University of Glasgow | School of Computing Science

Representing Jobs Using Auto-Generated Skill Tags

Gregor Thomson - 2029108t

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

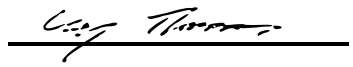
Level 4 Project — October 18, 2016

Abstract

This is a collaborative project with Amazon Development Centre to investigate whether auto-generated tags can accurately represent a job to improve job recommendations. This paper will investigate possible techniques to generate skill tags for a given job description to enhance job recommendations, searching capabilities and user experience. This work will discuss the steps taken to implement a web application which can generate skill tags for a given job description and also the research into ways which we can identify relations between skills and job descriptions. Additionally, the paper explores the difficulties encountered in implementing and evaluation this system. This work is hoped to identify which technologies and techniques are required to create an optimal ranking of skills for a large amount of job descriptions.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: GREGOR THOMSON Signature: 

Contents

Chapter 1

Introduction

Recommendation systems have become increasingly popular in recent years and are used in various areas ranging from Spotify's music recommendations[39] to Netflix's movie recommendations[15]. Recommendation system are software tools and techniques which provide tailored suggestions for the individual user[42][9]. Recommendation systems are primarily used in situations when the user is given an overwhelming number of alternative items[41] and in the era of big data they have become increasingly popular.

JobFinder, the Amazon internal job site have recently introduced job recommendations and are investigating ways to improve their recommendation system. They currently take location, job preferences and skills into consideration when finding matches. However, they use a keyword search for matching users skills to jobs which is computationally expensive and does not necessarily find all jobs which require that skill. Due to the poor results of this technique, they down-weighted the jobs which are matched by skills alone. This is a collaborative project with Amazon Development Centre to investigating whether auto-generated tags can accurately represent jobs to aid job recommendations.

1.1 Motivation

Amazon are wanting to add skill tags to all of their current and future jobs to improve job data and their recommendation system. By introducing this, they hope to speed up their searching capabilities, improve the quality of matches and enhance job cards (Figure 1.1) to provide more initial information about the job to the user. Currently, job cards contain the title, location, job id and hiring manager for that job (Figure 1.1). This does not provide much meaningful information about the job. For example, if someone searches for a software development job the only thing to distinguish the jobs, without reading the description, is the location and hiring manager. By introducing skill tags it will give the user some initial characteristics of that job (Figure 1.2). This will allow the user to easily distinguish between jobs and find the job they are looking for quicker. The introduction of skill tags will also allow the enhancement of job recommendation as they can now use these standardised tags to search for skills in a much more efficient and elegant manner. By having standardised tags this will help the search engine by increasing coverage and speed up process time.

Figure 1.1: Current job card

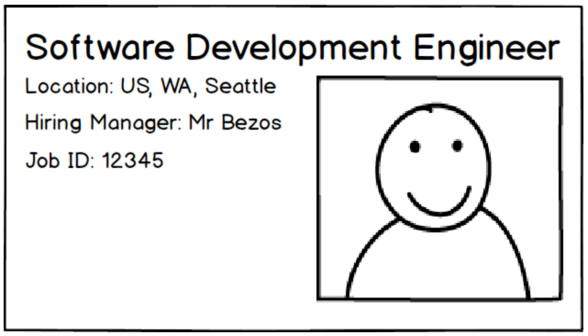
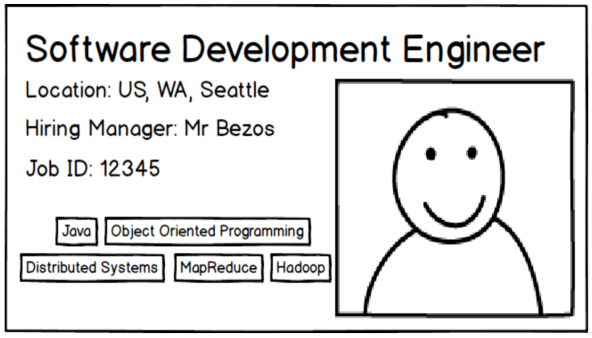


Figure 1.2: Possible job card with skill tags



Some of the problems Amazon are facing which adding skills tags could improve includes searching capabilities, recommendations, and poor job data. JobFinder's search can be improved and made faster from the introduction of tags. If a user searches for a job with a particular skill, it currently does a keyword matching through each job description. If each job had standardised skill tags, the search engine could use these to increase coverage and speed up process time which will subsequently decrease the time a user spends searching for a job they are looking for and improves the overall user experience.

By implementing a system which generates skill tags for each skill will improve job data held by Amazon. It will be designed to extend the data Amazon currently holds by generating the most relevant skill tags for each job description. The addition of tags to job data gives lots of possibilities for other systems - For example Alexa[49], Alexa will be able to summarise a job using the tags alone and without reading out the entire job description.

1.2 Aim

The objective of this project is to investigating whether auto-generated tags can accurately represent jobs to aid job recommendations. However, this can be split up into two key objectives. One being the implementation of an application which can generate tags for a given job description and the second being research into ways which we can identify relations between skills and job descriptions. The tags will be used to match user profiles with jobs, these skill tags should not be too specific that users have very few recommendations but equally, not too general that the user matches with most jobs. This work will investigate alternative technologies and techniques to produce the optimal ranking for a large number of job descriptions. To create this application the following key problems must be addressed:

- Obtain or create a skill set which includes a broad range of skills.

- Implement a web application which will recommend skills for an inputted job description.
- Look at alternative technologies and techniques to enhance the system's results.
- Create multiple ranking algorithms to compare performances of each technique.
- Obtain evaluation techniques to compare and analyse performance of each ranking algorithm.

1.3 Outline

In this work, I look at the the steps taken to investigate whether auto-generated skill tags can accurately represent jobs. This included constructing a skill ontology, implementing a suitable web application and user interface, implementing multiple ranking algorithms and performing the evaluation to identify which algorithm produced the best ranking. I will also present the alternatives I looked at, why I decided on the choices I made, and the issues I encountered. The remainder of this document is divided into the following chapters:

Chapter 2 - **Background and Related Work**
This chapter discusses current related systems and background knowledge needed.
Chapter 3 - **Requirements Gathering**
This chapter discusses the project's requirements and the variety of ways in which they were elicited.
Chapter 4 - **Design**
This chapter discusses the design of the application, presenting an overview of the overall system structure, the coding tools used, as well as an in-depth account of the design decisions.
Chapter 5 - **Implementation**
This chapter discusses the implementation of the system in detail, explanations of each ranking algorithm and the issues encountered.
Chapter 6 - **Evaluation**
This chapter goes into detail how the system was tested and evaluated, the techniques used and how the system meets the requirements.
Chapter 7 - **Conclusion**
This chapter summarises the report and suggests some ideas for future work.

Chapter 2

Background and Related Work

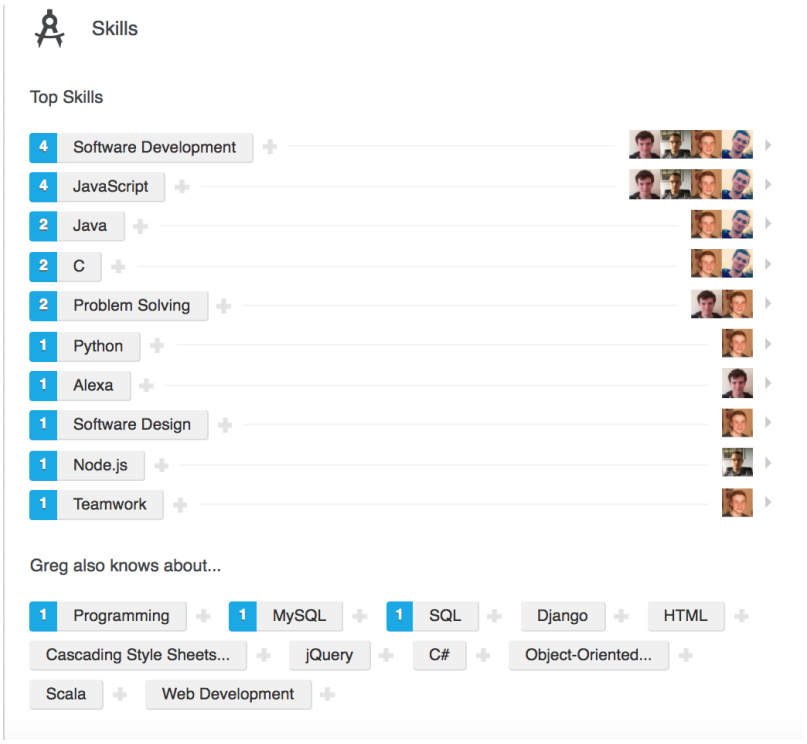
This chapter discusses current related systems and background knowledge needed.

2.1 Current Systems

2.1.1 LinkedIn

LinkedIn is the world's largest professional online social network[7], which allows their users to create profiles that represent their professional identity. "Skills and Expertise" is a feature on LinkedIn allowing their users to tag themselves with particular skills and topics which represents their expertise. The example below (Figure 2.1) shows an example of the skills and expertise feature on a user profile. LinkedIn have said that "Standardized entities help members be better found by increasing the coverage in search engines"[7] was the idea of the feature.

Figure 2.1: Example of skills and expertise feature. Taken from LinkedIn.com[23]



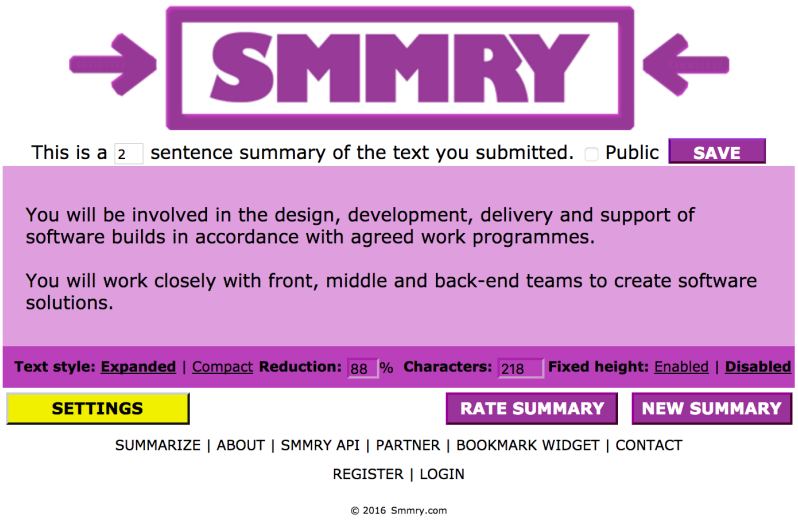
They have had recent research into work which is related to this project by creating a skills folksonomy and identifying the issues they encountered. The creation of the folksonomy was done by using users profiles directly. User profiles already had a free text skill input feature in place which they used to create their folksonomy. LinkedIn used the inputted free text skills as a starting point for their folksonomy and used multiple techniques to remove duplications, such as using Wikipedia to identify if two skill entries were the same. If two skills mapped to the same Wikipedia article then they were either duplicates or have overlap, and they then joint those skills to refine their folksonomy. This was useful to identify some of the problems which may occur when creating an ontology but this could not be used for this project for copyright reasons and that it does not just contain skills but also expertise.

2.1.2 Smmry

Smmry[45] is a website and API which summarizes a given block of text by reducing the text to only the most important sentences. It does this by using an algorithm to rank importance of sentences, reorganising the summary to focus on a topic, and removing transition phrases, unnecessary clauses and excessive examples. However, the core algorithm is made up of six key stages:

1. **Stemming** to associate words with their grammatical counterparts. (e.g. ”city” and ”cities”)
2. **Term-frequencies** to identify most occurring words.
3. **Scoring** each word based on their popularity.
4. **Identify sentences**, not all full stops represents an end of a sentence. (e.g ”Mr.” is not an end of a sentence)
5. **Rank** sentences by the sum of their words’ points.
6. **Return X** of the most highly ranked sentences in chronological order.

Figure 2.2: Example output from Smmry[45] using a software engineering job.



This system is similar to this work as it has a similar goal of summarising a block of text, however it could not be used for this project as it could not be used to aid the recommendation system as we need standardised tags to match user profiles and jobs.

2.2 Background Topics

2.2.1 Word2Vec Word Embeddings

Word2vec[13] is a two-layer neural net that processes text. It is trained to learn the similarities between words. It takes a large corpus as its input and creates a vector space where each word is assigned to a vector. A word embedding $W : words- > \mathbb{R}^n$ is a parameterized function mapping words to vectors (200 to 500 dimensions)[38]. For example:

$$W(\text{"}java\text{"}) = (0.2, -0.4, 0.7, ...)$$
$$W(\text{"}software\text{"}) = (0.2, -0.43, 0.6, ...)$$

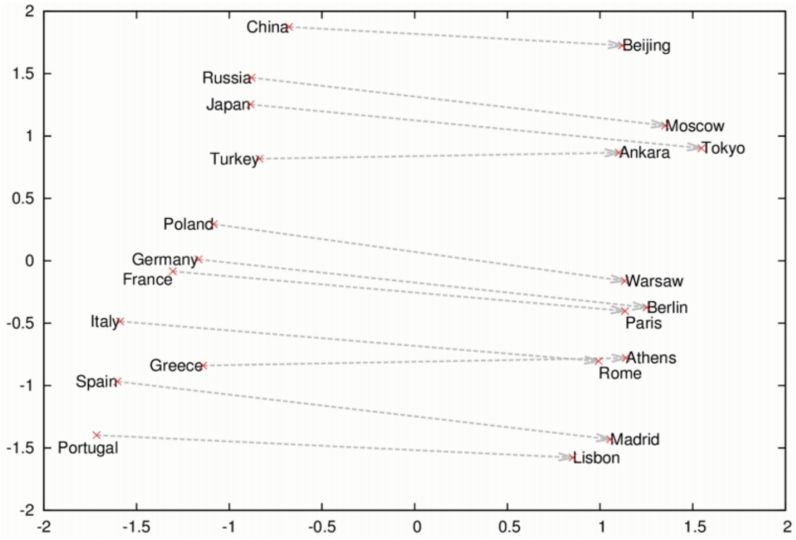
Words which share context within the corpus will have word vectors located in close proximity to one another in the vector space[33]. This gives a mathematical similarity between words and it can give an accurate guess of the association between words. For example, 'king' is to 'man' what 'queen' is to 'woman'. From the example below (Figure 2.3), you can see how a word has a mathematical associations with all other words. We can calculate the cosine distance between each vector to find the most relevant words to a given word.

