Spring 5-20-2019

NEXT LEVEL: A COURSE RECOMMENDER SYSTEM BASED ON CAREER INTERESTS

Shehba Shahab
San Jose State University

NEXT LEVEL: A COURSE RECOMMENDER SYSTEM BASED ON CAREER INTERESTS

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Shehba Shahab

May 2019

The Designated Thesis Committee Approves the Thesis Titled

NEXT LEVEL: A COURSE RECOMMENDER SYSTEM BASED ON CAREER INTERESTS

by

Shehba Shahab

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2019

Dr. Robert Chun	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science

ABSTRACT

Next Level: A Course Recommender System Based on Career Interests

by Shehba Shahab

Skills-based hiring is a talent management approach that empowers employers to align recruitment around business results, rather than around credentials and title. It starts with employers identifying the particular skills required for a role, and then screening and evaluating candidates' competencies against those requirements. With the recent rise in employers adopting skills-based hiring practices, it has become integral for students to take courses that improve their marketability and support their long-term career success. A 2017 survey of over 32,000 students at 43 randomly selected institutions found that only 34% of students believe they will graduate with the skills and knowledge required to be successful in the job market. Furthermore, the study found that while 96% of chief academic officers believe that their institutions are very or somewhat effective at preparing students for the workforce, only 11% of business leaders strongly agree [11]. An implication of the misalignment is that college graduates lack the skills that companies need and value. Fortunately, the rise of skills-based hiring provides an opportunity for universities and students to establish and follow clearer classroom-to-career pathways. To this end, this paper presents a course recommender system that aims to improve students' career readiness by suggesting relevant skills and courses based on their unique career interests.

ACKNOWLEDGMENTS

I would like to thank two important groups of people without whom this thesis would not have been possible: my committee and my family. I would like to first thank the members of my thesis committee for their intellectual contributions to my development as a computer scientist. To my thesis advisor, Dr. Robert Chun, thank you for your guidance throughout the research process and for connecting me with opportunities to get Next Level into the hands of students at San Jose State University. To my committee members, Dr. Thomas Austin and Dr. Pollett, without the appreciation and excitement inspired by your lectures in Advanced Programming Language Principles and Information Retrieval, respectively, I may never have pursued this challenging area. Finally, but not least, I want to thank my family for their constant support of all of my professional and academic endeavors.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Problem

Success starts with a plan. The earlier students take ownership of their academic plans, the more likely they are to graduate on time and find success in an increasingly competitive job market. Studies have shown that every additional year of enrollment in college costs students more than $26,000 in tuition, fees, books, and living expenses, as well as more than $22,000 in lost lifetime wages. It is estimated that students in the California State University system who take six years to earn a bachelor's degree will incur $110,900 in extra expenses and lost wages than students who graduate within four years[1]. Despite numerous incentives to begin academic planning early, many students feel ill-equipped to navigate the complicated process of course selection. Without adequate tools to facilitate their decision-making, students often spend more time hunting for data to make an informed decision than using it to craft a solid plan.

Educational needs vary from student to student based on their career goals and skills-gap. There is no one-size fits all solution for student success. In this context, personalized course recommender systems have proven to be useful supplements to traditional academic advising in helping students select relevant courses for their specific goals. Researchers have attempted to perfect the art of course recommendations for the last decade. [3] recommended courses using a hybrid recommender system that connected learners' preferences with ratings given to learning content by similar users, [5] recommended courses based students' job interests using a manually annotated corpora, and [6] recommended courses based

on the courses students had taken in previous semesters. After researching the strengths and weaknesses of each of these previous approaches, we propose Next Level, a course recommender system that helps students discover relevant skills and courses based on their unique career interests.

1.2 Proposed Solution

Skills-based hiring is a talent management approach that empowers employers to align recruitment around business results, rather than around credentials and title. It starts with employers identifying the particular skills required for a role, and then screening and evaluating candidates' competencies against those requirements. With the recent rise in employers adopting skills-based hiring practices, it has become integral for students to take courses that improve their marketability and support their long-term career success. A 2017 survey of over 32,000 students at 43 randomly selected institutions found that only 34% of students believe they will graduate with the skills and knowledge required to be successful in the job market. Furthermore, the study found that while 96% of chief academic officers believe that their institutions are very or somewhat effective at preparing students for the workforce, only 11% of business leaders strongly agree[11]. An implication of the misalignment is that college graduates lack the skills that companies need and value. Fortunately, the rise of skills-based hiring provides an opportunity for universities and students to establish and follow clearer classroom-to-career pathways. To this end, we propose Next Level, a course recommender system that uses content-based filtering and an ensemble of k-means clustering and TF-IDF keyword extraction to help students discover skills and courses based on their unique career interests. Unlike existing statistical keyword extraction techniques such as TextRank, RAKE

and TF-IDF that can only extract keywords from context, Next Level's ensemble algorithm combines the benefits of TF-IDF's keyword extraction with the power of unsupervised k-means clustering to mine large volumes of job descriptions and identify closely related skills to students' career interests. This allows Next Level to serve as a discovery tool for both skills and courses. We believe the expanded search scope will result in higher quality course recommendations than existing keyword extraction techniques and k-means clustering on its own. To test our hypothesis, we will benchmark Next Level's recommendations against k-means clustering, TextRank, TF-IDF and RAKE on the basis of precision and recall. Next Level's approach offers several advantages over previous recommender systems:

1. **The results are highly relevant:** Because content-based recommendations rely on characteristics of objects themselves, the course recommendations are likely to be highly relevant to the user's unique job interests and are not biased by course ratings from peers with dissimilar career goals.

2. **Recommendations are transparent:** The process by which any recommendation is generated can be made transparent, which may increase students' trust in their recommendations or allow them to tweak the results.

3. **New items can be recommended immediately:** Unlike collaborative-filtering, content-based filtering does not require a user to interact with an item before it can be recommended. Furthermore, Next Level's ensemble approach is able to rely on data outside of the user's basic query. This can be useful technique for automatically expanding the search scope when the user's query does not yield any matching courses.

4. **Users can get started more quickly:** Avoids cold-start problem because a

user must enter search terms in order to get recommendations.

5. **Benefits multiple stakeholders:**

   - **Students:** Students can discover both the skills and the courses needed to obtain their dream jobs.
   - **Educators:** Educators can learn trends in the industry and evolve Course Learning Outcomes accordingly.
   - **Employers:** Students enter the workforce with the skills needed to make an immediate impact to the organization.

1.3 Organization

The remainder of the paper is organized as follows: Chapter 2 surveys related literature on the topic of course recommender systems. In Chapter 3, we provide background on the topic of recommender systems. In Chapter 4, we review existing techniques for keyword extraction. In Chapter 5, we discuss strategies for taming unstructured text. In Chapter 6, we detail our technical implementation. In Chapter 7, we describe the results of our experiment. Finally, Chapter 8 concludes the paper and presents possible future work.

CHAPTER 2

Related Work

Crafting the perfect course schedule requires careful consideration of a number of factors ranging from a students' career interests to scheduling conflicts, major requirements, time commitment, grade potential and professor ratings. Over the last decade, several researchers have proposed recommender systems taking one or more of these factors into account. This chapter presents an overview of previous approaches and their relative advantages and disadvantages.

2.1 Learning Objects: Building Blocks for Course Recommendations

The explosive rise of e-learning in the early 2000s led to efforts to standardize media content for ease of search and retrieval. The term "learning object" was first coined by Wayne Hodgins in 1994 to describe media that was accessible by, reusable across, and interoperable with multiple learning management systems. This paradigm called for a standardized and structured data model for capturing descriptive attributes about media [2]. In his 2002 paper entitled "The Future of Learning Objects" Hodgins discussed the importance of widespread adoption of learning objects as a building block for personalized learning. It was his belief that standardization would be key to connecting learning content and learners in the future. While Hodgins did not build a course recommender system himself, his vision became the inspiration for several early course recommender systems. We will describe some of these approaches in the subsequent sections.

2.2 Hybrid Course Recommender System Based on Ontologies

In 2006, the National Cheng Kung University proposed a hybrid content-collaborative algorithm for learning content recommendations [3]. This was one of the earliest experiments in the domain of learning content discovery and extended Hodgin's idea of using learning objects to connect learners with learning content. In this study, a preference-based algorithm was used to calculate a learner's preference score and a neighbor-interest algorithm used the experiences of similar learners to calculate an interest score. The two scores were aggregated to generate a final recommendation score.

2.2.1 Feature Extraction and User Preference Profile

The study assumed the set of relevant learning objects for each course were previously generated. The relevant characteristics of the learning objects were defined in compliance with the IEEE Learning Object Metadata (LOM) standard. In pursuit of adaptive personalized recommendations, the recommendation system extracted the features of learning objects into the set c. Each f_i represented a feature of the learning object lo. These values were stored in a learning object profile data structure. Learners interactions with learning objects were captured in their Learner Profile History (LPH) and sorted by their unique preferences for the learning object. The user preference score was calculated such that if a user studied a learning object but did not rate it, a score of 1 was assumed. If the user later rated the learning object, the score was replaced with the user rating. The feedback range was between 1 and 5.

2.2.2 Preference-Based Algorithm

A content-based filtering preference algorithm was used to bias the recommendations. The algorithm assigned a preference score, *p-score* to a learning object if the feature was found in the user's preferences, *LPH*. A collaborative-filtering based nearest neighbor algorithm was used to incorporate the recommendations of similar users into the model. This was a two-part algorithm that first required finding neighbors with similar profiles to the active learner. The formula used the feedback scores for items rated by both the target and neighbor and the averages of feedback scores for all learning objects shared between the two users to determine similarity. The second part of the algorithm involved calculating an interest score for the neighbors returned in part one. The interest formula formula predicted the active user's interest in the recommended learning objects for the course based on the interests of similar learners. This value was normalized by the MaxScore an item could be given, the value 5.

2.2.3 Recommendations

Finally, learning objects were recommended by aggregating the preference and relevance scores using the formula *RS(lo)*, shown in Figure 1. The *p-weight* and *i-weight* were used to capture sentiment over a period of time to account for changes in the active learner and neighbors preferences over time. At the time of the study, the weights were assigned a period of 1 month.

$$RS(lo) = \frac{1}{\frac{p\_weight}{pscore} + \frac{i\_weight}{pscore}}$$

Figure 1: Recommendations Using Aggregating Preference and Relevance Scores

2.2.4 Strengths and Weaknesses

As a hybrid approach, this study was able to circumvent the problem of overspecialization that is characteristic of content-based recommendation systems. Likewise, it circumvented the cold-start problem characteristic of collaborative filtering based systems by falling back on learners' preferences if no similar neighbors were found. In this case, both approaches were handled in memory and therefore susceptible to scalability issues as the item and user databases grew in size. Furthermore, as an ontology-based approach, the model was highly accurate but required manual annotations by a domain-expert for optimal results. If the annotations were sparse or inaccurate then it could bias the recommendations. From a functional perspective, ratings from other users could be a highly subjective variable to introduce into a recommender system. Students often rate courses which are "easy" higher than courses which are more difficult but provide them the skills needed to move further in their careers. This could artificially inflate the value of a course in the eyes of a student.

