CECS 429 - Project Milestone 3

Final demo due during the final exam period. Paper (if applicable; online submission) due December 19.

Overview

In this milestone, you will complete your search engine project through the addition of a performance- or research-oriented feature.

Requirements

Each of the following requirements are worth a certain amount of points. Your group must select and implement additional features worth **at least P total points** from the following list, where P is equal to...

- **two times** the number of people in your group...
- **plus 2**, if *at least one* member of your group is enrolled in CECS 529.

Who Wrote the Federalist Papers?:

From 4 to 8 points. You get **2 points** for this option, **plus** the points associated with each of the classification methods (below) that you implement.

Solve the United States history mystery about the authorship of the *Federalist Papers*.

The *Federalist Papers* is a collection of 85 articles and essays written by Alexander Hamilton, James Madison, and John Jay promoting the ratification of the United States Constitution, published in 1787 and 1788. Each article was published using the pseudonym "Publius", and so at the time the actual authorship of each article was unknown. Eventually the authors identified which papers they had written, except for 11 papers which both Alexander Hamilton and James Madison claimed as their own. No one knows with absolute certainty who wrote the disputed papers, but through statistical analysis using your search engine, you will be able to make a solid guess.

Download the Federalist Papers corpus from BeachBoard. The folders JAY, HAMILTON, and MADISON form the three classes in the training set for your classifiers; each contains the documents that the three authors wrote without dispute. The folder DISPUTED contains 11 documents that are claimed by both Hamilton and Madison (numbers 49 through 57, 62, and 63), but in reality were written by only one of them. Your classifiers need information about the number of documents in each class, and the frequency that each term occurs in documents of each class; to make this information available, you should build one separate index **for each** of the three classes. You can then find, for example, $f_{t,c}$ for $t = judiciary$ and $c = Hamilton$, by retrieving asking your Hamilton corpus for the df_t of the term "judiciary".

You are now ready to begin your analysis. You must determine the authorship of all 11 disputed papers using one or both of the following classifiers:

1. **Bayesian classification**: (2 points)
   (a) Build the discriminating set of vocabulary terms T as described in lecture and in section 13.5 of the text, using mutual information ($I(t,c)$) for the selection.
   (b) Using those terms as the *evidence* and the three authors as the *classes*, train a Bayesian classifier on the papers whose authors are known: for each term t in the discriminating set, calculate $p(t|c_H)$, $p(t|c_M)$, and $p(t|c_J)$ using Laplace smoothing, where c_H, c_M, and c_J represent the classes for the authors Hamilton, Madison, and Jay respectively. This information will need to be saved/indexed for the next step.
   (c) For each unknown paper, calculate the most likely class c_d for that document using

$$\mathrm{c_d} = \underset{\mathrm{c} \in \mathrm{C}}{\mathrm{argmax}} \left[\log(p(c)) + \sum_{t_i \in T_d} \log(p(t_i|c)) \right],$$

   where c is each of the three author classes, $p(c) = \frac{N_c}{N}$ is the trained probability of a paper being in that class, T_d is the set of terms from the discriminating set T that are present in document d, and for each term t_i in that set, $p(t_i|c)$ is the probability (calculated in (b)) that the term occurs in a document of the given class c, using Laplace smoothing as in lecture.

2. **Rocchio classification**: (2 points)
   (a) Find the **centroids** of the three classes representing the authors. Reminder: in the Rocchio formula
   $$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d),$$
   $\vec{v}(d)$ is the **normalized** vector for document d, using the "default" $w_{d,t}$ values from Milestone 2 as the components of the vector, and normalizing by the Euclidian length L_d (also from Milestone 2).
   (b) Classify each of the unknown documents as being in the class corresponding with the closest centroid using Euclidian length.
3. **kNN classification**: (2 points)
   (a) Find the k nearest neighbors in vector-space to the unknown document. Select a strategy for breaking ties, using references from the text.
   (b) Experiment with different values of k and different tie-break strategies. Your final submission should enumerate those values and the effects they had on the accuracy of the system.

Precision-Recall Evaluation:

Either 2 or 4 points.

For this option you will evaluate Mean Average Precision and other metrics for your search engine after changing it in some way. You will choose **one** of the options below, implement it, then use a test collection data set that I will provide to evaluate the MAP of your new search engine. (See BeachBoard for the test collection(s).) Some of the options below will require a few more calculations, depending on the difficulty of the option. Regardless of the option, your ranked retrieval should return **50 documents** instead of 10 as previously.