Figure 2.3: words associated with 'Sweden' using Word2Vec. Image taken from the Word2Vec documentation[13]

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

The vectors used to represent words are called neural word embeddings. Word embeddings is a collective name for a set of language modelling and feature learning techniques in natural language processing(NLP) where words or phrases are mapped to vectors. Not only does related words grouped near each other, but their distances in vector space has a meaning. From the example below (Figure 2.4), you can see that countries and their capitals have a similar association.

Figure 2.4: Association between words in Word2Vec. Image taken from the Word2Vec documentation[13]



Word2Vec word embeddings will be discussed later in this work (Section 5.1.2), where it will be used in the creation of the ranking algorithms.

2.2.2 Wikipedia Pageviews

Wikipedia pageview[52] is a RESTful API which allows one to see how many visits a Wikipedia article has had during a given time period. The tool can be used to find the page view trends of specific articles or projects; the most viewed article of a project or timespan; you can also filter by agent type or access method.

To access the page views of a specific Wikipedia article the following request is made:

```
https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/
{access}/{agent}/{article}/{granularity}/{start}/{end}
```

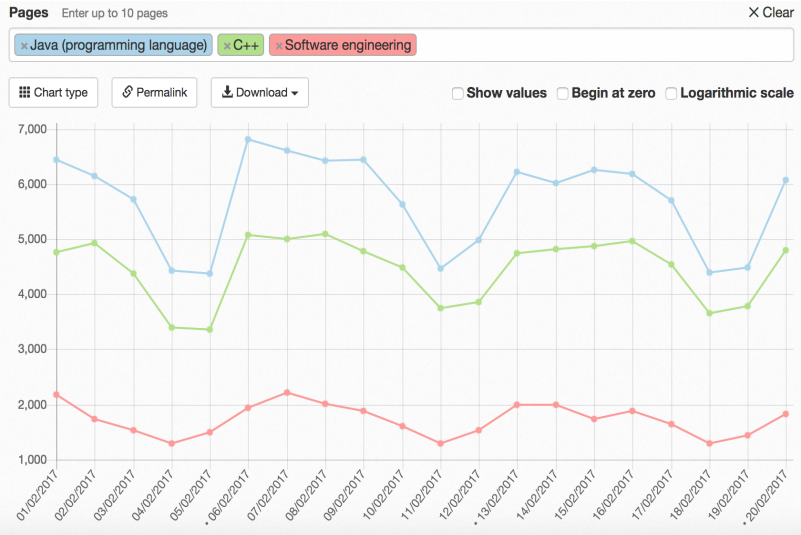
- **Project**: Defines the wikipedia project type. E.g en.wikipedia, de.wikipedia.
- **Access**: Defines the access method. E.g. mobile-web.
- **Agent**: Defines the agent type. E.g. user, spider.
- **Article**: Defines the article name. E.g. Java_(programming_language)
- **Granularity**: Defines the level of detail wanted for the return data. E.g. daily, monthly
- **Start**: Defines the starting time for the request
- **End**: Defines the end time for the request

For this project, Wikipedia Pageviews were used to find recent trends for each skill to find the most searched for skills. An example request to achieve this was:

```
https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-ac
```

This example returns a JSON response with the monthly views for the Java programming language Wikipedia article for each month of 2016. The example below (Figure 2.5) shows an example of skill trends using Wikipedia Pageviews.

Figure 2.5: Example of Wikipedia pageview. Taken from tools.wmflabs.org/pageviews

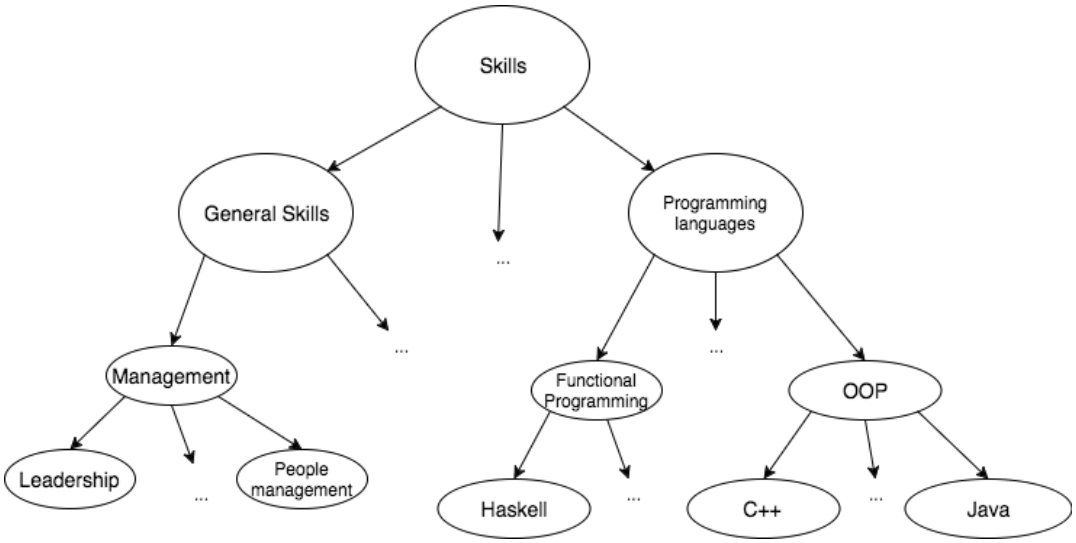


Wikipedia Pageviews are discussed later in this work (Section 5.1.2), where it is used to improve rankings by finding the most searching for skills.

2.2.3 Skill Ontology

An ontology is a formal representations of a set of concepts within a domain and the relationships between those concepts[17].Thomas Gruber, an AI specialist at Stanford University, says an ontology in the context of database systems can ”be viewed as a level of abstraction of data models, analogous to hierarchical and relational models, but intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals.”[18]. Later in this work I will discuss the creation of a skill ontology (Section 5.1.1) where the goal was to create a skill ontology where each skill had a hierarchical relationship with each other. Below (Figure 2.6) is an a small subsection of the ontology to give some insight on how the skill ontology will be structured. The below diagram shows how the top level skills are very general whereas further down the ontology you go more specific the skills get.

Figure 2.6: Example of a skill ontology



Chapter 3

Requirements Gathering

This chapter discusses the project's requirements and the variety of ways in which they were elicited.

3.1 Requirement Elicitation

3.1.1 Amazon Meeting

I had multiple meetings with the Talent Management team, who are responsible for internal HR systems in Amazon. From these meetings I gathered some of the main problems they are having which auto-generated tags could solve, how they will use the tags and some of the problems I may encounter. This gave me a bases of the project's requirements.

The requirements which were identified from these meetings were:

- Create a web application which generate skill tags for a given job description;
- find most relevant skills in job description;
- create a skill ontology of most common skills;
- start with Software Engineering jobs only;
- gather common words and phrases for each skill;
- parse job description to find skills which match;
- the skills used should be able to better recommendation matches;
- compare multiple techniques and approaches when implementing the ranking algorithm;
- implement and compare multiple ranking algorithms;
- and obtain sample jobs to test the system against.

There were also some stretch goals gathered:

- The ability to dismiss suggested skills;

- record dismissals for each skill tag;
- create a machine learning feedback loop which learns from user's skill dismissals;
- user profile page where they can input their preferences;
- and suggest jobs based on user's skill preferences.

Throughout the implementation of the system I had multiple meetings with the team again for requirement analysis and negotiation. This gave me the opportunity to refine my requirements, priorities key features and ensure the project was going in the right direction.

3.1.2 Previous Work

When setting out requirements for this project, it was useful to look at recent projects and research with similar functionality. LinkedIn has done some recent research[7] into similar work where they created a skills folksonomy of 'skills and expertise' and implementing a recommendation system for skills (Section 2.1.1). The similarity between these two projects is that both need to define a set of skills which will be used to aid a recommendation system. The research paper[7] gave some insight on how they created their skills folksonomy and some of the main problems they encountered. This gave me some insight of some of the problems I may face and how large a project creating a skills ontology is alone. From analysing this research paper it was clear that I should not spend a large amount of time creating the skill ontology as creating an optimal ontology would be a project of its own and the objective of this project it to investigate auto-generated tags to represent jobs. It also gave me some ideas of how to overcome some of the challenges they encountered.

Requirements gathered from previous work:

- Merge duplicate skills. Some of the skill tags may have the same underlying skill - these should be merged into one single skill;
- and require Wikipedia pages for each skill to allow the users to have a better understanding of what each skill is.

Another system which has similarities with this work is Smmry (Section 2.1.2). Smmry is a website and API which summarises a given block of text by returning the most important sentences. This project has similarities with this work as both of our goals are to summarise a block of text but the difference is that this project is to use tags rather than sentences to summarise the text. Smmry was a useful project to look at as it produces some very good results using simple techniques such as term frequency and stemming. It was also useful to look at the multiple optimisation stages they took to create optimal results. This shows that multiple iterations of improvements are needed when creating an information retrieval system.

3.2 Functional Requirements

Functional requirements are used to outline the behaviour of the system. The MoSCoW[4] method was used to document the requirements. MoSCoW is a prioritisation technique which groups requirements in terms of importance to the system by categorising them as 'must have', 'should have', 'could have' and 'would be nice to have'[46][51].

Must have

The following requirements must be satisfied in the final system.

Requirement	Description
Web application	A web application which generates skill tags for a given job description and gives some assurance that the skills generated are correct
Skill Ontology	Gather skills from multiple sources to create a skill ontology
Represent skills to match job descriptions	Represent each skill which allows the system to match it to a given job description.
Rank skills	Implement a ranking algorithm to find the top five related skills to a given job description. Each skill must have a score for a given job description.
Implement multiple ranking algorithms	Compare multiple techniques and approaches when implementing the ranking algorithm.
Basic Software Engineering Skills	Start by limiting skills to Software Engineering jobs.
Obtain sample jobs for testing	Gather sample job descriptions from multiple sites to use for testing.
Parse job description	The website should be able to parse a given job description to match the job to skills.

Could have

The following requirements are considered desirable but not necessary.

Requirement	Description
Dismiss suggested skills	Allow users to dismiss suggested skills to get the next most relevant skill.
Link skills to Wikipedia page	Have a link for each skill to give the user a better understanding what it is.
Merge duplicate skills	Merge skills which are the same underlying skill.

Should have

The following requirements are functionality the system should have, if possible.

Requirement	Description
Match common phrases to skills	Find common phrases which are used to identify each skill.
Use skills which will aid recommendation system	The skills used should be able to better recommendation matches.
Extend to all jobs	Extend skill ontology to all job types.
Encapsulate data from users	Users should not be able to see the data held in the ontology but should be able to query it to retrieve relevant skills.

Would be nice to have

The following requirements will not be included in the release but may be included through future work.

Requirement	Description
User profile page	Create a user profile page where they can input their preferences.
Suggest jobs	Suggest jobs based on users preferences.
Record feedback from users on generated tags	Keep a record of user feedback to be used in a feedback loop.
Feedback the user input of recommended tags (Machine learning)	Create a machine learning feedback loop which learns from users skill dismissals.

3.3 Non-Functional Requirements

Non-functional requirements are not features of the system but denote the specific criteria that can be used to judge the operation of the system.

Requirement	Description
Ease of use	The web application should be usable by untrained users and the results should be clear.
Modular	The system should be easily extended with new or updated ranking algorithms.
Efficient	The ranking algorithms used in the web application should not be computationally expensive.
Portability	Should work on all common browsers.

Chapter 4

Design

This chapter discusses the design of the application, presenting an overview of the overall system structure, the coding tools used, as well as an in-depth account of the design decisions.

4.1 System Architecture

When choosing an appropriate architecture for the system it was clear that there was going to be four main parts of the system, the website logic, the ranking algorithms, the skill ontology, and the user interface. For this prototype system, the ranking algorithms will be frequently updated and new ones added (Section 3.2) so it was particularly important that this part of the system was modular to ensure the ease of modifications and extension. Hence, a modbel view controller(MVC) architecture[43] would be an appropriate architecture as it is designed to separate the concerns.

The MVC architecture separates the system into three main components; the model for defining data structures and methods to interact with that data; the view which manages the user interface; and the controller which is the logic of the system[44]. The main benefit of this architecture is that it provides a separation of concerns. This is an appropriate architecture for this project because of its modularity it is easier to add and modify ranking algorithms. It was key for the project to be able to compare multiple ranking algorithms(Section 3.2) which means having the ability to easily make modifications and add new algorithms.

The data in the models should be encapsulated from the user as the skill ontology will be created offline and should only be read as discussed in the requirements (Section 3.2). The user should have no knowledge of how the data is stored or structured. Hence, this should be separated from the rest of the system and be a stand-alone entity only used to query. This prevents users from changing the ontology and makes it easily maintainable if a change is needed for the ontology. The separation from the rest of the system highly simplifies the process of swapping out the ontology for an extended skill set. This will be beneficial when the system is extended to all job types(Section 3.2), it will mean that the system will simply have to point to the updated skill ontology model.

