

Department of Computer Science



Submitted in part fulfilment for the degree of
MEng in Computer Science With Artificial Intelligence.

Extracting Hot Research Topics From Job Adverts

Adham Nouredin

03 May 2022

Supervisor: Dimitar Kazakov

Acknowledgements

I would like to thank my supervisor Dimitar Kazakov for his ongoing support in guiding my efforts to be able to fulfill the goals of the project.

Contents

Executive Summary	vii
1 Introduction	1
1.1 Motivation	1
1.2 Scope	2
1.3 Goals and objectives	2
2 Literature Review	3
2.1 Word embedding	3
2.2 Word2vec	4
2.3 Attention	4
2.4 Transformers (self-attention)	5
3 Problem analysis	6
3.1 Topic extraction and clustering	6
3.1.1 Semantic embedding	6
3.1.2 Off the shelf parser	7
3.1.3 State-of-the-art model	7
3.2 Evaluation	7
4 Design and implementation	9
4.1 Tfidf	9
4.2 Word2Vec	10
4.3 Off the self solution	11
4.4 Pre-trained model (GPT)	11
4.5 Saving the data	16
4.6 Hierarchical clustering	18
5 Results & Evaluation	19
5.1 Semantic modelling results	19
5.2 Database of job details	20
5.3 Keyword occurrences	23
5.4 Topic hierarchy and clustering	25
6 Conclusion & Further Work	28
6.1 Summary	28
6.2 Further work	28

A Appendix

30

List of Figures

3.1	Precision and recall illustration.	8
4.1	Sample email.	9
4.2	Sample downloaded email in plain text format.	10
4.3	Networks size comparison.	12
4.4	Example response from model.	14
4.5	MongoDBCompass.	17
5.1	Failed parsing example.	20
5.2	Salary field including a note example.	21
5.3	Response with multiple null jobs.	21
5.4	Missing information percentages.	22
5.5	Keyword occurrences counts from email body.	23
5.6	Keyword occurrences counts from topic (n-gram = 3).	24
5.7	Topic hierarchy.	25
5.8	Topic hierarchy (Highest counts).	26
A.1	Optimal number of clusters (elbow method).	30
A.2	Dendrogram hierarchy.	30
A.3	Keyword occurrences counts from topic (n-gram = 2).	31
A.4	Topic hierarchy (Medium counts).	31
A.5	Topic hierarchy (Lower counts).	32

Listings

4.1	Model API call with prompts	13
4.2	Regular expressions	15
4.3	Keyword extractor initialization	16
A.1	Topic hierarchy	33

Executive Summary

Natural language models have become increasingly popular due to their impressive capabilities in understanding pieces of text and being able to respond to them in a natural way. The aim of this project is to explore the possibility of using techniques from the natural language field to extract the topic or topics of jobs in email job advertisements to be able to calculate or visualize which topics have the most interest currently.

The motivation behind this project is to help researchers in being able to explore their field of interest to have a better understanding of where the industry is heading. This will not only aid them in making better and more informed decisions on where to take their careers but also it can possibly bring a certain job listing that is a very attractive proposition to their attention that they might've otherwise missed. Due to the ever-growing job market, the need for a way to automatically summarize these jobs and be able to have meta-information about the jobs themselves such as which institutions are behind these projects will be extremely useful for researchers to make the best decisions about their career.

Multiple methodologies ranging from more primitive to state-of-the-art will be applied and tested to see if they can achieve the desired results. The first attempt was using the Tfidf measure to try to calculate the relative importance of each word in every email and taking the top couple of scores to be the topic terms of each email. Secondly, a more advanced technique of word embedding was used called Word2vec where the semantics of each word was calculated and represented as a vector. This allowed the emails to be clustered based on the arithmetic distances of the vectors and the vectors that were closest to the centroid of each cluster were the ones that most represent them. Lastly, instead of using these more primitive techniques, I attempted to use an available pre-trained natural language model from one of the industry leaders to summarize each email to extract the information needed and present it in a consistent and structured manner to be converted into variables. Using the same model I attempted to rank the jobs based on their general topic into a hierarchy of topics to find out which topic has the most jobs, therefore, the most interest. Another way I tried to find out which topic has the most interest is by extracting the keywords and phrases from the extracted job topics and ranking the results based on counts and the most occurring keyword or phrase is the one that

Executive Summary

has the most interest.

To evaluate the results I aimed to achieve a parsing rate of over 80% and relatively high precision and recall scores. Despite having a parsing rate of 100%, the resulting topic keywords from the first two methodologies tried were poor due to the techniques not being able to accurately identify the correct important words as well as not being able to extract more than one-word long phrases. That resulted in poor topic extraction results as the keywords that are supposed to represent the topics were either too general or not containing useful information. The clustering that followed also was not ideal mainly due to the fact that each email contains multiple topics therefore clustering them based on whole emails resulted in no obvious relationships between the jobs that were clustered together. The state-of-the-art model however presented promising results. It managed to summarize and extract the topic of each job in every email separately with high precision and recall scores as well as a high parsing rate of emails without errors.

