Programming Assignment 4 – Learning to Rank

Due: Thursday, May 30, 2013, 4:00 PM

In the previous assignment, we have examined various ways of ranking documents given a query; however, weights for different features were not learned automatically but set manually. As more and more ranking signals are investigated, integrating more features becomes challenging as it would be hard to come up with a single ranking function like BM25 for arbitrary features. In this assignment, you will be investigating different approaches to the learning to rank task that you have learned: (1) the pointwise approach using *linear regression* and (2) the pairwise approach employing *support vector machines*. The goal is to let these algorithms learn weights automatically for various features.

1 Overview

This assignment should be done in teams of two or individually. More specifically, it involves the following tasks:

1. Implement an instance of the pointwise approach with *linear regression* based on basic tf-idf features (§2).
2. Implement an instance of the pairwise approach, the *ranking SVM* method, using basic tf-idf features (§3).
3. Experiment with more features such as BM25 and PageRank (§4).
4. Write up a summary report and answer questions we asked for each task.

5. For extra credit, you can implement other ranking approaches, e.g., instances of the pointwise/pairwise/listwise methods (§5).

1.1 Miscellaneous

In PA3, our training data included two files: (1) *queryDocTrainData* consisting of 288 queries along with documents retrieved for each query and (2) *queryDocTrainRel* listing relevance judgments for each query-document pair.

In this assignment, we use the same training data as in PA3 but split into two portions: (1) 173 queries for *training* (*queryDocTrainData.train*, *queryDocTrainRel.train*) and (2) 115 queries as a *development* test set (*queryDocTrainData.dev*, *queryDocTrainRel.dev*). The development test is used to tune your model parameters, choose features, etc. The idea is to avoid over-fitting parameters that may yield good performance on training data, but perform poorly on new unseen data. You are required to **train your models on the training set**, and **report performance evaluated on the development set** throughout. The evaluation metric used is **NDCG** as in PA3.

Note that there will be a **hidden test set** used in evaluating your work.

You need to submit your code (which we will run with an auto-grader) and a write-up via Coursera. See later sections for more details.

The description below on machine learning algorithms for the learning to rank task is largely based on Hang Li's tutorials at `http://research.microsoft.com/en-us/people/hangli/li-acl-ijcnlp-2009-tutorial.pdf` and `http://research.microsoft.com/en-us/people/hangli/l2r.pdf`.

2 Pointwise Approach and Linear Regression (Task 1 – 15%)

In ranking, each query q_i will be associated with a set of documents (like a group), and for each document j, we extract a query-document feature vector $\boldsymbol{x_{i,j}}$ as illustrated in Figure 1. There is also a label $\mathrm{y}_{i,j}$ associated with each query-document vector $\boldsymbol{x_{i,j}}$.

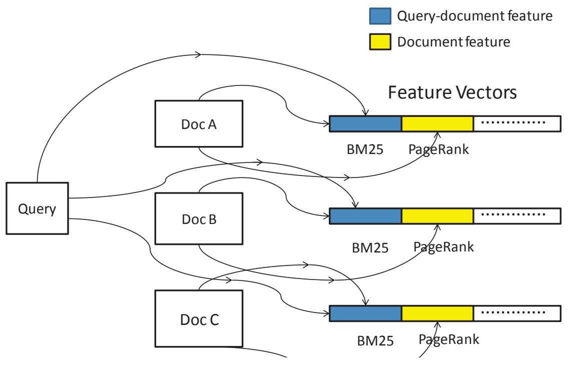

Figure 1: Feature Extraction.

In the pointwise approach, such group structure in ranking is ignored, and we simply view our training data as $\{(\boldsymbol{x_i}, \mathrm{y}_i)\}$, i=1 ... m, where each instance consists of a query-document feature vector $\boldsymbol{x_i}$ and a label y_i (which is a relevance score in our case, same as in PA3). The ranking problem amounts to learning a function f such that $f(\boldsymbol{x_i})$ closely matches y_i.