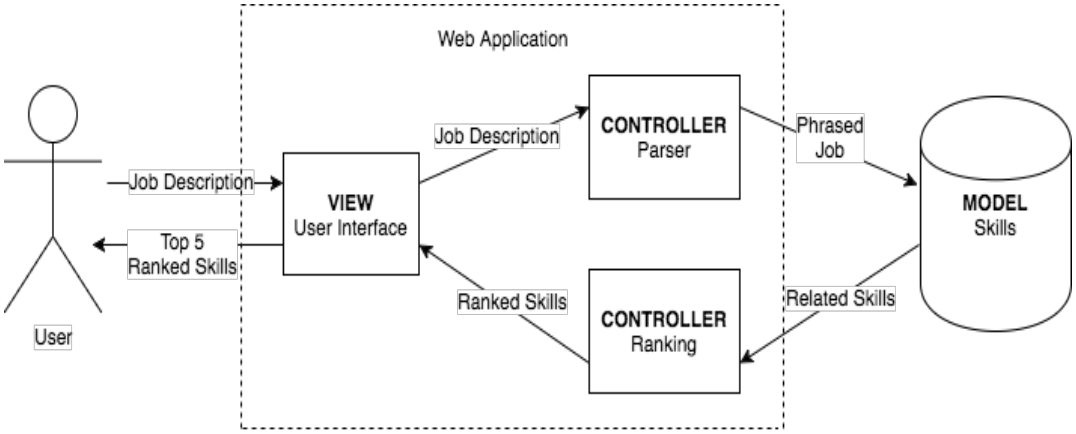
The following diagram (Figure 4.1) is a high-level view of the architecture. As noted in the below diagram the architecture is a slight variation of the normal MVC model as the data models do not directly update the views, however the controller is responsible for returning the information to the user interface. This is called the mediated MVC pattern[8] where the controller acts as a data model proxy for the view. This architecture is common for database applications as it allows the model to encapsulate the raw data[8]. Using this variation of the MVC it means when skills are requested from the models they are passed back to the controller to be scored and ranked before being passed to the views. This means the view will have no knowledge of the underlying data

model and therefore protecting the skill ontology from modification from users.

Figure 4.1: High level view of the architecture

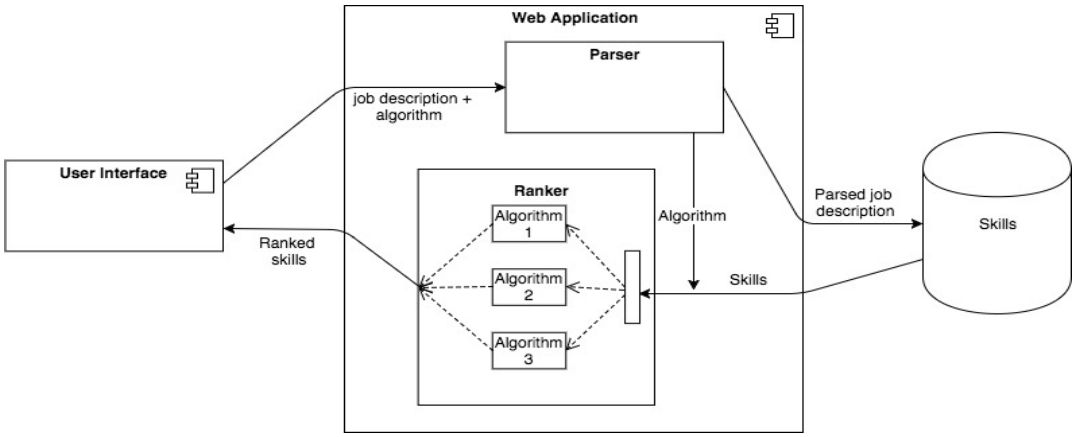
The next diagram (Figure 4.2) is a more detailed architecture diagram where it shows the flow of data. The user interface is the view of the MVC architecture where it takes the user input and outputs the results. The controller section of the MVC architecture is made up of the parser and ranker. The parser removes stopwords, splits the description into words and phrases, and passes them to the models to be matched with skills. These skills are then passed to the ranker where it allocates a score to each skill and sorts. They are then passed back to the user interface with some other metadata to be presented to the user. The view presents the top five skills to the user, which means that only the top five skills are needed to be sent to the view. This helps to minimise the network usage and speeds up the overall loading time for the results.

Figure 4.2: Detailed architecture



The final diagram (Figure 4.3) shows the interaction taken to decide which algorithm to use. When the user submits a job description they will also have to select which algorithm they would like to use. Both the job description and the algorithm identifier is passed to the web application where the parser parses the job description, passes the parsed job description to the skills model and the algorithm identifier to the ranker. When the skills are returned from the skill model it uses whichever algorithm which was passed from the parser to produce the ranking.

Figure 4.3: Multiple algorithms



4.2 Coding tools

In order to implement the system, an appropriate programming language and web framework must be selected. The website itself will not be a large application, it will only have a few pages with most of its behaviour done offline and in the back-end. The key requirement for the web framework was that is must be compatible with the chosen database engine. With this in mind, I also needed to choose a database engine which was suitable for this project and performs the actions needed.

When choosing a database engine it was clear that there was no need for a relational database as the database will only be used to store and read the skill ontology, as discussed in the requirements (Section 3.2). This meant a NoSQL database[22] would be a suitable option. MongoDB[35] was one option which uses JSON-like ”documents” to store data. Another option was Amazon’s DynamoDB[5] which uses a key-value store model. Due to the budget of this project, it was not feasible to use DynamoDB. MongoDB is the most used NoSQL engine today[30], it is easy to use and setup and it has all the functionality needed for this project. MongoDB is commonly used and therefore has a large amount of documentation. My advisors were also very familiar with this technology meaning there was a lot of resources for this technology if needed. Due to its quick setup time, large amounts of documentation and my advisor’s familiarity with this technology I decided to use it for this prototype system.

Once the database engine was selected I had to decide on a web framework which could be integrated with it. I opted for Node.JS[36] as it is commonly used with MongoDB due to it’s all JavaScript stack. Another key reason why they are used together frequently is that they both use JSON which means they integrate well together. NodeJS objects are JSON and MongoDB allows for storing and retrieving JSON documents. MongoDB’s documentation also states that it ”is a perfect fit for Node.JS applications”[35] due to them both using JavaScript and JSON. There was two key reason why I decided to use NodeJS for this project and that was quick setup time and the large amount of documentation and support with MongoDB. NodeJS has a quick setup time due to it not being a framework but an asynchronous event-driven JavaScript runtime[36], this means it is not framework heavy and ideal for a prototype system.

4.3 User Interface

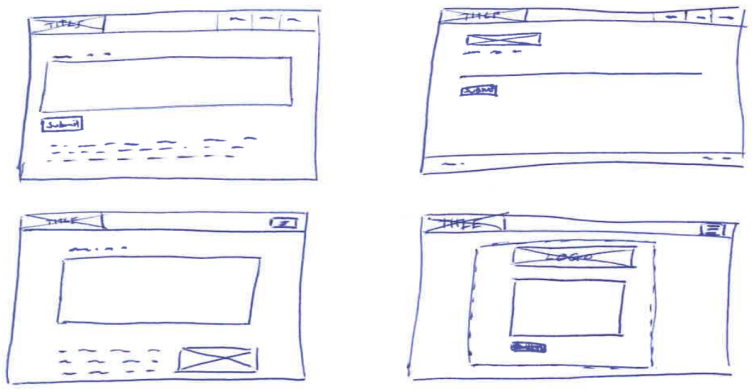
When designing the user interface for the system there were two key features which must be met:

- **Ease of use**: The web application must be usable by untrained users.
- **Clarity**: The results of the system must be clear to the user and give some assurance that the results are correct. For example, showing the words used to find the suggested skills.

A series of wireframes were created to illustrate how the application will satisfy the key functional requirements. The main use of this application will be on desktop and as the user interface was not particularly important for this project I decided not to investigate mobile use of the system. There will be two main pages for the web application, one being the homepage where the user will be able to input a job description and the other being the overview page where the results of the system are presented to the user.

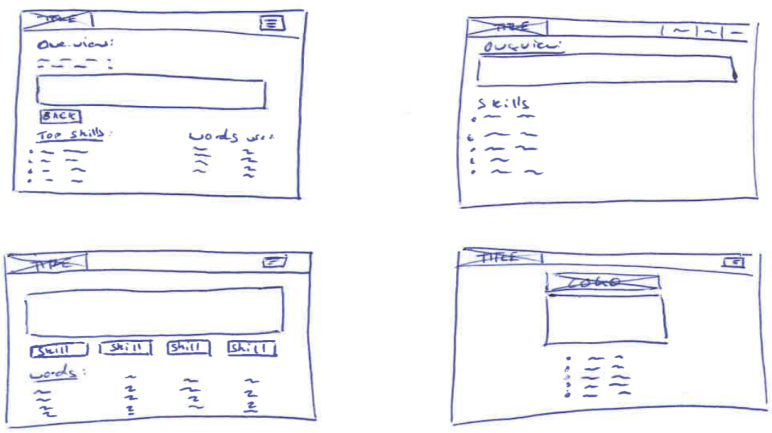
Wireframing is an interface planning process which takes place before development[40]. It is used to plan the layout of the application, and look at alternative structures. These will be used in the implementation phase of development to aid the developer with a visual guide of how to arrange elements to best accomplish a particular purpose. The following four wireframes (Figure 4.4) are initial ideas for the home page:

Figure 4.4: Home page initial sketches



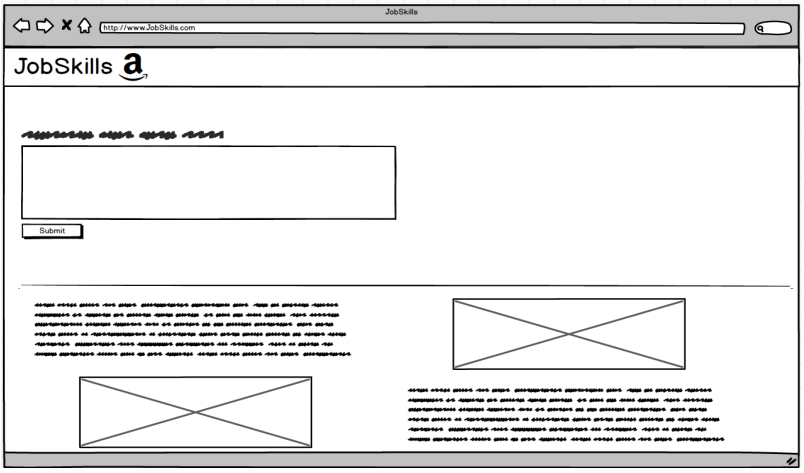
The home page will be the first page viewed by a visiting user. It will have some information about the project and what to do but will also have somewhere to input a job description, select a ranking algorithm and submit the job description. The following four sketched wireframes (Figure 4.5) are initial ideas for the overview page:

Figure 4.5: Overview page initial sketches



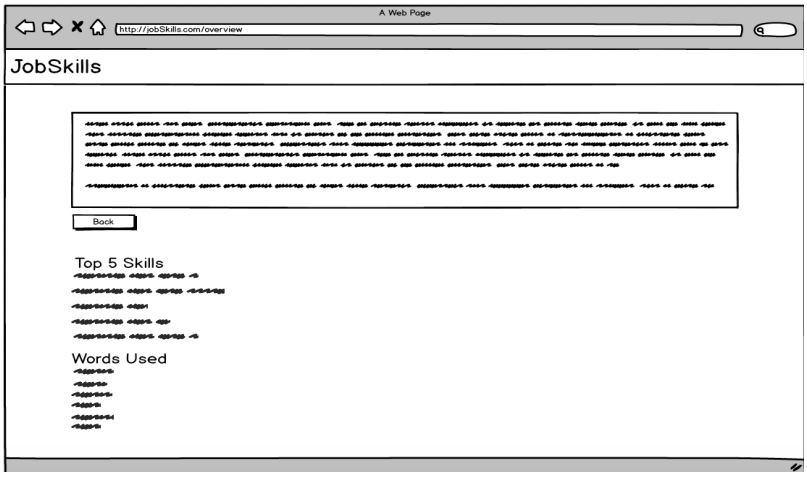
Once a user has submitted a job description they will be redirected to the overview page. This page will display the submitted job description, the top five related skills for that job, and the words and phrases used to identify those skills. The main purpose of this page is to allow the users to see the top ranked skills and give some assurance that those skills are correct. Refined wireframes for each of the two key pages, the home page (Figure 4.6) and the overview page (Figure 4.7) were then created using Balsamic[6]. These wireframes will be used as a visual aid when designing the front end of the application.

Figure 4.6: Final wireframe for the home page



A simplified design was chosen for the home page to ensure the ease of use was achieved. The user will be presented with a textbox with some text for guidance, a submit button and a tab bar to select which algorithm to use.

Figure 4.7: Final wireframe for the overview page



This overview page design was chosen to ensure clarity in the results. The list of top five skills in a standard list under the job description are clear to the user compared to skill tags appearing under the job description.