2.3 CourseRank

In 2007, the InfoLab at Stanford University developed CourseRank, a social tool for course planning and discovery [4]. CourseRank allowed users to read and write course reviews, view grade distributions for classes, and plan their course loads through personalized recommendations. The tool differed from traditional recommendation systems in that it was not "hard-wired" to support a fixed set of recommendation algorithms. Rather, students and designers (administrators) were provided the flexibility to define their own recommendation workflows that could be executed over the relational database in real-time. This approach aimed to address

three fundamental limitations with traditional course recommendation systems:

- Not all learners found "hard-wired" recommendations useful
- Designers could not experiment with new recommendation algorithms without modifying system code
- Recommendation systems were typically based on either item content or ratings, they did not utilize rich data representations

CourseRank's approach allowed designers to build recommendation workflows using either traditional content or collaborative filtering, or a custom hybrid approaches if neither was suitable. Designers could choose from among several similarity measures including cosine, pearson and jaccard. These formulas were abstracted to library functions and could be referenced from within the workflows. Additionally, designers were provided flexibility in choosing how items were weighted.

2.3.1 Recommendation Expression Trees

Recommendation workflows were translated into recommendation expression trees such as the one shown in Figure 2. In this example, the workflow has two recommend operators. The lower one finds students similar to the active student (StudId <> 444) using the euclidean distance of their ratings. The upper recommend operator finds courses recommended by these students and takes a weighted average of their ratings to make the final recommendation for the active user.

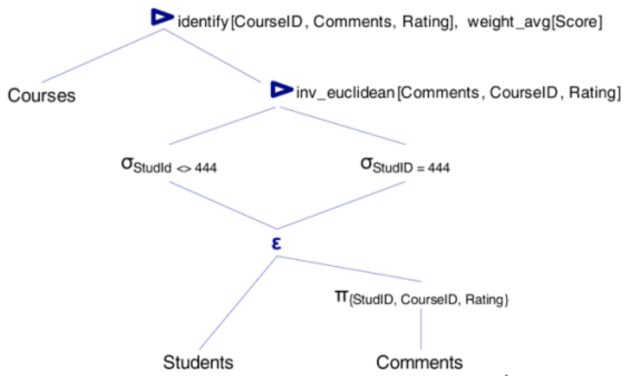

Figure 2: Recommendation Expression Tree

2.3.2 System Architecture

As illustrated in Figure 3, designers defined workflows using the Workflow Manager. The Query Parser parsed the workflows into expression trees such as the one shown in Figure 2. The Recommendation Plan Generator constructed a sequence of SQL statements based on the input expression tree. The Recommendation Generator interacted directly with the MySQL database to execute the recommendation plan and generate the recommendations for the active user. Finally, a student would interact with the CourseRank user interface to execute the workflows defined by the designers for their organization. Figure 4 shows a screenshot of the advice panel students interacted with in the 2009 version of the tool.

2.3.3 Strengths and Weaknesses

CourseRank improved transparency around recommendations results and was largely considered a success. Yet, it required designers to understand the data model in order to model relationships between classes. The recommendation engine, like its predecessors, was built in memory and susceptible to the same scalability

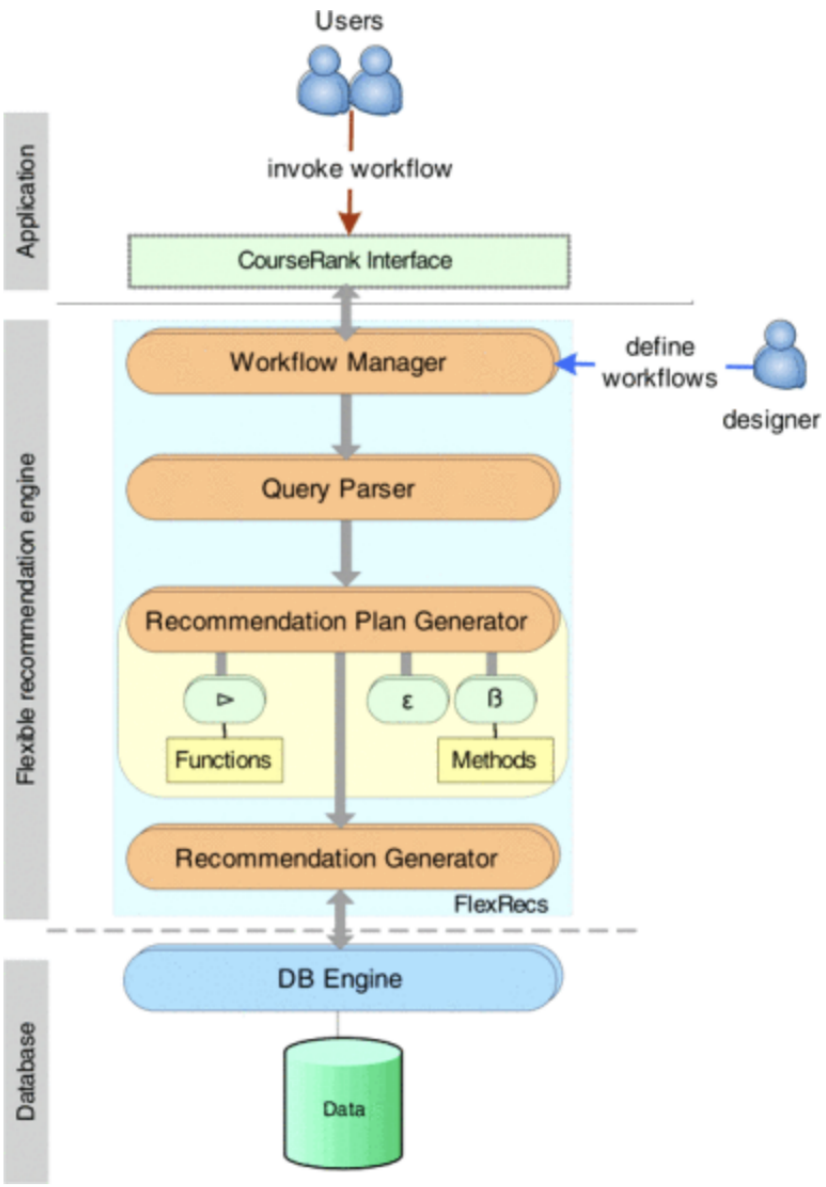

Figure 3: System Architecture for CourseRank

issues that plagued other recommendation systems of its time.

2.4 A Semantic Recommender System for Adaptive Learning

A 2015 research project by the Polytechnic University of Turin proposed a recommender system that suggested courses that would improve a learner's chances

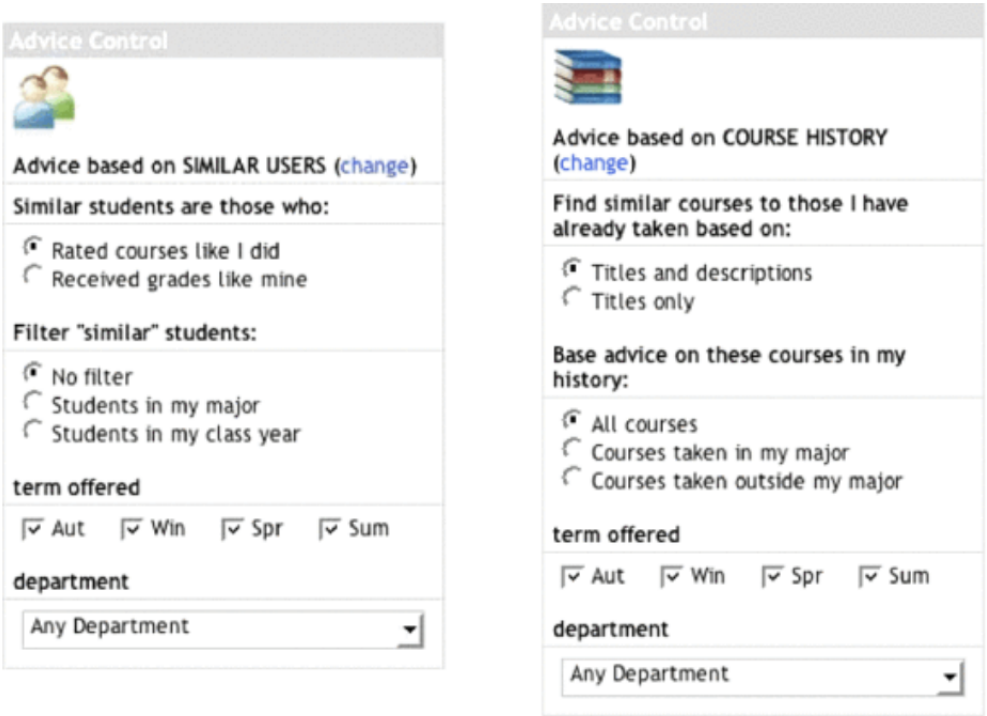

Figure 4: Advice Panel

of entering the workforce [5]. This recommender system first identified the competency gaps between a learner's profile and a job listing posted by a company, and then proposed recommended courses to correct the deficiencies. This approach benefited multiple stakeholders:

- **Students:** Allowed students to choose courses that would put them on track for landing their dream jobs.
- **Educators:** Provide an incentive to offer courses that aligned with the job market.
- **Companies:** Allowed companies to directly communicate skill requirements with both students and educators.

2.4.1 System Architecture

This proposed system used a hybrid strategy to create its ontology. Courses, resumes, job postings, and relevant competencies were expressed in terms of the Word Net semantic thesaurus and adhered to guidelines recommended by the European Qualification Framework. End-users could manually annotate metadata using this pre-defined ontology or rely on the system to automatically annotate the metadata on their behalf. Automatically annotated metadata tended to be less accurate than metadata annotated manually by a domain expert. All terms in the corpus were lemmatized and pre-processed to exclude stop words. The recommendation system used a content-based filtering algorithm to rank courses by computing the semantic similarity between the sentences used in the learner's resume with the company's requirements and then with the course descriptions. This process is illustrated in Figure 5. The recommendations were thus highly specific to the active learner and were not concerned with data on other learners in the system.

As in the 2015 recommendation engine proposed by Stanford University, this recommendation system also provided its learners flexibility in controlling the final output of the recommendation engine. Learners could control whether to give more weight to the course title, course summary or course sections, which ranking algorithm to use, as well as the depth of the search. This allowed the system to provide transparency around the recommendations.

