ORCREC: An Intelligent Course Recommendation System

J. Brofos, C. Orth, and C. Stimson

March 8, 2013

1 Post Milestone:

Since the milestone we have had to seriously reconsider the problem of recognizing course similarity on the basis of natural language descriptions in the ORC. In particular, we have revised our approach to stray from a pipeline of dimensionality reduction algorithms to a more simplified approach, as we were advised to do following the milestone presentation. To replace our previous approach, we now focus entirely on conceptions of distance within the TF-IDF feature space, and use varying metrics to capture course similarity.

The format of this document is roughly as follows:

- New learning algorithms and approaches
- Persistent faults and responses
- Results and discussion
- Future work and direction
- External software

2 New Learning Algorithms and Approaches

A major criticism in the milestone presentation was the failure to clearly describe the *motivation* behind the learning algorithms we employed. In this section, in addition to providing a brief overview of each algorithm's function, we have also attempted to justify its inclusion in the project.

Term-Frequency Inverse-Document-Frequency (TF-IDF): The course descriptions in the ORC consist of natural language that is not immediately representable as a numeric feature vector. To address this issue, we adopt the common approach of the TF-IDF [5], which transforms documents in a corpus to vectors on the basis of word presences. In particular, the TF-IDF vector is formulated as,

$$TF\left(t,d\right)=1+\log f\left(t,d\right),IDF\left(t,\mathbb{D}\right)=\frac{\left|\mathbb{D}\right|}{g\left(t,\mathbb{D}\right)+1} \tag{1}$$

$$TF\text{-}IDF=\log\left[1+TF\times IDF\right] \tag{2}$$

Where d is a document in the corpus, t is a term found within the corpus, $f(t,d)$ is the frequency of the term in the document, and $g(t,\mathbb{D})$ is the frequency of the term in the corpus.

We adopt this mechanism for constructing numerical representations of ORC descriptions over common word counts for the reason that the TF-IDF has a weighting mechanism that scales the relevance (magnitude) of particular features in the vector according to how rare or ubiquitous the word is in the corpus. Theoretically, this is advantageous for identifying high-relevance natural language identifiers in the ORC.

In the milestone we discussed that we had successfully completed parsing the ORC for natural language data, and that we had successfully generated TF-IDF vectors for the single word case. Since that time, we have successfully expanded this algorithm to also calculate TF-IDF numeric representations of classes for two-grams and three-grams (those being, word pairs and word triplets respectively). We implemented the additional calculations based on the assumption that they would describe second-order and third-order similarities between classes.

Manhattan Distance Metric: Given that the TF-IDF is an effective representation of courses within the course-space, it is necessary to further develop a method of quantitatively measuring similarity and dissimilarity. We say that two courses are similar if the distance measurement between them is close to zero, and courses are dissimilar if the distance measurement is relatively large. For the purpose of using distance as a metric, we have chosen the Manhattan distance, which is computed as,

$$\sum_{i=1}^{n} |x_i^{(j)} - x_i^{(k)}| \tag{3}$$

between vectors in the course space $x^{(j)}$ and $x^{(k)}$ [6]. We justify the adoption of the Manhattan distance metric over more common metrics (such as the Euclidean distance) on the basis that the course space is discrete in the number of points that are contained in it. Therefore, notions of Euclidean distance may not be appropriate in terms of defining similarities between objects.

Recommendation Centroids: The reason that we find it crucial to calculate distances and similarities between courses is because, ultimately, we want to discover courses that have possess a similar content base, as determined by the TF-IDF representation, to courses that an individual identified as being preferred. Let $\mathbb{C} = \{c_1, \ldots, c_n\}$ denote a set of courses specified by an individual at input and let $r_1, \ldots, r_n$ be course evaluations in the discrete interval $\{1, 2, 3, 4, 5\}$, where one is a heavily disliked course and five is a very well-liked course.