Chapter 5

Implementation

This chapter discusses the implementation of the system in detail, explanations of each ranking algorithm and the issues encountered. This section is outlined as follows:

- 5.1 **Back End**
  - 5.1.1 Skill Ontology
  - 5.1.2 Ranking
    - 5.1.2.1 Algorithm 1
    - 5.1.2.2 Algorithm 2
    - 5.1.2.3 Algorithm 3
  - 5.1.3 Results
    - 5.1.3.1 Algorithm 1
    - 5.1.3.2 Algorithm 2
    - 5.1.3.3 Algorithm 3
- 5.2 **Front End**

5.1 Back End

5.1.1 Skills Ontology

For this project, the goal was to create a skill ontology(Section 2.2.3) where each skill had a hierarchical relationship with each other. Since this system is being designed to improve a recommendation system the skills should be skills people might add to their profiles. The skill ontology will be stored in a database to allow each ranking algorithm to access the standardised skill tags. These are the skills which will be returned to the user if they match to a job description. We decided to limit the skill set to only software engineering jobs for this prototype system as discussed in the requirements (Section 3.2). The ontology should contain general skills which most people would expect like ”Problem Solving” but also more specialist and rare skills such as ”Machine Learning” to enhance matches when used for the recommendation system (Section 3.2).

We decided to limit the skill set to only software engineering jobs for this prototype system. The set should contain general skills which most people would expect like ”problem solving” but also more specialist and rare skills such as ”Machine learning” to ensure better matches. To start off I considered existing skill ontologies,

however I could not find a sufficiently large skill set. Instead, I decided to concatenate various sources to gather a large and broad scope of skills. The Association for Computing Machinery[2] created a classification[3] which was used as the basis for the skill ontology. It contains research areas within the computing field but also contains common programming languages and technologies. This was structured as a hierarchy tree represented in SKOS XML[50] with a root node of "General and reference" which branched off into multiple research areas ranging from "Networks" to "Mathematics of computing". However, some of the areas did not apply to the skill ontology, for example, "People in computing". This lead to my first task of refining the ontology and removing any areas not applicable to this project.

Another key problem with this ontology is that it does not contain any general skills because it is a classification for the computing field rather than a classification for skills. To solve this problem I extended the ontology with multiple sets of general skills and programming languages from multiple sources[31][14][11][54][37]. This extended the skill set by 412 skills ranging from "Problem solving" to "First Aid". This is not a perfect ontology, some of the skills have some overlap and the names could be refined. However, this is enough to get an initial insight on how the ranking algorithms will perform for this prototype system.

5.1.2 Ranking

The ranking algorithm will be the heart of the system which will score and rank each skill for a given job description. To investigate which techniques could be used to create an optimal ranking algorithm, I decided to implement three different algorithms and compare them as discussed in the requirements (Section 3.2). All three use Word2Vec word embeddings with the first algorithm using word embeddings on a word to word basis to create a similarity scoring between job descriptions and skills; The second algorithm uses word embedding vectors, and the third algorithm is an extension of the first algorithm by using a term frequency-inverse document frequency(TF-IDF) technique.

Algorithm 1

The main problem with how Amazon currently match jobs with skills is that there will only be a match if there is a complete match of the skill name. However, this will not necessarily match all the jobs which require that skill. For example, lots of technologies and tools use abbreviations to identify themselves and not all required skill will be precisely mentioned.

One way to get round this problem is to find commonly used words and phrases which are used to identify each skill. To do this I investigated Word2Vec word embeddings (Section 2.2.1). Word2Vec takes a large corpus as its input and creates a vector space where each word is assigned to a vector. Words which share context within the corpus will have word vectors located in close proximity to one another in the vector space. However, as the name may suggest, this will only work for words and not for phrases. To overcome this problem I had to find related words for each word in a skill. For example, if the skill is 'Ruby on Rails' it will find related words to 'ruby' and 'rails'. For each related word, it will be assigned a relevance score for that skill between 0 and 1, 0 being not relevant and 1 being a complete match.

Once I had the related words for each skill I needed to store them in a suitable data structure. If I kept the structure as it was $\langle skill, \langle word, score \rangle \rangle$ it would mean iterating through each word for each skill and search for it in the description. This would be computationally expensive and very inefficient as we would have to iterate through the entire ontology every time. By opting for the structure $\langle word, \langle skill, score \rangle \rangle$ it would improve performance as we will now only need to iterate through the job description compared to the whole skill ontology.

Once a suitable data structure was in place allowing us to query for each word in a description, we had the task of up-weighting commonly mentioned words. We first needed to strip stop words and then find the most

common words in the job description. One way of doing this is to use a word cloud which is a way of visualising term frequencies in a block of text. The following image (Figure 5.1) shows an example word cloud output for a description of a software engineering job at Amazon.

Figure 5.1: Word cloud example for an SDE job description. Generated using wordclouds.com[53]

However, I decided against using a word cloud as I wanted more flexibility with stopwords. From the example above, 'Amazon' is the most common term in the description, however Amazon should be a stopword for this system. I then decided to opt for a standard term-frequency technique by stripping stop words and counting the occurrences of each word in a job description. Once this was in place it was a simple task of iterate over the description summing up the scores for each skill and normalising. The below algorithm shows how the scores were generated by summing up the scores for each skill(s) by finding it's similarity with each word(w) in the job description(D).

$$\frac{\sum_{w \in D} sim(w, s)}{\texttt{max\_sim\_score}}$$

Once running the algorithm a couple of times there was an obvious problem, all the suggested skills were usually over three words long. This technique works on a word to word basis meaning longer skills are more likely to be matched as they have more related words compared to smaller skills. To fix this problem I divided each of the scores by the length of the skill. This gave the following updated algorithm where a score for a skill is the sum of the similarity between each word in the description divided by the length of the skill. This is then normalised by dividing by the maximum similarity score.

$$\frac{\sum_{w \in D} \frac{sim(w,s)}{|s|}}{\texttt{max\_sim\_score}}$$

`max_sim_score` is a free parameter[10] as the maximum similarity score is not bounded. This means the score can not be predicted precisely or constrained by a model[21]. The only way to normalise the similarity score is to make an assumption of the maximum length of a job description and the maximum number of times a skill appears in a job description. This will make an assumption that a job description cannot generate a score over this limit. However, this does mean we could craft a description which violates this assumption. Free parameters are common in information retrieval systems, especially ranking systems, for example, the Okapi BM25[24] which was used in the Google search engine uses two free parameters.

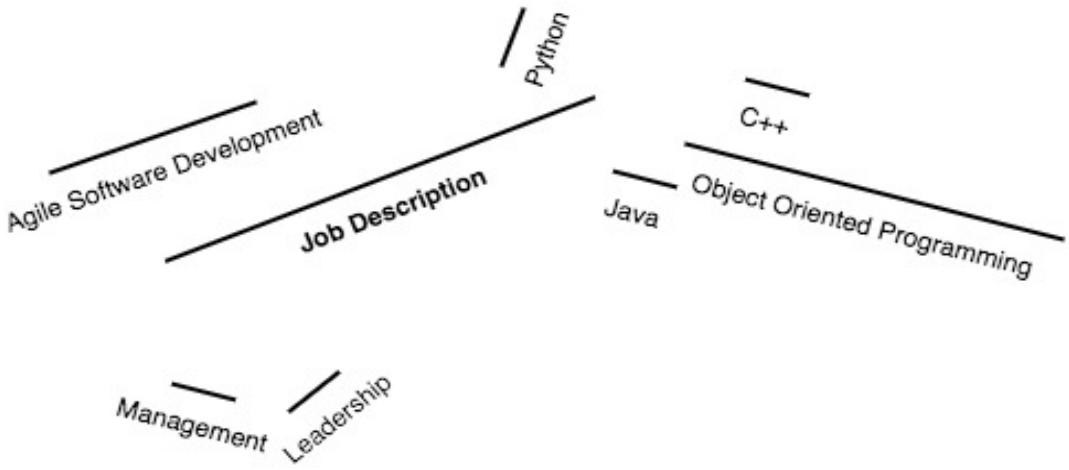
The problem with using word embeddings on a word to word basis for representing skills is that it calculates a similarity score with the words making up the skill rather than the skill itself. This does not give a good representation of the skill, however, creating a word embedding vector for the skill will give a better representation of the skill. The second algorithm will attempt to tackle this problem.

Algorithm 2

The second algorithm makes use of Word2Vec word embedding vectors. The problem with the first algorithm is that it finds related words to a skill on a word to word basis meaning some of the words do not relate to the skill but relate to a word in the skill. By representing each skill with a vector in a Word2Vec network, this will give a representation of the skill rather than words in the skill. Equally a vector can be created for a job description which can be used to find related skills. From the definition of neural word embeddings, vectors within the same proximity are related to each other, hence skill vectors located near the job description vector are related to that job. The closer the skill vector is to the job description vector, the more relevant the skill is to that job. By representing skills and job descriptions by vectors, it allows for an easy and fast comparison with a mathematical similarity between skills and it can give an accurate guess of the association between skills and the given job description. However, this means creating vectors to represent all the skills and being able to create a vector for an inputted job description.

To create the skill and job description vectors I stripped the stopwords and took the average of all the word vectors in the skill/job description. There are other ways of creating a word embedding vector for phrases or paragraphs, for example taking the average of all word vectors and using TF-IDF to up-weight more relevant words in the block of text. However, a skill is made up of a small amount of words and therefore, a TF-IDF technique would not help create more representative vectors for these skills. Once this was done, comparing the description vector with the skill vectors could be done by using the cosine distance between the two vectors, this will find the closest related skills to that job description. From the example below (Figure 5.2), closer the skills are to the job description vector more relevant they are. For this example, ”Python” and ”Java” are the most relevant skills to the job description where ”Management” and ”Leadership” are less relevant as they are placed further away from the job description vector.

Figure 5.2: Skill and job description word embedding vectors example



The problem with this algorithm and the previous algorithm is that they find the closest related skills to the job description, however, this may not find the most relevant skills for a job because these skills might not be highly relevant to the job type. For example, computer skills will always score a high similarity score with

a software engineering job but this is not particularly useful when used on a job card for that job. The third algorithm attempts to tackle this problem.

Algorithm 3

The motivation of this algorithm was to find a way to up-weight useful skills for the use case of skill tags. This algorithm is an extension of the first algorithm by introducing a term frequency-inverse document frequency(TF-IDF) technique. TF-IDF is a technique of how to decide the importance of a word to a document in a collection or corpus. For this example, the document would be the job description and the corpus would be all software development jobs. This technique should up-weight skills which are less often mentioned in its job type. To find the document frequency using all jobs would be a difficult task as it requires to gather a lot of job descriptions to be able to obtain the statistic. For this reason, I looked at alternatives.

The purpose of the document frequency is to find which skills would be most beneficial as tags. Another way of finding relevant skills is to look at how many times people search for a given skill. Wikipedia Pageviews (Section 2.2.2) is a tool which allows one to see how many people have visited an article during a given time period. This gives us the opportunity to find the most searched for skill in the ontology and up-weight them in the ranking. However, this does mean matching each skill to a Wikipedia page.

To gather the Wikipedia pages for each skill I scraped Google search. There are many more suitable alternative for doing this task such as Google Custom Search Engine[16] or Bing Search API[32]. However, both of these APIs cost to use with Bing Search API requiring a monthly subscription and Google Custom Engine only allowing for 100 requests a day for free. Therefore, we could not use these alternatives due to the budget of the project. This made this step tedious and time-consuming.

To use Google search to find suitable Wikipedia pages for each skill, I had to limit the results to only return Wikipedia articles by limiting the article source to en.wikipedia.org and then taking the top result. The following request was used to gather the appropriate Wikipedia article:

```
site:en.wikipedia.org {skill}
```

Once the Wikipedia pages had been collected they can be used to find the page view statistics for each skill. Wikipedia Pageview was used to find the average views per day for 2016 to find recent trends in searches. These scores can now be used to up-weight the most searched for skills to ensure the suggested skills are relevant to the job and as a skill. The Wikipedia scores were then added to the algorithm to produce the following:

$$(1-\lambda)(\frac{\sum_{w \in D} sim(w,s)}{\texttt{max\_sim\_score}}) \times \lambda(wikiScore(s))$$

However, this must be normalised by setting the Wikipedia score to $\frac{wikiScore(s)-min}{max-min}$. The maximum will be set as the maximum viewed skill in the ontology and the minimum score for a Wikipedia page will be 0 if the page had not been viewed. Hence, the algorithm will be:

$$(1-\lambda)(\frac{\sum_{w \in D} sim(w,s)}{\texttt{max\_sim\_score}}) \times \lambda(\frac{wikiScore(s)}{\texttt{max\_wiki\_score}})$$

Lambda (λ) is a balancing parameter which can be tweaked to find the optimal balance between similarity and relevance scores. If lambda is less than 0.5 then the similarity score is weighted higher, whereas if it is more than 0.5 the Wikipedia relevance scoring is weighted higher.

5.1.3 Results

There are problems with all three of the algorithms, this section will outline the key problems with all three of the algorithms.

Algorithm 1

The first algorithm has problems with producing skills which are not meaningful to the job type and it also has problems with finding highly precise skills such as programming languages and technologies. It seems to output general skills which most people would expect very well but has problems finding more specialist and rare skills. This will be a problem as the requirements of this system is to produce a broad range of skill from general to more precise skills to improve recommendation matches (Section 3.2).

Algorithm 2

On the other hand, algorithm 2 seems to work very well with finding specialist skills but has problems finding general skills. There are a few problems with using Word2Vec vectors, one being that not all skills have a word embedding vector. If a skill does not contain a word which is in the word embedding network then it can not be represented by a vector. For example, the network I use for this project does not contain a vector for JavaScript and so does not have a vector in the network. This problem can be resolved if the Word2Vec was trained with a larger corpus focused on learning common phrases in job descriptions.

Another problem with this technique is that the suggested skills for a job will usually be very similar. This is because the vectors of the suggested skill will usually be in close proximity in the network. This could be improved by introducing a TF-IDF technique, for example, find the closest vectors with a high relevance for the job type. This means that each skill will get a distance score but will also have a relevance score for the job type which will up-weight highly relevant skills and down-weight less relevant skills.

