Seek

Topic Modelling and Semantic Interpretation on Unstructured Data

Madalina Sas (*ms6413*)
Dragos Dumitrache (*dd2713*)
Daniel Hernandez (*dh1213*)
Alvaro Sevilla (*as11913*)
Tudor Cosmiuc (*tnc13*)
Claudia Mihai (*cam212*)
Supervised by Marc Deisenroth

Imperial College London

January 2016

Executive Summary

Ever since the invention of the first complete writing systems, the inefficiency of reading large amounts of unstructured text has consistently decelerated progress whenever one needed to research or investigate. Today, when billions of bits construct humanity's fragmented knowledge base, the drawbacks of this problem are no longer negligible. When reading stops being a leisure and becomes a necessity, a helper indexing tool is often required, to accompany the readers in their research and store the relevant data in a structured format. This problem is not domain-related; it can be observed in various fields. Linguists, historians, lawyers, investigators, sociologists, medics, scientists and many other need to go through hundreds of pages of text in order to find the relevant information they are looking for.

Some solutions exist, such as search engines capable of processing (some) questions written in natural language, expert question-answering (QA) systems, and various libraries and frameworks for analysing text. However, during our analysis of these solutions in contrast to the problem described above, we identified the following issues:

- The lack of a broad and comprehensive resource of structured data that can be downloaded, improved, or analysed. Such a resource was Freebase[1], but since it was acquired by Google (and used to build the Knowledge Graph[1], which is not directly accessible by users), it is read-only and soon to shut down, while most of its data is still in the process of being made accessible on Wikidata[2]
- The impossibility of users to easily contribute to a structured knowledge base. For example, writing a Wikipedia article requires good writing skills and time, with the result of an unstructured resource that moreover requires time to read
- The existing tools for extracting statistical or structured data from very large amounts of textual data, such as the Python Natural Language Toolkit (NLTK)[3], require rather advanced programming knowledge, which does not come in handy, for example, for people trained in humanities or social sciences

Our purpose is to reunite the most recent research in Information Retrieval, Natural Language Processing and Semantic Computing with Machine Learning techniques, to construct a framework capable of extracting data from any kind of format, and to build either a private knowledge base powering an expert QA system on the user's personal computer, or allow contributions to an open knowledge base, easy to use and available to anyone looking for answers.

Seek provides a framework capable of processing textual data from the Internet, raw formats like PDF and DOC, and scans of books in various image formats like JPEG or PNG. Moreover, since it is intended as an assistant, it implements a simple question-answer system that is interactive and user-friendly. Furthermore, as previously described, we aim to construct an open, user-submitted knowledge base, by indexing existing sources like Wikipedia and allowing users to upload documents from sources previously non-existent on the Internet. We provide the main Seek interface as a web platform, and an open-source package allowing users to run Seek on their own computer for the purpose of analysing their own private data.

[1] `www.freebase.com`
[2] `www.wikidata.org`
[3] `www.nltk.org`

Contents

1 Introduction

1.1 Motivation

For as long as humans have dealt with information we have constantly devised new and more efficient ways to store, process and exchange knowledge. This has become increasingly trivial ever since computers became affordable, whether we are talking about big corporations or simple individuals. However, computers entail one major drawback in dealing with information targeted at humans: they require structured data in order to be able to process given information. Humans, on the other hand, rely heavily on communication via (relatively) unstructured mediums such as pictures, videos, sounds and, obviously, language. While the Internet has allowed for all these large compilations of texts to be available for literally everyone to study and build upon, it becomes increasingly difficult to read and absorb that much information in the traditional way. This becomes apparent when you take into consideration factors such as the large variety of writing styles, the unapproachability of academic vocabulary, in the case of peer-reviewed journals or works of non-fiction, or the intricate interdependencies between these texts.

Several new technologies have appeared lately that allow for reliable interpretation of natural language for the purposes of statistical analysis, query analysis, voice recognition, linguistic research, semantic computing, and many other important fields in AI and human-computer interaction. Seek attempts to bring together these technologies in the hopes of exposing a larger part of the population to the wonders of today's plethora of information or, at the very least, allow them to preoccupy themselves with tasks more important than sieving through texts in the hopes of finding the needed information.

1.2 Background and Research

The techniques used by Seek in order to tackle the problem of modelling collections of textual data (called corpora) and extracting information from such data reunite various research fields such as Artificial Intelligence, Statistical Machine Learning, Natural Language Processing, Information Retrieval, Topic Modelling, Semantic Computing and Concept Networks, and Question-Answering systems.

These techniques are introduced below, along with notes on how we used them and on the research done in order to improve them for our intended purposes.

1.2.1 Natural Language Processing

The story of Natural Language Processing starts with Alan Turing and the historical question, "Can a machine think?", he asked in the 1950 seminal paper "Computing Machinery and Intelligence"[3]. The concepts that lie behind the Turing Test have inspired countless researchers to inquire on the structure of natural language from a computational perspective.

The history of NLP visits the Georgetown experiment in 1954[5], which achieved the fully automatic translation of over sixty Russian sentences into English. Pioneers of NLP in the sixties were two remarkable systems for their time, SHRDLU[6] (1968), an expert system which lived in a 'blocks world' and could answer questions, in natural-language, about the state of the world, and ELIZA[7] (1966), a remarkable simulation of a person-centered psychotherapist and the ancestor of today's chatterbots. ELIZA used a small knowledge base to answer specific questions; when the user's questions or statements exceeded its knowledge, it would come up with a generic response, such as responding to "My sister is dating a fish."

with “Are other members of your family dating a fish?”. During the 1970s, other natural language programs attempting to simulate AI were written, including PARRY[8], a simulator of a paranoid-schizophrenic, created to be the counterpart of ELIZA, and Racter[9], a program that spits English literature.

Unfortunately, no other remarkable results showed for the following two decades, as the researchers were stuck in a loop imposed by the hard-coding philosophy they employed. However, in the late 1980s, the stagnating NLP community turned toward the new Holy Grail of Artificial Intelligence: Statistical Machine Learning. The increase in computational power allowed researchers to look at corpus linguistics, and employ a statistical approach over language processing. For example, Decision Tree classifiers started to replace rule-writing that powered systems such as SHRDLU.

Nowadays, NLP builds upon grammar, semantics, and statistics in order to perform various tasks. Research in NLP takes an empirical perspective, and multiple papers are available providing resources such as algorithms, classifiers, grammars, dictionaries, and specialised corpora. These were of core value to training and improving our own statistically trained classifiers for information extraction.

Part-of-speech tagging

In corpus linguistics, part-of-speech tagging (POS tagging), is the process of labelling a word with a particular part of speech, based on both its definition and its context in a phrase, sentence, or paragraph. Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. The stochastic algorithms, like Viterbi’s, are based on hidden Markov models.[4]

Chunking
Part-of-speech tagging is the main means of chunking sentences. In order for a sentence to be classified properly, all the words composing it have to be classified to their respective parts of speech.
Chunking is the action of splitting a text into sentences and classifying those sentences based on their types. This task is traditionally done by implementing context-free grammars: sets of recursive rules forming the basis for construction of sentences. For example, the context free grammar below, represented with the help of Regex notation:

```
S:    {NP}{VP}{NP}
NP:   {<DT>?<JJ.*>*<NN>*}
VP:   {<VBP>?<VBZ>?<VBD>?<RB>?<V.*>}
PREP: {<IN>}
PRON: {<PR.*>}
PP:   {<PREP>?<PRON>?<NP>}
OBJ:  {<IN><NP|PP>*}
```

attempts to split the sentences into chunks of the general form noun phrase → verb phrase → noun phrase. Each of these phrases is defined by other sequences of parts of spech. For example,

Name Entity Recognition
Name Entities are words and sets of words which can be safely identified as belonging to one of the following categories: PERSON, ORGANIZATION, LOCATION, DATE, NUMERAL. More categories can be manually defined if necessary in order to increase the expressiveness

of the Natural Language Processing package. The most important Named Entities and the ones most commonly used are the first four from the enumeration above.

Their main goal is the mapping of relations between entities, offering a description of the manner in which they interact with each other.

Name entity recognition is done either by means of a Hidden Markov Model (HMM) or that of a Conditional Random Field (CRF).

Having used Stanford's[**?**] library in our project, we will only focus on the CRF classifier, as it is the one used by the Stanford NLP package. CRFs are used extensively for structured prediction in the context of machine learning. While regular classifiers would predict a label for a single sample, without any regard for its neighbours, a CRF classifier can predict sequences of labels for sequences of input by employing a maximum likelihood learning strategy. On their part, they also generalise the constraints on input and transform the initial constant probability distribution into an arbitrary function, which is then applied to the original input sequence.

Focused Named Entities
Focused Named Entities[16] represent a subset of the regular named entities with a higher relevance to the topic of a document. As such, it stands to reason that their main goals are the summarisation of a text as well as the search of documents related to a certain topic.

Once all the named entities are computed, by using a coreference algorithm on the chunked sentences, we try to find entities describing the same object, and combine them together, deriving even more relevant information and thus enlarging the subset of possible focused named entities.

The traditional method for employing focused named entities is based around the use of weighted sentences, where the sentence frequencies are computed according to the LDA tokens extracted from the input. Having obtained those weights and knowing that focused named entities are generally of the types PERSON or ORGANIZATION, sentences containing such entities are added to the summary.

As a short example, in a text about Thomas Jefferson, Thomas and Jefferson will both get mapped to PERSON. The chunker will at some point include both entities in the same sentence, preceded by the title 'President', and by use of this coreference algorithm, 'President Thomas Jefferson' will also be mapped to a PERSON.

Although misclassifications can easily appear as a result of the coreference stage, they can be easily fixed by passing the resulting entities to a classifier employing some learning method.

1.2.2 Topic Modelling

The terminology for statistical corpora-based NLP techniques is described below:

- **word**: a discrete unit of data, $w \in \{1, ..., V\}$, represented using unit-basis vectors as following $w_v = 1$ and $w_u = 0$ for v $\neq$ u.
- **document**: a sequence of n words denoted by $\mathbf{d}$ = (w_1, w_2,... , w_n), where $\mathbf{w}_i$ is the i^{th} word in the sequence.
- **corpus**: a collection of N documents denoted by D = d_1,d_2,... ,d_M.

Term frequency–inverse document frequency
The researchers in the field of information retrieval made the first step toward solving the problem of modelling corpora by introducing a potent model, which allows for further research on its features. The main techniques proposed by IR researchers rely on reducing each document in the corpus to a vector of real numbers, each of which represents word counts obtained by various measurements. One of the most popular is the `tf-idf` scheme (Term frequency–inverse document frequency)[10]. `tf-idf` is intended to reflect how relevant a word is for a document.

In this technique, a vocabulary (or dictionary) of reference is chosen, and, for each document in the corpus, the frequency count of each word in the vocabulary is stored in a vector. This vector is then normalised and multiplied with an inverse frequency count over the whole corpora, which usually gives a logarithmic measure of the number of occurrences of a specific word in the entire corpus.

$$\forall D \in \mathbb{D}.\, \forall t \in d.\, \forall d \in \mathbb{D}.\, \texttt{tf\_idf}(t, d, D) = \texttt{tf}(t, d) \times \texttt{idf}(t, D) \tag{1}$$

The result of this product is stored in a matrix for all words and all the documents in the corpus. These columns store the frequency information obtained, which can be analysed in order to further extract information necessary for tasks such as text-mining, sentiment analysis, etc. This matrix is obviously large and usually very sparse.