We assume that recommendations in the course-space can be formulated as a kind of weighted averaging of courses specified by an individual, where highly ranked courses serve as a point of attraction for the weighted average, and courses ranked lowly repel that average. In particular, we formulate our centroids as,

$$\bar{C} = \frac{1}{\sum_{r_i \in \mathbb{C}_r} r_i} \sum_{r_i, c_i \in \mathbb{C}} r_i c_i^* \tag{4}$$

Where c_i^* is equal to c_i if r_i is a relatively high value, and c_i^* is a class far away from c_i if r_i is a relatively low value. We assume that these centroids are an effective measure for capturing the an individual's preferred and desirable characteristics of courses because they are defined to be weighted averages of enjoyed classes which are repelled from unenjoyed classes.

Given this centroid, we can calculate a "closest class" (or multiple closest classes) which are defined to be,

$$\mathbb{R} = \min_i \sum_{\bar{C}_j} |d_{i,j} - \bar{C}_j| \tag{5}$$

Where d_i is the TF-IDF representation of the i^{th} course in the registrar's ORC. Notice that this is precisely the Manhattan distance between the centroid and all courses in the ORC, and the minimum course in terms of distance is the recommendation since it is most similar to the weighted average of an individual's preferred courses.

Within the actual centroid evaluating MATLAB code, we make a point to calculate, at each call, centroids for a subset of all courses specified by the user at input. We do not calculate a centroid using all courses specified by the user at once. We motivate this approach because it enables distance calculations to potentially identify a broader array of nearest courses, which would not be possible in the event that we evaluated centroids and obtained nearest courses using all available evaluation data.

Committee Voting Recommendation Scheme: A drawback of this approach is that the centroid may fall in a low-density region of coursespace, resulting in distant courses being returned. Our solution is to use a weighted committee voting scheme to account for this additional variable. Each course returned (a "vote") is paired with the inverse log of its distance to the respective centroid (a "confidence"); essentially, the confidence is a scaled prediction of the probability that the reccommendation is appropriate (inspired by the discussion in [2]).

The votes for each course are tallied by multiplying together the probabilistic inverses of each confidence score, and returning the probabilistic inverse of the result. For example, should course X recieve two votes with confidence $c_0 = 0.7$ and $c_1 = 0.6$, the final recommendation confidence would be,

$$\text{conf}(X; c_0, c_1) = 1 - (1 - 0.7)(1 - 0.6) = 1 - (0.3)(04) = 1 - 0.12 = 0.88 \tag{6}$$

Note that every vote, no matter how low its confidence rating, increases the final result monotonically (although not strictly); a zero confidence vote would result in an increase of 0, while a 1.0 confidence vote pushes the final result up to 1.0.

Formally, given the sets V of votes for a course X and $FP \subset V$ of false positives,

$$\text{conf}(v_i) = p(v_i \notin FP), v_i \in V, \tag{7}$$

and

$$\text{conf}(X) = p(FP \subsetneq V). \tag{8}$$

3 Persistent Faults and Responses

In the milestone, we received criticism for two missteps in particular. These were our failure to obtain a training data set with which we could quantify a reasonably accurate measure of the algorithm's success, and another was a lack of any kind of base-line, "naive" algorithm against which to compare our methodology.

We are dismayed to report that we were unable to obtain any type of official data set for this project. We attempted to contact the registrar to obtain data on student course selection choices with the intent to derive from that data an approximate pattern in course selection for every individual in the set. The registrar stated that such information could not be made available to us due to anonymity concerns. Failing this, we were aware that the Hacker's Club also possessed such data for use in their CoursePicker application web service. This additional effort failed however, as the Hacker's Club was also unable to provide our team with convenient data until the 7^{th} of March, by which time we could not effectively incorporate it into our project. Despite our inability to test effectiveness against existing data, the ORCREC system does permit individuals to input their own course evaluation data, and, from these, courses may be recommended and methods compared on a qualitative level.