Algorithm 3

The third algorithm was introduced to improve the first algorithm by introducing a TF-IDF technique by using Wikipedia Pageviews. The current problem with this algorithm is that skills with a high page views such as 'Java' will always have a high score if it is mentioned in a job description. However, this technique seems to have solved the problems with the first algorithm but now it is a matter of finding the right balance between the similarity scoring from the first algorithm and the Wikipedia Pageview score. One way to improve the relevance scoring is to cap the page views to prevent large outliers and ensuring all highly relevant skills will have a similar score.

5.2 Front End

When implementing the user interface it was important to reduce duplication of code and remove any redundancy. To reduce the chance of duplicate and redundant code I used Handlebars.js[19] which is a templating engine to enable developers to create templates for their web applications. By using handlebars, it meant avoiding to write boilerplate HTML code multiple times and allows for a consistent design throughout the website by using templates. The web application makes use of Materialize front-end framework created and designed by

Google[29] to create a flowing and unified interface. Google's goal for this system is to "unified user experience across all their products on any platform"[28], making it a perfect tool to ensure consistency throughout the websites pages. This helps the non-functional requirement of ease of use and portability (Section 3.3) as it creates consistent responsive user interfaces which can be used on all devices.

Implementation of the user interface began with the creation of the home screen to allow the user to input a job description. The user can input their job description into a textbox and select an algorithm using a tab bar above this. The implementation of the overview page required some more thought. As discussed in the requirements (Section 3.2), we wanted to clearly show the results of the algorithms and show how the results were achieved. Using a textbox highlighter we can show which words were used to generate the top five skills. There is also a link to an appropriate Wikipedia article next to each skill to help the user if they do not understand what the skill is. This was a could have in the requirements (Section 3.2) but felt it helped the users understanding of the skill. As discussed in the design (Section 4.3), both of the key pages in this application were created using the wireframes as structure references. Refer to appendix B for the final application screenshots.

Chapter 6

Evaluation

This chapter goes into detail how the system was tested and evaluated, the techniques used and how the system meets the requirements. This section is outlined as follows:

- 6.1 **Correctness**
- 6.2 **User Evaluation**
  - 6.2.1 Amazon Demo
  - 6.2.2 User Survey
- 6.3 **Technical Evaluation**
  - 6.3.1 Setup
  - 6.3.2 Trec Eval
  - 6.3.3 Results
  - 6.3.4 Summary

6.1 Correctness

Mocha[34] and Chai[12] were used to test the correctness of the web application. These technologies were used to perform unit testing by creating a test suite. Mocha is a Javascript test framework which runs on NodeJS for asynchronous testing[34] and Chai is a BDD / TDD assertion library which can be paired with Mocha[12]. Chai allows for very readable tests with its easily comprehensible syntax using English readable keywords. From the example below (Figure 6.1), constructing a test is like writing an English sentence.

Figure 6.1: Chai and Mocha example test

```
it('generateScore should return score between 0 and 1', function(done) {
  var score = generateScore(skillMap, skills[0]);
  expect(score).to.be.below(1);
  expect(score).to.be.above(0);
  done();
});
```

The application has four testable elements, the ranking algorithms,the website logic, the user interface, and the database. The goal for these tests is to ensure the system works as intended and meets the project requirements. Testing the user interface ensures that the users can see the elements they are supposed to see. For example, when a user submits a job description they should be redirected to the overview page where it should show the top five skills and the words used to identify those skills. Tests were created to ensure that the web application could connect to and retrieve data from the database. The example below (Figure 6.2) ensures that the collection ”skillWiki”, which holds the Wikipedia view statistics for each skill, can be accessed and used to retrieve the view statistics for the skill ”Java”.

Figure 6.2: MongoDB testing with Mocha and Chai

```
it('should find \'java\' in skillWiki collection', function(done) {
  this.timeout(15000);
  MongoClient.connect(url, function(err, db) {
    if(err) {
      assert.isOk(false, 'Failed to connect to database');
    }
    else {
      var collection = db.collection('skillWiki');
      collection.findOne({skill: 'java'}, function(err, result) {
        if (result){
          should.exist(result.skill);
          expect(result.skill).to.equal('java');
          done();
        }
        else {
          assert.isOk(false, 'Could not find \'java\' in skill collection');
        }
      });
    }
  });
});
```

Unit tests were created to test the code I had written, however testing some parts of the system was not feasible. The parts of the system which I could not test was ensuring that a word truly had a relation with a skill, that the Wikipedia page view statistics were correct and the word embedding vectors were correct. However, tested external libraries were used to perform these parts of the system such as Wikipedia Pageview and Word2Vec which provides trusted outputs.

6.2 User Evaluation

6.2.1 Amazon Demo

A demo was performed in front of six participants who were all in the team which will be using the system if it is a success (Section 3.1.1). When asked which ranking algorithm they preferred, most of the participants preferred algorithm 3 with one participant preferring algorithm 2. The reason for one preferring algorithm 2 was that they felt that the skills produced by algorithm 2 gave more meaning to the job compared to the skills suggested by algorithm 3. Algorithm 3 tends to output skills such as ”Java” or ”C++” whereas algorithm 2 produce skills such as ”Systems and tools for interaction design”.

The problem with user evaluation for evaluating IR systems is that it is very subject, for example, in this case, it is subjective to decide what skills are relevant or non-relevant for a given job. Once looking at the job in more detail, we agreed that a lot of the skills suggested by algorithm 2 were either weakly related or not related at all. One of the main reasons algorithm 3 got most of the votes was that the skills suggested were more precise and

they want short and precisely named skills because they will need to fit on the job cards on the JobFinder website (Section 3.2). The objective of the skill tags is to improve the preview of a job and accurately represent that job. Most of the participants taking part in this demo believed that algorithm 3 produced better results for the above objectives compared to the other two algorithms.

6.2.2 User Survey

A user survey was performed to highlight the key problems with the prototype system, evaluate how well the system's suggestions are compared to human suggestions and to compare the three ranking algorithms. There were three parts of the evaluation, the first asking the user to supply suggested skills for a job description to evaluate the algorithm's suggestions against human suggestions. The second part was to give a relevance score to the output to each algorithm and supply some comments. The third and final part of the survey asked participants what their preferred algorithm was and to supply some comments on why they felt that. Refer to appendix C for the full survey.

Results

Section 1

For the first section of the survey, the participants were given a job description and told to think about what skills they felt would be suitable for that job while thinking about the skill tags use case. This section will help to identify the kind of skills people would expect to see from this system. The output from this section can also be used to compare against the system's outputs to evaluate each algorithm's rankings. When analysing all the results from this section there was very little overlap in skills which shows that each participant had their own subjective views on what skills are relevant to the job and suitable for its use case. There were only three skills which occurred more than once and that was Java/object-oriented programming, web development, and software engineering. This shows how relevance is a subjective notion and that everyone will have their own views on what skills are relevant to the job.

Most of the skills from human suggestions were not in the skill ontology or were related to multiple skills in the ontology. Therefore, it was hard to produce a single value for pairings between human and computer suggestions. To evaluate each ranking algorithm I compared each of the suggested skills with the algorithm's outputted results to see if there were any complete matches or that the skills were related. A complete match meaning they both had the same name and a related match meaning that they were the same skill but with different names. For example, "Java", "C++/Java" and "Object Oriented Programming" were seen as related skills.

Table 6.1: Results from section one of the user survey

	Algorithm 1	**Algorithm 2**	**Algorithm 3**
Complete Match	6%	0%	10%
Relevant Match	17.3%	0%	20%
Overall	23.3%	0%	30%

From the table above (Table 6.1), algorithm 1 and 3 produced similar results with algorithm 3 marginally better. Algorithm 3 produced a highest complete match and relevant match with algorithm 2 not creating a single match. This however does not mean the skills produced by algorithm 2 are not relevant, but they are not the most relevant skills

Section 2

The second section of the survey required the participants to use each of the ranking algorithms with the same job description and assign a relevance score for each of them. They were also asked if they felt any skills were missing and if any of the suggested skills were not relevant. This supplied us with an average relevance scoring for each of the algorithms. The relevance scoring was on a scale of 0 to 5, with 0 being strongly disagree that the suggested skills were relevant to the job and 5 being strongly agree.

Table 6.2: Results from section two of the user survey

	Mean	**Standard Deviation**
Algorithm 1	3	1.095
Algorithm 2	1.83	0.983
Algorithm 3	3.5	0.548

Algorithm 1 received a high average and standard deviation which means it had a large variance in scoring but usually scoring a high relevance. It received a high relevance scoring from most participants but some felt that there was a lack of programming languages suggested. One of the recurring comments on this algorithm was that some of the suggested skills were very similar and that there was some overlap in them. However, that problem could be down to the underlying skill ontology rather than the algorithm itself.

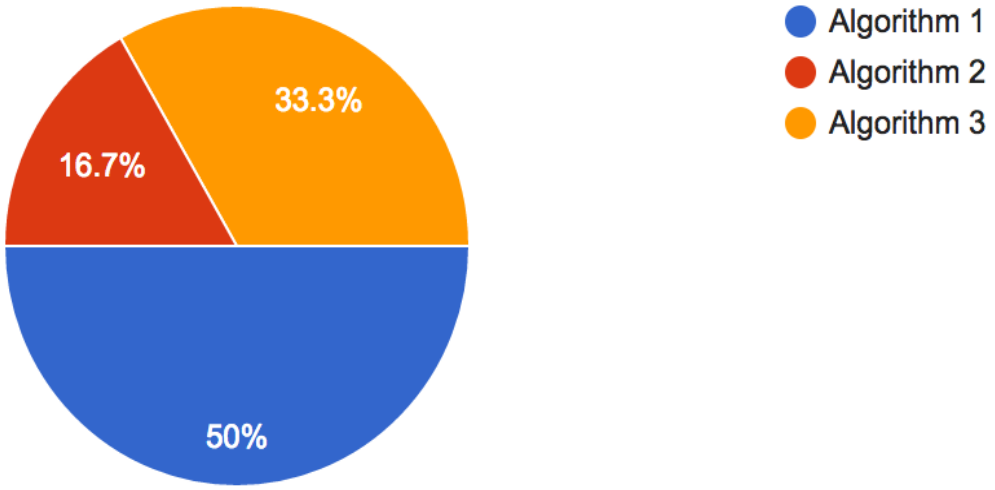
Algorithm 2 received the lowest average relevance score but this is still an improvement from the first section of the user evaluation (Section 6.2.2). When asked if the participants felt that skills were missed out, most felt that there was a lot missing from the results but some did not with one of the participants saying ”No, this time were more specific”. This backs up the fact how relevance is a subjective notion with some participants feeling the skills were more specific and others feeling that they were vaguely relevant.

Algorithm 3 produced the highest average relevance score and the lowest deviation meaning it received a consistently high relevance score from all participants. A common comment on this algorithm was that it produced specific skills but required a more broad range. Participants felt that a lot of the skills were similar but they did feel that it would be ”helpful for those applying for the role” and that ”it just needs a good balance between specific expertise and more general skills”.

Section 3

The third and final part of the survey asked participants to decide on a preferred algorithm and to supply some comments on why they felt that. The below pie chart (Figure 6.3) shows the results of this section.

Figure 6.3: Section 3 of the User survey results



This chart is a good insight of the limitations of user evaluations when it comes to evaluating IR systems. One of the main limitation being that participants are subjective, especially when it comes to relevance. Relevance is a subjective notion, with Keith van Rijsbergen, Professor at the University of Glasgow saying ”different users may differ about the relevance or non-relevance of particular documents to given questions”[48]. From the results above, there is no one algorithm better than the other with the participants split between all three of the algorithms. Algorithm 1 received half of the votes but this is not a reassuring result that it is the better ranking algorithm. A key problem with this evaluation is that it is too small, the participants are deciding which ranking is better based on one job description. To get a better representation of the results we need to test the algorithms on a larger scale.

6.3 Technical Evaluation

The introducing of information retrieval(IR) evaluation methods for this project allows for the measurement of progress, verifying performance and the comparison of algorithms. The reason to evaluate IR systems can be categorised into three main areas, economic reasons, scientific progress and verification. Economic reasons refers to the people paying for the system and that they want to know how effective it is; Scientific progress refers to when researchers want to know if progress is being made; And verification refers to the verification of the performance of the system. They are commonly used as they ”speed the transfer of technology from research labs into commercial products by demonstrating substantial improvements in retrieval methodologies on real-world problems”[1]. To measure the performance of each ranking algorithm we need to decide which algorithm produces the most relevant skills for a large number of job descriptions. The system will be tested based on a large test collection, which consists of documents, queries, and relevance scores.

6.3.1 Setup

IR Experiments traditionally, use the same queries and documents and are used repeatedly, with different systems or variants of systems. For this project, the documents are skills with the query being how relevant they are to a job description. The set of relevance judgements is given as a range from 0 to 3.

- 0 - not relevant
- 1 - weakly relevant
- 2 - relevant
- 3 - highly relevant

The relevance assessment is a very tedious task of analysing each job description and deciding on the most relevant skills for that job. For each job description, each skill will be assigned to a score for how relevant that skill is to that job. A skill is relevant to a job description if it is required for that job, not just if it is stated in the job description. If the skill is mentioned multiple times in a description then it will be highly relevant but if a skill is not precisely mentioned in the description but is related to a required skill or job type then it will be weakly related. When deciding on the size of the test collection we need it to be large enough to get an average performance over a fairly large test set but we do not want it to be too big as the creation of this test collection is time-consuming. Fifty information needs have been found to be a sufficient minimum[27] and hence why I opted for fifty job descriptions.