The advantages of `tf-idf` are that this technique allows for the analysis of similarities or differences between documents with respect to the vocabulary of interest. On the other hand, it conveys no relevant knowledge about the inner statistical structure of each document or the relationships between words, as it is just a frequency-based statistical measure.

For an illustrative example, take a corpus formed of the following five documents:

```
'Barack Obama was born in Hawaii.',
'Hawaii has active volcanoes. Volcanoes are hot.',
'Volcanoes are very hot.',
'I am very active when I drink hot coffee.',
'Hawaii is a very hot island. Sahara is a very hot desert.'
```

The vocabulary of the corpus is thus:

```
['.', 'a', 'active', 'am', 'are', 'barack', 'born', 'coffee', 'desert',
'drink', 'has', 'hawaii', 'hot', 'i', 'in', 'is', 'obama', 'sahara', 'very',
'volcanoes', 'volcanoes', 'was', 'when']
```

We notice that punctuation and words like 'a', 'am', 'has' are not relevant for our analysis due to their lack of out-of-context meaning. These words are stop words and function words. *Function words* are words that have ambiguous meaning, but instead serve to express grammatical relationships between words within a sentence, or to link sentences together. *Stop words* are by definition generally extremely frequent in an English language text. For example, in our corpus, the word 'a' is as frequent as the word 'Hawaii'. Thus, most models are constructed after removing such words.

There are multiple ways of calculating the word frequency and the inverse document frequency, ranging from binary to logarithmic and log-normalised measurements. After removing stop words, calculating the raw frequency of each word and then normalising the vector for each document, we get the following matrix:

Document/Term	d1	d2	d3	d4	d5
'active'	0	0.32	0	0.5	0
'barack'	0.5	0	0	0	0
'born'	0.5	0	0	0	0
'coffee'	0	0	0	0.5	0
'desert'	0	0	0	0	0.14
'drink'	0	0	0	0.5	0
'hawaii'	0.5	0.32	0	0	0.14
'hot'	0	0.64	0.7	0.5	0.28
'obama'	0.5	0	0	0	0
'sahara'	0	0	0	0	0.14
'volcanoes',	0	0.64	0.7	0	0

The inverse-document frequency measure chosen is the standard measure, calculated by:

$$idf(t, D) = \log \frac{N}{n_t} \tag{2}$$

where $N = |D|$ and $n_t = |d \in D | t \in d|$

And illustrated by the matrix below:

Document/Term	d1	d2	d3	d4	d5
'active'	0	15.63	0	10	0
'barack'	10	0	0	0	0
'born'	10	0	0	0	0
'coffee'	0	0	0	10	0
'desert'	0	0	0	0	35.71
'drink'	0	0	0	10	0
'hawaii'	10	15.63	0	0	35.71
'hot'	0	7.81	7.14	10	17.86
'obama'	10	0	0	0	0
'sahara'	0	0	0	0	35.71
'volcanoes',	0	7.81	7.14	0	0

II. Latent Semantic Indexing
The `LSI` (Latent Semantic Indexing, also called Latent Semantic Analysis) model has been introduced by Bellcore[11] to address the disadvantages of `tf-idf`. `LSI` uses singular value decomposition (SVD) on the `tf-idf` space. By doing so, the LSI model can extract the conceptual content of a given body of text by finding a numerical weighting that connects the terms in that document to how they are used in similar contexts. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings.

The technique is thus able to correlate semantically related terms that are latent in a corpus. LSI is employed by many search engines to index documents as it can be used to extract the meaning of the text in response to user queries (called concept searches). Concept searches against corpora indexed by LSI will return conceptually similar results with respect to the search criteria, even if the resulting body of text does not share any specific word with the search query.

Furthermore, it has been argued[12] that linear combinations of the original `tf-idf` features can capture some aspects of basic linguistic notions such as polysemy or synonymy. The accuracy of these results improves with the size of the corpora, which provides an incremental model that can be trained for more insightful information extraction. The technique is heavily used by search engines as well as for information discovery, document classification, text summarisation, essay scoring, spam filtering, etc.

III. Latent Dirichlet Allocation
`LDA` (Latent Dirichlet Allocation)[13] is a graphic model introduced in order to compensate for some disadvantages of `LSI`. Where `LSI` retrieves the distribution for a single topic, `LDA` will retrieve multiple possible topics and their distributions. In LDA, each document is seen as a mixture of topics, for which the topic distribution is assumed to take the form of a Dirichlet distribution.

For example, an LDA model might have topics that can be classified as `geography-related`. A topic has probabilities of generating various words, such as `river`, `mountain`, `island`, `canyon` for the geography topic. Obviously, the word that names the category, in our case `geography`, will have a high probability to be generated for this topic. Words without special relevance, such as the stop words or function words, will have roughly even probability between classes, and, as a result, are usually removed.

Each document is assumed to be characterised by a particular set of topics. This is akin to the standard bag-of-words model assumption, and makes the individual words exchangeable.

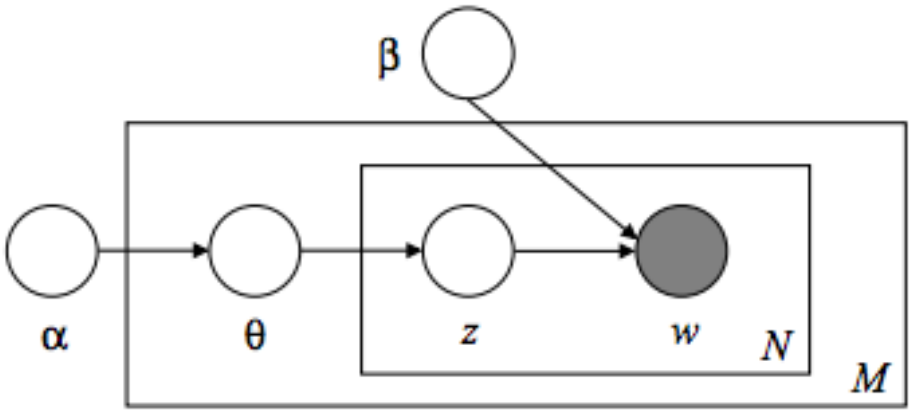

Figure 1: Graphical model representation of LDA

Figure 1 describes LDA in a graphic way. The boxes are called "plates". The outer plates represent documents, while the inner plates represent the repeated choice of topics and words per document.

`M` describes the number of documents, `N` the number of words in a document, while the remaining variables describe the following things:

- α is the parameter of the Dirichlet prior on the per-document topic distributions
- β is the parameter of the Dirichlet prior on the per-topic word distribution
- θ_i is the topic distribution for document i
- φ_k is the word distribution for topic k
- z_{ij} is the topic for the jth word in document i
- w_{ij} is the specific word.

1.2.3 Semantic Computing

Semantic computing refers to a combination of semantic analysis, natural language processing, and data mining techniques in order to achieve semantic capabilities for both user input (e.g., queries) and computational content (e.g., documents to analyse).

Many attempts to write semantic networks have powered multiple chatterbots. Examples of such semantic collections are MARGIE (Schank, 1975), SAM (Cullingford, 1978), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), Politics (Carbonell, 1979), and Plot Units (Lehnert 1981).

1.2.4 Concept networks

Concept networks[14] are layered semantic networks aiming to solve keyword similarity and thesaurus. A concept network combines a keyword concept net for extracting the keywords

of information and a concept network that merges this this results with a field concept network

1.3 Objectives and Achievements

Seek's main objective is to become both a personal assistant and a versatile search engine. It will be fed with unstructured information in both text and image form which it will then analyse. At the highest level of abstraction, it could be stated that Seek 'understands' the information that it is given. The extracted information can be queried; because we wanted to have a product that would feel as natural as possible, we intended for Seek to be able to understand natural language[4]. Therefore, questions may be presented in natural language as opposed to strict database calls. Any interaction with the user should feel smooth and simple, leaving all the computation regarding categorisation and database building to be processes behind the scenes.

With the initial version of Seek, the user will be able to upload files in order to increase Seek's knowledge, ask about any topic, and obtain an answer. We conceive Seek as a personal assistant as you are able to upload your own local data. An extra feature aimed at improving both the user experience and Seek's capabilities is a hooded version of supervised learning. The user will be able to rank, using natural language, whether they deem the answer satisfactory. This will be helpful in both keeping our knowledge base up to date and correcting any fault made by Seek's analysis algorithms.

So far, as it will further be discussed in the report, we are able to fully tag the content of a wide variety of file formats with their respective topic labels. These labels are then used as information to be stored in a database for later query. As previously mentioned, questions can be stated in natural language and a legible answer will be returned. The knowledge base is global and user agnostic, and a user's upload will affect their own experience and that of others.

Seek represents an advanced, powerful and easily improvable platform for linguistic semantic analysis and information retrieval, employing the tremendous capabilities of the technologies behind it together in a seamless fashion. As a result, it provides a package of features such as:

- robust conversion to raw text from multiple formats
- word-wise and sentence-wise tokeniser
- statistical analysis on word distrbution in sentences and documents
- named entities detection
- file uploading to increase the knowledge base
- persistent database containing previously stored information
- user-feedback interaction

[4] As it will be noted, there is a rather substantial amount of work on natural language processing throughout the entire project.

2 Product Design

2.1 Product Description

The scope of Seek is meant to be broad and offer full opportunities for customisability and scalability. The results of our design research showed that in order for Seek to be as accessible as possible, we need to provide a seamless interface for the casual user, but also one that allows full control for the proficient user and for the developer.

Thus, the final product is composed of two different packages: a Web app developed with the general user in mind, and a Python package providing the resources necessary for any developer to run their own version of the Seek core Python API on their personal computers.

2.1.1 User-centred design

Seek will run in the cloud, allowing the user to upload data in multiple formats (see TODO list of features) in order to have Seek search and index it, to query Seek, and to train Seek. Both come with a very friendly, futuristic user interface (see interface). This is designed for a speech-based interaction system between the user and Seek, but also for the use of the keyboard and mouse to actually collect, visualise and query the data should you be in a more professional environment.

2.1.2 Developer-oriented design

Seek will be available as a Python package to download and install on Linux and Mac OS. Developers are encouraged to build modules that would allow Seek to spread its skills. We are envisioning an open-source knowledge platform for Seek's developers and teachers.

While the Web interface provides a user-friendly way of interacting with Seek, the developer-oriented design comes under the form of a Python environment. Installation and activation scripts are provided, meant to facilitate interaction with Seek out of the box. All the dependencies are installed automatically, and shell variables are set and unset whenever the Seek Environment is activated or deactivated. The main benefit behind this is constituted by the fact that a user's machine will never be exposed to these changes, thus keeping it safe in the unlikely event of a dependency failure.

The package itself contains Seek's core, allowing the developer to use features for text extraction, natural language processing, and topic modelling. Furthermore, it also allows for training of specific classifiers for these tasks. Additionally, a scraper is included in the package, as a helper tool for grabbing large amounts of data from the Internet.

2.1.3 Human-Centered Design research

Following the initial research stage, we then approached the project as a product, using the Human-Centered Design principles we have learned and applied in the past. For this purpose, we utilised personas, a method we found to be both straightforward and extremely powerful in keeping the design of a product on the correct track. Here we present the personas we have kept in mind with every design decision along the way.

First of all, we have Tom, who is 18 years old and a university fresher in Computing. During his first year, he has to complete an individual learning project on a topic of his

choice. Being a dedicated student, Tom has already looked at several candidate fields which he finds interesting, but he is still at a loss on what to choose. Given an immense amount of specialised literature on these fields as well as very little time to conduct further research for his choice, the greatest help he could potentially get at this point is a quick overview of this literature. This would allow him to quickly grasp a better idea on his options and make his choice in order to fully delve into his chosen topic afterwards.

