UNIVERSITAT DE BARCELONA

FUNDAMENTALS OF DATA SCIENCE MASTER'S THESIS

GuruFinder

Author:
Marc CARDÚS GARCIA

Supervisor:
José MENA, Jordi VITRIÀ

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamentals of Data Science

in the

Facultat de Matemàtiques i Informàtica

September 26, 2018

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

GuruFinder

by Marc CARDÚS GARCIA

Imagine you are reading a newspaper, a blog, a scientific publication or a forum and you have become interested in a certain topic. After reading that site you may want to know more about it, so you click in a button and GuruFinder propose you a list of Twitter users to follow experts on that topic. The concept may be easy but the technology required underneath implies the combination of different disciplines including natural language processing, recommender systems, text mining, knowledge management systems and big data processing. GuruFinders pretends to explore state of the art techniques to build a prototype able to handle and process big volumes of tweets and offer real-time responses to the users.

Acknowledgements

To my friend and family who support me during this hard months, to my advisor for all what I have learned during this project and to Eurecat for providing me with their infrastructure.

Chapter 1

Introduction

1.1 Problem context

The Digital Revolution is changing every single aspect of our lives these days. The way we communicate, shop or perform mundane actions such as planning our vacations has been drastically changed last years due to the rise of the Digital Revolution. Nowadays anyone can easily publish on the internet its thoughts, experiences, dreams, pictures, ideas, technical doubts or professional profile. Different channels support this behaviour such as blogs (i.e. Blogger, Wordpress), microblogs (i.e. Twitter, Tumblr, Tecent Wibo), forums (i.e. Reddit, 4chan, ForoCoches), image-based social media (i.e. Flickr, Instagram), Q&A sites (i.e. Quora, Stackoverflow), employment-oriented social media (i.e. Linkedin, Indeed) and many others. In term of volume, Internet, smartphones, sensors, social media or multimedia content are contributing to an exponential growth of the accessible digital information (Hashem et al., 2015).

A possible use case for GuruFinder would be a user, let's say for example a Ph.D. student, that is reading a paper related to her thesis. After reading a section about Convolutional Neuronal Networks (CNN), she decides that is something worth to try in her research and wants to learn more about it, so she selects the section, clicks on the GuruFinder button and the tool proposes her a list of Twitter users to follow. Thus, she can access people that are publishing relevant information about CNNs: new papers, conferences, etc. Imagine now that you are reading an online newspaper. After reading an article about the Rohingya Refugee Crisis, you want to know more about the conflict, so you select the text and ask the tool to tell you which users to follow on Twitter: the opinion of different experts on the matter will provide you with different points of view.

1.2 Project method and outcomes

Last section mentioned channels hide useful information, but there is also non-relevant or non-related information to our interests. Being able to discern information by theme and relevance has become an important topic these days. To address this issue researchers in the field of Information Retrieval (Singhal, 2001) have been working to solve this challenge by applying state-of-the-art methods such as Latent-Dirichlet-Allocation (Blei, Ng, and Jordan, 2003) model or doc2vec neural network (Le and Mikolov, 2014), which over-perform classic methods like Tf-idf (*TF/IDF*) or vector-space-model (Singhal, 2001).

GuruFinder project address the problem with a specific approach. It does not assist sith searching and filtering of information but suggest you who has the relevant information (GuruFinder let you know who is the expert, also called Guru in a certain topic). Those people, that we call Gurus, have knowledge enough about the topic to detect, filter and curate the contents that are worthy to share. The outcome of this project is to create a prototype tool able to take a piece of text that is relevant for the user and recommend expert Twitter users (gurus) to follow.

1.3 Problem constraints

This thesis has two different constraints in order to have an accomplishable scope of work. On one side, the master's thesis focus in retrieving Gurus from an already defined list of expert users given a user input text (matching input texts topic with equivalent Gurus topic). There is no work thought on detecting new Gurus from Twitter (detecting whether a user is a Guru or it is not). On another side, GuruFinder uses an unsupervised (*Unsupervised learning intuition*) strategy as it is stated in chapter 3, so it is not bounded by topic, this means GuruFinder theoretically could work on any topic (any text), but this suppose a too open scope, so the training set (tweets) and the possible list of topics has been reduced to three different possibilities: feminism, cooking and technology. I selected these possible topics for being quite different between them and also to be popular topics on Twitter nowadays.

1.4 Motivations

For these reasons, GuruFinder is a challenging project which addresses a noble problem combining different state of the art techniques. Creating representations of texts using cutting-edge text models, indexing Twitter timelines and perform searches based on the selected texts in real time using big volumes of data, suppose an appealing challenge for anyone interested in a multidisciplinary project. GuruFinder is then an appealing idea where to apply obtained knowledge during master studies, combining all these mentioned fields and applying them to a real-life project with the collaboration of Eurecat.

Chapter 2

State of the Art Review

The GuruFinder main challenge consists of identifying topics from Twitter users timelines. This is relevant to accomplish the ultimate GuruFinder goal: to recommend a relevant list of Twitter users given an input text. To do so, GuruFinder establishes relations between a given input text and explored Twitter users.

To address this problem, the literature explores different strategies explained in the following lines.

One approach to tackle this problem concerns the extraction of named entities using the Named Entity Recognizer technique (NER) (Locke, 2009). This technique has been used in Twitter domain (Cooper, 2017) to obtain occurrences of specific objects. These objects are then classified into a predefined set of categories such as persons, locations, political organizations, facilities etc. For example, the sentence: "Gaudi planned the Sagrada Familia to be as tall as Collserola" would classify Gaudi as a person, and Sagrada Familia and Collserola as locations. Through the obtention of these entities, one can establish connections between different texts.

NER technique needs to be trained using a labelled set of data containing a label of each entity object. These labels may be obtained by hand using labelling services such as Amazon's Mechanical Turk (Finin et al., 2010) (*Human intelligence through an API*). Another possible approach to obtain these labels consist of using external sources of knowledge like Wikipedia. In the literature, there are examples of how to use these external sources of information to enhance the NER technique in Twitter (Yamada, Takeda, and Takefuji, 2015).

The literature also proposes a different approach rather than finding entities in a text, which consists of the creation of numerical representations from texts. There are different types of models capable to represent texts as vectors such as the vector space model (Singhal, 2001), also known as bag of words which is shortly explained in next paragraphs.

Latent Dirichlet Allocation (LDA) (Blei, Ng, and Jordan, 2003) model is also common in the literature (Duan and Ai, 2015) (Campr and Ježek, 2015) (Ramage et al., 2009) to generate numerical representations. LDA represents documents as a mixture of unobserved groups (topics) with a prior Dirichlet distribution. The topics are represented as a mixture of words with a probability representing the importance of the word for each topic.

Different types of embeddings are also used to represent texts as vectors specially doc2vec (Le and Mikolov, 2014), word2vec (Mikolov et al., 2013), glove (Pennington, Socher, and Manning, 2014) and fast text (Bojanowski et al., 2016). These embeddings are neural networks that overcome classical bag of words by taking consideration of the semantics and order of words.

The literature also proposes the usage of the singular-value decomposition, which is a matrix factorization technique. As can be read in the literature ("Text Summarization and Singular Value Decomposition") (Campr and Ježek, 2015), it is used over the bag-og-words matrix generating a vector representation though its singular vectors.

The creation of numerical text representations then it is a core idea in GuruFinder. To explain the intuitions behind it one may first understand the famous vector space model, where texts are transformed into vectors in the vector space. In the vector space model, a text is represented by a vector of terms where terms are typically words. Every term in the vocabulary (the total list of possible terms) represents an independent dimension in the vector space, so any text can be represented into this vector space (with length equal to the vocabulary size). If a text term exists in the vocabulary it gets a non zero value and not all terms are chosen to be part of the vocabulary since a tokenization process (which is explained later on) is applied to filter text terms.

So, given a numerical representation from a text, one may compute nearest vectors using, for example, K-Nearest-Neighbors (KNN) (Soni, 2018). KNN may use different ways to compute distances between vectors such as Euclidean distance, cosine similarity or Manhattan distance (Soni, 2018). Normally also more sophisticated models use the vector space model as input data to generate further numerical representations.

Another important GuruFinder challenge is the recommendation process in the Big Data context since GuruFinder pretends to be a tool able to work at scale. Working with large amounts of data requires distributed technologies (they provide horizontal scalability achieved by using clusters of machines) since single machine processes are strictly bounded by the single machine hardware resources.

The Big data technologies landscape (2018) has grown recently fueled by the interest of processing big sources of data, so many technologies have appeared within the last years. Elasticsearch (*Elasticsearch*), for example, is a distributed search engine commonly used to build recommendation systems due to its capabilities of processing text content at scale. Another state-of-the-art distributed tool is Spark () which is used to perform distributed computations. It may be a useful tool to deal with recommendation matrices and vectors. Spark has an API (*MLlib | Apache Spark*) with different recommendation models which may also found useful for implementing a distributed recommendation system. Actually, scalability is such an important piece of a recommender system since many times recommender systems have to deal with huge users and products matrices. For this reason, there is great literature about how to create a state-of-the-art recommender working at scale (Covington, Adams, and Sargin, 2016).

Another state-of-the-art technique to build a scalable system consists of building a microservices architecture. This idea is based into having a highly modular code, where each component runs a specific small isolated service. The microservice paradigm provides a decentralized approach to building software, which is opposed to classic monolithic services where all of the code is in one principal executable file. A microservices architecture allows running multiple instances of the required service when needed in one or multiple machines without the overhead of running the entire system.

Tokenization (*Tokenization*) is also an important task when working using text. Most common natural language processing models in multiple fields like topic modelling models, text summarization or automatic labelling need to pre-process text to extract tokens, where tokens are pre-processed words. Common tokenization preprocessing includes filtering most and less frequent words, removing stop-words (most common words in a language providing no meaning) or removing punctuation. More advanced tokenization processes include for instance the elimination of words prefixes or suffixes, cleaning words to their root. This technique it is called stemming.

Chapter 3

GuruFinder approach

3.1 Specific goals

After some research of the state of the art described in Chapter 2, a specific list of goals has been created in order to produce a successful prototype able to accomplish GuruFinder ideas, next to these goals are listed:

- GuruFinder should be able to generate and index Twitter timelines numerical representations using the following list of methods: Latent-Dirichlet-Allocation (LDA) and two different embeddings, Word2Vec and Doc2Vec. I decided to select the previous list of methods to be the most commonly used in the literature (Campr and Ježek, 2015) (Duan and Ai, 2015) (Thanda et al., 2016) (Gupta and Varma, 2017).
- In an analogous way, GuruFinder should be able to generate numerical representations of a user's given texts (user texts received from the API) using the following methods: Latent-Dirichlet-Allocation (LDA) and two different embeddings, Word2Vec and Doc2Vec.
- GuruFinder should be able to crawl and index Twitter users timelines.
- GuruFinder should be able to provide its service through a REST API.
- Given an input text about feminism, technology or cooking (introduced through the API), GuruFinder should be able to generate a list of recommended Gurus to follow. Recommended Guru's topic must be correlated with the topic of the given user input text.
- GuruFinder should be able to handle and process big volumes of tweets and offer real-time responses to the users.
- GuruFinder should be easily deployable in order to replicate it in different hosting providers.

3.2 GuruFinder first approach

GuruFinder has three different possible approaches in order to solve the problem main goal, to recommend a relevant list of Twitter users given an input text.