In response to the second suggestion to implement additional base-line, we chose to implement a naive k-means classifier to identify similar courses according the the TF-IDF vectors using a Euclidean distance metric [4, 1]. Rather than evaluate the k-means model on the basis of numerous, seemingly-arbitrary k, there exists a very natural choice for the parameter, which is equal to the number of academic departments and programs at Dartmouth, which is forty. Thus, in this case $k = 40$.

In the milestone there was no well-defined performance measure of the system. Since we lacked any kind of actual data regarding student course election and registration, evaluating the efficacy by such a measure became correspondingly more difficult. However, we were able to reason about the following criterion that relies only on course evaluations provided by the user:

1. Given an input of courses and evaluations $\mathbb{C}$ construct centroids and assign recommendations according to some distance metric and some methodology.
2. Denote the list of generated recommendations as $\mathbb{R}$.
3. For every course in $\mathbb{C}$ that was given a high rating, obtain its general subject area or department name and count how often this field or department was repeated in the courses present in the recommendation output.
4. The ratio of the count of such courses and the total number of recommendations received constitutes a kind of accuracy rate for the system since it describes how frequently courses that were expected to be enjoyable appeared in the actual recommendations. Correspondingly, subtracting this accuracy from one gives the appropriate error rate.

4 Results and Discussion

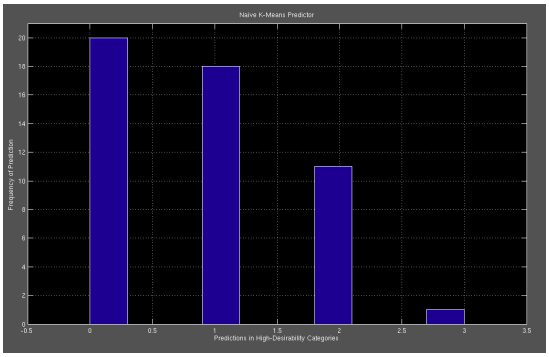
4.1 Sample Input and Baseline Model

As a baseline predictor, we used a k-means clustering algorithm on the course-space with the intention of clustering courses according to their department. To formulate a recommendation, the k-means algorithm finds courses that an individual identified as being highly preferable and obtains their cluster label. Then, a recommendation is returned by randomly sampling one other course in that cluster label and returning that course.

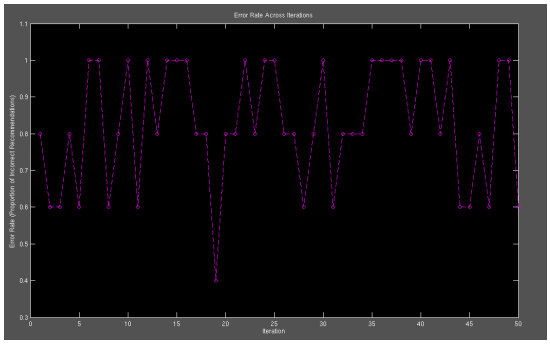
In particular, we take for example a student of biology and chemistry who offers as input into the system the following matrix of courses and evaluations. We begin with a particular example because it is most illustrative of how the system is supposed to operate when practically deployed, though we develop more general characteristics of the recommendation framework later on. We have,

$$\begin{bmatrix} BIO12 & BIO15 & CHEM57 & CHEM58 & CHEM67 & FILM40 & SART16 & SOCY10 & HIST30 \\ 4 & 5 & 4 & 5 & 4 & 2 & 1 & 3 & 2 \end{bmatrix}$$

Where the first row of the matrix index courses specified by the user and the second row specifies their impression of the corresponding course. Then by applying the k-means clustering and obtaining naive recommendations, we can obtain a histogram of the number of times that the recommendations returned by the k-means algorithm fell in the chemistry or biology departments (the most highly rated departments in the input) in each iteration.