Our second persona, Alyssa, is a 30-year old linguist. Alyssa is researching Shakespeare's work, analysing the language structures and vocabulary in order to infer the use of the English language in the sixteenth century. However, Shakespeare has 37 plays and 154 sonnets attributed, which makes for a very inconvenient task. Alyssa could use a tool that would be able to analyse all of this literature, and provide information on the use of language structures across the different works, or find facts such as "Shakespeare's most repeated word across his work being "king"".

2.1.4 Similar or related software

IBM Watson
As a product of IBM's R&D divison, Watson is a technology platform employing natural language processing and machine learning techniques for the analysis of unstructured data. It is able to understand and answer very complex questions. Therefore, it becomes quickly apparent how it is similar to our very own Seek. Moreover, having started its development back in 2005, it may first appear as if our quest has ended before it even began.

However, although an employed team of experts working on this product is an important advantage for IBM, we also consider it to be a significant disadvantage compared to our development of Seek. It is precisely because we allow general public contributions to our project that Seek will learn the needs of human users much more accurately. Moreover, releasing it as a web app, as well as an open-source package will allow for vast and quick improvement of Seek; the contributions would thus not be judged on the basis of hiring criteria and company hierarchies, but rather on the utility and effects of the ideas themselves, and will be open to everyone wishing to aid Seek's development.

Google Now
Google Now is an intelligent personal assistant delivered within Google Search and Google Chrome browser applications. With a Natural Language User Interface, not only does it answer user queries and follow their commands, but it also attempts to be one step ahead of the game by delivering additional information that it predicts the users may want based on their history.

While Seek has drawn inspiration from the personal assistant capabilities of already existing software such as Google Now, this is where the similarities end. We do not claim Seek to track users' behaviour outside of the application, and we do not wish for it to predict beyond users' current sessions. The privacy of users is safe with Seek, and Seek does not rely on personal information, only on voluntarily shared data.

Siri
Like Google Now (see 2.1.4), Siri is also an intelligent personal assistant based on the same general principles, available as a feature of the Apple products and services. The same similarities and differences as in the paragraph above apply, with the addition that unlike Siri, the Seek web app is platform-independent.

Alice and Eliza
Alice (Artificial Linguistic Internet Computer Entity) and Eliza are essentially chatter bots. While Eliza is extremely primitive in its natural language processing abilities, Alice is one of the strongest programs of its type. The most important similarity between programs of this type and Seek is their capability to process natural language. Moreover, they are also similar in their choice of drawing from a pool of basic trivial questions and their respective answers (e.g., "How are you?", "How's the weather?").

However, it is obvious that both Alice and Eliza have important limitations. For example, as one of the oldest chatter bots, Eliza's limitations are firstly acknowledged by its creator, having only been further confirmed since its creation; these limitations are essentially derived from the fact that the most serious Eliza scripts are the ones following certain psychotherapy techniques[7]. Therefore, the user can assume no knowledge from the other party's side, which severely decreases the program's ability to appropriately reply to queries. While Alice is a significantly improved chatter bot in comparison with Eliza, the opinions on its capabilities are mixed, as ultimately stated by its creator in the conclusion of its technical presentation[17].

It is precisely because of Seek's advanced learning capabilities that it is vastly superior to chatter bots, in general. It is able to learn from large amounts of unstructured data as well as from user feedback, which is meant to consistently increase its performance during its interactions with users.

NLTK and Stanford NLP
As Seek's core draws from both NLTK and Stanford NLP, it is impossible to offer an objective comparison. On the other hand, the web interface of Seek makes these toolkits accessible to users who may not be comfortable with programming.

2.1.5 User interaction

User interaction has been a leading principle all throughout Seek's design and development. Intended as a product, it was essential that the potential use of Seek was as straightforward and seamless as possible.

The core results of our extensive research on various theoretical fields' current states (see 1.2) are captured by the back-end. In contrast to the complexity of Seek's abilities, the user interface is exceedingly simple (see 3.6).

There is a single textbox in the interface, where the user poses questions to Seek.

3 Implementation

This chapter reviews the architecture of the Seek core, and how it implements the research presented in Section 1.2. It further presents the Web Platform and the Developer Package as well as how they implement the design in Section 2.1. Finally, we review the challenges and risks undertaken in the process of development along with a discussion of our respective decisions.

3.1 Overview of architecture

3.1.1 Web Platform

The web platform is a Django app that takes input from the user:

- ▷ file uploads
- ▷ commands, through text input or buttons
- ▷ feedback on results, through text input or a slider

The user command triggers the backend of the Django app which sends asynchronous calls to the core of Seek, namely to the Executor. This module then runs on the server, awaiting to process commands from the user. The Executor forwards the commands depending on the user input.

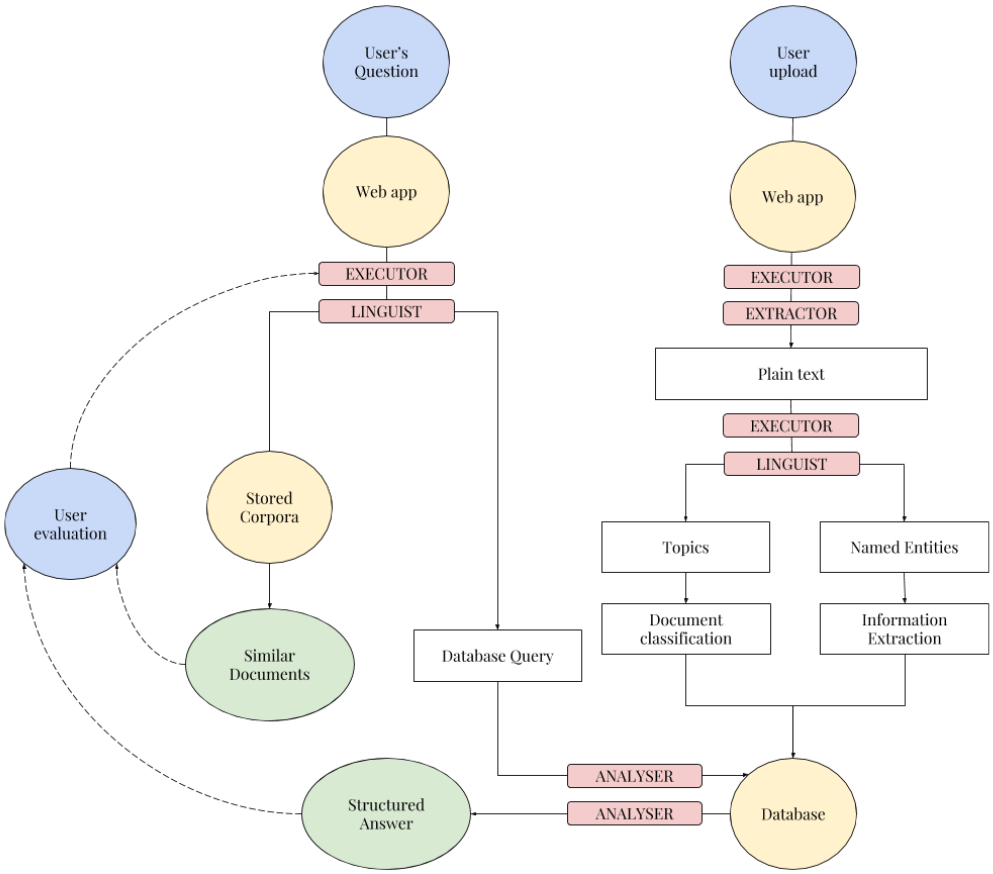

Figure 2: The flowchart diagram of the web platform.

If the user has uploaded some files, the Executor will upload them to the server and forward them to the Extractor, which converts the document(s) uploaded, and then caches them. During the session, it will process and index the document(s) in the background, save the document to the corpora, and, based on user confirmation, it will also save the relevant information to the semantic database.

If the user simply asks a question, the Executor calls the Linguist, which processes the question, and forwards it to the database by default. If the user has requested a restrictive search, the Linguist forwards the query solely to the documents cached in the session.

3.1.2 Developer package

The developer package contains all the modules that handle the extraction, machine learning, analysis, and database interaction required to run Seek on a local machine, i.e., the Core.

The package comes with an install script that prepares all the dependencies and constructs the Python environment in which Seek will run, then downloads the core files. Each of the packages in the core provide either their own command line interfaces, or initialisation scripts and APIs. Therefore, the user can utilise any module independently, modify the modules, or build something else entirely on top of Seek.

The developer is also given full control of the resources in the package, thus an initialising script is provided that allows the user to train the classifiers needed for various Machine Learning tasks. Seek-Dev can also be used out-of-the box with good results on statistical Natural Language Processing, but with significantly less proficient results in the Topic Modelling area.

3.2 Technologies used

Python was the programming language that was most used across our backend. Python was chosen not only due to its popularity and strong community support, but also due to the availability of numerous open source software for information extraction, natural language processing, and topic modelling. Moreover, the choice of writing the core of our program in Python allowed us to easily construct a web app in Django, which is also powered by Python.

Ruby was the preferred language for implementing the scraper. This decision was influenced by the fact that part of the group was already familiarised with crawling and scraping the web using Ruby, having done so previously. Although the same utilities used in Ruby were available as Python packages, there was no justification for making the switch, and for the time that would have been spent adjusting ourselves to the Python API at that point in time.

NLTK (The Natural Language Tool Kit) is one of the most popular resources for natural language processing. NLTK provides a comprehensive API for tasks such as word and sentence tokenization, word frequency analysis, part-of-speech tagging, chunking, collocation detection and analysis, semantic interpretation, information extraction, and other aspects of natural language analysis. Moreover, NLTK provides a large volume of resources including:

- ▷ WordNet, a semantically-oriented dictionary of the English language
- ▷ the Treebank and the Brown corpora, which contain processed text from multiple domains

- ▷ the Question Classifier corpus, which provides a set of questions labelled to their semantic meaning
- ▷ the Names corpus and the Countries Prolog Database, which helped us train an improved Named Entity recogniser

Furthermore, NLTK provides interfaces for statistically training classifiers (Decision Tree, Bayes, Probabilistic classifiers, etc.), which have proved useful for either generalising or specialising our information extraction features, according to our needs.

Stanford NLP Package (The Stanford Natural Language Processing Package) is another powerful tool for natural language processing. It is a Java Library for which limited NLTK support exists, allowing it to be programmatically used from Python. Despite the fact that all the features offered by Stanford's NLP package are virtually offered by NLTK as well, the Java library shines in at least one aspect: Named Entities, which we cover in more detail later on.

Textract is a Python package that congregates different types of text extractors. The biggest benefit in textract is the uniform API. Textract allowed us to extract information from: .doc, .docx, .ppt, .pptx, .jpg, .pdf, .xls, .csv, .pub, .tiff, .ps, .png, .gif[5].

Gensim is an invaluable Python library for the natural language processing and information retrieval communities. It provides efficient implementations of well-known algorithms for semantic analysis tasks such as topic modelling, document indexing, and similarity retrieval. Out of all of its features, we mainly utilised its implementations of Latent Semantic Analysis and Latent Dirichlet Allocation. In addition to the efficiency and scalability of the Gensim library, an important aspect was the existence of multiple tutorials highlighting its features, which smoothened the integration of the library's features into our project.