- **Supervised approach**: Supervised models are simple to understand and evaluate but rely heavily on a source of labelled information since they need a labelled dataset for training. These models also lack the unsupervised model's extendability since supervised models can only learn what it has been labelled, not being able to learn for instance new labels. In GuruFinder context, supervised models are not able to learn new topics without adding new labels. Some labelling services like Amazon Mechanical Turk (Finin et al., 2010) (*Human intelligence through an API*) may help to label data.

- **Weak supervised approach**: Weak supervised or also called semi-supervised models share supervised model benefits as they are also easy to evaluate and understand. The difference between these models and regular supervised ones consists of the source of information for obtaining labelled datasets. Semi-supervised models are attached to an external online and independent source of labelled data which is automatically generating new labelled samples, so there is no need to manually label training data. To depict the previous idea, let's imagine a classifier model to identify dogs within images. Since Instagram is a source of information offering thousands of dogs photographies containing hashtags, one may use these hashtags as labels to overcome supervision issues. Instagram is constantly adding new photographies containing dogs and new labels made by humans, so the dog's classifiers may keep learning without manually adding new training data. The drawback of this technique is the dependency of the external source of information.

  In the literature, semi-supervised models regarding the natural language processing field (NLP) usually rely on Wikipedia to obtain training data (Yamada, Takeda, and Takefuji, 2015)

- **Unsupervised approach**: Unsupervised models have no need for a source of labelled data. These models learn from data inherent properties like data distribution, elements order or elements co-occurrences. Unsupervised models then are more extendible than supervised ones since they may keep learning without human intervention (new training labels). These models are also more independent than weakly supervised models since they don't rely on any external service (such as Wikipedia or Instagram). The drawback of unsupervised models consists on one side to understand the models since they tend to be complex. On the other side, evaluate unsupervised models results may be difficult since each unsupervised model may have a different type of evaluation (supervised models mostly use same evaluation metrics such as accuracy, F1-Score, precision or recall). Sometimes unsupervised algorithms build a supervised layer on top of them in order to use simpler evaluation metrics.

After analyzing the different options I decided to use the unsupervised approach for GuruFinder. The main reason behind this decision has been the extendability. I consider GuruFinder have to be a "live" tool capable to change and learn from Twitter without any human intervention. This is especially relevant if one plan GuruFinder as a tool capable of detecting new trends. Another point in favour of the unsupervised approach is to find topics that may either not be in Wikipedia yet or are too colloquial to be in Wikipedia (which may be ignored in a weak supervised approach). In a constant growing digital world, unsupervised algorithms may easily capture tendencies and obtain new knowledge. Furthermore, as it is detailed later on in chapter 5, GuruFinder uses a simple supervised layer on top of the unsupervised layer in order to evaluate GuruFinder recommendations, so evaluating GuruFinder results becomes much easier.

3.3 GuruFinder text representations

As it has been stated in the state-of-the-art chapter, there are several algorithms able to generate numerical representations from a text. GuruFinder prototype implements Latent-Dirichlet-Allocation, Doc2Vec and Word2vec. Following, these models are introduced.

3.3.1 LDA

As was described by its authors, M.Blei, Y.Ng and I.Jordan, Latent-Dirichlet-Allocation (LDA) (Blei, Ng, and Jordan, 2003) is a generative probabilistic model for collections of discrete data such as corpora. LDA is an unsupervised learning algorithm that describes a set of observations as a mixture of distinct categories following a Dirichlet prior distribution. LDA categories (typically refered as topics) are represented as a mixture of data elements (typically words) with a probability representing the importance of the element for each category. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus.

3.3.2 Doc2Vec

As introduced for the first time in (Le and Mikolov, 2014), Doc2Vec is an unsupervised algorithm that learns fixed-length pieces of texts like paragraphs or documents from a collection of documents. Internally it implements a neuronal network. It has gained popularity for generating text numerical representation for many purposes as document retrieval, web search or spam filtering. It gained popularity to overcome well-known bag of words (BOW). One possible way of evaluating the model is to compare numerical representations of similar documents.

As most neural networks, Doc2Vec needs big amounts of data to perform well.

3.3.3 Word2Vec

Word2vec (Le and Mikolov, 2014) is a method similar to Doc2Vec, also an ensemble, it represents words as high dimensional vectors, so that words that are semantically similar will have similar vectors. Internally learns to encode of words from their co-occurrence information. The more they appear together in large text corpora, the more they have a similar encoding.

Since it encodes words and not documents, in this project it will be tested to use Word2Vec to encode complete set tweets calculating the vector mean of the vectors forming the words of the document.

Similarly to Doc2Vec, it may be evaluated by comparing the numerical representation of similar words.

3.4 Text cleaning

Before using the above mentioned models it is needed to clean the given input text. This is a crucial step since I'll affect vocabulary size and quality. In this project next text cleaning techniques have been used:

- Lower case transformation: Simple but mandatory, otherwise same word could be two times in the vocabulary.

- Frequency filters: Words appearing too much or only a few times may be not representatives for the text and add noise to the model.
- Literals filter stop words filtering: Removing punctuation and stop words will increase the quality of our vocabulary.
- Stem filter: Useful to remove morphological affixes and compact the vocabulary.

Chapter 4

GuruFinder architecture

This chapter focus on the design and implementation of an architecture to match GuruFinder goals defined in chapter 3. To do so, first, a list of requirements has been collected from GuruFinder goals. After obtaining these requirements a conceptual view of the architecture is presented followed by the implementation view of the final architecture. Finally, a short guide explaining how to deploy the architecture is given.

4.1 Architecture requirements

The following list of requirements defines the aspects to be fulfilled by the solution architecture in order to match GuruFinder goals defined chapter 3.

- **Scalability**: GuruFinder architecture should be highly scalable capable to process big amounts of data in near real-time. This is an important criterion for extending the prototype to a product able to crawl, store and process large size of tweets.
- **Easy deployment**: GuruFinder architecture needs to have a simple deployment since Eurecat may want to deploy it into different host providers.
- **Extendability**: GuruFinder implements different text models depicted in section 3.3, the number of models may increase in the future in case new models want to be tested, then an easy extendability is expected from the delivered code.
- **Security**: Since Eurecat may want to deploy GuruFinder in different hosts providers and environments, it is important to assert a certain level of security to avoid possible exploitable vulnerabilities. Delivering a small document of security best practices would be encouraged.

4.2 Logical view

The following section presents a conceptual view of the solution architecture. This type of view pretends to map every single process and data source involving the final solution without deepening into the implementation detail.

4.2.1 Logical view diagram

Next diagram represents an overview of the main logical entities regarding the solution architecture. For a detailed explanation of each logical entity consult following chapter sub section.

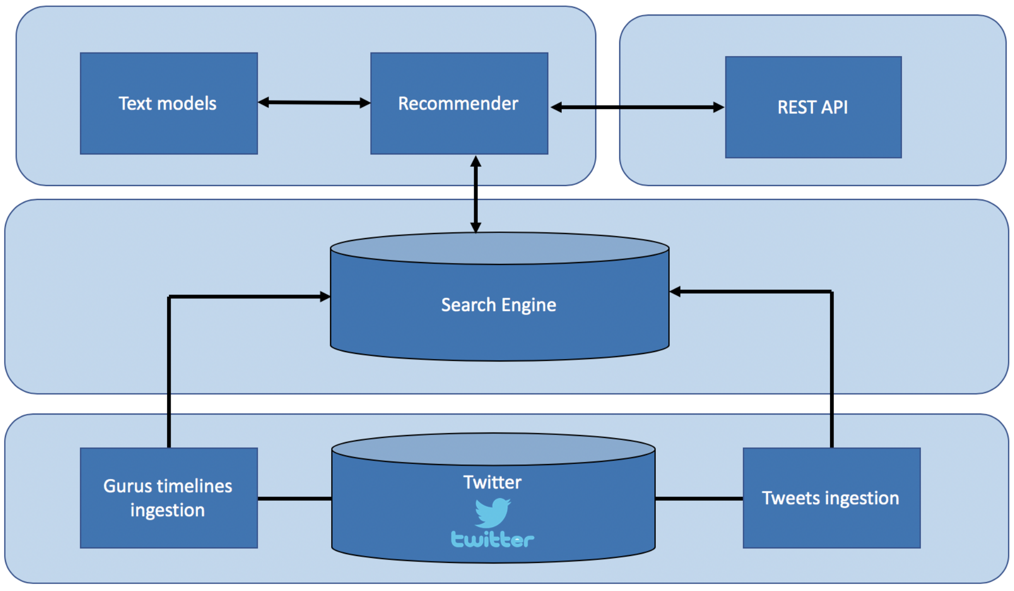

FIGURE 4.1: Main entities involving the architecture logical view.

4.2.2 GuruFinder ingestion

- **Gurus timelines ingestion**: This entity should access Twitter API and download specific selected users (users considered gurus) timeline. This data then is indexed into the search engine.
- **Tweets ingestion**: This entity should download real-time published tweets regarding the three GuruFinder topics: feminism, cooking and technology. To do so, it uses the Twitter API waiting for tweets containing a given list of keywords. The list of selected keywords is described in chapter 5. This data then is indexed into the search engine.

4.2.3 Search engine

The search engine plays a core role in this architecture. On one side it is responsible for storing every single document:

- **Gurus tweets**: complete gurus timelines
- **Keywords tweets**: Non Guru tweets. These tweets simulate the non Gurus users, which also help to train each text model.
- **Guru codes**: gurus timelines numerical representations
- **Exceptions**: Exception logs from all entities

On other side, it is also responsible to provide search functionalities to other entities. There are two search use cases:

- **To provide documents**: Provides stored documents to other entities (i.e. text models, recommender) applying different filters (dates, user id etc).

- **First recommendation layer**: Search engines are able to efficiently select similar documents. This utility may work as first recommendation layer, so a generation of preselected candidates in order to reduce the recommender (second recommendation layer) total search spectrum. The idea of having a first recommendation filter (two stages recommender) may be found in the literature for Big Data recommender systems (Covington, Adams, and Sargin, 2016) (Cheng et al., 2016).

Search engines uses different algorithms to select similar documents, for instance Elasticsearch uses vector space model and TF/IDF (*Theory Behind Relevance Scoring - Elasticsearch*) (Cutting and Pedersen, 1997) (*TF/IDF*) to find similar documents. Some Search engines are ElasticSearch, Solr or Azure search.

4.2.4 Text models

The text models entity implements the algorithms able to generate numerical representations from a text. This entity is in charge to fit and predict processes:

- **Fit process**: GuruFinder text models (LDA, Doc2Vec, Word2Vec and LDA) generate numerical representations (vectors) from a text. To fit GuruFinder text models from search engine data (tweets) is a crucial process since GuruFinder recommendation is based on numerical vectors representations similarity.

- **Predict process**: Predict process is in charge of loading already trained text models and use them to generate numerical representations of given texts. This process then is the one which generates numerical representations (one per text model) for each guru and also for input texts. Once the prediction process is finished, the recommender entity can look for nearest gurus respect to the input text.

4.2.5 Recommender

On one side, the recommender entity acts as the system orchestrator calling fit and predict processes when it is needed. It also uses the search engine to query data in order to feed the fit process from the text models entity.

On the other side, the recommendation process in GuruFinder is composed of two different layers (two stages). The reason behind this decision is to be able to run the recommender at scale. Since GuruFinder could potentially have thousands of millions of Twitter timelines, compute the nearest vectors for all of the numerical text representations may be too challenging, especially considering that one of GuruFinder's goals is to have a real-time recommendation response. This decision has been motivated by the literature where it is becoming popular to use a two stages recommender system in order to tackle Big Data scenarios (Covington, Adams, and Sargin, 2016) (Cheng et al., 2016).