The collection of relevance assessments is usually a time-consuming process involving humans. Obviously, we can not perform a relevance assessment on all document and query pairs like Cranfield[20] who uses a small collection. However, for far larger modern systems it is usual to only use a subset of documents for each query to generate a relevance assessment[25], this is called the assessed set. Pooling[26] is a standard approach to gather an assessed set, where it is formed from the top k documents returned by a number of different IR systems. However, we did not have access to another system to do this and hence we had to use human judgments. The problem with humans relevance judgments is that they are subjective, however Christopher Manning, Professor of Computer Science and Linguistics at Stanford University stated "a human is not a device that reliably reports a gold standard judgment of relevance of a document to a query. Rather, humans and their relevance judgments are quite idiosyncratic and variable. But this is not a problem to be solved: in the final analysis, the success of an IR system depends on how good it is at satisfying the needs of these idiosyncratic humans, one information need at a time"[26].

We decided to only test the top ten returned skills from each algorithm and hence only identified the top ten relevant skills for each job description in the test collection. Therefore, the resulting test collection consisted of fifty job descriptions with ten skills each with an associated relevance score. The same job descriptions, skills and relevance scores are tested against each ranking algorithm.

6.3.2 Trec Eval

Once the test collection had been created we needed a mechanism for performing an evaluation on this collection. Trec_eval is a program to evaluate TREC results using the standard, National Institute of Standards and Technology (NIST) evaluation procedures. It provides the infrastructure necessary for large-scale evaluation of text retrieval methodologies. To use Trec_Eval I first needed to produce the output of each algorithm for each query. For each job description, the algorithm will output the top ten related skills with an associated ranking and scoring. Trec_Eval used the scores and the rankings associated with each skill by the algorithm and uses them to be compared to the relevance judgments created in the test collection to produce a recall and precision scoring. Precision is the fraction of retrieved documents that are relevant:

$$Precision = \frac{\#(Relevant\_Items\_Retrieved)}{\#(Retrieved\_Items)}$$

Recall is the fraction of relevant documents that are retrieved:

$$Recall = \frac{\#(Relevant\_Items\_Retrieved)}{\#(Relevant\_Items)}$$

From the below table (Table 6.3), you can see that optimal ranking algorithm will have a minimum false positive rate, where it returns non-relevant skills and maximise true positive results, where it return relevant skills.

Table 6.3

	Relevant	**Non relevant**
Retrieved	True Positive (TP)	False Positive (FP)
Not retrieved	False Negative (FN)	True Negative (TN)

For this project, we are focused on high precision rather than high recall and minimising false positive results. We want the algorithms to produce the best quality of suggestions with all of the top five skills being relevant to the job. Therefore, we want to find the algorithm producing the best precision score for all job descriptions. The Mean Average Precision(MAP) is a widely used measure in research papers for summarising rankings from multiple queries by averaging average precision. Average precision is the average of precision values for a query. If no relevant documents are retrieved then the precision will be 0.0 and if all the documents are relevant then the precision value will be 1.0. MAP is a very useful metric as it measures both precision and recall, it is calculated using the following:

$$AP = \frac{\sum_{i=1}^{n}(Precision(i) \times Rel(d_i))}{|RelevantTotal|}$$

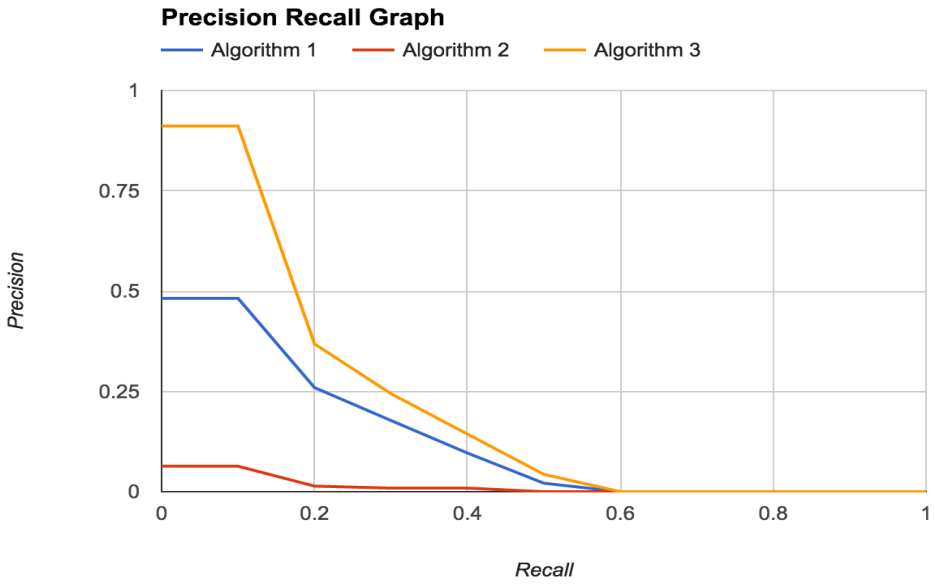
$$MAP = \frac{1}{|Q|}\sum_{q=1}^{Q} AP$$

MAP is the average of all average precision values for each query(q, $q \in Q$). The average precision is the sum of precision values multiplied by a document relevance for that query and then all divided by the total number of possible relevant documents for that query $RelavantTotal$. In this case, $RelavantTotal$ will always be ten as we set our assessed set to size ten for all queries. $Rel(d_i)$ denotes if the document is relevant to the query, 0 if it is not and 1 if it is. $Precision(i)$ denotes the current precision value for the query. MAP is not a precision metric but a metric to assess the ranking of the algorithms. However, this a good single value to evaluate the performance of a ranking algorithm. The algorithm with the highest MAP performance produces the highest average of relevant skills over a large amount of job description and therefore is the highest performing ranking algorithm.

6.3.3 Results

Once the test collection and resulting outputs from each algorithm were produced, they could then be evaluated using Trec_eval. The graph below (Figure 6.4) is a precision-recall graph which shows the precision values at standard recall values. If there is a high area under the curve then this signifies both high recall and precision, where high precision signifies a low false positive rate, and high recall relates to low false negative rate. For this system, we want to minimise false positive rates as we do not want to produce non-related skills for a job. From the graph below (Figure 6.4), it is clear that algorithm 3 has a much higher precision rate compared to the other two algorithms at all recall values. Therefore, algorithm 3 produces the lowest false positive rate between the three algorithms.

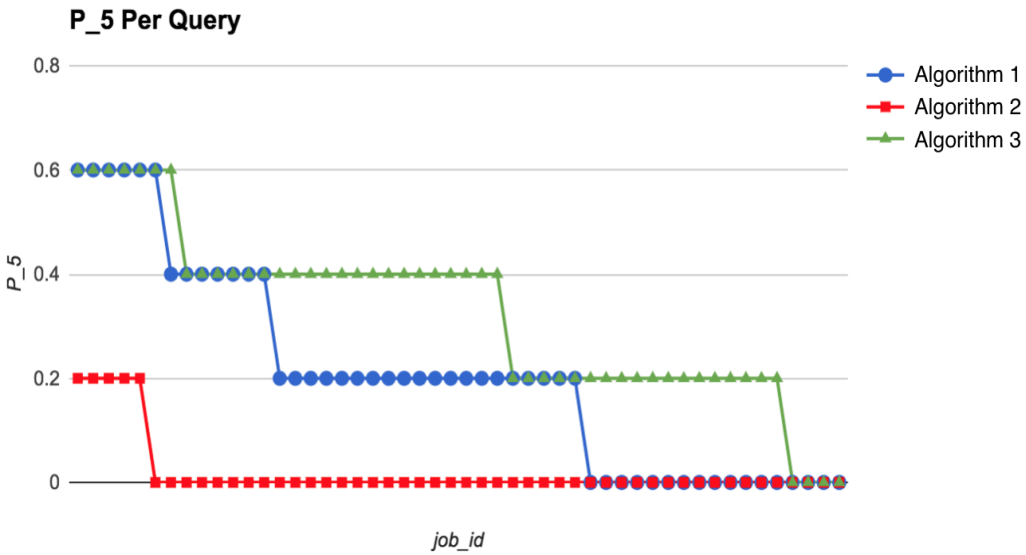
Figure 6.4: Precision-recall graph for algorithm 1,2, and 3



The mean average precision (MAP) is a very useful metric as it measures both precision and recall and it is also top-heavy as higher ranked documents have more weight. Since MAP is top-heavy it means it is a good measurement for indicating which algorithm produces the best ranking. This can be used as a single value to compare each of the ranking algorithms. When analysing the MAP values for each ranking algorithm, algorithm 3 produced a far superior value compared to the other two algorithms with a value of 0.1680. Algorithm 1 produced a MAP value of 0.0984 and algorithm 2 with a very poor result of 0.0084.

Another useful metric for this system is P_5 which is a measurement of precision at a given threshold, in this case the first five ranked documents. This is a useful metric for this system as it will tell us how many of the top five suggested skills were relevant to the job on average. Algorithm 2 produced the lowest P_5 value of 0.02 which means that only 2% of its suggested skills are relevant to the job on average. Algorithm 3 had a far superior P_5 value of 0.324 with algorithm 1 producing a value of 0.208. However, it is also useful to look at the distribution of P_5 values for each query to see the variance in performance for each algorithm. The following graphs (Figure 6.5) shows the P_5 values for each algorithm for each query.

Figure 6.5: Distribution of P_5



Algorithm 2 produces very low precision rates with the highest precision of 0.2. Only 10% of the queries produced a P_5 result which was not 0 for algorithm 2 which means it commonly produces no relevant skills for a job description. Algorithm 3 produces the least amount of P_5s with value 0, with four of the queries producing no relevant skills. This graph shows how algorithm 3 usually produces a precision value between 0.6 and 0.4 with algorithm 1 producing precision values between 0.6 and 0.2. On average algorithm 3 produces the best results and is least likely to produce no relevant skills. However, it is valuable to remember that the assessed set is very small with only ten skills given a relevance score for each query. Therefore, not all non-relevant documents in this evaluation will necessarily be non-relevant - they might just be non-assessed. However, this method of evaluation is a good insight on how the algorithms will perform and an effective way to compare each of them.

We can find the significance of the above results by comparing the algorithms using a paired t-test. By using a paired t-test we can calculate the two-tailed p-value which finds the probability of the first system being better than system two by chance. A small p-value (typically ≤ 0.05) indicates a strong evidence against the null hypothesis[47], with the null hypothesis being that the two systems perform the same. Therefore, we can reject the null hypothesis with a p-value less than 0.05. If a p-value is less than 0.01 it is statistically highly significant[47]. The below table (Table 6.4) shows the results produced by the paired t-test for each permutation of algorithms.

Table 6.4

	Two-Tailed p-value
Algorithm 1 vs. Algorithm 2	0.0001
Algorithm 3 vs. Algorithm 2	0.0001
Algorithm 1 vs. Algorithm 3	0.0001

From the results above we can see that algorithm 1 is statistically highly significant compared to algorithm 2, algorithm 3 statistically highly significant compared algorithm 2 but algorithm 3 is also statistically highly significant compared to algorithm 1. Therefore, we can reject the null hypothesis for all of the t-tests with all of them with a less than 1% probability that the results are produced by chance. The reason the p-value is so small is that we evaluated each of the ranking algorithms on fifty different job description and therefore there is high certainty that these results are correct and that there is significant evidence that algorithm 3 produces the best ranking.

6.3.4 Summary

In summary of the results of the technical evaluation, algorithm 3 produced the best results in the recall-precision graph, mean average precision(MAP) and P_5 value. These results were backed up with the paired t-test by showing there was less than 1% probability that these results were produced by chance and that there was a sufficient amount of job descriptions used to produce representative results. Therefore, based on these results there is significant evidence that algorithm 3 produces the best ranking for its use case.

Chapter 7

Conclusion

7.1 Summary

The aim of this project was to investigate whether jobs could accurately be represented using auto-generated tags in order to aid a job seekers recommendation system. This aim could be split up into two objectives, implementation of an application which can generate tags for a given job description and research into ways which we can identify relations between skills and job descriptions(Section 1.2). JobTagger is a prototype web application created for this project to satisfy the implementation of an application to generate skill tags for a given job description. It allows users to choose which algorithm they would like to use, submit a job description and receive the top five related skills to that job with some assurance that the skills are correct.

After investigating three different techniques (Section 5.1.2) for ranking algorithms we found that using Word2Vec word embedding to create a similarity score between a job description and a skill, paired with a relevance score for each skill using Wikipedia Pageviews produced the best ranking(Section 6). All three of the algorithms had their weaknesses, with the first algorithm having problems with finding specialist skills, the second algorithm not producing a broad range of skills and the third algorithm needing to find a good balance between similarity and relevance scoring. Although the third algorithm is not perfect it produced the best results during the evaluation with the highest preferred algorithm in the Amazon demo, producing the highest mean average precision value, recall precision rate and precision value (Section 6). The results from the evaluation was backed up with the paired t-test by showing there was less than 1% probability that these results were produced by chance and that there was a sufficient evidence that algorithm 3 produces the best ranking.

7.2 Reflection

This project has taught me greatly the challenges and complexity of creating and evaluation information retrieval systems. One of the key problems with evaluating these types of systems is that relevance is a subjective notion. It is very hard to decide which ranking is better using relevance as it is highly subjective, but we can make a subjective decision which ranking is most appropriate for the use case.

I believe that improving the underlying skill set will improve the ranking for all of the algorithms and will allow for a better evaluation. I believe that it is a garbage in, garbage out situation if the underlying skill set is not good enough, with a lot of the comments from the user evaluation saying how there is a lot of overlap with the suggestions. If the skill ontology was refined to ensure there are no duplications or overlaps and they use precise naming, the rankings of the algorithms can be improved. With saying that, skills are very hard to define

and possibly using tags rather than skills could produce a better result. For example, finding most significant terms in a job description to summarise it.