As I was able to convert most of the corpus of emails into a database of jobs, a clean interface could be built on top of it to present useful information using predefined queries. This can also allow a user to update or change the corpus of emails to run the topic extraction and classification code to get more up-to-date results. More effort could be put towards making the data extraction more reliable and decreasing the error rate during parsing even further.

There are no legal, social, ethical, professional, or commercial issues regarding this project as all the data is publicly available, and no personal data was used at any point during this project.

1 Introduction

Research jobs differ from regular jobs due to the differing nature of the jobs in terms of a research position is a project to be completed within a certain amount of time whereas a normal job does not necessarily have an end goal to work towards. Given this fact, researchers and academics will keep looking for new projects in multiple fields throughout their careers, so having information about a certain field of interest, for this project will be machine learning, will undoubtedly aid them in deciding which direction they would like to take their career based on what topics their field is most interested in.

Natural Language Processing has been an area of ongoing research and development for the past 50 years. In its beginnings, the techniques used were simple and inefficient, and as time progressed so did the research, and more and more complex and ingenious solutions and techniques started to arise and boost the performance of NLP models to never before seen heights. As of recently, the interest in this area of machine learning has skyrocketed due to massive leaps in the field therefore the models have become extremely good at understanding and responding to text prompts. These models have become so advanced that it is very difficult to distinguish between if it's an NLP model or a real human being talking in a conversation. Not only do they perform incredibly well in conversations, but they are also able to perform text summarization to an exceptional standard. This will be very useful to this project as it is much easier to extract the topic of a job advert from a summary of the advertisement rather than how it was presented originally.

1.1 Motivation

By just looking at the research being released nowadays doesn't give insight into what currently is being researched or what is going to be researched in the upcoming years as these projects started a couple of years ago, therefore, it only shows us what was being explored back then but by looking and analyzing the job adverts that are being posted today we can have insight on what fields and topics are currently being explored

and researched and by who which can act as a glimpse into the future of research. To do this I want to utilize an NLP (Natural Language Processing) model to be able to parse PhD Job adverts and extract the topic(s) of the advertisement and cluster them based on how similar the topics are, and finally output a list of the most popular topics. The content of the corpus I'm using to test my model is entirely collected from emails sent to the "ML-news" Google group. This group was chosen as it often has PHD Job adverts/listings advertised there with subjects relating to machine learning.

1.2 Scope

As I mentioned before, this work has the potential for some real-world applications for example, the database of ads and the extracted information about them this work would create can be browsed as you would a job listings website. The advantage of being able to autonomously extract the desired information from these ads would be that a user will be able to perform more detailed queries to find the specific information they are looking for. Another application is that because the corpus being used is made up of job ads, the information about the topics being advertised to be researched gives insight into what the research field will look like in the near future.

1.3 Goals and objectives

The goal of this project is to explore the field of research and find out what the current interests are and some meta-information about them such as which institutions are the ones behind the projects and who are the people supervising them. I want to be able to query a data set to get the answers to queries such as "What is the most popular topic" and "Who is funding these projects the most" as the answers to these questions based on the current jobs on offer will be able to help post-doctorates and researchers to find jobs in the field they're interested in and that are currently popular which means they are the most lucrative as well as give insight into the overall state of research as it currently stands within the field I choose to have my data set based on. To classify this project as a success I'm looking to achieve a parsing rate of above 80% over the relevant emails as well as having relatively high precision and recall scores in the topic extraction from the advertisements.

2 Literature Review

2.1 Word embedding

An early technique that was used to represent words inside of a model was word embedding. Word embedding worked as a translator of sorts to convert each word in its textual form into a numerical one, for example, a form of one-hot encoded vectors of length equal to the number of unique words in the corpus being used. This helped models learn and handle the data better as machine learning algorithms are much more efficient when dealing with structured numerical data than unstructured textual data. Some disadvantages of using the early word representation techniques include the very high number of dimensions of each vector representing the words, as well as the generalization problem which occurs due to the lack of ability in representing similarities between words therefore learned information about word relationships can't be leveraged. To advance this technique a new form of representation was used where instead of having a one-hot encoded vector, the models had a much shorter n -length vector with the numbers inside of this vector representing the semantics of a word. This allowed the representative vector for each word to be much shorter and denser with information regarding its corresponding word. The float values in the vector represent the word in a vector space that has similar words close to it as they appear in the same context in a piece of text, therefore, capturing the similarity between words. Another great feature of this representation is that you are able to do simple mathematical operations on a vector to get the vector of a word if the operations done have meaning if they were done onto the words themselves, for example if you take the vector of the word "King" and subtract the vector of the word "Man" and then add the vector of "Woman" then you would get a vector that is very close with the true vector representing the word "Queen" [1]. Vectors of relatively small size are able to capture multiple likenesses even when some of them aren't obvious straight away to a human reader. This technique falls short in the fact that it is only able to capture one meaning of a word therefore words with multiple meanings can sometimes be misrepresented/misunderstood by the algorithm as well as not being able to translate between dissimilar languages with very high accuracy.