Next, both first and seconds recommendation layers are described:

- **First recommendation layer**: This layer is applied by the search engine entity computing a list of similar gurus in respect of the input text vector. Search engines compute texts similarities to find near documents efficiently. One example is the Netflix pipeline, which uses Elasticsearch (a search engine) to query large amounts of logs at near real-time (less than a minute)(Blog, 2016).

- **Second recommendation laye**: This layer computes near gurus in respect of the user input text, considering only gurus filtered by the first recommendation layer. To do so, this layer generates numerical text representations of both gurus timelines and a user given text. Once the numerical text representations have been computed, it applies K-Nearest-neighbors (KNN) with the cosine similarity as the distance metric since it is the most common metric in the literature for GuruFinder domain (Soni, 2018) (Singhal, 2001). The second recommendation layer is expected to be much more accurate than the first one, but also slower to be computed.

4.2.6 REST API

The REST API is responsible to provide an interface to access to GuruFinder's services. It may be exposed to internet or may provide only local area network service for testing purposes.

4.2.7 Scalability by design

As it has been stated in the architecture requirements section, the resultant solution should be scalable in order to be deployed and extended as a product. To achieve this goal all entities should follow software engineering principles described in the state of the art review chapter, micro-services architecture and distributed architectures. These principles will lead to a high scalability, reliability and flexibility. Design the code following micro-services principle provides extra modularity and what is most important for GuruFinder, provides a simple way to distribute code in multiple machines deploying multiple instances of each entity (each service).

4.2.8 Security by design

To provide a certain degree of security should be guaranteed when deploying GuruFinder. To achieve this goal without going to deep into the cybersecurity domain, a three levels security layer may be a good solution:

- **Firewall**: Minimum possible ports should be exposed to the internet since each port may represent a vulnerability. In this context, a firewall blocking incoming petitions to ports not being used by GuruFinder performs an important job. Eurecat server (OpenStack) already provides a firewall, so this is already work done.
- **VPN**: To hide GuruFinder services under a VPN provides an extra level of security since a VPN user is needed in order to establish a connection with GuruFinder API. Eurecat TI also provides VPN for connecting Eurecat machines.
- **Access token**: Requiring an access token in order to access the GuruFinder API it is also an important security measure since it blocks petitions from not authenticated users.

4.3 Implementation view

Following section presents an implementation view of the solution architecture. This view depicts precisely which are the components and its technologies to be implemented and deployed into the final solution. In this section one will also find further details about each architecture component and its deployment.

4.3.1 Implementation diagram

Next diagram depicts the GuruFinder solution architecture. This diagram specifies which technology has been chosen for each component. Further details are exposed in following subsections.

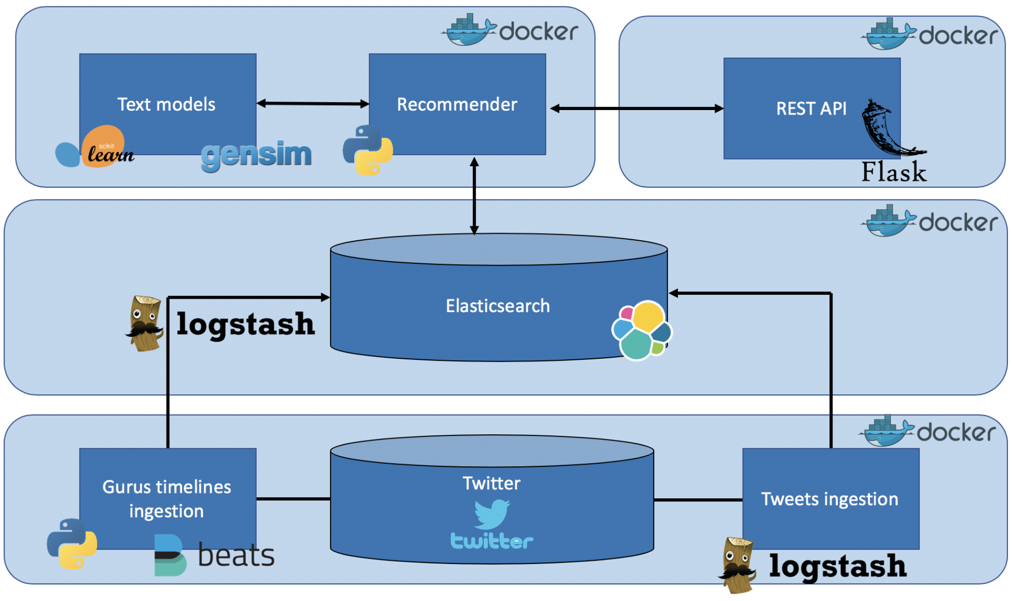

FIGURE 4.2: Main components and technologies involving the architecture implementation view.

4.3.2 Beats

Beats, also known as filebeats (*Filebeat*) it is a lightweight shipper for logs within the Elasticsearch ecosystem. It allows to send logs to Elasticsearch through Logstash, this helps the two Twitter ingestors to send their results to Elasticsearch.

4.3.3 Elasticsearch

Elasticsearch (*Elasticsearch*) is an open-source, distributable enterprise-grade search engine based on Apache Lucene (*Apache Lucene*). It is accessible through an extensive HTTP API, which can power fast searches that data discovery applications at near real-time. Big companies like Netflix (Blog, 2016) are using Elasticsearch on their pipelines to process their events.

Elasticsearch is in charge to store, and query all application data running into a multi-node fashion with a master node and a worker node, both deployed within a Docker container.

4.3.4 Logstash

Logstash (*Logstash*) is a Elasticsearch ecosystem technology to pre-process and transport data. It is in charge of ingesting and pre-process Gurus timelines which later on are shipped to Elasticsearch.

4.3.5 Tweets ingestion

Python Tweetpy (*tweepy*) is an easy-to-use Python library for accessing the Twitter API. It is used to ingests keywords (real time published) tweets.

4.3.6 Docker

All processes run within Docker (*Docker*) containers allowing simple deployments and isolation from internal and external dependencies. In future developments of GuruFinder, dockers could be orchestrated by a Kubernetes (*Production-Grade Container Orchestration*) cluster in order to gain scalability and control.

4.4 Deployment instructions

To deploy GuruFinder, use the folder called bin, which includes all the docker commands needed to deploy, fit, predict and recommend. Running bin/deploy.sh it's enough to wake up the base infrastructure: elasticsrach, filebeats and logstash.

4.5 API Acces

GuruFinder API only has one service, recommend, which may be used to receive recommendations, only using LDA text model (the one performing better as it is stated the chapter 5). GuruFinder API host IP is the 54.194.77.246. Next a sample request to acces GuruFinder API recommendations:

```
curl -X POST \
  https://54.194.77.246:8837/recommend \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "input_text": "Feminism␣it␣is␣about␣equallity",
    "num_recommendations": "15",
    "token": "3<1kdjDy47&9;>3"
    }'
```

4.6 Code repositories

All project code may be found in GitHub published by the user: @MCardus. The code has been split into two different repositories in order to increase reusability. On one side there is all the code regarding the text models, and on the other side, one may find the rest of the code into the following link GuruFinder.

Chapter 5

Results evaluation

5.1 GuruFinder evaluation

Selecting appropriated methods to evaluate an information retrieval system is a crucial decision in order to have a correct evaluation of performance. This section explores the GuruFinder evaluation pipeline and its different evaluation methods including visual ones and not visual ones.

5.1.1 Evaluation pipeline

GuruFinder ultimate goal, consist of recommending a relevant set of gurus to a user given an input text obtained through the GuruFinder API. A certain guru will be relevant to a user if the user's given input text topic is equal to the recommended guru topic. For example, given an input text about feminism, a recommended guru which is categorized as an expert in feminism will be a relevant recommendation. In this case, a recommended guru which is an expert in technology or cooking will be a non-relevant recommendation.

GuruFinder evaluation pipeline then it is about analyzing the relevance of the retrieved gurus given a set of gurus and a set of input texts. This type of evaluation is indeed the traditional one in an information retrieval system (IR) as shows the literature (Zuva and Zuva, 2012).

To analyze the GuruFinder performance one wants to observe how well GuruFinder retrieves gurus. To do so, an evaluation pipeline it has been built:

– **Pre-evaluation process**: The pre-evaluation process consist into all processes required before being able to start measuring the GuruFinder performance, this includes ingesting GuruFinder data (near two million tweets have been ingested in this phase), training the different text models (training phase) and generating the numerical representation for each guru (prediction phase).

  In order to evaluate the GuruFinder performance and following the problem constraints defined in chapter 1, a specific list of gurus has been ingested. This specific list of gurus has been designed to have three different topics: feminism, cooking and technology. This list of gurus it is explored later on in this chapter.

– **Evaluation process**: Once GuruFinder pre-evaluation process has been completed, it is time to start measuring GuruFinder performance. To do so, it is important to highlight two different facts. On one side, the system at this moment already contains a categorized list of gurus obtained during the pre-evaluation process which is constrained by containing only three different topics (gurus being experts either on feminism, technology or cooking). On another side, the GuruFinder evaluation is performed using a specific list of simulated user input texts which is explored and presented later on in this chapter. The list of simulated user input texts has been also designed to only contain three different topics: feminism, technology and cooking. Having a clear idea about this two highlighted facts, one can start measuring the GuruFinder performance.

To measure GuruFinder performance, an experiment has been realized for each text model and each simulated user input text combination (each text is about a specific topic from the list of possible topics). For instance, one possible test could be about measuring the performance for LDA using a feminism simulated input text.

The evaluated text models include LDA, Doc2vec, word2vec and elasticsearch vector space model (*Theory Behind Relevance Scoring - Elasticsearch*) (which corresponds to use the first recommendation layer only). As it has been stated in the last paragraphs, each experiment uses a different text model and simulated input text combination. An experiment consists then into obtaining the top N recommendations for each text model and simulated input text combination. Once the results are obtained, the different evaluation methods are applied to analyze how well have performed these N recommendations (these evaluation methods are described later on in this chapter). The GuruFinder evaluation has used an N value equal to 15, so 15 recommendations per experiment (15 gurus retrieved per experiment). The reason why I have selected N equal to 15 it is explained in the "Gurus list" subsection in this chapter.

5.1.2 Evaluation methods

Before explaining GuruFinder evaluation methods it is convenient to recall some basics about how to evaluate an information retrieval system.

Most frequently important basic measures for information retrieval effectiveness are precision, recall, and fall-out (Zuva and Zuva, 2012):

- **Precision**: Within a retrieval information system context, precision can be defined as the fraction of relevant retrieved items overall retrieved items.
- **Recall**: Also called sensitivity, it is the fraction of relevant items that have been retrieved over the total amount of relevant items.
- **Fall-out**: The proportion of non-relevant documents that are retrieved, out of all non-relevant documents available.

Figure 5.1 may help understanding above precision and recall definitions.