2.4.2 Strengths and Weaknesses

This proposed system was the first of its kind to combine heterogeneous data that benefitted multiple stakeholders when making comparisons. The

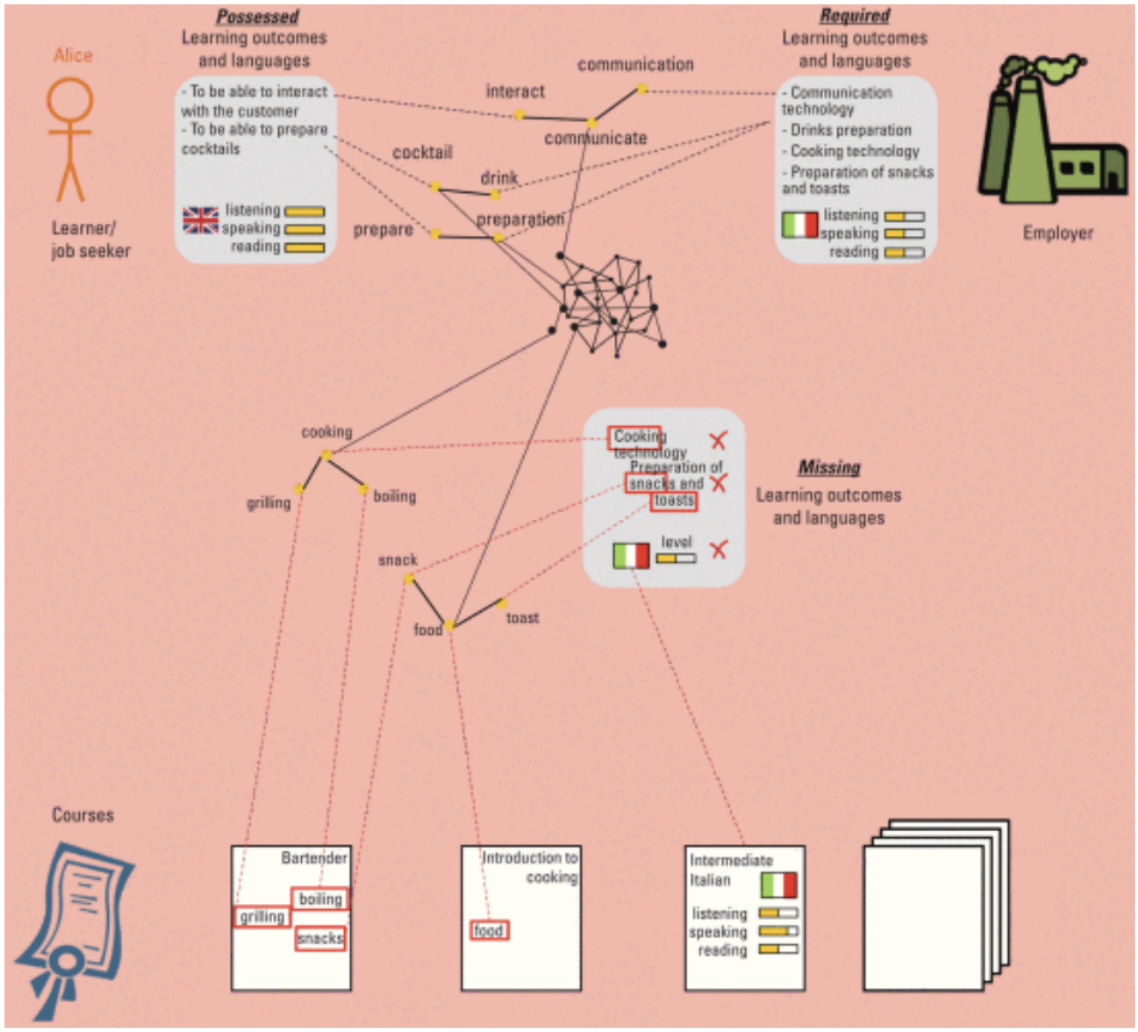

Figure 5: Course Recommendations Based on Competency Gaps

recommendation engine was also transparent in that end-users knew exactly why a set of courses were being recommended to them and had flexibility to further tailor the results. However, this was still susceptible to the cold-start problem where no recommendations could be made if the learner never completed their user profile or if educators or companies never populated their profiles.

2.5 Course Recommendations Using Markov Chains

In 2016, Elham S. Khorasani, Zhao Zhenge and John Champaign of the University of Illinois Springfield proposed a recommender system that considered the sequence of courses students had previously taken when recommending courses for the upcoming semester [6]. In an offline setting, course order tends to play a major role in discussions around course planning. For example, a faculty advisor might suggest taking a course in "data structures" before taking a course in "algorithms" or encourage students to take "algorithms" and "operating systems" in separate semesters because both courses are time-intensive. Khorasani et al. aimed to capture these traditional course sequences in their recommender system.

2.5.1 Data Preprocessing

Khorasani et al. used a dataset from a Canadian research university containing all students who had taken a computer science course at that university between September 2001 and December 2011. The study highlighted a number of data anomalies that were modified or omitted in the data pre-processing phrase:

- **Remove duplicate enrollments:** If a student enrolled in the same course more than once (due to failing the course in a previous semester), only the latest enrollment with the highest grade was retained.
- **Remove infrequent courses:** Courses that appeared less than six times were removed from the dataset on the basis that these courses were either unpopular or cancelled by the university.
- **Removed students with sparse data:** Students with less than two semesters of data were also removed from the data set.

Once the data had been cleaned, it was split into separate training and testing sets. For each student, the current semester's data was put into the training set and the previous semester's data was added to the testing set.

2.5.2 Markov Chains

Khorasani et al. modeled course sequences in their collaborative filtering-based recommender system using Markov chains. A state in this basic Markov model was represented as a set of k courses taken in k consecutive semesters. The transitional probability of going from one state to another was calculating using the Maximum Likelihood Estimation (MLE) formula as depicted in Figure 6. In this equation, the numerator represents the number of students who took c_{k+1} after taking the consecutive courses in k previous semesters and the denominator is the total number of students who took the consecutive courses in k consecutive semesters. Each student maps to several states in the state space because students tend to take multiple courses per semester.

$$p(s_2 = \{c_1, c_2, ..., c_k, c_{k+1}\} | s_1 = \{c_1, c_2, ..., c_k\}) = \frac{count_{st}(\{c_1, c_2, c_3, ...c_k\} \rightarrow c_{k+1})}{count_{st}(\{c_1, c_2, ..., c_k\})}$$

Figure 6: Maximum Likelihood Estimate

This approach calculates the recommendation score $r(s_t,\ c_j,\ j)$ for each course c that a student s_t is likely to take in a semester j, given their enrollments in k previous semesters by adding up all of the transitional probabilities for s_1 where s_1 is the sequence of consecutive courses taken by the student in k previous semesters.

$$r(st, c_j, j) \quad = \sum_{s_1=\{c_{j-1}, c_{j-2}, \ldots, c_{j-k}\}} p(s_1 \cup \{c_j\}|s1)$$

Figure 7: Recommendation Score

2.5.3 Skip Model

One of the problems with the basic Markov chain explained in the previous section is data scarcity. If the set of consecutive courses taken by a student does not match those taken by any other student in the dataset, then the model would not be able to make a recommendation for this student. Khorasani et al. addressed this issue by modifying their model such that recommendations for the next semester k+1 do not depend exclusively on k previous semester, but can also depend on semesters prior to that. This is referred to as a simple skip model. In the skip model approach, weights are assigned to each state to differentiate between built with and without skipping. The more semesters skipped in a state, the less the state should factor into the recommendation. Figure 8 illustrates the new weighted Maximum Likelihood Estimation formula:

$$p(s_1 = \{c_1, c_2, ..., c_k\} \rightarrow s_2 = \{c_1, c_2, ..., c_k, c_{k+1}\}) = \frac{\sum_{st} W(st, \{c_1, c_2, ...c_k, c_{k+1}\})}{\sum_{st} W(st, \{c_1, c_2, ..., c_k\})}$$

Figure 8: Weighted Maximum Likelihood Estimation

In the skip model, the recommendation score is computed as shown in Figure 9. If no semesters were skipped, this model reduces to the basic Markov model.

$$r(st, c_j, j) = \sum_{s_1=\{c_{j-1}, c_{j-2}, \ldots, c_{j-k}\}} W(st, s_1) p(s_1 \rightarrow s_1 \cup \{c_j\})$$

Figure 9: Weighted Recommendation Score

2.5.4 Strengths and Weaknesses

This approach is novel in that it weighs course ordering heavily when considering a recommendation. But course ordering alone is not a good criteria for recommendations because not all students follow a linear path toward graduation. If a student switches majors, for example, their course recommendations would incorrectly recommend courses for their previous academic plan. Secondly, a data scarcity issue exists if a particular pattern has not been before.

CHAPTER 3

Background on Recommender Systems

Recommender systems can be classified into two basic architectures: content-based filtering and collaborative filtering.

- **Content-Based Filtering:** Content-based systems focus on properties of items. Similarity of items is determined by measuring the similarity in their properties.
- **Collaborative Filtering:** Collaborative-filtering systems focus on the relationship between users and items. Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

Hybrid recommender systems combine the two basic approaches [7].

3.1 Content-Based Filtering

Content-based filtering recommends items based on their similarity to content a user has either explicitly or implicitly indicated a preference for in the past. To compute the recommendations, descriptive characteristics about each item, known as features, are captured in an *item profile* data structure. Similarly, users preferences for item features are captured in a *user profile* data structure. Most text mining techniques cannot process text directly. Instead, the text needs to be transformed to a list of numerical values representing characteristics of the text known as a feature vector. This process is known a vectorization. Historically, content-based filtering algorithms have relied on Gerald Salton's vector space model

for ranked retrieval. Under the vector space model, item and user profiles are stored as vectors in the same feature space. Given a user and a set of item vectors, the vector space model measures the degree of similarity between the vectors. The smaller the angle, the more similar the vectors. This is illustrated in Figure 10 [8].

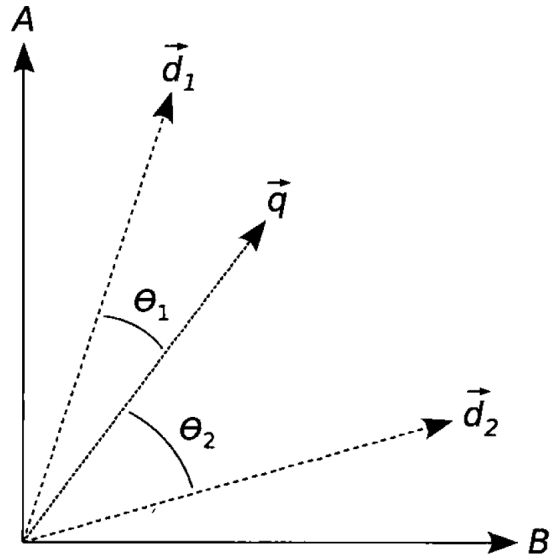

Figure 10: Document similarity under the vector space model.

Content-based filtering works well when it is easy to determine the features of an item. Term frequency inverse document frequency (TF-IDF) is a widely used technique for extracting features from items. Each feature in the item and user profile vectors is represented by its TF-IDF weight. The normalized vectors can then be used to compare the similarity of the item to the user's preferences. Cosine similarity is a popular measure for computing similarities between two vectors in the vector space model. For each item in the collection, this approach takes the dot product of the item and user profile vectors using the formula [8]:

$$sim(\vec{item}, \vec{user}) = \frac{\vec{item}}{|\vec{item}|} \cdot \frac{\vec{user}}{|\vec{user}|} \tag{1}$$

The items with the highest cosine similarity values are returned as

recommendations.

A major advantage of the content-based filtering approach is that recommendations are tailored to users' unique interests. This allows content-based recommenders to avoid the cold-start problem for new and unpopular items. Another advantage is that the logic behind the recommendations can be explained clearly, and users tend to like and feel more confident about recommendations that they perceive as transparent. There can be drawbacks to using content-based filtering in certain situations. The algorithm makes highly specialized recommendations and cannot recommend items outside of a user's preferences. This greatly reduces its scope as a content discovery tool. Furthermore, it is susceptible to the cold-start problem when there is insufficient data about an item or a user has failed to indicate their preferences. Lastly, content-based filtering cannot take advantage of quality assessments made by other users.

