Dynamic Ranking

February 9, 2011

1 Introduction

This is an implementation of the experiments described in C. Brandt, T. Joachims, Yisong Yue, J. Bank, Dynamic Ranked Retrieval, ACM International Conference on Web Search and Data Mining (WSDM), 2011.

The provided program compares the performance of the dynamic myopic (DM) and dynamic lookahead models (DL), which adapt to better serve diverse queries, to the static myopic (SM), or standard ranking model. The input data is assumed to be a set of query topics, each of which has several subtopics (or facets) that users search for. Each such subtopic is represented by a different user profile, and has a different set of documents which are relevant to it. The code can then be used to compare the performance of adaptive ranking versus nonadaptive ranking.

As in the paper, there are two versions of these tests: in the first, the algorithm has full knowledge of the user profiles, although when interacting with a user, it does not know that user's specific profile. User behavior can be deterministic (a user clicks on any relevant document) or nondeterministic (the user may sometimes refrain from clicking on a relevant document, or click randomly on an irrelevant document). In this case, the full-knowledge algorithm knows whether the user is random, but does not know if a given click is due to relevance or spurious clicks.

Please direct any questions, comments, or problems to ccb35@cornell.edu.

2 Quick Start: Running an example

All examples can be found in the "examples" directory; both the TREC 06-08 ("Interactive") and TREC 09 ("Web") datasets are provided.

The code is implemented in python 3.1; python 3.X is necessary to run it. (if 3.1 is not your default version, you can still install it and run it; for example, as `/usr/local/bin/python3.1 <function name>`)

Before running the code, you need to add /py_dynamic_ranking/src/ to your path. For example,

```
export PYTHONPATH = ~/fk_ddr/src/
```

The "examples" directory contains two subdirectories: `interactive_input` and `interactive_qes`. Each directory contains a python file, `fk_run.py` . This script contains all necessary functions to run an example. Each of the `fk_run.py` scripts runs 5 functions. The first function call (`process_qrels.py`) generates input files from the qrel (query relevance) files (found at `http://trec.nist.gov/`). The other functions generate query events, create paths through rankings using the specified algorithms, and evaluate these paths. This process is described in more depth below.

To run the examples, simply run the python script in the `fk_run.py` file.

2.1 Full knowledge interactive (fk_interactive)

The directory (fk_interactive) contains the interactive (TREC 6-8 Interactive Track) dataset. Input is generated from the included query relevance file, `qrels_interactive_full`.

As in the paper, the weight of a profile is proportional to the number of documents relevant to the profile. The `fk_run` file runs and evaluates with DCG over both DM and SM. By changing the parameter `algorithmType` in the function call of `createPaths`, the algorithms utilized to generate the ranking can be

altered.. In addition, the user can set and change the desired values for nondeterministic user behavior. By changing the values of the variables `prClickIfRel,prClickIfNotRel`, the probability that the users will click at random can be changed. The parameters for algorithm, utility metric, and nondeterminism are provided to show their syntax, but set at their default values. As in the paper, the probability of a particular subtopic is weighted by the number of documents in the subtopic; this distribution is set in the profile file.

2.2 Full knowledge web (fk_web)

As before, `fk_run` provides an example of running the algorithm using DM and SM with utility metric DCG and deterministic users. Again, the `process_qrels` file provided can be used to generate the input file from the query relevance file, `qrels_clueweb.txt`; the format of the qrel file for web is slightly different so note that this function is different than the interactive one.

Also note that as in the paper, profiles are weighted uniformly rather than, as with the interactive set, the number of documents.

3 Program Overview

The program has two packages: `ranking`, which contains the bulk of the program code, and `utils`, which contains some general utility functions used in the program. The package `examples` contains all of the input and code necessary to run the program on the web and interactive datasets used in the paper.

3.1 Input format and parsing input: `readInputFl`

3.1.1 Input format

The expected input is a directory of input files, each representing a separate topic, and each topic with a unique topic id. Each topic file must have a unique "topic id", and the file name should be `topicId.txt`. It is very important for the file name to be the topic id plus the extension `.txt`.

The input file for a particular topic contains information about the topic's candidate set of documents (the set of possible documents to be used in the ranking) and the topic's profiles (the set of possible user roles). Each line in the file contains the information for one profile. It contains the topic id, the profile id, the weight of the profile (that is, the relative likelihood of choosing this profile at random; can be set as 1 if all profiles are equally likely), and a subset of documents in the candidate set and their relevance to this particular profile.