Precision, recall, and fall-out may also be defined as:

$$Precision = \frac{relevantDocuments \cap retrievedDocuments}{retrievedDocuments} \tag{5.1}$$

$$Recall = \frac{relevantDocuments \cap retrievedDocuments}{relevantDocuments} \tag{5.2}$$

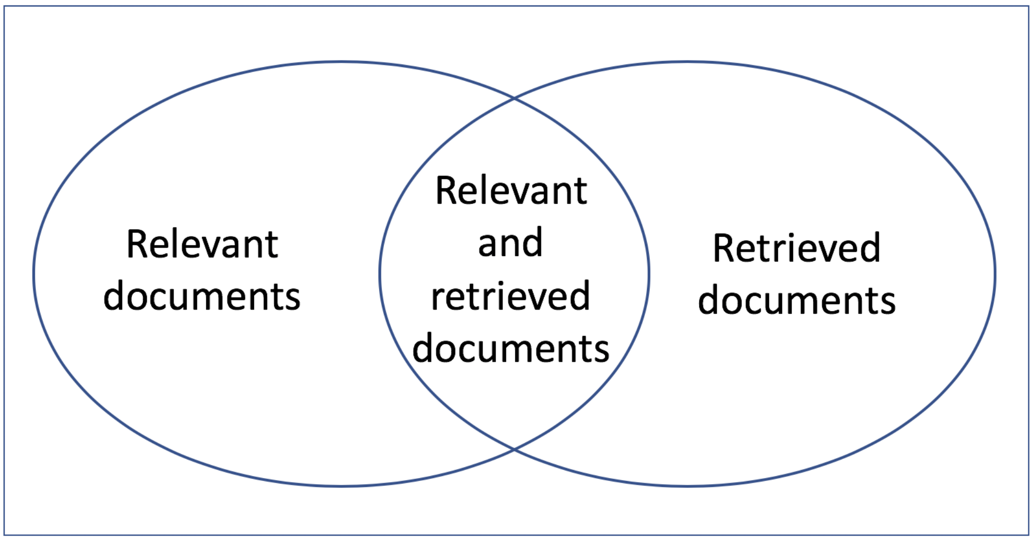

FIGURE 5.1: Information retrieval systems: relevant and retrieved documents

$$FallOut = \frac{nonRelevantDocuments \cap retrievedDocuments}{nonRelevantDocuments} \tag{5.3}$$

As it has been explained before in this chapter, a relevant recommendation (document) it is a recommendation which is relevant for the user, so the user input text topic matches the guru topic.

Following list explains used evaluation techniques for evaluating GuruFinder:

- Visual evaluations methods :
  * **Precision-recall curve**: Tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision.
  * **Roc-curve**: This curve provides the retrieval performance at all thresholds settings. Lowering the retrieval threshold classifies more documents as relevant, thus increasing both False Positives and True Positives
  * **Numerical representation plot**:
    · **LDAvis**: Visualization proposed in (Sievert and Shirley, 2014). It visualize topics in a 2D plot, which detect possible overlap between topics.
- Scalar evaluation:
  * **Mean Average precision**: Mean average precision for a set of queries is the mean of the average precision scores for each query, where the average precision is an evaluation method for systems returning a ranked sequence of documents such as retrieval information systems where the order in which documents are presented is a relevant feature (the recommendations are ordered by relevance).

Average precision computes the average value of $p(r)$ over interval $[0,1]$ where $p(r)$ is the precision-recall curve, that is the area under de precision-recall curve:

$$AveragePrecision = \int_0^1 p(r)dr \tag{5.4}$$

Finally, the equation (5.4), lead us to the computation of MAP (Mean average precision) which is represented by the next equation:

$$MAP = \frac{\sum_{q=1}^{Q} AveragePrecision(q)}{Q} \tag{5.5}$$

where Q is the number of queries.

The combination of visual and scalar evaluation methods gives a full picture and a definite answer about the performance of the system being evaluated as states (Zuva and Zuva, 2012). On one side scalar measures provide an overall value of performance of the system. On the other side, visual performance measure preserves all performance-related information about a retrieval system allowing to know whether system dominates the other system totally or partially.

5.1.3 Evaluation data

Different sets of data play an important role in evaluating gurus recommendations such as the list of ingested gurus, the list of simulated static input texts and the total numbers of tweets. Next, all these documents are listed in order to clarify which data is being used for the evaluation process.

Gurus list:

As it has been stated in chapter 1, GuruFinder problem has been constrained to three different topics: feminism, cooking and technology. To follow this constraint, a list of gurus has been designed where each guru is categorized belonging to one of the three possible topics.

The total amount of ingested gurus has been of 62 gurus. 30 feminism gurus, 15 technology gurus and 17 cooking gurus. Originally it was planned to have an equal number of gurus for each topic, but elasticsearch lost some data during the evaluation process, so the resultant dataset of gurus timelines is quite unbalanced.

Next, the list of gurus is presented for each topic:

- **Feminism**: EvrydayFeminism, FeminismInIndia, artandfeminism, WhovianFeminism, IncFeminism, feminiscience, geekfeminism, WomAgainstFem, HoodFeminism, FlyoverFeminism, BFandFem, NoToFeminism, nicolettemason, transscribe, pollyn1, bigmamamontse, bonniegrrl, Reductress, rossilynne, KristyPuchko, Ashadahya, naomirwolf, anitasarkeesian, pinkness, MeganMFinnerty, CTGreenParty, PrachiVidwans, CE_Zielinski, Ina_Steinbach, gillianmsmith.

– **Cooking**: CookingLight, CookingChannel, Wolfiesmom, cookingwithdog, CookingMatters, cookingvinyl, HonestCooking, SunnyAnderson, cookingfever, SandraLee, lorrainepascale, CookingDomains, rkhooks, kraftcanada, HairyBikers, rachelallen1, cookingdiary.

– **Technology**: techreview, technology, MIT, GIGABYTE_News, UTC , Huawei , SlackHQ, ZebraTechnology, LithiumTech, mapr , BBCTech, Akamai, CAinc, CenDemTech, DXCTechnology, NASA_Technology.

Since the minimum amount of gurus in a topic is 15, which is the total number of technology related gurus, the total number of experiments per text model and simulated input text has been set to also 15, otherwise, the experiment using the technology simulated input text will obtain for sure wrong recommendations.

As could be noticed, labelled texts do not contain their topic word in order to make tests a bit more realistic.

Simulated user input texts

The evaluation process ultimate goal consists of analyzing how well GuruFinder retrieves gurus given a user input text. When a retrieved guru belongs to a topic that matches the user input text topic (text introduced through the API by the user) the recommendation is considered a relevant one. Considering this scenario, a categorized list of gurus and a categorized list of simulated user input texts are needed. There are three simulated user input texts and each of these texts belongs to exactly one topic. These texts have been obtained from specialized web pages following two different criteria. One one side, I have look for texts not containing the topic keyword (for example, a feminism text not containing the 'feminism' word) in order to have a more realistic user text (a user looking for a feminism gurus may introduce a text about feminism which is not containing the word feminism itself). On the other side, I have tried to select texts which have a clear inclination for a topic in order to avoid an overlap of gurus. Following, the three simulated user input texts are presented:

– **Feminism**: They are not angry lesbians who hate men. They do not believe women are better than men, or that women deserve special privileges. They do not believe women are victims. In order to be considered as one, you only need to be on board with one idea: All humans, male and female, should have equal political, economic and social rights. Although more and more people are beginning to understand the true definition of this philosophy and openly identifying with it, there has always been a negative stigma attached to it. Part of this problem is the way our media sensationalizes things, trying to pass the most radical and extreme versions as the standard which, in this case, depicts it as a man-hater who hates lipstick, crinkles her nose at stay-at-home moms, and unapologetically supports abortions on demand.

– **Technology**: On 6 February 2018, Elon Musk's SpaceX successfully launched its Heavy Falcon rocket from the same Kennedy Space Center launch pad used for the Apollo missions. Atop the Falcon was Musk's own Tesla electric roadster, piloted by a mannequin named Starman who is seen lifting off to the tune of David Bowie's "Life on Mars." What would Nikola Tesla, the famous inventor, have thought about this? As a showman who took

250,000 volt shocks to demonstrate the safety of his alternating current AC inventions, Tesla would have been thrilled to see the car named after him orbiting in space. Such gestures, Tesla believed, wake people up and help them imagine on science in new ways. He would have wholeheartedly agreed with Musk who admitted that launching the car into space was "kind of silly and fun, but I think that silly and fun things are important."

– **Cooking**: In the East, you've got crunchy spring rolls, sushi, egg fried rice, steamed prawns, soy sauce, nuoc mam (a type of fish sauce), Peking duck, caramelised pork, chicken noodle soup, etc. In the West, you're more likely to find pasta, fondue, fish and chips, rack of lamb, etc. We're not going to argue which one is better since it all comes down to taste. Nevertheless, it's important to note the key ways in which Chinese, Vietnamese, and Japanese one, for example, can differ to western dishes even when the ingredients, like seafood, fish, chilli, and honey, are the same. Culturally speaking, Asian one tends to use opposing flavours in dishes, like mixing salty with sweet or sweet and sour together while a Western dish often focuses on a particular type of flavour like savoury or sweet.

5.2 Results presentation

After the previous section introducing results evaluation set up, this section plans to present and discuss obtained results.

5.2.1 Vocabulary

Final vocabulary has been formed by 500.371 keyword tweets and 1.174.113 Gurus tweets. Guru's tweets though are reduced filtering by current year. Since Gurus may change their main topics it is better to no consider old tweets, so any tweet older than the current year is ignored.

5.2.2 Recommending through text model LDA

Following, this subsection describes the obtained text model LDA recommendation results:

At this point, all needed flow has been completed and it is possible to start querying the system.

As it has been explained in the evaluation methods section, evaluation visual and scalar methods will be applied to evaluate and compare text models.

Appendix tables A.1, A.2 and A.3 depicts raw recommendation results using LDA. As it could be appreciated there is only 7 failed recommendation over 45 Gurus which are at the last positions of the ranking (I have performed the same test with more Gurus and results are the same, so I have preferred to use a small amount of Gurus). Consequently, the precision-recall curve (figure A.1) has a great are below the line.

Finally, scalar Mean average precision gives a score of 0,9344 out of 1.

5.2.3 Recommending through text model Doc2Vec

Current subsection presents Doc2Vec performance evaluation.

As can be seen in appendix tables A.4, A.5 and A.6 Doc2Vec did not have a good performance, it actually got a 0,4084 over 1 in the Mean average precision evaluation. After some study I have found five possible reasons why LDA outperforms that much Doc2Vec:

– **Unbalanced Gurus**: As will be clear during the evaluation of the rest of the models, there is an unbalance problem in the Gurus dataset. It remained hidden, but most of the text models (all but LDA) have great recommendation results with feminism topic (the one having more Gurus or bigger Gurus) and quite bad results with topics technology or cooking. Since Doc2Vec is an unsupervised algorithm by its nature it can not suffer on an unbalancing problem, but since is it being used as a supervised algorithm there may be problems regarding the unbalanced dataset.

– **Trainning set size**: Another feasible reason is that Doc2Vec, due to its nature of the neural network, may need more data for being correctly trained. At the moment it has been fit with 1 million tweets.

– **Model implementation**: GurFinder uses Gensim library for Doc2vec. Gensim is a popular machine learning library so its models should work fine, but may be interesting to test another library such as sklearn or tensor flow.

– **Hyperparameters tunning**: Finally, during this project, I have spent more time working on LDA, so maybe Dock2Vec could have a better hyperparameter tunning, which could improve performance. Currently, Doc2Vec is being trained using 20 epochs, a space of 300 dimensions per vector, alpha 0,025 and same tokens frequency filter as LDA.