In this task, we consider a very simple instance of the pointwise approach, the *linear regression* approach. That is, we will use a linear function f which gives a score to each query-document feature vector $\boldsymbol{x}$ as follows $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$. Here, the weight vector $\boldsymbol{w}$ and the bias term b are parameters that we need to learn to minimize a loss function as defined below:

$$\sum_{i=1}^{m} (f(\boldsymbol{x_i}) - y_i)^2 \tag{1}$$

This formulation is also referred to as the *ordinary least squares* approach.

2.1 What to do

Here are the specific things you need to do for this task:

1. Represent each query-document pair as a five-dimensional vector of tf-idf scores, each of which corresponds to a field – url, title, header, body, and anchor.

   Specifically, given a query vector q (idf scores) and a term frequency vector t_f of a document field f, the tf-idf score is $q^\top t_f$. (Note: you could try using either the raw or the normalized term frequency vectors as in PA3.)

2. Train a linear regression model. We strongly recommend using the scikit-learn package `http://scikit-learn.org/` (see Section 7 for more details).

3. Given a learned weight vector $\boldsymbol{w}^*$, you can now directly compute score for each query-document vector x as $f(x) = \boldsymbol{w}^{*\top} \boldsymbol{x} + b$ and rank based on that score. **Report the NDCG performance** achieved on the development test data.

   How does this score compared to the performance obtained using the weights manually set in PA3? Do the values of w^* make sense (e.g., should a field be weighed higher than another)? **Report your finding**.

3 Pairwise Approach and Ranking SVM (Task 2 – 25%)

To recap, in the pairwise approach, ranking is transformed into a pairwise classification task in which a classifier is trained to predict the ranking order of document pairs. In other words, instead of giving a numeric rank to each document, e.g., rank 1, 2, 3 for documents A, B, C respectively, the classifier will judge if a document is better than another for each pair of documents, e.g., if A is better than B, B is better than C, and A is better than C. This is the idea behind the Ranking SVM method, an instance of the pairwise approach as illustrated in Figure 2. In this task, you will be applying your knowledge about Support Vector Machines (SVMs) to build such a classifier.

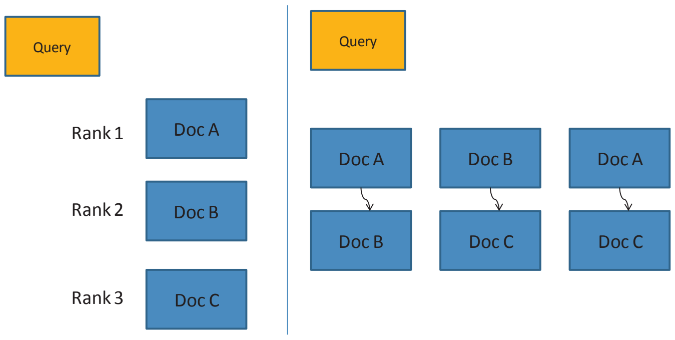

Figure 2: **Pairwise Classification**: if we have 3 documents with ranks as shown in the left side picture, then we produce pairwise ranking facts as in the right side picture, from which we proceed to train a classifier.

Specifically, instead of working in the space of query-document vectors, e.g, $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, and $\boldsymbol{x_3}$, we transform them into a new space in which a pair of documents is represented as the difference between their feature vectors (with respect to a query), e.g. $\boldsymbol{x_1}-\boldsymbol{x_2}$, $\boldsymbol{x_2}-\boldsymbol{x_3}$, and $\boldsymbol{x_1}-\boldsymbol{x_3}$. For each vector $\boldsymbol{x_i}-\boldsymbol{x_j}$, a label $+1$ is assigned if document i is more relevant than document j and for the reverse case, a label -1 is used.