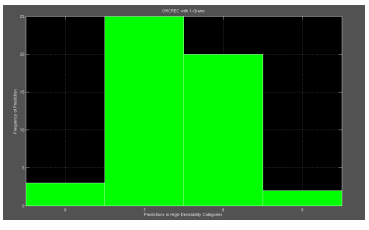


(a) Recommendations returned by the k-means recommendation framework. Notice that of a set of five recommendations returned at each iteration for fifty iterations, k-means returned no "correct" predictions most frequently and occasionally returned one or two meaningful recommendations. k-means was rarely capable of delivering three courses in biology or chemistry in a single recommendation.

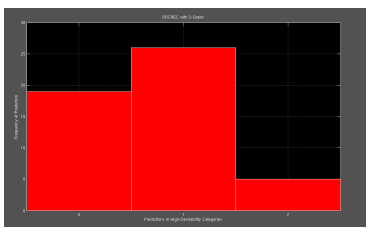


(b) The error rate corresponding to these predictions using the metric defined in the previous section. Notice that these errors are non-convergent as expected because this is simply the reported errors over a number of iterations. The mean error in this example was 0.8280, which suggests that, on the average, a course recommeded by the naive system will be undesirable 80% of the time.

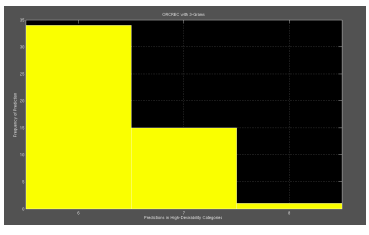
Figure 1: Frequency of desirable recommendations from naive k-means framework and corresponding error rate across iterations.



(a) 1-Gram Histogram for closest classes successes



(b) 2-Gram Histogram for closest classes successes



(c) 3-Gram Histogram for closest classes successes

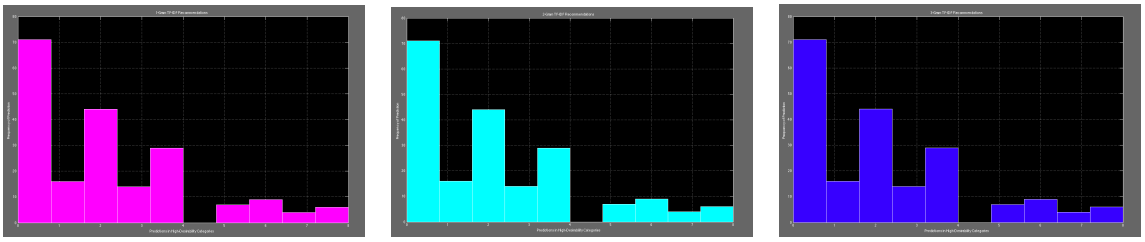
Figure 2: Histograms for the number of "correct" recommendations returned by the ORCREC methodology when queried for twenty-one recommendations on a sample piece of data of a student preferring chemistry and biology courses.

We also queried the MATLAB to find the closest classes for the sample biological sciences and chemistry input where we looked to obtain twenty-one closest classes. For the one-grams we can observe that the error rate, using the metric described earlier, is 0.6467; the error rate for the two-grams is 0.9657; the error rate for the three-grams is 0.6981.

There are several components of this analysis to note. First of all, one-grams and two-grams convincingly beat the naive k-means recommendation framework in terms of identifying classes that belong to departments or fields of high-desirability in the case of chemistry and biology, obtaining a ten to fifteen percentage point drop in error. Also interesting is that the two-grams failed so absolutely to identify courses similar to biology and chemistry.

4.2 Randomly Generated Data

An issue we encountered in using our approach revolved around the question, "What happens when a user enters essentially random inputs into the MATLAB distances system? What types of courses are recommended in that case?" To test this scenario, we generated two-hundred random instances of user data and queried the MATLAB algorithm to produce nearest classes for all TF-IDF n-gram representations.