I believe the best technique to create an accurate representation of a job using auto-generates tags is to use a combination of Word2Vec to find similarity scores between a skill and a job description and a skill relevance scoring using Wikipedia Pageviews. This technique with an optimal skill set, a well balanced algorithm between similarity and relevance score, and a trained word embeddings on a large corpus of job descriptions and articles will create a sufficient ranking for this use case.

The key purpose for this system is to improve recommendations by using job skill tags to match with skills on user profiles. However, there are multiple business cases for this application. Some examples are:

- **Searching:** Improve/speed up job searching. By adding skills as metadata to each job, this can be used to improve the searching capabilities and also improve speed of the search. By having standardised tags this will help the search engine by increasing coverage.
- **Recommendations:** Taking skills into consideration for recommendations to find the best job for the user.
- **Trends:** These tags can be used to find trends in data. For example, they can be used to find the most common skills used for a job type.
- **Ranking/Improving:** Could be use to better quality of job descriptions by giving some suggestions to the writer of common trends. For example, "Your description only describes X amount of skills, the average is Y - Are you sure you are finished?"

There are also various systems which would benefit from this work. For example, an Alexa skill could use these tags to summaries a job meaning it would not have to read out the entire job description.

7.3 Future Work

If I was to continue working on this project, there are a number of tasks I would like to accomplish:

1. Investigate if improving the underlying skill ontology would improve rankings for each of the algorithms.
   (a) Using LinkedIn's technique of using Wikipedia to identify duplicate or overlapped skills.
   (b) Perform user evaluation to find short and precise names for each skill to be suitable to be used as tags on job cards.
2. Evaluate if skill tags can be used to improve job recommendations by creating a profile page for users and evaluate if job recommendations are improved by the introduction of skill tags.
3. Improve word embeddings by training them on a large corpus of job descriptions and articles to optimise vectors.
4. Add the ability for users to dismiss suggestions and implement a machine learning feedback loop to learn from these dismissals to optimise rankings.
5. Investigate whether the ranking algorithms work for all job types by extending the skill ontology to all job types.
6. Investigate whether the SKOS XML tree structure could be used to find more relevant skills.

Appendices

Appendix A

Running the Program

An example of running from the command line is as follows:

```
> mongod
```

In other terminal:

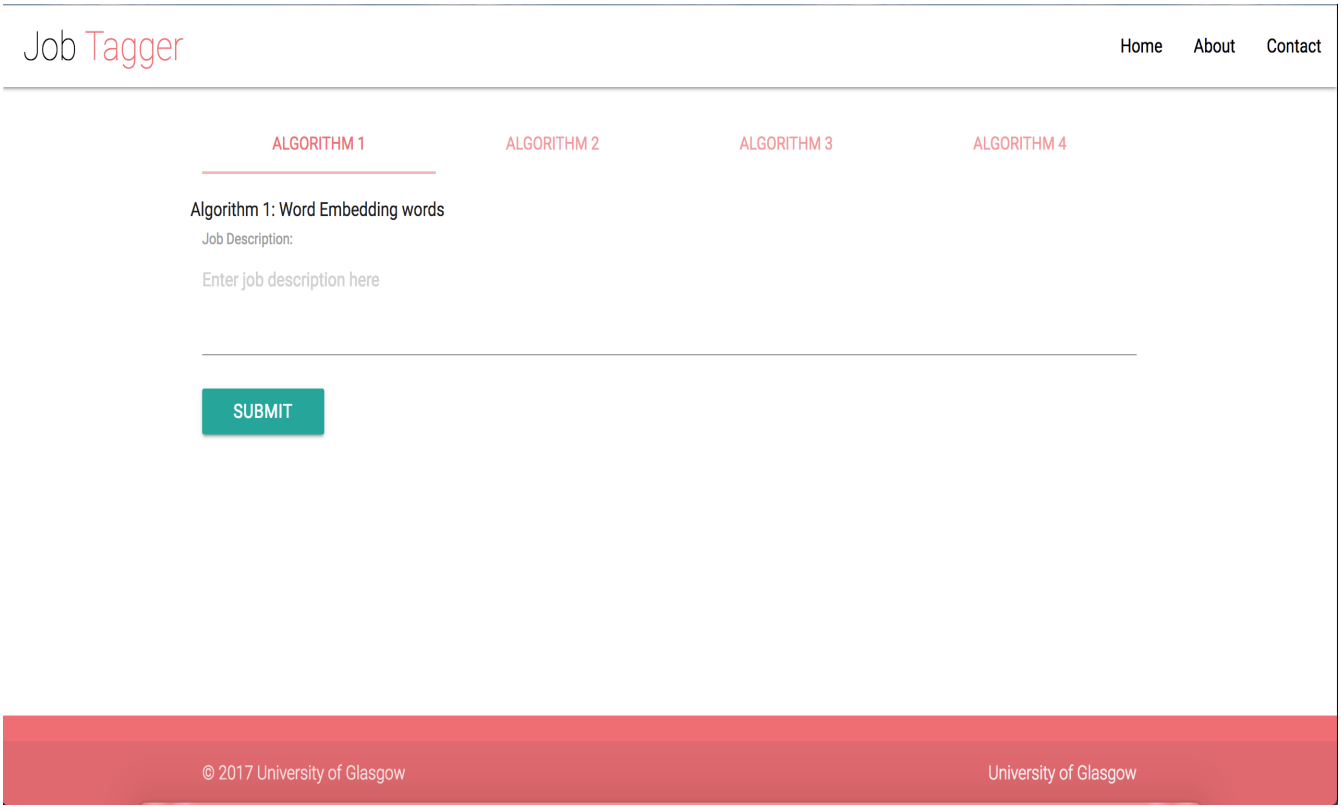
```
> npm install
> node index.js
```

Goto http://localhost:8080/ and it will take you to the index page of the website.

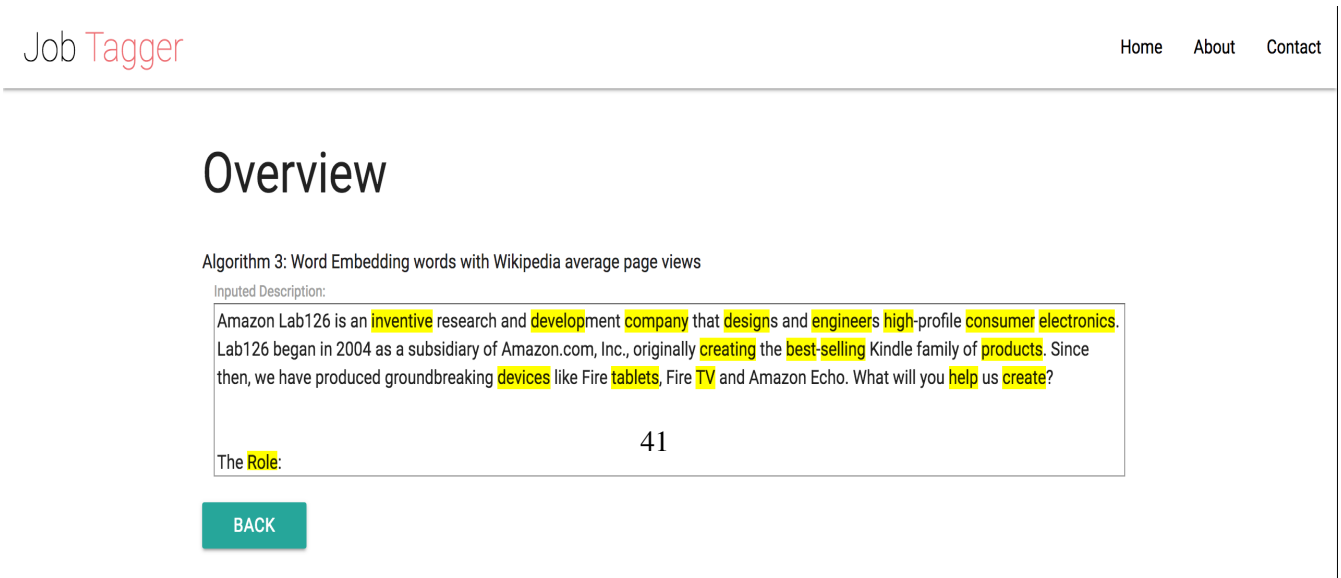
Appendix B

JobTagger Screenshots

Home page



Overview page 1: Shows job description with highlighted words used to identify skills



Overview page 2: Top five skills located below highlighted job description

Top 5 Skills

Click (?) to find out more about the skill

1. **Java**(2093) (?)(0.41984219740785067):
2. **Operating systems**(555) (?)(0.22791588329687335):
3. **Engineering**(1944) (?)(0.20152558993369177):
4. **Software development**(544) (?)(0.18287983873966043):
5. **Agile software development**(719) (?)(0.17895685920867904):

Overview page 3: List of words used to identify skills

Words Used

Words used to determine the skills needed

ability
agile
ambiguous
articulate
availability
basic
best
broad
build
building
center
centers
challenges
cloud
code
company
complex
computer
creating
critical
customer
customers
data

Appendix C

User Survey

Evaluation Survey

Job Tagger
Gregor Thomson - 2029108t

OVERVIEW & PURPOSE

This is a collaborative project with Amazon Development Centre Scotland to investigate whether auto-generated tags can accurately represent a job to improve job recommendation.

We are wanting to extend the job data Amazon holds by adding 'skills tags' to all current and future jobs. Job Tagger is a prototype system used to suggest skill for a given job description. Input a job description and Job Tagger will return five suggested skills needed for that job.

The purpose of this evaluation is to highlight the key problems with the prototype system, evaluate how well the system's suggestions are compared to human suggestions and to compare the three current ranking algorithms.

OBJECTIVES

1. Find possible improvement in the system
2. Compare alternative approaches
3. Evaluate the ranking algorithms
4. Does the system achieve the project objective

Looking at the following job description, what are the top five required skills?

Amazon Marketplace is now in UK, France, Germany, Italy, and Spain. We are growing fast, with customers across all 27 European states. Amazon's platform is the engine that powers Amazon's Marketplace businesses, and Sellers rely on this platform and our support to start selling on Amazon and to grow their business.

Amazon Marketplace enables millions of Sellers worldwide to list hundreds of millions of products and make inventory and price updates for inventory across dozens of different categories and languages. While constantly ingesting new selection, we also strive to continuously improve the quality of existing catalog created by Amazon Sellers by identifying defects and fixing them using automated technology solutions. You'll be

building cutting edge and highly distributed systems to support merchants around the world selling on Amazon. You will join a highly technical and entrepreneurial culture defining and building a selling experience to complement Amazon's world-class ecommerce websites. Selling on Amazon is one of the fastest growing businesses at Amazon.com with about half of all items currently sold originating from 3rd party merchants.

We are looking for software engineers with a strong sense of ownership and a passion for delivering creative solutions for complex problems on an unprecedented scale. As part of the team, you will be given the chance to have a significant impact on our systems, our business and most importantly, our customers as we take on significant challenges that can reshape the ecommerce industry.

Successful candidates must also be innovative, flexible, self-directed, and able to design and write high-performance, reliable, maintainable code. The ability to function at a very high level in a fast paced environment along with a team of very talented engineers is essential. If you enjoy working in a dynamic environment to deliver world class mission critical systems, this may be the career opportunity for you!

BASIC QUALIFICATIONS

- In-depth knowledge of Java or C/C++.
- Familiarity with Ruby, Perl, JavaScript, AJAX, XML/XSLT, SOAP, SQL, Oracle/Berkeley databases, caching technologies, web protocols, Web services.
- Understanding of Object-Oriented design and concepts
- Strong analytical skills
- BS or MS in Computer Science.
- 2+ years of industry experience.

PREFERRED QUALIFICATIONS

- 1+ years of industry experience developing large scale distributed systems
- Fluency in C/C++ or Java
- Proficient with Web technologies.
- Familiar with SOA and large scale design for 3-tier architecture
- Experience developing software in a Unix/Linux environment
- Excellent communication skills

Skill 1: _____

Skill 2: _____

Skill 3: _____

Skill 4: _____

Skill 5: _____

Section 2: Algorithm 1

Use the following description to evaluate algorithm 1.
Amazon Marketplace is now in UK, France, Germany, Italy, and Spain. We are growing fast, with customers across all 27 European states. Amazon's platform is the engine that powers Amazon's Marketplace businesses, and Sellers rely on this platform and our support to start selling on Amazon and to grow their business.

Amazon Marketplace enables millions of Sellers worldwide to list hundreds of millions of products and make inventory and price updates for inventory across dozens of different categories and languages. While constantly ingesting new selection, we also strive to continuously improve the quality of existing catalog created by Amazon Sellers by identifying defects and fixing them using automated technology solutions. You'll be building cutting edge and highly distributed systems to support merchants around the world selling on Amazon. You will join a highly technical and entrepreneurial culture defining and building a selling experience to complement Amazon's world-class ecommerce websites. Selling on Amazon is one of the fastest growing businesses at Amazon.com with about half of all items currently sold originating from 3rd party merchants.

We are looking for software engineers with a strong sense of ownership and a passion for delivering creative solutions for complex problems on an unprecedented scale. As part of the team, you will be given the chance to have a significant impact on our systems, our business and most importantly, our customers as we take on significant challenges that can reshape the ecommerce industry.

Successful candidates must also be innovative, flexible, self-directed, and able to design and write high-performance, reliable, maintainable code. The ability to function at a very high level in a fast paced environment along with a team of very talented engineers is essential. If you enjoy working in a dynamic environment to deliver world class mission critical systems, this may be the career opportunity for you!

BASIC QUALIFICATIONS

- In-depth knowledge of Java or C/C++.
- Familiarity with Ruby, Perl, JavaScript, AJAX, XML/XSLT, SOAP, SQL, Oracle/Berkeley databases, caching technologies, web protocols, Web services.
- Understanding of Object-Oriented design and concepts
- Strong analytical skills
- BS or MS in Computer Science.
- 2+ years of industry experience.