2.2 Word2vec

One of the most revolutionary techniques based on word embedding that was developed is word2vec. This approach came from Mikolov et al.'s paper [2] in which they introduced it in two differing forms. One form is called Continuous bag-of-words where the neural network is designed in a way to take a couple of words before and after the current word and tries to guess the target word based on them and because of this, the order of the words does not matter therefore is named as such. The second form is Continuous skip-gram where the current word is what is fed into the neural network and it tries to predict the words around it based on a predefined range within a sentence.

2.3 Attention

Nowadays the most used architecture for large NLP models is a mechanism called attention. This architecture was designed to mimic how humans read and comprehend text. Humans spend more time looking at, meaning giving more attention to, certain words in a piece of text as they are recalling the context of how the word they are reading is called in and various other variables that contribute to the comprehension of it [3]. Attention was designed and introduced by Bahdanau et al. in 2015 [4] where they applied this concept to surpass the bottleneck of the encode-decoder architecture, which was the most used architecture before attention was introduced, where the encoder transforms an input to a fixed length vector and the decoder used that vector to transform it into the desired output. The bottleneck in that design was the fact that there's so much information that a fixed-length vector can hold therefore that compression will result in information loss which results in the output being not ideal. Attention fixed this issue by giving the model the ability to selectively "focus" on certain parts of an input string when generating an output token. This is done by computing the weights of the input vector using different key vectors to relate it to. This allows the model to surpass the limitation of the encoder-decoder architecture as the information window isn't bound by the vector size anymore and is only bound by the computational resources at the model's disposal when training as the decoder can now access any part of the original input string which is encoded into a vector by the encoder. These models were mainly aimed at language translation and used qualitative performance tests to assess each model's translation quality which in Bahdanau's paper, where attention was in its initial stages, achieved results that rival those of the state-of-the-art systems in those times.

2.4 Transformers (self-attention)

The current state-of-the-art models are mostly based on a transformer architecture [5]. This is a more advanced version of the attention model where instead of computing the attention weights based on a comparison between the input and different key vectors, the weights are calculated based on a comparison between the input string and itself. This enables the model to capture long-distance relationships between words in the input string as the distance isn't a contributing factor in the weights calculation as it evaluates all possible pairs of words without including where they are in the sequence. There are other advantages self-attention has over regular attention. Self-attention is much more computationally efficient due to its high parallelization potential as each word pair evaluation can be computed independently from each other.

This architecture has dominated the industry of NLP models as a result of its outstanding performance and relative efficiency of training. The main models to note are OpenAI's GPT-3 [6] and GPT-4 [7] models and Google's Bard model which as of writing this paper Google hasn't released the research paper behind it as it's still in the experimental phase. These were both designed using the transformer model as their goal is natural language comprehension and question answering which this type of model excels in alongside text summarization and machine translation.

3 Problem analysis

Topic extraction and clustering could be done in multiple different ways ranging from simpler methods such as using simple word embedding like Tfidf and the Word2vec model to much more complex solutions such as utilizing a very large natural language model. There are also other methods such as an off-the-shelf parser I could use to parse the emails and extract the topic information.

3.1 Topic extraction and clustering

3.1.1 Semantic embedding

As mentioned in the literature review, word embedding can have multiple forms. I chose to use the Tfidf measure to start with as it is a very simple way to encode the relative "importance" of words in a corpus of documents per document. This works well in my case due to the fact that my corpus of job advertisements is relatively small as well as each email being only one or two paragraphs long therefore not containing that many unique words so the size of the feature vectors in the model is not too large to cause memory issues. The vectorized documents fit well in a dendrogram due to its numerical nature I can calculate the euclidean distances between the vectors therefore essentially being able to find out how similar two documents are and clustering them accordingly.

To use a more advanced embedding technique I chose the Word2vec model. I chose this model as an attempt to see whether or not the change in document vectorizing helps improve topic extraction and cluster quality or not. Unlike the Tfidf method, here I printed out the most representative terms as well as the most representative document per cluster. This gave me a better insight into what the results of the vectorizing were and I used that to compare against the clustering results of using the first method. The most representative terms per cluster can also be used to answer the overarching project question where the clusters with the most documents mean that those terms have the most interest in them due to the cluster having the relatively highest document count.

3.1.2 Off the shelf parser

Trying to change the approach towards extracting the information from the emails I attempted to use an off-the-shelf document parser. To set up the parsing I have to forward an email to the parser's email and locate the fields of text I want to be extracted and assign each field with a variable name. Having set up those fields I will just have to forward the emails I want to have those exact same text locations to be extracted to their corresponding variables and it will automatically extract the text.

3.1.3 State-of-the-art model

Finally, as the cutting edge of the natural language processing field currently is transformer based models I'm going to attempt to make use of one of the available models that currently are the best in the field. These models will be able to "understand" the emails I provide them and extract the information I need better than any semantic embedding technique I can use short of creating one of these networks myself. The point of using this model is to essentially convert an email containing an arbitrary amount of jobs on offer to a response message where the information I'm seeking is presented in a consistent and standardized fashion. This will allow me to create regular expressions based on this format to extract the information consistently achieving the parsing rate goal. Not only can this model summarize the emails to present the information I'm looking for, but it can also construct a topic hierarchy based on the general topics of the job descriptions. Having this hierarchy will show us the topics of the jobs present in the corpus as well as the popularity of each of them through the count of jobs classified into that topic, therefore, answering the main project question. To query the jobs further I will save the model's responses into a database to perform queries over the entire dataset and find more information about specific jobs in any of the topic clusters.

