UNIVERSITY OF AMSTERDAM

An Empirical Study of Document Similarity and Term Extraction Using Apache Spark

by

Baris Can Vural

Graduate thesis submitted in partial fulfillment for the degree of Master's

in the
Graduate School of Sciences
University of Amsterdam

July 2017

UNIVERSITY OF AMSTERDAM

Abstract

Graduate School of Sciences
University of Amsterdam

MSc

by Baris Can Vural

We tackle two problems in this study: Document similarity comparison among data science related CVs, job ads and competency documents; and term extraction from a collection of data science job ads. These problems were selected to help EDISON project update the frameworks that define Data Science as a profession. Apache Spark's machine learning module makes it convenient to implement different methods in a reasonable amount of time and compare their results. We use Spark's featurization methods such as HashingTF, Countvectorizer, and Word2Vec for extracting features from text for the purpose of document similarity comparison. We use Spark's FP Growth algorithm to extract frequent terms from a collection of documents for term extraction to capture the job market trend.

Contents

Chapter 1

Introduction

The term "Big data" is often associated with 3 Vs: *Volume*, *Variety* and *Velocity*. *Volume* refers to the massive size of the data, *Velocity* refers to the speed at which the data is coming in and going out and lastly *Variety* refers to the wide scope of data formats and sources. These characteristics of big data makes it crucial that parallel computing be utilized if valuable information from this data is to be extracted.

Apache Spark as a parallel processing platform of big data is gaining popularity since it was created in 2009. Its popularity can be attributed to its performance gains over other platforms in tasks such as iterative machine learning tasks and interactive querying of data[1] by keeping large amounts of data in-memory. As the platform got older its ecosystem grew to include a SQL module for processing intermediate data using SQL syntax, a machine learning library to construct powerful ML pipelines for analysing data and a graphx graph processing engine which leverages optimizations that pre-existing graph processing engines could not enjoy.

In this study, we will mainly tackle the problem of matching CVs[1] with job ads using document similarity measures. The machine learning module within Spark will be the main module that will be leveraged. The machine learning module or *MLlib* provides various techniques out of the box which makes it possible to implement different approaches in short amount of time and compare their results.

1.1 Motivation and Problem Statement

Data Science is a new, emerging field in research and industry. The definition of a data scientist ranges from being able to apply a few statistical algorithms to being able

[1]Curriculum Vitae

to perform data mining, data analysis and machine learning.The EDISON project [2] is designed to create a foundation for establishing the profession of Data Scientist for European research and industry. The EDISON Data Science Framework (EDSF) within the Edison project is a collection of documents that aims to define the Data Science as a profession.

EDSF consists of 4 parts: *Data Science Competence Framework (CF-DS)*, *Data Science Body of Knowledge (DS-BoK)*, *Data Science Model Curriculum (MC-DS)*, and *Data Science Professional profiles (DSP profiles)*. These documents have been developed to guide educators and trainers, employers and managers, and Data Scientists themselves. This collection of documents collectively breakdown the complexity of the skills and competences need to define Data Science as a professional practice.

Upon starting this study, the work of Spiros Koulouzis was the source of inspiration. He developed the EDISON Competencies Classification (E-CO-2)[3]. E-CO-2 is a distributed automated service aimed to perform gap analysis for Data Science profession. It makes use of the EDISON taxonomy to identify mismatches between education and industry. E-CO-2 also continuously collects data from job market to identify the trends and demands of the industry.

The E-CO-2 service was developed using the Hadoop MapReduce framework. This framework is one of the first popular big data processing platforms. However, it suffers from performance penalties such as loading data to disk for each task. It also makes it difficult to develop code since its limited API forces developers to think every problem as a *Map*(generating intermediary key/value pairs) and a *Reduce* (reducing the intermediate key/value pairs to a value).

The motivation for starting this study was aimed at achieving similar results with the E-CO-2 service but by using the Apache Spark platform. Apache Spark provides performance benefits over Hadoop MapReduce framework by keeping the data in memory of the cluster computers [1]. In machine learning tasks such as *logistic regression* where intermediate data is constantly being updated, Apache Spark performs 120x faster than Hadoop MapReduce[1].

Apache Spark's MLLib (machine learning module) makes it easy to use conventional machine learning algorithms. The API allows us to implement different methods quickly and compare their results in a short amount of time. In order to experiment with this module and help EDISON project improve the Data Science framework, we consider two problem statements.

One of the problem statements for this study is to use the aforementioned data science competencies defined by EDISON to match data science CVs with data science job ads.

Various document similarity measures and techniques are used to determine how similar two given documents are. The idea is to implement a system where users can see the most relevant data science CVs for a particular data science job ad.

Second problem statement we consider is implementing a version of E-CO-2 project's gap analysis for the data science profession. An analysis of the job market for mostly searched-for skills is useful for updating the competencies defined by EDISON. In a world where the needs of research and industry constantly evolve, it is crucial that EDISON's framework for defining the Data Science profession remains up-to-date and relevant. Therefore this study also includes a chapter (4) for extraction of common terms from a collection of job ads.

The dataset for this study was inherited from the E-CO-2 project. This dataset includes around 850 data science job ads from summer 2016. The dataset also has a collection of text documents specifying a specific competency; mostly related to data science. For example, a file named "predictive_analytics.txt" explains in human language what "predictive analytics" means and what a person who has this competency should be able to do.

In chapter (2), the reader will become familiar with the document similarity and term extraction techniques whose various implementations are explained in the following chapters (3) and (4).

Chapter 2

Document Similarity and Term Extraction

This chapter includes the theoretical background for document similarity and term extraction within the scope of this study. Document similarity techniques are used to compare documents such as job ads, CVs and competency text documents. The term extraction section will pertain to finding the most popular word or word groups within a collection of job ads to provide advisory information to the EDISON project so that the competencies defining the Data science profession could be updated.

2.1 Document Similarity

Document similarity is the metric that defines how similar two given texts are. The texts in question are written in human language but need to be analyzed by programs. In this study's scope, the documents that are to be compared are CVs, job ads, and competency texts - documents that specify a particular competency, mostly related to data science.

There are various techniques to measure document similarity such as TF-IDF and cosine similarity, which will be explored within the Apache Spark framework.

2.1.1 Featurization

When we talk about document similarity we mean how similar two documents are in terms of what they convey. Text documents, pictures, videos, speech do not mean the same to machines; machines need numbers to work on. Therefore, in order to be able to work on text documents for similarity measure we need a way of translating documents (CVs, job ads, competence documents) into vectors of numbers. This process of extracting vectors from documents is called *featurization* and the extracted vectors are called *feature vectors*. The feature vectors can be passed into a learning algorithm and the conclusion reached by the algorithm can be related to the document the feature vector belongs to. However, one has to be careful about picking the right features to extract. The extracted features must be relevant to the the task at hand.

Documents are made up of terms which have two qualities that help with the document's classification: semantic quality and statistical quality[4]. Semantic quality refers to what the term means or how much a term contributes to the content of the text. Statistical quality refers to term's ability to help classify a document in which the term appears. Within the scope of this study two different methods of featurization of text documents are explored: *Term Frequency - Inverse Document Frequency (TF-IDF)* and *Word2Vec*. *TF-IDF* is a method based on counting which highlights the statistical quality of terms while *Word2Vec* aims to create a vector space in which semantically closer words are located close to each other. Word2Vec aims to represent both the semantic and statistical quality of the terms.

2.1.1.1 TF-IDF Measure

Term Frequency - Inverse Document Frequency (TF-IDF)[5] is a way of weighing terms based on how important a term is to a document in a collection of documents. It is the value acquired by multiplication of *Term Frequency (TF)* and *Inverse Document Frequency (IDF)*. TF-IDF measure is used widely in information retrieval. 83% of text-based recommender systems in the domain of digital libraries use TF-IDF[6].

Term Frequency or TF is calculated by $TF(t, d)$ = number of times term t appears in document d. The assumption here is that if a term appears many times within a document, it must be important to that document. In order to calculate the term weights per document, we have to construct vocabulary vectors. Figure 2.1 explains this process. Vocabulary of words is collected in a vector and this vector is passed on to each individual document. Then, term weight values for this particular document are filled. This particular implementation of TF-IDF is based on word counting. We will see in chapter 3 that a hashing-based implementation of TF-IDF is available in Spark's machine learning module. This approach provides performance benefits.

If we only consider counting terms as an important metric we could attribute importance to words that occur very often across the whole corpus but in fact have very little meaning. Stop words such as "the", "has", etc. do not contribute to the overall meaning

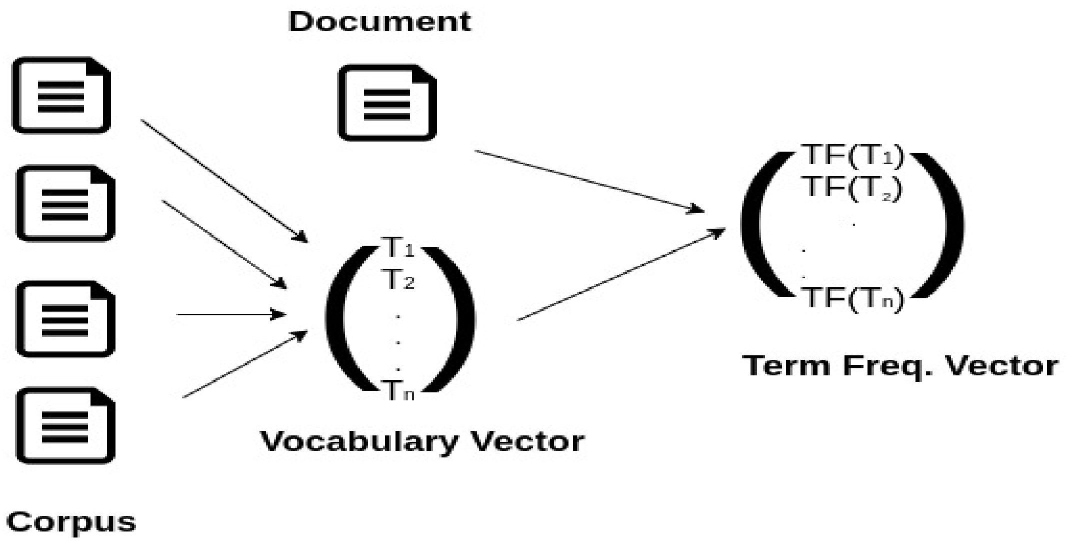


FIGURE 2.1: Term frequency calculation per document

of the document too much. This consideration does not have to be limited to stop words only; there could be terms that documents with common context might have a lot in common. For example, one would expect the word "data" to appear a lot in documents related to data science. Having this particular word does not distinguish a particular document having this word. Therefore, a down-weighting scheme for terms is used to curb the TF(t, d) of term t in document d if the term t appears many times across the corpus D.The following is the formula for this down-weighting:

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1} \tag{2.1}$$

where $|D|$ is the total number of documents in the corpus and $DF(t, D)$ is the number of times the term t appears in the corpus D. (+1) values are added to avoid the case of division by zero. Note that if a term appears in all documents the inside of the *log* becomes 1, the TF-IDF value becomes 0 and this reflects our assumption that this term has little importance. Finally combining both TF(t, d) and IDF(t, D) we acquire:

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

TF-IDF values are collected for each term in the corpus and for a given document. Collection of these values in a vector gives us the feature vector for that document.

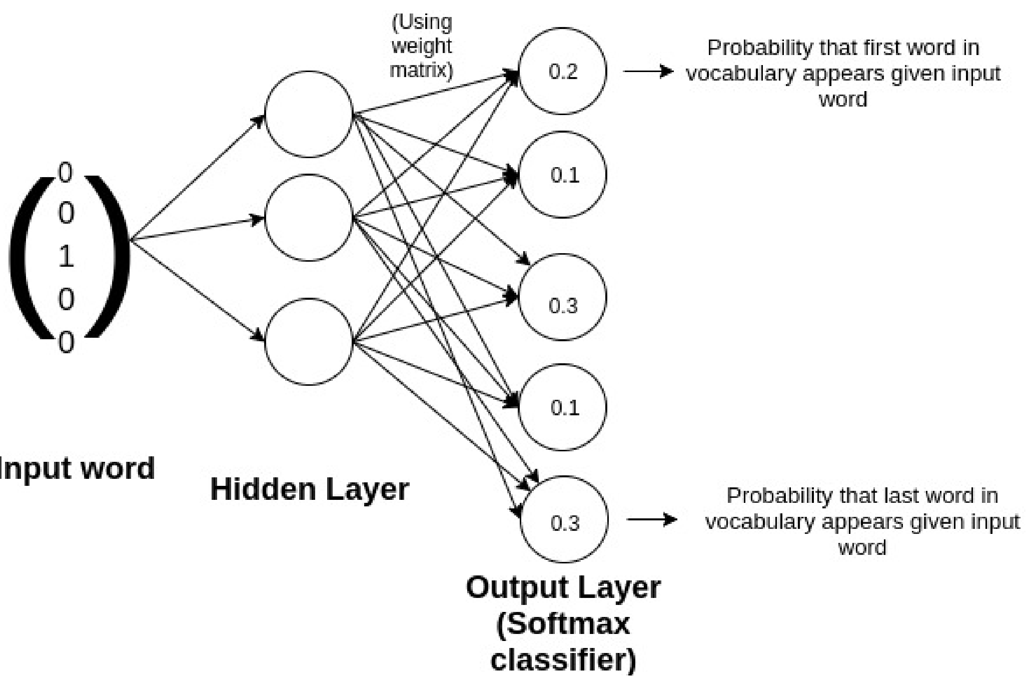

(Using weight matrix)
0.2
Probability that first word in vocabulary appears given input word
0.1
0
0
1
0
0
0.3
0.1
Input word
Hidden Layer
0.3
Probability that last word in vocabulary appears given input word
Output Layer (Softmax classifier)

Figure 2.2: Simple neural network that calculates word vectors

2.1.1.2 Word2Vec

Word2Vec[7] is a *word embedding*; meaning that it is a language model where words are mapped to vectors of real numbers. Words that have similar contexts will be close to each other in this vector space. This is useful in natural language processing (NLP) tasks as text documents have high dimensionality of one dimension per word. Reducing this dimensionality in a way that it accurately represents the word in the document makes the machine learning tasks more efficient.

Another advantage of having words that have similar contexts being closer to each other is that it makes the model generalize to test data from a given training data; i.e. it reduces overfitting to the training data.

Word2Vec model that is within the scope of this study makes use of neural networks with skip-gram model. To illustrate how word2vec with skip gram works, consider the figure 2.2, a simple neural network that has one input layer(a word), one hidden layer, and one output layer that is a softmax classifier. The reason for the softmax is to get the output probabilities to add up to 1. Moreover, the network layers do not use the logistic activation function.

Let us say we are working on a vocabulary of 5 distinct words and we represent each word as a vector of size 5 in which only the index corresponding to current word is 1 and the rest is 0 (i.e. one hot vector). Our input word [0, 0, 1, 0, 0] in this case corresponds to the third word in the vocabulary. What we want to train is that we want a neural network classifier that gives probabilities of every word in the vocabulary to appear in a given word's vicinity. By vicinity, we mean a certain training window k for which we consider each word. We are trying to find the context of given word. For example, given the word "data" what are the probabilities of seeing the other words in the vocabulary in the vicinity of this input word? Given the right training data we would expect the probability of "science" to be higher than probability of "orangutan" since we expect "data" and "science" to be situated closer to each other than "data" and "orangutan". But we would get probabilities for both words regardless. As the neural network encounters more words that are in the same training window k it back-propagates the errors to the previous layers and updates each layer's weight matrix.

The question remains however: why are we interested in these probabilities? In fact, we are not. We use the neural network as a means to calculate something we want. Training the neural network for the task of calculating probabilities of a word's vicinity helps us acquire the weight matrix of the hidden layer in the neural network. This weight matrix happens to be the matrix whose rows are *word vectors* of the words in the vocabulary. This weight matrix is the matrix that maps the hidden layer to the output layer. The dimensions of this weight matrix is 5x3 which is vocabulary size x feature size per word. In our simple example this feature size is 3 for simplification. Higher feature size will make the model more accurate. However, computational costs will also increase as this number increases.

Once the weight matrix of the hidden layer is calculated, we have the word vectors for our document. Once these word vectors are acquired we can construct the feature vector of the document they are in. One simple way to do this is to average all the word vectors in a document which is what Apache Spark's MLLib's implementation does to acquire document vectors from word vectors [8]. There are more sophisticated methods of acquiring phrase, sentence, or paragraph vectors[9] but within the scope of this study the way of acquiring the document vector from individual word vectors is to take the average of all the word vectors.

2.1.2 Cosine similarity as a similarity metric

The featurization is the process of acquiring vectors from documents and TF-IDF and Word2Vec are the methods we use to acquire feature vectors from documents. In order to compare two documents, it is necessary to compute these documents' vectors' similarity.

Cosine similarity is used as a similarity metric for comparing two documents. It has proven to be a robust metric for scoring the similarity between two strings[10]. It measures the cosine of the angle between two vectors i.e. the divergence between them.

Cosine of 0 is 1, and cosine of every other angle is smaller than 1. Therefore this metric is a measure for orientation of vectors and not their magnitude. The following is the formula for cosine similarity for vectors A and B with θ as angle between them:

$$\text{cosine similarity(A, B)} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}} \tag{2.2}$$

In this study mainly Python programming language was used and the calculation of cosine similarity is done using the scipy[11] library. In this library *cosine distance* calculation is provided as a method. Cosine distance and cosine similarity give us similar information and their conversion is given as follows:

$$(\textit{cosine similarity}) = 1 - (\textit{cosine distance}) \tag{2.3}$$

In the following chapters cosine distance will be used as the similarity metric. Higher cosine distance will imply a lower cosine similarity.

2.2 Term Extraction

In the context of this study, term extraction refers to the process of extracting meaningful word or word groups from a collection of documents. The documents in question are limited to the data science job ads in our dataset. Terms that appear in many job ads could signify importance in the job market. If we encounter the term "machine learning" across many job ads, we may conclude that this is an important skill to have if one is looking for a data science job in the industry. Moreover, it might also be useful to know which terms appear together in many documents. If the terms "machine learning" and "data mining" appear together one could reach a conclusion that data science job ads are becoming increasingly more development/programming oriented. Our goal is being able to provide such popular terms to the EDISON project so that the definition framework for Data Science could be updated in accordance with the job market's requirements.

2.2.1 Apriori Algorithm

In determining which terms occur most frequently and which group of terms appear together most frequently, an algorithm to consider is the Apriori algorithm [12]. Apriori

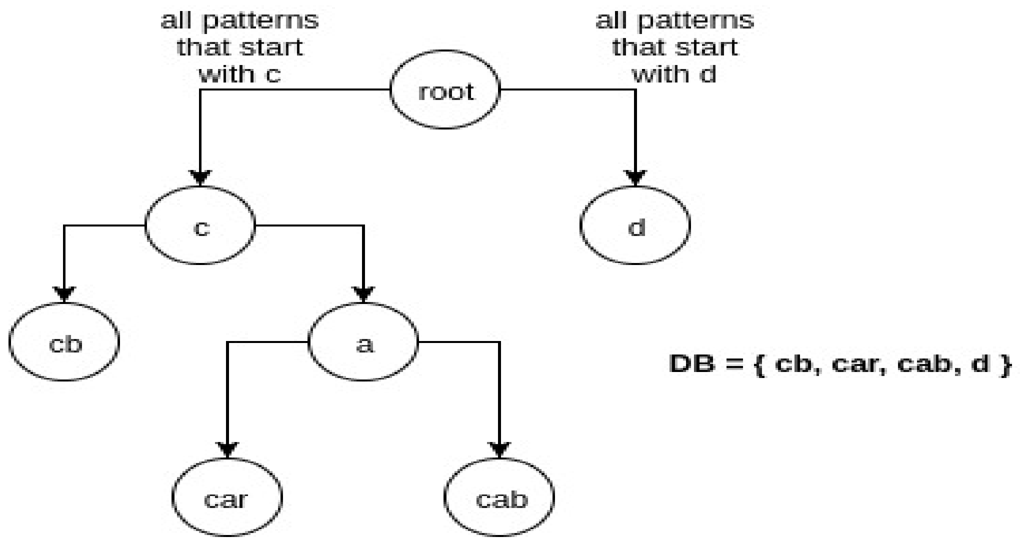

all patterns that start with c
root
all patterns that start with d
c
d
cb
a
DB = { cb, car, cab, d }
car
cab

FIGURE 2.3: Example of a prefix tree for a given database (DB)

algorithm can be best summarized with an analogy about a supermarket. Consider a supermarket's database which keeps records of which items are bought. In this database, one could look at which items are bought in the same particular time frame to determine the association between the items. For example, a supermarket may notice that on Friday nights, males who are older than 30 who buy baby diapers also buy beers at the same time. This fact could prompt the officials of the supermarket to place these items close to one another in the store to increase profit or it might help with the planning of the acquisition of the said items from their respective vendors. Mining association between items in a transaction database (records of which items are bought) can help the supermarket's business. In essence, apriori algorithm identifies frequent items in a transaction database and this helps with capturing trends in the database.