(a) 1-Gram histogram for closest classes successes on random data

(b) 2-Gram histogram for closest classes successes on random data

(c) 3-Gram histogram for closest classes successes on random data

Figure 3: Histograms for the number of "correct" recommendations returned by the ORCREC methodology when queried for twenty-one recommendations. Notice that these histograms do not vary from one-grams to two-grams to three-grams.

From these "correctness" histograms, we are forced to acknowledge in particular that the distribution of meaningful closest classes is *identical* across all of the TF-IDF matrices. This suggests that one cannot derive significantly more explanatory power in terms of recommendation by employing additional n^{th}-order similarity measures. This was a disappointing result since it appears to be unintuitive and implying that our methodology for discovering similarity is ineffectual beyond the singleton term level for random inputs to the system. We also noticed that these results for random input demonstrated significantly lower accuracies than did the ORCREC distance calculations in the constructed, biology and chemistry student case. Because we believed that random data more closely reflects the actual user-specified input than does a single constructed example, we necessarily admit that ORCREC has widely varying prediction accuracies for measuring nearest course similarities on the basis of distances.

We further observed that in the case of random user input, the recommendation system consistently reported certain classes as being very near to the calculated preference centroids. In particular,

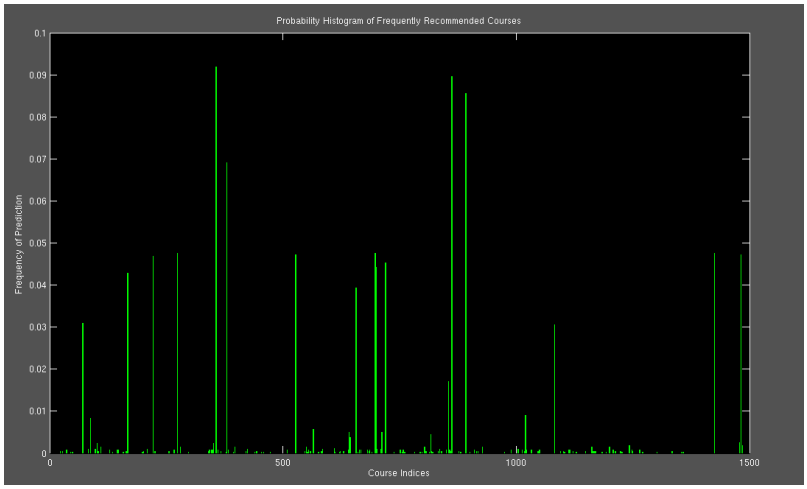


Figure 4: A bar graph that results when making one-hundred calls to the MATLAB function that finds the courses with the closest distances with respect to the one-grams, two-grams, and three-grams. At each iteration, we specify randomly generated user data consisting of five course selections and evaluations. Presented here are the proportion of times that the courses present in the recommendations were observed through all one-hundred iterations.

The consistent recommendation of these courses prompted us to arrive at the conclusion that these frequently-occuring courses were generic and not particularly close to any category of class, which is why they were recommended to the user specifying essentially random points in the space. Unfortunately, the courses that ORCREC recommends in this case are generic only in their natural language representation. Take for example Italian 89 (indexed as course 357 in the bar plot above), which is the department's Honors Seminar and appears in nearly one-tenth of all recommendations observed in the random case. The ORC description of this course is,

> "Honors students will arrange a program of study and research during any term of the senior year on a tutorial basis with individual faculty members. A thesis, written in Italian, and a public presentation are the normal culmination of this course. A proposal, signed by the faculty advisor, must be submitted to the Departmental Committee on Independent Studies and Honors Theses for approval by the fifth day of classes of the term [3]."

Which reads almost precisely like the seminar descriptions in seminar-style classes in numerous other departments. Yet clearly this is an advanced course in the department and most likely unfit for recommendation to the average user, and yet this recommendation is unavoidable because the course's ORC entry is not particularly descriptive of its contents. The issues of the ORCREC system with regard to random inputs are deeply concerning because we believe that random inputs may closely reflect the inputs of an individual who has taken courses across a broad spectrum of disciplines. In this entirely reasonable instance, ORCREC gives less than spectacular results.