3.2 Evaluation

As stated in the project goals and objectives (section 1.3) I'm aiming to achieve a parsing rate of over 80%. This specifically means that for the entire corpus of emails, I'm aiming to be able to extract at least the topics from 80% of them. This is slightly complicated by the fact that there could be more than one job listing per email. To be slightly pessimistic I'm going to assume that if there is an error when parsing an email then I didn't

3 Problem analysis

manage to extract any data from that email and calculate the parsing rate as emails that ran into errors over the total number of emails.

The other aspect of this evaluation is the qualitative analysis of the cluster and topic extraction quality. Tackling the latter first, there is no way to automate the evaluation of how well the topic of a job in the email is extracted as there is no ground truth to compare against. A manual review of at least 5 emails will be done and I will extract the topics of all the jobs I find in those emails before running the automated extraction methods. This will give an unbiased opinion on what the ground truth for the topic of the job is and how many jobs there are in those emails. This will give me enough data points to be able to calculate the recall by taking the number of job topics the extraction method produces and dividing it by the number of topics I manually extracted as well as calculating the recall by reviewing the topics automatically extracted and calculating the number of these topics correspond to the ground truth topics over the total number of topics the method produced.

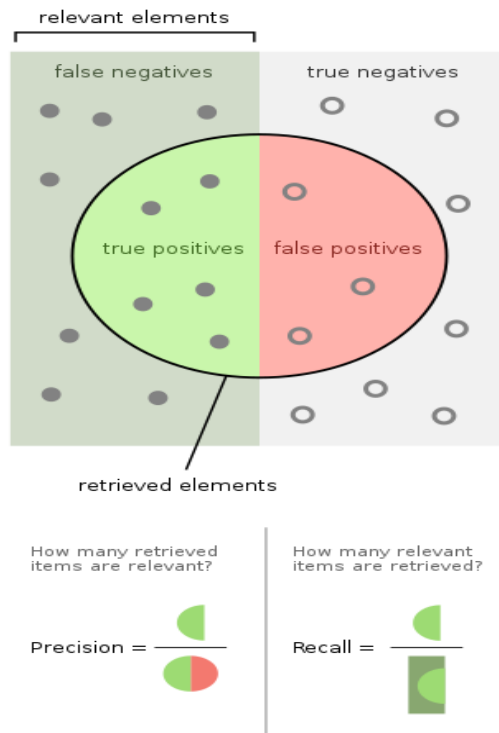



Figure 3.1: Precision and recall illustration.

4 Design and implementation

To collect the data to parse into the trained NLP I had to collate emails sent to the "ML-news" Google group as mentioned in the introduction to be able to extract the most brought-up research topics. I did this by labeling each email I received from the Google group with a custom label in Gmail, then using the Google Takeout service, which allows users to extract their Google Account's data, to download all the tagged emails using the label and have them all in the mbox file format. Using the Python library mailbox, I was able to extract the content body of each email and pass it into the NLP for it to extract the topics. I noticed that not all of the emails sent to the ML-news group are job advertisements so I needed to filter them somehow. I did this by extracting the title of the email alongside the body and searching if it contains some indication that the email contains job advertisements such as having "[Job]" in the title or mentioning a PhD or fellowship.

INDUSTRIAL CASE PhD Studentship: A Co-operative Algorithm Framework for Solving Large-Scale Heterogeneous Problems with Multiple Objectives 28 views

 masa@cs.man.ac.uk
in Machine Learning News

Qualification Type: PhD
Location: Manchester
Funding for UK Students
Funding source: EPSRC
Hours: Full Time
Closed: 20th February 2023

Under this research studentship, the successful candidate will conduct research as part of the EPSRC Industrial CASE Award.
<https://www.epsrc.ac.uk/industrialcase/industrialcase.asp>

This project is available ONLY for UK (Home) students.
<https://www.ukaea.org.uk/Information/About/Press-and-Media/English-Higher-education/epsc>

Theme of the PhD project

Rapid developments needed in new energy technologies (e.g. for scalable generation plants) that can provide a continuous supply will depend upon solving hard and large-scale global optimisation problems. However algorithms for academic benchmark problems are well studied, progress in more realistic optimisation settings is being offset by a lack of deep understanding about real problems as a multiplicity and an interplay of non-convexities, noise, and what's needed is a deep dive into how to efficiently solve such problems cooperatively. There is a number of related research themes that can be brought together to tackle this challenge and make progress. These include: using heuristics of deep-objective solvers to cooperatively tackle multi-objective problems; automatic algorithm configuration and selection methods that can combine and tune algorithmic components; and so-called "benchmark" problems. Our aim will be to discover common principles among these ideas, and bring them together into a new paradigm.

For more details about this project, please see:
<https://openresearch.manch.ac.uk/industrialcase-phd2023>

Supervisor

The successful candidate will be supervised by Dr. Masaru Yoshida (University of Manchester, PhD, Japan Institute of Science) who act as external supervisor and lead the industrial input into this research.