One can think of text documents as transaction databases if words that make up the documents are thought of as the items in our supermarket analogy. We could run the Apriori algorithm on the raw text of all the job ads in our data set while treating a single document as a set of transactions that occurred in a unit time frame. We should expect that the words "machine" and "learning" appear together frequently since our data set is made up of data science job ads.

In Apriori algorithm, frequent item sets in a transaction database are mined iteratively. The idea is that subsets of a frequent item set will also be frequent. For example, if the item set $\{"machine", "learning"\}$ is a frequent item set with number of elements k=2, then item sets $\{"machine"\}$ and $\{"learning"\}$ with $k = 1$ will be at least as frequent as their superset. Therefore, apriori algorithm follows a bottom up approach to find the frequent item sets with k many elements. Algorithm starts with $k = 1$ and expands the frequent item sets with possible additions to reach level $k + 1$. The algorithm

Table 2.1: List of Documents for Construction of FP Tree

ID	**Words in Document**	**Words Ordered by Frequency**
1	machine learning business	machine, learning, business
2	business analytics	business, analytics
3	useful machine learning	machine, learning, useful

will search in the transaction database every possible frequent item set combination with $k+1$. For example, if the algorithm encounters the item sets of level $k = 1$ $\{"machine"\}$ and $\{"learning"\}$, it will try to see if $\{"machine","learning"\}$ $(k = 2)$ is also in the transaction database. It will also try to search for non-existent item sets such as $\{"machine","orangutan"\}$, given that the word "orangutan" appears at least once in some document. This stage in the algorithm is called *candidate generation* and it is a performance drawback for the Apriori algorithm. For example, if there are 10^4 many frequent $k = 1$ item sets, the Apriori algorithm will need to generate more than 10^7 many $k = 2$ candidates and accumulate and test their occurrence frequencies[13].

In this study, in order to overcome the performance drawbacks of the Apriori algorithm, *Frequent Pattern Growth (FP Growth)* algorithm was used.

2.2.2 FP Growth Algorithm

Frequent Pattern Growth algorithm is an algorithm that finds frequent item sets in transaction databases, just like the *Apriori* algorithm which was discussed in chapter 2.2.1. Apriori algorithm suffers from generating candidate item sets and checks for those candidate item sets' existence in the transaction database. This is a huge performance drawback since the database has to be searched over and over for every candidate. *FP-growth* algorithm overcomes this problem by using a *FP tree* data structure to store the frequent item sets. The *FP tree* is a *prefix tree* (also called *radix tree*) data structure which is used in data compression. In figure 2.3 we can see an example of such a prefix tree on a small database of items. Note that representing overlapping patterns in this data structure reduces our search space by putting the overlapping patterns under the same subtree. *FP-growth* greatly reduces the search time within the database by recursively traversing the *FP tree* data structure[13].

To explain how the FP tree is constructed, let us consider the table 2.1 as a list of documents containing words. We make one pass over the whole corpus to determine which words are most frequent.For example, the word "machine" appears in documents with ID 1 and 3 therefore it's frequency is 2. Then, within each document we put the words in sorted order with respect to each word's frequency across the whole corpus. In column 3 of the table 2.1 words in sorted order are given for convenience. To construct

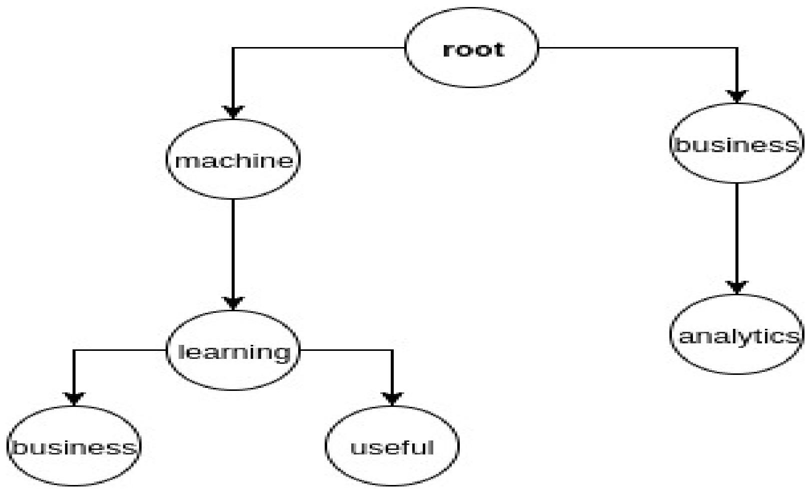


FIGURE 2.4: Resulting FP tree from the list of documents

the tree, we start from document with ID 1 and construct the leftmost branch of the tree which is illustrated in figure 2.4. This branch contains the words {”machine”, ”learning”, ”business”}. We move on to processing the document with ID 2. The first item in the sorted list is ”business” however we do not have a node connected to the root named ”business” at this point. So we create a node connected to the root named ”business”. Next, we move on to the second item in document ID 2 which is ”analytics”. This word becomes a child node of ”business” since they appear in the same document. Next, we move on to document ID 3. The first word in the sorted list is ”machine”. The root node of the tree already has a node named ”machine” so we move on to this node. Next word in the list of document ID 3 is ”learning” and we see that this node also already exists therefore we move to this node. Lastly, the last item to be considered is ”useful” and there is no node named as such yet. Therefore we create a node named ”useful” and make it the child node of ”learning”. We have now constructed our FP tree. Note that overlapping patterns such as ”machine learning” are represented only once in the tree.

In FP-growth algorithm, the database (corpus in our case) is traversed exactly twice. In the first pass, we determine the frequencies of each word in the corpus and generate the sorted lists such as column 3 in table 2.1. This has the benefit of finding common patterns when documents are processed and the FP tree is constructed. For example, the documents with IDs 1 and 3 have the overlapping pattern ”machine learning” which became apparent after sorting the terms by their frequencies. This overlap was not apparent before sorting as the first word in document 3 is ”useful”. In the second pass over the database (corpus), we traverse the FP tree recursively and insert a node in the tree as child nodes of the overlapping term nodes. We also increment the count of the nodes we pass through to remember later on that these nodes are a part of a frequent pattern. In the explanation of the construction of the FP tree, an example to this would be the insertion of the node ”useful”.

FP growth algorithm performs better than Apriori algorithm because it avoids creating candidate item sets and searching for these item sets. Construction of the FP tree greatly reduces the search time of frequent patterns. Another benefit of using a tree data structure is the ability to parallelize the task, with *partitioning* based approaches such as *divide and conquer*. In Apache Spark's FP growth implementation, this approach is followed. In this *Parallel FP-growth algorithm*, constructed FP trees are distributed among workers[14][15].

Chapter 3

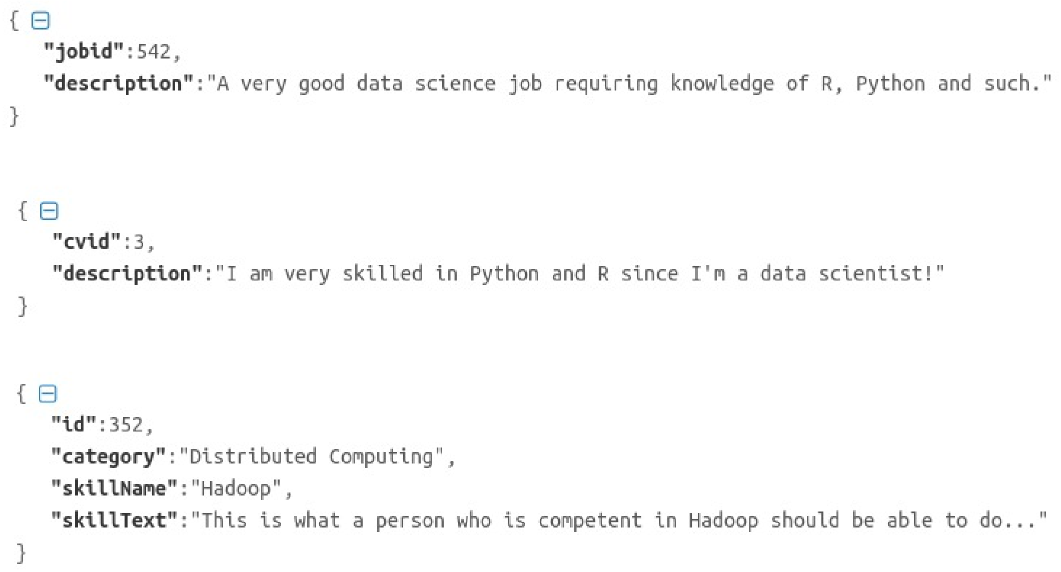
Implementations of CV - Job Ad Matching

In chapter 2, it was explained that the method for comparing two text documents is to compare these documents' feature vectors using the cosine distance metric. An implementation of this concept was developed to solve the problem of matching CVs with job ads. The documents being compared are the text content of job ads and CVs. 26 CVs were collected from the web to be compared with 624 job ads in the dataset that was inherited from the EDISON competencies project. In order to be able to process these documents, scripts were written to prepare them for Spark's processing. After this pre-processing document structures for jobs, CVs and competency documents are structured as they are presented in figure 3.1. Note that we will be performing document similarity between "description" fields of jobs and CVs and "skillText" field for competency documents. Feature vectors for these texts are extracted and compared with cosine distance metric.

This chapter includes details of the implementations of document similarity between job ads and CVs.

3.1 Document Similarity Visualization Tool

In order to evaluate the document similarity techniques, a visualization tool was developed to quickly see the differences between different methods. This is the end-product of this study and also the very means by which we evaluate the results of different approaches. When approaching the problem of finding the best CVs for a particular job ad, two different scenarios were considered. Scenario one: given a particular job ad,



```
{
   "jobid":542,
   "description":"A very good data science job requiring knowledge of R, Python and such."
}
```

```
{
   "cvid":3,
   "description":"I am very skilled in Python and R since I'm a data scientist!"
}
```

```
{
   "id":352,
   "category":"Distributed Computing",
   "skillName":"Hadoop",
   "skillText":"This is what a person who is competent in Hadoop should be able to do..."
}
```

FIGURE 3.1: Structure of a job ad, a CV and a competency document in json format

what are the best CVs for this job ad and how well these CVs do with respect to certain competencies? By "competencies" we mean the competency text documents we mentioned in chapter 1.1 that describe what a person having a specific competency should be able to do. We let the user decide which competency documents should be used for evaluating the CVs. For scenario two, the user of the system do not have to specify the competency texts. Instead, EDISON[2] competency texts are used automatically to display results. For both scenarios, when deciding which CVs are best suited for a particular job ad the cosine distance measurement was the value by which the CVs were sorted.

In Figure 3.2 the document comparison logic is presented. Note that both job ads and the CVs are compared with competency documents. The idea is to compare the job ads and CVs with respect to their comparison to a third text document: the competency document. The similarity between job ads' and CVs' cosine distance results with respect to the competency document can give us an idea about how well a CV is suited for a particular job ad.

In chapter 2, it was mentioned that there are different featurization methods which were used to acquire vectors from documents. One of these different featurization methods may be selected in the visualization tool. For example the user may select one of the TF-IDF based featurization methods and comparison between CVs and the job ad will be based on the selected featurization method. The following subsection will give information about use cases of the visualization tool, without touching upon the details of each featurization method's inner workings. Details of each of the featurization methods will be explained further in this chapter.

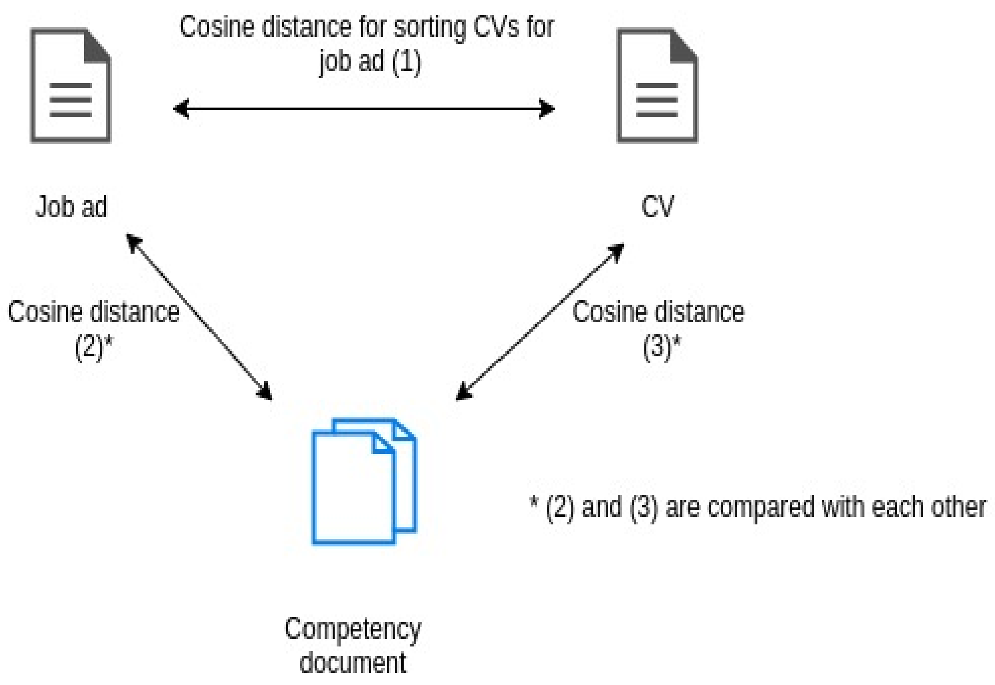

Cosine distance for sorting CVs for job ad (1)
Job ad
CV
Cosine distance (2)*
Cosine distance (3)*
* (2) and (3) are compared with each other
Competency document

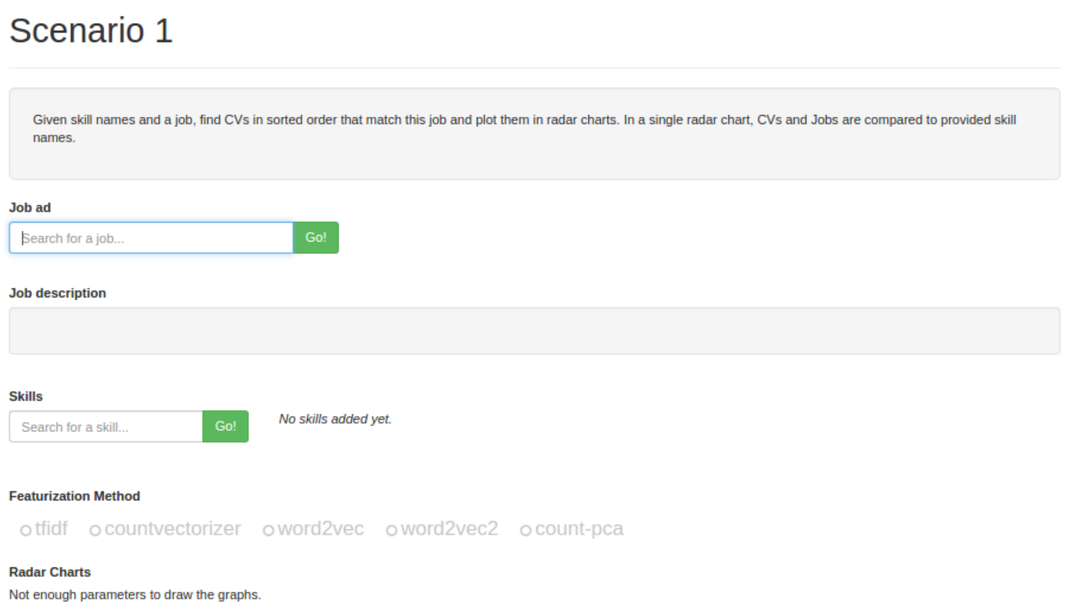
FIGURE 3.2: Document comparison logic

3.1.1 Visualization Tool Use Cases

In figure 3.3, the "scenario 1" page for CV and job matching page is presented. Considered scenario is the following: "Given skill names and a job, find CVs in sorted order that match this job and plot them in radar charts. In a single radar chart, CVs and Jobs are compared to provided skill names." User provides a job id, and 5 competency document names, and also selects one of the featurization methods. Inner workings of these featurization methods will be explained further in this chapter.

An example walk-through of the system is as follows: User provides 344 as job id and the raw text of the job ad is displayed in the box underneath. This text is useful for judging the results of different methods by reading the job ad and making a judgment ourselves. User provides the names of the competency documents against which the jobs and the CVs will be compared. As mentioned in chapter 1.1, these competency texts could be defined by EDISON or anyone else. Finally, user selects one of the featurization methods to see the comparison results which are displayed as radar charts.

In figure 3.4, an example of filled parameters are presented. After these parameters are filled, the tool will show us the CVs in sorted order based on cosine distance. In this particular example some of the names of the competency texts are Hadoop and Matlab. These are in fact text documents specifying in human language what a person having these competencies should be able to do.



Scenario 1

Given skill names and a job, find CVs in sorted order that match this job and plot them in radar charts. In a single radar chart, CVs and Jobs are compared to provided skill names.

Job ad

Search for a job... Go!

Job description

Skills

Search for a skill... Go!

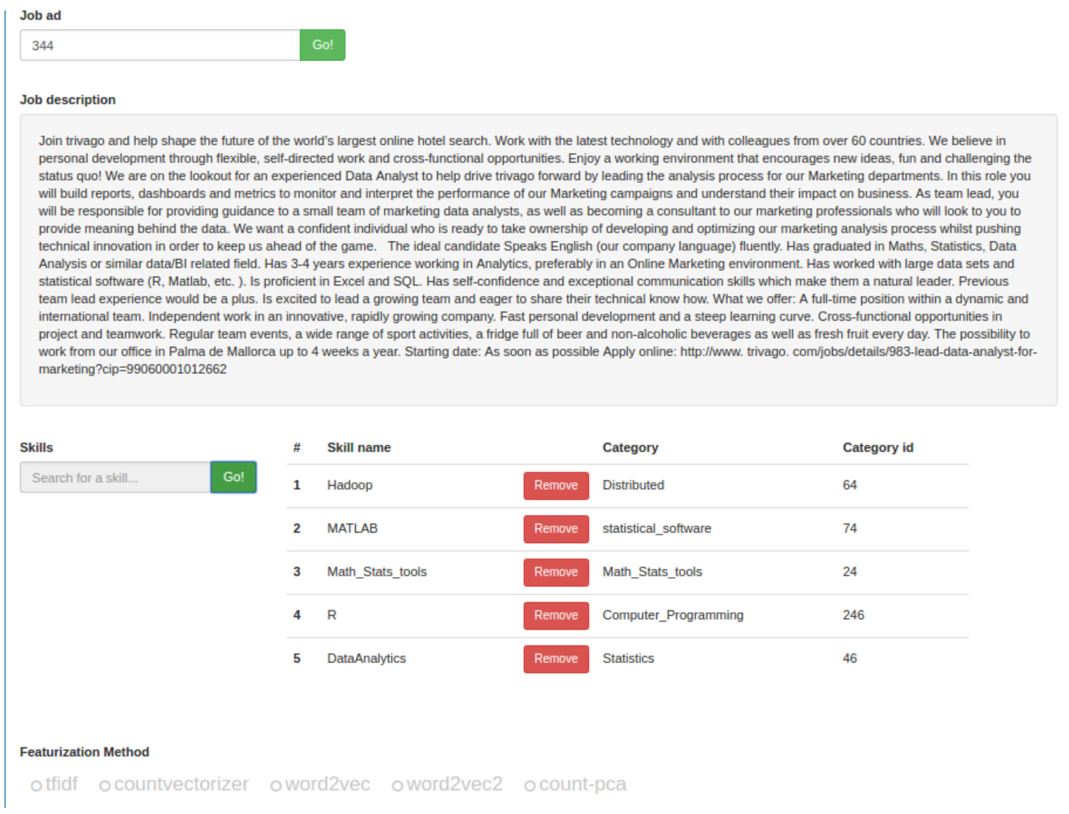
No skills added yet.

Featurization Method

Radar Charts

Not enough parameters to draw the graphs.