– Finally, meanwhile, Doc2Vec focus on generating similar vectors for similar docs, LDA pretends to describe the whole dataset as a mixture of distinct categories. This fact may prevent LDA to have problems with unbalanced datasets since has a more global perspective with the whole dataset respect Doc2Vec.

In case I had more time I would have started to test Doc2Vec generating numerical representations of similar texts in order to understand if Doc2Vec results have a sense or not.

5.2.4 Recommending through text model Word2Vec

Word2Vec results in appendix tables A.7, A.8, A.9 are a bit similar to Doc2Vec ones (similar behavior), but being worse. In this case, though it is not a surprise since the usage of word2vec in this problem was a bit experimental. Let's remember that I have been using word2vec to generate 300 dimensions vector for each word in a tweet, this is many vectors per tweet. Then, at predict time a computation calculates a mean single vector of only 300 dimensions from each of the words vector. I wanted to test this since due to the nature of tweets being short documents.

Word2vec got a MAP score of 0.4117 out of 1.

5.2.5 Recommending through text model Elasticsearch firs recommendation layer only

Elasticsearch filter alone (results in appendix tables A.10, A.11, A.12) achieved a MAP score of 0,5257 , better than Doc2Vec and Word2Vec. This affirms the initial theory that is worth it to have a more simple model as a first recommendation layer. These simpler models run much quicker and reduce the search spectrum for the heavier seconds recommendation layer. Also this is fully provided by Elasticsearch engine, to it does not require to be trained on our side, neither a line of code.

5.3 Conclusion

Before concluding this project it is interesting to recall the initial objectives stated in chapter 3 in order to have an objective conclusion. This thesis, GuruFinder, had as the ultimate goal the creation of a prototype tool able to recommend Twitter expert users (gurus) from input texts given by the user. In order to accomplish this ultimate objective, a list of goals was created regarding the different aspects of the solution: a numerical representation of Twitter gurus and user's input text, the ingestion of historical and real-time Twitter data, the construction of a scalable architecture able to process big amounts of data, the construction of an API, the usage of security best practises and the integration of all these pieces into an scalable gurus recommender.

Considering the previous paragraph, I think that I have succeeded in the creation of a working prototype able to fulfil the GuruFinder ultimate goal. This project, which is a result of many hours of work, have room for improvement as it is stated in chapter 6, but nevertheless, the project succeeds in the construction of a working GuruFinder prototype.

Stepping into the details and going objective by objective, it is possible to find major and minor degrees of fulfilment, so the next subsections pretend to state a conclusion for each of the differences pieces regarding the project GuruFinder.

5.3.1 Gurus and input texts numerical representations

This project has implemented three different state-of-the-art models for generating numerical representations from a text. The obtained results are the expected ones, where LDA is the most performant model. LDA is the most common algorithm in the literature for solving this type of problem where we want to represent a collection of text as a mixture of topics. Word2vec, for instance, was initially not expected to perform well since it was made for representing words and not pieces of texts. Finally, doc2vec could probably have given better results with further work since it is a bit sensitive to the fine hyperparameters tunning.

In general, even having possibilities of improvement I think the numerical text representations coverage is quite okay, especially considering LDA have a good performance.

5.3.2 Twitter data ingestion

Twitter data ingestion, have been coded using two different technologies. On one side there is logstash, a distributed ETL technology in charge of ingesting real-time tweets and on the other side, there is Tweetpy, which has been used for the ingestion of historical tweets following the micro-services principle (being modular, lightweight, concrete and a containerized service). I think these two technologies provide a great tool of ingestion, capable to scale. A result of this is the near two million ingested tweets counting gurus and real-time tweets.

5.3.3 Architecture: scalability, replicability and security

In terms of architecture, this project offers a state-of-the-art solution, implementing a containerized microservices approach and also using distributed technologies such as elasticsearch, beats and logstash. This solution follows the current trends in the industry where these technologies are being used by leading companies such as Netflix.

In terms of replicability, thank docker-compose orchestration files, this project is quite easy to be re-deployed, which is mandatory for having a proper engineering solution.

Finally, about security, this project uses different security by design principles including the usage of a firewall, a proxy and an access token protecting the API and the entire system.

Chapter 6

Future work

6.1 Adding new text models

I had the idea of adding SVD as a possible model to generate numerical representations of tweets but I run out of time before having any serious result.

6.2 Work with Doc2Vec

If I would have more time, I would like to deepen into Doc2Vec performance problem. As has been explained in the previous chapter I have a different idea about why is it underperforming. Also using a different doc2vec implementation rather than gensim, could be an interesting test.

6.3 GuruFinder Heuristics

I had no time to do research on heuristics to detect Gurus. Since I already have the flow of incoming tweets from no gurus (keywords tweets), would have been nice to start doing some research to further develop a tool to locate Gurus from the keywords stream of tweets.

6.4 Chrome App

Creating a front end may be also a interesting step to go forward with GuruFinder increasing its usability.

6.5 Text models improved evaluation

A possible GuruFinder improvement may be to have better text models evaluations.

Implemented text model evaluations are good enough since they provide useful information to discern which model performs better. There is though at list two point that may be improved:

- **To have more balanced Gurus data set**: As it could be appreciated into the text models evaluation, there is no between Gurus for each different topic. This is not a crucial issue since it affects all models, so balance problem still allows to fairly compare models between each other. Still, to have a more balanced Gurus data sets could help to increase extracted insights.
- **Numerical representation 3D plot**: An interesting tool to analyze text models would be to show numerical representations in a plot with only three dimensions through PCA technique which reduce vectors dimensionality. Three dimensions allow 3D plotting usage. This feature may be an interesting utility to check whether vectors form 3D clusters or not.

Appendix A

GuruFinder results appendix

TABLE A.1: LDA Feminism topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	1	0.27	1	0
4	1	0.33	1	0
5	1	0.4	1	0
6	1	0.47	1	0
7	1	0.53	1	0
8	1	0.6	1	0
9	1	0.67	1	0
10	1	0.73	1	0
11	1	0.8	1	0
12	1	0.87	1	0
13	1	0.93	1	0
14	1	1	1	0

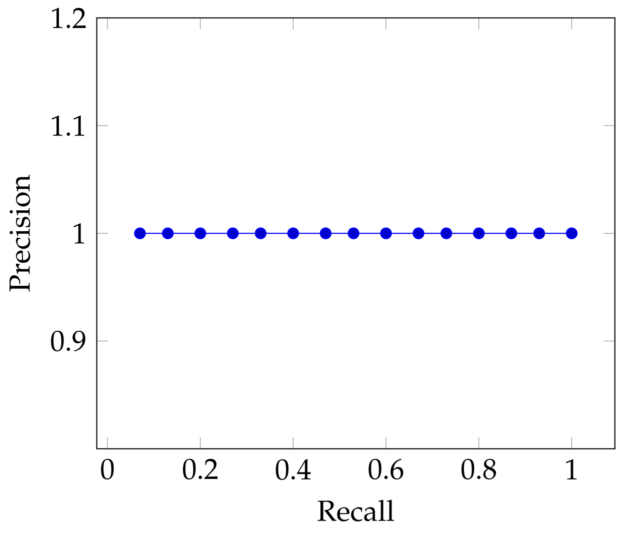

FIGURE A.1: Precision-recall curve

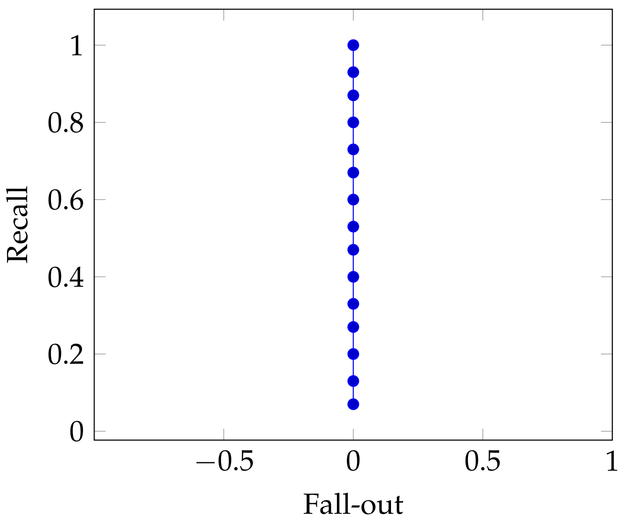

FIGURE A.2: Roc curve

TABLE A.2: LDA Technology topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	1	0.27	1	0
4	1	0.33	1	0
5	1	0.4	1	0
6	1	0.47	1	0
7	1	0.53	1	0
8	1	0.53	0.89	$3.000 \cdot 10^{-2}$
9	1	0.6	0.9	$3.000 \cdot 10^{-2}$
10	0	0.6	0.82	$7.000 \cdot 10^{-2}$
11	0	0.6	0.75	0.1
12	0	0.6	0.69	0.13
13	0	0.6	0.64	0.17
14	0	0.6	0.6	0.2

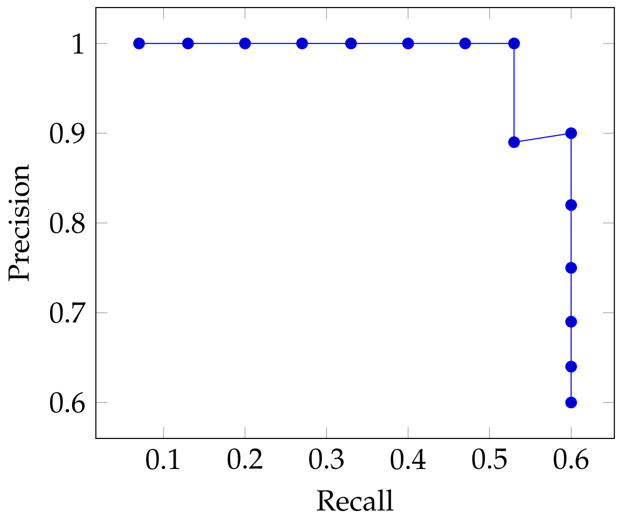

FIGURE A.3: Precision-recall curve

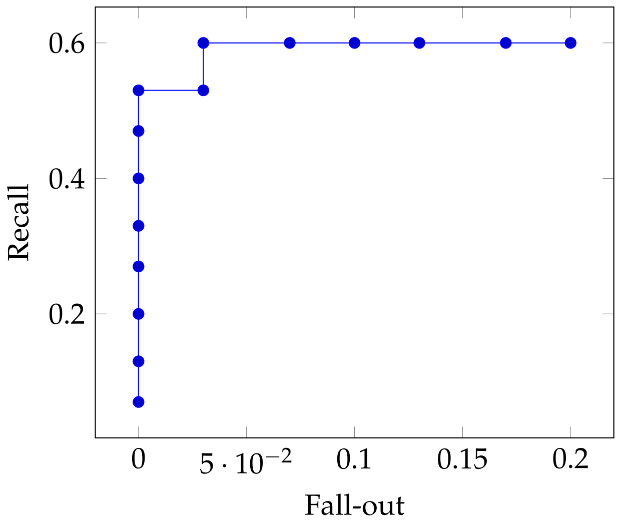

FIGURE A.4: Roc curve

TABLE A.3: LDA Cooking topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	1	0.27	1	0
4	1	0.33	1	0
5	1	0.4	1	0
6	1	0.47	1	0
7	1	0.53	1	0
8	1	0.6	1	0
9	1	0.67	1	0
10	1	0.73	1	0
11	1	0.8	1	0
12	1	0.87	1	0
13	0	0.87	0.93	$3.000 \cdot 10^{-2}$
14	0	0.87	0.87	$7.000 \cdot 10^{-2}$