Nature of the studentship

Figure 4.1: Sample email.

4.1 Tf-idf

Before I used google takeout as mentioned before, I downloaded some sample emails manually by using the Microsoft Outlook application and downloaded each email in a plain text format and the output format of that method is presented in figure 4.2. As you can see from the figure there

4 Design and implementation

are a couple of lines above the text that include metadata on the email as well as the subject. Since this format doesn't download the body of the text alone, I have to include some preprocessing of the text to isolate the body of the email, extract the subject line and remove the last couple of lines in the file as they are in every email received from a google group which doesn't add any useful information about the email. After that is completed I had a body of text with a subject line which I used to name each email then I removed all the English stopwords from the text as they don't contribute to the meaning of the sentences therefore will not be needed in the later steps as well as filtering all the punctuation. By filtering out all of those I now only have the words in the body of the email that are useful to extract the topics that the email is about. I attempted to use the Tfidf method to calculate the importance of each word in the document to be able to extract the highest values as the topics of that email. I did this by implementing the TfidfVectorizer method from the Sklearn python library by setting the max features to a smaller and more manageable number of 100 also with the specified stopwords I mentioned earlier being removed. This outputted the "importance" value of each word relative to every separate email and then using KMeans clustering, with the elbow method to determine the number of clusters that best fit this data, I clustered the documents that are the most similar to each other. I used the elbow method as it is regarded as one of the most popular methods used to calculate the optimal number of clusters for any given dataset.

```
From: ml-news@googlegroups.com on behalf of Hasti Seifi <hseifi@asu.edu>
Sent: 22 December 2022 1:43 AM
To: Machine Learning News
Subject: [ML-news] 2-Year Postdoc in Natural Language Processing

We are seeking to hire a 2-year postdoc on natural language processing (NLP), haptics, and human-computer interaction (HCI) at the University of Copenhagen (UCPH) in Denmark. The position also includes a research stay at Arizona State University (ASU) in the US. The postdoc can closely collaborate with NLP and HCI groups at UCPH and ASU and can build their academic network across the world. The project is about haptic captioning (akin to image captioning) which aims to predict user descriptions for programmable touch signals.

The application deadline is Jan 6, 2023. The start date for the position is March 1st 2023 or soon after.

Candidates should have a strong background in natural language processing, haptics, or human-computer interaction. Additional expectations include:
* PhD in computer science, electrical engineering, or a closely related fields
* A publication record in top international conferences and journals
* Excellent programming skills
* Strong writing and communication skills

Description of Scientific Environments
University of Copenhagen
The Department of Computer Science has globally recognized, highly collaborative groups of international researchers working at the forefront of Natural Language Processing and Human-Centered Computing. The position is located in Copenhagen, consistently recognized as one of the world's most livable cities, in the country of Denmark, ranked as one of the top three happiest countries in the world.

Arizona State University
We are part of the School of Computing and Augmented Intelligence (SCAI), one of the seven schools in the Ira A. Fulton Schools of Engineering at Arizona State University (ASU). ASU SCAI is consistently ranked among the top CS departments in the world. We are located at the beautiful Tempe campus, part of the Phoenix metropolitan area, the nation's fifth largest city.

Candidates can find more information here: https://tinyurl.com/haptic-language For questions about the position, please contact hasti.seifi@asu.edu.

--
You received this message because you are subscribed to the Google Groups "Machine Learning News" group.
To unsubscribe from this group and stop receiving emails from it, send an email to ml-newsunsubscribe@googlegroups.com.
To view this discussion on the web visit https://groups.google.com/d/msgid/ml-news/1a3f978e-4bf8-411a-a3d7-0a3f583022d6n040@googlegroups.com.
```

Figure 4.2: Sample downloaded email in plain text format.

4.2 Word2Vec

To improve on this attempt, I switched from using Tfidf to represent words to using a Word2Vec model to embed the semantics of each word, relative to the words around it, in a one-hundred-element long vector which I believe is long enough to capture as much data on the semantics of each tokenized

word as possible without being exceedingly large to cause memory issues. After training the Word2Vec model on the data, I calculate a feature vector for every document in the corpus by taking the average over all the word vectors per document and using that to represent the document. Finally, I clustered the documents together using Mini-batches Kmeans clustering. This method was chosen because of the efficiency boost it provides as it takes random samples from the data which decreases the training time therefore being more applicable when I run the algorithm on the full corpus of documents. To analyze and review the results of this attempt I calculate the most representative terms of each cluster based on the centroids and compare the vector of that to the individual word vectors to get the most similar words to the centroid for each cluster. With that, I can also print out the most representative documents per cluster to check manually how similar those documents are and how well they fit into the cluster based on the most representative words of it.