FIGURE 3.3: Visualization tool - Scenario 1



Job ad

344 Go!

Job description

Join trivago and help shape the future of the world's largest online hotel search. Work with the latest technology and with colleagues from over 60 countries. We believe in personal development through flexible, self-directed work and cross-functional opportunities. Enjoy a working environment that encourages new ideas, fun and challenging the status quo! We are on the lookout for an experienced Data Analyst to help drive trivago forward by leading the analysis process for our Marketing departments. In this role you will build reports, dashboards and metrics to monitor and interpret the performance of our Marketing campaigns and understand their impact on business. As team lead, you will be responsible for providing guidance to a small team of marketing data analysts, as well as becoming a consultant to our marketing professionals who will look to you to provide meaning behind the data. We want a confident individual who is ready to take ownership of developing and optimizing our marketing analysis process whilst pushing technical innovation in order to keep us ahead of the game. The ideal candidate Speaks English (our company language) fluently. Has graduated in Maths, Statistics, Data Analysis or similar data/BI related field. Has 3-4 years experience working in Analytics, preferably in an Online Marketing environment. Has worked with large data sets and statistical software (R, Matlab, etc.). Is proficient in Excel and SQL. Has self-confidence and exceptional communication skills which make them a natural leader. Previous team lead experience would be a plus. Is excited to lead a growing team and eager to share their technical know how. What we offer: A full-time position within a dynamic and international team. Independent work in an innovative, rapidly growing company. Fast personal development and a steep learning curve. Cross-functional opportunities in project and teamwork. Regular team events, a wide range of sport activities, a fridge full of beer and non-alcoholic beverages as well as fresh fruit every day. The possibility to work from our office in Palma de Mallorca up to 4 weeks a year. Starting date: As soon as possible Apply online: http://www. trivago. com/jobs/details/983-lead-data-analyst-for-marketing?cip=99060001012662

Skills

Search for a skill... Go!

#	Skill name		Category	Category id
1	Hadoop	Remove	Distributed	64
2	MATLAB	Remove	statistical_software	74
3	Math_Stats_tools	Remove	Math_Stats_tools	24
4	R	Remove	Computer_Programming	246
5	DataAnalytics	Remove	Statistics	46

Featurization Method

FIGURE 3.4: Visualization tool - Scenario 1 - Filled parameters

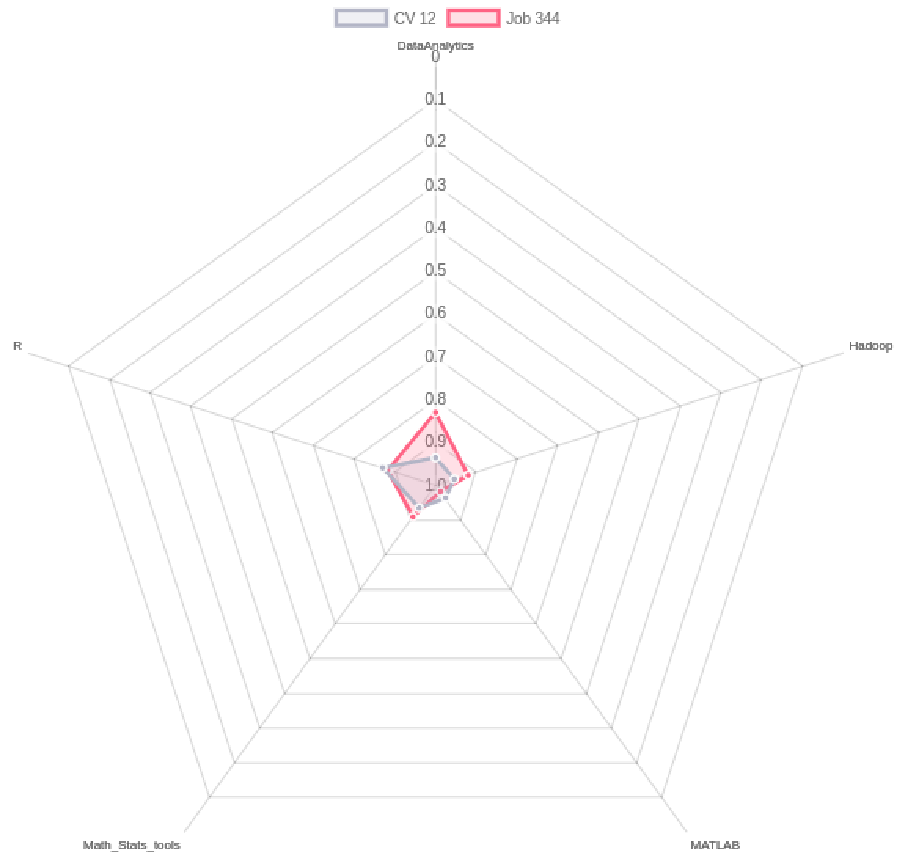


FIGURE 3.5: Visualization tool - Scenario 1 - Radar chart for a CV

In figure 3.5, a radar chart for one of the CVs is presented. The pink area covered by the job represents this particular job's document similarity with respect to the five competency documents whose names were provided by the user. Likewise, the blue area represents this CV's document similarity with the same competency documents. The idea is that a larger area covered implies a more competent CV. Looking at the figure 3.5 we can say that CV is slightly lacking when it comes to "data analytics" since the job (pink area) scores a higher similarity with the "data analytics" competency document than CV does. When it comes to "Matlab" CV document scores a higher similarity score than the job ad. From a job recruiter's perspective, looking at these radar charts can give us an idea about which CVs to pick for a particular job ad. However, one must not forget that these document similarity scores and charts and their interpretations are only as good as the text documents that represent them.

"Scenario 2" for the CV and job matching was to use the competency documents created by EDISON[2]. In this scenario, the user do not have to specify the names of the competency documents. An example of a radar chart drawn this way is given in figure 3.6.

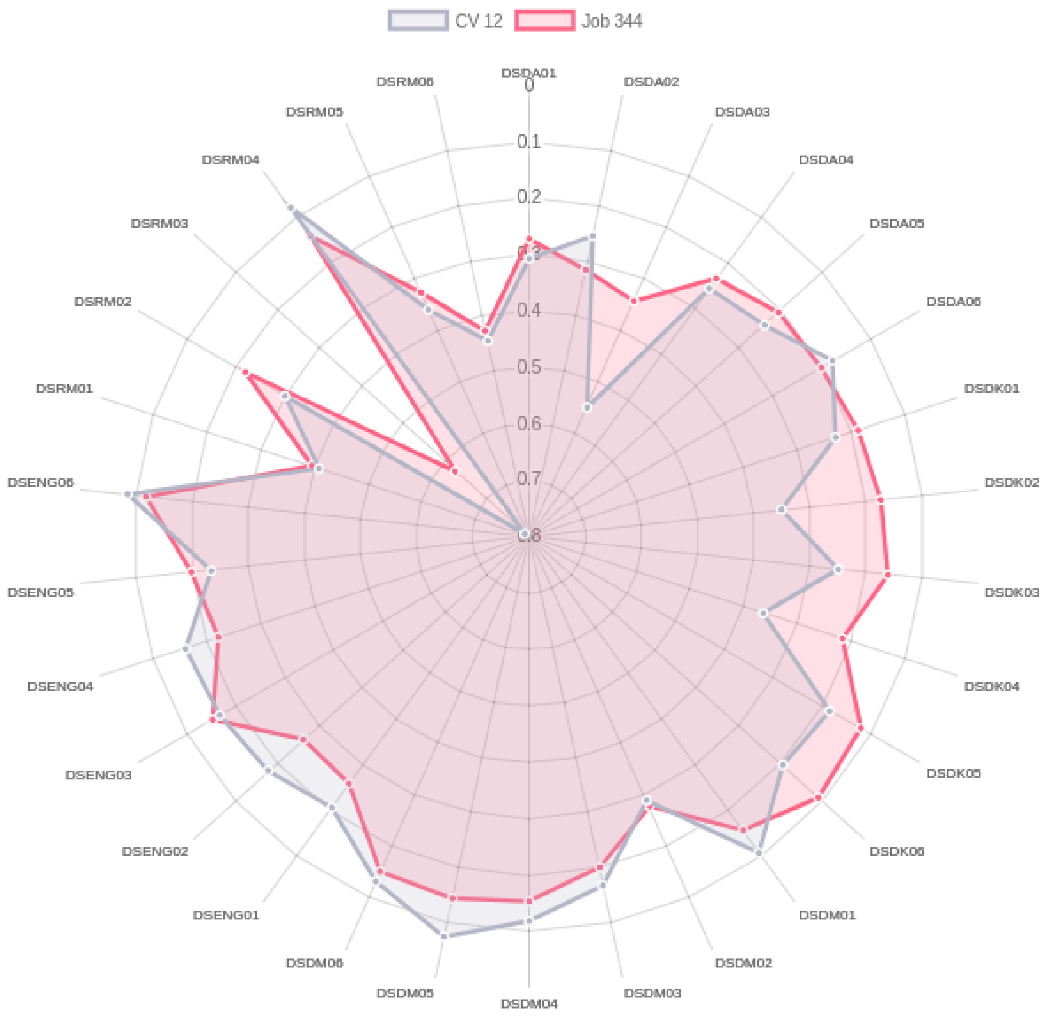


FIGURE 3.6: Visualization tool - Scenario 2 - CV compared against EDISON competencies

The use cases described above exemplify how a user can interact with the visualization tool to see the job ad - CV comparisons.In the figures, ”word2vec” featurization method was selected but this is only one of the methods implemented. The following subsections will describe the inner workings of the featurization methods.

3.1.2 Visualization Tool Architecture

The CV - job ad matching system consists of many moving parts. The document similarity calculation takes place via Spark. However, to visualize the data additional technologies and tools were needed. In figure 3.7, the architecture overview of the visualization tool is presented. The numbered texts signify the processes and their respective order. Firstly, the raw text documents are made ready for Spark's processing. This raw text documents are job ads, CVs, and competency texts. Spark's default parsing is reading input lines with endline character as delimiter. Moreover, Spark's dataframes API allows us to provide JSON input. Therefore in the text pre-processsing phase, the raw text is converted into JSON format and written such that there is one JSON object per line. Second process in the figure is running of the Spark jobs. Details of these jobs' implementation is explained in the implementations part further in this chapter. Running the Spark jobs produces output in CSV format which brings us to the third process in the figure: the database writer script. This python script writes the output of the Spark job files into a MongoDB instance which is running locally. Next process is the server script serving the data within the MongoDB to an Angular web application which visualizes the data produced by the Spark jobs.

The input text documents are fed to the Spark from the local file system as opposed to a distributed file system such as HDFS. Reason for this is that the whole system is running locally in a single computer since the data at hand is not large enough and Spark jobs take little time to complete.

When deciding on which technologies to use the requirements were considered. Python was chosen as the main programming language since it is easy to construct powerful coding statements using just a few lines such as array manipulation and duplicate elimination in single line of code. Python also makes it very easy to get started with Spark and Spark code in Python is very readable. Moreover, Spark's Python API has almost all the functionality Java and Scala enjoys.

When it comes to visualizing the data Spark produces, Angular front-end web framework was chosen for a few reasons. The idea for evaluating results of different methods was to see results without page reload. In that regard Angular's out-of-the-box data binding comes in handy. Moreover, Angular framework is highly modular, readable, and reputed to be one of the latest technologies. Lastly, the choosing of MongoDB as the database technology is based on the ability to insert documents in a non-pre-defined structure. As this study evolved requirements changed every week and relational databases with rigid schemas would have been harder to work with.

3.2 Spark's available featurization methods

In chapter 2, it was mentioned that there are two main methods of extracting features from documents for comparison. These are TF-IDF based methods and Word2Vec. In this study, Spark's available featurization methods were utilized therefore implementations of the said methods were explored within the capabilities of Spark.

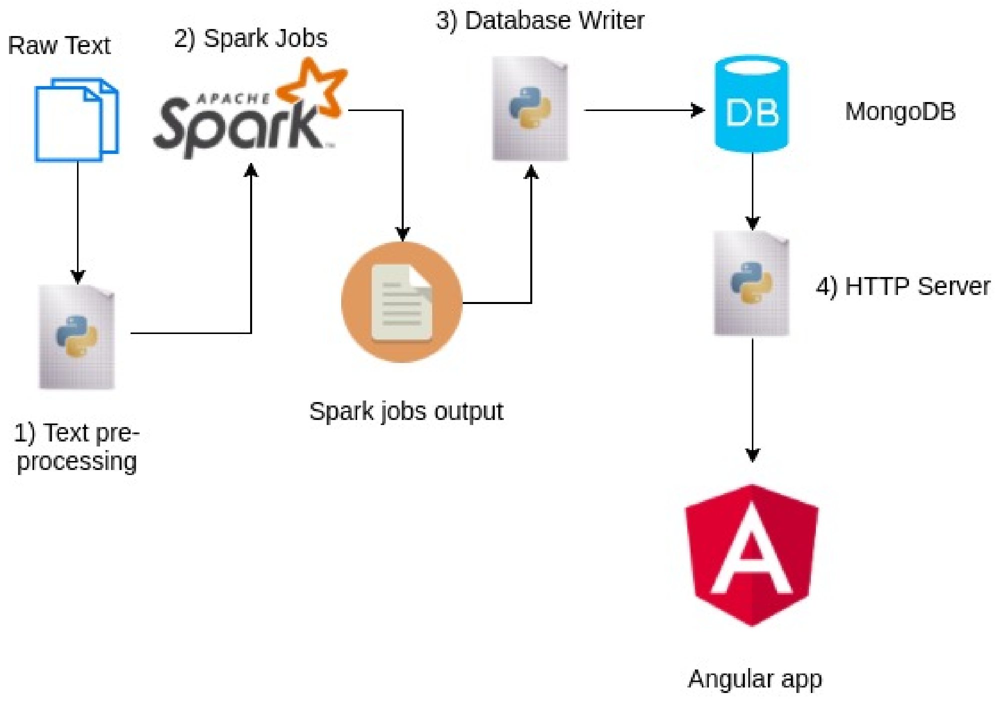


FIGURE 3.7: Visualization tool - Architecture

HashingTF, Countvectorizer and Word2Vec are the names of the available featurization methods in Spark. HashingTF and Countvectorizer pertain to TF-IDF based featurization. HashingTF is based on a hashing algorithm while Countvectorizer relies on counting terms. Word2vec on the other hand is word embedding which is a seperate method. In this section Spark's methods will be explained and in the section following that their specific implementations in the visualization tool will be covered.

3.2.1 HashingTF

HashingTF is one of the Spark's available featurization methods. It employs a hash function to map terms to a frequency vector indices. Same terms appearing in different parts of a given document will map to the same index in the frequency vector since the output of the hash function will be the same for these terms. However, employing a hash function bears the risk of hash collisions therefore risking representation of different terms in the document as if they were the same.

The HashingTF method can be used as the "TF"(term frequency) part of the TF-IDF method explained in chapter 2. This method does not count words but the fact that same words map to the same hash output forms the basis for the term frequency calculation. The motivation for using of the hash function is based on potential performance benefits. Generally, in a TF-IDF based featurization one would have to construct a vocabulary

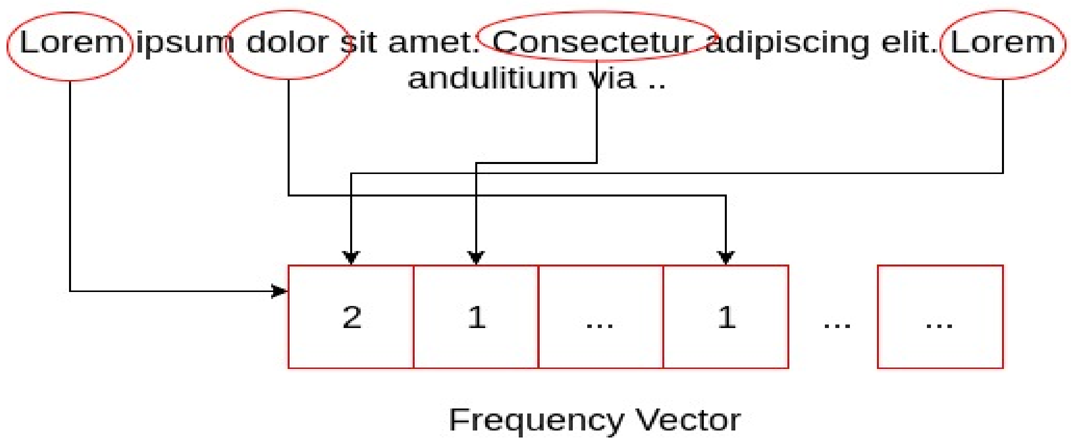

Lorem ipsum dolor sit amet. Consectetur adipiscing elit. Lorem
andulitium via ..
2
1
...
1
...
...
Frequency Vector

FIGURE 3.8: HashingTF - Mapping terms to indices

of words from the whole corpus. This would mean traversing all of the documents to collect unique words which could be expensive in terms of running time and space.

In the figure 3.8, the logic for mapping terms to indices is presented. Consider the lorem ipsum text as the document and each word as a term. The arrows represent the application of the hash algorithm whose output is an index in the frequency vector. Same terms map to the same index however this is not always true because of hash collisions. One way to avoid getting hash collisions is to increase the size of the frequency vector. Spark recommends using a power of 2 for the size of the feature vector for even mapping of the terms.

The HashingTF method in Spark makes use of MurmurHash3 algorithm[16]. This algorithm is used for hash-based look-ups. It is not used for cryptographic purposes therefore it is easy to generate hash collisions but for our purposes it will be enough.

3.2.2 Countvectorizer

Countvectorizer is another featurization method in Spark. Like HashingTF, it can also be used for calculating the "TF"(term frequency) part of the TF-IDF method explained in chapter 2. As opposed to using a hash function for mapping terms to vector indices, this method is based on counting terms and incrementing their respective index in the vocabulary vector. In figure 2.1, this process is explained. A vocabulary of words is extracted by traversing the corpus. Every document receives this vocabulary vector and for each document, the term weights for the terms in the vocabulary vector are calculated by counting. This process is the same process as the TF (Term Frequency) part of the TF-IDF featurization method which was explained in chapter 2.

In addition, Spark provides the parameters *vocabSize* and *minDF* in the implementation of Countvectorizer. *vocabSize* parameter signifies the number of most frequent words to be considered. *minDF* on the other hand is the cutoff minimum document frequency. A given term has to appear *minDF* many times across the corpus to be able to be considered in the term weighting process.

3.2.3 Word2Vec

In Spark's implementation of Word2Vec, the skip-gram model is used[8]. Skip-gram model implies predicting the context of a given word. Also, words closer to a given input word will be considered as the input word's context with higher probability. This is achieved through random sampling of the words[17] based on a maximum distance parameter which is the k training window in our case.

Given input words $w_1, w_2, \ldots, w_T$ Spark tries to maximize the average log-likelihood

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j=-k}^{j=k}\log p(w_{t+j}|w_t) \tag{3.1}$$

The formula 3.1 can be broken down as follows: The first summation iterates over the words of a given document. Second summation iterates over the iterated word's immediate surroundings - words that fall within the training window k. The inside of the probability statement can be read as: "What are the probabilities of the words within the training window k, given a specific word?". Every word is featurized and characterized by its surrounding words.

The formula 3.1 can be interpreted as learning the probabilities of a given word's context within a training window k. When probabilities are calculated a given word's immediate surroundings is more likely to be considered than the words that are further away. This is due to random sampling with a bias for picking the closer words more frequently. The maximization of average probabilities from the formula helps the neural network acquire the word vectors as it was explained in chapter 2.1.1.2. The output layer of the said neural network is a softmax classifier which is a multiclass classification method useful in producing outcome probabilities for each possible outcome.

3.3 Implementations

This subsection includes the specific implementations of the Spark's available featurization methods which were explained above. In order to evaluate the results a specific

Table 3.1: Profiles of CVs - Determined by inspection

	Number of CVs	**CV Profile**
	9	Exceptional Data Science CV
	4	Average Data Science CV
	6	MBA, Business, and Finance related CV
	5	Engineer (Not Data Science)
	1	Human Resources
	1	UX Designer
Total	**26**	

job ad and 5 relevant competency texts were chosen. This particular job ad is relatively larger in size than other job ads. The assumption is that a longer text document will result in better results. Also, the job ad seems to contain a lot of keywords related to data science as well as some software development technologies. The applicants for this job ad are asked to be familiar with or be competent in certain areas such as machine learning, big data processing, applying probabilistic algorithms, Spark, python, R, SQL, Hadoop, git version control, Ruby on Rails, ReachtJS, AWS cloud, etc. The job seems to require 5+ years of experience and applicants are promised to be given the potential to influence big decisions within the company as the company is still a start-up and they follow the agile software development methodology.

Upon inspection of the job ad's text, the following competency documents were found to be relevant in evaluating the CVs: "Data Analytics", "Math Statistical Tools", "Statistical Techniques", "Machine Learning", "Data mining". As it was mentioned in chapter 1.1, these competency text documents are written in human language and they specify what a person having these particular competencies should be able to do.

