8-15-2013

Development of a Jobs Database for Tracking Knowledge and Skills Expectations in the Workplace

Esa M. Rantanen
Rochester Institute of Technology

Christopher Claeys

Daniel Roder

R·I·T

Rochester Institute of Technology

RIT/PSY/TR–13/2
Final Technical Report

Development of a Jobs Database for Tracking Knowledge and Skills Expectations in the Workplace

Esa M. Rantanen
Christopher Claeys
Daniel Roder

Prepared for
College of Liberal Arts and
the Office of Cooperative Education and Career Services at
Rochester Institute of Technology, and
Human Factors and Ergonomics Society, Santa Monica, CA

August 15, 2013

Department of Psychology
College of Liberal Arts
18 Lomb Memorial Drive
Rochester, New York 14623-2953

Technical Report Documentation Page

1. Report Title **Development of a Jobs Database for Tracking Knowledge and Skills Expectations in the Workplace**	**2. Report No.** RIT/PSY/TR-13/2
	3. Report Date August 15, 2013
4. Authors Esa M. Rantanen, Christopher Claeys, and Daniel Roder	**5. Type of Report** Final Technical Report
6. Performing Organization Name and Address Rochester Institute of Technology, Department of Psychology, College of Liberal Arts, 18 Lomb Memorial Drive Rochester, New York 14623-2953	**7. Performing Agency Code** NA
8. Sponsoring Organization Name and Address College of Liberal Arts and the Office of Cooperative Education and Career Services at Rochester Institute of Technology, and Human Factors and Ergonomics Society, Santa Monica, CA.	**9. Sponsoring Agency Code** NA

10. Supplementary Notes

The authors wish to thank the College of Liberal Arts and the Office of Cooperative Education and Career Services at Rochester Institute of Technology and Human Factors and Ergonomics Society, Santa Monica, CA, for their generous support of this work. We would also like to thank Production Services at the Wallace Center at RIT and Raman Bhalla for programming the database and hosting it on a server at RIT. Many thanks are also due to the many volunteers who had gathered data from human factors jobs announcements over the past several years that could be read into the new database. We would like to thank Nathan Sugarman for entering much current data into the database.

11. Abstract

The primary objective of college education in applied disciplines is that it is relevant to the expectations for new professionals entering the labor market. Academic institutions should therefore pay close attention to the ever-changing skills and knowledge expectations in the labor market. These trends are not easy to track, however. Surveys of new professionals about their experiences in their first jobs or surveys of employers about their experiences with new hires suffer from low response rates, nonresponse bias, and the one-time nature of survey research. A better way to track labor market trends is to continually analyze human factors job postings for education and experience requirements specified in them. This report describes development of a database for that purpose. We also discuss ways of analyzing unstructured text data in the database. The results of analyses of these data include summary statistics of frequencies and their correlations, clusters of similar jobs, and a continually updated mathematical model to classify jobs in the database. These results may be subjected to longitudinal analyses when the database contains sufficient data. If desired by other departments, this style of database and functionality could be repurposed for other domains or fields of employment. Anyone interested in applying this database for domains other than HF/E should contact Production Services at the Wallace Center at RIT.

Contents

Background

The primary objective of college education in applied disciplines is that it is relevant to the expectations for new professionals entering the labor market. This project was premised on the notion that the employers of new professionals are in the best position to experience and anticipate changes in the skills and knowledge demands of their workforce, and are therefore the best people to advise academic curriculum development to keep higher education relevant to their needs. The key question is then what are the best conduits between employers of new professionals and the professors of students in colleges and universities.

Human Factors/Ergonomics Education

These questions have recently become particularly critical in the field of human factors and ergonomics (HF/E), where there is a serious concern about the "pipeline" of new HF/E professionals for the needs of both industry and academia. Education and training of the future HF/E workforce has been of interest to the Human Factors and Ergonomics Society (HFES) throughout its history. The HFES currently has several active committees (the Education and Training Committee and the Early Career Committee) and task forces (the Workforce Issues Task Force) working on evaluating the effectiveness of HF/E education and identifying areas for improvement. Anticipation and preparation of members for the future requirements and issues of human-centered design is also a stated strategic objective of HFES. Growing the ranks of HF/E professionals to promote human factors in all areas of life is arguably the primary purpose of the society, and the means to that goal are through educational programs in HF/E.

Past Research

There have been periodic inquiries into the education and training needs of HF/E professionals (Cooke & Gorman, 2004; Stone & Derby, 2009). Recently, Rantanen and Moroney conducted two surveys, one administered to new HF/E professionals, or individuals who had been working in the field for less than five years (Rantanen & Moroney, 2011) and another administered to the employers of new HF/E professionals (Rantanen & Moroney, 2012). In a nutshell, both surveys sought answers to the question "What do you wish you (or that your recent hires) had learned in college given what you know about the expectations (or expect from your new hires) in the workplace".

The results of both surveys were remarkably congruent. Both new professionals and their managers were unified about several particular areas they felt theyÑor their recent hiresÑwere insufficiently prepared for given the demands of their jobs. These areas included design experiences, exposure to the processes used in the "hard" engineering disciplines, and experience in communicating as members of interdisciplinary teams and making persuasive arguments for human factors in all project phases. The most common academic areas that the respondents wished had been addressed in greater depth in college were research methods and statistics, application of knowledge learned, and various aspects of design.

The survey method has several critical shortcomings that make it a less than optimal tool for the purpose of tracking changing trends in education and skills needs of HF/E professionals in the workplace. The response rates of surveys tend to be very low, meaning that the data they yield may not be representative of the field. Furthermore, surveys only provide snapshots at a given time and need to be repeated frequently for a more continuous flow of information. Frequent surveys, however, may be impractical because of the time and effort they require to develop, administer, and analyze, and they would only exacerbate the problem of low response rates. Better methods are therefore needed to bridge the communications gap between the workplace and educational institutions.

Another way to track changing trends in education and skills needs in the workplace is to analyze job postings for HF/E professionals. It is assumed that the stated educational and experience requirements as well as job descriptions accurately reflect the demands of HF/E jobs, and that these data should therefore be taken into account in development and revision of HF/E curricula and programs in the academia. Analysis of job advertisements has a long tradition within the HFES. Moroney and his collaborators have reported data on placement opportunities for HF/E professionals annually for over 10 years (1996-2007) in the proceedings of the HFES annual meetings. These data describe the hiring expectations during the aforementioned time period. For a brief meta-analysis of the results from these studies, see Rantanen, Claeys, Roder, and Moroney (2013). The reports by Moroney and collaborators have been one-time efforts and little additional value could be gained by reanalyzing the data. Furthermore, the data in these reports does not easily lend themselves for longitudinal research on trends in HF/E labor market.

About This Report and Project Description

To provide a long-term solution to continual tracking of changing education and skills expectations for new HF/E professionals entering the workforce, an online database was developed for continual archiving of HF/E jobs data and to facilitate their analyses as necessary or desired. This report describes the problem, the development of the database, the use of the database for both viewing data within it and entering new data to it, and some initial methods of analysis of the data.

This project was undertaken between December 2012 and August 2013. Funding for the work was provided by grants from the College of LIberal Arts Faculty Research Fund and the Office of Cooperative Education and Career Services at Rochester Institute of Technology (RIT), and the HFES. Specifications for the database were developed by Esa Rantanen with Raman Bhalla and and Daniel Roder at the Production Services at the Wallace Center at RIT. Chris Claeys developed the algorithms and R code for analyses of the data. Data collected into an Google spreadsheet by several HFES member volunteers over the past several years were reformatted and read into the database. Nathan Sugarman entered new data into the database between March and July of 2013.

A beta version of the database was put online in the end of February 2013 and the final database was hosted on an RIT server in May 2013. Volunteers will continue to enter data into the database for the foreseeable future. Data are also freely available for analyses by any interested individuals.

The remainder of this report will provide detailed descriptions of the database development, use of the database, and analyses of the data in the database. Copies of the database are are available from the the Production Services at the Wallace Center at RIT to be used in other that HF/E domains.

Database Development

The human factors jobs database development was undertaken by the Production Services at the Wallace Center at RIT. The goal was to create a usable interface for entering, storing, and searching a collection of HF/E job postings. This database is searchable in a Google-type fashion yielding relevant results to the entered search terms.

The database was created in MySQL, a relational database management system. MySQL is an easily used database system that allows for the storage of large datasets. For this database, only one table was necessary to store all of the fields that were required. There were a few fields added for use in programming the front end for the database in addition to those originally specified. These fields were a JobId and AddedTimestamp. The JobId is a unique identifier that gets assigned to each row in the table. This is necessary for maintaining uniqueness for every entry. For example, if there were two jobs with the same title, this ID uniquely separates the two entries that otherwise could not be differentiated. The AddedTimestamp is

simply a timestamp that indicates when the Job was entered into the database. These fields have no bearing on the job information. The other fields are as follows: Date of Job Posting, Organization/Employer Name, Position Title, Location, Job Type, Required Education, Required Fields (of education), Required Experience, Preferred Experience, Job Description, Salary, Benefits, and Disciplinary Area. All of these fields are free form fields to allow flexibility of data entry but they are also all fully searchable.

The front end of the site is simple and usable. On the home page the users are presented with a Google style search box that effectively searches through the database and displays the results on a page in readable form. The search may seem simple but behind the scenes is fairly complex. The functionality to search by two terms is available to any user by simply typing two words into the search box e.g. Òmasters developer.Ó This would search for any job posting containing the words ÒmastersÓ OR ÒdeveloperÓ anywhere in any of the fields yielding more results. Once a search has been entered and a user is viewing the results, there is an option to download the results into a CSV (Comma Separated Value spreadsheet) with only the results from the search. Additionally, on the home page, all jobs in the database can be downloaded into a single spreadsheet.

The add a job page is a simple web form that can be filled out in any format with any number of characters for each of the previously listed fields. The first field, Date of Posting presents a date picker that allows for more easily added dates so that they are stored in a uniform fashion. The rest of the fields are free form text fields allowing for flexibility of job entry. Once the form is filled out and submitted, a processing script filters out any special characters that are not handled well by the database and inserts the data into the correct rows in the database. From there they are immediately searchable through the site search. The add a job form is a password protected file so that malicious entries cannot be made. This is handled by an htaccess file. This file restricts access to a page or set of pages to authorized users.

The Human Factors Jobs Database is the first of its kind produced by the RIT Production Services but will potentially yield successful research in the field of human factors. If desired by other departments, this style of database and functionality could be repurposed for other domains or fields of employment. Anyone interested in a copy of the database for domains other than HF/E should contact Production Services at the Wallace Center at RIT.

Database Use

Public Interface and Database Search

The database has two distinct interfaces. One interface is open to the public, that is, to anyone on the Internet (Figure 1). This interface consists of three pages: “Search”, “About”, and “List of Jobs”. The database may be searched by entering a keyword in the search box and clicking on the search button. The database will return all jobs that meet the search criterion, and these may be downloaded as a csv file for further analysis. If the search box is left empty, the entire database may be downloaded for analyses.

The “About” page contains links to basic documentation for the database, including this technical report, as well as the R code for initial data analyses. We hope that users of the database will send copies of their analyses and results to be linked from this page, too, for others to use. Finally, the “List of Jobs” page simply lists all the database contents. This page maybe sorted from oldest to newest entries or vice versa.

Data Entry Interface

The data entry interface is available only to those who have committed to maintaining the database and adding data to it. A username and password are required to access it. This interface has three pages:

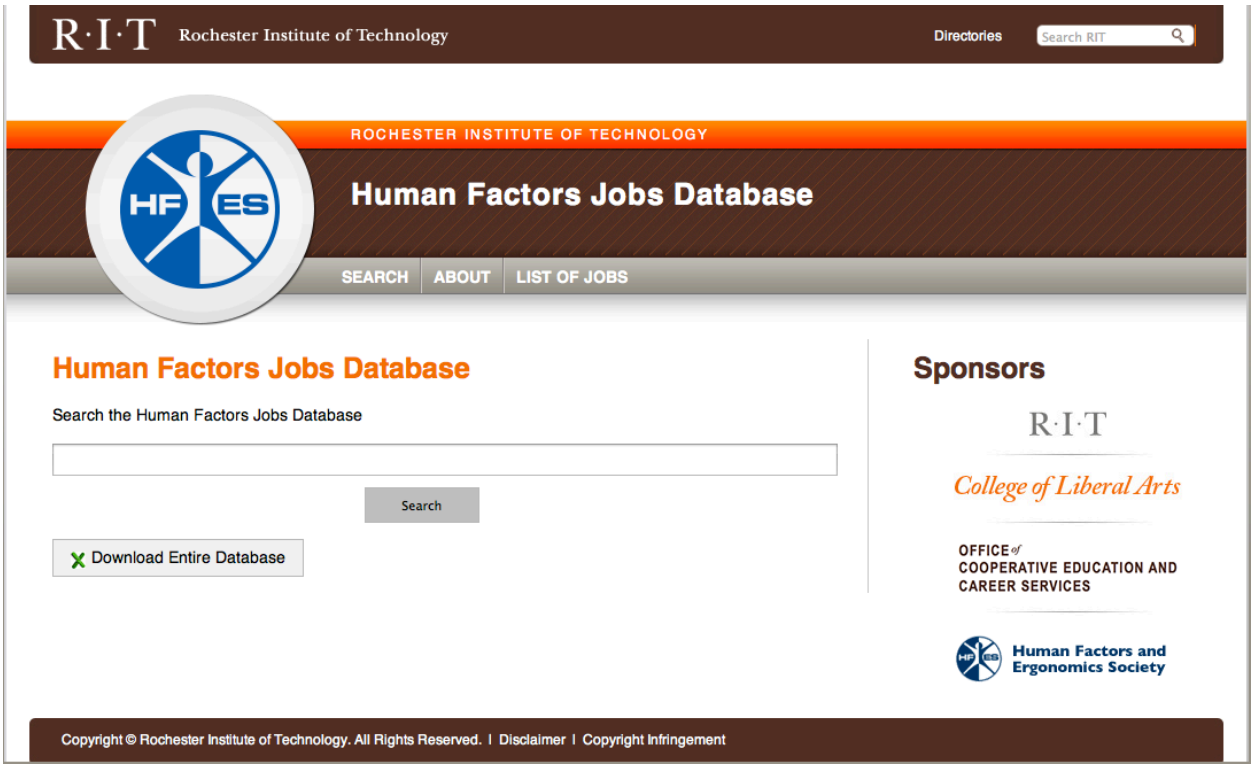

Figure 1. Screenshot of the "public" homepage of the HF jobs database. The database may be searched by entering a keyword in the search box and clicking on the search button. The database will return all jobs that meet the search criterion, and these may be downloaded as a csv file for further analysis. If the search box is left empty, the entire database may be downloaded for analyses.