4.3 Off the self solution

After attempts at using natural language embedding techniques to try to calculate the importance of each word in the job advertisements, I tried to use an off-the-shelf document parser to extract the relevant data I want. To do this I chose to use an application called "Parseur" [8] due to a recommendation from my supervisor as well as the fact that it allows parsing of emails specifically which is where my raw data is sourced from. An immediate and catastrophic issue is that due to the lack of consistency in the structure of the documents, it is impossible to assign a set location where a piece of text that contains relevant information will be. This would force me to create a parsing profile for every unique email which will end up having me essentially extract the data manually from all the documents. This solution would not be able to be applied on a larger scale of data and it defeats the purpose of this study as I'm attempting to automatically extract this data.

4.4 Pre-trained model (GPT)

At this point, I've only tried to train a network from scratch without considering the possibility of using any of the existing pre-trained networks available. Using an already trained network is beneficial in multiple ways such as that I cannot possibly generate a network as large and with such a complex architecture as attention and transformers as the companies in

4 Design and implementation

the industry can produce. Leveraging these networks will not only be able to extract accurate and descriptive data from the emails they are provided to parse, but it is also much more cost and energy efficient to use an already existing model than to train a new one. An example of how much computational power it requires to train one of these larger networks such as the BERT model [9] in 2019 used an equivalent amount of energy as would a trans-American flight would as shown in Emma Strubell's 2019 paper [10]. Comparing BERT's number of network parameters to a newer architecture such as GPT-3, from the company OpenAi, puts into scale how much larger and therefore more computationally and resource intensive the models are becoming as the field of NLP research progresses.

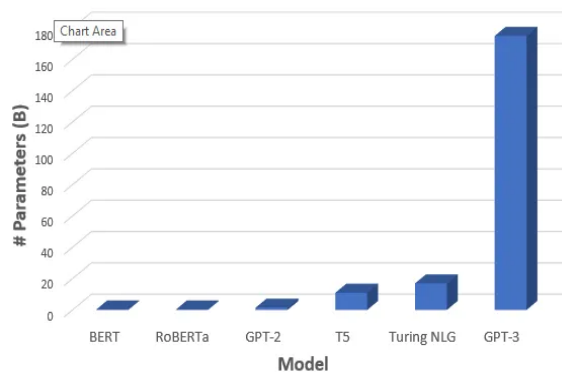


Figure 4.3: Networks size comparison.

Given this information, it's extremely inefficient and will most likely produce much lower quality results if I train a model from scratch to perform the summarising task. Even if I follow the cutting-edge techniques I explained in the introduction, my model won't be nearly as tuned and perform nearly as well as the current state-of-the-art models out there as the effort of one student can never compete with the resources of time and money a large company (OpenAi) backed by an even larger company (Microsoft) has at its disposal.

I chose to utilize the "gpt-3.5-turbo-0301" to summarize the job advertisement emails through the use of the OpenAI API [11] to interact with the model. I decided to pick this model out of the range of models available due to its ease of access and use as a result of the well-documented and intuitive API which I have some experience in using beforehand. I had the option of either using that model or using GPT-4 which accepts multi-modal inputs (images and audio) as well as text but In my use case I would not benefit from this advantage as all I need it to do is text summarization and I would have to bare the higher cost of using that more advanced model. Within the GPT-3.5 family of models, there are multiple different variations of models to choose from such as text-davinci-003 and code-davinci-002 which are optimized for general language tasks and code completion respectively.

4 Design and implementation

The gpt-3.5-turbo model was trained and optimized for chat completion tasks which the text-davinci-003 is also very capable of but the turbo model costs a 1/10th of the price per token, therefore, the most efficient from my perspective. I chose to specifically use the gpt-3.5-turbo-0301 version of the model as this version is a snapshot of the gpt-3.5-turbo from March 1st, 2023 which means that the responses will not vary as much from constant updates to the model itself. Determinism is a feature I value highly from the model as to get the information from the responses I will have to use regular expressions to capture them. These regular expressions will not be as effective to capture the data if the data, in the case of the response from the model, is not formatted consistently. Another variable I am able to set when calling the model's function is temperature. This variable sets the sampling temperature within the model where a higher value makes the model's responses seem more random and allow it to go off on tangents while a lower value will make the response more focused to the question and more deterministic. The code below is how I present the model with the task using a combination of system and user messages.

Listing 4.1: Model API call with prompts

```
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo-0301",
    messages=[
        {"role": "system", "content": "I will present
        you with the content of email messages
        advertising events or positions in the area
        of machine learning and you must extract
        certain information from them and nothing
        more."},
        {"role": "user", "content": "Return job topic
        , duration , institution , supervisor and
        salary from the following job advert email
        formatted as (variable): (information) with
        no extra notes and if there are multiple
        jobs then separate each job's subject's
        information under the header Job(number):
        and if there is missing information about a
        variable then still write the variable
        with n/a as the information" },
        {"role": "user", "content": emailBody}
    ],
    temperature=0,
    max_tokens=450
)
```

4 Design and implementation

The system message instructs the model about what its purpose is and what I expect it to do and the user messages are asking questions with more details and specifications with the last user message presenting the email body text. There are numerous hyper-parameters I could adjust such as frequency-penalty and presence-penalty which both push the model to either increase the likelihood of the model talking about new topics by penalizing new tokens depending on if they've appeared in the response so far or the opposite. I chose not to adjust these parameters as text summarization doesn't benefit from whether or not the model sticks to one topic or multiple.

```
Job 1: Harrison Assistant Professorship
- Job topic: Statistics
- Duration: 3-year fixed-term position
- Institution: Department of Statistics at Warwick
- Supervisor: n/a
- Salary: n/a

Job 2: CRISM Research Fellowship
- Job topic: Statistics
- Duration: 3-year research fellowship
- Institution: Department of Statistics at Warwick
- Supervisor: Warwick colleagues
- Salary: n/a
```

Figure 4.4: Example response from model.