In table 3.1, the profiles of the CVs in the dataset are presented. The profiling of these CVs were based on manually reading through them. Out of the 26 CVs acquired from the web 9 of them are exceptionally well data science CVs. Owners of these CVs have around 5 to 10 years of experience with significant achievements. These CVs should be able to qualify for most of the job ads based on our human judgment. However, the document similarity techniques will care only about whether certain keywords are present within a document. Therefore these CVs might not rank the highest but they are expected to be the top contenders among other CVs. The CV dataset also includes 4 data science CVs of varying degrees of competency, 6 MBA or business related CVs, 5 non-data scientist engineers (mostly electrical engineers), 1 human resources manager, and 1 UX designer CV. The CVs which are different than data science CVs are there to give an idea about how accurate our results are. It is expected that irrelevant CVs will rank lower in document comparisons.

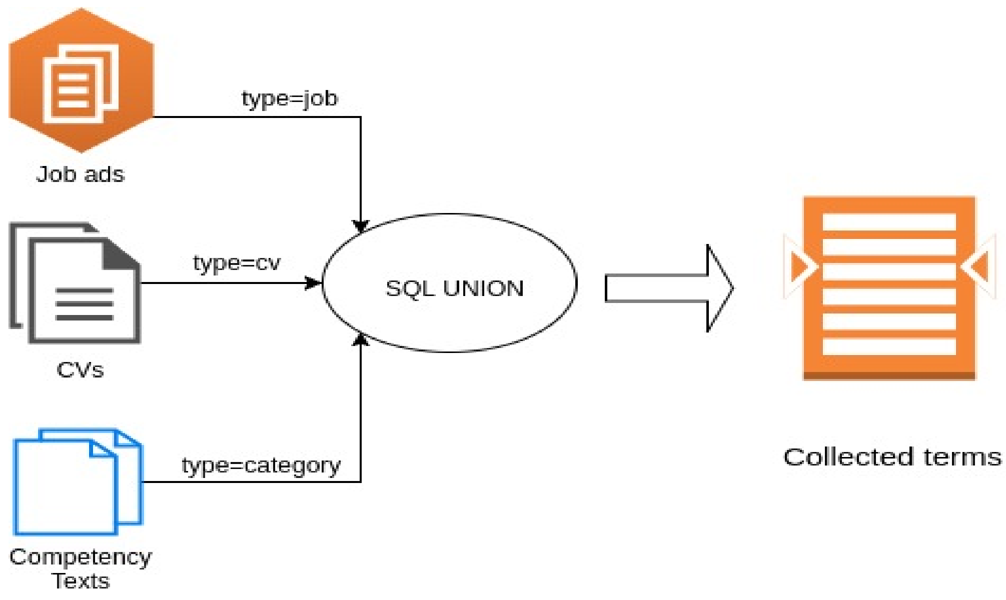


FIGURE 3.9: Collecting raw text of different document types

For each implementation, we select the top 4 ranking CVs based on the cosine distance values results. We display their radar charts to get an idea on how well these CVs score in comparison with the competency documents. We will also manually inspect the contents of these CVs to determine how well a given implementation does in picking the right CVs for the job ad at hand.

3.3.1 HashingTF Implementation

In chapter 3.2.1, Spark's HashingTF featurization method was mentioned. In simple terms HashingTF is a hash function that maps terms in a document to indices in document vectors. This section covers the specific implementation of this featurization method in solving the problem of CV and job ad matching. The data used for this implementation is the 624 job ad data set inherited from the EDISON competencies project and 26 CVs acquired from the web.

Firstly, the raw text of the job ads, CVs and competency texts are collected. This is due to the document comparison logic presented in figure 3.2. The collection of the raw text is illustrated in figure 3.9. In this figure, we see the power of the dataframe API of Spark. In Spark, dataframes - resilient distributed data sets (RDDs) with homogeneous schemas - can be registered as SQL tables within Spark driver program code and after that typical SQL operations can be performed on these dataframes to operate on the data as if they were tables in a SQL based database. In the figure it is shown that a SQL UNION is performed to concatenate the dataframes vertically, if the reader may visualize the SQL tables as tables of rows extending downward. Three different document types

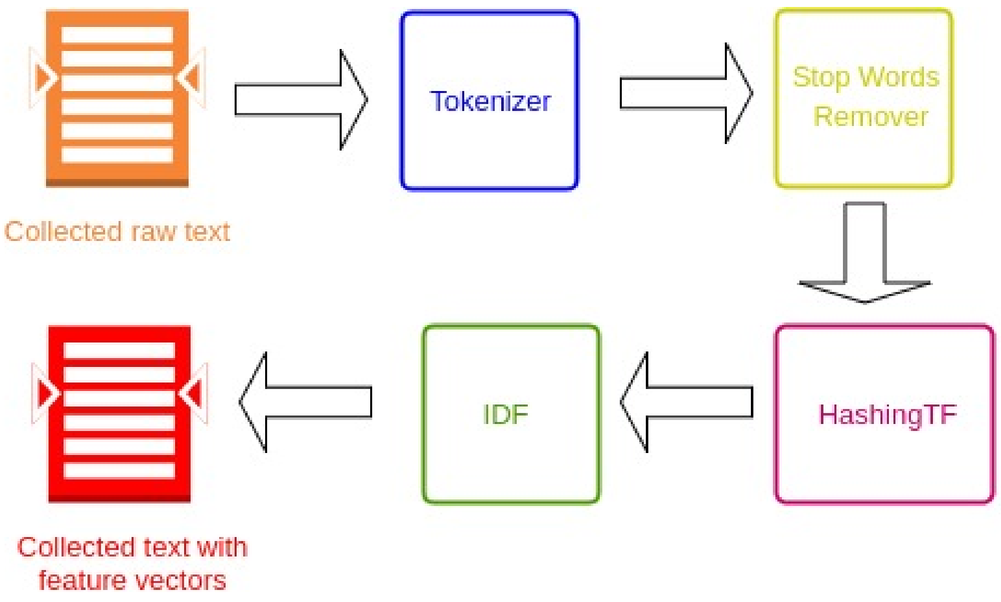


FIGURE 3.10: HashingTF method - Data pipeline

are concatenated with an extra column "type" for differentiating different tables later on. For example, every row in the job table is labeled with "type=job" column so after performing a few operations on the collected data we are able to separate the different document types with a SQL SELECT on the "type" column.

Secondly, the collected raw text is put through a series of operations which are named as data pipeline operations. In figure 3.10, different operations on data are illustrated. The collected data starts being processed with *Tokenizer*, which creates collection of tokens from a raw document. The words in a document are parsed with space character as the delimiter character. Terms that fill document feature vectors are simply individual words in a document. After the tokenization, the tokenized data goes through *Stop Words Removal* process. Stop words are words such as *has, does, is, should, must, ...* - words that do not contribute to what is being conveyed in the document. TF-IDF method especially takes care of the noise generated by these words however removal of these terms completely provides healthier results as we know for sure these words do not matter to our judgment of how well a CV matches with a job ad. After the *Stop Words Removal* process, the data continues to flow to the *HashingTF* process. This is Spark's implementation of the term frequency (TF) implementation of the TF-IDF concept. What we end up with at this point is term weight vectors acquired by mapping each term in the document to an index in the document vector. The size of the document vectors are chosen to be 256. If the reader recalls in 3.2.1, Spark recommends picking a power of two for this value. Next up, the data goes to the process: *IDF*. The terms which were weighted are down-weighted with the IDF formula explained in 2.1. This down-weighting process takes places considering the entirety of the corpus. Finally, the

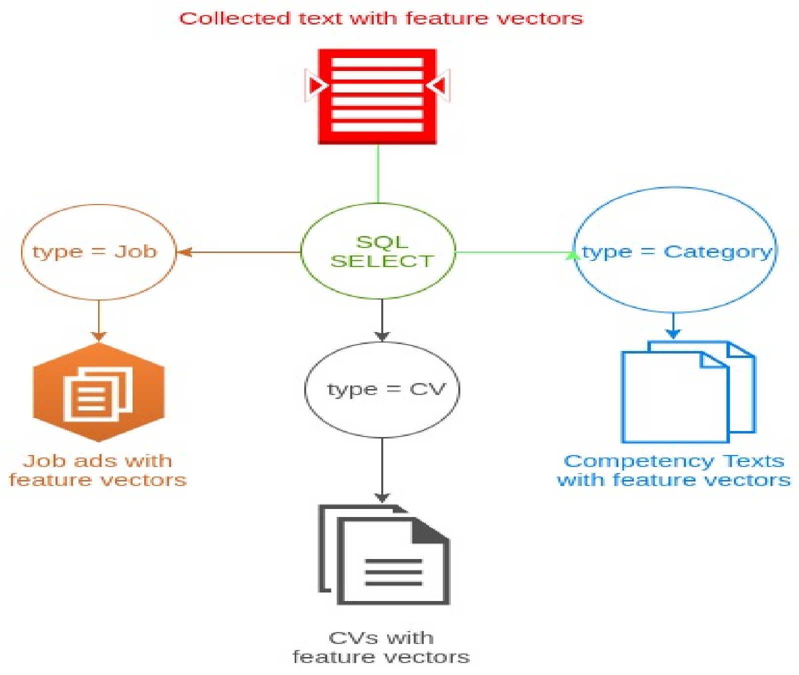

Collected text with feature vectors
SQL SELECT
type = Job
type = CV
type = Category
Job ads with feature vectors
CVs with feature vectors
Competency Texts with feature vectors

FIGURE 3.11: Seperating document types using SQL select

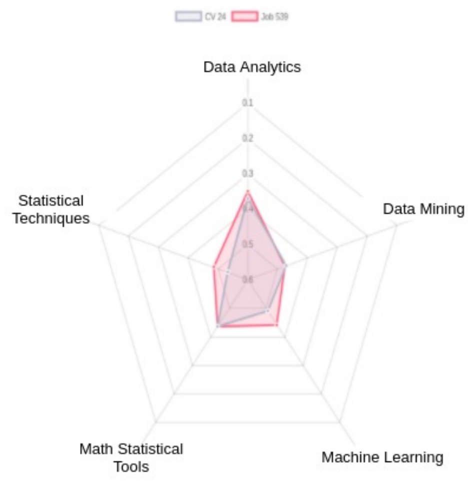
end of the data pipeline operations leaves us with feature vectors per document; ready to be compared with the cosine distance metric.

After the data pipeline operations are finished, we are left with a collected text data of all the job ads, CVs and competency texts. Next step is to separate them and compare their calculated feature vectors with the cosine distance metric. In figure 3.11, we see the separation of document types with the SQL SELECT statement on the collected text dataframe. The separation is done so that the document comparison logic presented in figure 3.2 can be executed. Job ads, CVs and competency texts are compared in combinations of two and the result of this comparison as written as output in CSV format.

3.3.1.1 Results

The top 4 ranking CV from the results of HashingTF implementation is presented in two separate figures in figure 3.12 and 3.13. For each CV, firstly the reasons for it ranking high will be discussed and secondly, competency text comparison results will be discussed. The blue area in the charts represent the CVs document comparison area while the pink area represents job ad's. A bigger area implies higher competency.

The top ranking CV, CV with id 24, is one of the engineer CVs in our dataset. The owner of the CV claims that he/she has strong experience in integration of electrical and mechanical systems. Since we are looking for someone competent in data science and



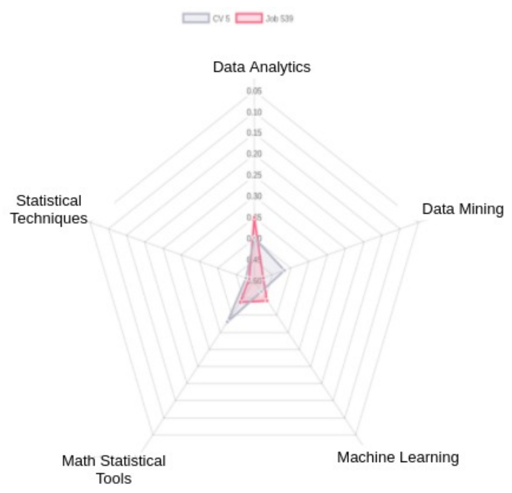
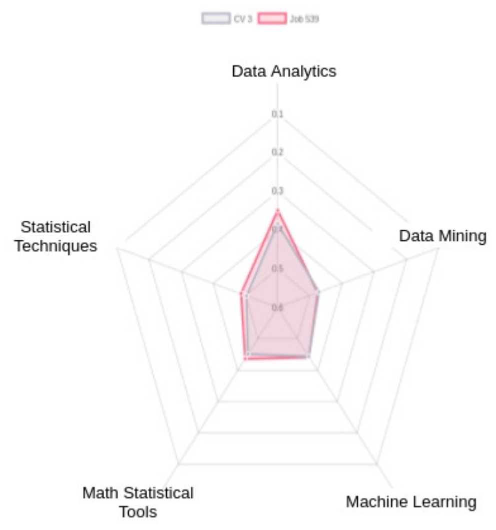


FIGURE 3.12: HashingTF results - Top 2 CVs

machine learning, the selection of this CV as top ranking CV seems to be a bad result. The CV text seems to hit some keywords mentioned in the job ad text such as Excel and Matlab however one would think these would not be enough for this CV to rank so high. Upon further inspection it was deduced that this CV seems to coincidentally mention some keywords that are common with the job ad text; however these words do not truly convey the competency of the CV owner. Some of these words are: project, work experience, senior, systems etc. As for the competency text comparisons, this CV seems to convey some meaningful results. The CV mentions Matlab and Excel and the result tells us that the CV is as good as the job when both the job and the CV are compared to the "Math Statistical Tools" competency document. The CV is an electrical engineer CV therefore there is no mention of machine learning. The "Machine Learning" competency text comparison result seems to be accurate in that regard. The comparison result with the "Data Mining" competency seems to be as good as the job ad's requirement which seems to be a bad result. "Data Analytics" comparison also looks high whereas it should have been lower than the job's requirement. Lastly the "Statistical Techniques" comparison seems to be accurate as the pink area covered by the job ad is higher than the CV's.

The second ranking CV, CV with id 5, is one of the exceptional data science CVs in the dataset. There could be a couple of explanations for why this CV ranks so high. Firstly, this CV is rather long. Longer the text is, higher the chances of terms mentioned in the job ad to be mentioned in the CV and this CV seems to touch upon some of these terms. Candidate seems to have experience in big data technologies, machine learning, and statistical tools. The CV's high ranking seems to be an accurate result. The candidate appears as though their skills in machine learning are not adequate however this is definitely not the case. The document comparison regarding "Math Statistical Tools", "Data Mining" and "Statistical Techniques" seem to suggest that



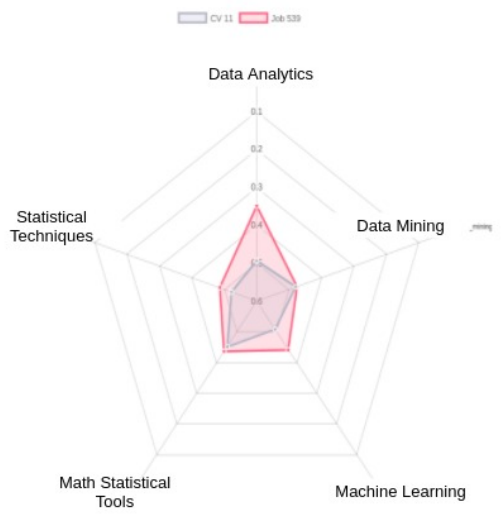


FIGURE 3.13: HashingTF results - Top 3rd and 4th CVs

this CV is overqualified. The "Machine Learning" competency text comparison seems to be inaccurate as the job ad seems to score higher than the CV.

The third ranking CV with id 3 is one of the exceptional data science CVs in our dataset. The competency document comparisons suggest that this CV fits the job ad the most as there is a clear overlap between blue(CV) and pink(job) areas. The owner of the CV is a PhD data scientist who mentions many data science keyword terms in the CV such as R, SQL, Python and some machine learning terms such as supervised/unsupervised learning. The high ranking of this CV seems to be reasonable. The competency text comparisons also confirm this good result. The comparison between the job ad and the competency texts overlap almost perfectly with the comparison between CV and the competency texts.

Lastly, the fourth ranking CV with id 11 is one of the average data science CVs in our dataset. Upon inspection this CV is determined to be the most suitable CV among the average data science CVs. The CV text contains many data science keywords such as Python, SQL, R, Matlab, Hadoop, MapReduce, HBase, Splunk and many more. One would expect this CV to rank high and this result seem to confirm this. As per the competency text comparisons, "Data Mining" and "Math Statistical Tools" rank similar to the job ad text. This seems to be accurate as the CV mentions data mining and math tools such as Matlab. "Data Analysis" comparison result seems to be accurate as the CV does not contain many keywords in this particular competency document. Lastly, "Machine Learning" competency text comparison seems to be wrongfully evaluated since the CV mentions many machine learning concepts such as classification, statistical modeling and such.

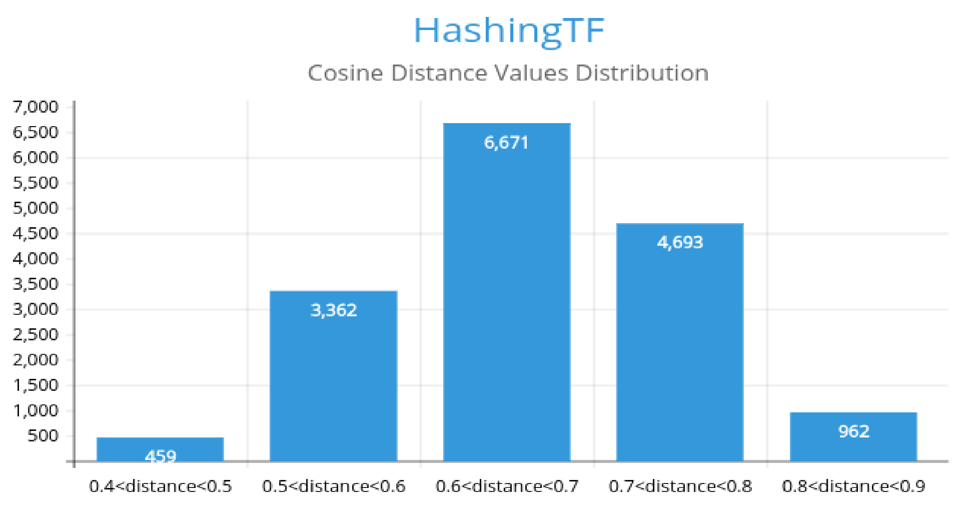


Figure 3.14: HashingTF results - Cosine distance values distribution

Overall, the HashingTF featurization method seems to produce effective results. Among the top 10 CVs, 5 of them were the exceptionally well data science CVs in our dataset. 2 CVs were average data science CVs with entry level or a few years of experience. 2 CVs were MBA or manager CVs that are irrelevant to the job ad. Lastly, the top scoring CV was an engineering CV that had many common words with the job ad text.

The 624 job ads were compared with 26 CVs which amounts to 16224 comparisons. In figure 3.14, the distribution of the cosine distance values are presented. The cosine distance metric outputs values between 0 and 1. The distance values obtained from the HashingTF method are mostly between 0.6 and 0.7. The distance values have a mean of 0.6648 and a variance of 0.00785.

3.3.2 Countvectorizer Implementation

In chapter 2, the Countvectorizer featurization method within Spark framework is presented. Countvectorizer is a method that is based on counting each term in each document and downgrading terms that occur across many documents. This section covers the specific implementation of this concept.

Just like the HashingTF implementation, the raw text from job ads, CVs and competency texts are collected. This collection of data is illustrated in figure 3.9. The raw text of the documents are put together with a SQL UNION on the dataframes which are registered as SQL tables. Each dataframe table is tagged with a type column to be differentiated and separated after feature vectors for each document are calculated.