4.3 Website Implementation

Bearing in mind that it was our ultimate goal to produce a web application, we find it necessary to include some discussion of the difficulties and trade-offs that had to be explored in the effort to realize that purpose. In particular, the website incorporates CGI scripting to distribute tasks to executable files that are written in MATLAB and Python, and the recommendation algorithm we developed suffers from several issues that make its incorporation into a website difficult to say the least. Namely,

1. Because MATLAB is required to terminate at each call (or else other programs would be unable to proceed in their duties because the MATLAB session is never-ending), MATLAB must be necessarily booted from scratch every time a recommendation is requested from the website. This takes approximately five to ten seconds.
2. Because MATLAB is terminated at the conclusion of each recommendation request, the large TF-IDF matrices are no longer stored in memory and must therefore be re-initialized. Loading from .csv files, this operation could take upwards of one-hundred-and-eighty seconds.

3. The two former factors contribute substantially to timeout errors observed on the server-side that prevent recommendations from being delivered to the user at termination. Recommendations using the ORCREC system are successfully produced, but are never presented to the user because MATLAB is unable to operate at a speed sufficient to handle the TF-IDF matrices and incorporate them into a recommendation framework.

These technical problems proved challenging for us to address. We felt as though the first issue regarding MATLAB initialization was essentially unavoidable given the construction of algorithms we had developed thus far.

We were, however, able to make substantial headway in reducing the amount of time required by MATLAB to reintroduce the TF-IDF matrices into its local workspace at each call. In particular, we realized that .csv files were time-intensive to load into the MATLAB workspace, and that this operation could be drastically simplified by saving each TF-IDF matrix into a *compressed* .mat representation and reloading those (rather than the .csv files). This was accomplished by using a data compression feature of MATLAB's `save` function. Performing this change allowed MATLAB to successfully load all TF-IDF matrices in approximately twenty-four seconds.

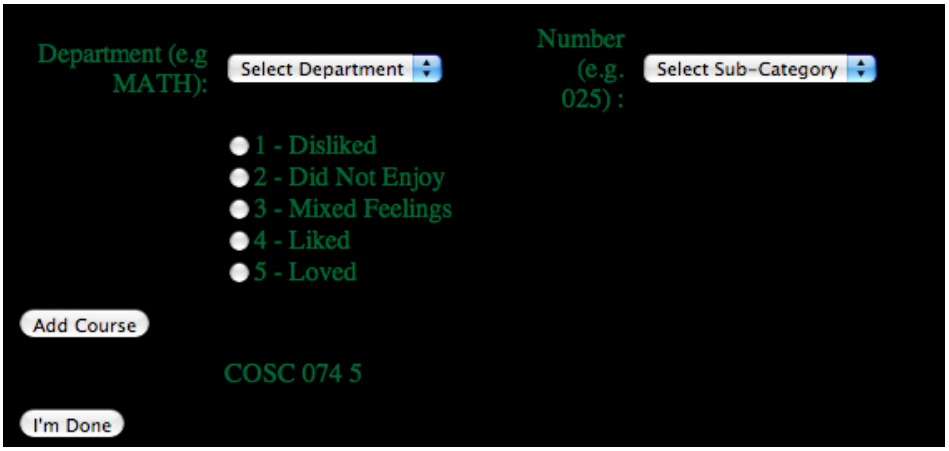


Figure 5: The simple splash page of the website developed to serve as a front-end to the recommendation system. Notice that there exists exactly the functionality to add courses to a list of courses taken, and to submit these courses for final analysis by the underlying algorithm. A version of the website can be found at: `http://cs.dartmouth.edu/~cameron/ORCREC/app`

5 Future Work and Direction

Presently, the largest obstacle to the development of the web application is found in the need to re-initialize the large TF-IDF matrices at every call to the MATLAB functions that identify the nearest classes. Though we managed to decrease the amount of time required by MATLAB to generate a list of nearest classes by centroid formulations, this speedup was not significant enough to prevent server timeout errors. Presently we believe that if MATLAB could be further enhanced then the web framework could become an effective recommendation system. This is the direction in which this project has most potential to develop as a substantive and interactive tool that could serve as a viable alternative to the Hacker Club's CoursePicker application.