The format of each line is:

```
<topicid> <profileid> <profile weight> <d1>:<relevance of d1> <d2>:<relevance of d2>...
```

where `d1,d2,...` are document ids. The format for listing these documents is quite flexible. The documents do not have to be listed in any particular order, and both relevant and irrelevant documents can be listed for any particular profile, with their relevance or irrelevance to that profile noted as a binary value. To designate a document `d1` as relevant for that profile, write `d1:1`. To designate the document as irrelevant, write `d1:0`. All documents listed at any point in the file are considered part of the candidate set, so documents which are irrelevant to all profiles, but still part of the candidate set, only need to be listed in one profile (although it is also fine to list them in multiple profiles).

Example: a topic 555, with 3 profiles. Profile 1 has relevant documents d1, d2, d3, profile 2 has relevant documents d2 and d3, and profile 3 has relevant documents d4, d5. All the profiles have equal weight. Topic 555 has two additional documents in the candidate set, d0 and d6, which are not relevant to any profile. To save space, they are listed only in the document listing for profile 1, although also listing them in the document list for 2 or 3 would not change the interpretation of the file.

```
555 1 1 d1:1 d2:1 d3:1 d6:0 d0:0
555 2 1 d2:1 d3:1
555 3 1 d4:1 d5:1
```

3.1.2 `readInputFl.py`

This module contains code to parse the input file and produce the an object which represents a topic's set of profiles and candidate set. It is utilized by `createQEvents`, `createPaths`, and `evaluatePaths`, and does not need to be called directly by a program user.

3.2 Creating query events: `createQEvents`

A "query event" represents a single instance of a user performing a search for a specified topic, where the user is characterized by a specific user profile. Each "query event" is assigned a unique "instance id". The user profile of a query event is chosen randomly according to the profile weights specified in the input file.

The python function `createQEvents.createQEvents(inputDir,outputDir)` with optional parameters `numQEvents=10,prClickIfRel=1,prClickIfNotRel=0,verbosity=1` is used to create a set of query event files, one file for each topic in inputDir, and writes them in outputDir.

A query event specifies the user's profile id, which, when combined with the input file, specifies all of the documents which are relevant to that user, and a set of documents that hte user clicks on. For deterministic users, the list of clicked documents is the same as the list of documents relevant to the profile. For for nondeterministic users, the set of clicked documents is calculated using the parameters "prClickIfRel" and "prClickIfNotRel". For each document, a random value is chosen uniformly between 0 and 1; if the document is relevant to the user and the value is less than "prClickIfRel", the document is placed in the click list; if the document is not relevant and the value is less than "prClickIfNotRel", then the document is placed on the click list.

The parameters of

```
ranking.createQEvents.createQEvents(inputDir, outputDir,
       numQEvents, prClickIfRel, prClickIfNotRel, verbosity)
```

- inputDir: the directory holding the input files
- outputDir: the directory which will hold the query event files created by the function.
- numQEvents: number of query events to create for each topic.
- prClickIfRel: probabliity that a user will click on a relevant document presented to them; if the user behaves deterministically, this is 1.
- prClickIfNotRel: probability that a user will click on a document presented to them if it is not relevant to their profile; if the user behaves deterministically, this is 0.
- verbosity: constrols how much information is provided in the console; default is 1.

For each query event created by the function, the program generates an instance id at random, chooses a query profile according to the distribution specified in the input file, and generates the set of clicked documents by examining each document in the candidate set and selecting at random whether it will be clicked according to prClickIfRel and prClickIfNotRel. For each topic, an query event file is generated. This file has one query event on each line, specified in the following form

```
<instid> <topic id> < profile id> : <clicked doc 1> <clicked doc2> ... <clicked doc n>
```

An example of output for our example could be the following.

```
inst1 : 555 : 1 : d1 d2 d3 d6
inst2 : 555 : 2 : d2 d3
inst3 : 555 : 1 : d1 d3
```

The file above represents three different query events for topic 555, for profiles 1, 2, and 1, respectively. This exhibits a case where users are not deterministic, because a user with profile 1 occurs twice yet clicks on a different document set; in inst1, d6 is not relevant to profile 1, but the user clicks on it anyway, and in inst3, d2 is relevant to profile 1, but the user doesn't click on it. Note, however, that both instances have the same set of relevant documents (as specified by the profile file)

3.3 Creating rankings: `createPaths and calcUtilities`

When the function `createPaths` is run, it reads the set of query events and simulates each query event on a ranking, then outputs the resulting path. Rankings are created according to the algorithms Static Myopic (SM), Dynamic Myopic (DM), and Dynamic Lookahead (DL) discussed in the paper. The interaction between a user and a dynamic ranking can be though of as a path through a tree of potential user paths. Since this tree is exponential in the length of the ranking, rather than generating the full ranking, we interactively generate the user's path through the tree.