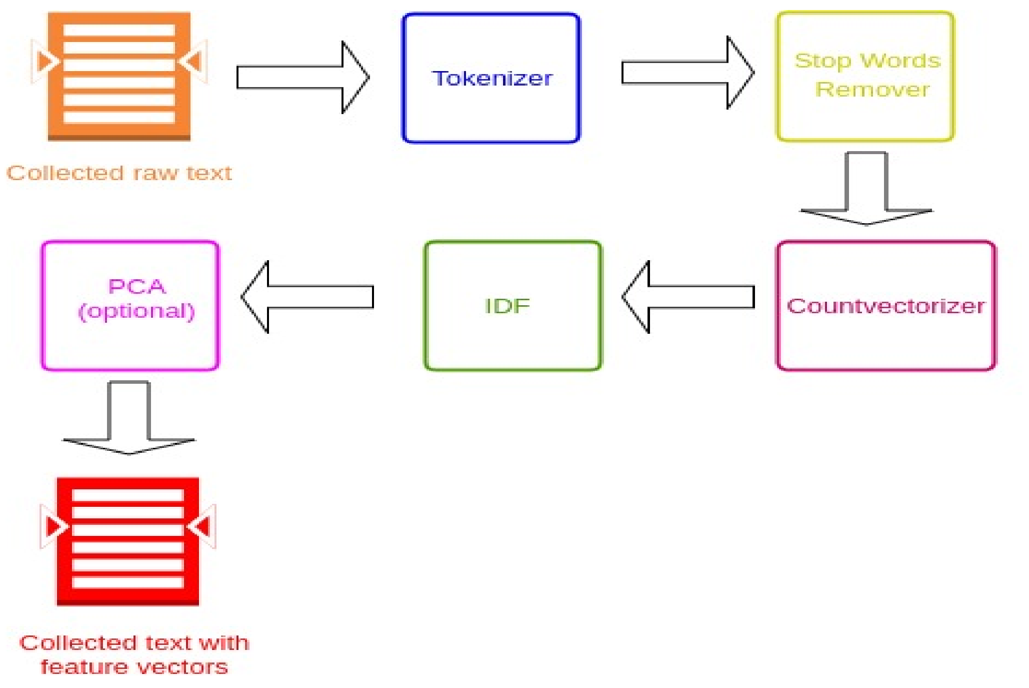


FIGURE 3.15: Countvectorizer - Data pipeline

After each document's raw text is put together, the data goes through the data pipeline operations. In figure 3.15, this process is illustrated. The collected data starts to flow following the arrows. It goes through the *Tokenizer* which creates tokens from a document by parsing words with the space character as delimiter. Then the *Stop Words Remover* process removes words that do not contribute to the quality of the document such as *has, does, is, ...* Next, the data flows to the *Countvectorizer* process which firstly constructs a vocabulary of size 150 (a chosen number taking into consideration the fact that CVs and job ads are short texts) from unique words in the entirety of the collected text and secondly counts each term and increments their respective index in the vocabulary vector. Terms are assigned importance proportional to the number of times they occur within a document. After the *Countvectorizer* process the data flows to *IDF* process which down-weights the terms that occur across many documents. Note that the terms were weighted linearly by counting and with the *IDF* process, they are down-weighted logarithmically. The next process in the data pipeline is *PCA(Principal Component Analysis*, which is an additional operation that will be explained in the next subsection. There are two different implementations of the Countvectorizer method: Countvectorizer without *PCA*, and Countvectorizer with *PCA*. This subsection will cover the former. Finally after all the processes in the data pipeline which were given above, the feature vectors for every document are constructed.

After the data pipeline operations are finished, we are left with a collected text data of all the job ads, CVs and competency texts whose respective feature vectors are calculated. Now it is needed to separate the collected text data with a SQL select statement on the

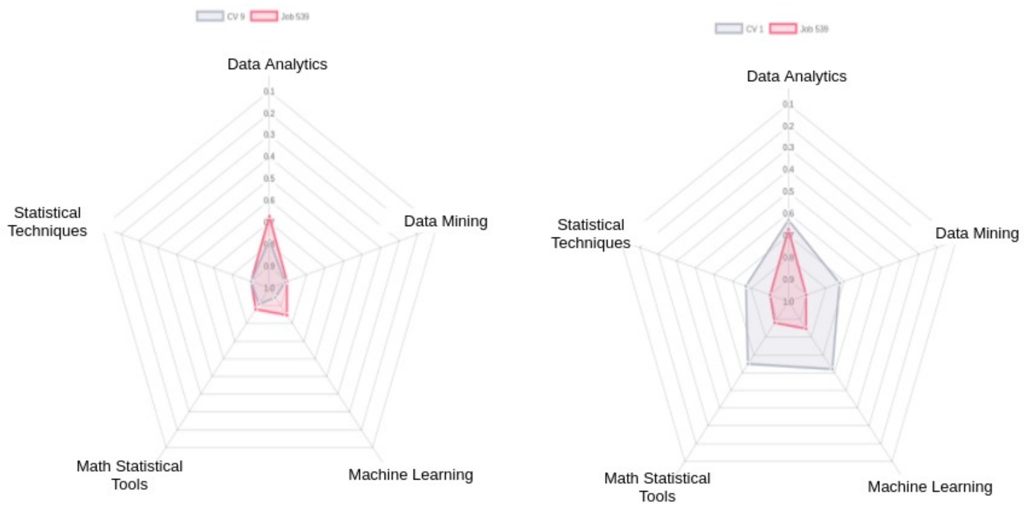


Figure 3.16: Countvectorizer results - Top 2 ranking CVs

"type" column, just like it was done in the HashingTF implementation. This process is illustrated in figure 3.11. After separating CVs, job ads and competency texts their respective feature vectors are compared with the cosine distance metric. This comparison is done based on the document comparison logic which was presented in figure 3.2. The feature vectors of each document type are compared in combinations of two and the result is written as output in CSV format.

3.3.2.1 Results

The top 4 results of the implementation of the Spark's *Countvectorizer* featurization method are presented in two separate figures 3.16 and 3.17. For each CV, firstly the reasons for it ranking high will be discussed and secondly, competency text comparison results will be discussed. The blue area in the charts represent the CVs document comparison area while the pink area represents job ad's. A bigger area implies higher competency. The top ranking CV with id 9, is one of the exceptionally well data science CVs in our dataset. The owner of this CV claims to be a "Mathematics Specialist" and claims to possess skills such as Matlab/Octave and Python. He/she led facebook and linkedin ad campaigns which makes him/her a good candidate for this job since the job ad in question is looking for people who are willing to take ownership of projects, people who are project leaders. This seems to rank high despite having such a short text. The CV seems to be right where job ad's requirements are when it comes to competency texts: "Data Mining", "Statistical Techniques", and "Math Statistical Tools". Judging by the CV owner's skills in Matlab/Octave and his experience in leading marketing projects that require strong statistical methods, this comparison results seem to be accurate. However CV seems to be lacking competency in the other two fields: "Machine

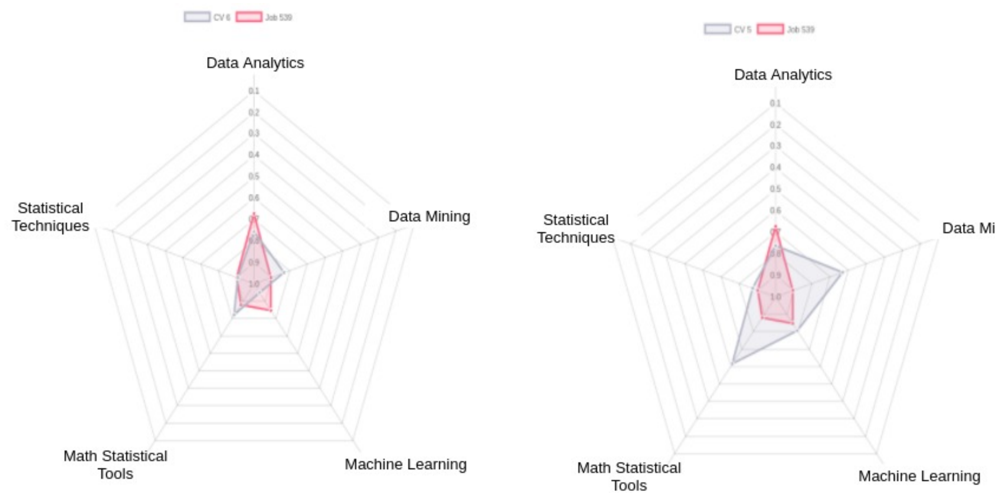


FIGURE 3.17: Countvectorizer results - Top 3rd and 4th ranking CVs

learning" and "Data Mining". Although one can deduce that the CV owner is probably very competent in machine learning, he/she does not mention keywords related to this field for our algorithm to figure out this fact. Likewise, the CV owner does not mention much related to the "Data Mining" competency and hence the pink area being bigger with respect to this competency.

The second ranking CV with id 1, is one of the exceptionally well data science CVs in our dataset. Upon inspection one can confidently say that this CV is indeed a good match against the job ad in question. The CV owner claims to have strong math background and experience in big data and machine learning. He/she lists some related keywords such as Python, R, SQL, Hadoop (Hive, MapReduce), some python machine learning modules like scikit-learn, numpy, scipy, pandas, gensim, and some statistical methods such as time series, regression models, hypothesis testing, etc. As for the competency text comparisons, the blue area that represents the CV owner's overall competency seems to completely dominate the requirements by the job ad. It is no surprise that there is an overwhelming over-qualification with the competency texts: "Data mining", "Math Statistical Tools", "Statistical Techniques", and "Machine learning". The CV seems to be also overqualified when it comes to "Data Analytics", but only by a small margin. Upon manual inspection, the CV seems to agree with these document comparison results.

The third ranking CV with id 6 is also one of the exceptionally well data science CVs in our dataset. The owner of the CV claims to have taken leadership role in analytics and analytical technologies. He/she has 15 years of experience in predictive analytics, informatics, and visualization. The CV seems to fit the job ad's requirement of people who are willing to take ownership of projects. In this rather long CV many related key words are mentioned such as Python, SQL, Hadoop, Big Data, R, Octave, Matlab and many more. As for the competency document comparisons, one surprisingly sees that CV's

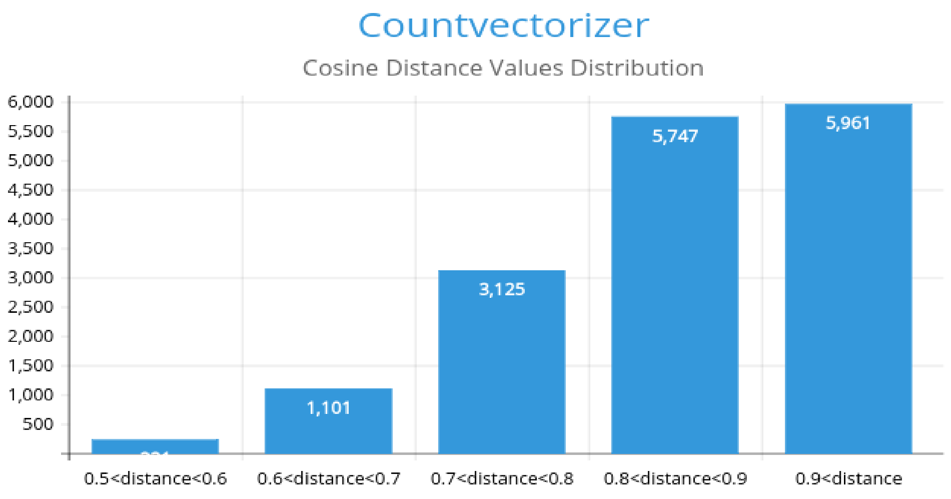


FIGURE 3.18: Countvectorizer results - Cosine distance values distribution

competency area is smaller than job ad's completely. Job ad's requirements seem to be higher than the CV when it comes to competency texts "Data Analytics" and "Machine Learning". This seems to be an inaccurate result since the CV mentions competency in these fields with keywords such as "Linear/Logistic Regression", "Neural Network", "Support Vector Machine", "Decision Trees", "K-means clustering" and many more. Someone who has taken leadership role in analytics and analytical technologies ranking lower than the job ad in question is unmistakably a faulty result.

Lastly, the fourth ranking CV with id 5 is also one of the exceptionally well data science CVs in our dataset. This CV also appeared on HashingTF method's results in chapter 3.3.1.1. In there, the CV's high ranking was attributed to his/her experience in big data technologies, machine learning and statistical tools. Similar conclusions can be drawn for the *Countvectorizer* method as well. As for the competency documents comparisons, the CV seems to be overqualified with respect to "Machine learning" and "Math Statistical Tools" comparisons. Manual inspection seems to confirm this result's accuracy as there are many related keywords. The competency text comparison results seem to be similar with the *HashingTF* method. Both methods seem to find job ad's requirement in "Data Analytics" higher. The *Countvectorizer* method seem to highlight the CV's competency in "Data Mining" and "Math Statistics Tools" more than *HashingTF* does. One big difference between these two methods seem to emerge in the "Machine learning" as *Countvectorizer* seem to find the CV competent whereas the *HashingTF* method concludes otherwise.

Overall the *Countvectorizer* method seem to do quite well with the accuracy of the results. Out of the top 10 results, 6 of them were the exceptionally well data science CVs in the dataset. 3 of the results were the average data science CVs with entry level

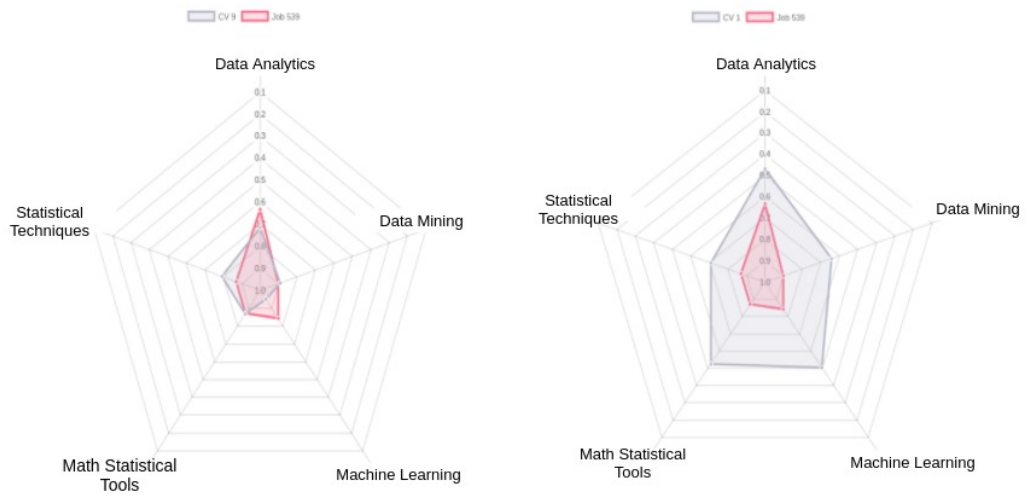


FIGURE 3.19: Countvectorizer with PCA results - Top 2 ranking CVs

or a few years of experience. 1 of the top CVs belongs to one of the management related CVs in the dataset.

The 624 job ads were compared with 26 CVs which amounts to 16224 comparisons. In figure 3.18 the cosine distance values distribution is presented. One thing that is noticeable is that the cosine distance values mostly take the value greater than 0.9. The cosine distance outputs values between 0 and 1. The *Countvectorizer* method produced cosine distance values with a mean of 0.84910 and a variance of 0.00972. The distance values seem to be much higher than the *HashingTF* method hence the higher mean however the variance is also higher which means the cosine distance values are more spread out from their mean than they are when *HashingTF* method is used.

3.3.3 Countvectorizer with PCA algorithm

Upon inspection of the cosine distance values of the *Countvectorizer* method, it was seen that the distance values mostly between 0.9 and 1. One interpretation of this result is that the counting of terms creates high dimensional vectors and this results in low document similarity scores(high cosine distance). Therefore *PCA(Principal Component Analysis)* algorithm was added to the *Countvectorizer* method to reduce the dimensionality of the feature vectors and see how that affects the results. In figure 3.15 this process is illustrated as an optional part of the data pipeline operations. This subsection covers the results that are derived from adding *PCA* algorithm to the *Countvectorizer* method's data pipeline.

PCA, principal component analysis, is the process of extracting linearly uncorrelated variables from a set of variables. When there is a vector with high dimension, PCA helps express the same vector in lower dimensions using orthogonal transformations to

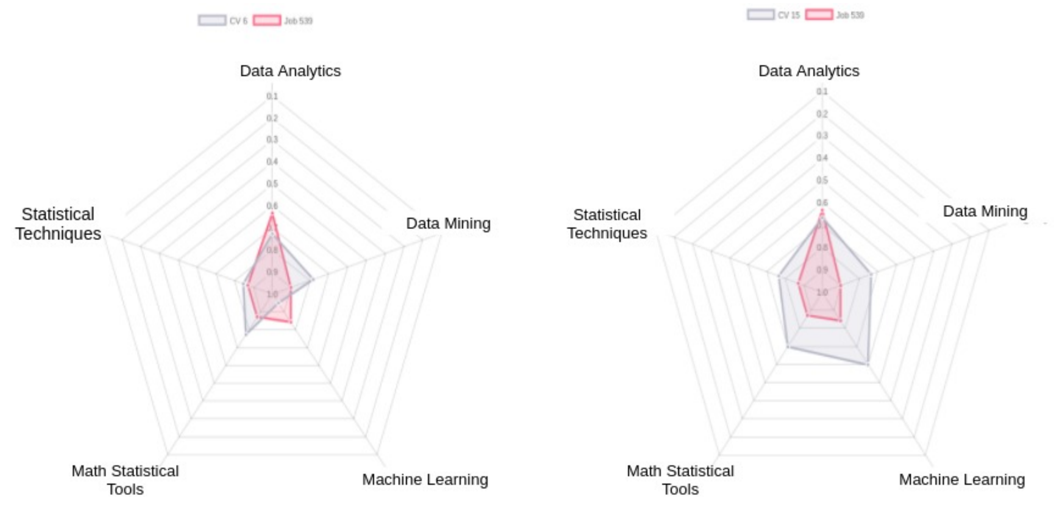


FIGURE 3.20: Countvectorizerwith PCA results - Top 3rd and 4th ranking CVs

eliminate any correlation among variables while trying to maximize variance[18]. In our case, the high dimensionality of the feature vectors obtained from the TF-IDF method with *Countvectorizer* as featurization method could be reduced using PCA.

In the *Countvectorizer* implementation the size of the feature vectors were chosen to be 150. This is the dimension of our feature vectors. PCA can reduce this number further. PCA has the variable k for determining the number of features to be selected. This variable k should be lower than 150 and was chosen to be 50.

3.3.3.1 Results

In figures 3.19 and 3.17, the top 4 ranking CVs are presented in their respective order. Note that the top 3 ranking CVs are the same CVs as the *Countvectorizer* method without PCA. However, their competency document comparison results are slightly different.

The top ranking CV with id 9 is one of the exceptionally well data science CVs in our dataset. The difference between this CV's results and the *Countvectorizer* method without PCA is that the competency documents "Statistical Techniques" and "Math Statistical Tools" score a relatively higher score when *PCA* is applied. This could be interpreted as an accurate statement as the CV's owner claims to be a "mathematics specialist" and mentions keywords such as Matlab and Octave.

The second ranking CV with id 1 is also one of the exceptionally well data science CVs in our dataset. The difference between this CV's results and the *Countvectorizer* method without PCA is that the CV's overall competency seems to be highlighted even more. One could argue this is an accurate change since the CV is exceptionally well.

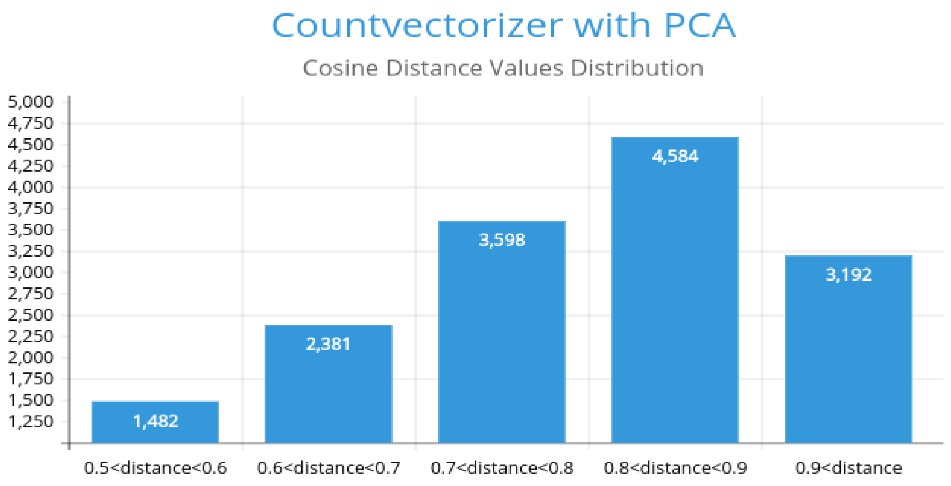


FIGURE 3.21: Countvectorizer with PCA - Cosine distance values distribution

The third ranking CV with id 6 is also one of the exceptionally well data science CVs in our dataset and also the third ranking CV of the results of *Countvectorizer* method without PCA. Difference between the two methods seem to be very little.

Lastly, the fourth ranking CV with id is 15, is one of the average data science CVs in our dataset. When the overall graph of this CV's result is compared to the *Countvectorizer method without PCA*, one can see that differences are minimal. PCA seems to highlight the differences between each of the competency text document comparison results but overall the pink and blue areas look similar. However, the competency text comparison results for this CV seem to mark this CV as overqualified in four areas: "Machine learning", "Statistical Techniques", "Math Statistical Tools", and "Data Mining". These results seem to be accurate since the CV text mostly mentions skills in using data mining and machine learning techniques to produce actionable intelligence from disparate data.