Variant ranking formulas - 2 points: Evaluate the performance of the **Default**, **tf-idf**, **Okapi BM25**, and **Wacky** formulas, if you implemented them in Milestone 2. Compare each formula in the following ways:

- the MAP for the test collection data set
- the mean response time to satisfy a query
- the throughput (queries/second) of the system

Finally, plot a precision-recall curve for one query of your choice across the four ranking formulas.

Vocabulary elimination - 2 points: Implement vocab (index) elimination for inexact ranked queries as described in the text. Select a threshold for $w_{q,t}$ and skip terms in a ranked query if they do not exceed the threshold.

Compare your new search engine with your baseline search engine from Milestone 2 in the following ways:

- the MAP for the test collection data set
- the mean response time to satisfy a query
- the throughput (queries/second) of the system

You must experiment to determine an acceptable value for the threshold. Your submission should include each threshold that you tried, and the three statistics you measured for that threshold; then your final analysis and decision on which threshold to use.

Tiered index - 4 points: Implement a three-tier index as described in the text in section 7.2.1.

Perform a complete in-memory indexing using your `PositionalInvertedIndex` class as before, but now instead of creating a single on-disk index, create 3, and place documents into only one of the three based on some $tf_{t,d}$ threshold you choose with care. (Speak with me if you don't know where to start with this.)

For Boolean queries, you will need to pull postings from all 3 indexes prior to merging results. For Ranked queries, you will attempt to satisfy the query using only documents from the “top” index. If you score fewer than K documents in this way, then you need to start over and score documents using the top 2 indexes; likewise involve the third index only if this second attempt fails. For this requirement, let $K = 20$.

Compare your new search engine with your baseline search engine from Milestone 2 in the following ways:

- the MAP for the test collection data set
- the average number of nonzero accumulators used in the queries
- the mean response time to satisfy a query
- the throughput (queries/second) of the system
- the percentage of queries that were satisfied only using documents in the top tier

Finally, plot a precision-recall curve for one query of your choice, comparing the two search engines.

Impact ordering - 4 points: Implement an impact-ordered index, in which postings lists for terms are sorted in decreasing $\mathrm{tf}_{\mathrm{t,d}}$ order.

Rewrite your Boolean query processor to merge postings lists that are not sorted using an efficient process. (A hash set may be helpful.) For ranked queries, select an appropriate $\mathrm{tf}_{\mathrm{t,d}}$ threshold so that you stop reading postings (and thus ranking documents) when the contribution of a particular term (with a low $\mathrm{tf}_{\mathrm{t,d}}$) has very little effect on the document’s overall score.

Compare your new search engine with your baseline search engine from Milestone 2 in the following ways:

- the MAP for the test collection data set
- the average number of nonzero accumulators used in the queries
- the mean response time to satisfy a ranked query
- the mean response time to satisfy a Boolean query (running the queries from the test collection as Boolean queries)
- the throughput of both ranked and Boolean queries

Finally, plot a precision-recall curve for one query of your choice, comparing the two search engines.

Cluster pruning - 4 points: Implement a cluster-pruned index, in which you randomly select $\sqrt{N}$ leader documents at index-creation time, compute the nearest leader for every document in the corpus (saving that information to disk), and then satisfy queries by first selecting the highest-scoring leader and then ranking all of that leader’s followers.

Compare your new search engine with your baseline search engine from Milestone 2 in the following ways:

- the MAP for the test collection data set
- the average number of nonzero accumulators used in the queries
- the mean response time to satisfy a ranked query
- the throughput (queries/second) of the system

Finally, plot a precision-recall curve for one query of your choice, comparing the two search engines.

Paper - 4 points:

Turn in a 4-6 page paper regarding the final milestone. I will give more thorough requirements at a later date, but I will look for a high quality of work as expected of advanced students in computer science. You will give a background on the research topic you have chosen, discuss the methods needed to solve your topic, discuss your implementation, and present the final results.

Your own interest:

Speak to me if there is another information retrieval topic you would like to implement for this milestone. IDEAS OFF THE TOP OF MY HEAD: implement a caching system for query results; implement dynamic indexing with invalidation vectors and logarithmic merging; investigate and compare alternatives to Levenshtein distance in spelling correction SO MANY OPTIONS!