Neo4J is a database framework for handling large graph-based databases. It allows for fast, simple, and efficient querying through its proprietary query language: Cypher. This graph-based approach to storing and referencing data is crucial for the kind of information Seek attempts to process as it allows us to build a semantic net directly. This net is built based on previous data analysis and feedback, by carefully manipulating the topology of the graph. The GraphDB thus produced will contain nodes for the various types of subjects and objects as well as directed or undirected edges between these nodes for storing relationships between said agents and objects. It is worth noting that both nodes and edges can have labels and fields attributed to them, and we use these to model the attributes of specific entities or the manner of interaction between them.

Linking between the database and the core of the program has been achieved by utilising the 3rd party Python module **neomodel**. This module allows us to perform various actions on the database from within a Python script, but, most importantly, to interpret nodes and relationships in a straight-forward manner as Python objects. On top of that, it provides a clean framework for generating arbitrarily complex Cypher queries for retrieving data from the database.

Cypher is the proprietary query language of Neo4J. It is a simple yet powerful declarative language that is meant to provide a simple, clean, and readable format for queries. It is extremely easy to use and is largely responsible for the highly scalable qualities of this specific database framework. A Cypher query makes strong use of pattern matching in order to retrieve results in a fast and reliable manner. We have thus designed our result identification algorithm to capitalise on this extremely convenient feature.

[5]Currently text recognition works in image formats only for simple, clean data.

Django is a web aplication framework written in Python. It is purposed for building robust websites in a quick way, and it follows the Model-View-Controller design paradigm. Being written in Python, it integrated well with all the other parts of our code, as we could just import our modules in Django in order to have them process user requests. While we definitely did not need all the capabilities Django offers, it provided us with most of the web infrastucture we needed to deploy the application.

Front-end technologies utilised within the development of the product mainly consist of the traditional HTML-CSS-Javascript combination. For efficiency purposes, we combined fragments of several CSS frameworks, i.e., Normalize, Skeleton, and Bootstrap. In doing so, we have yielded better and faster results than we would have by attempting to re-invent the wheel. It is thus noteworthy that the front-end of our web app is extremely light, both in the absence of unnecessary code and its evidently clear and straightforward functionality.

3.3 Core

The Seek Core is the 'brain' of our project. It gets called by our Django backend, but can also be called by command-line. It is a Python module consisting of the following:

- ▷ the Executor, a script which handles files and commands, and forwards them to the other modules, allowing for parallel and asynchronous operations on large amounts of data
- ▷ the Extractor, which extracts plain text from various raw formats
- ▷ the Scraper, a script that scrapes web pages for textual content and downloadable documents
- ▷ the Linguist, a script which performs various information extraction techniques
- ▷ the Statistician, a module containing various statistically trained classifiers for the methods used by the Linguist
- ▷ the Analyser, which inserts entries into the database and queries it for results
- ▷ the Initialiser, a script that performs the training for all the classifiers in the Statistician and outputs them as object files (pickles) to be used later on

3.3.1 Executor

The Executor provides a command line interface that allows batching commands, e.g., extracting plain text from multiple files or directories, extracting data from the text, inserting into the database, and indexing large corpora with topic models. It uses the `argparse` Python library to grab a comprehensive command and forward it to the other modules.

Concurrency control
In order to optimise the speed of task performance, the Executor uses the `concurrent.futures` Python library to spawn a balanced number of threads. Each of these threads will call one of the other scripts to perform operations on the data.

The information we use and process is layered in such a way to allow maximum fine-grained concurrency. In the context of very large corpora, file-locking can be used without large overhead compared to the size and complexity of the entire task. Each module takes a file

from a 'pool' and outputs another in another 'pool'. The files are write-locked, but multiple processes can read from the same file.

Data flow
To exemplify how the Executor shapes our data flow, we take the concrete situation of a document, say in `.pdf` format, which is uploaded to Seek. The user expects that all information extracted from that document to be inserted to the database. This describes the full life cycle of a document in our system:

1. The Executor uploads the file to the `raw` file pool
2. The Extractor runs on the `raw` file pool, where uploaded files are collected and queued for processing, and outputs plain text files in the `txt` file pool, from where the Linguist takes over
3. The Linguist runs in the `txt` file pool, which either outputs to the user interface (either the web app or the command line), or calls the Analyser.
4. The Analyser grabs the commands from the Linguist and writes in the database.

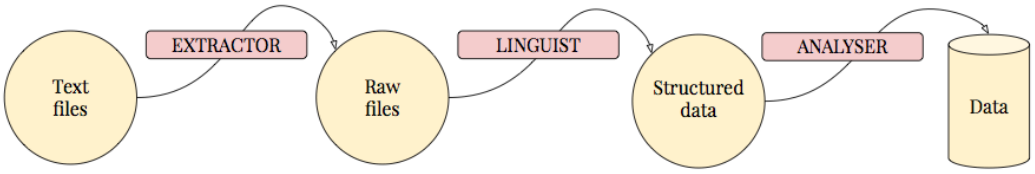

Figure 3: The data flow diagram for a full life cycle of the information in a document.

3.3.2 Extractor

The Extractor is the initial unit in Seek's processing chain that converts any given file to `.txt` format. There is no loss of information on the process, and, for most of the file types, the extraction is 100% accurate. Having all information stores as text files greatly simplifies our challenge, yet compression should be considered in the future for scalability.

The Extractor simply reads the file, calls the conversion function `textract.process` in the `Textract` package, which then handles the filetype and returns the plain text to be then written to a `.txt` file.

3.3.3 Scraper

In order to train Seek on natural language processing, we needed large amounts of information. It had to be unambiguous and well written, as it was going to be the foundation on which later text would be analysed. We turned our heads to Spiral[6], Imperial College's open access repository for journal articles and varied research output. Spiral's database contains over 10.000 articles divided into four faculties, engineering, medicine, natural sciences and business school. Because of its size, we could not hope to manually download every single file. So we built scraper, which retrieves every file from Spiral for training purposes.

During the duration of the project, the Scraper was refactored several times following a

[6] `https://spiral.imperial.ac.uk/`

more object-oriented approach. The initial scraper now only receives flags and arguments, calling the respective methods from the attached module.

As development progressed, more and more data was necessary. Thus, the Scraper was improved to perform various other specialised crawling skills. Among the capabilities it now possesses are the scraping of HTML pages, accessed either online or from local storage, extracting <p> tags, as well as the scraping of whole Wikipedia categories.

Efficient connections
When multiple files are to be downloaded from the same server, the scraper establishes a persistent TCP connection and pipelines requests on 5 different threads, thus reducing the download time considerably. The decision of only using 5 threads was made after testing the scraper in different scenarios and with different values. As a result, we found that too many parallel requests would cause the server to lose some of the files which in turn led to results varying from simply skipping files to dirty writes to disk or attempting to read from ghost files.

The Scraper is not a part of the core architecture, but will be published with the open source package to allow people to easily gather data from various on-line resources.

3.3.4 Linguist

The Linguist performs the Information Retrieval tasks on bodies or corpora of text. It's main tasks are:

1. (Focused) Name Entity Recognition
2. Topic Modelling
3. Text summarisation
4. Information extraction into semantic relationships
5. Question classification

As the Linguist is a script that gets called concurrently, it relies on an attached module, the Statistician, which contains the classes that perform the natural language processing tasks.

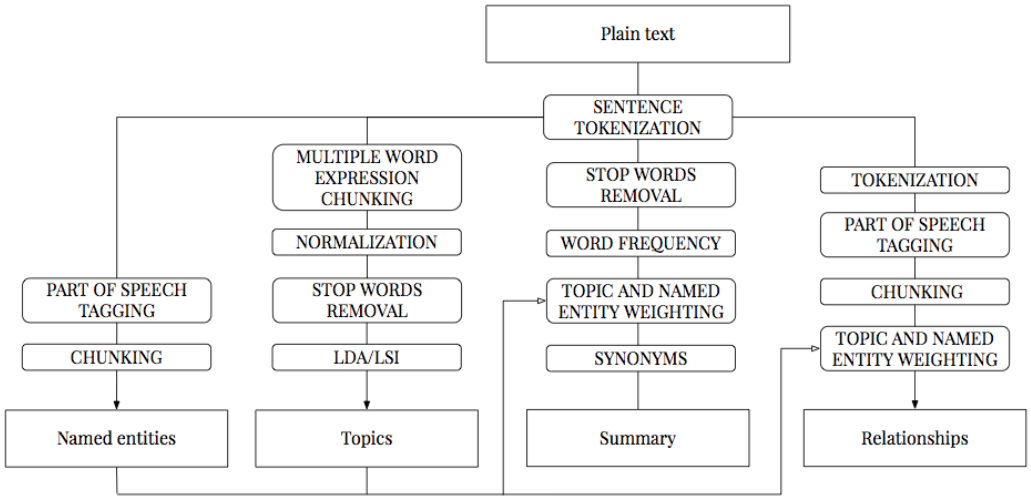

Figure 4: Flowchart of techniques performed by the Linguist over documents

Given a document or a set of documents as an input, the body of text is then either tokenized into a list of words, or in a list of list of words, each representing a sentence with the punctuation filtered out. Even though NLTK provides a sentence tokenizer by default, we preferred to train our own Sentence Classifier (see 3.3.5).

The words are chunked together into collocations or multi-word expressions (MWEs) where necessary. As specified in 1.2, our multi-word expression detector is based on the research performed by the Stanford NLP group, and uses a merge of the Sharoff[18], the Baldwin[19], and the McCarthy[20] dictionaries of commonly used multi-word expressions.

Topics
In order to improve results of the topic models, the list of words is filtered to remove irrelevant parts of speech (such as determiners, articles, modals, existentials, and pronouns), and also the *stop words* (words which occur with a very high frequency in an English language text without any nominal meaning or concept to convey). The purpose of such filtering is to allow for the classifiers to only index words which convey meaningful information on their own. As these topics rely heavily on joint distributions and frequencies, it is very important not to imbalance these measurements with words which do not introduce semantic or conceptual information.

Named entities
Named entities extraction requires more natural language analysis than Topic Modelling. In order to extract name entities efficiently and improve the semantic information from the results, we are first chunking the sentences into Noun Phrases (NP). In order to do so, a part of speech tagger and a chunker have been trained, and only after each sentence is chunked the Name Entity detector is used. For their training and implementation details, see 3.3.5

Summarising
Text summarising is done by calculating a weight for each sentence. This weight is equal to an average of the word frequencies of each relevant word in the sentence (i.e. the words left after filtering out the stop words). This weight is then augmented with a count of *focused named entities* that appear in the sentence (as seen in 1.2), or with the weights of the topic words which appear in that sentence. The sentences are then sorted by weight and a number of sentences (generally inputted by the user) is returned as the summary.

For example, the text 'Science' found in the appendix has been summarised by Seek in 10 sentences as following:

> Since classical antiquity, science as a type of knowledge was closely linked to philosophy. Not until the 17th century did natural philosophy emerge as a separate branch of philosophy, which is today called natural science. The dominant sense in ordinary use has a narrower use for the term science. In this period it became more common to refer to natural philosophy as natural science. Similarly, several other major areas of disciplined study and knowledge exist today under the general rubric of science, such as formal science and applied science.

> Since classical antiquity, science as a type of knowledge was closely linked to philosophy. Not until the 17th century did natural philosophy emerge as a separate branch of philosophy, which is today called natural science. Science continued to denote reliable knowledge about a topic; it remains in use in modern terms such as library science or political science. In this period it became more common to refer to natural philosophy as natural science. Several other major areas of disciplined study and knowledge exist today under the general rubric of science, such as formal science and applied science.

The first text has been summarised by augmenting the frequent sentences with the weight of the LDA topics, the second with the weight of the LSI topics, and the third by duplicating the weight of the most frequent ten Named Entities.