PREFERRED QUALIFICATIONS

- 1+ years of industry experience developing large scale distributed systems
- Fluency in C/C++ or Java
- Proficient with Web technologies.
- Familiar with SOA and large scale design for 3-tier architecture
- Experience developing software in a Unix/Linux environment
- Excellent communication skills

The algorithm produced relevant skills. Strongly Disagree - Strongly Agree

● ● ● ● ●

Are there any skills you felt were missed out?

Would you remove any skills? If yes, what?

Would you change the names of any of the suggested skills? If yes, what would you change it to?

Section 3: Algorithm 2

Use the following description to evaluate algorithm 1.
Amazon Marketplace is now in UK, France, Germany, Italy, and Spain. We are growing fast, with customers across all 27 European states. Amazon's platform is the engine that powers Amazon's Marketplace businesses, and Sellers rely on this platform and our support to start selling on Amazon and to grow their business.

Amazon Marketplace enables millions of Sellers worldwide to list hundreds of millions of products and make inventory and price updates for inventory across dozens of different categories and languages. While constantly ingesting new selection, we also strive to continuously improve the quality of existing catalog created by Amazon Sellers by identifying defects and fixing them using automated technology solutions. You'll be building cutting edge and highly distributed systems to support merchants around the world selling on Amazon. You will join a highly technical and entrepreneurial culture defining and building a selling experience to complement Amazon's world-class ecommerce websites. Selling on Amazon is one of the fastest growing businesses at Amazon.com with about half of all items currently sold originating from 3rd party merchants.

We are looking for software engineers with a strong sense of ownership and a passion for delivering creative solutions for complex problems on an unprecedented scale. As part of the team, you will be given the chance to have a significant impact on our systems, our business and most importantly, our customers as we take on significant challenges that can reshape the ecommerce industry.

Successful candidates must also be innovative, flexible, self-directed, and able to design and write high-performance, reliable, maintainable code. The ability to function at a very high level in a fast paced environment along with a team of very talented engineers is essential. If you enjoy working in a dynamic environment to deliver world class mission critical systems, this may be the career opportunity for you!

BASIC QUALIFICATIONS

- In-depth knowledge of Java or C/C++.
- Familiarity with Ruby, Perl, JavaScript, AJAX, XML/XSLT, SOAP, SQL, Oracle/Berkeley databases, caching technologies, web protocols, Web services.
- Understanding of Object-Oriented design and concepts
- Strong analytical skills
- BS or MS in Computer Science.
- 2+ years of industry experience.

PREFERRED QUALIFICATIONS

- 1+ years of industry experience developing large scale distributed systems
- Fluency in C/C++ or Java
- Proficient with Web technologies.
- Familiar with SOA and large scale design for 3-tier architecture
- Experience developing software in a Unix/Linux environment
- Excellent communication skills

The algorithm produced relevant skills. Strongly Disagree - Strongly Agree

○ ○ ○ ○ ○

Are there any skills you felt were missed out?

Would you remove any skills? If yes, what?

Would you change the names of any of the suggested skills? If yes, what would you change it to?

Section 4: Algorithm 3

Use the following description to evaluate algorithm 1.
Amazon Marketplace is now in UK, France, Germany, Italy, and Spain. We are growing fast, with customers across all 27 European states. Amazon's platform is the engine that powers Amazon's Marketplace businesses, and Sellers rely on this platform and our support to start selling on Amazon and to grow their business.

Amazon Marketplace enables millions of Sellers worldwide to list hundreds of millions of products and make inventory and price updates for inventory across dozens of different categories and languages. While constantly ingesting new selection, we also strive to continuously improve the quality of existing catalog created by Amazon Sellers by identifying defects and fixing them using automated technology solutions. You'll be building cutting edge and highly distributed systems to support merchants around the world selling on Amazon. You will join a highly technical and entrepreneurial culture defining and building a selling experience to complement Amazon's world-class ecommerce websites. Selling on Amazon is one of the fastest growing businesses at Amazon.com with about half of all items currently sold originating from 3rd party merchants.

We are looking for software engineers with a strong sense of ownership and a passion for delivering creative solutions for complex problems on an unprecedented scale. As part of the team, you will be given the chance to have a significant impact on our systems, our business and most importantly, our customers as we take on significant challenges that can reshape the ecommerce industry.

Successful candidates must also be innovative, flexible, self-directed, and able to design and write high-performance, reliable, maintainable code. The ability to function at a very high level in a fast paced environment along with a team of very talented engineers is essential. If you enjoy working in a dynamic environment to deliver world class mission critical systems, this may be the career opportunity for you!

BASIC QUALIFICATIONS

- In-depth knowledge of Java or C/C++.
- Familiarity with Ruby, Perl, JavaScript, AJAX, XML/XSLT, SOAP, SQL, Oracle/Berkeley databases, caching technologies, web protocols, Web services.

- Understanding of Object-Oriented design and concepts
- Strong analytical skills
- BS or MS in Computer Science.
- 2+ years of industry experience.

PREFERRED QUALIFICATIONS

- 1+ years of industry experience developing large scale distributed systems
- Fluency in C/C++ or Java
- Proficient with Web technologies.
- Familiar with SOA and large scale design for 3-tier architecture
- Experience developing software in a Unix/Linux environment
- Excellent communication skills

The algorithm produced relevant skills. Strongly Disagree - Strongly Agree

Are there any skills you felt were missed out?

Would you remove any skills? If yes, what?

Would you change the names of any of the suggested skills? If yes, what would you change it to?

Section 5: Review

Which algorithm do believe produced the best results?

- Algorithm 1
- Algorithm 2
- Algorithm 3

Any further comments, improvements or suggestions?

Bibliography

[1] Text REtrieval Conference (TREC) Overview. http://trec.nist.gov/overview.html, 2017. Accessed: 13 March 2017.

[2] Acm.org. Association for Computing Machinery. http://www.acm.org/, 2017. Accessed: 7 March 2017.

[3] Acm.org. The 2012 ACM Computing Classification System Association for Computing Machinery. http://www.acm.org/about/class/class/2012, 2017. Accessed: 7 March 2017.

[4] AgileBusiness. MoSCoW Prioritisation. https://www.agilebusiness.org/content/moscow-prioritisation-0, 2016. Accessed: 24 February 2017.

[5] Amazon. Amazon DynamoDB. https://aws.amazon.com/dynamodb/, 2017. Accessed: 1 March 2017.

[6] Balsamiq. Balsamiq mockups. https://balsamiq.com/products/mockups/, 2017. Accessed: 6 March 2017.

[7] Mathieu Bastian. LinkedIn Skills: Large-Scale Topic Extraction and Inference. ACM Conference on Recommender Systems, 2014. Accessed: 15 October 2016.

[8] James Bucanek. *Learn objective-c for java developers*, pages 355–356. Springer Nature, 2009.

[9] Robin Burke. Hybrid web Recommender systems. volume 4321 of *Lecture Notes in Computer Science*, pages 377–408. Springer, 2007. Accessed: 20 February 2017.

[10] G. Calvert. *The Handbook of Multisensory Processes (1st ed.)*, page 160. MIT Press., Cambridge, 2004.

[11] Pierre Carbonnelle. PYPL PopularitY of Programming Language. http://pypl.github.io/PYPL.html, 2016. Accessed: 7 March 2017.

[12] Chai. Chai Assertion Library. http://chaijs.com/, 2017. Accessed: 7 March 2017.

[13] DeepLearning4J. Introduction to Word2Vec. https://deeplearning4j.org/word2vec.html, 2017. Accessed: 23 February 2017.

[14] Alison Doyle. List of General Skills. https://www.thebalance.com/list-of-general-skills-2063753, 2016. Accessed: 7 March 2017.

[15] Carlos A. Gomez-Uribe. The Netflix Recommender System: Algorithms, Business Value, and Innovation. http://dl.acm.org/citation.cfm?id=2843948, 2016. Accessed: 20 March 2017.

[16] Google. Custom Search JSON/Atom API. https://developers.google.com/custom-search/json-api/v1/overview, 2017. Accessed: 8 March 2017.

[17] Thomas Gruber. A Translation Approach to Portable Ontology Specifications . Knowledge Acquisition, 1993. Accessed: 7 March 2017.

[18] Thomas Gruber. Ontology. http://tomgruber.org/writing/ontology-definition-2007.htm, 2009. Accessed: 7 March 2017.

[19] Handlebars. Handlebars.js. http://handlebarsjs.com/, 2017. Accessed: 9 March 2017.

[20] Karin Friberg Heppin. Test collections and the Cranfield Paradigm. https://spraakbanken.gu.se/sites/spraakbanken.gu.se/files/6IR12.pdf, 2012. Accessed: 16 March 2017.

[21] Rex B. Kline. *Principles and Practice of Structural Equation Modeling*, page 128. Guilford Publications., 2015.

[22] Neal Leavitt. Will NoSQL databases live up to their promise? IEEE Xplore document, 2017. Accessed: 1 March 2017.

[23] LinkedIn. LikedIn. https://www.linkedin.com, 2017. Accessed: 20 March 2017.

[24] Christopher D. Manning. *Introduction to information retrieval (1st ed.)*, pages 232–234. Cambridge University Press., New York, 2008.

[25] Christopher D. Manning. Introduction to Information Retrieval: Assessing relevance. https://nlp.stanford.edu/IR-book/html/htmledition/assessing-relevance-1.html, 2008. Accessed: 16 March 2017.

[26] Christopher D. Manning. *Introduction to Information Retrieval: Evaluation in information retrieval*, pages 164–165. 2008.

[27] Christopher D. Manning. Introduction to Information Retrieval: Information retrieval system evaluation. https://nlp.stanford.edu/IR-book/html/htmledition/information-retrieval-system-evaluation-1.html, 2008. Accessed: 16 March 2017.

[28] Materialize. About - Materialize. http://materializecss.com/about.html, 2017. Accessed: 9 March 2017.

[29] Materialize. Documentation - Materialize. http://materializecss.com/, 2017. Accessed: 9 March 2017.

[30] Matthew Mayo. Top NoSQL database engines. http://www.kdnuggets.com/2016/06/top-nosql-database-engines.html, 2017. Accessed: 1 March 2017.

[31] Mayor2.dia.fi.upm.es. Human Resources Management Ontology. http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/ontologies/99-hrmontology/, 2017. Accessed: 7 March 2017.

[32] Microsoft. Bing Search API. http://datamarket.azure.com/dataset/bing/search, 2017. Accessed: 8 March 2017.

[33] Tomas Mikolov. Efficient Estimation of Word Representations in Vector Space. pages 4–5, 2013. Accessed: 12 February 2017.

[34] Mochajs.org. Mocha - the fun, simple, flexible JavaScript test framework. https://mochajs.org/, 2017. Accessed: 7 March 2017.

[35] MongoDB. A basic introduction to Mongo DB MongoDB Node.JS driver 1.4.9 documentation. https://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html, 2013. Accessed: 1 March 2017.

[36] NodeJS. Node.Js. https://nodejs.org/en/about/, 2017. Accessed: 1 March 2017.

[37] The University of Texas. Research Areas — Department of Computer Science. http://www.cs.utexas.edu/research/areas, 2017. Accessed: 7 March 2017.

[38] Christopher Olah. Deep Learning, NLP, and Representations. http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/, 2017. Accessed: 8 March 2017.

[39] Adam Pasick. The magic that makes Spotifys Discover Weekly playlists so damn good. https://qz.com/571007/the-magic-that-makes-spotifys-discover-weekly-playlists-so-damn-good/, 2017. Accessed: 20 March 2017.

[40] Damian Rees. What is wireframing. http://www.experienceux.co.uk/faqs/what-is-wireframing/, 2015. Accessed: 6 March 2017.

[41] Paul Resnick. Recommender systems. volume 40 of *Communications of the ACM*, pages 56–58, 1997. Accessed: 17 February 2017.

[42] Francesco Ricci. Introduction to Recommender Systems Handbook. pages 1–2, 2011. Accessed: 12 February 2017.

[43] Kishori Sharan. *Learn JavaFX 8*, pages 419–434. Springer Nature, 2015.

[44] Kishori Sharan. *Learn JavaFX 8*, pages 419–420. Springer Nature, 2015.

[45] Smmry. Smmry.com. http://smmry.com/about, 2016. Accessed: 16 March 2017.

[46] Jennifer Stapleton. *Dynamic systems development method*, pages 28–29. 1997.

[47] StatsDirect. P Values (Calculated Probability) and Hypothesis Testing. http://www.statsdirect.co.uk/help/basics/p_values.htm, 2017. Accessed: 18 March 2017.

[48] Keith van Rijsbergen. Information Retrieval. http://www.dcs.gla.ac.uk/Keith/Chapter.7/Ch.7.html, 1979. Accessed: 15 March 2017.

[49] Maria Vultaggio. What's Amazon echo? How Alexa Works as A speaker and 'personal assistant'. 2014. Accessed: 23 February 2017.

[50] W3. SKOS Simple Knowledge Organization System Primer. https://www.w3.org/TR/skos-primer/, 2017. Accessed: 7 March 2017.

[51] Martin Wieczorek. *Software quality: State of the art in management, testing, and tools*, pages 26–27. Springer-Verlag, Berlin, 2001.

[52] Wikipedia. Wikipedia Pageview API. https://wikitech.wikimedia.org/wiki/Analytics/PageviewAPI, 2017. Accessed: 23 February 2017.

[53] WordCloud. Free online word cloud generator. www.wordclouds.com, 2017. Accessed: 20 March 2017.

[54] yourdictionary. Examples of Skills. http://examples.yourdictionary.com/examples-of-skills.html, 2017. Accessed: 7 March 2017.