"Search", "Add a Job", "Edit List of Jobs". The search function is as in the public interface. The "Add a Job" page has a form for data entry (Figure 2).

The data entry page of the HF jobs database has several freeform data entry fields. Relevant data may be just typed in or copied from job ads and pasted in the respective fields. However, it is important to edit the copied entries to make them short and succinct and use no other punctuation than commas to separate multiple experience requirements or different aspects of the job description, for example. As employers conform to no standard in composing their job announcements, the person entering the data into the database should attempt to capture only the essential information about each job and consider the later analyses of the data while doing so. The "Edit List of Jobs" page allows for deleting duplicate entries, correction or errors that may have occurred in data entry, and adding missing information to existing records.

Data Analysis

General Overview

Given that the Human Factors Job Database will have the potential to store a limitless amount of job posting data, the analysis task is one that should be concerned with data mining and machine learning (DMML) techniques. DMML is the search for structure in large, high-dimensional data sets where dependency structures within the data are often commonplace and the data themselves are unstructured and varying (Clarke,

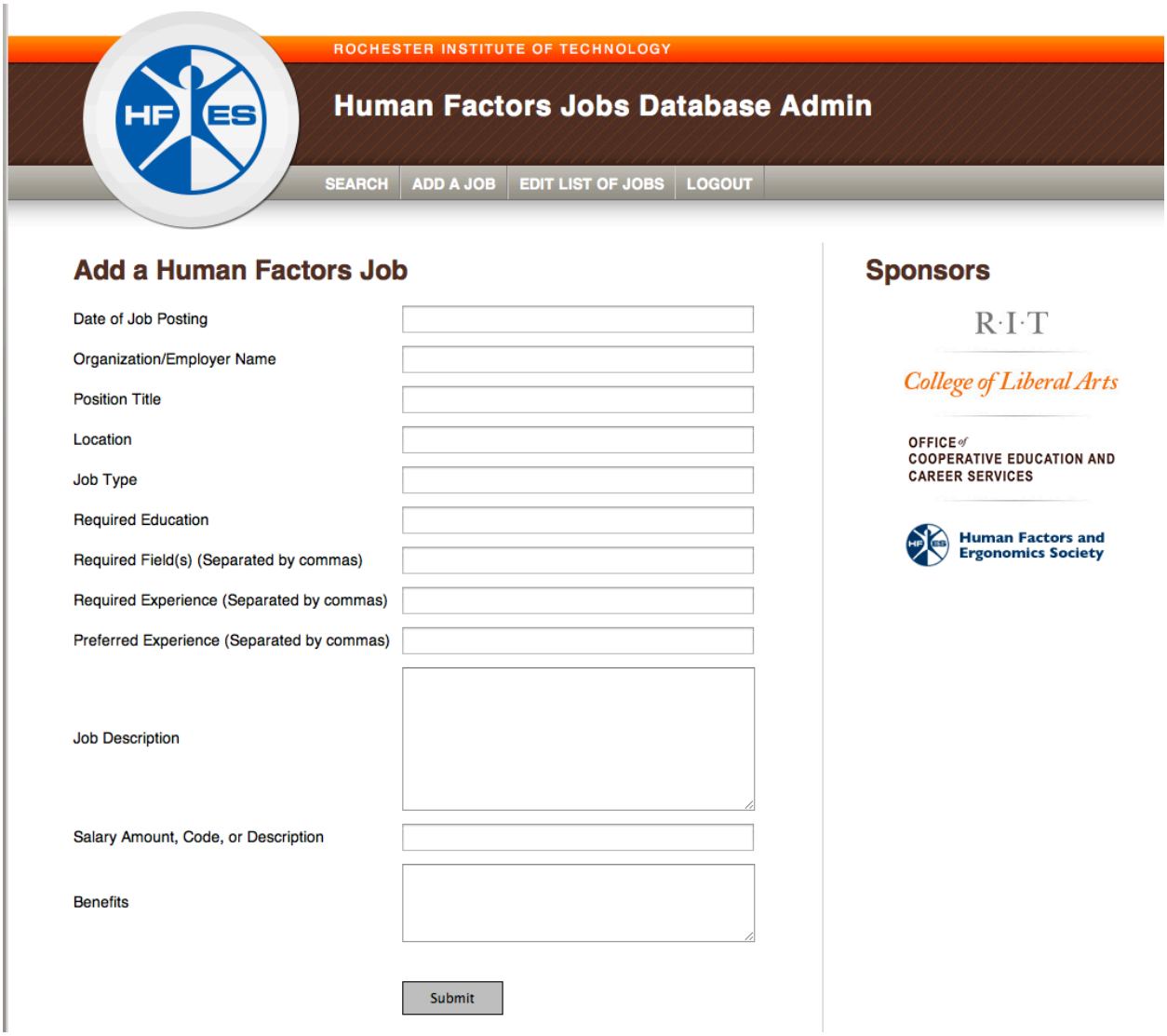

Figure 2. Screenshot of the data entry page of the HF jobs database. All fields are freeform and relevant data may be copied from job ads and pasted in the respective fields. However, it is important to edit the entries to make them short and succinct and use no other punctuation than commas to separate multiple experience requirements or different aspects of the job description, for example, as this helps in later analysis of the data.

Fokoué, Zhang, 2009). Given that our data are formatted as text, the incumbent analysis differs slightly from traditional DMML techniques. However, classical methods in so-called text mining do come from the data mining community (Feinerer, Hornik, Meyer, 2008; Weiss, Indurkhya, Zhang, Damerau, 2004). These methods include document clustering (Boley et al., 1999; Zhao Karypis, 2005) and document classification (Sebastiani, 2002). There is one main difference between data mining and text mining: data mining deals with data in numerical format, whereas text-mining deals with data in text format. Therefore, for DMML methods to be useful we need to find a convenient numerical, spreadsheet-like representation of our text data.

If we consider traditional data structures with rows as subjects and columns as variables, or information about those subjects, then we can see the first challenge of text data: what constitutes a subject and what constitutes the information about that subject? With our data, in particular, we can consider the information about each single job posting (employer, required education, state, city, job description, etc.), collectively, as a document and then each document as a subject. The information about each document is then given from a transformation of the pure text to a strict frequency measurement of the number of times a distinct word appears in each document (Weiss, et al., 2004). In this manner we can see how our text data is represented in a more traditional structure, that this structure contains a much larger set of variables (each distinct word is a single variable), and that our analysis problem subsequently requires the use of DMML techniques due to the much higher dimensionality.

Text Mining Objects

As discussed above, for proper analysis of text data we need three main objects: a text document on a particular topic, a collection of such documents, and a numerical representation of the data in that collection. This leads us to the terms:

- **Document** - A singular, unique text for a given topic. In our application: a human factors job posting.
- **Corpus** - A collection of documents.
- **Term-Document Matrix (TDM)** - A matrix of distinct word counts for each document in the Corpus.

Text Mining Procedures

Before we create a Term-Document Matrix from a Corpus several steps must be performed ahead of time to ensure useful information retrieval. These steps all revolve around formatting the raw text of each document. The steps are:

- Remove white space.
- Covert all text to lower case so frequency count isn't case dependent.
- Remove stop-words, or words so common their information value is generally zero (words such as "for", "the", "is", etc.).
- Stemming, or the process of erasing word suffixes to retrieve their radicals and thereby achieve conflation of term variants (i.e. "computing" and "compute" are both stemmed to "comput").

It should be noted that there are two types of stemming inflectional and root. Inflectional stemming is when the stemming procedure is concerned with the regularization of grammatical variants such as singular/plural and past/present while root stemming is concerned with reaching a root form with no inflectional or derivational suffixes. Root stemming is the more severe of the two procedures, however the drastic reduction in the number of distinct words in the text collection makes the distributional statistics more reliable (Weiss, et al., 2004). For this reason, we have chosen to use root stemming as opposed to inflectional stemming for our application. Further, the root stemming algorithm that we will be using is the well-known Porter stemming algorithm (Porter, 1980).

Methods

There are three main analysis methods that we consider:

1. Simple frequency counts of distinct words in the Corpus.
2. Clustering of documents.
3. Document classification.

Frequency counts are generated through construction of the TDM and can give a macro-level view of the documents. Further, if we partition the Corpus by a certain criterion (i.e. Year) ahead of time and generate separate TDM's we can analyze trends in job posting language, and thus the marketplace, as they pertain to this criterion.

Clustering will work on the entire Corpus and provide clusters of documents (job postings) based on latent similarities between the documents. Documents within the same cluster would therefore possess similarities of a non-variable dependent nature (that is, there is no response parameter that they are being measured on). Summarization based on a Term-Document matrix (as described above) for each cluster can provide information useful for student advisement, resume development, and smarter job searching purposes. See subsequent sections for a more in depth discussion on the particular clustering method that we employ.

Document classification presupposes that our documents will be labeled ahead of time based on a classification criterion. This criterion is considered as our dependent variable and will be the variable for which we will build a model classifier. Building a classifier for a given variable will allow employers to analyze their document and ascertain whether or not it reads similar to other documents of its kind and thus whether or not it will reach its intended audience. Further we can assess the accuracy of this classifier ahead of time by iteratively splitting our data into training and test sets, building the classifier on the training data, and then testing the accuracy with the test data (since we already know what the document should be classified as). At the end of the iterative cycle we would average the accuracy measurements from the independent trials and obtain an expected classification rate. Therefore, we could say that we would expect the model to correctly classify a new document x% of the time. Subsequent sections give more information as to the particular classification method that we will be using.

Kernel Methods

As discussed in section 1, text mining procedures are generally concerned with data mining and machine learning techniques. Further, due to the large dimensionality of the data sets we'll be studying, kernel-based learning methods will be advantageous to use for both their computational efficiency and information retrieval capabilities. Kernels operate by using an implicit mapping of the input data into a higher dimension feature space as defined by a function returning the inner product $\langle \Phi(x), \Phi(y) \rangle$ (the kernel function) between data points x and y. If the inner product based on Φ converts to a function of the two inputs then learning can take place in the feature space and explicit use of Φ is not needed; this is often referred to as the "kernel trick" and is more explicitly defined as: given a projection of $\Phi : X \to H$, the inner product $\langle \Phi(x), \Phi(y) \rangle$ can be represented by a kernel function, k

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle \tag{1}$$

which is computationally simpler than explicitly projecting x and y, through Φ, into feature space H. Thus, once we have a valid kernel function we can effectively work in spaces of any dimensionality without ever needing the explicit feature mapping. This is an important aspect of kernel based methods as they allow

us to work with high-dimensional data sets without any loss of information Clarke, Fokoué, Zhang, 2009; Schölkopf Smola, 2002).

The Spectrum String Kernel

Selecting the appropriate kernel function is an important step in any data mining application as the kernel function inherently defines the feature information extracted from the data. String kernels, over the alternative vector representation of text, have been shown to be particularly effective in kernel support vector machine text classification and kernel based clustering methods (Karatzoglou Feinerer, 2006). The basic idea behind string kernels is to compute the inner product between two text strings (x, x') by counting the occurrence of common substrings (s) within each full string . Thus, the generic string kernel:

$$k(x, x') = \sum_{s \sqsubseteq x, s' \sqsubseteq x'} \lambda_s \delta_{s,s'} = \sum_{s \in A^+} num_s(x) num_s(x') \lambda_s \tag{2}$$

where A^+ represents the set of all non-empty strings, and λ_s is a weighting factor. Previous research [0, 0] on documents with sentence-based structures has shown the Spectrum string kernel [0] performs best in classification and clustering tasks. This string kernel alters the generic form by considering substrings of length n exactly. For example, consider two words: `computer` and `computing`; computing the spectrum kernel with $n = 4$ and $\lambda = 1$ gives three matches (`comp, ompu, mput`) and thus a kernel value of 3. This is the kernel that we'll use for both our classification and clustering tasks.

Spectral Clustering

The following is referenced from Shi and Malik (2000) except where otherwise noted. Considering a set of points in an arbitrary feature space and its undirected, weighted graph $G = (V, E)$ where the points in the feature space are the nodes of the graph and the edges between the nodes are weighted as a function of the similarity between the two nodes $(w(i, j))$. We then seek a partitioning into disjoint sets where the similarity within sets is high and the similarity between sets is low. This partition can be constructed by removing all edges connecting nodes in the disjoint sets and the degree of dissimilarity is then the sum of the weight of edges removed. This adjacency matrix is called the cut:

$$cut(A, B) = \sum_{i \in A, j \in B} w(i, j), \tag{3}$$

and the optimal partitioning is the one that minimizes the cut. However, this cut algorithm can be impractical as it inherently favors cutting into small sets of isolated nodes given that a minimum is best achieved when there are fewer connecting edges. Thus a method for normalizing the cut and avoiding this bias is computed by looking at the cost of the cut as a fraction of the total edge connections to all nodes in the graph and is given as:

$$NCut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{4}$$

where $assoc(A, V)$ is the total connections from nodes in set A to all nodes in the graph ($assoc(B, V)$ is similarly defined). Since this problem is NP-hard (insolvable in a realistic time frame) the $NCut$ algorithm was introduced as a continuous approximation for solving the problem. This algorithm was then further refined by Ng, Jordan, and Weiss (2001) as:

1. Create an affinity matrix from the data as $K_{ij} = exp(-\sigma(\|x_i - x_j\|)^2)$.

2. Normalize the affinity matrix as $L = D^{-1/2}KD^{-1/2}$, where $D_{ii} = \sum_{j=1}^{m} K_{ij}$.
3. Take the top k (number of clusters) eigenvectors from L and form X as an $n \times k$ matrix.
4. Form matrix Y by renormalizing X to unit length.
5. Use kmeans clustering on the rows of Y.

