1. General Usage

Open the terminal under the RankLib directory and type in

```
> java -jar bin/RankLib.jar
```

You should see all necessary parameters as below Parameters in the square brackets are optional with their default value shown at the end of the line

Usage java jar RankLib jar Params	
Params	
[+] Training tuning and evaluation	
train file	Training data
ranker type	Specify which ranking algorithm to use
	0 MART gradient boosted regression tree
	1 RankNet
	2 RankBoost
	3 AdaRank
	4 Coordinate Ascent
	6 LambdaMART
	7 ListNet
	8 Random Forests
[ ]feature file	Feature description file list features to be considered by the learner each on a separate line If not specified all features will be used
[ ]metric2t metric	Metric to optimize on the training data Supported MAP NDCG k DCG k P k RR k ERR k default ERR 10
[ ]gmax label	Highest judged relevance label It affects the calculation of ERR default 4 i e 5 point scale 0 1 2 3 4
[ ]-silent	Do not print progress messages which are printed by default
[ ]validate file	Specify if you want to tune your system on the validation data default unspecified If specified the final model will be the one that performs best on the validation data
[ ]tvs x in 0 1	Set train validation split to be x 1 0 x
[ ]save model	Save the learned model to the specified file default not save
[ ]test file	Specify if you want to evaluate the trained model on this data default unspecified
[ ]tts x in 0 1	Set train test split to be x 1 0 x tts will override tvs
[ ]metric2T metric	Metric to evaluate on the test data default to the same as specified for metric2t
[ ]norm method	Normalize feature vectors default no normalization Method can be
	sum normalize each feature by the sum of all its values

		zscore normalize each feature by its mean standard deviation
		linear normalize each feature by its min max values
	[ ]-kcv k	Specify if you want to perform k fold cross validation using ONLY the specified training data default NoCV
		-tvs can be used to further reserve a portion of the training data in each fold for validation
	[ ]-kcvmd dir	Directory for models trained via cross validation default not save
	[ ]-kcvmn model	Name for model learned in each fold It will be prefix ed with the fold number default empty
-	[-] RankNet specific parameters	
	[ ]-epoch T	The number of epochs to train default 100
	[ ]-layer layer	The number of hidden layers default 1
	[ ]-node node	The number of hidden nodes per layer default 10
	[ ]-lr rate	Learning rate default 0 00005
-	[-] RankBoost specific parameters	
	[ ]-round T	The number of rounds to train default 300
	[ ]-tc k	Number of threshold candidates to search 1 to use all feature values default 10
-	[-] AdaRank specific parameters	
	[ ]-round T	The number of rounds to train default 500
	[ ]-noeq	Train without enqueuing too strong features default unspecified
	[ ]-tolerance t	Tolerance between two consecutive rounds of learning default 0 002
	[ ]-max times	The maximum number of times can a feature be consecutively selected without changing performance default 5
-	[-] Coordinate Ascent specific parameters	
	[ ]-r k	The number of random restarts default 5
	[ ]-i iteration	The number of iterations to search in each dimension default 25
	[ ]-tolerance t	Performance tolerance between two solutions default 0 001
	[ ]-reg slack	Regularization parameter default no regularization
}-	[-] {MART LambdaMART specific parameters	
	[ ]-tree t	Number of trees default 1000
	[ ]-leaf l	Number of leaves for each tree default 10
	[ ]-shrinkage factor	Shrinkage or learning rate default 0 1
	[ ]-tc k	Number of threshold candidates for tree spliting 1 to use all feature values default 256
	[ ]-mls n	Min leaf support minimum samples each leaf has to contain default 1
	[ ]-estop e	Stop early when no improvement is observed on validaton data in e consecutive rounds default 100