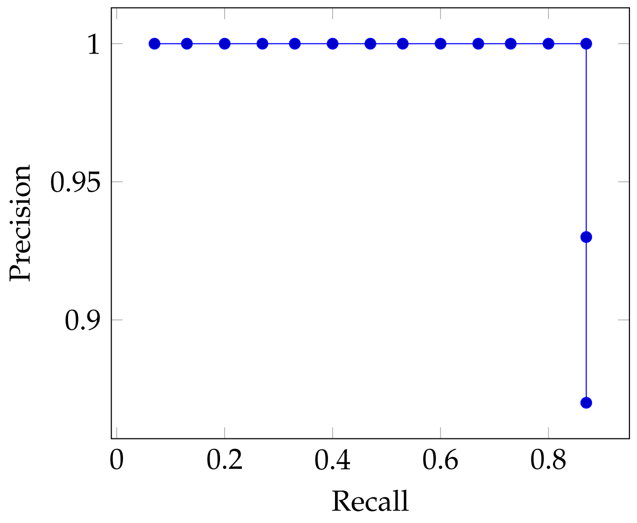

FIGURE A.5: Precision-recall curve

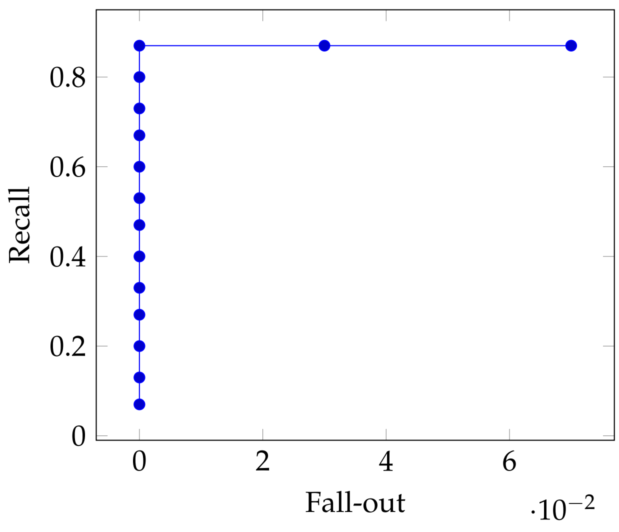

FIGURE A.6: Roc curve

TABLE A.4: Doc2vec Feminism topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	1	0.27	1	0
4	1	0.33	1	0
5	1	0.4	1	0
6	1	0.47	1	0
7	1	0.53	1	0
8	1	0.6	1	0
9	1	0.67	1	0
10	1	0.73	1	0
11	1	0.8	1	0
12	1	0.87	1	0
13	1	0.93	1	0
14	1	1	1	0

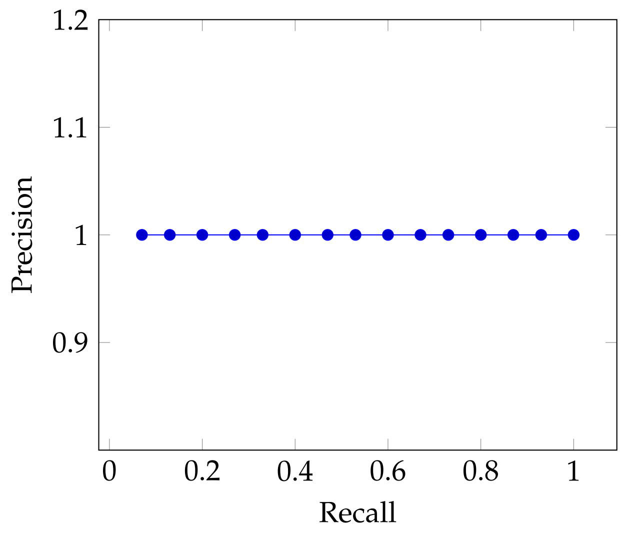

FIGURE A.7: Precision-recall curve

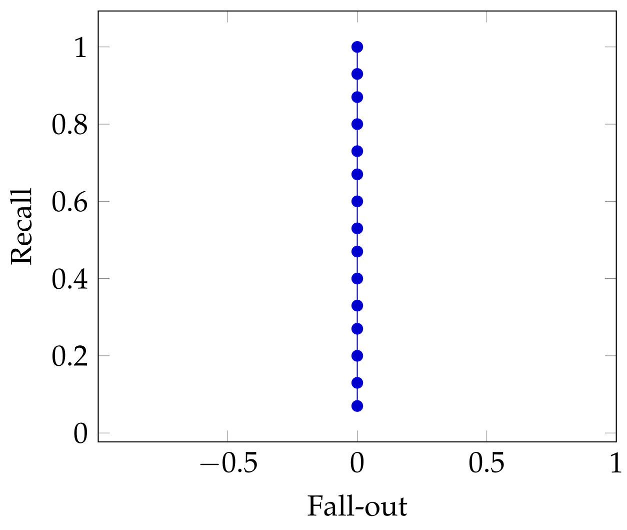

FIGURE A.8: Roc curve

TABLE A.5: Doc2vec Technology topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	0	0	0	$3.000 \cdot 10^{-2}$
1	0	0	0	$7.000 \cdot 10^{-2}$
2	1	$7.000 \cdot 10^{-2}$	0.33	$7.000 \cdot 10^{-2}$
3	0	$7.000 \cdot 10^{-2}$	0.25	0.1
4	1	0.13	0.4	0.1
5	1	0.2	0.5	0.1
6	0	0.2	0.43	0.13
7	0	0.2	0.38	0.17
8	0	0.2	0.33	0.2
9	0	0.2	0.3	0.23
10	0	0.2	0.27	0.27
11	0	0.2	0.25	0.3
12	1	0.27	0.31	0.3
13	1	0.33	0.36	0.3
14	0	0.33	0.33	0.33

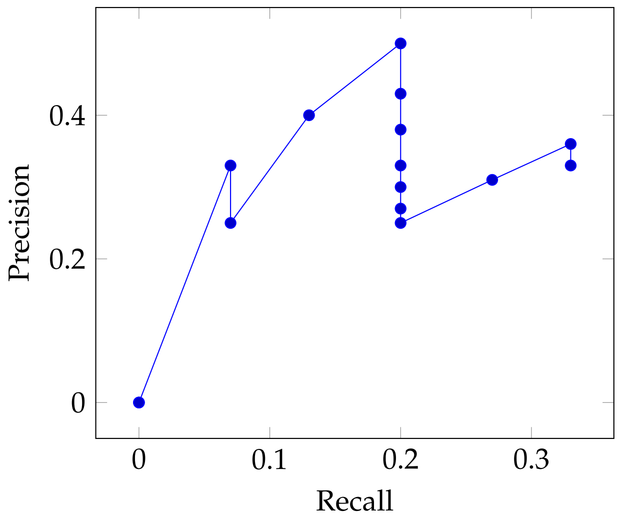

FIGURE A.9: Precision-recall curve

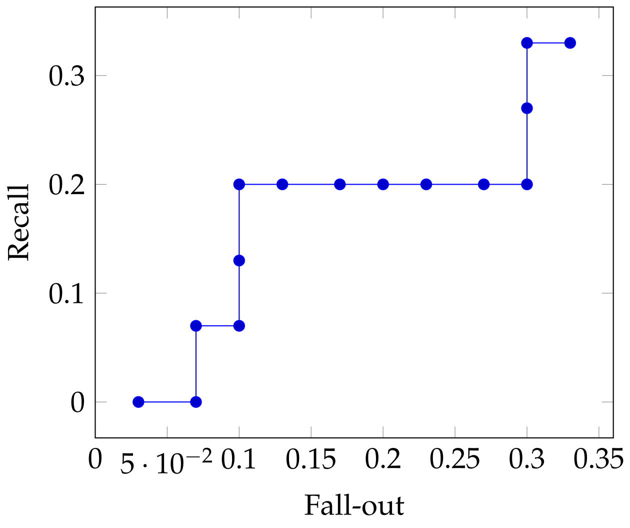

FIGURE A.10: Roc curve

TABLE A.6: Doc2vec Cooking topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	0	0	0	$3.000 \cdot 10^{-2}$
1	1	$7.000 \cdot 10^{-2}$	0.5	$3.000 \cdot 10^{-2}$
2	0	$7.000 \cdot 10^{-2}$	0.33	$7.000 \cdot 10^{-2}$
3	0	$7.000 \cdot 10^{-2}$	0.25	0.1
4	0	$7.000 \cdot 10^{-2}$	0.2	0.13
5	0	$7.000 \cdot 10^{-2}$	0.17	0.17
6	0	$7.000 \cdot 10^{-2}$	0.14	0.2
7	0	$7.000 \cdot 10^{-2}$	0.13	0.23
8	1	0.13	0.22	0.23
9	1	0.2	0.3	0.23
10	1	0.27	0.36	0.23
11	1	0.33	0.42	0.23
12	0	0.33	0.38	0.27
13	0	0.33	0.36	0.3
14	1	0.4	0.4	0.3

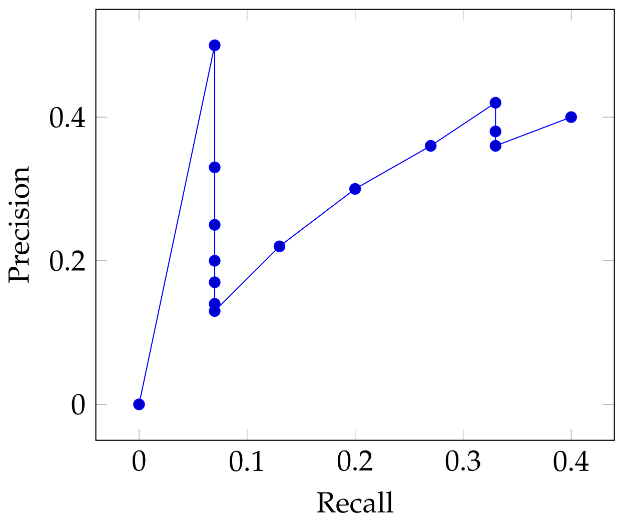

FIGURE A.11: Precision-recall curve

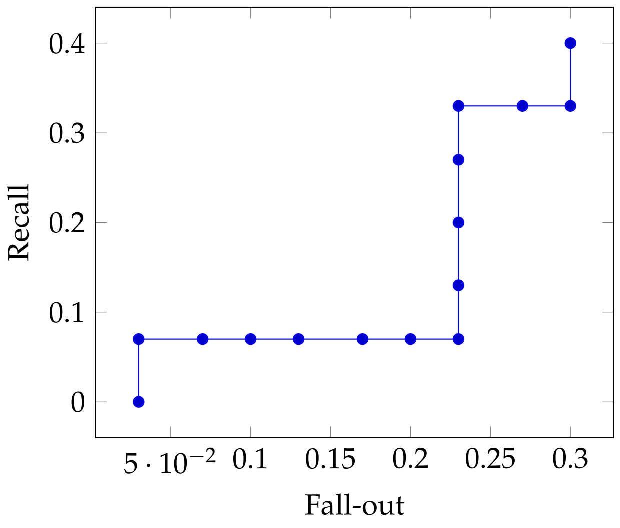

FIGURE A.12: Roc curve

TABLE A.7: Word2vec Feminism topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	1	0.27	1	0
4	1	0.33	1	0
5	1	0.4	1	0
6	1	0.47	1	0
7	1	0.53	1	0
8	1	0.6	1	0
9	1	0.67	1	0
10	1	0.73	1	0
11	1	0.8	1	0
12	1	0.87	1	0
13	1	0.93	1	0
14	1	1	1	0