This is the algorithm implemented for spectral clustering. Since the data are embedded into the eigenvector subspace the resultant clusters are generally tighter than they would be from the typical $kmeans$ clustering algorithm. Karatzoglou and Feinerer (2006, 2010) show this to be true in text mining applications. In our application of this algorithm there is one difference: instead of creating an affinity matrix using the formulation in step 1, we will create one using the formulation for the Spectrum string kernel (the formulation in step 1 is for a Gaussian radial basis kernel).

Support Vector Machines

A Support Vector Machine (SVM) (Vapnik, 1998) is a type of supervised learning algorithm that seeks to define a separating hyperplane between disjoint groups of data. SVMs have exhibited excellent generalization performance in practice and have a strong theoretical foundation in statistical learning theory. Further, SVMs can be extended to use kernels making them a natural choice for modeling high dimensional data sets. We'll start with a discussion of SVMs for a linear separable case first, then move into a discussion of how SVMs change for non-linear problems, and finally show how we can introduce kernel methods. The material in this section is referenced from Clarke, Fokoué, and Zhang (2009) and is intended to provide a brief overview of SVMs and how we use kernel methods with them. For a full discussion and formulations of all of the involved mathematics please see the primary text.

Given a data set with input vectors $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i = 1...m$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ and $\boldsymbol{y}_i \in \{\pm 1\}$ we can define a linear function that separates the two classes of $\boldsymbol{y}$ as:

$$h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b, \tag{5}$$

where $\boldsymbol{w}$ is a vector of coefficients and and b is a constant. The SVM classifying function is then the $h(x)$ that achieves:

$$min_{\boldsymbol{w},b} = \frac{1}{2}\|\boldsymbol{w}\|^2, \tag{6}$$

subject to:

$$y_i h(\boldsymbol{x}_i) \geq 1, i = 1, ..., n. \tag{7}$$

This formulation of the SVM classification is a nonlinear constrained convex optimization in "primal" space. While this formulation can be solved, the primal problem is often transformed into an unconstrained optimization by way of Lagrange multipliers resulting in the so called "dual" problem. This formulation is preferred as it's easier to manipulate and gives more intuitive results. The final SVM linear classifier is then:

$$f(\boldsymbol{x}) = sign(\sum_{j=1}^{n} \hat{\alpha}_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + \hat{b}) \tag{8}$$

This type of SVM classifier is the most basic and often referred to as the hard, or maximal, margin classifier as it finds the hyperplane that correctly classifies the data and maximizes the distance between the nearest support vector training points. However, in most real world problems, there is no such solution because the data are not linearly separable and the "hard" boundaries will invariably misclassify some of the

data; so, we need a method for dealing with non-linear data. One possible way to circumvent this problem is to "soften" the margins by introducing a "slack" and penalization variable into the objective function. The primal optimization problem is then to find a function:

$$h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b, \xi \tag{9}$$

that achieves:

$$min_{\boldsymbol{w},\xi} = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \tag{10}$$

subject to:

$$y_i h(\boldsymbol{x}_i) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, ..., n. \tag{11}$$

ξ are the errors, or misclassifications, and C is the penalization term, a constant to be specified. Thus, small values of C penalize the complexity of the model at the cost of misclassifications, while large values of C penalize the errors at the cost of better model generalization. As it turns out, after solving the dual optimization problem, the final formulation is the same as before:

$$f(\boldsymbol{x}) = sign(\sum_{j=1}^{n}\hat{\alpha}_i y_i \boldsymbol{x}_i^T\boldsymbol{x} + \hat{b}), \tag{12}$$

A second approach to non-linear problems with SVMs is to linearize the data by using a kernel. In equation 10 we see the familiar Euclidean inner product term ($\boldsymbol{x}_i^T\boldsymbol{x}$) and an intuitive way to alter the formula for a kernel method:

$$f(\boldsymbol{x}) = sign(\sum_{j=1}^{n}\alpha_i y_i \boldsymbol{\Phi}(\boldsymbol{x}_i)^T\boldsymbol{\Phi}(\boldsymbol{x}) + b), \tag{13}$$

And we have the kernel trick from the earlier section that maps the inputs to a higher dimensional feature space:

$$f(\boldsymbol{x}) = sign(\sum_{j=1}^{n}\alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b), \tag{14}$$

Support vector machines with a Spectrum string kernel will be used in our classification procedures.

R Infrastructure

All calculations are performed in R (R Core Team, 2013). Additionally, we make use of the following, non-base, packages:

`ggplot2`	Used for creation of frequency plots (Wickham, 2009).
`gridExtra`	Used for grid arrangement of multiple ggplots (Baptiste, 2012).
`kernlab`	Provides kernel, support vector machine, and spectral clustering methods Karatzoglou, Smola, Hornik, & Zeileis, 2004).
`Rgraphviz`	Graphics package used for creation of word association plots (Gentry et al., 2013).
`Snowball`	Provides stemming algorithms, including the aforementioned Porter stemming algorithm (Hornik, 2007).
`tm`	Provides capabilities for corpus creation, term-document matrix creation, and raw text formatting (strip white space, remove stop words, etc.). Additionally, functions are provided for studying the relationships between words and documents (Feinerer, 2008).

R Script and Functions

The main script ("hfjp_functions.R"), available for direct sourcing from the job postings website, contains the functions listed below. Before any analysis can be performed this script should be sourced first:

R Code

```
## change 'localhost' to correspond with the website directory as instructed on the HFJP website
source("localhost/hfjp_functions.R")
```

There are seven main functions used for analyzing data from the Human Factors Job Postings database. We will lay them out in a typical order of operation and provide some simple examples of use.

preProc

Description. `preProc` is used for pre-processing a raw dataset from the Human Factors Job Postings database. This function will remove punctuation, postings with zero informational value (no data in variables "Required Education", "Required Fields", "Required Experience", "Preferred", and "Job Description"), and create two new variables:

- *Year* - created by using pattern recognition on the "Date of Posting" variable to extract the year only.
- *Degree Level* - created by using pattern recognition on the "Required Education" variable to categorize postings into 7 predefined groups: PhD, Masters, Graduate, BS, BA, Undergraduate, and Unspecified.

Usage.

```
preProc(data)
```

Arguments.

`data`	an R data frame read in using the read.table function or its children functions (read.csv, etc.)

```
## set working directory
path<-'C:/Data/'
setwd(path)

## Read in CSV file
jobs <- read.csv("AllHumanFactorsJobs2013-04-18.csv", header=TRUE)

## Pre-Process data
test.data <- preProc(jobs)
```

genData

Description. `genData` is used to create several different data objects for use in later analysis. These include data frames, term document matrices, and collections of text documents (job postings), or, Corpus'. This function takes one necessary argument and three optional arguments. The number of data objects generated is dependent on the optional grouping and variables arguments. All data objects are returned in a list.

Usage.

```
genData(data,grouping=FALSE,group.names=FALSE,variables=FALSE)
```

Arguments.

`data`	An R data set returned by the `preProc` function.
`grouping`	A list of named vectors where the name corresponds to a column name in the data passed to the `data` argument and the values of each vector are the values to group together.
`group.names`	A vector of names for assignment to the groups in `grouping`.
`variables`	A vector of names to subset the data by. Names should correspond to the column names of the given `data` argument. This argument does not "split" the data by the given variables, but rather returns an aggregate paragraph of the given columns. If FALSE the default is to compose each document as an aggregate paragraph from the "Required Education", "Required Fields", "Required Experience", "Preferred", and "Job Description" columns. Example:

```
## Document composed by default

doctoral degree psychology experimental engineering personnel
    training
cognition physiological psychology research methodology    35  pass
    navy
physical standards candidates wil  commissioned  various positions
military  3   aviation systems development  acquisition process
    research
development test  evaluation aircrew simulation training systems
applications   navy  aviation selection program human factors
engineering human performance

## Document composed from Required Experience variable only

20  experience     crew system development  army navy air force nasa
     human
factors experience
```

Value. An object returned from `genData` is a list containing at least the following components:

`df.final`	The final dataframe used for construction of the corpus.
`corpus.full.s`	The full corpus (generated from all of the data), stemmed.
`corpus.full.ns`	The full corpus (generated from all of the data), not stemmed.
`tdm.full.s`	The term-document matrix for the full corpus, stemmed.
`tdm.full.ns`	The term-document matrix for the full corpus, not stemmed.

Additional components that may be returned depending on the call to `genData` include the following:

`corpus.list.s`	A list of stemmed corpus', one for each specified subgroup.
`corpus.full.ns`	A list of non-stemmed corpus', one for each specified subgroup.
`tdm.sub.stemmed`	A list of stemmed term-document matrices, one for each specified subgroup.
`tdm.sub.nstemmed`	A list of non-stemmed term-document matrices, one for each specified subgroup.

```
## regroup Degree.Level variable into two groups
## provide more descriptive names

grouping <- list(Degree.Level=c("masters","phd","graduate"),Degree.Level=c("ba","bs","
    undergrad"))
group.names <- c("Grad","Undergrad")

## generate data
info.test <- genData(test.data,grouping,group.names)

## second example
## would subgroup by year - one subgroup for each unique year
## would generate descriptive data for Required Experience...
## and Required Education variables only

info.test <- genData(test.data,"Year",c("Require.Experience","Required.Education"))

## third example
## regroup year variable into pre 2005 and post 2005

pre.2005 <- as.character(1990:2005)
post.2005 <- as.character(2006:2013)

grouping <- list(Year=pre.2005,Year=post.2005)
group.names <- c('< 2005','> 2005')

info.test <- genData(test.data,grouping,group.names)
```

descrAnl.freqs

Description. `descrAnl.freqs` is used to create ordered word frequency bar charts for data generated using the `genData` function. Frequency plots automatically subset the data to display only the 20 most frequent words in a term document matrix. If subsets were specified in the `genData` function then one plot for each subset will be created. Returns a plot and a full vector of frequencies (not limited to 20 terms).

Usage.

```
descrAnl.freqs(data,stemmed=TRUE)
```

Arguments.

`data`	A list of data objects returned by the `genData` function. Function will automatically select the appropriate data object from the list.
`stemmed`	TRUE or FALSE specifying whether or not to use the stemmed or non-stemmed term-document matrix. Defaults to TRUE.

Value. An object returned from `descrAnl.freqs` is a list containing the following components:

`plot`	The frequency bar chart. Each subgroup will have its own chart and will need to be accessed separately; see examples.
`table`	The full table of frequency counts; **not** restricted to the top 20. Each subgroup will have its own table and will need to be accessed separately; see examples.

```
## create word frequency plots from the first genData example
## two plots will be created - one for "Grad" & one for "Undergrad"

word.freqs.test <- descrAnl.freqs(info.test,stemmed=FALSE)

## extract frequency vectors for each group
word.counts <- lapply(word.freqs.test,function(x) x$table)

## plot bar charts for each group

lapply(word.freqs.test,function(x) x$plot)

## write graduate frequencies to new file

write.csv(word.freqs.test$Grad$table,'gradFreqs.csv')
```

descrAnl.corplot

Description. `descrAnl.corplot` is used to create word association plots for data generated using the `genData` function. Word association plots automatically subset the data to display only those correlations greater than 0.5 for the 20 most frequent words in a term document matrix. If subsets were specified in the `genData` function then one plot for each subset will be created. Correlations between two words are printed on the edges between their nodes.

Usage.

```
descrAnl.corplot(data,stemmed=TRUE,type="dot")
```

Arguments.

data	A list of data objects returned by the genData function. Function will automatically select the appropriate data object from the list.
stemmed	TRUE or FALSE specifying whether or not to use the stemmed or non-stemmed term-document matrix. Defaults to TRUE.
type	The type of plot to create. Default is "dot" and is generally most appropriate for these data. Other options are: "neato","twopi","circo", and "fdp"

```
## create word association plots from the first genData example
## two plots will be created - one for "Grad" & one for "Undergrad"

word.assoc.test <- descrAnl.corplot(info.test,stemmed=FALSE,type="dot")
```

spectral.clust

Description. spectral.clust is used for document clustering on data generated using the genData function. Clustering is performed using the specc function from the kernlab package. A spectrum kernel with a length of 4 is used to generate the kernel matrix. Returns a stemmed term-document matrix and corpus for each cluster in a list format. This list may be passed to the descrAnl functions to gain a sense of the latent trends in each cluster. Manually inspecting the corpus for each cluster can provide some information, to this same end, as well. **Ignores any groupings specified in the call to genData; clustering is performed on the full Corpus.**

Usage.

```
spectral.clust(data,nclust)
```

data	A list of data objects returned by the genData function. Function will automatically select the appropriate data object from the list.
nclust	The number of clusters to group documents into.

Arguments.