3.2 Collaborative Filtering

Collaborative filtering recommends items based on how similar users have rated the item. This paradigm is based on the assumption that users with similar interests in the past will have similar interests in the future. Two major classes of collaborative filtering algorithms exist today: memory-based and model-based.

3.2.1 Memory-Based Filtering

Memory-based collaborative filtering algorithms use the entire database to generate a recommendation. These algorithms use the notion of distance to find a set of users or items, known as neighbors, that are similar to the active user. The preferences of the neighbors are then joined to produce a recommendation for the

active user. The memory-based approach is further categorized into user-based and item-based algorithms.

User-Based Collaborative Filtering: User-based collaborative filtering algorithms recommend items based on similar user's ratings, *ie: users who are similar to you also liked.* This approach is a generalization of the k-nearest neighbor (KNN) algorithm and can be reduced to four simple steps:

1. Compute the similarity between the active user and all other users
2. Sort the results and return the top k users
3. Predict the rating the active user would give to unseen items based on neighbors' ratings
4. Recommend the highest rated items.

There are many ways to quantify distance between two users. The four commonly used similarity measures are euclidean distance, pearson's correlation coefficient, jaccard coefficient and cosine similarity. Once the top k neighbors have been identified, a user-item utility matrix, such a the one shown in Figure 11, is used to predict how the active user would rate unseen items based on an average or weighted average of the ratings for those items provided by their neighbors.

	Course 1	**Course 2**	**Course 3**	**Course 4**
User A	4		5	1
User B	5	4	3	
User C		2	4	5
User D	5			5

Figure 11: A utility matrix representing ratings of courses on a 1-5 scale.

User-based collaborative filtering was a popular algorithm in the early days of recommender systems due to its relative ease of implementation and context independence but its widespread use revealed several problems:

- **Data sparsity:** Recommendations were inaccurate when the system had many items but comparatively few ratings. Similarly, new items would not have enough ratings to show up as recommendations.
- **Scalability:** The more neighbors the algorithm considers, the more computationally expensive it becomes. Furthermore, user profiles change frequently and therefore the entire model needs to be re-computed each time.
- **Cold-start problem:** Not possible to generate accurate recommendations when the system has little to no information about a user. Similarly, it is not possible to generate recommendations for users with unique ratings patterns as there may not be any neighbors they can be compared against.

Item-Based Collaborative Filtering: In 1998, Amazon proposed item-based collaborative filtering to address some of the limitations of the user-based approach [9]. Rather than matching the active user to similar users, item-based algorithms match items the active user has rated to similar items. The algorithm then aggregates and recommends the similar items, *ie: users who liked this item also liked*. Like user-based collaborative filtering, similarity between two items can be calculated using any similarity measure. Item-based algorithms return top k recommendations, but many approaches also simply return all items with a similarity score above a certain threshold. With user-based collaborative filtering, pre-computing the user neighborhood can lead to poor predictions because user similarity is a dynamic measure and changes constantly. Therefore, all

computations must be completed online. Item-based collaborative filtering avoids this problem because item similarity is more static. This allows for pre-computation of the item-item similarity and leads to vast improvements in performance.

3.2.2 Model Based

Memory-based recommender systems use the entire database to make recommendations. These algorithms do not perform well in a real-world setting with a large datasets. Model-based recommendation systems address this issue by extracting a subset of information about users and items to use as a representative 'model' for making recommendations. The reduced dimensionality offers benefits of both speed and scalability. Common models for reducing the dimensionality of a ratings matrix include Bayesian Networks, Clustering Models, and Latent Semantic Models such as Singular Value Decomposition (SVD), Principal Component Analysis (PCA) and Probabilistic Matrix Factorization. The overall goal of these techniques is to uncover latent factors that explain observed ratings. One drawback of the model-based approach is that generalizations often result in lower levels of accuracy than their memory-based counterparts [10].

CHAPTER 4

Keyword Extraction Techniques

Keywords describe the main topics expressed in a document. Previous course recommender systems employed professional curators to manually annotate metadata using relevant keywords. In this section, we will describe statistical techniques for automatic keyword extraction.

4.1 Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF, short for term frequency-inverse document frequency, is a text mining technique to weigh a term according to its importance in a document: the higher the term frequency, the larger its weight will be. At the same time, it weighs the term inverse to its frequency across all documents. That means words such as *the, a, and is* which are likely to show up in multiple documents but are not useful for recommendation are weighed less than words that are more unique to the document. TF-IDF can be computed for a term using the formula shown in Figure 12 [8]:

$$IDF = log(N/N_t),$$

$$TF = \begin{cases} log(f_{t,d}) + 1 & \text{if } f_{t,d} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Figure 18. TF-IDF Formula

Where:

- t is the term
- N is the total number of documents in the collection
- N_t is the number of documents containing t
- $f_{t,d}$ is the number of occurrences of the term t within the document d

Figure 12: TF-IDF Formula.

4.1.1 Strengths and Weaknesses

TF-IDF is a widely used technique due to its simplicity and ease of implementation. Its advantages include that it is unsupervised, domain-independent and language-independent. One of the drawbacks of the TF-IDF technique is that it ignores the conceptual meaning of words, and therefore, it suffers from issues with synonymy and polysemy. As an extension of the bag of words model, the algorithm does not take word order into account and does not strip stopwords; however, these strategies can be employed in the pre-processing phrase prior to TF-IDF calculations. Lastly, TF-IDF weights are typically computed on a per term basis and the algorithm does not take meaningful phrases into consideration.

4.2 TextRank

TextRank is a graph-based ranking model for keyword extraction from text-based documents. The approach builds a graph of word co-occurrences and ranks the importance of individual words using Google's PageRank algorithm.

4.2.1 Candidate Keywords

The first step in the TextRank algorithm is to tokenize a document into basic lexical units. The lexical units represent the vertices that are added to the text graph. Various tokenization strategies can be employed to refine the selection to the most relevant lexical units. In traditional implementations of the TextRank algorithm, it is common to apply stopword lists and syntactic filters, such as Part of Speech Tagging (POS) to remove terms that are not nouns or adjectives. The use of nouns and adjectives follows the observation that human annotators tend to use nouns rather than verbs to summarize documents.

4.2.2 Undirected Weighted Graph of Co-occurrences

Next, all candidate keywords are added to the graph. To avoid excessive growth of the graph size, TextRank only considers single words as candidates for addition to the graph. An edge is added between lexical units that co-occur between a window of N words. In TextRank, the window of co-occurrence is always fixed. To illustrate how this works, when the co-occurence window is 2, no occurrence edge would be created for the sentence "Peter likes pasta" because *likes* is a verb that did not pass the syntactic filter. However, if the co-occurence window changed to 3, then Peter and pasta would become connected. Figure 13 shows an example of a co-occurrence graph created for a small corpus [13].

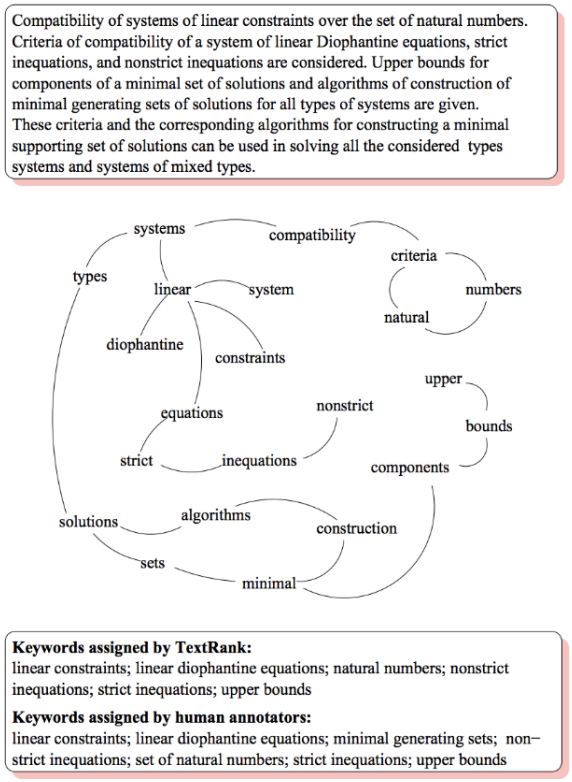

Figure 13: Graph of Co-occurrences

4.2.3 PageRank

After the co-occurrence graph is constructed, the score associated with each vertex is set to an initial value of 1. Then the algorithm goes through the list of nodes and collects the influence of each of its inbound connections. The influence is the value of the sum of the connect vertices summed to determine the new score for the node. Then these scores are normalized, the highest score becomes 1, and the rest are scaled from 0 to 1 based on that value. Each time through the algorithm gets closer to the actual "value" for each node, and it repeats until the values converge - usually for 20-30 iterations. Once a final score is obtained for each vertex in the graph, vertices are sorted in reversed order of their score, and the top T vertices in the ranking are retained for post-processing. By default, T is set to one third of the number of vertices in the graph. During post-processing, all lexical units selected as candidate keywords by the TextRank algorithm are marked in the text, and sequences of adjacent keywords are collapsed into a multi-word keyphrase.

4.2.4 Strengths and Weaknesses

Like TF-IDF, TextRank is an unsupervised, domain-independent, and language independent method for extracting keywords. TextRank's advantage over TF-IDF is that it is able to generate keyword phrases which might have more meaning than individual words. A major drawback of this technique is that it does not have the context of outside word usage and therefore cannot reliably predict the most important words for a specific document.

4.3 Rapid Automatic Keyword Extraction (RAKE)

RAKE, short for Rapid Automatic Keyword Extraction, is an algorithm for extracting keywords from individual documents. The algorithm tries to determine key phrases in a body of text by analyzing the word frequencies and their co-occurrences with other words in the text. The RAKE algorithm is based on the observation that keywords frequently contain multiple words but rarely contain punctuation or stop words such as *the, a, and is*. As input, RAKE accepts a list of stop words, a set of phrase delimiters, and a set of word delimiters. RAKE uses stop words to partition the document into candidate keywords, which are sequences of content words as they occur in the text. Co-occurence of words within the candidate keywords allows the algorithm to gauge a word's meaningfulness.

4.3.1 Candidate Keywords

RAKE generates a list of candidate keywords by splitting the document text into an array of words by the specified word delimiters. This array is then split into sequences of contiguous words at phrase delimiters and stop word positions. The words within a sequence together form a candidate keyword. Figure 14 illustrates an example of a candidate keyword list [11].