Various rephrasing techniques can then be applied to the final text. (see 3.3.5)

Information extraction
For information extraction, all the methods from the above are applied. Each sentence is chunked into Noun Phrases (NP) and Verb Phrases (VP). By using this structure we aggregate the possible attributes about an entity together with the entity itself in a NP, and also, thanks to the VP, we can attempt to establish relationships between the two NPs linked by it. This analysis is done usually when the sentence contains a frequent topic or a named entity.

There is no universal solution to this problem. Sometimes, a very naive extractor yields very accurate results, while a statistically trained classifier fails.

Question classification
The question classification task is crucial for user communication. It uses a mixture of naive and stochastic methods to extract as much information as possible from a question (a sentence finishing in '?'). The naive methods consist of simply identifying a list of keywords together with any named entity, and assumes some basic structure to the question. The other uses a classifier trained with decision trees that looks at a much larger list of features, such as word ordering and the type of named entities present, to decide what kind of answer does the user expect.

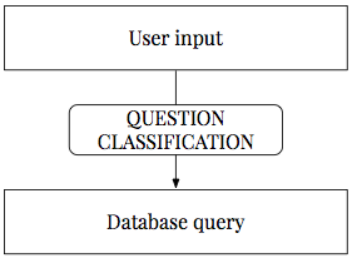

Figure 5: Techniques performed over questions

Based on this information, a query will be constructed for further interaction with the database.

3.3.5 Statistician

Sentence tokenization
The `SentenceTokenizer` is a NLTK classifier that merges lists of tokens into sentences by collecting punctuation features. The classifier is built on the `nltk.DecisionTreeClassifier`, as it turned to be the most efficient(See table 1)). These features, help separate punctuation that, for example, delimits an acronym, from punctuation that truly delimits sentences. The classifier has been trained on the sentences of the Treebank corpus.

POS Tagging
After verifying the accuracy of Part-of-Speech taggers recommended by NLTK, we decided to chose the recommended one, which offers the highest classification rate: `nltk.pos_tag`. This tagger has been extensively trained by experienced linguists on the Treebank corpus, known as a very general and balance corpus. Due to the highly modular implementation, it is simple to replace with a different tagger, given the case that we might encounter texts of very different structures.

Chunking
NLTK uses by default a chunker trained on the ACE corpus, containing Arabic, English and Chinese sentences and words. However, since the data we are currently working on is only in English, we decided to train our own chunker on the Brown corpus with a `nltk.NaiveBayesClassifier`.

As the Brown corpus is very broad, it generated a more accurate tagging of sentences than others. Using the same example as in section 1.2, the classifier trained on the default corpus would have split 'President Thomas Jefferson...' into a sentence with two branches, one containing 'President' and the other one 'Thomas Jefferson...'. However, the Brown chunker only builds one branch containing 'President Thomas Jefferson...', which gives more relevant information.

Named Entity Recognition
As previously mentioned, both Stanford's[**?**] NLP Library and NLTK provide out-of-the-box Named Entity Classifiers. However, they have been trained on different sets of data and as such yield different results on our own models.

In order to overcome this issue, we have trained our own classifier and combined it with one of the classifiers provided by Stanford, all the while comparing performance measures to make sure we are on the right track.

Tables 2 and 3 describe two of the default classifiers, while Table 4 represents our own classifier. Despite giving a slightly smaller performance value, our classifier is capable of correctly identifying dates, making it more expressive than the default ones.

Simple Rephraser
The `SimpleRephraser` contains some naive classifiers and techniques for rephrasing text. Even if these techniques showed promising results, they are far from perfect. Rephrasing is yet an unsolvable problem, but we believe our techniques prove a strong starting point and help make the text more readable. We have also attempted techniques such as passivisation, but unfortunately the grammatical classifiers required by such technique also require extreme processing power to train.

- ▷ **Synonyms**: Word repetition is to be avoided and the usage of synonyms is always recommended for good writing. Therefore, we make use of the Wordnet dictionary to retrieve synonyms and use them where necessary, such as when a word is repeated

in three consecutive sentences. We use the topics extracted for the original text to deduce what context is relevant and which synonyms to use from the synset of a given word or entity.

- ▷ **Pronouns**: In order to make the text seem more natural, a classification on given names is performed, in order to find the gender of an agent and use pronouns like 'she','he','it'.
- ▷ **Adverbs**: Adverbs are usually to be avoided in a short text. Especially since the summary is constructed by picking sentences in the original text, there is a high possibility for an adverb that refers to a sentence which has not been selected to be in the final text. Therefore, a general approach is to remove adverbs, especially the ones which appear at the beginning of a sentence. A classifier has been trained with a hand-built corpus to choose which adverbs should be removed.

3.3.6 Analyser

As a conclusion to the whole abstraction process Seek performs, the Analyser handles the data provided by the previous layers of text processing and is responsible for both constructing an appropriate database as well as interpreting the user's request in order to query the information we have processed so far. It is designed to look at the GraphDB it builds and traverse it appropriately depending on the user's concern. This is accomplished by taking into account information provided by the linguist and the learner in order to most efficiently make use of the identified relations between nodes. Results are returned to the user, who is prompted to rate the accuracy and efficiency of the results, in order to tap into the reinforcement learning capabilities of the product. The database is updated according to user feedback so that it eliminates noise.

Typical Questions
In order to streamline the process of interpreting the kind of answer a question expects, we have taken the liberty to support the interpretation of only a few select typical question words, and questions posed to the system are thus required to contain one of these words in a syntactically sound manner. These question words are, in no particular order, the classic *Who, Where, When* and *What*, maped to certain types of target answers, respectively: *Entity, Location, Time, Object/Attribute/Relation*. For the purposes of this prototype, we considered this sufficient to showcase the semantic capabilities of Seek. The same way that a word would be tagged with a topic whilst analysing a document, questions are also processed. The information gathered from a question will depend on its phrasing. For instance in: 'Obama is the president of the US', we see that Obama is the subject of the phrase and that 'of the US' compliments the subject noun, and as such will be taken as an attributes of the former.

Statistical Breadth-First
The Analyser relies on the Linguist to identify the subject of the question, based on the question classifier, and any other auxiliary information the question specifies. This may include attributes of the subject, it's relation to other entities, time or location in the case of events etc. This provides a means of deciding when a sufficiently appropriate answer has been identified. A breadth first approach is used to search for requested information while taking into account relevant labels, attributes and relationships between local entities.

Recursive Pathfinding
As long as the Analyser manages to correctly identify a starting node in the database (assuming the question implies several levels of relations), it then makes use of the statistical and structural information provided by the linguist in order to find one (or more) suit-

able steps into the adjacent nodes as per the breadth-first specification. This breadth-first approach ensures the proper identification of multiple answers if that is the case. If the question couldn't have more than one answer, then the steps into diverging nodes would be blocked by the implemented heuristics which rely on the contextual information, both of the text as well as the question. In this way we adopt a dynamic approach to identifying patterns in our database. The Analyser thus prunes the restrictions of the query that have already been identified and added to the path (both nodes and relations) in order to discover a pattern in the database that semantically fits the interpretation of the initial question.

3.4 Data

Having trained different types of classifiers, it stands to reason that training data was required. We have used various sources, based on the target classifier we were trying to obtain.

As mentioned earlier, NLTK provides a multitude of corpuses. The **Brown**[**?**] corpus was used to train the Naive-Bayes based chunker, the **Treebank**[21] corpus for sentences, the **Names** corpus together with all the location tags provided by the **Gazzetteer** corpus and various Wikipedia articles were used in extending the default Named Entity classifiers provided by Stanford's NLP package.

While access to the data was given to us, the data itself was not in the required format. As such, sentences had to be tokenized and words had to be manually annotated. Automatic annotation was used for names and locations, however, Wikipedia articles had to be manually annotated to ensure accuracy.

The performance measures of these classifiers are later described in the section 5.1.2.

3.5 Backend

As mentioned earlier, we used Django as the base for our backend. The backend works at sending the HTML views to the client's browser, receiving the queries from the user, processing these and sending them to the executor for them to be processed. Once the results are computed, it presents them back to the user. For the information exchange, we used HTML forms to send the queries to the server, and the reverse is done by injecting the results into Django's View Templates.

In order to perform user command interpretation, a simple function that maps selected keywords to the Executor commands.

The executor calls are done with the Command-line interface, making the system calls from the Django request handler in Python.

3.6 Front-end

3.6.1 Command-line interface

The Executor, as discussed in 3.3.1, provides an argument-based interface for Linux and OS X operating systems. The flags have been chosen to be as descriptive as possible and error messages and comments are abundant for a simple use.

3.6.2 Web interface

The interface of Seek's web application is intended for the general user. Therefore, it only provides the most basic and essential functionality, such as uploading files for Seek to analyse and add to its knowledge base, and inputting queries. All of the processing takes place in the background while the user is presented with a brief 'thinking' animation, followed by concise results on their actions. Here, the user is given the option of asking Seek for further knowledge on the query, as well as that of informing Seek whether the answer was satisfactory.

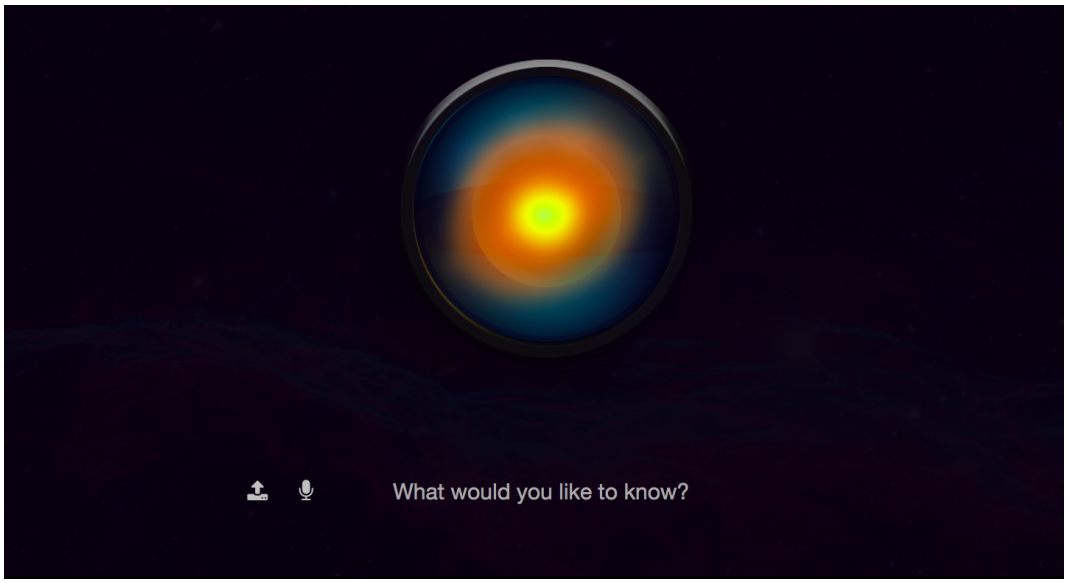

Figure 6: The web app homepage.

The leading design principle throughout the development of Seek was seamlessness, which also applied to the UX/UI design. This resulted in a clean and simple user interface, as explained above and shown in figures 6, 7 and 8. The interface reacts with animations, colours, and immersive effects. Seek's light turns red when given bad feedback, and turns green for positive feedback.