Figure 4.4 shows an example response from the model from one of the emails in the corpus. This email contained two job advertisements and as I instructed the model, it split up each job into its own section with its own variables to be extracted. The model also followed my instruction of filling in a variable's information that isn't present in the advertisement with "n/a". I instructed the model to do so as an attempt to standardize the response format as much as possible to allow the extraction of the information to be as clean and consistent as possible using the regular expressions I will discuss shortly.

4 Design and implementation

Listing 4.2: Regular expressions

```
Topic = "- *Job topic *| - *Job Topic *| - *Topic *|  
Topic *|Job topic *|Job Topic *|Job *[0-9] *|  
Job *\[0-9]\) *:"  
Duration = "- *Duration *| Duration *| - *Job duration  
*|Job duration *:"  
Institution = "- *Institution *| Institution *:"  
Supervisor = "- *Supervisor *| - *Supervisor\s\ *|  
Supervisor *| Supervisor\s\ *:"  
Salary = "- *Salary *| - *Salary/Stipend *| Salary  
*|* Salary/Stipend *:"  
JobNumber = "Job *[0-9] *| Job *\[0-9]\) *:"
```

The above regular expressions I made to extract each variable's information from the response the model returns. From testing the model's responses to select emails multiple times I've found it does not conform to the exact instructions I passed to it. For example, it sometimes returns the Job Topic variable as "Job Topic:", "Job Topic :", "- Job Topic:" or another variation that slightly differs but differs enough to have the regular expression not return clean data. After collecting all the ways the model phrases these variables I designed the above regular expressions that are not sensitive to the number of spaces between the word and the punctuation marks as the model does not seem to be able to have a standard way of phrasing them. This is due to the fact that every email I pass to the model starts a new conversation therefore it does not know what it has returned for the emails prior so it cannot standardise a format. To try to solve this issue I tried to leverage the message object "assistant", which refers to the model, by asking having the same prompts as in the first code block but then after generating one response that I can use as a baseline format I input that response as the ideal response through the "assistant" object and then adding another user message telling the model to use the same format as the one in the assistant's message before with another email that I parse to it in another user message. The issue I found with this solution is that the model's maximum token length is 4097 tokens but as this method involves including two emails and a response message from the model alongside the instruction messages, this exceeded the token limit therefore is not a viable way to fix this problem. This problem doesn't need fixing though as the regular expressions already account for all the different phrasings the model uses for all the required variables.

To be able to visualize the most occurring keywords in the job advertisements I used the YAKE [12] python library to extract the keywords per email as the body of the text of each email may contain multiple advertisements but I'm unable to consistently divide the body into each separate job listing.

Listing 4.3: Keyword extractor initialization

```
yake.KeywordExtractor(lan="en", n=3, dedupLim= 1/(1+  
    math.exp(-1.5*(len(noOfads))+4)), top=len(noOfads)  
    * 5, features=None)
```

I chose the hyper-parameters as shown above with the intention of increasing the allowance of duplicate keywords, through the dedupLim variable, using an adjusted sigmoid function so that as the number of job listings increases so does the allowance for duplicate keywords. I chose to extract 5 keywords, with a maximum length of 3 words for each keyword, per advertisement in each email as I thought that would provide enough headroom to gather all the descriptive keywords in each advertisement without gathering too much useless information. To get clean the keywords that get extracted, meaning to remove the useless keywords, I only plot the keywords that have a significance value of above 0.01 and that get extracted more than a minimum of 4 times over the entire corpora of emails.

To gather the data from the emails I've downloaded I ran the above API calls for every email in the mbox file that met the criteria of having some indication of it being a job advertising email in its title. This filtered the number of emails down from around 10.2k to around 2k emails. This is because a large portion of the emails I've found are calling for papers to be submitted to research conferences and so on.

4.5 Saving the data

There are multiple ways to store the data I collected with each having its own benefits and drawbacks. My options were as follows either a relational database, a NoSQL database, or cloud-based storage. A relational database works well with highly structured data and for large amounts of data that need a long term storage solution. NoSQL databases are ideal if the data doesn't have any structure or is semi-structured and is great in terms of scalability and provides an API to be able to query the dataset easily. Finally, cloud-based storage offers the most flexibility in terms of ease of access and not needing to have the compute power and storage locally to query and store the entire dataset but the main drawback is that for good and reliable cloud storage it would be complicated to set up and maintain as well as it probably being a paid platform to use.