Overall, *Countvectorizer with PCA* seem to produce results which are similar to the *Countvectorizer without PCA*. Out of the top 10 CVs, 6 of them were exceptionally well data science CVs with many years of experience. 3 of them were the average data science CVs. Lastly 1 CV was a human resource manager CV which was completely irrelevant.

The 624 job ads were compared with 26 CVs which amounts to 16224 comparisons. In figure 3.21, the cosine distance values distribution is presented. Cosine distance metric outputs values between 0 and 1. The *Countvectorizer method with PCA* method produced values with a mean of 0.765 and a variance of 0.0219. In comparison with the *Countvectorizer without PCA* method the mean is lower and the variance is much higher. This makes sense since the PCA algorithm tries to maximize the variance in order to

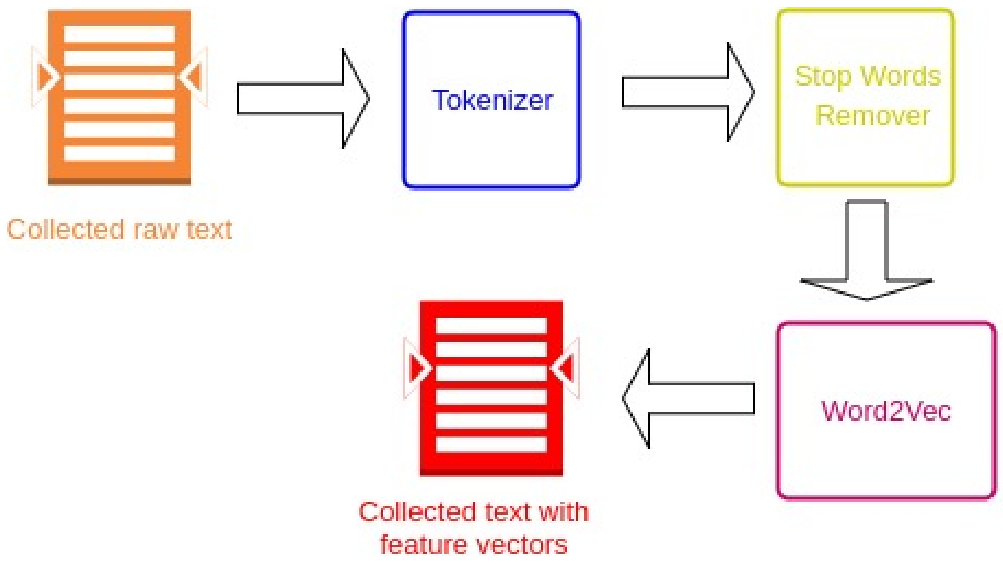


FIGURE 3.22: Word2Vec - Data pipeline

preserve the variability of the values in the feature vectors. This fact also helps with the visualization in the graphs. Since the distance values are more varied, the graphs emphasize the differences between the distance values more. This can result in easier interpretation of the graphs.

3.3.4 Word2Vec Implementation

As described in 2.1.1.2, word2vec method aims to create a vector representation of words in a lower dimension vector space. In this space, vectors that are closer to one another implies that the words that correspond to these vectors are similar in meaning. This section includes the specific implementation of this technique within Spark framework.

The raw text of the job ads, CVs and competency texts are collected, just like they were in the previously explained featurization methods. This is illustrated in figure 3.9. The raw text of these documents are put together using SQL UNION statement after dataframes for each document type are registered as temporary SQL tables in the Spark's execution. The document types are labeled with an extra "type" column to be differentiated after their individual feature vectors are calculated.

After the collection of the raw text of different document types, the data is processed in the data pipeline. First, data flows to the *Tokenizer* so that individual terms can be extracted from the raw text using the space character as delimiter. Second process in the pipeline is the *Stop Words Remover* where words that do not convey significant meaning within the text are removed. These words are *has, does, is, the, ...* etc. Next up, the remaining raw text is processed with the word2vec model which takes into

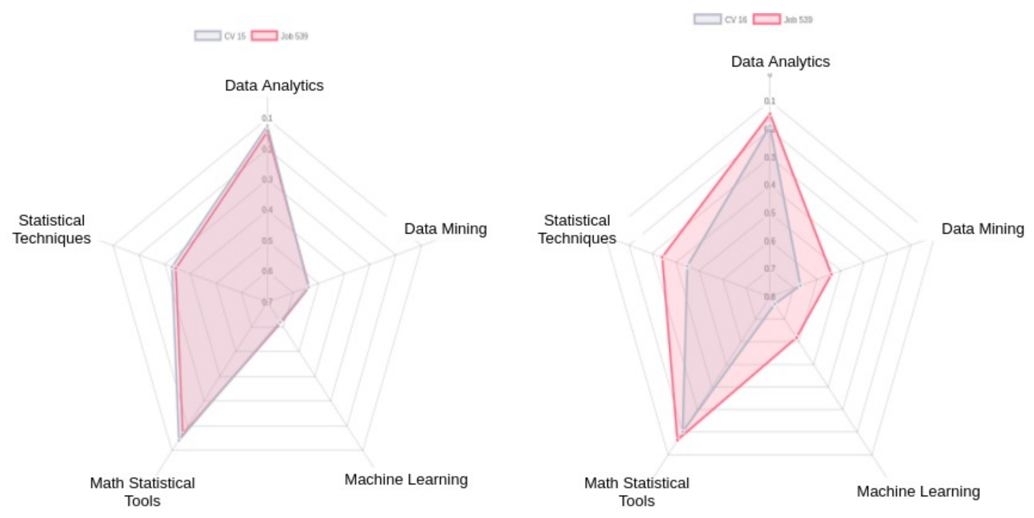


FIGURE 3.23: Word2Vec results - Top 2 ranking CVs

consideration the context of the words to create a vector representation of words. From these word vectors document feature vectors are calculated by averaging all the word vectors within a document. These feature vectors are the input to the cosine distance metric.

3.3.4.1 Results

In figures 3.23 and 3.24, top 4 highest ranking CVs are given respectively. The graphs drawn with this method as well as the order of the CVs look completely different from the results of the previously mentioned featurization methods. This is possibly related to the fact that Word2Vec takes into consideration the context of a given word.

The top ranking CV with id 15 is one of the average data science CVs in our dataset. The competency area covered by this CV seems to almost completely match the job ad's area. This result looks suspiciously too well. Upon inspection the job ad and the CV do not match as much as the graph would have us believe. This seems to be a poor result.

The second ranking CV with id 16 is one of the management CVs in our dataset. The competency area covered looks similar to the top ranking CV which raises more suspicion to the accuracy of these results. The only difference between them seems to be that this CV lacks competency in ”Machine learning” and ”Data Mining”. These results are hardly accurate.

The third ranking CV with id 13, is one of the data science CVs in our dataset. This seems to be an accurate result since the owner of the CV has 6+ years of experience and mentions some keywords such as Python, SQL, and R and some business analytics

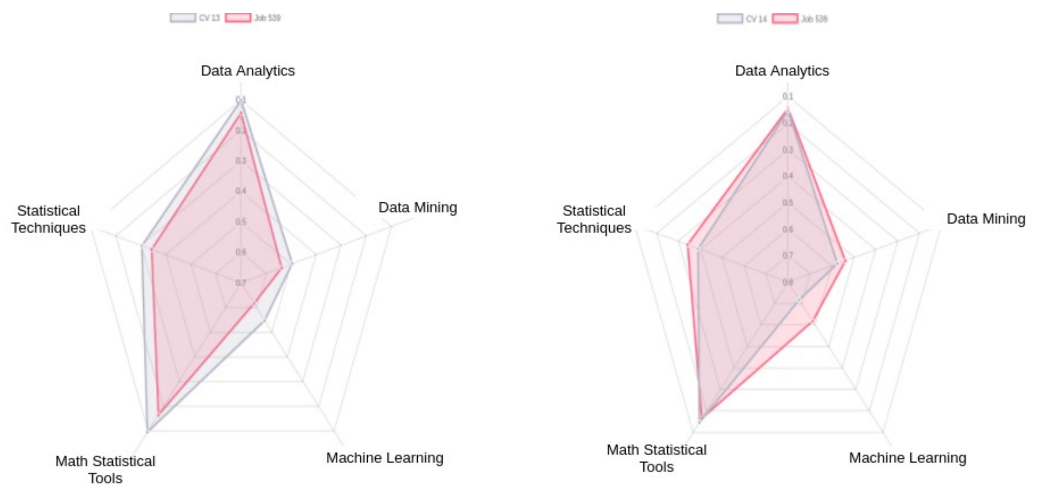


FIGURE 3.24: Word2Vec results - Top 3rd and 4th ranking CVs

terms. The CV's competency area (blue) seems to dominate the pink area covered by the job ad's comparisons.

The fourth ranking CV with id 14, is one of the engineering CVs in our dataset. When the competency text comparisons are considered the CV seems to be only competent enough in areas: "Math Statistic Tools" and "Data Analysis". This seems to be completely inaccurate as the CV does not talk about these areas very often. The CV text is rather long, which may have helped the CV rank higher.

Overall, *word2vec* featurization method seems to produce poor results. Out of the top 10 ranking CVs, none of the exceptionally well data science CVs were selected. The top 10 consists of 2 average data science CVs, 1 human resource manager CV, 4 engineering CVs (not data science), 3 business or finance related CVs. The competency document comparison results seem to produce similar areas among job and CVs which makes it difficult to interpret the charts.

The 624 job ads were compared with 26 CVs which amounts to 16224 comparisons. In figure 3.25, the cosine distance values distribution is presented. The values seem to take values between 0.8 and 0.9 however the distribution is quite spread out. The cosine distance outputs values from 0 to 1. The *Word2vec* method produced cosine distance values with a mean of 0.4143 and a variance of 0.08631. The mean is a much more average value when compared to other techniques. The variance is also high and the wide distribution of the values confirm this. The mean and variance can be explained by the fact that *Word2vec* model uses a softmax model that rounds out the probabilities at the last layer of its neural network.

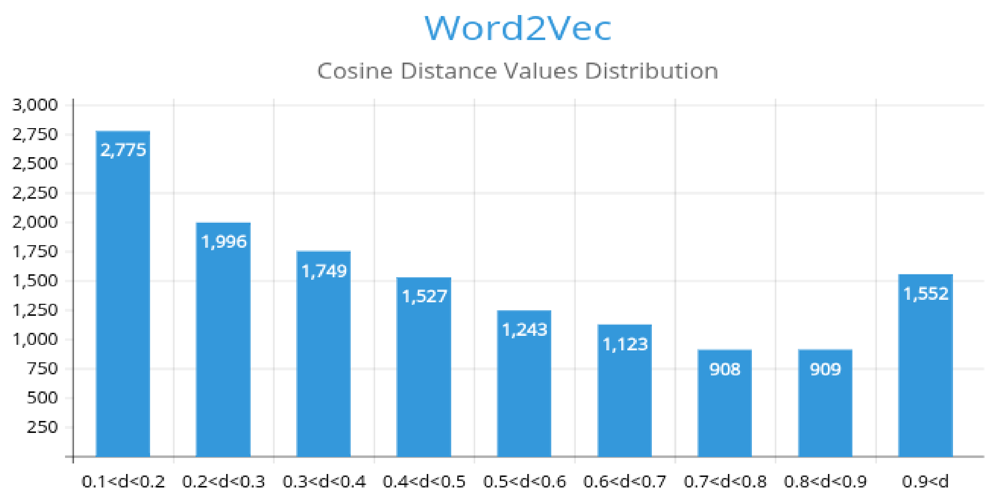


FIGURE 3.25: Word2Vec results - Cosine distance values distribution

3.4 Multilayer Perceptron Classifier Experiment

Apache Spark's machine learning library contains many machine learning algorithms which can be connected to the data pipeline operations. Examples of the said data pipeline operations were given in figures 3.10 or 3.15. One of such machine learning algorithms is provided in the *Multilayer Perceptron Classifier (MLPC)* within *MLlib* (machine learning library) module. This classifier is a multi-class classifier which means it can perform classification when there are more than $k > 2$ classes(where k denotes the number of classes). It is also based on artificial neural networks[19] which gains popularity within the machine learning world. This subsection includes an experiment with *MLPC* and how it can be used on the problem of matching CVs with job ads.

Spark's *MLPC* allows users to specify the number of nodes in every layer of its underlying neural network. Last layer of the neural network has to be equal to the number of classes in our classification problem (denoted by k). The number of layers chosen for each method is 3 and number of nodes per layer has the following general rule:

$$[FeatureVectorSize * 2]- > [FeatureVectorSize]- > [k] \tag{3.2}$$

There is no golden rule for picking the number of layers and the nodes in every layer. However, picking a small network size has some benefits. Smaller networks require less computation power. Moreover, bigger networks generally need larger number of training examples for achieving good generalization performance since it has been shown that the required number of training examples grow almost linearly with the number of hidden units[20]. Therefore we chose to have double the size of the feature vector in the first

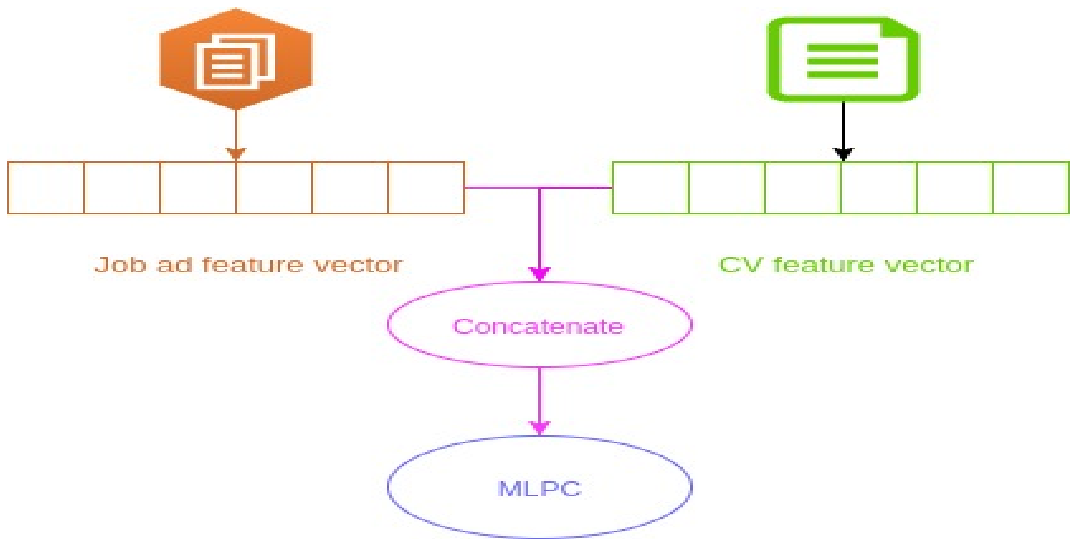


FIGURE 3.26: Multilayer Perceptron Classifier - Preparing to feed data to the model

layer, size of the feature vector in the second layer, and the number of classification classes we have in the last layer (k)

3.4.1 Labeling the Training Data

The *MLPC* is a multi-class classifier that is used to solve supervised learning algorithms. Supervised learning algorithms require training data sets to train a model, and then the trained model is used to perform predictions on a different data set. In our problem of matching CVs with job ads, the problem at hand is not a supervised learning algorithm. We do not have a set of CV and job ad pairs which are marked as "good match" or "bad match" and so on. Therefore, in order to be able to use *MLPC*, we need to label the CV and job ad pairs. One way of labeling the CVs and job ads is to go through all of them manually, then use our human judgment to label them as "good match" or "bad match". However, this would be too time-consuming considering the fact that we have about 16000 CV-job ad pairs. Another way of labeling the data would be to use the results of the different featurization methods such as *HashingTF, Countvectorizer,* and *Countvectorizer with PCA*; which were explained in the previous chapters.

For the labeling of the CV and job ad pairs, for each featurization method, we split the cosine distance values obtained by the said featurization methods by 2 and by 4. In the case of splitting by 2 (k=2), we can see this as labeling the CV and job ad pairs as "bad match" or "good match". In the case of splitting by 4 (k=4), labeling of the CV and job ad pairs can be seen as "very bad match", "bad match", "good match" and "very good match". In both cases the splitting was done so that every piece of the split has equal number of CV-job ad pairs.

Table 3.2: Multilayer Perceptron Classifier - Results

Featurization Method	**Model Accuracy (k=2)**	**Model Accuracy (k=4)**
Countvectorizer	%87	%63
Countvectorizer + PCA	%93	%80
HashingTF	%87	%65

3.4.2 Feeding Data to the Model

Data to be used in the *MLPC* is a pair of documents; namely CV and a job ad. What we initially have is their individual feature vectors. In order to feed a single vector to the *MLPC* model, we simply concatenate the feature vectors of the documents. This process is illustrated in figure 3.26. The individual feature vectors of CVs and job ads are concatenated and fed to the MLPC model. Note that this concatenated vector bears a label such as "good match" or "bad match" so the MLPC model could be trained.

3.4.3 Training and Testing of the Model

The data for the *MLPC* was labeled using the featurization methods *HashingTF, Countvectorizer,* and *Countvectorizer with PCA* and the main method of feeding feature vectors to the *MLPC* model is to concatenate them. Now we can train and test our model. After labeling all of the CV-job ad pairs, these pairs were randomly split for training the model and testing the model. %60 percent of the data was used for training the model and %40 percent of the data was used for testing the model.

In table 3.2, the results are presented. The model is good at seeing the patterns from the concatenated feature vectors. Note that these results are based on the assumption that the said featurization methods are a reliable way of labeling the CV-job ad pairs. As we can see in the figure, the *MLPC* model scores the highest accuracy with %93 when *Countvectorizer with PCA* featurization method is used and there are two labels such as "good" or "bad" match (k=2). The *MLPC* model scores %80 percent with the *Countvectorizer with PCA* featurization method when there are 4 labels (k=4).

Overall, *Multilayer Perceptron Classifier* seems to be a promising classifier judging by the results obtained from various featurization methods which were used to label the training data. The results are good if one assumes that labeling the data based on cosine distance values is a reliable way of generating training data.

Chapter 4

Implementations of Term Extraction

In a world where the needs of research and industry constantly evolve, it is crucial that EDISON's framework for defining the Data Science profession remains up-to-date and relevant. In order to be able to provide the EDISON project[2] advisory information about changing job market requirements, we use term extraction algorithms on the collection of job ads in our dataset. In chapter 2, we explained the *Apriori* and *FP Growth* algorithms. These algorithms are used to mine frequent patterns in a collection of text documents. We also mentioned that *Apriori* algorithm suffers from performance drawbacks and we chose to use the *FP Growth* algorithm for frequent pattern mining. This chapter includes the specific implementations of this algorithm.