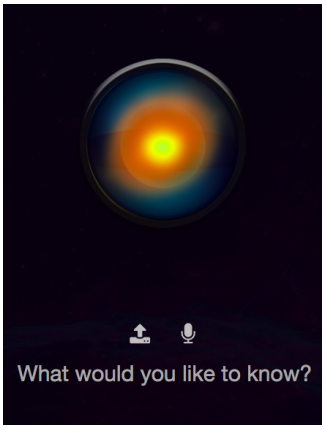

Figure 7: Techniques performed over questions

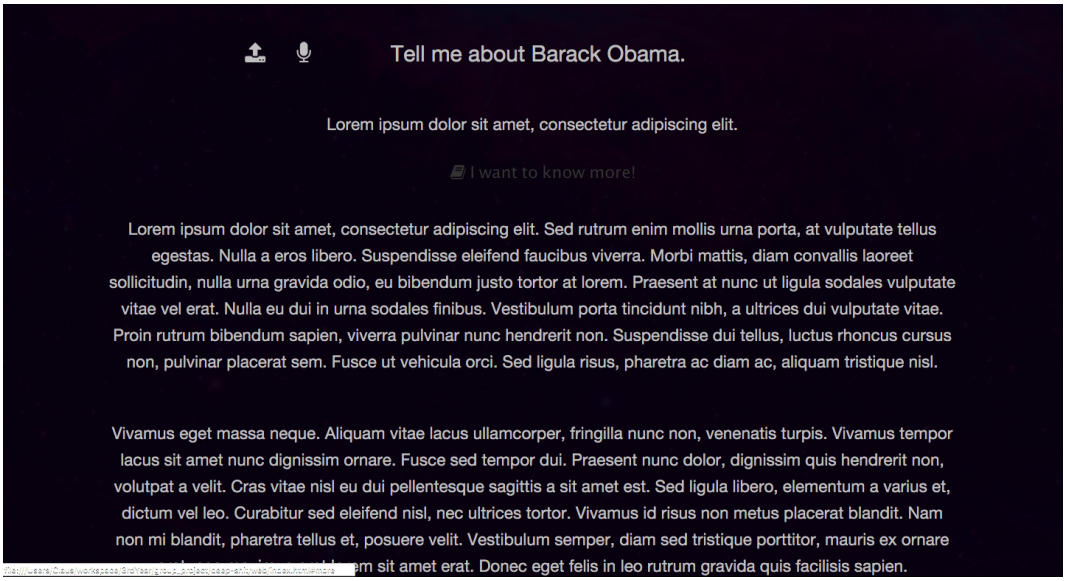

Figure 8: The results section of the web app.

3.7 Risks and challenges

3.7.1 Risk assessment

Natural language processing has some inherent risks that, although could be overcome with further research, have not yet been fully addressed. If these turn out to be unavoidable issues, then Seek's comprehensive capabilities will be greatly and permanently hindered.

Validating user input
If a user uploads a file, it will immediately be analysed, and its content will be stored by Seek. There are currently no methods implemented in our product for discerning whether the files contain valid information, or wrong, possibly even malicious, information. There exist several different solutions to this problem, such as ranking users based on their uploads, or reducing the validity of nodes of information according to how many times contradictory or equivalent information has been found.

The limits of unsupervised learning
Named entities classifiers cannot classify the types of entities that they have not yet encountered. In the phrase: 'Thomas Jefferson likes red roses', an unsupervised classifier cannot classify red as a colour if it does not know what a colour is. Although this would be feasible in supervised learning, it is not acceptable for our goals to leave such tasks to the user; besides having no means to prove that the supervised approach would truly succeed, it is also a lengthy operation that adds unnecessary complications to an otherwise smoothly running product.

3.7.2 Challenges

Devising a product from research
The beginning of the project was completely research-oriented. We dived into research of trending and old topic modelling algorithms, and began reading up on different machine learning approaches to these problems. Rather than starting with a strong idea of what we wanted to achieve, we first explored on the available technologies. We had no previous

experience on creating a product from an almost complete lack of specification or design document. Piecing together the output of our research with the technologies that we knew were available out there, we came up with our final idea. Every search engine allows you to query information that is already online, but no engine allows for queries on unstructured local data with an increasing knowledge base. It was certainly challenging to move from pure research to coding and designing a product that could be used by both the general public and specialists in various fields. This grew ever simpler as we progressed through the project, and gained experience in merging both creation and research.

Deciding over a database structure
MongoDB is a document based NoSQL database. Mongo's strength relies on being non-structured; this appeared to work well with how we envisioned Seek, as we will incrementally fill our knowledge base with data that will not be complete at the beginning.

Ultimately, we decided against using MongoDB as it offered no special help when storing relations between entities. We need to link different named entities together to create an enormous knowledge graph. When we realised that keeping these relations was as important as the entities themselves, we knew that we had to use a graph database.

Using a NoSQL database
Due to the unstructured nature of our data, classic relational database management systems (RDBMS) were not a feasible option. To the user it makes no difference whether a relational or a NoSQL database is used, but it means the world to an application such as ours. Identifying what information to store is always a challenge for any database programmer. However, in our case, mere information was not enough, as we also needed to know the relations between different pieces of data and their surroundings. Following research on many NoSQL approaches such as column family, document, and graph based databases, we opted for the last one as the best fit to our purposes.

4 Project Management

A thorough analysis of our objectives was necessary in order to efficiently organise and prioritise work. Moreover, given the large amounts of research and work we estimated we had to dedicate to this project, we decided to approach development in a tightly structured manner. The project was divided into several self-contained programs that are meant to work together in a cascading fashion in order to achieve the final goal. We approached these programs in the same way we would approach any other full project, by dividing tasks between us along the lines of research, development and testing.
We extensively made use of Slack and Trello for dividing the work between ourselves and communicating efficiently when personal meetings were not an option. Between these two, the latter proved itself crucial to the tight structure we designed for the development plan.

4.1 Milestones

In order to visualise and grasp the full extent of our project and the tasks we need to achieve, a plan containing the list of tasks was necessary, followed by chunking these into milestones. The result consisted of three large milestones, each encompassing five weeks of the total development time given for the project. Concomitantly with the temporal milestones, the set of tasks was divided into incremental phases. Each phase contained tasks that could have been performed concurrently by multiple developers, and built upon the completed tasks of the previous phase.

Below is a structured list of the milestones, the development phases and tasks encompassed by them, together with the change log of program versions and the features added for each phase.

MILESTONE 1 (12 Oct - 8 Nov)

PHASE 0: Prerequisites

This phase was crucial to the project insofar as it allowed us to develop the core objectives and learn about methods. The results of this phase are described in Section 1.2: Research, Section 2.2: Group organization and management and Section 4: Design. As this project is strongly research focused, this was an incremental process that carried on for the entire length of the project.

PHASE 1: Gatherer

Surely, the most important part in analysing large amounts of data is acquiring said data in an easy and efficient manner. During this phase we have conducted research into various modules we could use for raw text extraction from documents and decided on what data to use for training and development purposes. We have also implemented the scrapers to provide said training data.

Tasks	Features added
▷ crawl websites for documents ▷ scrape websites for plain text information ▷ extract plain text from multiple file types ▷ indexing large corpora (Wikipedia)	▷ build 0.1.0 : converts documents to plain text ▷ build 0.1.1 : scrapes Imperial Spiral repository for PDF documents ▷ build 0.1.2 : scrapes HTML files at given URL and extract text ▷ build 0.1.4 : extracts text from images

MILESTONE 2 (9 Nov - 6 Dec)

PHASE 2: Executor

Developing a fully-fledged product requires a certain degree of dedication to ease its use and accessibility. Thus, we started this phase by writing a prototypical bash script which would make sure the client benefits from all the intricate dependencies our program requires. More so, we developed this crucial part of the program with multi-threading as a first priority, given that it is to be used for handling large numbers of files.

Tasks	Features added
▷ install script ▷ run/update script ▷ run parallel batch commands for many files ▷ set up server ▷ process data in the cloud ▷ interface for running commands locally or in the cloud unit test the main processes	▷ build 0.2.0: separate script that triggers scraping, extraction, processing ▷ build 0.2.1: concurrency improvements ▷ build 0.2.2: command-line flag interface for the above ▷ build 0.2.3: interface allows client-server communication

PHASE 3: Text Processing

Most relevant to the project at hand, text processing began as a simple tokenisation feature meant to provide us with access to text at word and sentence level. From here, we employed a statistical (LDA) approach to determining topics of given documents based on training on word frequencies and vocabularies. More complex features such as detection of named entities or multi-word expressions have been implemented in a more rudimentary way than initially desired because of unexpectedly expensive training times.

Tasks	Features added
▷ basic linguistic analysis (tokenization, vocabularies, word frequencies, word filtering) ▷ text normalization ▷ topic extraction ▷ Named Entity extraction ▷ text summarising	▷ build 0.3.0: word and sentence tokenize, find the vocabulary, word frequencies, and filter out stop words and specific parts of speech ▷ build 0.3.1: summarising based on word frequency ▷ build 0.3.2: untrained topic extraction with LSI and LDA models ▷ build 0.3.3: statistically trained sentence tokenizer ▷ build 0.3.4: statistically trained collocation and multi-word expression analysis ▷ build 0.3.5: statistically trained chunker ▷ build 0.3.6: statistically trained Named Entity recognition ▷ build 0.3.7: chaining features for realistic results on text

MILESTONE 3 (7 Dec - 3 Jan)

PHASE 4: Question-Answering

The final step in the functionality chain is, of course, retrieving the data in an intuitive manner. The question answering system accomplishes this by searching the constructed database based on the identified subject of the query and the additional secondary information provided by the question inputted by the user. It is worth noting that this answering system accepts questions formulated in natural language thanks to our extensive language interpreting capabilities.

Tasks	Features added
▷ set up database ▷ document classification ▷ smart text summarising ▷ information extraction for populating the database ▷ improvement and training of current features ▷ improve the database by manually feeding structured data ▷ question classification ▷ transform question into database query ▷ query corpus for specific documents	▷ build 0.4.0: text summarising augmented with topics and Named Entities ▷ build 0.4.1: statistically trained question classifier ▷ build 0.4.2: queries database to receive simple answer ▷ build 0.4.3: queries data corpus to receive summarised information

PHASE 5: User interface

Building upon the executor's capabilities, the user interface is meant to link the data handling capabilities of Seek with both the client and the server, depending on the user's needs, all within a sleek interface. The executor's flags can be easily hooked into the Django framework used for the actual interface, while the question answering system presented above will act as the natural language input-output system of the app.

Tasks	Features added
▷ setup Django app ▷ link Django backend to Seek core ▷ basic user interface ▷ improved user interface ▷ speech recognition and synthesis ▷ basic chatbot skills	▷ build 0.5.1: web interface ▷ build 0.5.2: Seek can speak! ▷ build 0.5.3: chatting skills

4.2 Group organization and management

This section describes the techniques and workflows utilised by our team to manage tasks and workload. The main approach used is Extreme Programming, but we also employed other techniques.

4.2.1 Extreme Programming (XP)

Extreme programming (XP) is a software development methodology aimed at improving software quality adjustable to changes in customer requirements. As a type of agile process, this is accomplished by: short development cycles, unit testing, simple and minimalist code, pair programming and a flat management structure. Extreme programming teams are self organized and focus on solving problems and adding features only when they are explicitly required.

The following sections describe the techniques we have employed after researching various techniques used in industry for project management. Most of the techniques we used are part of the XP workflow, but techniques from Kanban have been utilized as well.