The next step in dealing with the MATLAB issue would be to use some of the dimensionality reduction techniques we implemented early on and replacing MATLAB code with C calls to do matrix math with little to no load time. This will likely become necessary as Hacker Club has already expressed interest in incorporating our recommendation system into the existing CoursePicker, which will demand the quick performance. The biggest advantage of our system is its extensibility, so adding it to the HackTown site will allow every review and any other data they have gathered to enter into the calculations with only slight changes to the information passed to Committee.py. Ideally, the server would regularly, perhaps weekly, overwrite its one-gram, two-gram and three-gram matrices to reflect any information gained.

Currently, we store each user's information in files based around their IP address, which is fine for testing but obviously terrible for large-scale use by Dartmouth students coming from the same handful

of addresses. Including web authentication and accessing the server variables like UID would be an easy fix that we will implement once the timeout issue has been solved. On that front, Tim Tregubov has recently informed us that the MATLAB processes were staying open and doing little work on the web server, so we will look into making remote calls to MATLAB on different lab computers to alleviate the stress on the web server.

6 External Software

pyPDF: `http://pybrary.net/pyPdf/`. Although the original python file to extract text from the .pdf version of the ORC has been lost, it was a straight-forward application of the pyPDF package.

Fast K-Means: `http://www.mathworks.com/matlabcentral/fileexchange/31274-fast-k-means`. Tim Benham provides a fast version of k-means that performs batch clustering. The code implemented in ORCREC draws on many of the efficiencies of Benham's implementation but strips out some irrelevant features and performs calculations of distances and label assignment slightly differently.

Fast Euclidean Distance: `http://mi.eng.cam.ac.uk/~at315/MVRVM.htm`. The MATLAB implementation of Euclidean distance is inspired by Tipping's RVM implementation of distance squared. Tipping inspired the idea of vectorized distance calculations, but actual implementation of the idea in ORCREC looks quite different.

Django Frameworks: `https://www.djangoproject.com/`. We use some utilities from the Django framework library to convert default Python strings into Unicode strings.

References

[1] Benham, Tim. Fast K-means. Computer software. MATLAB File Exchange. The MathWorks, 02 May 2011. Web. 7 Mar. 2013.

[2] Bishop, Christopher M. "Chapter 14: Combining Models." Pattern Recognition and Machine Learning. New York: Springer, 2006. 653-74. Print.

[3] "ITAL 89 Honors Seminar." Organization, Regulations, and Courses 2012. Dartmouth College, 2012. Web. 8 Mar. 2013. `http://dartmouth.smartcatalogiq.com/2012/orc/Course-Descriptions-Undergraduate/ITAL-Italian/ITAL-89`.

[4] "k-means++: The Advantages of Careful Seeding", by David Arthur and Sergei Vassilvitskii, SODA 2007.

[5] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schutze. "Document and Query Weighting Schemes." Introduction to Information Retrieval. New York: Cambridge UP, 2008. 118. Print.

[6] Paul E. Black, "Manhattan distance", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. 31 May 2006. (accessed March 7, 2012) Available from: `http://www.nist.gov/dads/HTML/manhattanDistance.html`

[7] "The Web Framework for Perfectionists with Deadlines — Django." The Web Framework for Perfectionists with Deadlines — Django. N.p., n.d. Web. 08 Mar. 2013. `https://www.djangoproject.com/`.