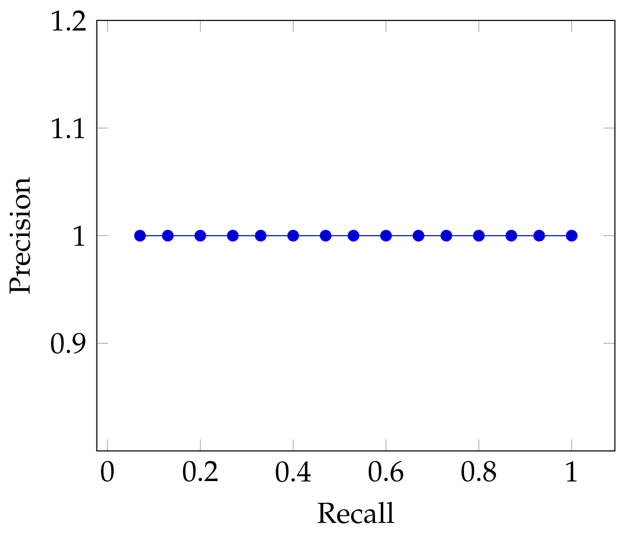

FIGURE A.13: Precision-recall curve

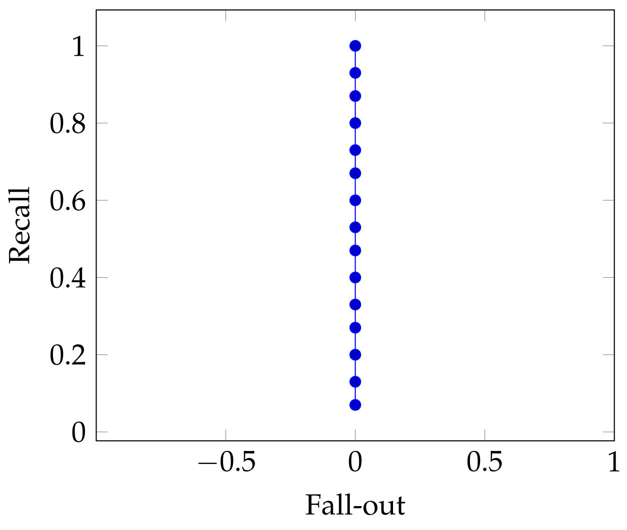

FIGURE A.14: Roc curve

TABLE A.8: Word2vec Technology topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	0	0	0	$3.000 \cdot 10^{-2}$
1	0	0	0	$7.000 \cdot 10^{-2}$
2	0	0	0	0.1
3	0	0	0	0.13
4	0	0	0	0.17
5	0	0	0	0.2
6	0	0	0	0.23
7	1	$7.000 \cdot 10^{-2}$	0.13	0.23
8	0	$7.000 \cdot 10^{-2}$	0.11	0.27
9	0	$7.000 \cdot 10^{-2}$	0.1	0.3
10	0	$7.000 \cdot 10^{-2}$	$9.000 \cdot 10^{-2}$	0.33
11	0	$7.000 \cdot 10^{-2}$	$8.000 \cdot 10^{-2}$	0.37
12	0	$7.000 \cdot 10^{-2}$	$8.000 \cdot 10^{-2}$	0.4
13	1	0.13	0.14	0.4
14	1	0.2	0.2	0.4

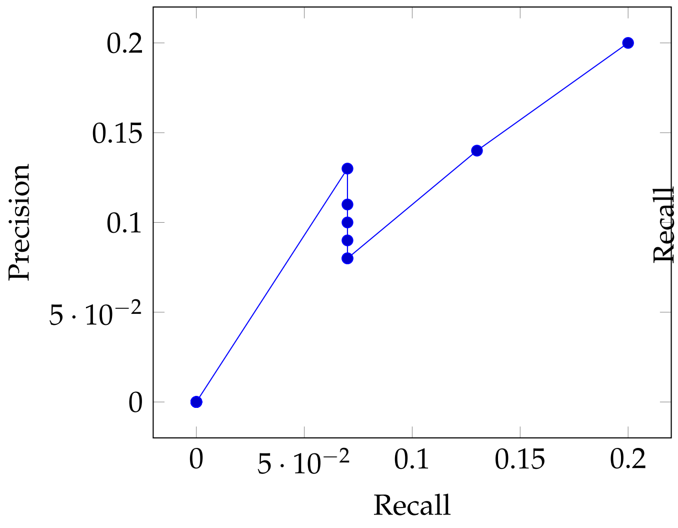

FIGURE A.15: Precision-recall curve

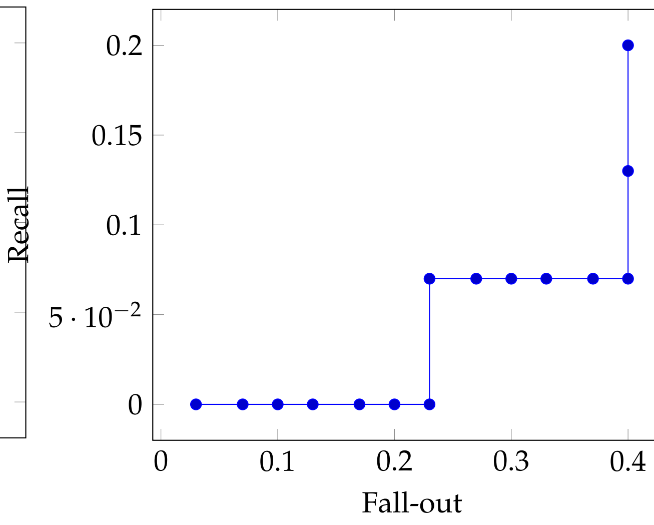

FIGURE A.16: Roc curve

TABLE A.9: Word2vec Cooking topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	0	0	0	$3.000 \cdot 10^{-2}$
1	1	$7.000 \cdot 10^{-2}$	0.5	$3.000 \cdot 10^{-2}$
2	0	$7.000 \cdot 10^{-2}$	0.33	$7.000 \cdot 10^{-2}$
3	0	$7.000 \cdot 10^{-2}$	0.25	0.1
4	0	$7.000 \cdot 10^{-2}$	0.2	0.13
5	0	$7.000 \cdot 10^{-2}$	0.17	0.17
6	0	$7.000 \cdot 10^{-2}$	0.14	0.2
7	0	$7.000 \cdot 10^{-2}$	0.13	0.23
8	1	0.13	0.22	0.23
9	1	0.2	0.3	0.23
10	1	0.27	0.36	0.23
11	1	0.33	0.42	0.23
12	0	0.33	0.38	0.27
13	0	0.33	0.36	0.3
14	1	0.4	0.4	0.3

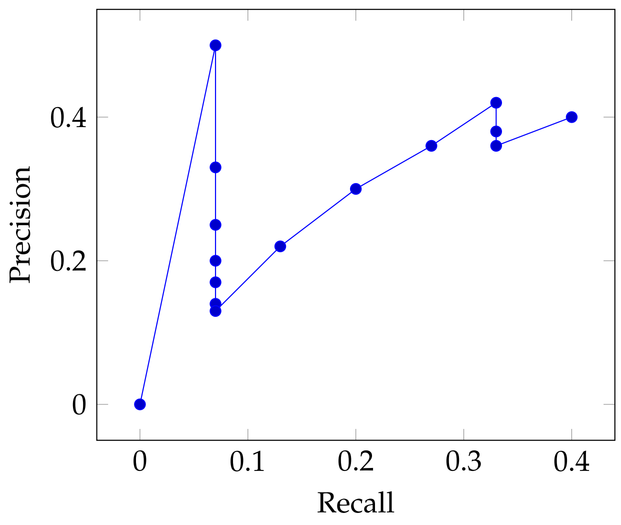

FIGURE A.17: Precision-recall curve

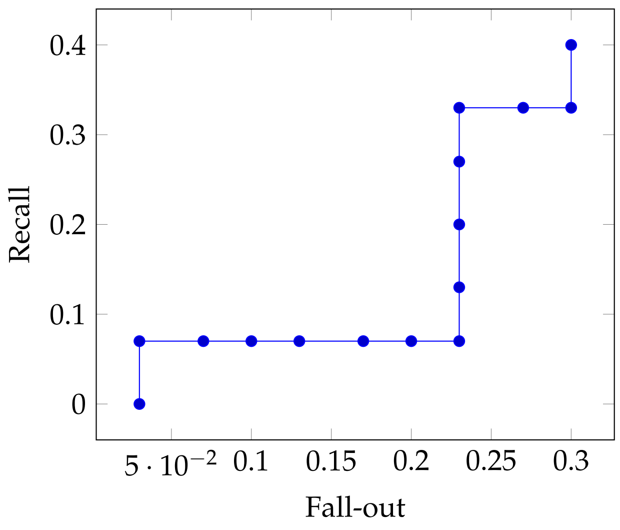

FIGURE A.18: Roc curve

TABLE A.10: Elasticsearch Feminism topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	1	0.27	1	0
4	1	0.33	1	0
5	1	0.4	1	0
6	1	0.47	1	0
7	1	0.53	1	0
8	1	0.6	1	0
9	1	0.67	1	0
10	1	0.73	1	0
11	1	0.8	1	0
12	1	0.87	1	0
13	1	0.93	1	0
14	1	1	1	0

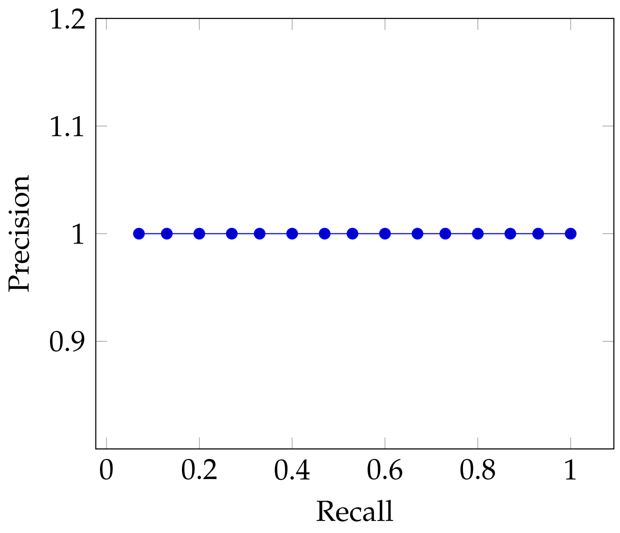

FIGURE A.19: Precision-recall curve

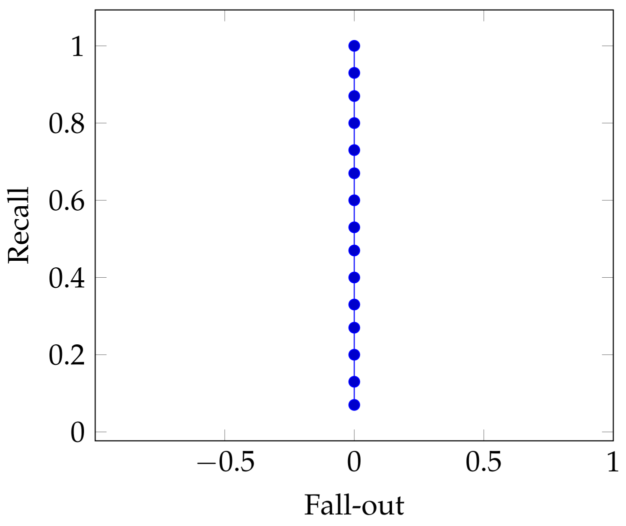

FIGURE A.20: Roc curve

TABLE A.11: Elasticsearch Technology topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	1	$7.000 \cdot 10^{-2}$	1	0
1	1	0.13	1	0
2	1	0.2	1	0
3	0	0.2	0.75	$3.000 \cdot 10^{-2}$
4	0	0.2	0.6	$7.000 \cdot 10^{-2}$
5	1	0.27	0.67	$7.000 \cdot 10^{-2}$
6	1	0.33	0.71	$7.000 \cdot 10^{-2}$
7	0	0.33	0.63	0.1
8	0	0.33	0.56	0.13
9	0	0.33	0.5	0.17
10	1	0.4	0.55	0.17
11	0	0.4	0.5	0.2
12	0	0.4	0.46	0.23
13	1	0.47	0.5	0.23
14	1	0.53	0.53	0.23