Before the *FP Growth* algorithm can be used, we have to prepare the job ad corpus. Data is passed through a series of operations which we call data pipeline operations before it can be processed. In figure 4.1, this process is illustrated. The raw text of the collection of job ads is collected into a Spark dataframe. Then the raw text is *Tokenized*, the words in documents are parsed using the space character as the delimiter. We regard every word as a potentially important term in this case. After *Tokenizer*, the data flows to the *Stop Words Remover*, which eliminates stop words such as *has, does, is, ...* which do not contribute to the quality of the document. Next, the data flows to the *Lemmatizer*. *Lemmatization* is the process of creating dictionary versions of inflected words. For example, the word "went" would become "go" when it is lemmatized. *Lemmatization* is necessary to map the different inflections of the same word together so we can detect frequent patterns more accurately. After the data is processed in the *Lemmatizer*, it optionally goes through the *Blacklist* where certain terms are eliminated. After *Lemmatization* or *Blacklist* (if applied) the data is ready to be processed by the

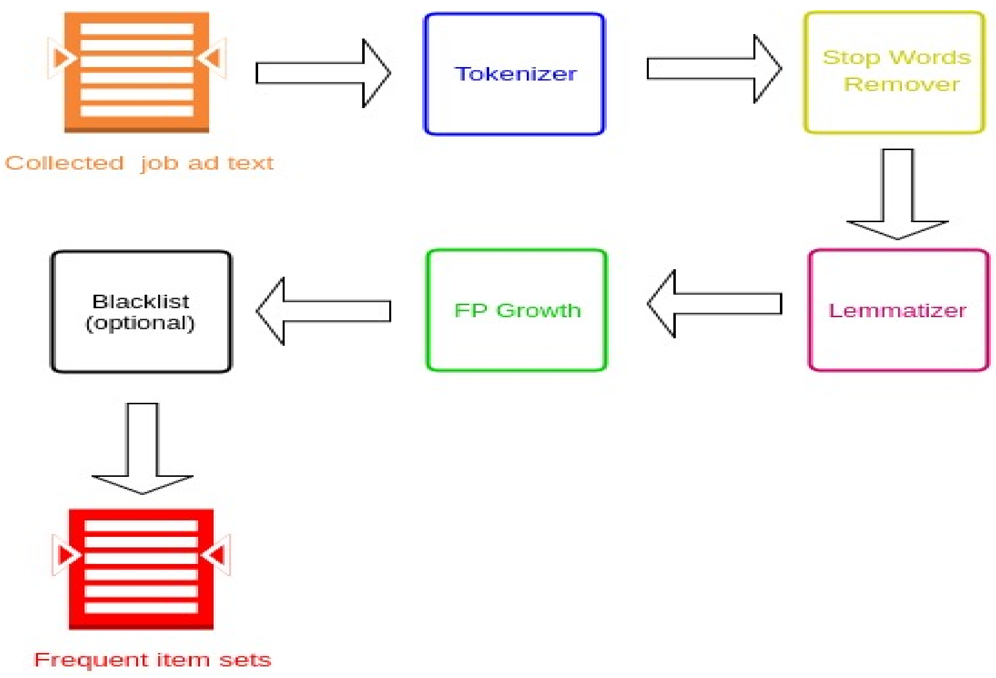


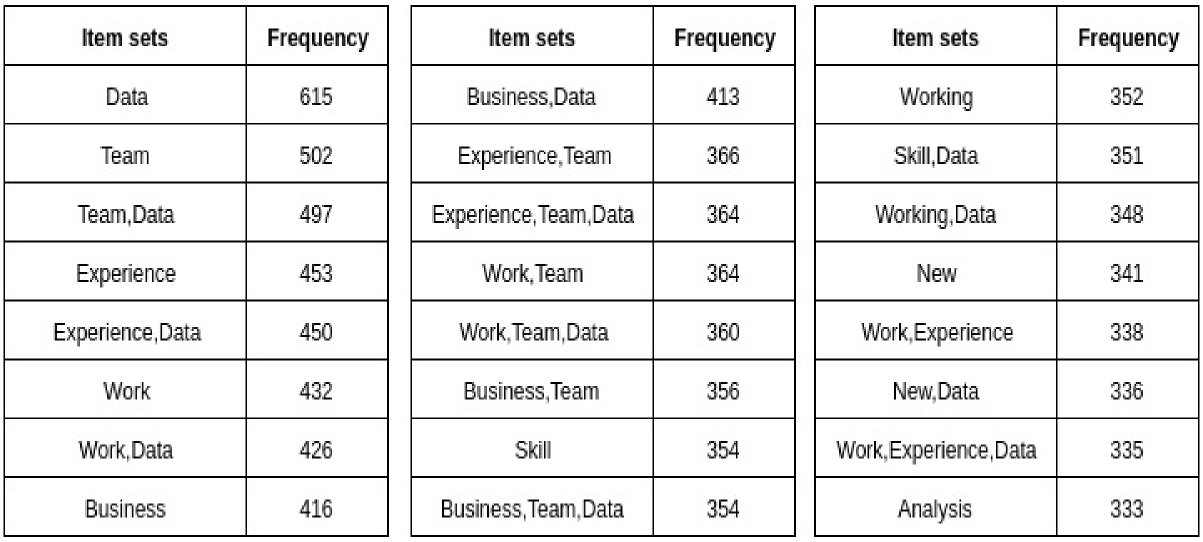
FIGURE 4.1: Frequent pattern growth algorithm - Data pipeline

FP Growth algorithm which produces frequent patterns in job ads. Note that different implementations of *FP Growth* will share similar data pipeline operations (except for the TagMe service implementations in chapter 4.4).

4.1 Applying FP-Growth Directly

Our first approach to finding frequent patterns in job ads is to use *FP Growth* algorithm directly on the data after it has been processed in the data pipeline which is illustrated in figure 4.1. Note that we treat each word as a term within a document by processing with the *Tokenizer*. Our assumption is that the important terms will emerge higher in ranking with respect to frequency when the corpus is processed as a whole, and unimportant terms will not have such high frequencies.

In figure 4.2, the results of applying *FP Growth* algorithm directly after the data pipeline operations are given. First part of each column contains the frequent item sets separated by commas and the second part of each column contains the frequency values. The frequency values indicate how many times that particular item set appears in the job ad corpus. For example, the first item set contains "Data" with frequency 615 which means this word appeared 615 times in the corpus. When we look at the results as a whole, we notice that the top ranking results are too generic to reveal any useful information about what the job market's requirements are. We see that words such as "Data", "Team", "Experience", "Business", etc. appear with high frequencies. It goes without



Item sets	Frequency
Data	615
Team	502
Team,Data	497
Experience	453
Experience,Data	450
Work	432
Work,Data	426
Business	416
Business,Data	413
Experience,Team	366
Experience,Team,Data	364
Work,Team	364
Work,Team,Data	360
Business,Team	356
Skill	354
Business,Team,Data	354
Working	352
Skill,Data	351
Working,Data	348
New	341
Work,Experience	338
New,Data	336
Work,Experience,Data	335
Analysis	333

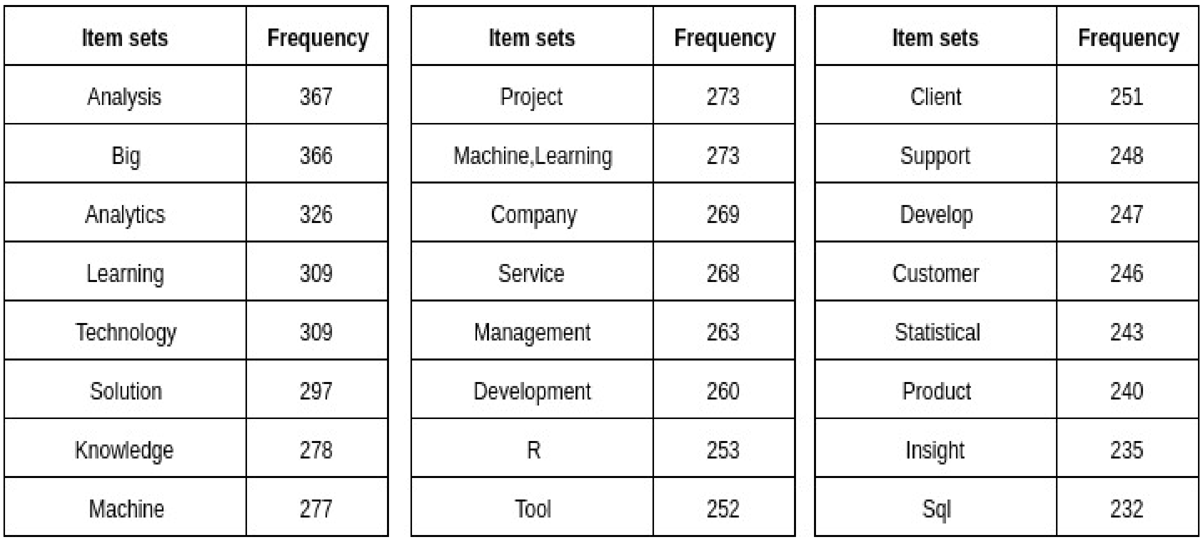
FIGURE 4.2: FP Growth algorithm high frequency item sets

saying that every job in industry nowadays integrate ”data” in their work, looking for ”experienced” members to join their ”teams”, and it is only natural they would talk about their ”business”. It seems that applying *FP Growth* algorithm directly yields results which do not give useful or actionable information.

4.2 FP-Growth with Filtering

In the previous section 4.1, we gave the results of applying the *FP Growth* algorithm directly after the data pipeline operations. The results were not useful for our purpose of providing the EDISON project with the job market trend in industry. After reviewing the said results, we came up with a blacklist of terms which we decided to eliminate from the job ad text before the data is processed with the *FP Growth* algorithm. In figure 4.1, note that the data goes through the *Blacklist* step after *Lemmatization* step. This blacklist is our filter and it consists of the words: ”Data”, ”Experience”, ”Team”, ”Work”, ”Science”, ”Scientist”, ”New”, ”Working”, ”Looking”, ”Role”, ”Strong”, ”Use”, ”Based”, ”Business”, ”Help”, ”Well”, ”Across”, ”Within”, ”Understanding”, ”Ability”, ”Offer”, ”Opportunity”, ”Required”, ”Make”, ”Including”, ”Need”, ”Etc”, ”High”, ”Highly”, ”Understand”, ”Include”, ”Including”, ”Best”, ”Using”, ”Good”, ”Join”, ”Job”, ”Year”, ”Also”, ”Large”, ”Skill”, ”World”, ”Please”, ”Salary”. The blacklist was constructed after consecutive executions of the algorithm and looking at the results obtained. These words were deemed to not convey useful or actionable information about the requirements of the job market.

In figure 4.3, the results of applying the blacklist is presented. The results seem to be better than the ones acquired without applying the blacklist. We see that the terms



Item sets	Frequency	Item sets	Frequency	Item sets	Frequency
Analysis	367	Project	273	Client	251
Big	366	Machine,Learning	273	Support	248
Analytics	326	Company	269	Develop	247
Learning	309	Service	268	Customer	246
Technology	309	Management	263	Statistical	243
Solution	297	Development	260	Product	240
Knowledge	278	R	253	Insight	235
Machine	277	Tool	252	Sql	232

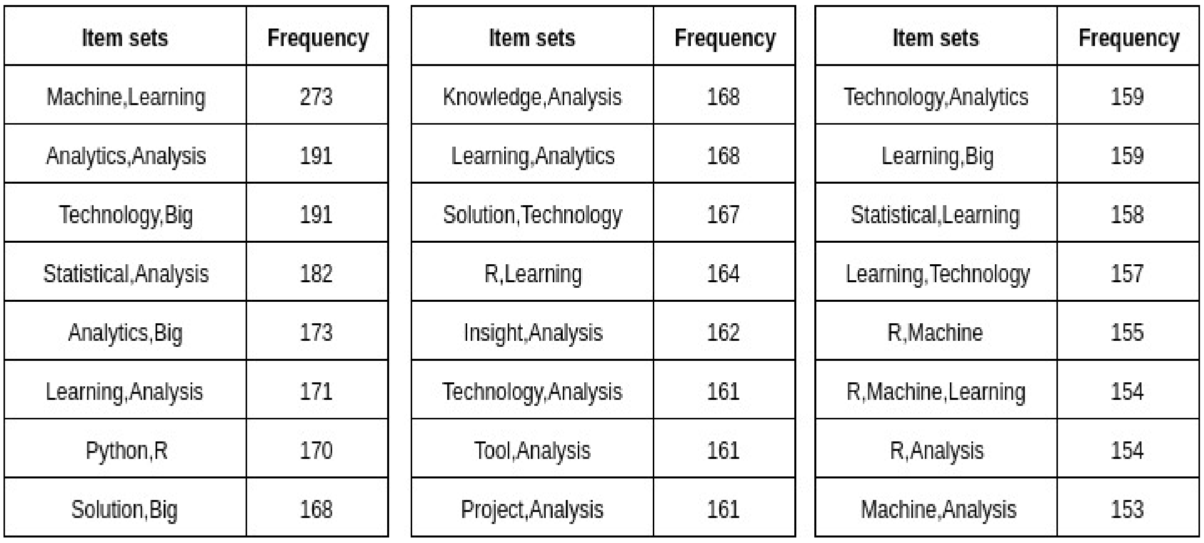
FIGURE 4.3: FP Growth algorithm high Frequency item sets - Filtered with blacklist

"Machine Learning", "Big Data", "R", "SQL", "Analytics", "Analysis", etc. appear which gives us some idea about the job market requirements. We can deduce that applying machine learning algorithms on big data is quite in demand. We also see that tools such as R for statistical computing and SQL for querying of data are also in demand. Using these tools to do data analytics is useful in data science profession.

4.3 FP-Growth with Post-Processing

In the previous section 4.2, we presented the results for applying the *FP Growth* algorithm with applying a filter of blacklist. The obtained results seems to be better when a filter is applied but there is still room for improvement. If we look at the said results in figure 4.3, we notice a certain pattern. Item sets that have more than one word have their individual words appear higher in the list since statistically individual words have a higher chance of appearing across many documents. For example, the terms "Machine" and "Learning" occur more frequently than the term "Machine Learning". The word "Machine" appeared 277 times while "Machine Learning" appeared 273 times. We can post process these results to eliminate individual words when these words appear together in an item set lower in the list. In the given example we would remove the individual words "machine" and "learning" after encountering "machine learning". Post-processing the results in this manner allows us to see the results in a more compact way.

We ran the aforementioned post-processed version of the *FP Growth* algorithm on two different job ad dataset this time. On one hand we have 850 job ads we inherited from the E-CO-2 project which are from summer 2016 and on the other hand we have a new



Item sets	Frequency
Machine,Learning	273
Analytics,Analysis	191
Technology,Big	191
Statistical,Analysis	182
Analytics,Big	173
Learning,Analysis	171
Python,R	170
Solution,Big	168

Item sets	Frequency
Knowledge,Analysis	168
Learning,Analytics	168
Solution,Technology	167
R,Learning	164
Insight,Analysis	162
Technology,Analysis	161
Tool,Analysis	161
Project,Analysis	161

Item sets	Frequency
Technology,Analytics	159
Learning,Big	159
Statistical,Learning	158
Learning,Technology	157
R,Machine	155
R,Machine,Learning	154
R,Analysis	154
Machine,Analysis	153

FIGURE 4.4: FP Growth algorithm high frequency item sets (Jobs from summer 2016) - Filtered with Blacklist and post processed

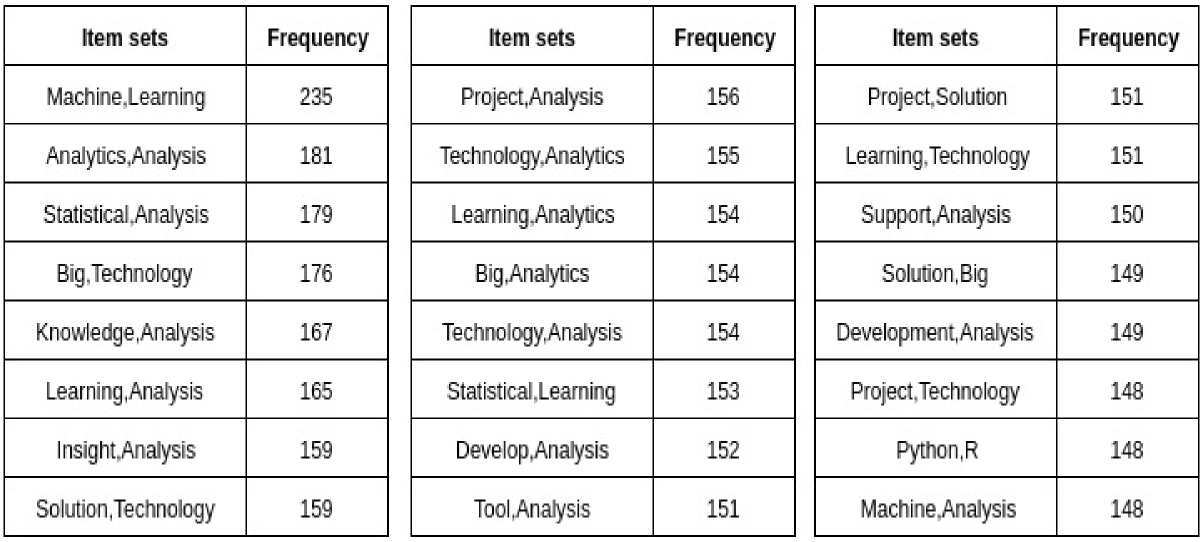
job ad dataset of 2000 jobs which are from December 2016. In figures 4.4 and 4.5 these results are displayed, respectively. Just like in the results without post-processing, we see that terms "Machine Learning", Big Data"(the word "data" was in blacklist, so we know that "big" refers to "big data"), "R" appear again. We see the term "Analysis" appear a lot as well. We can say that these skills are still very much in need for both job ad data sets. When we compare the results of two job ad data sets, we notice the differences to be very small. Both job ad collections seem to value "Machine Learning" a lot since this is the top item set in both. We see that "Python,R" goes down in the list as we move from jobs from summer 2016 to jobs from December 2016. However, this does not necessarily mean that demand for these skills are going down. Only big difference that catches the eye seems to be the fact that "R" appears together with many other terms when we look at the jobs from summer 2016. Perhaps one can attribute to this to the fact that job ads from summer 2016 overall required their applicants to have a wider variety of skills than the job ads from December 2016.

The analysis of job ads from different time periods could give us more information about the job market trend if the analyses continued to be performed in the upcoming years.

4.4 Using TagMe Annotation Service for Term Extraction

4.4.1 TagMe Annotation Service

In the previous sections 4.1, 4.2, 4.3, we use the same data pipeline operations as we prepare the data for the *FP Growth* algorithm. In the data pipeline which is illustrated



Item sets	Frequency	Item sets	Frequency	Item sets	Frequency
Machine,Learning	235	Project,Analysis	156	Project,Solution	151
Analytics,Analysis	181	Technology,Analytics	155	Learning,Technology	151
Statistical,Analysis	179	Learning,Analytics	154	Support,Analysis	150
Big,Technology	176	Big,Analytics	154	Solution,Big	149
Knowledge,Analysis	167	Technology,Analysis	154	Development,Analysis	149
Learning,Analysis	165	Statistical,Learning	153	Project,Technology	148
Insight,Analysis	159	Develop,Analysis	152	Python,R	148
Solution,Technology	159	Tool,Analysis	151	Machine,Analysis	148

FIGURE 4.5: FP Growth algorithm high frequency item sets (Jobs from December 2016) - Filtered with blacklist and post processed

in figure 4.1, we see that *Tokenizer* operation is used to generate a set of items from raw text by parsing the raw text using the space character as delimiter. This means that we treat every word as a term in a document and run the *FP Growth* algorithm so that we may highlight the important (frequent) words. In this section, different from the said approach, we use a third party text annotation service called TagMe[21] to do generating the terms in a document.

TagMe[21] is a free to use annotation service that identifies on-the-fly meaningful short-phrases in text and link them to a Wikipedia page. It claims to work especially well in short text which makes it appealing since we are working with short job ad description texts. For an example of how it works, consider the following sentence: ”On this day 24 years ago Maradona scored his infamous ”Hand of God” goal against England in the quarter-final of the 1986”. TagMe service is able to process this sentence and find out that ”Maradona” refers to the famous football player, ”Hand of God goal” refers to his famous goal with his hand in 1986, and that ”England” refers to the English football team. TagMe figures this out by linking possible entities in the sentence to Wikipedia pages. We aim to leverage this annotation service to find out more about the job market trends in the market.

In figure 4.6, we see the data pipeline operations for the implementations that rely on TagMe annotation service for term extraction from individual documents. Note that *Tokenizer* and *Stop Words Removal* operations are not employed here since TagMe handles the extraction of terms. The output of TagMe is then put through the execution of the *FP Growth* algorithm so we may see the frequent patterns in data science job market.