Value. An object returned from spectral.clust is a list containing the following components:

tdm.sub.stemmed	A list of stemmed term-document matrices, one for each cluster.
corpus.list.s	A list of stemmed corpus', one for each cluster.

```
## cluster documents into 2 clusters
spectral.clusters <- spectral.clust(info.test,nclust=2)

## inspect the first document in cluster one
spectral.clusters$corpus.list.s$'Cluster 1'[[1]]

## generate word association & frequency plots for the clusters.
## cannot use stemmed=FALSE
```

```
cluster.freqs <- descrAnl.freqs(spectral.clusters,stemmed=TRUE)
cluster.corplots <- descrAnl.corplot(spectral.clusters,stemmed=TRUE)
```

accProj

Description. `accProj` is used for making accuracy projections of a kernel support vector machine classification model for a given response. Iteratively splits the data into 80% training and 20% test to build a model and then test it's accuracy. Uses the `ksvm` function from the `kernlab` package with a spectrum kernel of length 4. **Ignores any groupings specified in the call to genData; classification is performed on the full Corpus.**

Usage.

```
accProj(data,resp,its,cost=1)
```

`data`	A list of data objects returned by the `genData` function. Function will automatically select the appropriate data object from the list.
`resp`	A vector containing the values for the response or dependent variable. This vector/-variable must be a factor/categorical.
`its`	Number of iterations to run. Higher number of iterations provides a more reliable assessment of the model's classification accuracy but at the cost of time to run.
`cost`	cost of constraints violation (default: 1) this is the "C" constant of the regularization term in the Lagrange formulation in `kernlab's ksvm` function. Higher values increase run time. See section 2.4 on support vector machines for a discussion on the side effects of changing this value.

Arguments.

Value. An object returned from `acc.Proj` is a list containing the following components:

`values`	A list containing the accuracy at each iteration.
`meanACC`	The accuracy averaged over all iterations.

```
## accuracy projection for classification of Degree Level
## using 10 iterations
model.check <- accProj(info.test,jobs.df[,"Degree.Level"],10)

## the accuracy at each iteration
model.check$values

## the mean accuracy
model.check$meanACC
```

modelBuild

Description. `modelBuild` is used for building a kernel support vector machine classification model on data generated using the `genData` function. Uses the `ksvm` function from the `kernlab` package with a spectrum kernel of length 4. **Ignores any groupings specified in the call to genData; classification is performed on the full Corpus.**

Usage.

```
modelBuild(data,resp,cost=1)
```

`data`	A list of data objects returned by the `genData` function. Function will automatically select the appropriate data object from the list.
`resp`	A vector containing the values for the response or dependent variable. This vector/-variable must be a factor/categorical.
`cost`	cost of constraints violation (default: 1) this is the "C" constant of the regularization term in the Lagrange formulation in `kernlab's ksvm` function. Higher values increase run time. See section 2.4 on support vector machines for a discussion on the side effects of changing this value.

Arguments.

```
## build a model for classification of Degree Level

class.model <- modelBuild(info.test,jobs.df[,"Degree.Level"])

## use the above model to make predictions for Degree Level on new data

new.data.file <- read.csv('New/Data') ## replace with data file

## pre-process new data file
new.data.pp <- preProc(new.data)

## make predictions of degree level requirements for new data
predictModel(class.model,new.data.pp)
```

predictModel

Description. `predictModel` is used for making predictions on a new set of data using a model returned from the `modelBuild` function. This is simply a wrapper function that helps to abstract the process of making predictions.

Usage.

```
predictModel(model,newData)
```

Arguments.

`model`	A kernel support vector machine model returned by the function `modelBuild`.
`newData`	The new data to make predictions on. If this data set happens to contain the response vector that we wish to predict it should be removed ahead of time. This data should be pre-processed ahead of time using the `preProc` function.

```
## build a model for classification of Degree Level

class.model <- modelBuild(info.test,jobs.df[,"Degree.Level"])

## use the above model to make predictions for Degree Level on new data

new.data.file <- read.csv('New/Data') ## replace with data file

## pre-process new data file
new.data.pp <- preProc(new.data)

## make predictions
predictModel(class.model,new.data)
```

Example Analysis

This section provides commentary and code for a complete analysis using the above functions. We'll start by setting our working directory and reading in a data file we've downloaded from the database.

Year Subgroups

Pre-processing. Next we will want to process this data set to prepare it for use in later analysis stages.

R Code

```
path <- 'C:/Users/Chris/Data/'
setwd(path)
jobs <- read.csv("AllHumanFactorsJobs2013-06-02.csv", header=TRUE)
```

R Code

```
test.data <- preProc(jobs)
```

At this point we need to decide what we want to look at from a descriptive/exploratory point of view. Since our data set has two categorical variables (Year and Degree Level) we can feasibly re-group the data based on levels of either of these variables. Additionally, we may only want to view descriptive information for a given field in the data (i.e. Required Experience). This is the kind of information that we would pass along as additional arguments to the `genData` function. Suppose we wanted to look at each year; first lets see how many documents belong to each year:

R Code

```
table(test.data[,"Year"])

0000 1969 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993
  21    1   15   19   31   18   25   20   58   29   39   53   53   60   17
1994 1995 1996 1997 1998 2003 2005 2006 2010 2011 2012 2013
  24    3   11   17    6    1    1    2   76   71   16   20
```

If we want to continue looking at Year subgroups it would make more sense to regroup the years into 10 year bins. Since the `0000` collection could possibly be made up of an aggregate of documents with 21 different years we're unlikely to gain any insight by analyzing it. Further, since there is only one document for the year `1969` and only 4 documents for the years `2000-2009` we will disregard these in our analysis. Lastly, we'll look at the main informational variables (Required Education & Required Fields grouped together, Required Experience, Preferred Experience, and Job Description) separately and thus make 5 separate calls to the `genData` function. If you're unsure of what the exact column names are you can always use the `colnames` function.

R Code

```
grouping <- list(Year=as.character(1980:1989),Year=as.character(1990:1999),Year=as
    .character(2010:2019))
group.names <- c("80's","90's","2010's")

info.exp <- genData(test.data,grouping,group.names,"Required.Experience")
info.jdescr <- genData(test.data,grouping,group.names,"Job.Description")
info.pexp <- genData(test.data,grouping,group.names,"Preferred")
info.edu.fields <- genData(test.data,grouping,group.names,c("Required.Education","
    Required.Fields"))
```

After running the above code we see output similar to the following:

R Code

```
[1] "0.78 sparsity factor applied"
[1] "0.83 sparsity factor applied"
[1] "0.95 sparsity factor applied"
Warning: High sparsity factor for subgroup.
Term-Document Matrix may have little informational value.

[1] "0.97 sparsity factor applied"
Warning: High sparsity factor for subgroup.
Term-Document Matrix may have little informational value.

[1] "0.89 sparsity factor applied"
[1] "0.96 sparsity factor applied"
Warning: High sparsity factor for subgroup.
Term-Document Matrix may have little informational value.

[1] "0.97 sparsity factor applied"
Warning: High sparsity factor for subgroup.
Term-Document Matrix may have little informational value.

[1] "0.9 sparsity factor applied"
Warning: High sparsity factor for subgroup.
Term-Document Matrix may have little informational value.
```

This is information relating to the creation of the term-document matrices and can generally be ignored. What it tells us though is that, for some of the groupings, the sparsity factor needed to be increased to reach the minimal number of terms (15) in the term-document matrix. Lower sparsity levels prune the term-document matrix more rigorously while higher levels are more lenient. As we can see, not every term-document matrix is the same and if we were to apply a common value across the board we might end up pruning all of the terms for a given matrix or retaining too many. High sparsity factors are an indicator of a low number of documents in a corpus.

Descriptive Analysis. Now that we've generated our data objects (corpus', term-document matrices, etc.) we can perform a descriptive analysis to help uncover any potential trends. We'll start with the word frequency plots and tables using the `descrAnl.freqs` function for each data group from above. In the code below we've extracted the plots, using the `lapply` function, into their own object, opened a new graphics windows, and plotted & arranged the frequency charts in a 2x2 grid. The plot is saved for use in this document and the graphics window is closed.

R Code

```
exp.freqs <- descrAnl.freqs(info.exp,stemmed=FALSE)
jdescr.freqs <- descrAnl.freqs(info.jdescr,stemmed=FALSE)
pexp.freqs <- descrAnl.freqs(info.pexp,stemmed=FALSE)
edu.fields.freqs <- descrAnl.freqs(info.edu.fields,stemmed=FALSE)

list.freqs <- list("Required Experience"=exp.freqs,"Job Description"=jdescr.freqs
    ,"Preferred Experience"=pexp.freqs,"Required Education & Required Fields"=edu.
    fields.freqs)

lapply(1:length(list.freqs), function(x) {
    plots <- lapply(list.freqs[[x]], function(y) {
      y$plot
    })
    title <- names(list.freqs)[x]
    windows(7,7)
    grid.arrange(plots[[1]],plots[[2]],plots[[3]],ncol=2,main=title)
    savePlot(filename=paste(path,"Plots/degreeFreqs-",title,sep=""),type="pdf",
        device=dev.cur())
    dev.off()
})
```

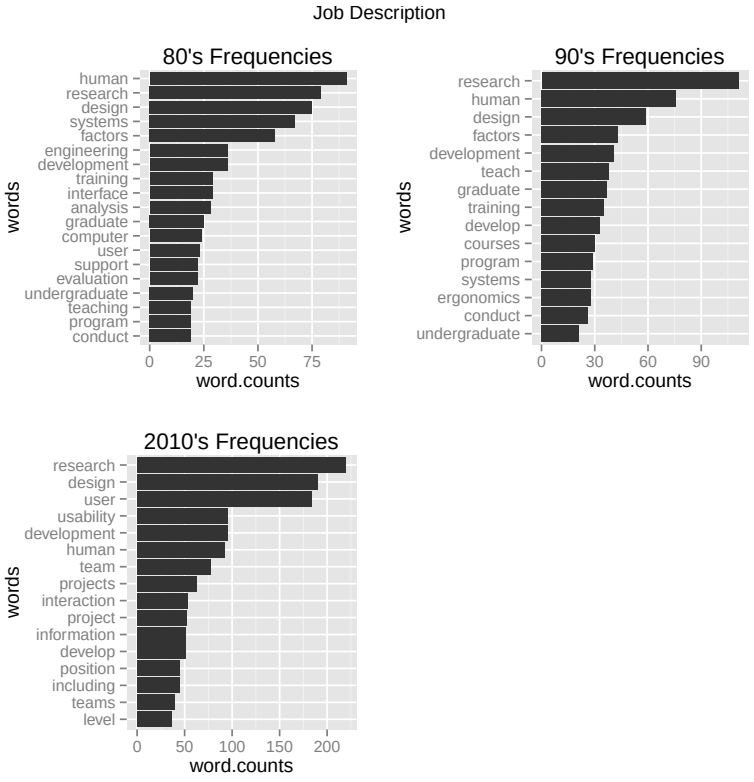

Figure 3. Frequency Plots for Job Description (Year Subgroups)

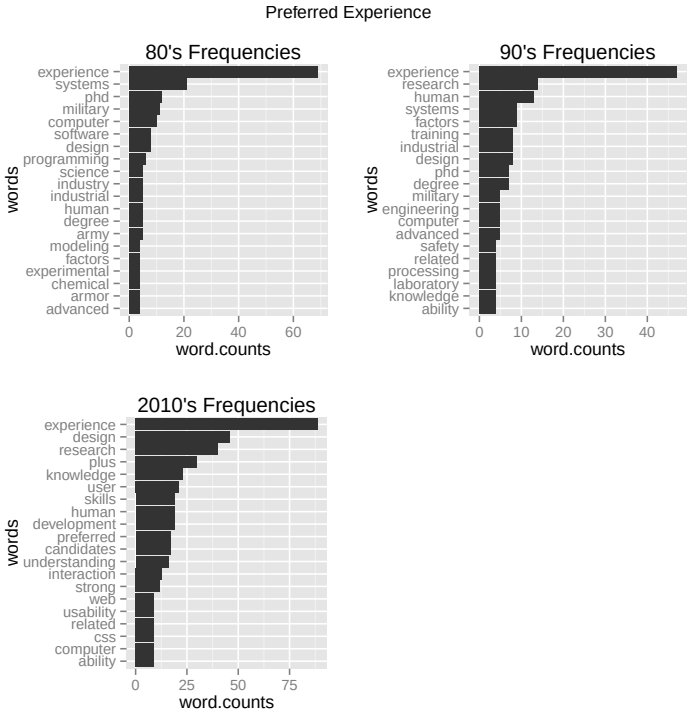

Figure 4. Frequency Plots for Preferred Experience (Year Subgroups)

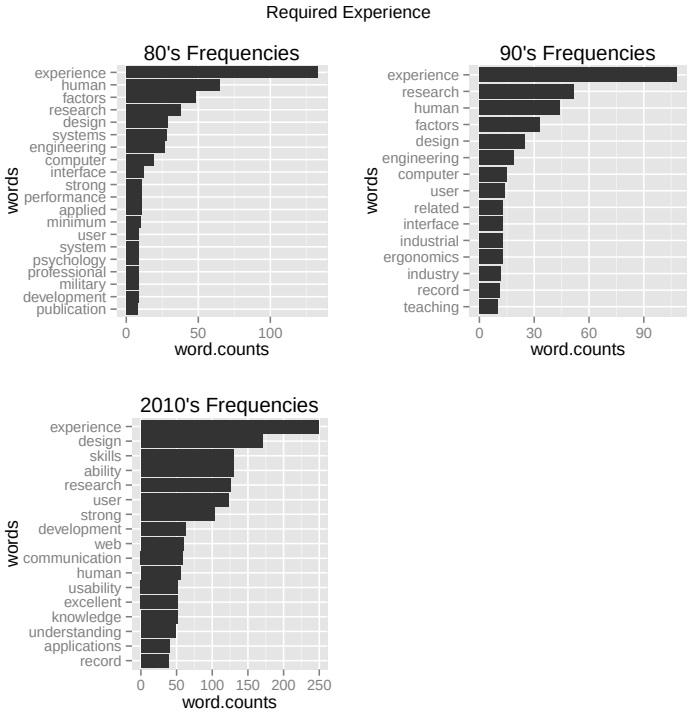

Figure 5. Frequency Plots for Required Experience (Year Subgroups)