- [-] Random Forests specific parameters	
[]-bag r	Number of bags default 300
[]-srate r	Sub sampling rate default 1 0
[]-frate r	Feature sampling rate default 0 3
[]-rtype type	Ranker to bag default 0 i e MART
[]-tree t	Number of trees in each bag default 1
[]-leaf l	Number of leaves for each tree default 100
[]-shrinkage factor	Shrinkage or learning rate default 0 1
[]-tc k	Number of threshold candidates for tree spliting 1 to use all feature values default 256
[]-mls n	Min leaf support minimum samples each leaf has to contain default 1
[]-estop e	Stop early when no improvement is observed on validaton data in e consecutive rounds default 100
[+] Testing previously saved models	
-load model	The model to load
-test file	Test data to evaluate the model specify either this or rank but not both
-rank file	Rank the samples in the specified file specify either this or test but not both
[]-metric2T metric	Metric to evaluate on the test data default ERR 10
[]-gmax label	Highest judged relevance label It affects the calculation of ERR default 4 i e 5 point scale 0 1 2 3 4
[]-score file	Store ranker s score for each object being ranked has to be used with rank
[]-idv	Print model performance in test metric on individual ranked lists has to be used with test
[]-norm	Normalize feature vectors similar to norm for training tuning

2. Examples

Go to the LETOR website and download any of their datasets For instance let s pick MQ2008 from the LETOR 4 0 dataset Suppose we put it under the *RankLib* directory

2.1. Training on held -out data

Type this into the command line

```
> java -jar bin/RankLib.jar -train MQ2008/Fold1/train.txt -test MQ2008/Fold1/test.txt -validate
```

What we specified means we want to train a LambdaMART ranker train on the training data and record the model that performs best on the validation data The training metric is NDCG 10 After training is completed evaluate the trained model on the test data in ERR 10 Finally the model will be saved to a file named *mymodel txt* in the current directory

The parameter validate is optional but it often leads to better models In particular validate is very important for RankNet MART LambdaMART Coordinate Ascent on the other hand works pretty well without validation data ~~As a result RankLib s implementation of Coordinate Ascent completely ignores the validation data to reduce training time this is the only exception in RankLib~~ Starting from **version 2 1 patched**, Coordinate Ascent will utilize validation data if specified just like any other algorithms

Important Note: metric2t e g NDCG ERR etc only applies to list wise algorithms AdaRank Coordinate Ascent and LambdaMART Point wise and pair wise techniques MART RankNet RankBoost due to their nature always use their internal RMSE pair wise loss as the optimization criteria Thus metric2t has no effects on them ListNet is a special case Despite being a list wise algorithm it has its own optimization criteria as well Therefore metric2t also has no effect on ListNet

> **Tips**:-Instead of using a separate validation dataset you can use tvs
> **Tips**:-Instead of using a separate test dataset you can use tts

2.2. k-fold cross validation

Although the LETOR dataset comes with train test validation data in this example let s pretend that we only had MQ2008 Fold1 train txt as the only dataset We now want to do 5 fold cross validation experiment

a) Sequential partition

```
> java -jar bin/RankLib.jar -train MQ2008/Fold1/train.txt -ranker 4 -kcv 5 -kcvmd models/ -kcvmn
```

The command above will sequentially split the training data into 5 chunks of roughly equal size The i th chunk is used as the test data for the i th fold The training data for each fold consists of the test data from all other folds

RankLib will train a Coordinate Ascent model ranker 4 on each fold that optimizes for NDCG 10 metric2t NDCG 10 This model is then evaluated on the corresponding test data for the current fold using ERR 10 metric2T ERR 10 After the training process completes RankLib will report the overall performance across all folds and save all 5 models one for each fold to the specified directory *models* : *f1 ca*, *f2 ca*, *f3 ca*, *f4 ca*, and *f5 ca*

> **Tips**:-You can use tvs to reserve a portion of training data in each fold for validation

b) Randomized partition

Let s say in the training data ranked lists e g queries are ordered in a certain way such that sequentially partitioning them might introduce some bias We want k partitions such that each partition contains a random portion of the input data We can do this simply by shuffling the order of ranked lists in the training data

```
java -cp bin/RankLib.jar ciir.umass.edu.features.FeatureManager -input MQ2008/Fold1/train.txt -o
```

The command above will create the shuffled copy that we want called train txt.*shuffled*", stored in the specified output directory *mydata* ."Cross validation can then be done using the command above but with the shuffled data instead

c) How do I obtain the data used in each fold?