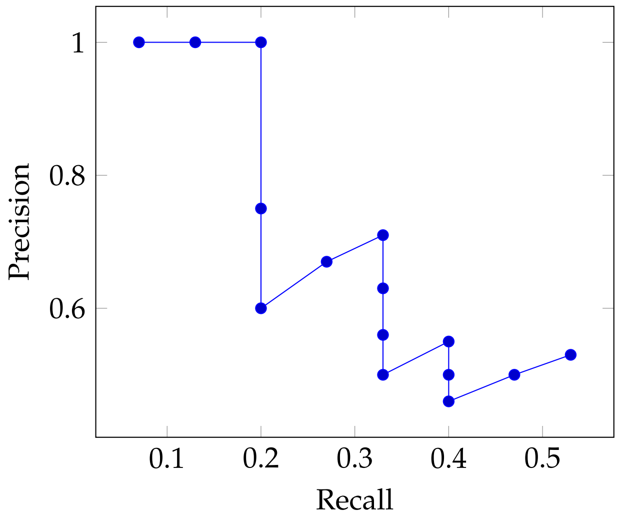

FIGURE A.21: Precision-recall curve

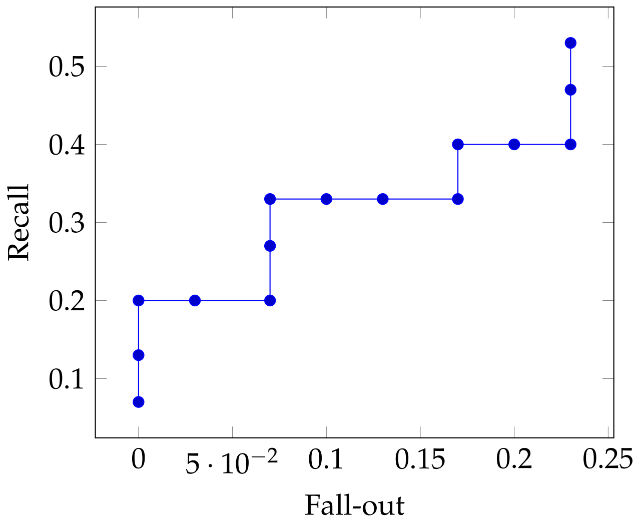

FIGURE A.22: Roc curve

TABLE A.12: Elasticsearch Cooking topic experiment results

Experiment	Relevant	Recall	Precision	Fall-out
0	0	0	0	$3.000 \cdot 10^{-2}$
1	1	$7.000 \cdot 10^{-2}$	0.5	$3.000 \cdot 10^{-2}$
2	0	$7.000 \cdot 10^{-2}$	0.33	$7.000 \cdot 10^{-2}$
3	0	$7.000 \cdot 10^{-2}$	0.25	0.1
4	0	$7.000 \cdot 10^{-2}$	0.2	0.13
5	0	$7.000 \cdot 10^{-2}$	0.17	0.17
6	0	$7.000 \cdot 10^{-2}$	0.14	0.2
7	0	$7.000 \cdot 10^{-2}$	0.13	0.23
8	1	0.13	0.22	0.23
9	1	0.2	0.3	0.23
10	1	0.27	0.36	0.23
11	1	0.33	0.42	0.23
12	0	0.33	0.38	0.27
13	0	0.33	0.36	0.3
14	1	0.4	0.4	0.3

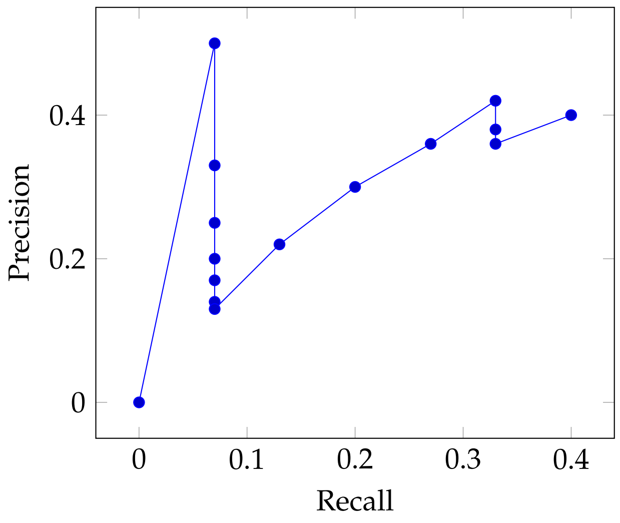

FIGURE A.23: Precision-recall curve

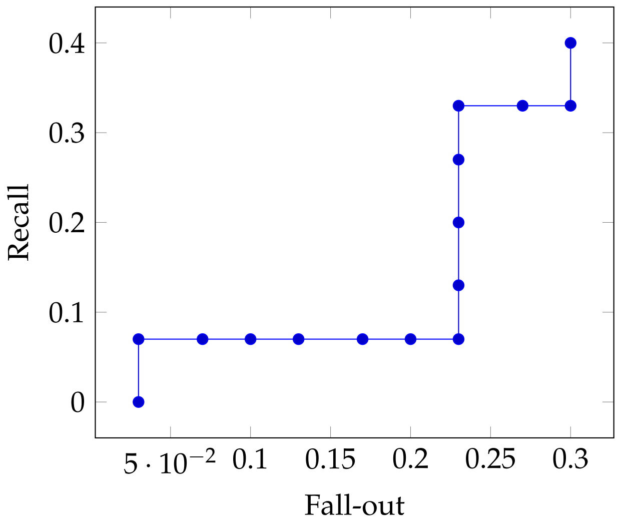

FIGURE A.24: Roc curve

Bibliography

URL: https://spark.apache.org/.

(2018). URL: https://pkghosh.wordpress.com/2013/09/22/big-road-map-for-big-data/.

Apache Lucene. URL: http://lucene.apache.org/.

Blei, David M, Andrew Y Ng, and Michael I Jordan (2003). “Latent dirichlet allocation”. In: *Journal of machine Learning research* 3.Jan, pp. 993–1022.

Blog, Netflix Technology (2016). *Netflix Data Pipeline*. URL: https://medium.com/netflix-techblog/evolution-of-the-netflix-data-pipeline-da246ca36905.

Bojanowski, Piotr et al. (2016). “Enriching word vectors with subword information”. In: *arXiv preprint arXiv:1607.04606*.

Campr, Michal and Karel Ježek (2015). “Comparing semantic models for evaluating automatic document summarization”. In: *International Conference on Text, Speech, and Dialogue*. Springer, pp. 252–260.

Cheng, Heng-Tze et al. (2016). “Wide & deep learning for recommender systems”. In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, pp. 7–10.

Cooper, George (2017). *George Cooper*. URL: http://blog.thehumangeo.com/twitter-ner.html.

Covington, Paul, Jay Adams, and Emre Sargin (2016). “Deep neural networks for youtube recommendations”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, pp. 191–198.

Cutting, Douglass R. and Jan O. Pedersen (1997). “Space Optimizations for Total Ranking”. In: *Computer-Assisted Information Searching on Internet*. RIAO ’97. Montreal, Quebec, Canada: LE CENTRE DE HAUTES ETUDES INTERNATIONALES D’INFORMATIQUE DOCUMENTAIRE, pp. 401–412. URL: http://dl.acm.org/citation.cfm?id=2856695.2856731.

Docker. URL: https://www.docker.com/.

Duan, Jianyong, Yamin Ai, et al. (2015). “LDA topic model for microblog recommendation”. In: *Asian Language Processing (IALP), 2015 International Conference on*. IEEE, pp. 185–188.

Elasticsearch. URL: https://www.elastic.co/products/elasticsearch.

Filebeat. URL: https://www.elastic.co/products/beats/filebeat.

Finin, Tim et al. (2010). “Annotating named entities in Twitter data with crowdsourcing”. In: *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon’s Mechanical Turk*. Association for Computational Linguistics, pp. 80–88.

Gupta, Shashank and Vasudeva Varma (2017). “Scientific Article Recommendation by using Distributed Representations of Text and Graph”. In: *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, pp. 1267–1268.

Hashem, Ibrahim Abaker Targio et al. (2015). "The rise of "big data" on cloud computing: Review and open research issues". In: *Information Systems* 47, pp. 98 –115. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2014.07.006. URL: http://www.sciencedirect.com/science/article/pii/S0306437914001288.

Human intelligence through an API. URL: https://www.mturk.com/.

Ježek, Karel and Josef Steinberger. "Text Summarization and Singular Value Decomposition". In:

Le, Quoc and Tomas Mikolov (2014). "Distributed representations of sentences and documents". In: *International Conference on Machine Learning*, pp. 1188–1196.

Locke, Brian William (2009). "Named entity recognition: Adapting to microblogging". In:

Logstash. URL: https://www.elastic.co/products/logstash.

Mikolov, Tomas et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

MLlib | Apache Spark. URL: https://spark.apache.org/mllib/.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Production-Grade Container Orchestration. URL: https://kubernetes.io/.

Ramage, Daniel et al. (2009). "Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora". In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*. Association for Computational Linguistics, pp. 248–256.

Sievert, Carson and Kenneth Shirley (2014). "LDAvis: A method for visualizing and interpreting topics". In: *Proceedings of the workshop on interactive language learning, visualization, and interfaces*, pp. 63–70.

Singhal, Amit et al. (2001). "Modern information retrieval: A brief overview". In: *IEEE Data Eng. Bull.* 24.4, pp. 35–43.

Soni, Devin (2018). *Introduction to k-Nearest-Neighbors – Towards Data Science*. URL: https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26.

TF/IDF. URL: http://www.tfidf.com/.

Thanda, Abhinav et al. (2016). "A Document Retrieval System for Math Queries." In: *NTCIR*.

Theory Behind Relevance Scoring - Elasticsearch. URL: https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html.

Tokenization. URL: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html.

tweepy. URL: http://www.tweepy.org/.

Unsupervised learning intuition. URL: https://whatis.techtarget.com/definition/unsupervised-learning.

Yamada, Ikuya, Hideaki Takeda, and Yoshiyasu Takefuji (2015). "Enhancing named entity recognition in twitter messages using entity linking". In: *Proceedings of the Workshop on Noisy User-generated Text*, pp. 136–140.

Zuva, Keneilwe and Tranos Zuva (2012). "Evaluation of information retrieval systems". In: *International Journal of Computer Science & Information Technology* 4.3, p. 35.