Note that we do not make pairwise ranking facts out of either pairs of documents with the same relevance score or pairs of documents that were returned for different queries. If $\boldsymbol{x_i}-\boldsymbol{x_j}$ is a positive example, $\boldsymbol{x_j}-\boldsymbol{x_i}$ could be used as a negative one and vice versa, so using either $\boldsymbol{x_i}-\boldsymbol{x_j}$ or $\boldsymbol{x_j}-\boldsymbol{x_i}$ is sufficient.[1]

3.1 Linear SVM Formulation

Formally, training data for the ranking SVM is given as $\{(\boldsymbol{x_i}^{(1)}, \boldsymbol{x_i}^{(2)}), \mathrm{y}_i)\}$, i=1 $\ldots m$ where each instance consists of two feature vectors $(\boldsymbol{x_i}^{(1)}, \boldsymbol{x_i}^{(2)})$ and a label $\mathrm{y}_i \in \{+1, -1\}$. The learning is framed as a Quadratic Programming problem:

$$\begin{gathered}\min_{\boldsymbol{w},\boldsymbol{\xi}} \tfrac{1}{2}||\boldsymbol{w}||^2 + C\textstyle\sum_{i=1}^{m}\xi_i \\ \text{s.t. } y_i \boldsymbol{w}^\top \left(\boldsymbol{x_i}^{(1)} - \boldsymbol{x_i}^{(2)}\right) \geq 1 - \xi_i \\ \xi_i \geq 0, i = 1 \ldots m,\end{gathered} \tag{2}$$

where ξ_i are slack variables for soft-margin classification as you have learned in the SVM lecture and C is a regularization term to control over-fitting. The weight vector w corresponds to a linear function $f(x) = w^\top x$ which can score and rank documents.

3.2 What to do

Here are the specific things you need to do for this task:

[1]With respect to the Quadratic Programming formulation in Eq. 2, a negative instance essentially yields the same constraint as the positive one.

1. Using the same basic tf-idf features as in Task 1, construct training data for the pairwise approach.

2. Train a linear SVM classifier. We strongly recommend the use of the scikit-learn package `http://scikit-learn.org/` (see Section 7 for more details).

3. Given a learned weight vector $\boldsymbol{w}^*$, you can now directly compute the score for each document vector x as $f(x) = \boldsymbol{w}^{*\top}\boldsymbol{x}$ and rank based on that score. **Report the NDCG performance** achieved on the development test data.

4. Compare your ranking outputs given by the linear regression and the ranking SVM. Find 2-3 queries in which results of one method are markedly different from those of the other and use your relevance score file to judge which one is better.[2]

   Then, for each of these queries, pick a URL and examine the train data file (or the page content) with reference to the weight vectors learned to find out why that URL is ranked higher/lower in one method and lower/higher in the other. **Report your finding**.

   If you notice examples in which both methods fail, it would be useful to start thinking about why your systems did not work. That would help your next task!

4 More Features and Error Analysis (Task 3 – 50%)

With the machine learning algorithms built in previous tasks, let's try to use more signals. Here is a list of features that you could try:

1. Cosine similarity value computed with your best weights in PA3 Task 1.

2. BM25F value derived with your best weights in PA3 Task 2.

3. Smallest window features of different fields from PA3.

4. PageRank.

You could also be creative! Examine the different types of ranking errors your system made (similar to what you did in Task 3). Can you propose new features to fix them? For example, if your system tends to rank wrongly URLs that link to PDF documents, perhaps you might want to add a binary feature to indicate if a URL ends in ".pdf".

4.1 What to do

Here are the specific things you need to do for this task:

1. From the list of features above, find out which combinations of features help boost performance. **Report the NDCG scores** achieved on the development test data for these combinations.

[2] `http://kdiff3.sourceforge.net/` might be useful.

Note that, you need not consider all fields for features like window or tf-idf. Sometimes, you might find that leaving out a feature helps boost performance. It is up to you to use raw or scaled counts. **Document your choices in the report**.

2. Examine your ranking output, list 1-2 types of errors that your system tends to make, and suggest features to fix them. Do they help fix the problems you observed? What about the NDCG scores achieved on the development test data? **Report your finding**.

   You could also propose new features that help improve performance. **Report the best NDCG score** achieved on the development test data.