To do this type

```
java -cp bin/RankLib.jar ciir.umass.edu.features.FeatureManager -input MQ2008/Fold1/train.txt.sh
```

This will extract and store the train test data used in each fold which is **exactly the same** as the in memory partitions used for learning Partitioning is ***always*** done sequentially

> **Tips** Type java cp bin RankLib jar ciir umass edu features FeatureManager for help

2.3. Evaluating previously trained models

```
> java -jar bin/RankLib.jar -load mymodel.txt -test MQ2008/Fold1/test.txt -metric2T ERR@10
```

This will evaluate the pre trained model stored in *mymodel txt* on the specified test data using ERR 10

2.4. Comparing models

Let s assume our test data *test txt* contains a set of queries and lists of documents or more precisely their feature vector retrieved for each of the queries using BM25 Let s say we have trained two models *ca model txt* (a Coordinate Ascent model and *lm model txt* (a LambdaMART modeL from the same training set The task is to see if using the Coordinate Ascent model and the LambdaMART model to re rank these BM25 ranked lists will improve retrieval effectiveness @NDCG 10

It goes like this

```
> java -jar bin/RankLib.jar -test MQ2008/Fold1/test.txt -metric2T NDCG@10 -idv output/baseline.n
> java -jar bin/RankLib.jar -load ca.model.txt -test MQ2008/Fold1/test.txt -metric2T NDCG@10 -id
> java -jar bin/RankLib.jar -load lm.model.txt -test MQ2008/Fold1/test.txt -metric2T NDCG@10 -id
```

Each of the output files specified with idv provides the ndcg 10 each system achieves for each of the test queries These files are stored in the *output* directory Note that these commands are different from the one used in Section 2 3 above which only reports the average measure e g ndcg 10 across all queries These 3 commands on the other hand report ndcg 10 on each of the queries not just the average

Here s an example of an output file showing individual and all query performance levels in terms of the selected metric

```
    NDCG@10   170   0.0
    NDCG@10   176   0.6722390270733757
    NDCG@10   177   0.4772656487866462
    NDCG@10   178   0.539003131276382
    NDCG@10   185   0.6131471927654585
    NDCG@10   189   1.0
    NDCG@10   191   0.6309297535714574
    NDCG@10   192   1.0
    NDCG@10   194   0.2532778777010656
    NDCG@10   197   1.0
    NDCG@10   200   0.6131471927654585
    NDCG@10   204   0.4772656487866462
    NDCG@10   207   0.0
    NDCG@10   209   0.123151194370365
    NDCG@10   221   0.39038004999210174
    NDCG@10   all   0.5193204478059303
```

Now to compare them do this

```
> java -cp bin/RankLib.jar ciir.umass.edu.eval.Analyzer -all output/ -base baseline.ndcg.txt > a
```

The output file *analysis txt* is tab separated Copy and paste it into any spreadsheet program for easy viewing Everything should be self explanatory It looks like this

```
Overall comparison
------------------------------------------------------------------------
System  Performance     Improvement     Win     Loss    p-value
baseline_ndcg.txt [baseline]    0.093
LM_ndcg.txt     0.2863  +0.1933 (+207.8%)       9       1       0.03
CA_ndcg.txt     0.5193  +0.4263 (+458.26%)      12      0       0.0

Detailed break down
------------------------------------------------------------------------
           [ < -100%)  [-100%,-75%)  [-75%,-50%)  [-50%,-25%)  [-25%,0%)  (0%,+25%]  (+25%,+50
LM_ndcg.txt    0        0           1            0            0         4            2
CA_ndcg.txt    0             0            0            0            0        1            6
```

This output shows performance comparisons of two saved models CoordinateAscent and LambdaMART against a baseline The table shows performance differences and percent improvements between each saved model and the baseline Also numbers of queries that were better or worse than baseline and P value for statistical confidence in the better model Note only queries where performance metrics showed different values are listed in the win loss columns

The final part of the output is a simple histogram of performance differences over percent change intervals Again only queries that provided differences in metric values are listed and the percent values actually represent difference values between base and test in chosen metric X 100

> **Tips:** Type java cp bin RankLib jar ciir umass edu eval Analyzer for help

2.5. Using trained models to do re-ranking

Instead of using a model to re rank documents in some test data and examining the effectiveness of the final rankings as shown in 2 3 we want to get a hold of the final rankings themselves i e these rankings might serve as input to some other systems

This can be achieved by

```
> java -jar bin/RankLib.jar -load mymodel.txt -rank MQ2008/Fold1/test.txt -score myscorefile.txt
```

The output file *myscorefile txt* provides the score that the ranker assigns to each of the documents higher means more relevant To obtain the final ranking of documents simply sort the documents by this score with respect to each query you will have to write some codes to do this