Compatibility – systems – linear constraints – set – natural numbers – Criteria – compatibility – system – linear Diophantine equations – strict inequations – nonstrict inequations – Upper bounds – components – minimal set – solutions – algorithms – minimal generating sets – solutions – systems – criteria – corresponding algorithms – constructing – minimal supporting set – solving – systems – systems

Figure 14: Candidate keywords parsed from the sample abstract

4.3.2 Keyword Scores

Using the candidate keywords generated in Figure 14, the RAKE algorithm builds a graph of word co-occurrences across all candidate keywords. This is illustrated in Figure 15 [12].

	algorithms	bounds	compatibility	components	constraints	constructing	corresponding	criteria	diophantine	equations	generating	inequations	linear	minimal	natural	nonstrict	numbers	set	sets	solving	strict	supporting	system	systems	upper
algorithms	2						1																		
bounds		1																							1
compatibility			2																						
components				1																					
constraints					1								1												
constructing						1																			
corresponding	1						1																		
criteria								2																	
diophantine									1	1			1												
equations									1	1			1												
generating											1			1					1						
inequations												2				1					1				
linear					1				1	1			2												
minimal											1			3				2	1			1			
natural															1		1								
nonstrict												1				1									
numbers															1		1								
set														2				3				1			
sets											1			1					1						
solving																				1					
strict												1									1				
supporting														1				1				1			
system																							1		
systems																								4	
upper		1																							1

Figure 15: The word co-occurrence graph for candidate keywords

A score is calculated for each candidate keyword and defined as the sum of its member word scores. The scores for the word co-occurrence graph are shown in Figure 16 [12]. The algorithm allow for flexibility in choosing the metric by which to score the words:

- **Degree of the word, deg(w):** Favors words that occur often and in longer candidate keywords
- **Frequency of the word, freq(w):** Favors words that occur frequently regardless of the number of words with which they co-occur
- **Ratio of degree of word to its frequency, deg(w)/freq(w):** Favors

words that predominantly occur in longer candidate keywords. This is the default metric.

	algorithms	bounds	compatibility	components	constraints	constructing	corresponding	criteria	diophantine	equations	generating	inequations	linear	minimal	natural	nonstrict	numbers	set	sets	solving	strict	supporting	system	systems	upper
deg(w)	3	2	2	1	2	1	2	2	3	3	3	4	5	8	2	2	2	6	3	1	2	3	1	4	2
freq(w)	2	1	2	1	1	1	1	2	1	1	1	2	2	3	1	1	1	3	1	1	1	1	1	4	1
deg(w) / freq(w)	1.5	2	1	1	2	1	2	1	3	3	3	2	2.5	2.7	2	2	2	2	3	1	2	3	1	1	2

Figure 16: Word scores calculated from the word co-occurrence graph

4.3.3 Adjoining Stop Words

It is important to note that RAKE does not blindly omit stop words from its candidate keyword generation. The algorithm accounts for the possibility that two content words could be joined together by one or more stop words, such as "Day of the Dead." If the algorithm finds such keywords adjoined by a stop word, and the pair occurs twice in the same document and in the same order, then a new candidate keyword including the interior stop words is generated. Because of the two occurrence restriction, it is more likely that these combinations will be found in larger documents than smaller ones.

4.3.4 Extracting Keywords

After candidates are scored, the top T candidate keywords with the highest scores are selected as the keywords for the document. T is defined as a third of the number of words in the graph.

4.3.5 Strengths and Weaknesses

Like TF-IDF and TextRank, RAKE is an unsupervised, domain-independent, and language independent method for extracting keywords. RAKE's advantage over

TF-IDF is that it is able to generate keyword phrases which might have more meaning than individual words. It is also a fast and computationally efficient solution. Experiments have shown RAKE is faster than TextRank while achieving higher precision and comparable recall scores [12]. Like TextRank, a major drawback of this technique is that it does not have the context of outside word usage and therefore cannot reliably predict the most important words for a specific document. It also requires generation of a stop word list which need to take consideration of the domain and language of the document being processed.

CHAPTER 5

Strategies for Taming Unstructured Text-Based Data

Document preprocessing is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable in the context of recommender systems. In this section, we will describe various strategies for preprocessing documents with the goal of 1) increasing recommendation accuracy by eliminating noise features, 2) improving computational time and efficiency by decreasing the size of the effective vocabulary.

Normalization is the process of transforming unstructured documents into a more uniform sequence. By transforming the terms in a document to a standard format, subsequent processing will not have to deal with issues that might compromise the recommendations. For example, converting all terms to lowercase simplifies feature comparisons. Text normalization can take many forms: correcting punctuation and capitalization, stemming, lemmatization and stopping. We will describe each of these strategies in the subsequent sections.

5.1 Punctuation and Capitalization

Natural language contains a number of features that create minor tokenization problems. Many of them are related to punctuation and capitalization. Word tokenization may seem simple in a language that separates words by a special 'space' character; however, not every language does this and often, white space alone is not sufficient even for English. For example, "Los Angeles" and "rock 'n' roll" are independent thoughts despite containing spaces and punctuation. Properly accounting for these nuances is critical to maintaining high recommendation

accuracy. Similarly, it is important to take case normalization considerations into account before adjusting the tokenization procedure. Case normalization is the process of converting all characters to a common case, (i.e. upper or lower). While this can improve similarity comparisons in some cases, there is a risk that the meaning of the term is not preserved after the normalization. For example, the acronym "US" and the word "us" can become conflated when case normalization is applied.

5.2 Stemming

It is not uncommon for unstructured documents to use different forms of a word such as *help, helps, and helping*. Stemming is a normalization technique that attempts to reduce each word to its root form by removing the differences between its inflected forms. The root form of a word may not even be a real word. The words *jumping* and *jumpiness* may both stem to *jumpi*. When a stemmer transforms a term into its root form, it is not directly concerned with the linguistic validity of this transformation, but only with the measurable impact on retrieval effectiveness.

Stemming is not an exact science and careful consideration must be given to avoid understemming and overstemming.

- Understemming is the failure to reduce words with the same meaning to the same root. For example, *jumped* and *jumps* may be reduced to *jump*, while *jumping* may be reduced to *jumpi*. Understemming reduces retrieval–relevant documents are not returned.

- Overstemming is the failure to keep two words with distinct meanings separate. For instance, *general* and *generate* may both be stemmed to *gener*.

Overstemming reduces precision–irrelevant documents are returned.

5.3 Lemmatization

Lemmatization, like stemming, tries to group related words, but it goes one step further than stemming in that it tries to reduce words to a lemma, or word in the sense of a dictionary entry. In an unstructured text, the same word may represent two meanings–for example, the word wake can mean to wake up or refer to a funeral. While lemmatization would try to distinguish these two word senses, stemming would incorrectly conflate them. Lemmatization is a much more complicated and expensive process that needs to understand the context in which words appear in order to make decisions about what they mean. In practice, stemming appears to be just as effective as lemmatization, but with a much lower cost.

5.4 Stopping

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words. For a small number of queries, stop words form an essential form of a phrase. "To be or not to be" is a well-known example. Eliminating these stop words would prevent us from discovering the term. The general consensus is that for features that do not consider proximity between terms, stopwords may be eliminated. For features that do consider proximity between terms, particularly to match their occurrence in the phrase, it may be appropriate to retain stop words.

CHAPTER 6

Implementation

6.1 System Architecture

The overall system architecture for the Next Level course recommender system is shown in Figure 17. The architecture can be summarized into four parts:

1. **Application:** Accept user query.
2. **Text Pre-Processing Engine:** Normalize the query.
3. **Learning Engine:** Extract top N skills from context using TF-IDF. Extract the top N skills from nearest cluster using k-means. Combine the lists into a set of relevant skills.
4. **Recommendation Engine:** Compute the similarity score between skills and courses using cosine measure. Recommend top N courses with highest similarity score.

6.2 Technical Stack

The following libraries and frameworks were used in the development of the Next Level application:

Component	Library/Framework
UI	Flask
Keyword Extraction	Scikit-learn, Rake-nltk, Gensim
Text Processing	nltk, pandas, bs4

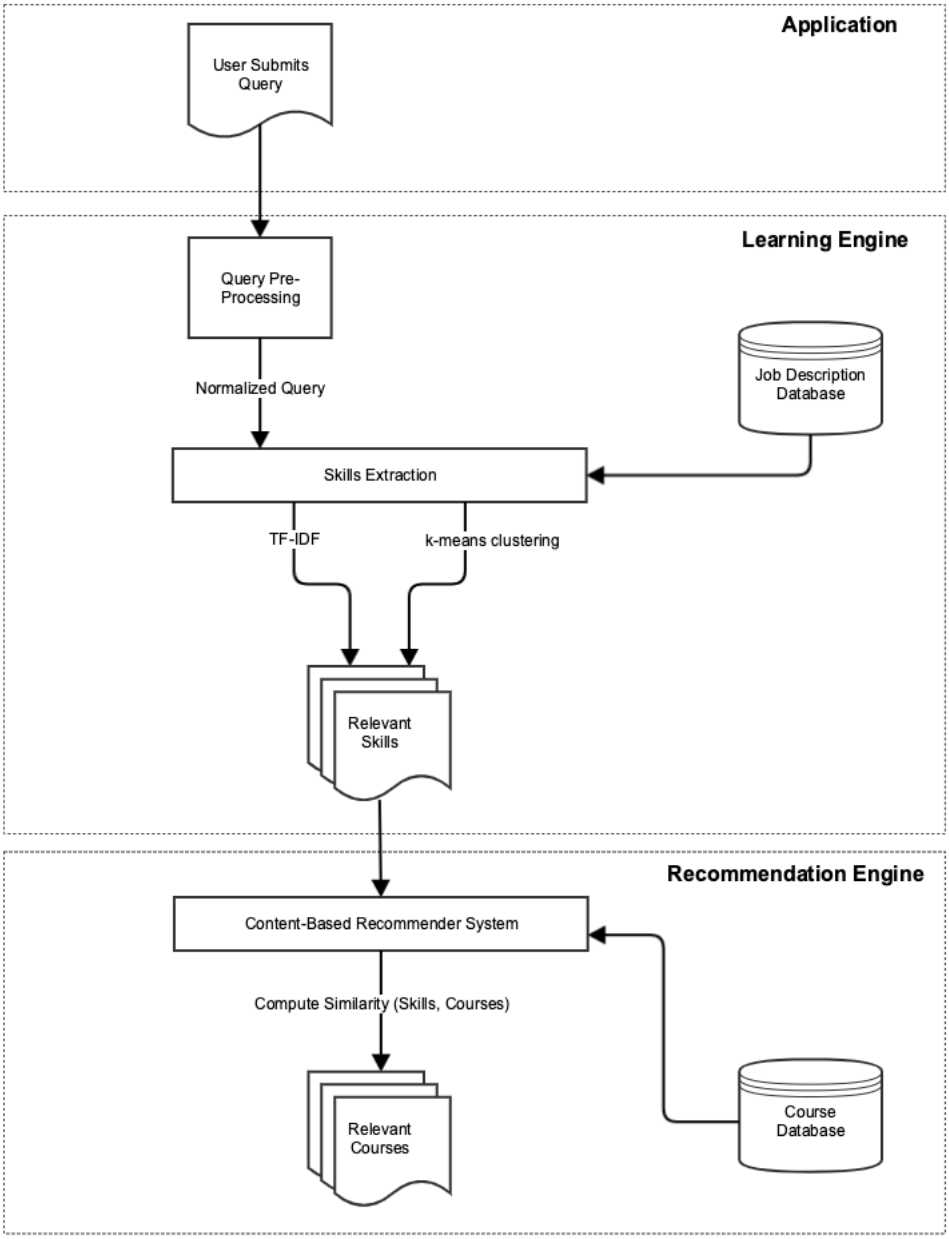

Figure 17: Next Level Architecture Diagram

6.3 Data Preparation

An important prerequisite to the implementation shown in Figure 17 is data preparation. Noisy and unreliable data can greatly hinder knowledge discovery

during the training phrase. To avoid "garbage in, garbage out" and improve data quality and therefore model performance, we executed a series of steps to collect, clean and transform our data for training.