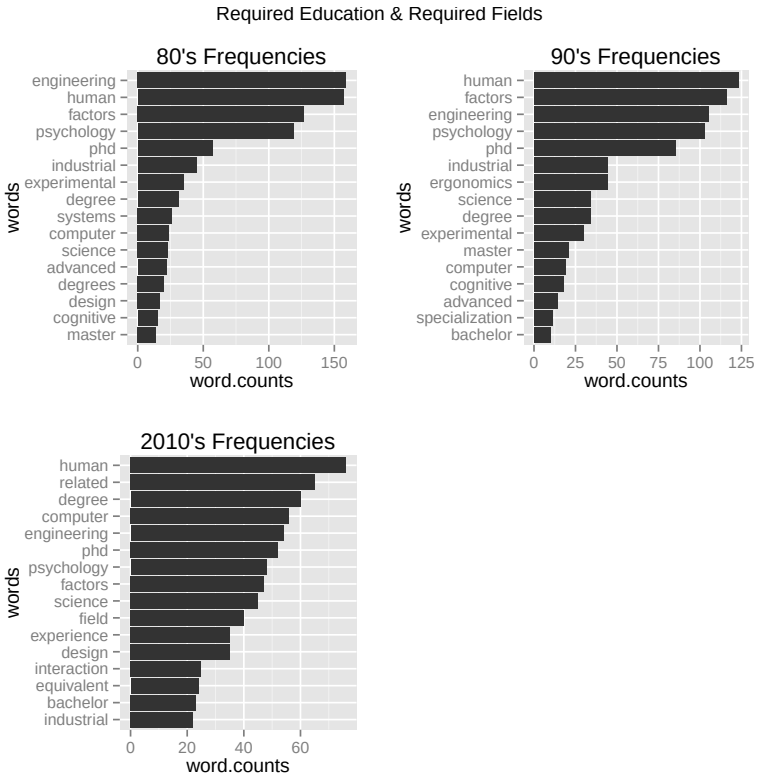

Figure 6. Frequency Plots for Required Education & Required Fields (Year Subgroups)

We can garner a few things from the above figures. From Figure 3 (Job Description) the most prominent trend is the decrease in systems development with the synchronous increase in projects/developments that are more user focused; from a macro level to a micro level. In both Figure 4 (Preferred Experience) and Figure 5 (Required Experience) we can see an upward trend for candidates with computer experience from the 80's to the 90's and then a focus on web development from the 90's to the 2010's. This is reflected in Figure 6 (Required Experience & Required Education) as well; by 2010 the biggest difference revolves around computers. A second point of interest from Figure 6 is the strong focus on ergonomics in the 1990's.

Degree Level Subgroups

Pre-Processing & Descriptive Analysis. A second analysis we may wish to perform is on required Degree Level. The code below lays out how we'll regroup the original Degree Level variable into only two groups. Further we are no longer specifying a subset of variables, but rather are using the default to develop our corpus. We ask for word association plots on the last line as well.

R Code

```
grouping <- list(Degree.Level=c("masters","phd","graduate"),Degree.Level=c("ba","bs","undergrad"))
group.names <- c("Grad","Undergrad")

info.test <- genData(test.data,grouping,group.names)
word.freqs.test <- descrAnl.freqs(info.test,stemmed=FALSE)

plots <- lapply(word.freqs.test,function(x) x$plot)

windows()
grid.arrange(plots[[1]],plots[[2]],ncol=2)
savePlot(filename=paste(path,"Plots/degreeFreqs",sep=""),type="pdf",device=dev.cur())
dev.off()

descrAnl.corplot(info.test,stemmed=FALSE,type="dot")
```

We can see from Figure **??** that the terms are mostly the same between the two groups with one major difference: job postings requiring a graduate level degree are more research based, while postings requiring only an undergraduate degree would, then, be more applied based. The word association plots would tell us this as well.

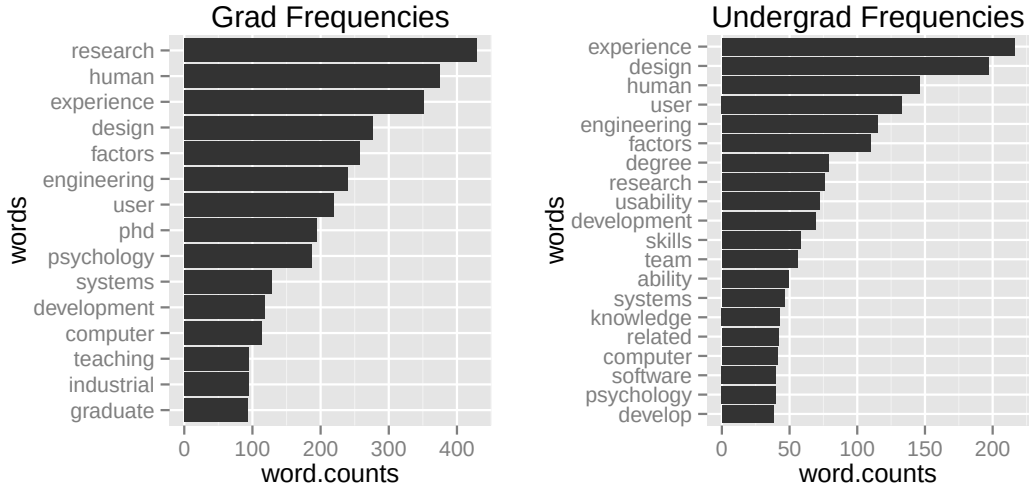

Figure 7. Frequency Plots for Degree Subgroups

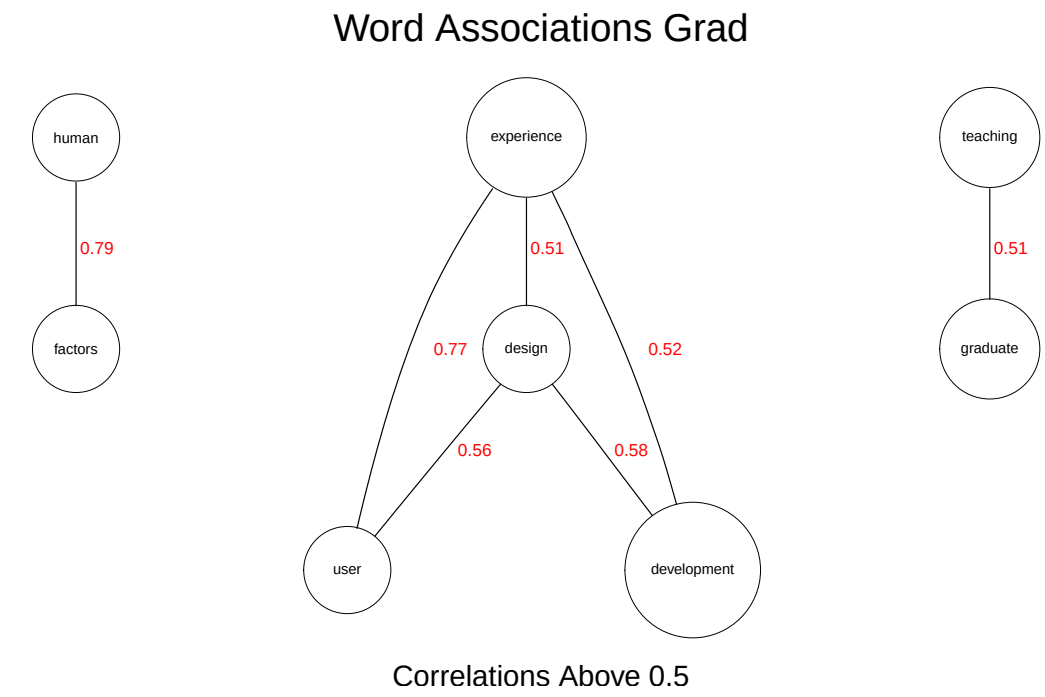

Figure 8. Word associations, graduate degrees

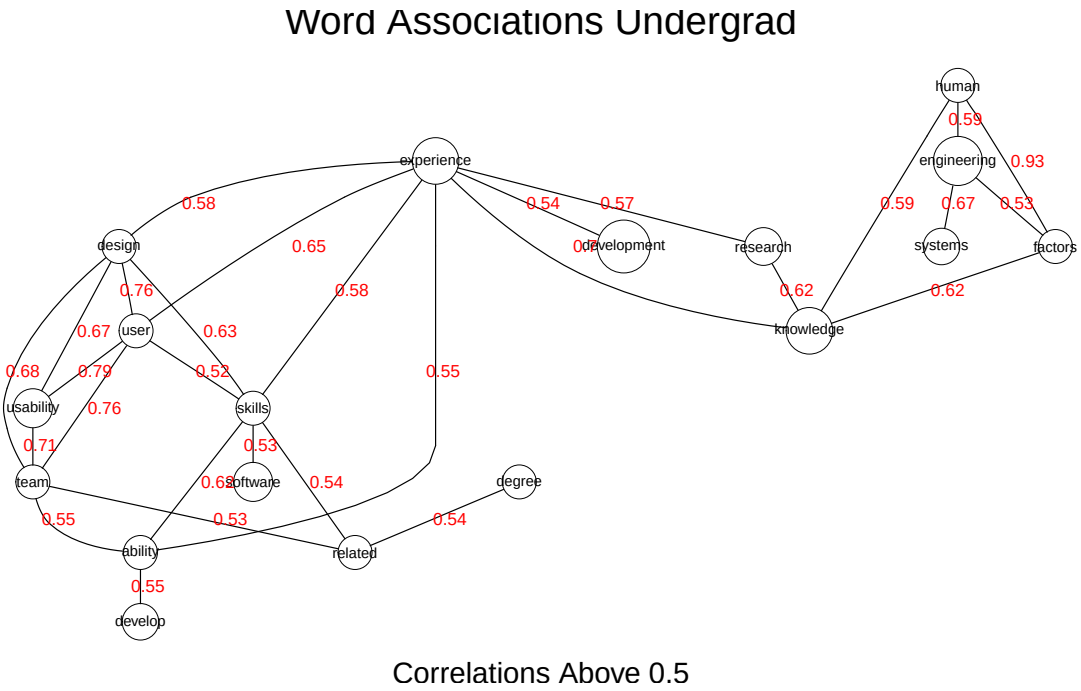

Figure 9. Word associations, undergraduate degrees

Model Classification. In order to assess the difference between Degree Levels more scientifically we may run the `accProj` function with Degree Level as our response. Lower model accuracy would tell us that the model is having a hard time distinguishing between the classes and thus that they're may not be much of a difference in the language of their respective postings. In the code below, before we run the accuracy projection, we create a new response vector that recodes the "Degree Level" variable into "Graduate", "Undergrad", and "Unspecified". As noted in the documentation for this function, we build a model on the entire corpus of documents, not on the corpus of each subgroup; therefore, we cannot rely on the Degree Level grouping we originally passed to the `genData` function. Further, we are running 30 iterations in the code below and we specify a penalization value on the errors (cost) of 1.5 to allow for a little more slack in hopes of achieving better model generalization.

R Code

```
graduate <- c("phd","masters","graduate")
undergrad <- c("ba","bs","undergrad")
grad.rows <- which(test.data[,"Degree.Level"] %in% graduate)
ugrad.rows <- which(test.data[,"Degree.Level"] %in% undergrad)

resp.vec <- as.character(test.data[,"Degree.Level"])
resp.vec[grad.rows] <- "Graduate"
resp.vec[ugrad.rows] <- "Undergrad"
resp.vec <- as.factor(resp.vec)

model.check <- accProj(info.test,resp.vec,30,cost=1.5)
model.check$meanAcc
[1] 0.7073286
```

On average, we would expect a model seeking to classify postings into a "Graduate", "Undergraduate", or "Unspecified" degree level category to be 71% accurate. These groups may, therefore, be separable by the language of their postings with our current model. From this point we could use the `modelBuild` function to generate a model from all of the data. This model could then be used to classify the degree level of new job postings. As an example, we will separate out the "Unspecified" postings, build a model on the rest of the data, and then use the model to predict whether those postings are seeking "Graduate" level candidates or "Undergraduate" level candidates.