Note that not all features will be as helpful as you might have expected to your overall NDCG score (C'est la vie!). However, if a signal does not improve your ranking output for certain queries, we would still be interested to know that you have tried. We will evaluate based on the depth of your analysis.

5 Other Learning Methods (Extra Credit – 5-10%)

Please consider whether it is wise to attempt extra credit problems at this point in the quarter versus to practice exam problems for your final exam.

For extra credit, we suggest you to look at other learning methods as below and **pick only one extension**:

1. (5%) Add *regularization* to your linear regression model in Task 1: L_2 (Ridge Regression), L_1 (Lasso), or a combination of both (Elastic Net).[3] However, they are only effective if you have many features, and you need to analyze the effect of the regularizer that you chose.

2. (5%) Try the *Support Vector Regression*[4] for the pointwise approach.

3. (10%) Implement another instance of the pointwise approach, the *PRank algorithm*. This is a simple and efficient online algorithm with pseudo-code described in `http://books.nips.cc/papers/files/nips14/AA65.pdf`.

For the extension you tried, compare the NDCG performance achieved with the performance in Task 1. Briefly discuss why one method is better than the other.

If the above list is not challenging enough for you, try one of the following in your spare time – **we will not grade them** (technical details in `http://research.microsoft.com/en-us/people/hangli/l2r.pdf`):

1. Implement another instance of the pointwise approach, the *OC SVM* (SVM for Ordinal Classification).

[3]See `http://scikit-learn.org/0.13/modules/linear_model.html`.
[4]See `http://scikit-learn.org/stable/modules/svm.html`.

2. Implement another instance of the pairwise approach, the *IR SVM* (Ranking SVM for Information Retrieval).
3. Implement an instance of the listwise approach, the *SVM MAP*.

Since these approaches require modifications of the Quadratic Programming formulations, you might need a convex optimization library. There is one in Python `http://abel.ee.ucla.edu/cvxopt/userguide/coneprog.html#quadratic-programming`.

6 Deliverables

6.1 Input/Output format

The starter code contains a script named *l2r.sh* which can be invoked as below. **Please read this carefully for a smooth submission process**.

```
$ ./l2r.sh <task> <test_data_file>
```

The arguments are:

- *task*: either 1 (Task 1), 2 (Task 2), 3 (Task 3), or 4 (Extra Credit).
- *test_data_file*: test data file with the same format as files *queryDocTrainData*.

You will need to change the script content to: (1) build the ranking algorithm based on the task given and (2) output ranking results for the input test data. **Do not alter the arguments and make sure you use the best set of features that produces the best NDCG score on the development test data for each task**. If we cannot reproduce similar NDCG performances as you reported, marks will be deducted.

You can read in whatever training data you need to build your model but we will not pass that as inputs to the script (so in a nutshell, hardcode the relative paths to such files either in the script or in the code). The script should output (to **stdout**) for each query, the query followed by the documents (in the form of urls) in decreasing rank order specified by your ranking. You can print anything you want to stderr.

For example, if query $q1$ has three documents and the file is listed as follows:

```
query: q1
  url: http://xyz.com
  ...
  url: http://def.edu
  ...
  url: http://ghi.org
```

Iff your ranking algorithm gives a rank of 1 to ghi.org, 2 to xyz.com and 3 to def.edu, then you should output the order in the following format:

```
query: q1
 url: http://ghi.org
 url: http://xyz.com
 url: http://def.edu
query: q2
 url: ...
  .
  .
```

In your experiments, you could execute the following commands:

```
$ ./l2r.sh <task> queryDocTrainData.dev > ranked.txt
$ python ndcg.py ranked.txt queryDocTrainRel.dev
```

which will give you performance results on the development data.

6.2 Report

Answers to questions we asked for each task should be included in a write-up file *report.pdf*. If there is any important design choices undertaken or different configurations used in your machine learning algorithms that boost performances, you could report them. It must be a maximum of **4 pages long**. Reports that are too terse or those that do not contain enough analysis will not get full credit.