6.3.1 Data Selection

There are two sources of data that we use: the course database (which contains information related to the courses such as name, description, and other metadata), and job description database that contains thousands of job descriptions pertaining to roles in the field of Computer Science.

Course Data: The course database was populated with course data from the 2018-2019 San Jose State University Computer Science Department Course Catalog [14]. The dataset includes both undergraduate and graduate courses. To conduct a controlled experiment, we limited our dataset to courses in the Computer Science Department at San Jose State University. In the future, the scope of the recommender system can be easily expanded to include multiple departments and universities by simply expanding the dataset.

Each course in the course database contains the Course ID, Course Name and Course Description. The descriptions found in the Course Catalog summarize the Course Learning Outcomes (CLO) for each course. CLOs describe the learning that will take place through concise statements, made in specific and measurable terms, of what students will know and/or be able to do as the result of having successfully completed a course. CLOs were selected in favor of course syllabi because their contents do not change between course offerings. We did not want to bias our model with unique keywords only found in a single offering of a course. In addition, CLOs containing sparse descriptions were omitted from our database. In total, 74 courses

were selected for addition into the course database.

Job Description Data: In the absence of a public dataset of job descriptions for roles in the field of Computer Science, we built a web scraper to collect job descriptions from recent job postings on Indeed.com. To generate a list of trending skills in Computer Science, we combined StackOverflow's 2018 Developer Survey [16] results of most popular tools and technologies for professional developers along with Kaggle's 2018 [17] list of most in demand skills for data scientists to form a dataset of 157 top technical skills for Computer Scientists in 2018. Then for each skill, we queried Indeed.com for job postings containing the term. In total, approximately 70 job postings were returned per skill. After removing duplicate postings from the database, we were left with a total of 1,128 unique job descriptions. Once again, for the purposes of conducting a controlled experiment, we limited our dataset to job descriptions written in the English language and pertaining to roles in the field of Computer Science. In the future, the model can be extended to support additional languages and fields of study with the help of domain-specific training data and preprocessing techniques.

6.3.2 Data Preprocessing

After extensive data exploration, including visualization of frequently occurring words and phrases (unigrams, bigrams, trigrams), and cluster analysis to identify outliers in the dataset, the following dimensionality reduction and normalization techniques were applied to the course and job description databases:

- Exclude:
  - HTML tags, links, email addresses
  - Punctuation
  - Excess whitespaces
  - Non-ASCII characters
  - Stop words and stop phrases
  - Verbs, possessive endings and cardinal digits
- Standardized the vocabulary, e.g. oop, object-oriented programming and object-oriented programming were all replaced with "oop"
- Expand contractions
- Lowercase the text
- Lemmatize the text

For parity, the same preprocessing was applied to both courses and job descriptions. To optimize runtime performance, preprocessing of the datasets was handled offline. It should be noted that preprocessing of the course database joined the course name and course description into a single field. This followed the observation that both fields contain unique keywords that could be useful for recommendation purposes. Furthermore, cluster analysis of the job description dataset revealed the presence of non-informative company overviews, compensation details and equal opportunity statements. These blocks of text were removed manually.

6.4 Application

Next Level is a web application built using Python and Flask. The main entry point to the Next Level application is shown in Figure 18. On the client, users can provide either the job description for a role they are interested or directly list skills they are interested in learning. The recommender system is flexible enough to handle either form of input.

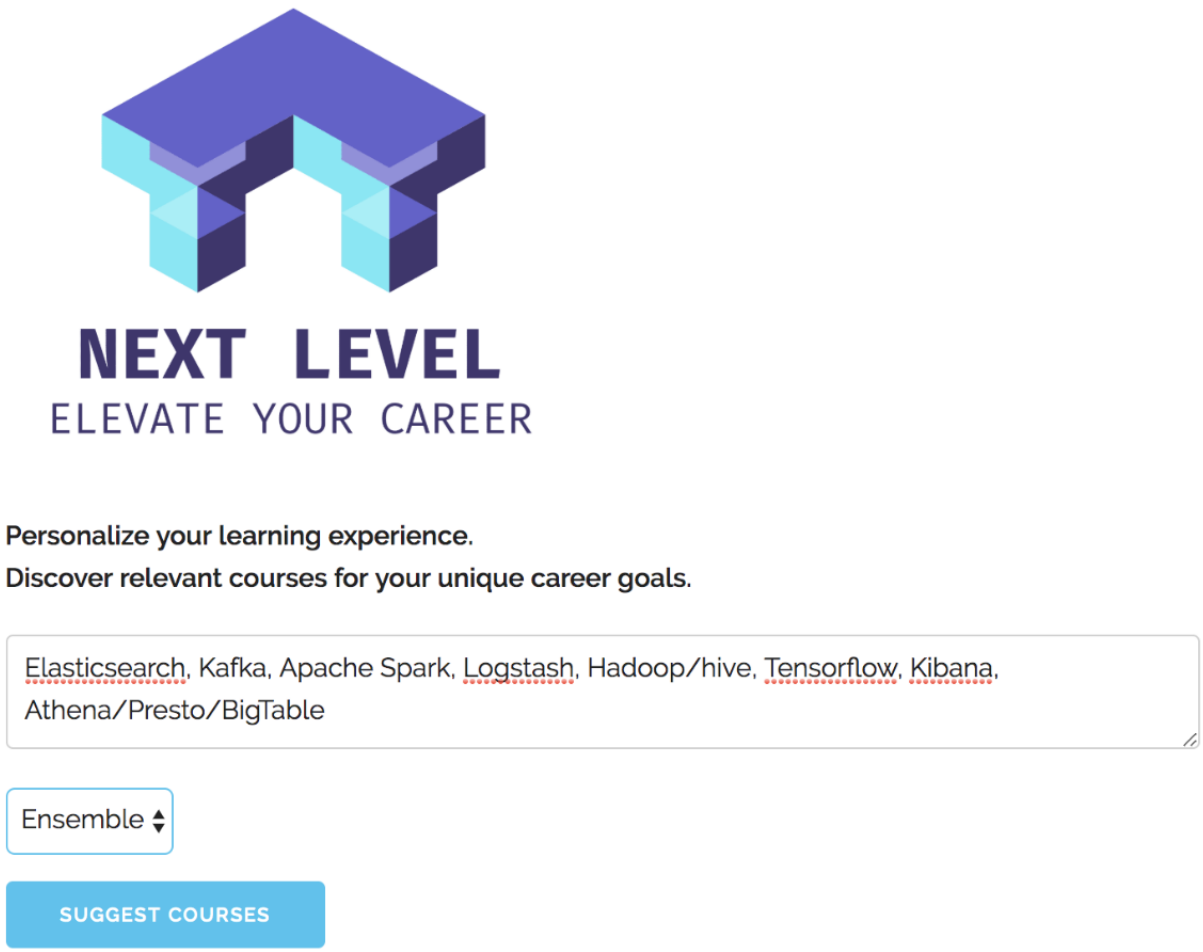

Figure 18: UI Entry Point

When a user submits the form on the client, the form data is sent as a POST request to the server. The server then orchestrates the data through the text

processing, learning and recommendation engines. When the synchronous task orchestration is complete, the server responds back to the client with the lists of suggested skills and courses. An example of the final output can be seen in Figure 19.

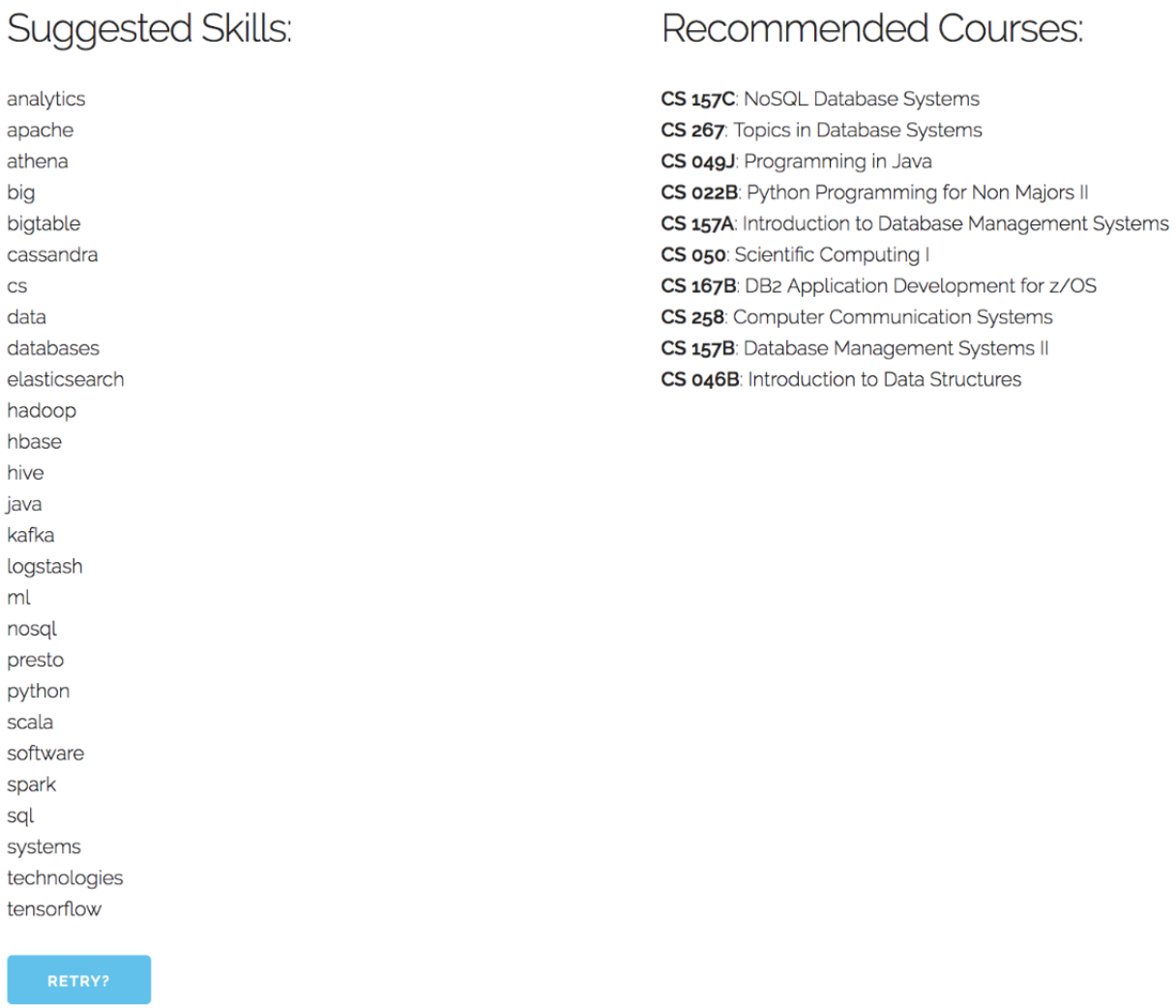

Figure 19: UI Recommendations

6.5 Text Preprocessing Engine

For parity, queries were preprocessed using the same normalization method applied to the training data. This step aims to dynamically standardize the query in order to improve its chances of successful semantic comparison with the course database. Because user queries, courses and job descriptions do not follow a common ontology, step step is instrumental to improving the quality of the recommendations.