Paths are generated by the `createPaths` function in the eponymous module. For each topic, this function reads in the corresponding input and query event files and generates one path for each query event. For SM, since the ranking is independent of user behavior, the ranking can be generated without interacting with the user. For DM and DL, the algorithm chooses a document and simulates "presenting" it to the user. To obtain the user's feedback, it checks the corresponding query event to see if the document presented appears among the list of clicked documents. Using this feedback, the algorithm chooses another document and repeats the process. Both myopic methods choose the documents which have the highest predicted utility to the user given the user's previous examples. As described in the paper, the lookahead method chooses the document which provides the highes utility, which is estimated by finding a lower bound on the utility that could be obtained from the candidate document and the best subtrees that could be placed below it. The parameters of the function

```
createPaths.createPaths(inputDir, qEventDir, outputDir,
                utilityMetric=['DCG'],algorithmType=['SM','DM'],
                cutoff=10, prClickIfDocRel=1,
                prClickIfDocNotRel=0, verbosity=2)
```

- inputDir: the directory holding the input files
- qEventDir: the directory holding the query events generated
- outputDir: the directory to write the path files created
- utilityMetric: a list of utility metrics used by the algorithm to select the best documents to place in the ranking. `utilityMetric` $\subseteq$ `['PREC','DCG','NDCG','MAP']`. Note that if more than one metric is given, the algorithm will run each in turn on all provided algorithm types.
- algorithmType: A list of algorithms to run. `algorithmType` $\subseteq$ `['SM','DM','DL']`. You can run and evaluate several different algorithms: the static myopic (SM), the dynamic myopic (DM) and the dynamic lookahead (DL), as discussed in the paper. If multiple algorithms are chosen here, then each of the algorithms will be run using each utility metric. Note that DL is slow and performs comparably (actually, usually slightly worse) than the faster DM.
- cutoff: length of ranking to produce
- prClickIfRel: probabliity that a user will click on a relevant document presented to them; if the user behaves deterministically, this is 1. This provides the algorithm with information about how randomly the users will behave, which affects the probability of a particular profile given a user interaction and therefore affects a docuemnt's expected utility.
- prClickIfNotRel: probability that a user will click on a document presented to them if it is not relevant to their profile; if the user behaves deterministically, this is 0.
- verbosity: constrols how much information is provided in the console; default is 1.

The algorithm can optimize according to any arbitrary utility function. The implemented utility functions are MAP (mean average precision), DCG (discounted cumulative gain) nDCG, and precision@k. The estimation of the utility to be gained by adding a particular document is performed in the `calcUtilities` module. This module provides cunctions to estimate the utility of a single document or the utility of a document set. It is utilized both in creating and evaluating paths.

For each topic-algorithm-utility metric combination, the function produces a file in the outputDir. The files have the following naming convention:

`<topic-id>_<algorithmType>_<utilityMetric>.pth`

For example, a file where the path was generated on topic 555 using dynamic myopic as the algorithm and the utility metric DCG to generate the path would be stored in a file called `555_DM_DCG.pth`. Each path file has one line for each query event, representing the user's path in the ranking. The line contains the instance id, topic id, and user profile, each separated by a space. It then provides a list of the documents that the user interacts with, and the result of the interaction: for example, "d3:1 d2:0" represents a user interaction where the user first saw document d3 and clicked on it, then was presented with document d2 and did not click on it. The format of the lines is

`<inst id> <topic id> <profile id> <d1>:<0/1> <d2>:<0/1>`

example line:

`inst2 555 2 d4:0 d2:1 d1:0 d3:1 d5:0`

This says that for query event with id inst2, the documents presented were d4, d2, d1, d3, d5, and the user clicked on d2 and d3 (note that these click/no click reactions were specified by the query event file.)