6.3 Partner

A list of people who worked together on the assignment, in *people.txt* One line per student:

```
<sunet id1>
<sunet id2>
```

7 Code Guide

We encourage you to use the *scikit-learn* toolkit in Python for this assignment as it has a wide range of libraries for various machine learning algorithms including those mentioned in this assignment. It will save you from intensive coding and you will benefit from it not only in PA4, but also in the long run. For a detailed installation guide, please see the Piazza post `https://piazza.com/class#spring2013/cs276/476`.

We provide a skeleton code *l2r.py* which contains stub functions that you will need to implement. Running as follows:

```
$ python l2r.py queryDocTrainData.train queryDocTrainRel.train
                <test_data_file> <task> > ranked.txt
```

where *test_data_file* is the test data file, e.g., "queryDocTrainData.dev", while *rank* is either 1 (Task 1), 2 (Task 2), 3 (Task 3), or 4 (Extra Credit).

7.1 Linear Regression

In *l2r.py*, you could set *task* to 0 to get started and familiarize yourself with the script, which will walk you through steps of training a linear regression model:

Step 1 – build training data X and labels y:

```
X = [[0, 0], [1, 1], [2, 2]]
y = [0, 1, 2]
```

Step 2 – train a linear regression model:

```
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(X, y)
```

Step 3 – build testing data X_1:

```
X1 = [[0.5, 0.5], [1.5, 1.5]]
```

Step 4 – predict labels y_1:

```
y1 = model.predict(X)
```

If everything is correct, your y_1 will be $[0.5, 1.5]$.

7.2 Support Vector Machines

In scikit-learn, to create an SVM model, use:

```
from sklearn import svm
model = svm.SVC(kernel='linear', C=1.0)
```

Here, we will only use a **linear kernel** (do not change it). You could vary the regularization term C, but the default value of 1.0 should work just fine.

7.3 Miscellaneous

To retrieve weights of a model, use:

```
weights = model.coef_
```

To get normalized weights:

```
import numpy as np
weights = model.coef_ / np.linalg.norm(model.coef_)
```

Dot product could be done as:

```
np.dot(X[ii], weights)
```

8 Grading

We will be evaluating your performance on a different test dataset, which will have queries drawn from the same distribution as the training set. The format for the dataset is the same as *queryDocTrainData*.

Task 1: 10% if your NDCG score on the development data is above 0.84 and 5% for answering questions in the report.

Task 2: 15% if your NDCG score on the development data is above 0.86 and 10% for answering questions in the report.

Task 3: 10% for experimenting with different combinations of features and reporting their performances, 15% for new features proposed with your rationale reported, 10% if your best set of features improves the performance in Task 2 considerably (relative to performances of your peers' systems), and 15% for the error analysis described in the report (scaled by the depth of your analysis).

Correctness/Code: 10%. A check to ensure that your code is doing the right things.

Extra Credit: 5-10%. See Section 5.

We will give extra credit for best ranking systems in the entire class, which is based on the NDCG scores computed on our hidden test data. 10% for the top 5 systems and 5% for the next 15 systems.

9 Submission

Make sure the shell script *l2r.sh* runs as explained in Section 7. Then, call the below command and select an option to submit (report, tasks 1, 2, 3, or extra credit):

```
python submit.py
```

The submit script will execute your model on the development and test data on your machine. It will also check to see whether you have the right number of query-document pairs for each data set. Remember, it is an honor code violation to knowingly take a look at the test set.

Since *ndcg.py* will be used to evaluate the NDCG scores of your model on the test data, your output file should conform to the format mentioned in 6.1. Additionally, we have provided starter code in Python for your ease. See the included README file.

As a last step, you will have to submit your code through Coursera. Zip your assignment directory using a zip archiver (without the PA1 corpus), name it as *SUNetid1_SUNetid2.zip* and upload it on the Coursera assignments page for the "Code" section using the simple uploader you have used before. Do not worry about making submissions for the other parts as we will populate those with scores using your reports and code.

Only one person in the group needs to submit the assignment (Please submit everything from the same member's SUNet ID and with the same *people.txt* file)