6.6 Learning Engine

Unsupervised machine learning algorithms infer patterns from datasets without reference to known, or labeled, outcomes. They can be useful techniques for discovering the underlying structure of the data when labeled data is unavailable. Clustering algorithms, a subclass of unsupervised machine learning, organize unlabeled data into similar groupings known as clusters. Documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters. Next Level uses a two-step ensemble approach to generate a list of relevant skills based on the user's query: k-means clustering and TF-IDF keyword extraction. First, the learning engine uses the k-means clustering algorithm to group the job description training dataset into k unique clusters. Once the k-means algorithm has been run and the groups are defined, any new data can be easily assigned to the correct group. This particular algorithm was selected because it scales well to very large training sets and medium sized clusters [15]. Once the k-means algorithm has identified the cluster for the user query, Next Level stores the top 10 keywords from the cluster in a list. Then, we run TF-IDF keyword extraction to store the top 10 keywords from the user query into a separate list. The two lists are combined into a set of relevant skills

and passed along to the recommendation engine for similarity scoring against the course database.

6.6.1 Step 1: k-means Clustering

The goal of the k-means clustering algorithm is to find groups in the data, with the number of groups represented by the variable k. This algorithm requires the number of clusters to be specified when the model is defined.

Properties of the k-means Algorithm

- There are always k clusters
- There is always at least one item in each cluster
- The clusters are non-hierarchical and they do not overlap
- No approximation guarantees

k-means Algorithm and Implementation

1. Choose the number of clusters, k
2. Select k points at random as the initial centroids (cluster centers)
3. Repeat until the centroids converge (e.g. the cluster assignment has not changed)
    - Assign each sample to its nearest centroid according to the Euclidean distance formula:
    $$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2}$$
    - Find new centroid by taking the mean value of all of the samples assigned to each previous centroid

6.6.2 Step 2: TF-IDF Ensemble

After retrieving the list of related skills using the k-means clustering algorithm, we apply the TF-IDF keyword extraction algorithm, as described in Chapter 4.3, to the input query. The vocabulary of the TFIDF vectorizer are the term tokens from the job description database. The list of skills extracted from both algorithms is combined into a set. A set, by definition, stores only unique skills and therefore any overlap between the two lists is consolidated.

6.7 Recommendation Engine

Next Level uses content-based filtering to recommend courses containing at least one of the relevant skills obtained from the learning engine. As described in Chapter 3, content-based filtering algorithms focus on properties of items. Similarity of items is determined by measuring the similarity in their properties. The central theme of our approach is to use skills as features to represent both users and courses. Under the vector space model, queries as well as courses are represented as vectors in a high-dimensional space in which each vector corresponds to a term in the vocabulary of the collection. Given a query vector and a set of course vectors, one for each course in the collection, we rank the courses by computing the cosine similarity between them:

$$similarity(\vec{skill}, \vec{course}) = \frac{\vec{skill}}{|\vec{skill}|} \cdot \frac{\vec{course}}{|\vec{course}|} \tag{3}$$

The cosine similarity is computed as the dot product of the document and query vectors normalized to unit length. Provided all of the components of the vectors are nonnegative, the value of the similarity score ranges from 0 to 1, with its

value increasing with increasing similarity.

CHAPTER 7

Experiments and Evaluation Metrics

When we speak about the accuracy of machine learning algorithms, we typically refer to a measure of comparing the "true" label to the predicted label. Unsupervised learning algorithms work on "unlabeled" datasets. This means accuracy cannot be directly applied as a measure of evaluation. In the absence of such a metric, we evaluated the quality of our clusters on the basis of silhouette scores and the overall effectiveness of our course recommendations using traditional measures of information retrieval evaluation: precision, recall and F-scores.

7.1 Silhouette Analysis

Silhouette analysis is used to measure the quality of a clustering. The silhouette score can be calculated using the following formula:

$$\frac{b^i - a^i}{max(b^i, a^i))} \tag{4}$$

where a^i is the average distance of all data points in the same cluster and b^i is the average distance from all data points in the closest cluster. This measure has a range of [-1, 1]. Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters and is the ideal value for a clustering algorithm. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters. Negative values indicate that samples might have been assigned to the wrong cluster [18].

Figure 20 shows the results of our silhouette analysis on the job description

dataset for different values of k. This analysis was conduced to both identify an optimal k value for our model and to evaluate the quality of our clustering algorithm. Our experiments found that the silhouette score declines when $k > 900$. Furthermore, we observed that the highest score occurs when k falls between 500 and 800. In our final model, we chose $k = 500$ due to diminishing returns beyond this point.

k	Silhouette Score
5	0.00706743
10	0.0072343
20	0.00828608
30	0.00955933
40	0.01017165
50	0.01564824
100	0.03695071
200	0.06497945
300	0.07773386
400	0.08913854
500	0.09329283
600	0.09973131
700	0.1028957
800	0.10350797
900	0.10320679
1000	0.04438647

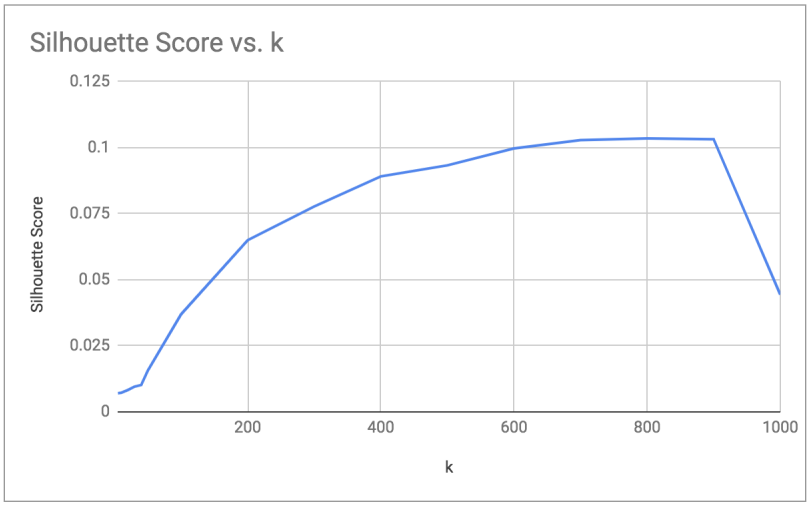

Figure 20: Silhouette Analysis

At $k = 500$, we calculated a silhouette score of 0.09329283035401542. While this score is on the lower end of the spectrum and definitively indicates the need for further dimensionality reduction, which we will discuss in Chapter 8, the negative impact of the suboptimal clustering is mitigated due to the fact that only the top 10 terms in each cluster are used to generate skill recommendations. Moreover, Next Level uses the combination of k-means clustering and TF-IDF feature extraction to determine relevant skills to the user's query; therefore the silhouette score is just an indicator of the quality of the k-means clustering and not the overall quality of the recommendations. In the subsequent sections, we will describe how our overall

recommender system performed against other keyword extraction algorithms.

7.2 Testing Recommendation Quality

To test the quality of our recommendations, we evaluated recommendations generated by Next Level on the basis of precision, recall and F-score.

Precision is the fraction of relevant documents among the documents retrieved by the system. It can be calculated using the formula:

$$Precision = \frac{|relevant\ documents \cap retrieved\ documents|}{|retrieved\ documents|} \tag{5}$$

Recall is the fraction of relevant documents contained in the set. As an effectiveness measure, recall quantifies how exhaustively the search results satisfy the user's information need. It can be calculated using the formula:

$$Recall = \frac{|relevant\ documents \cap retrieved\ documents|}{|relevant\ documents|} \tag{6}$$

F-score is the harmonic mean of precision and recall. It can be calculated using the formula:

$$F - score = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{7}$$

All three scores will be used to benchmark recommendations generated using our ensemble algorithm against recommendations generated using RAKE, TextRank, TF-IDF and k-means.

7.2.1 Experiment

In our experiment, we executed 7 independent queries per method. The queries varied in size from single keyword to full-length job descriptions. We then

computed the precision, recall and F-scores for each query in the context of the top 1, 5 and 10 results. For each algorithm, we averaged the score for each metric across all queries. The final scores are shown in Figure 21.

Our experiments follows the assumption that given a user's information need (career interests), represented as a search query, the course recommendations returned are either relevant or irrelevant with respect to this information need. In this case, a course was considered "relevant" if it contained the specific skill, or a generalization of the skill in its course title or description. For example, if the user queries for "nosql" then a course would be considered relevant if it contained terms such as "nosql" or "sql" or "database" in its title or description. The results of the experiment are shown in Figure 21.

Based on the results of the experiment, it would appear that our ensemble algorithm performed better on average than TextRank, about the same as the k-means and TF-IDF standalone algorithms, and worse than RAKE. However, recommendations made using the ensemble and k-means models essentially take a user's query, transform the results into a new query based on relevant skills, and then compute the semantic similarity between the skill and course vectors. By virtue of being transformations of the original requests, these algorithms will always yield a different set of course recommendations than approaches based strictly on context. This can help explain part of the reason why the scores for the ensemble and k-means approaches are lower than expected. Particularly in the category of precision and F-score. Unfortunately, we could not reliably account for related skills in this experiment as that added a layer of subjectivity to our measure that could not be quantitively measured. In Chapter 8, we will propose possible longer-term experiments that can be conducted to test the overall quality of the

recommendations taking related skills into consideration.

Algorithm	Mean Precision	Mean Recall	Mean F-Score
Ensemble	0.57	0.69	0.6
K-Means	0.56	0.68	0.6
RAKE	0.76	0.73	0.71
TextRank	0.59	0.54	0.54
TF-IDF	0.67	0.63	0.62

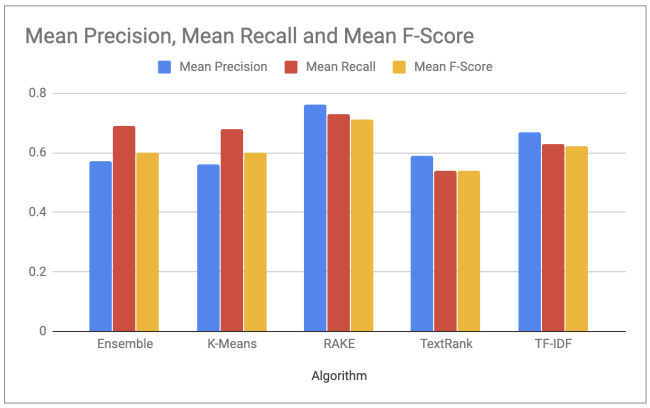

Figure 21: Information Retrieval Evaluation

It should also be noted that because the ensemble method combines TF-IDF and k-means, it returns the same set of documents as the standalone TF-IDF case, but in a different order than if the related skills from the clustering algorithm were not included in the results. This results in low recall@1 and recall@5, but as shown in Figure 22 and in detail in the Appendix, results in high recall@10 as it allows recommendations to include courses containing terms that were not found in the original query but are closely related. Next Level's ensemble approach is better able to identify all relevant courses in the database. This demonstrates a major advantage of using an ensemble approach, as many recommendations would not have been returned if only top skills from the query were used to compute similarity scores.