R Code

```
resp.num <- match("Degree.Level",colnames(test.data))
unspec.degree.data <- test.data[-c(grad.rows,ugrad.rows),-resp.num]
train.data <- test.data[c(grad.rows,ugrad.rows),-resp.num]

## use the recoded response vector we generated above
train.resp.vec <- as.factor(resp.vec[c(grad.rows,ugrad.rows)])

train.list <- genData(train.data)

dl.model <- modelBuild(train.resp.vec,train.list)

predictModel(dl.model,unspec.degree.data)
  [1] Graduate  Graduate  Undergrad Graduate  Graduate  Graduate  Graduate
  [8] Graduate  Undergrad Undergrad Graduate  Graduate  Graduate  Graduate
 [15] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [22] Graduate  Graduate  Graduate  Undergrad Undergrad Graduate  Graduate
 [29] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [36] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [43] Graduate  Graduate  Undergrad Graduate  Graduate  Graduate  Graduate
 [50] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [57] Graduate  Graduate  Graduate  Undergrad Graduate  Graduate  Graduate
 [64] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [71] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [78] Graduate  Graduate  Graduate  Undergrad Graduate  Graduate  Graduate
 [85] Graduate  Graduate  Graduate  Undergrad Graduate  Graduate  Undergrad
 [92] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
 [99] Undergrad Graduate  Undergrad Graduate  Graduate  Graduate  Graduate
[106] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[113] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[120] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[127] Graduate  Undergrad Undergrad Graduate  Graduate  Graduate  Graduate
[134] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[141] Graduate  Graduate  Graduate  Undergrad Graduate  Graduate  Graduate
[148] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[155] Graduate  Graduate  Graduate  Graduate  Graduate  Undergrad Graduate
[162] Graduate  Undergrad Graduate  Graduate  Graduate  Graduate  Graduate
[169] Graduate  Undergrad Graduate  Graduate  Graduate  Undergrad Graduate
[176] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Undergrad
[183] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[190] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[197] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[204] Graduate  Graduate  Graduate  Graduate  Undergrad Undergrad Undergrad
[211] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[218] Graduate  Undergrad Graduate  Graduate  Undergrad Graduate  Graduate
[225] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[232] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[239] Graduate  Undergrad Graduate  Graduate  Graduate  Graduate  Graduate
[246] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[253] Undergrad Undergrad Graduate  Graduate  Graduate  Graduate  Graduate
[260] Graduate  Graduate  Graduate  Graduate  Graduate  Undergrad Graduate
[267] Graduate  Graduate  Graduate  Graduate  Graduate  Graduate  Graduate
[274] Graduate  Graduate  Graduate  Undergrad Undergrad Graduate  Undergrad
[281] Undergrad Undergrad Graduate  Undergrad Undergrad Undergrad Undergrad
[288] Graduate
```

Spectral Clustering. Additionally, we may wish to see what clusters of documents arise in an unsupervised task - that is, when we do not specify a response variable. After some trial and error it was determined that 3 clusters provided some of the most useful interpretations after passing the returned data object to the `descrAnl` functions.

R Code

```
spectral.clusters <- spectral.clust(info.test,nclust=2)
cluster.freqs <- descrAnl.freqs(spectral.clusters,stemmed=FALSE)

lapply(cluster.freqs,function(x) x$table)
$`Cluster 1`
  research      human      teach     experi  psycholog      engin        phd
       527        205        189        175        157        140        139
    factor    program    graduat    develop     scienc undergradu      cours
       137        119        114        110         78         67         65
  industri     strong
        60         60

$`Cluster 2`
  design   experi     user  develop interact research     team  project
     668      461      396      280      202      198      185      168
   human    skill   comput   system interfac   applic    engin    relat
     166      160      146      141      137      131      129      103
   degre
      91

$`Cluster 3`
    human    factor     engin    experi    system    design psycholog
      468       373       322       237       190       147       130
  develop  industri    comput  interfac  research     degre       phd
      114       104        80        66        66        64        57
  perform
       56
descrAnl.corplot(spectral.clusters,stemmed=TRUE,type="dot")
```

Inspecting the above frequency tables we can see the job postings may be generally classified as those referencing research and faculty/teaching oriented positions (Cluster 1) , those referencing more application based, systems development positions (Cluster 2), and those dealing primarily with human factors engineering (Cluster 3). The word association plots bear this out as well.

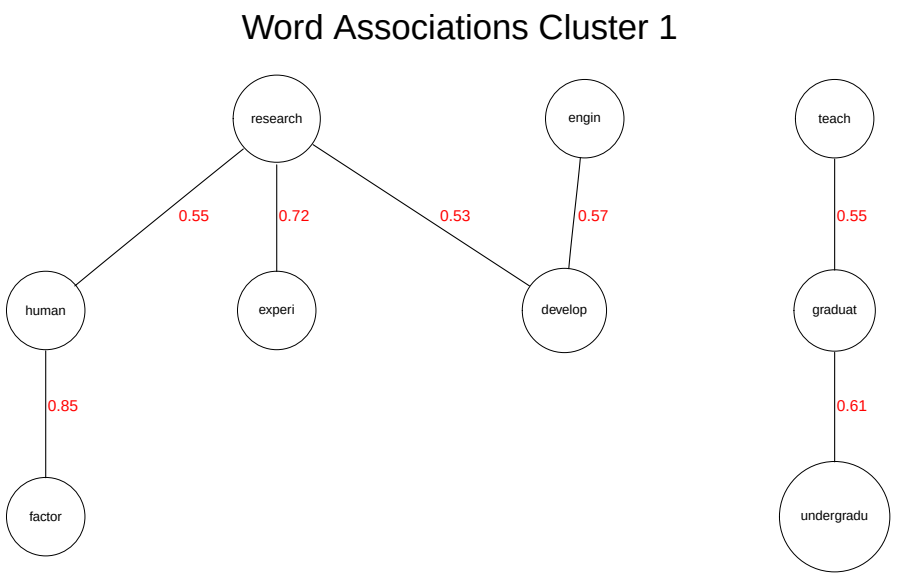

Figure 10. Word associations cluster 1

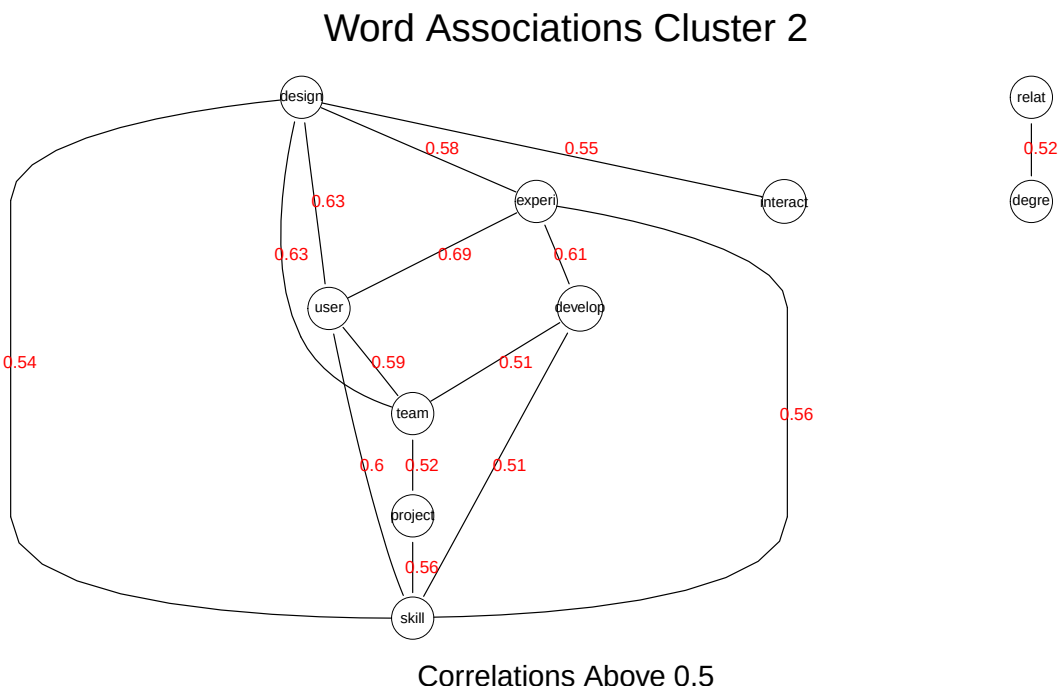

Figure 11. Word associations cluster 2

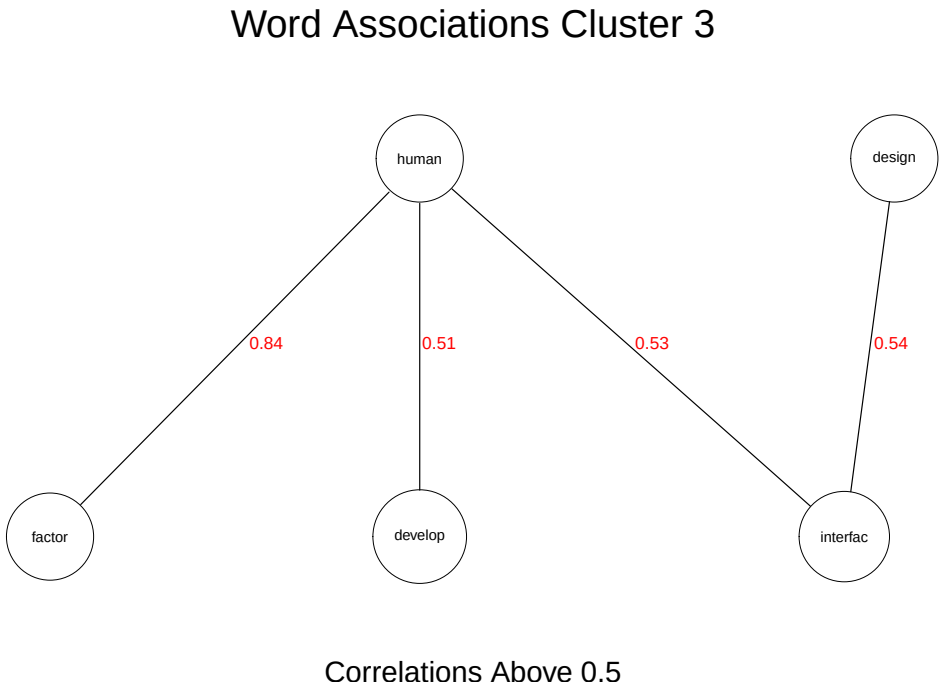

Figure 12. Word associations cluster 3

Summary and Conclusions

The purpose of the database described here is to continually track skills and knowledge expectations in the labor market for new HF/E professionals. Eventually, this task will involve longitudinal analyses of trends across periods of several years. Presently, upon creation of the database, it contains data collected by a small number of people involved in the Workforce Issues Task Force of the HFES over the last two and half years. These data represent a sample of HF/E jobs advertised at the HFES Career Center, various HFES technical group mailing lists, and sites such as ACM-SIGCHI jobs and HFcareers.com. Clearly, we cannot expect the current dataset to offer a comprehensive picture of HF/E jobs 2010-2013, nor can there be guarantees that the new database will in time contain a representative sample of HF/E jobs. Databases such as this are only as good as the data they contain, and the greatest effort is therefore to continually enter new data into it. We hope, however, that this effort will not prove to be too onerous with the new web interface of the database and if a sufficient number of people will commit to tracking HF/E job announcements and entering the relevant variables into the database. Given the overall size of the HF/E profession, a dozen or so committed individuals could probably sustain the database with good quality data.

The design of the database also considers foreseeable analysis needs. The data are available and downloadable by anyone with Internet access, and our hope is that as the database grows, it will stimulate interest in tracking labor market trends and making corresponding adjustments in academic curricula, helping keep HF/E programs relevant. We intend to publish a variety of different analyses and new results in different outlets, including publications by the HFES, and we invite others to do likewise. Our objective is to influence the development of HF/E curricula and ultimately develop more skilled HF/E professionals who are responsive to the needs of industry, government, and consulting organizations.

References

Baptiste, A. (2012). *gridExtra: functions in Grid graphics*. R package version 0.9.1. http://CRAN.R-project.org/package=gridExtra.

Boley, D., Gini, M., Gross, R., Han, E.H., Karypis, G., Kumar, V., Mobasher, B., Moore, J., Hastings, K. (1999). Partitioning-based Clustering for Web Document Categorization. *Decision Support Systems, 27*(3), 329-341. ISSN 0167-9236.

Clarke, B.S., Fokoué, E., & Zhang, H.H. (2009). *Principles and theory for data mining and machine learning*. Springer. ISBN 0387981357.

Cooke, N. J. & Gorman, J. C. (2004). What do HFES members need to know? *HFES Bulletin, 47*(4), pp. 1, 4-6. Santa Mon-ica, CA: HFES.

Feinerer, I. (2008). *tm: Text mining package*. R package version 0.3-2, http://CRAN.R-project.org/package=tm.

Feinerer, I., Hornik, K., & Meyer, D. (2008). Text mining infrastructure in R. *Journal of Statistical Software, 25*(5), 1-54. http://www.jstatsoft.org/v25/i05.

Gentry, J., Long, L., Gentleman, R., Falcon, S., Hahne, F., Sarkar, D., & Hansen, K. D. (2013). *Rgraphviz: Provides plotting capabilities for R graph objects*. R package version 2.2.1.

Hornik, K. (2007). *Snowball: Snowball Stemmers*. R package version 0.0-1.

Karatzoglou, A. & Feinerer, I. (2006). Text Clustering with String Kernels in R. *Research Report Series / Department of Statistics and Mathematics, 34*. Department of Statistics and Mathematics, WU Vienna University of Economics and Business, Vienna.

Karatzoglou, A., Smola, A., Hornik, K., & Zeileis, A. (2004). kernlab - An S4 Package for Kernel Methods in R. *Journal of Statistical Software 11*(9), 1-20. URL http://www.jstatsoft.org/v11/i09/

Karatzoglou, A. & Feinerer, I. (2010). Kernel-based machine learning for fast text mining in R. *Computational Statistics & Data Analysis, 54*(2), 290-297.

Leslie, C., Eskin, E., & Noble, W.S. (2002). The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Biocomputing Symposium*.

R Core Team (2013). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Ng, A.Y., Jordan, M.I., & Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems, 14*, 1-8.

Porter, M. (1980). An algorithm for suffix stripping. *Program, 3*, 130-137.

Rantanen, E. M. Moroney, W. F. (2011). Educational and skill needs for new human factors/ergonomics professionals. *Proceedings of the 55th Annual Meeting of the Human Factors and Ergonomics Society* (pp. 530-534). Santa Monica, CA: HFES.

Rantanen, E. M. & Moroney, W. F. (2012). Employers' expectations for education and skills of new human factors/ergonomics professionals. *Proceedings of the 56th Annual Meeting of the Human Factors and Ergonomics Society* (581-585). Santa Monica, CA: HFES.

Rantanen, E. M., Claeys, C., Roder, D., Moroney, W. F. (2013). Archival human factors jobs database as a tool for tracking trends in skills and knowledge expectations in the labor market. *Proceedings of the 57th Annual Meeting of the Human Factors and Ergonomics Society*. Santa Monica, CA: HFES.