4.2.2 Pair programming

The technique that we adapted to a greater extent from Extreme Programming was pair programming. Everyone in our group had a different programming background as each of us had worked on different projects previously. We split the group into 3 pairs and gave a task to each one. Our goal was to match someone with previous knowledge on the topic allocated to a pair with another group member who wanted to learn more about it. As new pairs were formed at the beginning a new milestone, we kept on introducing members to new challenges whilst ensuring that someone with some expertise on the topic would lead the pair in a good direction. Pair programming also encouraged new ideas during the development process and made sure that there was always one person that could work on Seek and be present during meetings.

4.2.3 Flat management structure

Slack is a professional messaging app for teams. Different 'channels' can be created for each individual part of the project. Taking advantage of the flat hierarchical structure in XP, we made sure that everyone had access to every channel so that anyone could read all discussions. In this way, everyone could judge, review and weigh the choices that other pairs were taking as well as giving feedback to how they were proceeding.

4.2.4 Trello boards

One of the main features of Kanban is the existence of a whiteboard containing the full planning specification of the project. Maintaining a physical board for such a long period of time was unfeasible, so the whiteboard has been replaced by digital Trello boards for each Phase, Task and Research Material.

These boards allowed everyone 24/7 access to their tasks and the current group progress, thus preserving everyone's individual schedule and their ability to work whenever they could.

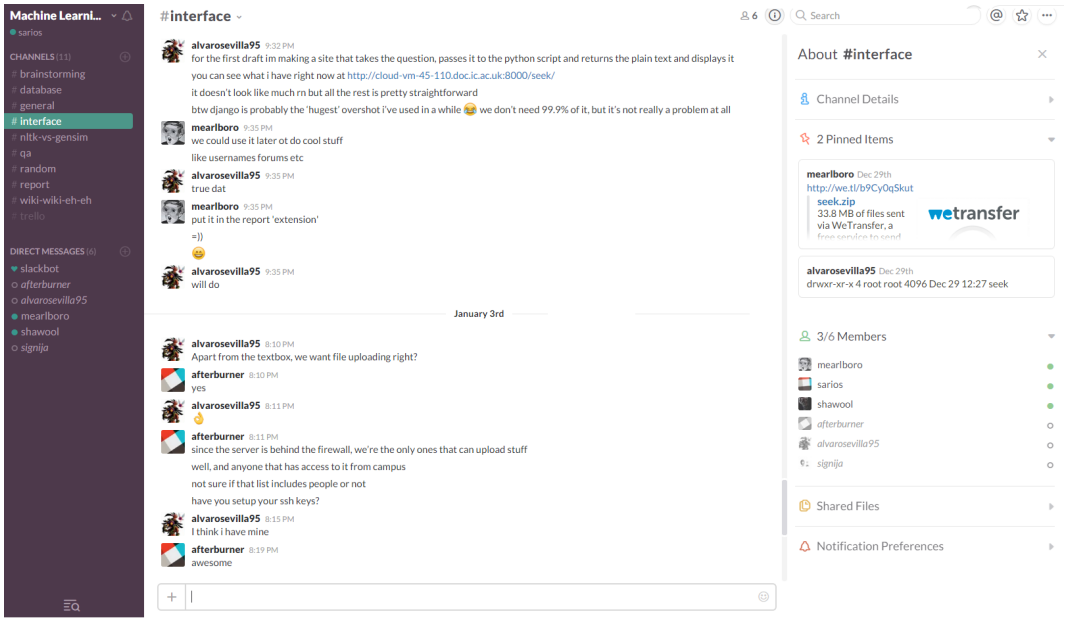

Figure 9: An example Slack board.

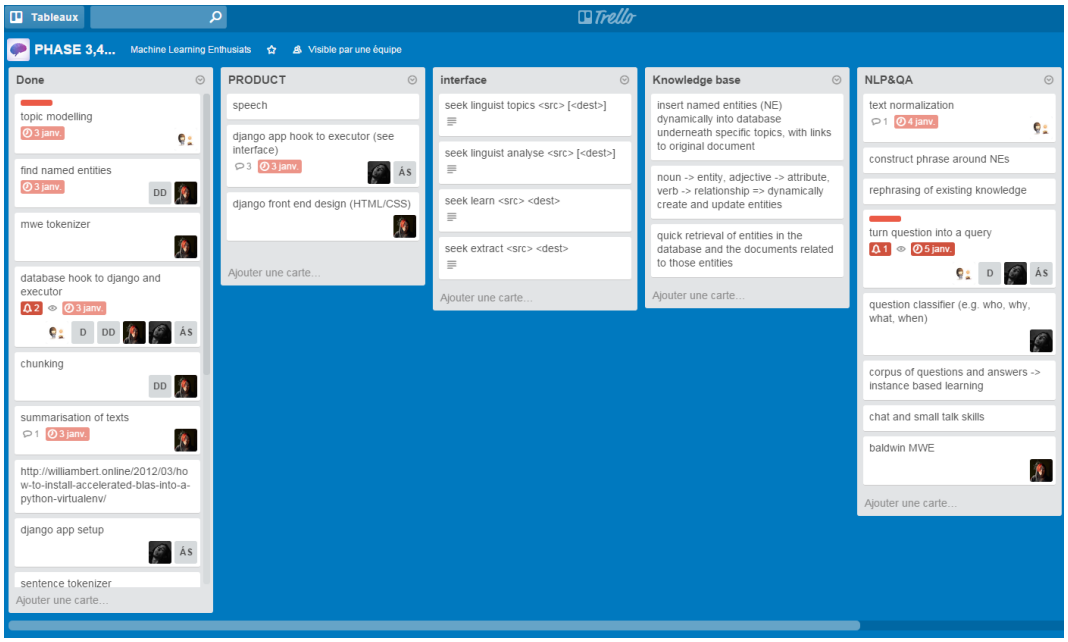

Figure 10: An example Trello board.

4.2.5 Group meetings

Thanks to the technologies we employed and the workflow suggested by XP, we did not have to fully rely on our group meetings. Various meetings of indeterminate duration a week were obviously scheduled between pair members, and moreover members of different teams met when they needed to discuss research findings and discuss the interfaces of communicating modules.

On top of everything, we made sure to hold a meeting with every group member at the end of every phase and every milestone. In these we let everyone know what each pair had been up to, shared any thoughts and feedback on the project so far, discussed the report, decided on how to proceed and what new pairs were going to be formed next.

4.2.6 Supervisor meetings

Working on such a large project does not come without challenges, be they related to research or implementation alike, the meetings with our supervisor proved invaluable. We have attempted to have weekly meetings for the most part of the term during which we would discuss what we are currently working on, what problems we are currently facing or what future theoretical and technological issues might arise. Throughout this guiding process, our supervisor eased the process by either pointing our research in the right direction, suggesting various resources as well as indicating any unforseen issues we were likely to encounter, thus keeping us from wasting time.

4.3 Task allocation

4.3.1 Main task allocation

Madalina Ioana Sas lead the research and general design, organised the code and overviewed the entire process. Implemented various information retrieval techniques and trained classifiers with Dragos, and helped Claudia with the front-end.

Dragos Dumitrache was tasked with natural language processing and machine learning research, namely LDA and LSA algorithms. He trained classifiers used for Named Entity Recognition and sentence chunking, helped Alvaro with the back-end implementation and designed the scraper tool for data gathering. He also assisted Madalina in overviewing the project and maintaining a clean instance of the project on the server.

Claudia Mihai was in charge of the open-source software and copyright research throughout the project. She has also worked together with Madalina on developing the front-end of the web application.

Tudor Cosmiuc conducted research and assisted with the implementation of the various open-source modules used at the beginning of the project. Designed and helped implement the question-answer module together with Dani and Madalina.

Daniel Hernandez was responsible for choosing and working on a fitting database design for Seek. He partially developed the question - answer module that would translate natural language questions into database queries that would yield an appropriate answer.

Alvaro Sevilla worked on the server infrastructure and maintenance, and on the back-end for the web application.

4.3.2 Support tasks

As our project involved large quantities of research and a very broad knowledge of technologies and means to achieve various tasks, it became imminent that we had to distribute our responsibilities over the entire length of the project. We learned that developing a product isn't merely writing code, creating an algorithm, or some complicated script. Developing a product requires integration, automation, and pleasant user interaction, things we had to tackle separately from the implementation of core features. Therefore, after the thorough evaluation of the project's goals and the product's features and what we need in order to do so, we came up with a list of tasks that had to be continuously handled separately from the development itself:

1. Open-source software and copyright research
2. Documentation and reports
3. Testing
4. Server, web app and database management

In order to most efficiently deal with the tasks above, one or two members of the team have offered to oversee each tasks and been 'elected' to perform those tasks.

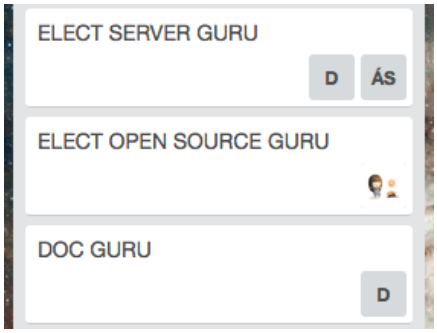

Figure 11: Trello voting board for task assignment

5 Testing and Evaluation

5.1 Testing

Not only from practical reasons, but also at a philosophical level, we value the consistency and accuracy of any powerful tool – and Seek is no exception. Such a promising tool needs the guarantee that it fully satisfies its specification. Thus, we have implemented multiple methods to continuously test and integrate new features, and maintain the core as fail-safe as necessary.

Given the text processing nature of our project, it stands to reason that we needed to retrieve a vast amount of documents in order to train our algorithms. Perfect execution was not necessary in these cases. If we are to retrieve, for example, thousands of articles from Wikipedia, it does not make an appreciable difference whether we lose 1% of the files that we should be downloading, but it makes all the difference if we manage to download and process them faster. Although it could be argued that the lost data could have contained very relevant information, we assume that important information or facts would appear frequently in other documents. Therefore, the likelihood of losing very important facts is negligible.

5.1.1 Unit testing

All of our core features such as the extractor, executor, or linguist are fully unit tested. A purely bug-free implementation of these modules was mandatory as a minor error on any of them would propagate through the entire system. Due to Seek's incremental learning nature, a single bug fired by an interaction with one user could negatively affect the experience of everyone else using our system.

We tested our modules according to unit test standards: every test should verify a single assumption about a unit of work. Test sizes were kept to a minimum, which in turn allowed for more tests to be written, and made it easier to find and correct bugs.

Although we initially intended to have full test coverage, there are isolated modules of our code (e.g., our data gatherers) for which this would be unfeasible. For these, classical unit testing at system level is inconclusive. As evaluating whether they work perfectly would require too much processing time (i.e., checking if any input file has been corrupted in any way during the uploading process). In such case we have used incremental development to assert the quantity of files downloaded and the speed of processing, and then refined our code in order to improve these metrics. We have pushed concurrency to the maximum, and we have learned that we need to do our best to reduce redundancy by not processing or downloading the same file multiple times. Our own server is an example of incremental development. As we had to fine-tune our modules' demands in order to optimize efficiency over the exhaustion of system resources.

5.1.2 Performance measures

The tables below represent the performance measures of our current classifier compared to two of the defaults Stanford classifiers.

They have been computed on a manually annotated train file, with an entity per line, and comprising of 339 lines, out of which only 54 were mapped to one of the entity types, and the remaining 285 where mapped to 'O'.

All the training has been done on a dual core i7 processor machine with 8GB of RAM, the named entities classifier alone requiring 7GB as Heap space.

