use your L2R model to *re-rank* these documents[2]
   (e) Combine the re-ranked top-100 and the remaining ranked documents (if any) and return those as the result.

■ **Problem 5.** (5 points) Which features does the model care about? Once you have finished training your model, let's take a look by examining the feature importance scores in the model. Make a plot using the `feature_importances_` field of the LightGBM model and show its value for each of the features *except* the binary has-category features (this should result in far fewer). Be sure to label the features by name (you should know which features are in which order based on how you construct the feature vector). In a few sentences, describe what you see in the plot in terms of which features are most useful. Do any surprise you?

[2] They were already ranked before, but we're going to use a more sophisticated L2R to better order them!

■ **Problem 6.** (5 points) Following up on the previous problem, which of the binary features for the presence of top article categories were useful for ranking? (There were 118 of these features.) Sort the categories by their feature importance scores and add a table with the top 5 categories and their score. Which if any are useful when compared to the scores for the non-category features?

■ ***Optional*** **Problem 2.** (0 points) The current model training doesn't make use of the development data at all, in part, to keep this assignment simpler. However, most machine learning systems do support including the development data during training to assess when to stop. The LightGBM ranking model does support this too! All you would need to do is featurize the development data and include it as an argument. As a *purely optional exercise*, try extending the code so that the development data is loaded and used during training.

6 Evaluation

How good is our new L2R based model? We'll score the outputs using MAP and NDCG and compare against our baseline BM25 retrieval system.

■ **Problem 7.** (10 points) Train the L2R system using all the features. Also, rank the results using a BM25 ranker with the default hyperparameters. Score the predictions for both models using the test set. We recommend training the L2R in code outside of the files provided with the class, like in a Jupyter notebook. You can import the relevant classes and train separately, which will make plotting a lot easier. Score your ranking function using MAP@10 and NDCG@10. Plot the mean performance across all queries using a bar plot using different bar colors for each of the models. If you use Seaborn, you can have it plot the mean and show a 95% confidence interval, which is nice. In a few sentences, describe what you see.

■ ***Optional*** **Problem 3.** (0 points) How do the hyperparameters for LightGBM influence the L2R model's performance? Try configuring the L2R model with different hyperparamters and report how changing each influences the performance on the development and the test sets. Does the best performing model on the development data also perform best on the test data? What might that tell you about the stability of the performance? You can use a table or a figure to show your results on different hyperparameter settings.

■ ***Optional*** **Problem 4.** (0 points) What happens if we use a different method to initially rank our documents? Would we get a better set to re-rank using our L2R model? Try ranking initially using other relevance scoring methods (e.g., just Pivoted Normalization). How does this influence the performance on the test set?

■ ***Optional*** **Problem 5.** (0 points) Try ranking by changing the different hyperparameters of the IR system other than the IR model (e.g., increasing the minimum word frequency). What effect do these changes have on performance?

7 Grading and Points

Answers to the questions above constitute 40 points. A correct implementation of the code is worth 60 points. Partial credit will be given wherever possible, provided you show your work.

8 Late Policy

You have **three** free late/flex days for this course. All late days are managed through the autograder. Any submission that is after the scheduled time on Canvas (or the autograder) will use one late day. You do not need to ask permission for free late days. We intend you to use them *first* before asking for any other forms of extensions, unless you experience a serious and unexpected life event.

9 What to Submit

Autograder Submission Procedure Your implementation of all the files should be uploaded to the Homework 2 assignment in https://autograder.io. You should already be added as a student to the course but if you do not see the course there, please notify the instructional team immediately to be added. The autograder will tell you which files you have to upload.

1. Your code to the autograder
2. A PDF with your answers and plots to all questions

Academic Honesty Policy

Unless otherwise specified in an assignment all submitted work must be your own, original work. Any excerpts, statements, or phrases from the work of others must be clearly identified as a quotation, and a proper citation provided. Re-using worked examples of significant portions of your own code from another class or code from other people from places like online tutorials, Kaggle examples, or Stack Overflow will be treated as plagiarism. **The use of ChatGPT or Co-Pilot for generating solutions is prohibited; any solution using these tools will receive an automatic zero, regardless of how much of the code these tools were used for.** The instructors reserve the right to have you verbally describe the implementation of any part of your submission to assess whether you wrote it. If you are in doubt on any part of the policy, please contact one of the instructors to check.

Any violation of the University's policies on Academic and Professional Integrity may result in serious penalties, which might range from failing an assignment, to failing a course, to being expelled from the program. Violations of academic and professional integrity will be reported to Student Affairs. Consequences impacting assignment or course grades are determined by the faculty instructor; additional sanctions may be imposed.