3.4 Evaluating rankings: `evaluatePaths and summarizeEvals`

The paths generated by the program can be evaluated on standard metrics (MAP, DCG, NDCG, precision@k) using the `evaluatePaths` function. It accepts the following input:

`evaluatePaths(inputDir, pathDir, outputDir, metrics=['DCG'], cutoff=10, verbosity=1)`

- inputDir: the directory holding the input files
- pathDir: the directory of paths produced with `createPaths`
- outputDir: the directory to write the output (.eval) files to. One evaluation file is produced for each path file-evaluation metric combination.
- metrics: a list of metrics to evaluate each path on. `metrics` $\subseteq$ `['PREC','DCG','NDCG','MAP']`
- verbosity: constrols how much information is provided in the console; default is 1.

The `evaluatePaths` function evaluates the utility of each path by using the path's user profile id to look up the true relevant document set for that particular profile. It utilizes the `calcUtilities` function to calculate the path utility.

For each evaluation metric-path file combination, an evaluation file is produced in the outputDir. The files have the following naming convention:

`<topic-id>_<algorithmType>_<utilityMetric>_<evaluationMetric>.eval`

For example, a file where the path was generated on topic 555 using dynamic myopic as the algorithm and the utility metric DCG to generate the path, then evaluated using MAP would be stored in a file called `555_DM_DCG_MAP.eval`. Each line in an evaluation file represents the results of the interaction between one query event instance and the ranking. It contains the query event instance id, topic, user profile, and value of the chosen metric for that particular instance, all separated by spaces.

`<instance id> <topic id> <profile id> <value of evaluation>`

example file: the following might come from a file,

`555_DM_gmetric_DCG_emetric_DCG_tst.eval`

indicating that the dynamic myopic algorithm was run over an instance taken from topic 555, where the algorithm utilized DCG as its evaluation metric to create the file, and the file is evaluated on DCG.

```
inst1 555 1 0.23
inst2 555 2 1.6544
inst3 555 1 1.22
```

Since the results of this evaluation can be difficult to interpret, a second function is provided in `summarizeEval` to provide more compact feedback. The following parameters are used in

`summarizeEval(evalsDir,summOutFl=None, verbosity = 1)`

- evalsDir: directory where eval files were written out to.
- summOutFl: (optional) a file name where the summary results produced can be written
- verbosity: constrols how much information is provided in the console; default is 1.

This function calculates the mean value of each algorithm-generation utility-evaluation utility combination and reports it in the console, or writes it out if a file is specified.

3.5 `utils Module`

The utils module contains several general utilities used throughout the program.

3.5.1 `utils.flExts`

Contains a list of the file extensions used to specify each type. As default, the extension for input files is '.txt', query events '.qes', paths '.pth', and evaluation files '.eval'. These extensions can be changed by modifying this file.

3.5.2 `utils.utils`

Contains several useful utilities for stripping comments from files, getting a list of files, and deleting temporary files on exit.

3.5.3 `utils.errorWr`

Controls output and style of error messages produced by the functions.

4 Adding New Utility Functions

The current implementation can evaluate using DCG, NDCG, precisionk, and MAP. To add an additional evaluation metric, only one function needs to be altered: `getMetricFunction(metric)`. This function returns a reference to a function which takes, as input, the following 3 parameters

5 Adding New Algorithms

To add and test a new algorithm, modify the _getBestDoc() function in createPaths. The _getBestDoc() function returns a list of "next best documents" sorted in decreasing order of estimated utility. In the case of SM, it returns a list (of size cutoff) of "best documents" because the utility of a document in SM is independent of documents in previous positions. For DL and DM, however, only one document is returned because the utility of a document is dependent on the documents above. The _getBestDoc() function takes as input the following:

- algType the algorithm to use (SM, DM, DL)
- predictDocRelsFun the function to use to evaluate relevance
- currPath the current path
- candDocsSet the set of documents to choose the next best document from.
- cutoff the length of the ranking to produce (used in estimating utilities in DL)
- metric the metric to use to calculate, e.g. DCG; this is used for DL.

To add a new algorithm, just give it a name (e.g. 'myAlg') and add a conditional expression "elif algType=='myAlg'".

6 References

- C. Brandt, T. Joachims, Yisong Yue, J. Bank, Dynamic Ranked Retrieval, ACM International Conference on Web Search and Data Mining (WSDM), 2011.