Classifier	Classification rate
Naive Bayes	0.9276703111858705
Decision Tree	0.9671993271656855

Table 1: Sentence tokenizer on Treebank corpus

Entity	P	R	F1	TP	FP	FN
LOCATION	0.7273	0.8889	0.8000	8	3	1
ORGANIZATION	0.5000	0.3333	0.4000	2	2	4
PERSON	0.7273	0.6667	0.6957	8	3	4
TOTAL	0.6923	0.5455	0.6102	18	8	15

Table 2: General English classifier on Wikipedia Article

Entity	P	R	F1	TP	FP	FN
LOCATION	0.5823	0.7778	0.6667	7	5	2
ORGANIZATION	0.1429	0.1667	0.1538	1	5	5
PERSON	0.6667	0.6667	0.6667	8	4	4
TOTAL	0.4706	0.4848	0.4776	16	18	17

Table 3: English CONLL classifier on Wikipedia Article

Entity	P	R	F1	TP	FP	FN
DATE	1.0000	1.0000	1.0000	6	0	0
LOCATION	0.7237	0.8889	0.8000	8	3	1
ORGANIZATION	0.5000	0.3333	0.4000	2	2	4
PERSON	0.7237	0.6667	0.6957	8	3	4
TOTAL	0.6857	0.7273	0.7059	24	11	9

Table 4: Seek classifier on same Article

As can be observed, our classifier, described by table 4 has a slightly lower performance value than the default classifier described in table 2. However, our classifier can accurately describe the DATE entity type, making it more expressive than the default one. As such, this expressivity together with the increase of the F1 measure made the performance drop negligible.

ChunkParse	Score
IOB Accuracy	0.9700
Precision	0.9080
Recall	0.9390
F1-measure	0.9230

Table 5: Decision-Tree based chunker trained on the Treebank corpus

ChunkParse	Score
IOB Accuracy	0.9750
Precision	0.9310
Recall	0.9430
F1-measure	0.9370

Table 6: Naive-Bayes based chunker trained on the Treebank corpus

We have tried two different chunkers, one based on on `nltk.DecisionTreeClassifier` and the other one based on the `nltk.NaiveBayesClassifier`. The performance measures for both of these can be found in table 5 and table 6. From these two tables it is easily observed that the Naive Bayes Classifier yields slightly better results and as such supporting our decision for using it.

5.2 Evaluation

5.2.1 Performance evaluation

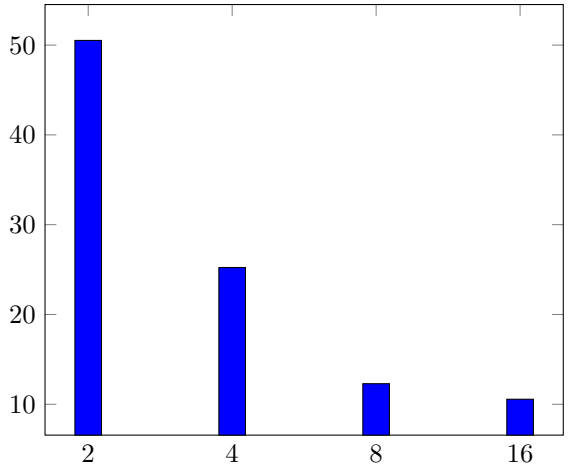

Figure 12: Performance on the executor over number of threads used over a fixed task of 295 documents

In terms of speed, our classifier has tagged 6282 words in 228 sentences at a rate of 1148.03 words per second. According to Amazon's Text Stats [22], the average word count for a book is 64.000 words. That means that we are able to fully tag the average book in under a minute. The average word count greatly varies depending on the book's field.

Let us also take into account the aforementioned Imperial's Spiral repository, which contained about 10.000 scientific reports, ranging from 2500 to 7500 words. Currently it would take us a whole month to fully process every single document. It is important to note that Seek needs to process a file only once, therefore loading whole libraries into our databases does not become unfeasible with regard to the processing effort. Moreover, our processing power is currently capped by our server's modest capacity; in a real case scenario, Seek's speed would of course be much greater.

5.2.2 Functionality evaluation

Seek manages to traverse the whole abstraction process, from prime, unstructured and unformated text to the syntactic decomposition of the converted raw text and finally to a well-structured and statistically balanced semantic network. To achieve this in an efficient manner, it doesnt make full use of the power behind its technologies, although the curious user would find it straight-forward to add functionality to Seek's statistical and analytical capabilities.

6 Extensions and future plans

This section mentions the possible features that could be added to further improve Seek and transform it into a marketable product. At the moment, Seek works as a proof-of-concept as it is meant to be released as an open-source package to be further improved by the community.

6.1 Advanced Speech Recognition & Synthetiser

Obviously, one of the more superficial yet appealing and desirable features of such a product is the possibility of interacting with it in such a highly personalised (and personalisable) way. Features such as choice of artificial voice, usage of Seek as a personal assistant, or teaching it specific commands would greatly appeal to the general user, essentially serving as encouragement to interact with it, and thus aid its teaching.

6.2 Complete and Sound Question Interpretation

One of the more relevant improvements regarding user experience is allowing the user to formulate questions, however unstructured. In fact, coupled with the speech capabilites mentioned above, this would create a seamless system that can be interacted with by holding an actual verbal conversation.

7 Conclusion

Although it does not yet reach human level comprehension capabilities, natural language processing is still far from being a fully developed field. While indexing is the most famous method through search engines, our alternative approach was based on understanding the topics digested by our application and building a gargantuan knowledge graph. We are attempting to address a global issue of text summarising and information retrieval using tools that are already out there, but have not yet been used or approached by technicians in these fields.

As it can be imagined, there are also drawbacks and challenges to our approach. For instance, large volumes of data suffer from an inherent and unavoidable problem: inconsistency. Most relevant and important facts, scientific or historical, are unanimously accepted. This is not true for every event, which could result from incomplete historical accounts. Not wanting to make Seek biased on its information retrieval, a future plan could include a subsystem that would contrast conflicting records. Potential metrics would be based on the relevance of their source or how well accepted they are with respect to other documents that are about the same topic. For obvious reasons, natural language processing is not a panacea for possible future search engines. However, their development and practical uses are becoming more entangled as time and computer theory progress.

8 Appendix

8.1 Acknowledgements

TODO

8.2 Licence discussions

While working on the project, we looked to only work with open-source technologies with an appropriate license for us to be able to use them. These licenses included:

1. [27] Apache 2.0 (NLTK) This licence allows changes in the code and redistribution, as long as the original copyright is not altered, the licence file included, changes are stated and the notice file is included if it exists.
2. [28] MIT (Textract, Bootstrap, Skeleton): Less restrictive than Apache, it only demands for copyright to be included as well as the license.
3. [29] GPL v3 (Genism, Neo4J, Stanford NLP): It asks for the original software (or instructions on how to get it) to be included, as well as copyright and license. It also ask to state changes, and disclose the source along install instructions.
4. [30] BSD (Django): It demands for copyright and license to be included.

As we can see, all of these licences are very permissive, with the most notable restriction being GPL's requirement that we release the source code; in our case, this is definitely not a problem.

References

[1] Amit, S. *Introducing the Knowledge Graph: Things, Not Strings.* Google Official Blog, May 16, 2012, `https://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html`, retrieved 2016-01-08

[2] Bird, S., Loper, E., Klein, E., *Natural Language Processing with Python.* O'Reilly Media Inc, 2009. `http://nltk.org/`

[3] A. Turing. *Computing Machinery and Intelligence*, Mind LIX (236): 433–460, 1950, `doi:10.1093/mind/LIX.236.433`, ISSN 0026-4423, retrieved 2016-01-10

[4] DeRose, Steven J. 1988. "Grammatical category disambiguation by statistical optimization." Computational Linguistics 14(1): 31–39. [1]

[5] Reynolds, A. Craig (1954). "The conference on mechanical translation.". Mechanical Translation 1 (3), 47-55.

[6] Shrdlu. Stanford Website. `http://hci.stanford.edu/winograd/shrdlu/`, retrieved 2016-01-10

[7] Weizenbaum, J. *ELIZA - a computer program for the study of natural language communication between man and machine*, Communications of the ACM, v.9 n.1, Jan. 1966, `doi:10.1145/365153.365168`, retrieved 2016-01-10

[8] Bowden, Margaret A. (2006), *Mind As Machine: A History of Cognitive Science*, Oxford University Press, ISBN 978-0-19-924144-6. p. 370

[9] *Rachter.* Wikipedia, The Free Encyclopedia. Wikimedia Foundation. `https://en.wikipedia.org/wiki/Racter`, retrieved 2010-01-10

[10] G. Salton, M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[11] THIS WHERE IS IT PLEASE SORRY where are we referencig it?

[12] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, R. Harshman. *Indexing by latent semantic analysis.* Journal of the American Society of Information Science, 41(6):391–407, 1990.

[13] Blei, M., Ng, A., Jordan, M. Lafferty, J., ed. *Latent Dirichlet allocation.* Journal of Machine Learning Research 3 (4–5): pp. 993–1022. `doi:10.1162/jmlr.2003.3.4-5.993`. retrieved 2016-01-05

[14] Moon, S., Yoon, C. *Keyword-base Concept Nets Model for Information Retrieval in the Mobile Cloud*, Oxford University Press, 2014 2051-1329.

[15] J. R. Finkel, T. Grenager, C. Manning. *Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling.* Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370. `http://nlp.stanford.edu/ manning/papers/gibbscrf3.pdf`

[16] L. Zhang, Y. Pan, T. Zhang. *Focused Named Entity Recognition using Machine Learning.* Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, Sheffield, UK, 2004.

[17] Wallace, Richard S. *The Anatomy of A.L.I.C.E.*. Parsing the Turing test. London: Springer Science+Business Media. 2009, pp. 181–210. ISBN 978-1-4020-6710-5.

[18] Sharoff, S. *What is at Stake: a Case Study of Russian Expressions Starting with a Preposition*, Proceedings of the ACL 2004 Workshop on Multiword Expressions: Integrating Processing, Barcelona, Spain, 2004.

[19] Baldwin, T., Villavicencio, A. *Extracting the Unextractable: A Case Study on Verb-particles*, Proceedings of the Sixth Conference on Computational Natural Language Learning (CoNLL 2002), Taipei, Taiwan, 2002. pp. 98-104.

[20] McCarthy, D., Keller, B., Carroll, J. *Detecting a Continuum of Compositionality in Phrasal Verbs*, Proceedings of the ACL-SIGLEX Workshop on Multiword Expressions: Analysis, Acquisition and Treatment, Sapporo, Japan, 2003.

[21] The Penn Treebank project. University of Pennsylvania. `http://www.cis.upenn.edu/ treebank/`

[22] Brave New World Amazon Book Statistics, Amazon.com, Amazon.com, Inc. `http://www.amazon.com/Brave-New-World-Aldous-Huxley/dp/sitb-next/0060850523`, retrieved 2016-01-09

[23] Manning, C., Raghavan, P., Schütze, H., *Introduction to Information Retrieval*, Cambridge University Press, 2008, `http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html`, retrieved 2016-01-06

[24] R. Baeza-Yates, B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, 1999.

[25] The Stanford Multiword Expression Project, Standford `http://mwe.stanford.edu/resources/`

[26] *Sentence Segmentation: Handling multiple punctuation characters*. Winwaed Blog. 2012-06-13., retrieved 2016-01-07
`http://www.winwaed.com/blog/2012/06/13/sentence-segmentation-handling-multiple-punctuation-`

[27] Apache 2.0 license `http://www.apache.org/licenses/LICENSE-2.0`

[28] MIT license `https://opensource.org/licenses/MIT`

[29] GPL v3 license `http://www.gnu.org/licenses/gpl-3.0.en.html`

[30] BSD license `http://www.linfo.org/bsdlicense.html`