Schölkopf, B., Smola, A. (2002). *Learning with Kernels*. MIT Press.

Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Computing Surveys, 34*(1), 1-47. ISSN 0360-0300.

Shi, J. & Malik, J. (2000). Normalized cuts and image segmentation. *Transactions on Pattern Analysis and Machine Intelligence, 22*(8), 888–905.

Vapnik, V.N. (1998). *Statistical learning theory*. Springer.

Weiss, S., Indurkhya, N., Zhang, T., & Damerau, F. (2004). *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer-Verlag. ISBN 0387954333.

Wickham, H. (2009). *ggplot2: elegant graphics for data analysis*. Springer New York.

Zhao, Y., Karypis, G. (2005). Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery, 10*(2), 141-168.

Appendix: R code

This code was written in R by Chris Claeys for initial analyses of the HF/E jobs data as contained in the database in June 2013.

```
#################
### Functions ###
#################

## for MAC
Sys.setenv(NOAWT= "true")

## load Rgraphviz separately ##
if(require("Rgraphviz",character.only=TRUE)==FALSE) {
  source("http://bioconductor.org/biocLite.R")
  biocLite("Rgraphviz")
}
library("Rgraphviz")

##function to load required packages##
##will attempt to download package if it is not found locally##

load.package<- function(name) {
  temp.result<-require(name, character.only=TRUE)
  if(temp.result==FALSE) {
    install.packages(name)
    library(name, character.only=TRUE)
  }
}

##the list of packages##
libs <- c('ape','tm','kernlab','ggplot2','Snowball')

#iterate through the libs vector and run the function##
invisible(lapply(libs,load.package))


##if error "JAVA_HOME cannot be determined from the Registery"##
##uncomment & manually set the path with the line below (sys.setenv...)##
##change location to your Java location##
##if you're using 32-bit R make sure you point it to 32-bit Java##
##vice-versa for 64-bit##

#Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre7')


## function to plot word correlations ##
## data is a Document Term Matrix or Term Document Matrix ##
## hcs is a vector of high correlation terms ##

plot.term.cors <- function(data,hcs,type,title="") {
  if (!require("Rgraphviz"))
    stop("could not find (bioconductor.org) Rgraphviz package")

  m <- if(inherits(data,"TermDocumentMatrix"))
    t(data) else data
  #subset matrix to high cor terms#
```

```
  counts <- apply(as.matrix(data),1,sum)
  w.counts <- names(counts)[order(counts,decreasing=TRUE)][1:20]

  m <- as.matrix(m[,hcs[hcs %in% w.counts]])

  #generate correlation matrix and replace diagonal and
  #values less than .5 with 0
  c <- cor(m)
  c[(c < .5) ] <- 0
  diag(c) <- 0

  #instantiate graph object
  g1 <- as(c,"graphNEL")

  #build node size for each node
  nodes <- buildNodeList(g1)
  fixed.size <- rep(FALSE, times=length(nodes))
  names(fixed.size) <- names(nodes)

  #build correlation labels for edges
  labels <- sapply(edgeNames(g1), function(x) {
    t.loc <- regexpr("~",x)
    first.word <- substr(x,1,t.loc-1)
    sec.word <- substr(x,t.loc+1,nchar(x))
    round(c[first.word,sec.word],2)
  })

  #set edge and node attributes for each edge & node
  eAttrs <- list(label=labels)
  nAttrs <- list(fixedsize=fixed.size)

  #set some additional node attrs across all nodes
  nodeRenderInfo(g1) <- list(textCol="black", lwd=1,fontsize=12,cex=.8)

  #set some additional edge attrs across all edges
  edgeRenderInfo(g1) <- list(headport="s",cex=.
8,lty=1,lwd=1.2,col="black",textCol="red")

  #set graph attributes
  parRenderInfo(g1) <- list(graph=list(main=paste("Word Associations",title,sep="
"),
                                       sub="Correlations Above 0.5", cex.main=1.8,
                                       cex.sub=1.4, col.sub="black"))
  #generate layout
  g1 <- layoutGraph(g1,edgeAttrs=eAttrs,nodeAttrs=nAttrs,layoutType=type)

  #render
  renderGraph(g1)
}

## find high correlation terms (>.5) in a Term Document Matrix
## data parameter is a Term Document Matrix
```

```
high.cts <- function(data) {
  if(is(data,"TermDocumentMatrix")==FALSE)
    stop("Data needs to be of class TermDocumentMatrix")
  terms <- findFreqTerms(data)
  term.cors <- sapply(terms,function(x) findAssocs(data,x,.5))
  (high.cor.terms <- names(unlist(lapply(term.cors,function(x)
 which(length(x)>0)))))
}



preProc <- function(data) {

  ## create year variable from date variable
  date.col <- grep("[Dd]ate",colnames(data),perl=TRUE)
  year.pattern <- "\\d{4}"
  year.char.start <- regexpr(year.pattern,data[,date.col],perl=TRUE)
  data.new <- cbind(data[,-
 date.col],substr(data[,date.col],year.char.start,year.char.start+3))
  colnames(data.new)[ncol(data.new)] <- 'Year'

  phd.vec <- c('phd','doctorate','doctorate or masters','doctoral')
  masters.vec <- c('master','masters','ms','bachelors or masters')
  bs.vec <- c('bs','bachelor of science')
  ba.vec <- c('ba','bachelor of arts','bachelor','bachelors')
  undergrad <- c('undergraduate','degree')
  grad <- c('graduate','advanced degree','advanced')


  ## make an education level columns
  education.rows <- data[,"Required.Education"]
  splits <- lapply(education.rows,function(x) {
    strsplit(tolower(as.character(x)),split=" ")
  })
  level.match <- function(vec) {
    unlist(lapply(1:length(splits),function(x) {
      if(any(splits[[x]][[1]] %in% vec))
        x
    }))
  }

  phd.rows <- level.match(phd.vec)
  full <- phd.rows

  masters.rows <- level.match(masters.vec)
  masters.rows <- masters.rows[-which(masters.rows %in% full)]
  full <- c(full, masters.rows)

  grad.rows <- level.match(grad)
  grad.rows <- grad.rows[-which(grad.rows %in% full)]
  full <- c(full,grad.rows)
```

```
  bs.rows <- level.match(bs.vec)
  bs.rows <- bs.rows[-which(bs.rows %in% full)]
  full <- c(full,bs.rows)

  ba.rows <- level.match(ba.vec)
  ba.rows <- ba.rows[-which(ba.rows %in% full)]
  full <- c(full,ba.rows)

  undergrad.rows <- level.match(undergrad)
  undergrad.rows <- undergrad.rows[-which(undergrad.rows %in% full)]
  full <- c(full,undergrad.rows)

  unspec.rows <- seq_len(length(education.rows))
  unspec.rows <- unspec.rows[-which(unspec.rows %in% full)]

  degree.list <-
list(phd=phd.rows,masters=masters.rows,graduate=grad.rows,bs=bs.rows,ba=ba.rows,und
ergrad=undergrad.rows,unspecified=unspec.rows)
  degree.level <- c()
  for(i in 1:length(degree.list)) {
    degree.level[degree.list[[i]]] <- names(degree.list)[i]
  }

  data.new <- cbind(data.new,degree.level)
  colnames(data.new)[ncol(data.new)] <- 'Degree Level'
  jobs.df <- as.data.frame(data.new)

  ## remove punctuation
  ## character replacement for "." is separate
  jobs.df <- as.data.frame(lapply(jobs.df,gsub,pattern="\
\.",replacement="",perl=TRUE))
  jobs.df <- as.data.frame(lapply(jobs.df,gsub,pattern="[[:punct:]]",replacement="
",perl=TRUE))
  jobs.df

}

genData <-
function(data,grouping=list(FALSE),group.names=c(FALSE),variables=c(FALSE)) {
  if(is(data,"data.frame")==FALSE)
    stop("data argument must be of class data.frame")
  if(any(grouping[[1]]!=FALSE)) {
    if(is(grouping,"list")==FALSE && is(grouping,"character")==FALSE)
      stop("grouping argument must be a list or a character string")
    if(is(grouping,"list")) {
      if(any(unlist(lapply(grouping,function(x) any(!is.character(x)))))==TRUE)
        stop("individual elements within grouping vectors must be character
strings")
      if(any(!is.vector(grouping)))
        stop("grouping list should contain vectors")
    }
```

```
  }
  if(any(group.names!=FALSE)) {
    if(any(!is.character(group.names)))
      stop("group names must all be character strings")
  }
  if(variables[1]!=FALSE) {
    if(is(variables,"vector")==FALSE)
      stop("variables argument be a vector")

    if(any(!is.character(variables)))
      stop("all elements of 'variables' vector must be character strings")
  }
  #group.data <- function(data,grouping) {
    if(any(grouping[[1]]!=FALSE)) {
      if(is(grouping,"list")) {
        if(length(group.names)==length(grouping)) {
          names.list <- group.names
        } else {
          names.list <- paste("Group",1:length(grouping))
        }
        data.list <- lapply(1:length(grouping),function(x) {
          var.name <- names(grouping)[x]
          rows.df <- unlist(lapply(1:length(grouping[[x]]),function(y) {
              which(data[,var.name]==grouping[[x]][y])
            })
          )
          as.data.frame(data[rows.df,])
        })
      } else {
        unique.levs <- levels(data[,grouping])
        names.list <- unique.levs
        data.list <- lapply(unique.levs,function(x) {
          rows.df <- which(data[,grouping]==x)
          as.data.frame(data[rows.df,])
        })
      }
    } else {
      data.list <- list(data)
      names.list <- "Full"
    }
    #list(data.list,names.list)
  #}
  #grouped.data <- group.data(data,grouping)
  #ames.list <- grouped.data[[2]]
  #data.list <- grouped.data[[1]]

  #data.list <- list(jobs.df)
  ## Default variable selection
  nn.vars <-
c('Degree.Level','Year','Added.Timestamp','Area','Benefits','Salary','Organization'
)
  nn.cols.num <- unlist(lapply(nn.vars,grep,x=colnames(data)))
```

```
 variables.deaf <- colnames(data)[-nn.cols.num]
 df.alt <- data[,-nn.cols.num]
 data.list.full <- lapply(list(data), function(x) {
   x[,variables.deaf]
 })
 if(variables[1]==FALSE) {
   data.list <- lapply(data.list, function(x) {
     x[,variables.deaf]
   })
 } else {
   data.list <- lapply(data.list, function(x) {
     x[,variables]
   })
 }

 #data.list <- lapply(data.list, function(x) {
  # x[,'Required.Experience']
# })
 ## convert everything to a character string
 ## necessary for proper corpus creation
 data.list <- lapply(data.list,function(x) apply(as.data.frame(x),2,as.character))
 data.list.full <- lapply(data.list.full,function(x) apply(as.data.frame(x),
2,as.character))

 ## remove "NA"
 remove.na <- function(data) {
   temp.list <- lapply(data, function(w) {
     w <- as.data.frame(w)
     w <- apply(w,2,as.character)
     w <- as.data.frame(w)
     lapply(1:nrow(w), function(x) {
       lapply(1:ncol(w), function(y) {
         str.vec <- paste(unlist(w[x,y]),collapse=" ")
         splits <- strsplit(str.vec," ")
         lapply(splits[[1]], function(z){
           if(z!="NA")
             z
         })
       })
     })
   })
   lapply(temp.list,function(x) {
     lapply(x,function(y) {
       paste(unlist(y),collapse=" ")
     })
   })
 }
 data.list.a <- remove.na(data.list)
 data.list.full <- remove.na(data.list.full)

 ## turn each entry into block paragraph structure
```

```
  if(any(grouping[[1]]!=FALSE))
    names(data.list.a) <- names.list

  ## create corpus
  ## will create one corpus for each grouping
  corpus.full.stemmed <- lapply(data.list.full,corpusCreate,stemmed=TRUE)[[1]]
  corpus.full.nstemmed <- lapply(data.list.full,corpusCreate,stemmed=FALSE)[[1]]
  tdm.full.stemmed <- tdmCreate(corpus.full.stemmed,sparsity=.6)
  tdm.full.nstemmed <- tdmCreate(corpus.full.nstemmed,sparsity=.8)

  return.list <- list(df.final = df.alt, corpus.full.s =
list(corpus.full.stemmed),corpus.full.ns = list(corpus.full.nstemmed),tdm.full.s =
list(tdm.full.stemmed),tdm.full.ns = list(tdm.full.nstemmed))
  if(grouping[[1]]!=FALSE || variables[1]!=FALSE) {
    corpus.list.stemmed <- lapply(data.list.a,corpusCreate,stemmed=TRUE)
    corpus.list.nstemmed <- lapply(data.list.a,corpusCreate,stemmed=FALSE)

    ## create TDM
    ## will crate one TDM for each grouping
    tdm.stemmed <- lapply(corpus.list.stemmed,tdmCreate,sparsity=.6)
    tdm.nstemmed <- lapply(corpus.list.nstemmed,tdmCreate,sparsity=.8)

    return.list <-c(return.list,corpus.list.s = list(corpus.list.stemmed),
corpus.list.ns = list(corpus.list.nstemmed),tdm.sub.stemmed =
list(tdm.stemmed),tdm.sub.nstemmed = list(tdm.nstemmed))
  }
  return(return.list)
  #return(list(data.list,data.list.a,data.list.full))
}


corpusCreate <- function(data,stemmed) {
  if(is(data,"list")==FALSE)
    stop("data argument must be of type list")
  if(!isTRUE(stemmed) && !identical(stemmed,FALSE))
    stop("stemmed argument must be either TRUE or FALSE")


  ##create a Corpus (collection of documents) from our data##
  ##and run pre-processing/sanitization functions##
  ##requires "tm" pacakge##

  ds <- Corpus(VectorSource(data))

  ##remove all whitespace##

  ds <- tm_map(ds, stripWhitespace)

  ##convert to lowercase##

  ds <- tm_map(ds, tolower)
```

```
  ##Remove Stopwords##

  ds <- tm_map(ds, removeWords, stopwords("english"))

  ##Stemming##
  ##requires package 'Snowball'##


  if(stemmed==TRUE) {
    ds.stem <- tm_map(ds, stemDocument)
    ds.stem
  }
  else {
    ds
  }
}


tdmCreate <- function(data, sparsity=.8) {
  if(is(data,"Corpus")==FALSE)