7.3 Non-Empirical Measures of Effectiveness

Because it is difficult to assess the effectiveness of recommender system quantitatively, select hand-picked anecdotal cases should also be considered.

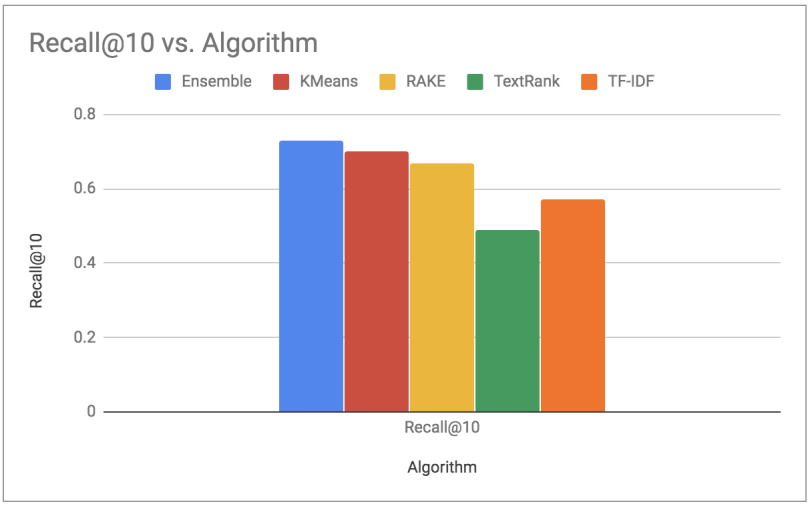

Figure 22: Recall@10

7.3.1 Insufficient Data on Query Terms

Next Level outperforms other algorithms when the course metadata does not match the vocabulary used in the query. To demonstrate, we issued a query for the skills: Elasticsearch, Kafka, Apache Spark, Logstash, Hadoop/hive, Tensorflow, Kibana, Athena/Presto/BigTable. None of the course titles or descriptions in the course database contained these search terms and therefore context-based methods failed to return any results for this query. The ensemble and k-means methods drew from cluster knowledge to recommend a set of relevant skills for which there were matching terms in the course database. The ensemble approach went one step further to also recommended all of the context-based keywords as "relevant skills" to keep the user informed of skills they should attempt to gain based on their original query. In this case, ensemble was the optimal algorithm of choice.

7.3.2 Full Job Descriptions

Both the ensemble and standalone k-means algorithms perform better on average when the query contains a full job description. This is because the larger the query, the less effective traditional keyword extraction becomes at filtering out irrelevant information.

7.3.3 Queries Belonging to Zero or Multiple Clusters

One limitation of the Next Level approach is that k-means always assigns the input query to its closest cluster; however, there are times when the query will not belong to any cluster, or may belong to multiple clusters. In both cases, the quality of the recommendations is adversely impacted. In the future, we could experiment with other clustering algorithms that are better suited to these types of problem. This point will be elaborated on further in Chapter 8.

CHAPTER 8

Conclusion and Future Works

In this paper, we presented a course recommender system that uses content-based filtering and an ensemble learning algorithm of k-means clustering and TF-IDF to suggest relevant skills and courses based on students' career interests. As is the case with most unsupervised machine learning models, quantitatively evaluating the overall effectiveness of the model quantitatively has proven to be difficult. The model is better suited to evaluations that are subjective and domain-specific in nature. Based on generalizations of empirical data, we observed that traditional keyword extraction techniques tend to have higher precision and recall than our ensemble method when the number of query terms is small; however, recall tends to favor the ensemble approach when the number of query terms is high, such as when a job description is entered as the input. The larger the context document, the better the algorithm is at predicting the nearest cluster. This is because the model was training on full job descriptions. Furthermore, recall tends to favor the ensemble algorithm when larger top k results are considered, as the algorithm uses relevant skills to expand the search scope, whereas the other algorithms either exclusively rely on the search terms found in the user's query or generate results based exclusively on relevant skills and do not take into account keywords from context. Our experiments conclude that precision is not a good measure for determining the effectiveness of clustering-based recommendations because clustering algorithms will always return "related skills" and it is not possible to objectively determine if these are relevant based on the user's query. The low precision scores therefore also result in lower F-scores for the

ensemble algorithm.

To answer the broader question of whether or not students' job outcomes improve as a result of using the Next Level recommender system, a longer-term study must be conducted that follows the professional careers of students who built their academic plans around Next Level recommendations. Theoretically, students who use personalized course recommendations to inform their academic planning would obtain employment in relevant fields faster than students who did not take advantage of personalized recommendations.

Taking the subjectivity of the measures out of the equation, the results of our experiments definitively indicate that our k-means is sensitive to outliers and noisy data. Our model would benefit from improved silhouette scores as a result of a smaller feature space. This should, in turn, yield higher precision and recall. To this end, we may consider integrating Principal Component Analysis and Singular Value Decomposition into our model to reduce dimensionality. Another weakness of k-means clustering is that data will always map to one cluster, whereas in a real-world situation, keywords may map to multiple clusters or none at all. In the future, we could consider clustering using DBSCAN or hierarchical clustering which are better suited to this problem, but these would need to be coupled with a dimensionality reduction technique as neither can scale to large volumes of data.

From a functional perspective, there are many ways the Next Level application can be optimized. We could allow users to filter their courses by career level, such as graduate or undergraduate. Furthermore, if we tracked the courses a student has already taken then those could be eliminated from the recommendations. It would be worthwhile from a planning perspective to generate course sequences based on the recommendations. We could also better organize our results so it is more

apparent to the user which recommendations are based off skills extracted from context, and which are based on relevant trending skills in the industry.

In the future, Next Level could be integrated with San Jose State University's existing self-service course planning tool, "My Planner." This tool allows students to plan their courses for an individual term, multiple terms, or for their entire stay at the university. This would allow students to track their progress towards meeting the requirements associated with their career objectives, manage personalized academic plans where requirements are directly linked to course registration, and eliminate the need to access separate systems to track a student's history, grades, academic plans and other information.

In conclusion, Next Level's approach is novel and offers several advantages over prior course recommender systems. This approach turns a simple course recommender into a discovery tool for both relevant skills and courses. It empowers students to make informed decisions about their academic plans and to discover exactly what is expected from them to get the job of their dreams. In addition, Next Level allows educators to gain an understanding of trending skills in the industry, which in turns empowers them with the information needed to either update their CLOs to maximize searchability, or to tailor their course curriculum to better cater to industry demands. Lastly, the training data can be refreshed periodically to ensure it is relevant. Because this is an unsupervised solution, this is a low overhead task that ensures the model maintains relevance over time. This benefits employers as students enter the workforce knowledge of and experience in the skills needed to make an immediate impact in the workforce.

LIST OF REFERENCES

[1] The Real Cost of College: Time and Credits to Degree in California. The Campaign for College Opportunity, July 2014. `realcostofcollegeinca.org/wp-content/uploads/2014/06/Real-Cost-of-College_Full-Report_CSU.pdf`

[2] Hodgins, H. Wayne. "The Future of Learning Objects." Educational Technology, vol. 46, no. 1, 2006, pp. 49?54. JSTOR, JSTOR, `www.jstor.org/stable/44429269`

[3] Tsai, Kun Hua et al. "A Learning Objects Recommendation Model based on the Preference and Ontological Approaches." Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06) (2006): 36-40.

[4] G. Koutrika, B. Bercovitz, R. Ikeda, F. Kaliszan, H. Liou and H. Garcia-Molina, "Flexible Recommendations for Course Planning," 2009 IEEE 25th International Conference on Data Engineering, Shanghai, 2009, pp. 1467-1470.

[5] P. Montuschi, F. Lamberti, V. Gatteschi and C. Demartini, "A Semantic Recommender System for Adaptive Learning," in IT Professional, vol. 17, no. 5, pp. 50-58, Sept.-Oct. 2015.

[6] E. S. Khorasani, Z. Zhenge and J. Champaign, "A Markov chain collaborative filtering model for course enrollment recommendations," 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, 2016, pp. 3484-3490.

[7] Leskovec, Jure, et al. "Recommendation Systems." Mining of Massive Datasets, 2nd ed., Cambridge University Press, Cambridge, 2014, pp. 292?324.

[8] Buttcher, Stefan, et al. Information Retrieval: Implementing and Evaluating Search Engines. MIT, 2016.

[9] G. Linden, B. Smith and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," in IEEE Internet Computing, vol. 7, no. 1, pp. 76-80, Jan.-Feb. 2003.

[10] Bokde, Dheeraj, et al. & Girase, Sheetal & Mukhopadhyay, Debajyoti. "Matrix Factorization Model in Collaborative Filtering Algorithms: A Survey." Procedia Computer Science, 2015, pp.136-146.

[11] "2017 College Student Survey: A Nationally Representative Survey of Currently Enrolled Students." 2017.

[12] Berry, Michael W., and Jacob Kogan. Text Mining: Applications and Theory. John Wiley & Sons, 2010.

[13] Mihalcea, Rada & Tarau, Paul. TextRank: Bringing Order into Text. 2004.

[14] "San Jose State University." Computer Science Department. `info.sjsu.edu/web-dbgen/catalog/departments/CS-courses.html`

[15] Trevino, Andrea. "Introduction to K-Means Clustering." Oracle DataScience.com. `www.datascience.com/blog/K-means-clustering`

[16] "Stack Overflow Developer Survey 2018." Stack Overflow. `insights.stackoverflow.com/survey/2018`

[17] "The Most in Demand Skills for Data Scientists." Kaggle. `www.kaggle.com/discdiver/the-most-in-demand-skills-for-data-scientists`

[18] "Selecting the Number of Clusters with Silhouette Analysis on KMeans Clustering." Scikit. `scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html`

APPENDIX

Experiment Details

A.1 Detailed Recall@10

Recall@10 was computed for 7 queries per algorithm. Queries varied in length from a single keyword to full job descriptions.

Recall@10: Ensemble	
Query 1	0.4
Query 2	1
Query 3	1
Query 4	0.8
Query 5	0.8
Query 6	0.8
Query 7	0.3
Sum:	**0.73**

Figure A.23: Recall@10 Ensemble

Recall@10: KMeans	
Query 1	0.4
Query 2	1
Query 3	1
Query 4	0.8
Query 5	0.6
Query 6	0.8
Query 7	0.3
Sum:	**0.7**

Figure A.24: Recall@10 K-Means

Recall@10: RAKE	
Query 1	0.2
Query 2	1
Query 3	1
Query 4	0.8
Query 5	1
Query 6	0.7
Query 7	0
Sum:	**0.67**

Figure A.25: Recall@10 RAKE

Recall@10: TextRank	
Query 1	0.2
Query 2	1
Query 3	1
Query 4	0
Query 5	0.6
Query 6	0.6
Query 7	0
Sum:	**0.49**

Figure A.26: Recall@10 TextRank

Recall@10: TF-IDF	
Query 1	0.2
Query 2	1
Query 3	1
Query 4	0.8
Query 5	0.4
Query 6	0.6
Query 7	0
Sum:	**0.57**

Figure A.27: Recall@10 TF-IDF