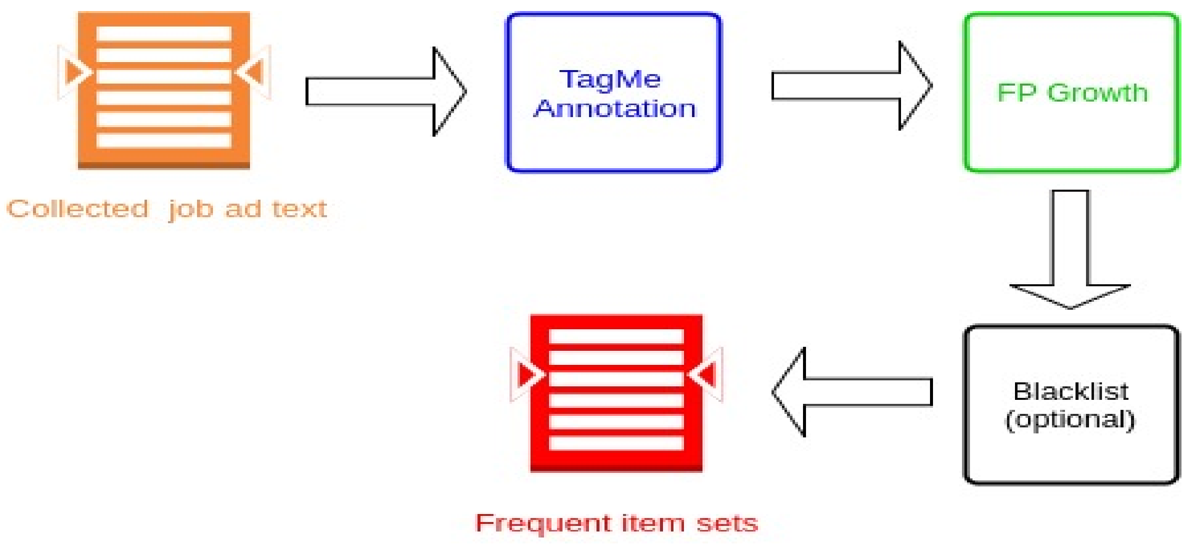


FIGURE 4.6: Using TagMe annotation for term extraction - Data pipeline

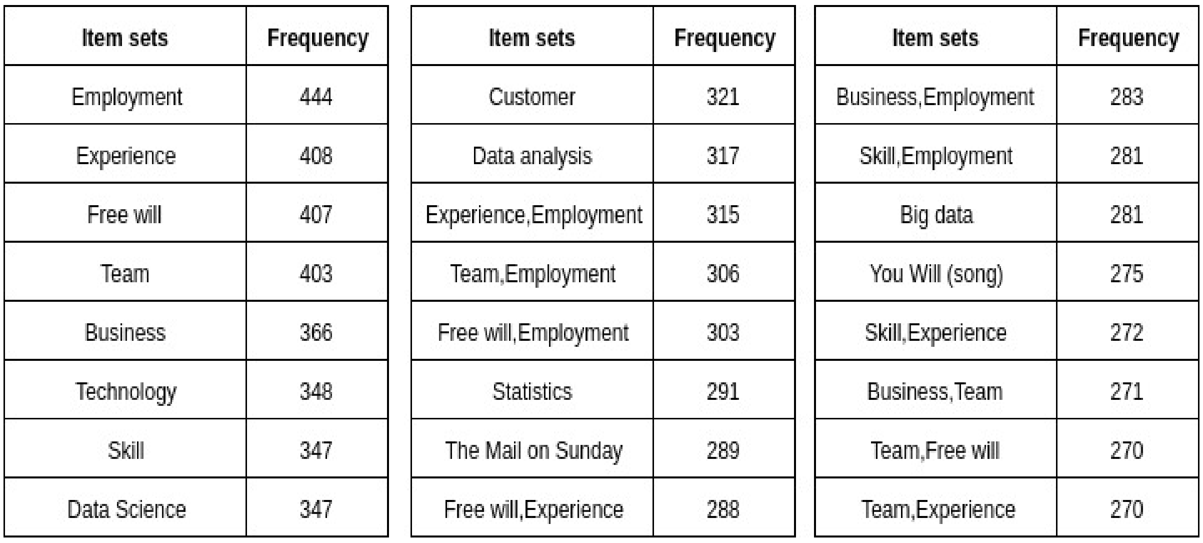
4.4.2 TagMe Validation

TagMe annotation makes it easy to extract entities from short text documents. One thing that sparks interest is the accuracy of the entity extraction of this service. There are some pre-annotated texts online such as biomedical texts and newspapers such as Reuters. We can use these texts to compare TagMe's output with these texts and calculate the accuracy.

Biomedical texts which are used for testing accuracy comes from *Natural Language Processing Laboratory (NLPLab)*[22]. *NLPLab* is a virtual lab that hosts various tools and resources such as biomedical texts for natural language processing tasks. These texts include annotated protein, gene, bacteria, disease, process (ex: infection), chemical, etc. When the biomedical text data from 8 different corpus was collected and put through TagMe service, the output on average matched with the pre-annotated entities with %54.37 accuracy.

Another group of texts used for the validation task is the Reuters Corpora[23]. This is a collection of news stories which were made available by Reuters Ltd. in 2000, for use in research and development of natural language processing, information retrieval, and machine learning systems. The pre-annoated text consists of people, location and abbreviations. When the collection of 13000 news stories was put through TagMe service, the output on average matched with the pre-annotated entities with %44 accuracy.

Overall, TagMe annotation service is able to identify around %50 percent of all the pre-annoated entities.



Item sets	Frequency
Employment	444
Experience	408
Free will	407
Team	403
Business	366
Technology	348
Skill	347
Data Science	347

Item sets	Frequency
Customer	321
Data analysis	317
Experience,Employment	315
Team,Employment	306
Free will,Employment	303
Statistics	291
The Mail on Sunday	289
Free will,Experience	288

Item sets	Frequency
Business,Employment	283
Skill,Employment	281
Big data	281
You Will (song)	275
Skill,Experience	272
Business,Team	271
Team,Free will	270
Team,Experience	270

FIGURE 4.7: TagMe annotation with FP Growth algorithm

4.4.3 TagMe service with FP-Growth algorithm

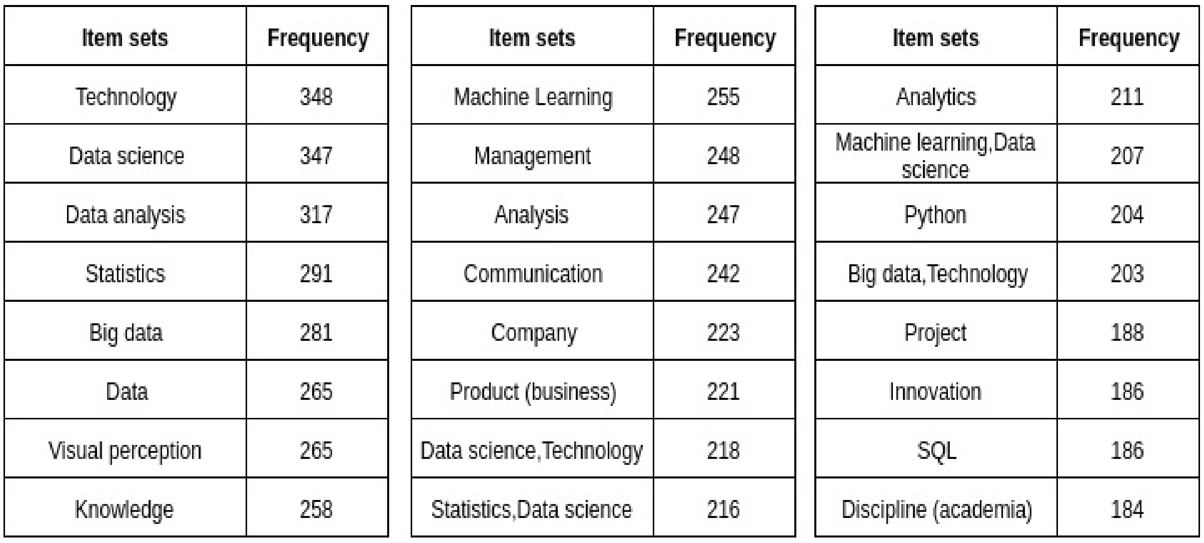
,

As it is illustrated in 4.6, collection of the raw text of the job ads is processed by the TagMe annotation service to extract entities per document. Then, the extracted entities are processed by the *FP Growth* algorithm so that we can capture the frequent entities in the job market. The results are given in figure 4.7 and they do not appear to be too informative. "Employment", "Business" or "Experience" are too generic terms that one could come across in any job ad. Also, there seems to be some noise in the results such as "Free Will" or "You Will (Song)" These entities do not convey any information about what the job market. We need to eliminate some of these entities.

4.4.4 TagMe with FP-Growth Algorithm with Filtering

As the results from applying *FP Growth* algorithm to the output of the TagMe annotation service produces poor results, we find it useful to filter some of the uninformative or irrelevant entities. Therefore we introduce a blacklist of terms: "Free will", "You Will (song)", "Skill", "Experience", "The Mail on Sunday", "Understanding", "Western (genre)", "Team", "Business", "Employment", "Solution", "Customer", "Tool", "Insight", "Organization", "Complexity", and so on. There are some more terms that we do not list here for brevity. The list consists of words that are deemed noise(ex. the extracted term "You Will (Song)" or too generic to be considered (ex. "Experience").

We ran the processed text that comes from TagMe service *FP Growth* algorithm on two different job ad data sets: job ads from summer 2016 and December 2016. The

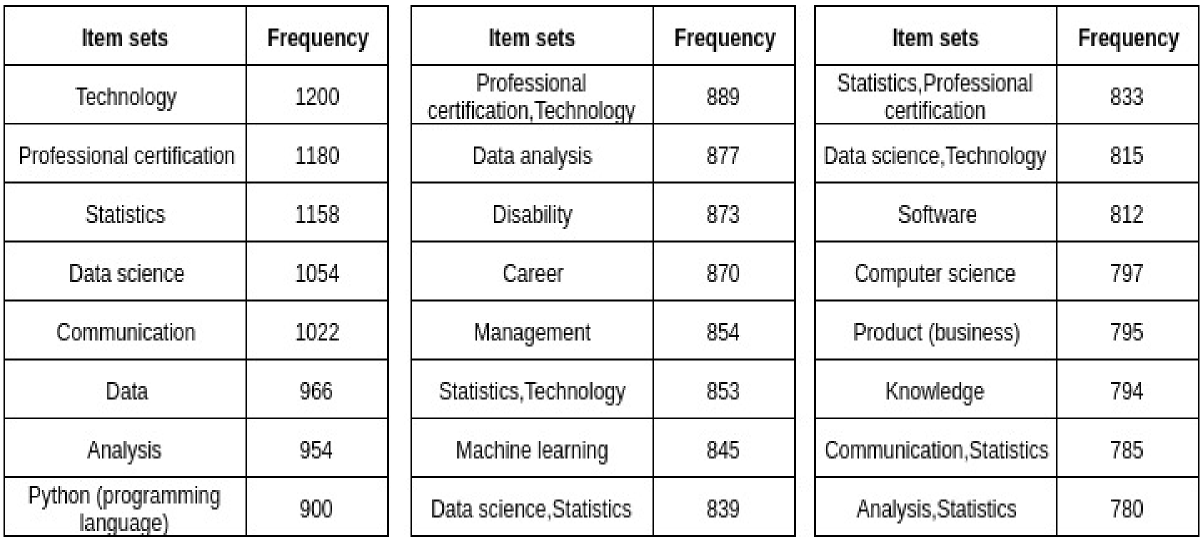


Item sets	Frequency
Technology	348
Data science	347
Data analysis	317
Statistics	291
Big data	281
Data	265
Visual perception	265
Knowledge	258
Machine Learning	255
Management	248
Analysis	247
Communication	242
Company	223
Product (business)	221
Data science,Technology	218
Statistics,Data science	216
Analytics	211
Machine learning,Data science	207
Python	204
Big data,Technology	203
Project	188
Innovation	186
SQL	186
Discipline (academia)	184

FIGURE 4.8: TagMe annotation with FP Growth algorithm on jobs from summer 2016 - Filtered with blacklist

results of these data sets are given in 4.8 and 4.9 respectively. One thing that quickly catches the eye is the clarity of the entity item sets. This is because TagMe links the extracted entities to Wikipedia articles. When we look at the results of summer 2016 dataset we see among the most the frequent item sets the notions "Data science", "Data Analysis", "Statistics", "Big Data", "Machine Learning" as well as tools such as "Python" and "SQL". These results gives us some idea about what the data science job market demand looks like. When we look at the results of December 2016 we see some overlapping terms such as "Statistics", "Data Science" and "Machine Learning".The demand is that a data scientists should be able to apply statistical machine learning algorithms. Also, the "Python (programming language)" appear on both data sets highlighting the importance of python language in data science tasks. There are also some differences between the results from these two data sets such as "Professional Certification" appearing very frequently in the December 2016 data set. This could be attributed to the job market willing to hire certified data scientists more.

Using TagMe annotation service with *FP Growth* algorithm produces clear results to extract frequent entities from our job ad corpora. Continuation of the analysis of the frequent patterns in job ad texts in the upcoming years could give us a better picture of how the data science job market demand changes over time.



Item sets	Frequency
Technology	1200
Professional certification	1180
Statistics	1158
Data science	1054
Communication	1022
Data	966
Analysis	954
Python (programming language)	900
Professional certification,Technology	889
Data analysis	877
Disability	873
Career	870
Management	854
Statistics,Technology	853
Machine learning	845
Data science,Statistics	839
Statistics,Professional certification	833
Data science,Technology	815
Software	812
Computer science	797
Product (business)	795
Knowledge	794
Communication,Statistics	785
Analysis,Statistics	780

FIGURE 4.9: TagMe annotation with FP Growth algorithm on jobs from December 2016 - Filtered with blacklist

Chapter 5

Future Work

5.1 Accumulate Traning Data for CV-Job Ad Matching

Apache Spark's MLLib module contains many supervised learning algorithms such as *Logistic Regression*, *Decision Tree Classifier*, *Random Forest Classifier*, *Naive-Bayes Classifier*, and more. However, these algorithms are *Supervised Learning Algorithms*, which means these classifier models require training data. One of the problems we tackled in this study is matching CVs with job ads and this is an *Unsupervised Learning Problem*. We did not have a set of CV and job ad pairs which were labeled as "good match" or "bad match". Therefore we were unable to leverage the classifiers mentioned above.

In chapter 3.4, we tried to get around the problem of not having a training data set by labeling the data we process using document similarity techniques. We assumed that the document similarity results which were based on TF-IDF were reliable results and we trained our *Multilayer Perceptron Classifier* using the labeled data.

One way of being able to make use of the supervised learning algorithms of the MLLib (machine learning library) module would be to accumulate training data for the task of CV and job ad matching. This would make our unsupervised learning problem a supervised one. One way of achieving this could be manually labeling the CV-job ad pairs using our human judgment. In this study we have around 16000 CV-job ad pairs therefore manually labeling data can be a cumbersome task. Another way could be using the results from other featurization methods, as it was done in chapter 3.4, but later on correcting the badly labeled CV-job ad pairs. After each such correction however, the underlying neural network has to be retrained.

5.2 Parsing CVs and Context-Specific Feature Engineering

In this study, we use the raw text of documents when document similarity techniques are used. We treat the whole of the text as our input and parse it by the space character. This implies we treat every word as a potentially important item that contributes to the overall meaning conveyed by the document. We rely on Spark's featurization methods which we run on the raw text of documents to extract our feature vectors. This could be improved.

If we had more in-depth information about a CV such as country, QS ranking of the university the CV owner graduated from, the modern technological keywords that are mentioned (such as Python, SQL, R, Machine Learning), experience in the field, etc. These extracted pieces of information can be collected in a vector. Then, the vector can be put through a number of transformations to generate a desired feature vector that is thought to represent the CV at hand. Similar in-depth analysis of the job ad descriptions can also be done. These context-specific feature vectors then can be passed to learning algorithms and we may acquire more accurate results.

The parsing of CVs and job ads is not a simple task. These documents have no obligation to follow consistent structures. One person may indicate all of their personal information in one line whereas others may list one piece of information per line. The number of cases one has to handle is quite high. This task alone could take months to implement therefore we refrained from doing it in this study. Alternative ways of parsing CVs could be achieved however, such as accepting CVs in predefined input structures. For example, when a CV is to be used in the system certain fields such as "experience", "graduate school", "country", etc. can be asked to fill in. Working with a structured data would ease the task.

5.3 Long Term Analysis of the Term Extraction Algorithms

In this study we make use of the *FP Growth* algorithm to detect frequent item sets in a collection of data science job ads. We do this to notice the trend in the data science job market so that EDISON project framework documents that define the data science profession may be updated. However, we only run the term extraction algorithms and methods on two different data sets: summer 2016 and December 2016. This does not encapsulate a large enough time window for our analysis to be complete. Therefore executing the algorithms and techniques we use in the study for a longer time window can help capture the changing demand of the data science job market.

Chapter 6

Conclusion

Apache Spark is a cluster computing framework that has been gaining popularity since its creation in 2009. Its ecosystem grew to include a SQL module for processing intermediate data using SQL syntax, a machine learning library to construct powerful machine learning pipelines for analyzing data and a graphx graph processing engine. In this study we aimed to make use of Apache Spark's MLLib (machine learning module), by considering two problem statements: Performing document similarity among CVs, job ads and competency documents; and performing term extraction in a collection of job ads to capture the changing job market demand. Both of these problem statements were considered to help the EDISON project improve the framework that defines Data Science as a profession.

In tackling the document similarity problem, we considered two main approaches: TF-IDF based approaches and Word2Vec.

TF-IDF based approaches mostly rely on counting terms. If a term appears many times within a document, it must be important to that document (TF). However, if the same term appears many times in many different documents then it's importance decreases (IDF). Spark provides two featurization methods within MLLib which are used to implement TF-IDF: *HashingTF*, and *Countvectorizer*. HashingTF is a hashing based approach that maps terms to vector indices using a hash function. Countvectorizer relies on counting terms and it is the standard way of implementing TF-IDF. We also implemented *Countvectorizer with PCA* which combines *Countvectorizer* and *Principal Component Analysis*. We could implement all of the said methods in a reasonable amount of time due to the expressivity of Apache Spark's MLLib API.

Another approach we used in tackling the document similarity problem was Word2Vec. Word2Vec creates a vector representation of words by taking into consideration the context of a word. In this vector space, closer vectors imply closer semantic meaning of the words that correspond to the said vectors.

We considered three types of documents to compare: CVs, job ads and competency texts. We used the aforementioned featurization methods and their combinations of algorithms like *PCA*. When evaluating the results, we used our human judgment to decide which featurization methods produced the most accurate results. When we considered all of the different methods employed, our results suggest that the methods can be listed from best to worst in this following order: *Countvectorizer with PCA*, *Countvectorizer without PCA*, *HashingTF*, and *Word2Vec*. Both *Countvectorizer* methods produced very similar results and they were best at finding out the most competent CVs in our data set. However, *PCA* dimensionality reduction increased the cosine distance values' variance and this made it easier to interpret the radar charts. *HashingTF* method also produced acceptable results. The fact that it uses a hash function makes it appealing if one is working with performance intensive applications. Lastly *Word2Vec* method produced very poor results. This can be attributed to the fact that when the Word2Vec data is trained, the words' vicinity is considered. We are working with CVs and job ads where word order is not quite important to consider.

In tackling the term extraction problem, we considered *FP Growth* algorithm. We preferred *FP Growth* over *Apriori* because of performance reasons. We applied the *FP Growth* algorithm either directly on our job ad dataset, or we used the TagMe annotation service first and then applied the *FP Growth* algorithm. In both cases we introduced a blacklist of terms to filter out the noise (unwanted, uninteresting terms).

We applied the term extraction algorithms and methods on two different data sets: summer 2016 and December 2016. Our aim was to capture the change in the job market demand. In both cases we discovered that terms such as "Machine Learning", "Python", "R", "Analysis", "Statistics" are quite important for data scientist employers. Our results only cover a small time window however. An analysis of common terms for a longer time period can help us identify the change of trend in the industry.

Apache Spark makes it possible to explore/implement different methods in a reasonable amount of time. Its highly expressive, readable API helps the developers focus more on thinking and testing different implementations and methods rather than spend time on details of implementing a particular method. Apache Spark is currently a framework that every aspiring data scientist should have a decent knowledge of.

Bibliography

[1] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[2] Edison Project. http://http://edison-project.eu/edison/edison-data-science-framework-edsf.

[3] Spiros Koulouzis. The EDISON COmpetencies ClassificatiOn (E-CO-2) service. Master's thesis, University of Amsterdam, Amsterdam, 2016.

[4] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. A comparative study of tf* idf, lsi and multi-words for text classification. *Expert Systems with Applications*, 38(3): 2758–2765, 2011.

[5] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

[6] Joeran Beel, Stefan Langer, Marcel Genzmehr, Bela Gipp, Corinna Breitinger, and Andreas Nürnberger. Research paper recommender system evaluation: a quantitative literature survey. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, pages 15–22. ACM, 2013.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] Word2Vec. http://spark.apache.org/docs/latest/ml-features.html#word2vec, .

[9] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[10] Sandeep Tata and Jignesh M Patel. Estimating the selectivity of tf-idf based cosine similarity predicates. *ACM Sigmod Record*, 36(2):7–12, 2007.

[11] Scipy Python Library. https://www.scipy.org/.

[12] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[13] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

[14] Parallel FP Growth Implementation. http://spark.apache.org/docs/latest/mllib-frequent-pattern-mining.html, .

[15] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 107–114. ACM, 2008.

[16] Austin Appleby. Murmurhash3 64-bit finalizer. Technical report, Version 19/02/15. https://code. google. com/p/smhasher/wiki/MurmurHash3.

[17] Stergios Stergiou, Zygimantas Straznickas, Rolina Wu, and Kostas Tsioutsiouliklis. Distributed negative sampling for word embeddings. In *AAAI*, pages 2569–2575, 2017.

[18] Steven M Holland. Principal components analysis (pca). *Department of Geology, University of Georgia, Athens, GA*, pages 30602–2501, 2008.

[19] Sonali B Maind, Priyanka Wankar, et al. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1):96–100, 2014.

[20] George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

[21] TagMe Short Text Annotation Service. https://tagme.d4science.org/tagme/.

[22] Natural language processing laboratory (nlplab). http://nlplab.org/.

[23] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5 (Apr):361–397, 2004.