    stop("data argument must be of type Corpus")
  if(is(sparsity,"numeric")==FALSE)
    stop("sparsity argument must be of type numeric")
  if(sparsity>1 || sparsity < 0)
    stop("sparsity must be a value between 0 and 1")
  ##Term-Document Matrix##

  ds.tdm <- TermDocumentMatrix(data)

  ##remove sparse terms##
  ##remove terms sparisty% empty/occur 1-sparsity% of the time##
  ds.tdm.trim <- removeSparseTerms(ds.tdm,sparsity)

  sparsity <- .6
  sparsity.start <- sparsity
  nwords.tdmt <- nrow(as.matrix(ds.tdm.trim))

  while(nwords.tdmt < 15 && sparsity.start < 1) {
    sparsity.start <- sparsity.start+.01
    ds.tdm.trim <- removeSparseTerms(ds.tdm,sparsity.start)
    nwords.tdmt <- nrow(as.matrix(ds.tdm.trim))
  }
  print(paste(sparsity.start,"sparsity factor applied",sep=" "))
  ds.tdm.trim
}


descrAnl.corplot <- function(data,stemmed=TRUE,type="dot",title="") {
  if(is(data,"list")) {
    if(is.null(data$tdm.sub.stemmed)==FALSE) {
      if(stemmed==TRUE) {
        data <- data$tdm.sub.stemmed
```

```
      } else {
        data <- data$tdm.sub.nstemmed
      }
    } else {
      if(stemmed==TRUE) {
        data <- data$tdm.full.s[[1]]
      } else {
        data <- data$tdm.full.ns[[1]]
      }
    }
    lapply(1:length(data), function(x) {
      if(is(data[[x]],"TermDocumentMatrix")==FALSE)
        stop("data argument must be of type TermDocumentMatrix")

      #find terms with correlation >.5
      #see high.cts function at start of script
      high.cor.terms <- high.cts(data[[x]])

      #plot term correlations
      #see plot.term.cors function at start of script
      name.list <- gsub("(^|[[:space:]])([[:alpha:]])", "\\1\\U\\2", names(data)
[x], perl=TRUE)
      plot.term.cors(data[[x]],high.cor.terms,type,name.list)
    })
  }
  if(is(data,"TermDocumentMatrix")) {
    high.cor.terms <- high.cts(data)

    #plot term correlations
    #see plot.term.cors function at start of script
    plot.term.cors(data,high.cor.terms,type,title)
  }
}

descrAnl.freqs <- function(data,stemmed=TRUE) {
  if(!is.null(data$tdm.sub.stemmed)) {
    if(stemmed==TRUE) {
      data <- data$tdm.sub.stemmed
    } else {
      data <- data$tdm.sub.nstemmed
    }
  } else {
    if(stemmed==TRUE) {
      data <- data$tdm.full.s
    } else {
      data <- data$tdm.full.ns
    }
  }
  out <- lapply(1:length(data), function(x) {

    if(is(data[[x]],"TermDocumentMatrix")==FALSE)
      stop("data argument must be of type TermDocumentMatrix")
```

```
    ## get word counts##

    word.counts <- apply(as.matrix(data[[x]]),1,sum)

    ## sort word counts
    word.counts <- sort(word.counts,decreasing=TRUE)

    ## keep the top 20 only
    if(length(word.counts)>20) {
      word.counts <- word.counts[1:20]
    }

    ## create dataframe of counts
    df.counts <- as.data.frame(cbind(words=names(word.counts),word.counts))

    ## reorder factor levels to correspond with counts
    word.levels <- names(word.counts)[order(word.counts,decreasing=FALSE)]

    ## set "words" variabel to above & reset counts to numeric
    df.counts$words <- factor(df.counts$words, levels = word.levels)
    df.counts$word.counts <- c(word.counts)

    ## capitalization of names of current iteration
    name.list <- gsub("(^|[[:space:]])([[:alpha:]])", "\\1\\U\\2", names(data)[x],
 perl=TRUE)
    #generate plot of frequencies
    barplot <- ggplot(df.counts, aes(x=words,y=word.counts))
    barplot <- barplot + geom_bar(stat="identity", position="dodge") + coord_flip()
    barplot <- barplot + ggtitle(paste(name.list,"Frequencies",sep=" "))
    barplot

    list(plot=barplot,table=word.counts)
  })
  names(out) <- names(data)
  out
}


nclust.ideal <- function(data) {
  if(is(data,"TermDocumentMatrix")==FALSE)
    stop("data argument must be of type TermDocumentMatrix")
  set.seed(123)
  wcss <- sapply(1:10,function(x) kmeans(as.matrix(data),x)$tot.withinss)
  ang <- atan(abs(diff(log(wcss))))*(180/pi)
  point.inf <- which(scale(ang)<0)[1]
  point.inf-1
}

descrAnl.radial <- function(data,nclust) {
  if(is(data,"TermDocumentMatrix")==FALSE)
    stop("data argument must be of type TermDocumentMatrix")
```

```
  ##hierarchical clustering of terms using ward's method##
  hc <- hclust(dist(as.matrix(data)), method = "ward")

  word.counts <- apply(as.matrix(data),1,sum)

  ## radial hierarchical clustering plot of terms ##
  clust.cut <- cutree(hc, nclust)
  mypal <- c(1:nclust)
  cex.scale<-.7+log(word.counts)-min(log(word.counts))
  par(bg="#ffffff",mar=c(1,1,1,1),oma=c(1,1,1,1))
  plot(as.phylo(hc), type="fan", tip.color=mypal[clust.cut], label.offset=1,
       cex=cex.scale, col="red")
}

spectral.clust <- function(data,nclust,sparsity=.6) {

  data <- data$corpus.full.s[[1]]

  if(is(data,"Corpus")==FALSE)
      stop("data argument must be of type Corpus")
  ## Document Clustering ##
  ## this should be performed on the corpus not the TDM ##

  ## specc doesn't provide centers
  ## combine with word correlation plots to get a sense of what
  ## each document cluster may represent

  ## define string kernel
  string.kern <- stringdot(type = "spectrum",length=4, normalized=TRUE)

  ## generate kernel matrix from data
  kernmat <- kernelMatrix(kernel=string.kern,x=data)

  ## spectral clustering
  set.seed(123)
  specc.one <- specc(kernmat,centers=nclust)

  ## plots word correlations for each cluster
  data.corpus <- lapply(1:nclust, function(x) {
    corpus.list.s <- data[which(specc.one==x)]
  })
  data.tdm <- lapply(1:nclust, function(x) {
    new.data <- tdmCreate(data.corpus[[x]],sparsity=sparsity)
  })
  names(data.corpus) <- paste("Cluster",1:nclust,sep= " ")
  names(data.tdm) <- paste("Cluster",1:nclust,sep= " ")
  return.list <- list(tdm.sub.stemmed = data.tdm,corpus.list.s=data.corpus)
  return(return.list)
}

##kSVM classification##
```

```
##response needs to be a factor##

accProj <- function(its,resp,info) {
    stemList <- info$corpus.full.s[[1]]
    origData <- resp
    if(is(its,"numeric")==FALSE)
      stop("its argument needs to be of type numeric")
    if(is(info,"list")==FALSE)
      stop("info argument needs to be of type list")
    if(is(resp,"vector")==FALSE && is(resp,"factor")==FALSE)
      stop("resp argument needs to be of type vector or factor")

    string.kern <- stringdot(type = "spectrum",length=8,normalized=TRUE)
    accuracies <- lapply(1:its, function(x) {
      idt <- sample(1:length(stemList),length(stemList)*.8)
      js.train <- stemList[idt]
      js.test <- stemList[-idt]

      js.train.y <- resp[idt]
      js.test.y <- resp[-idt]

      ksvmTrain <- ksvm(js.train,as.factor(js.train.y),type="C-svc",
 kernel=string.kern)
      test.preds <- predict(ksvmTrain,js.test)
      mod.acc <- length(which(test.preds==js.test.y))/length(js.test.y)
      mod.acc
    })
    meanAcc <- mean(unlist(accuracies))
    list("values"=accuracies,"meanAcc"=meanAcc)
}




modelBuild <- function(resp,stemList) {
  stemList <- stemList$corpus.full.s[[1]]
  if(is(stemList,"list")==FALSE)
    stop("stemList argument needs to be of type list")
  if(is(resp,"vector")==FALSE && is(resp,"factor")==FALSE)
    stop("resp argument needs to be of type vector or factor")
  string.kern <- stringdot(type = "spectrum",length=8,normalized=TRUE)
  model <- ksvm(stemList,as.factor(resp),type="C-svc",kernel=string.kern)
  model
}
```

```
## working example of analyzing HFJP database

## set working directory
path<-'C:/Users/Chris/Data/'
setwd(path)

## Read in CSV file
jobs <- read.csv("AllHumanFactorsJobs2013-04-18.csv", header=TRUE)


## preProc performs numerous preprocessing steps on the data
## removes punctuation
## creates a year column
## creates a degree level column
test.data <- preProc(jobs)

## genData generates list of data objects
## corpus', tdm's, and dataframes for use in later analysis
## function description: genData(data,grouping,group.names,variables)
## data = data returned from the preProc function
## grouping = list of groupings
#### strucutre list as list(VariableName = c("level1","level2"),etc.)
#### see example
## group.names = names for each group, a vector
## variables = vector of variables to keep
#### for example: c("Required.Experience")
#### would generate data for the Required.Experience field only

## regroup Degree.Level variable into two groups
## provide more descriptive names
grouping <-
 list(Degree.Level=c("masters","phd","graduate"),Degree.Level=c("ba","bs","undergrad
 "))
group.names <- c("Grad","Undergrad")

## generate data
info.test <- genData(test.data,grouping,group.names)

## second example
## would subgroup by year - one subgroup for each unique year
## would generate descriptive data for Required Experience...
## and Required Education variables only
## spectral analysis and ksvm model building would still...
## be done on the full data set
## uncomment below line and run to see the difference

### info.test <-
 genData(test.data,"Year",c("Require.Experience","Required.Education"))


## third example
## regroup year variable into pre 2005 and post 2005
```

```
## uncomment below lines and run to see the difference

### pre.2005 <- as.character(1990:2005)
### post.2005 <- as.character(2006:2013)

### grouping <- list(Year=pre.2005,Year=post.2005)
### group.names <- c('< 2005','> 2005')

### info.test <- genData(test.data,grouping,group.names)


## word association plots
## uses the returned data object from the genData function
## function description: descrAnl.corplot(data,stemmed,type,title)
## data = pass data object returned from genData function
## stemmed = TRUE or FALSE...
## dictates whether to used stemmed words or not, defaults to TRUE
## type = "dot", "neato","twopi","circo",or "fdp", defaults to "dot"
## title = titles for plots
## best to leave blank and let function automatically title the plot
## makes one word association plot for each grouping
#### in the above example, one plot for Grad and one for Undergrad

descrAnl.corplot(info.test,stemmed=FALSE,type="dot")

## term frequency plot
## function description: descrAnl.freqs(data,stemmed)
## data = pass data object returned from genData function
## stemmed = TRUE or FALSE, defaults to TRUE
## returns a list of objects
#### one frequency plot for each grouping
#### one table of counts for each grouping
#### in the above example, one plot& table for Grad & Undergrad

descrAnl.freqs(info.test,stemmed=FALSE)


## spectral clustering - performed on corpus not a TDM
## function description: spectral.clust(data,nclust,sparsity)
## data = pass data object returned from genData function
## nclust = number of clusters to divide data into
## sparsity = pruning level for new Term-Document Matrices...
## where higher values = less pruning...
## values between 0 and 1 only, defaults to 0.6
## function returns a list of objects:
#### one corpus for each cluster
#### one tdm for each cluster
## the data object returned can be used in the
## descrAnl.freqs and descrAnl.corplot functions however,
## the "stemmed" argument should always be TRUE
## use the output of descrAnl functions to identify what each...
## cluster might represent
```

```
spectral.clusters <- spectral.clust(info.test,nclust=2,sparsity=0.6)
descrAnl.freqs(spectral.clusters,stemmed=TRUE)
descrAnl.corplot(spectral.clusters,stemmed=TRUE)


## function description: accProj(its,resp,data)
## its = number of iterations to run
## resp = response column, a factor or a vector...
## from a data object returned by the preProc function
## data = data returned from genData function
## compute accuracy of a kSVM model for predicting
## the degree level over 10 iterations
## NOTE: this process can take a long time
## ~1 minute for each iteration
## not advisable to run high # of iterations

#model.check <- accProj(10,jobs.df[,"Degree.Level"],info.test)
#model.check$values ## see accuracy for each iteration
#model.check$meanACC ## see overall accuracy

## build a model on the full data
## use for making future predictions on new data
## function description: modelBuild(resp, data)
## resp = response column, a factor or vector...
## from a data object returned by the preProc function
## data = data object returned by the genData function

#class.model <- modelBuild(jobs.df[,"Degree.Level"],info.test)

## use the above model to make predictions for year
## on new data

#new.data.file <- read.csv('New/Data') ## replace with data file

## pre-process new data file
#new.data.pp <- preProc(new.data)

## generates data objects, but only saves the full, stemmed corpus
## full, stemmed corpus is what we need for predictions
#new.data <- genData(new.data.pp)$corpus.full.s

## make predictions of degree level requirements for new data
#predict(class.model,new.data)
```