I decided to use mongoDB [13], which is a NoSQL database, to store my data as due to the nature of my data, I would class as it semi-structured and I wouldn't need the relationship functionality that relational databases provide and are built on. As I don't require or really be able to utilize the

4 Design and implementation

flexibility that cloud storage provides which would defeat the purpose as that is its main purpose besides having the storage be off-site as well as the compute power to run the queries over the dataset. I wouldn't fully utilize those functionalities either due to the relatively small size of my dataset which is bottlenecked by the speed of the OpenAI's API calls to the model, the credit allowance I have on my OpenAI account to make the API calls, and the number of emails I have access to download from the ML-news group (you can only download emails from your inbox). Because of this, it doesn't make sense to go through all the effort to set up the cloud environment migrate my data there and learn from scratch how to maintain and query the data on that platform for no added benefit.

To set up and interact with the database I used the MongoDB Compass application which has a simple GUI and features to be able to query the data with any specificity as I need.

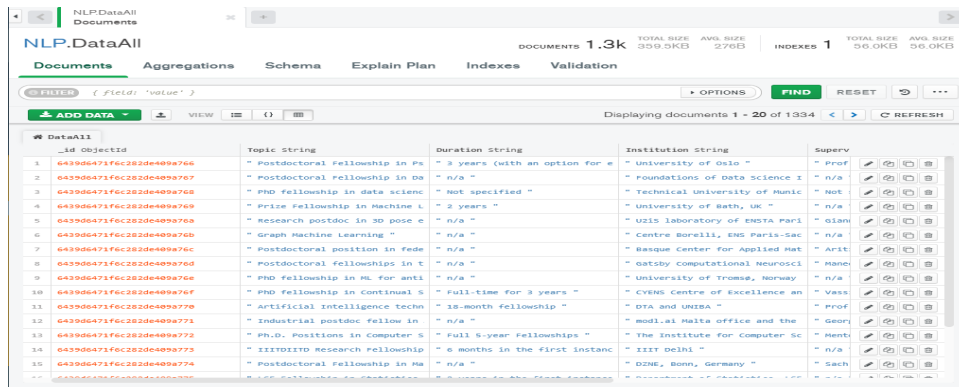


Figure 4.5: MongoDBCompass.

As I mentioned at the beginning of this section, I check each email's title for a sign that it contains a job advertisement so I decided to make a collection per term I search for in the title. This is because I'm working within a tight credit limit with the API calls so as not to exceed it and lose the progress of all the parsed ads till the point it runs out I only parse the emails from one indicating term at a time while checking it doesn't contain a term that I've searched for before so that I avoid the problem of parsing the same email multiple times. To collate all the documents from each collection I save the data in a csv file alongside saving it to the corresponding collection and import that data from all the csv files into a new collection from the MongoDB Compass. I do that to be able to run a single query over my entire dataset instead of fractions of it at a time by querying each collection individually.

4.6 Hierarchical clustering

For the clustering aspect of this project, I chose to represent the clusters as hierarchies due to the nature of the clusters as each cluster is defined as a general topic in the field of machine learning and it contains the jobs that are best characterized by that topic. Each job can only be assigned to one cluster to reduce the complexity of the hierarchy and so that the graph does not get overly full with too much information. I used the same model of NLP (gpt-3.5-turbo-0301) to assign each job description to a cluster. I also used that model to extract all the general topics out of all the job topics first then went through the jobs again to classify each one. This method was chosen as an attempt to normalize the general topics and not variations of the same topics. For example, if I were to try to do the classification on the first go then the model will keep adding new topics that only fit one of the job descriptions therefore it will result in many clusters with only one job in them. By having the topics predefined, the model's job is simpler as it only has to pick one of the provided topics as a classification for each job description. To do the topic extraction I used the prompt: ***"Take the following list of job descriptions from the area of ML and return the general topics that are at the top of the hierarchy:"*** then provided a section of the list of job descriptions as each API call to the model can only support 4096 tokens in total which include the input prompts and output response. This then resulted in a list of 70 general topics in the field of machine learning ranging from very broad terms such as "machine learning" itself to much more specific topics such as "neuroscience" and "robotics".

Taking this list I ran another loop over sections of the list of job descriptions while giving the model the list of topics it should use to classify. I used the prompt "Take the following list of general topics and job descriptions from the area of ML and classify each job into only one of the general topics provided in the format Job description: general topic". This prompt was designed to assign a topic from the list of topics I provide to the model in the same API call to each job topic description and return that information in a normalized manner. The response from this API call, after converting the text into an appropriate format, is a dictionary that has the general topics that were assigned to a job description at least once as the keys and their corresponding jobs as a list as the value. This dictionary then represents the hierarchy of the general topics to job descriptions.

5 Results & Evaluation

5.1 Semantic modelling results

Firstly, the Tfidf attempt yielded some interesting results. Since the calculations are done by a robust Sklearn function, It achieves a 100% parsing rate as it manages to convert all the documents into the matrix of Tfidf features. Despite this achievement, the clustering methods, which are Kmeans using the elbow method to decide the number of clusters and a dendrogram, give different results as shown in figures A.1 and A.2. I believe this is because the elbow method that was used to identify the optimal number of clusters is essentially a heuristic so it is an educated calculated guess at best, therefore, it is very likely to have a different outcome than the clustering done based on the euclidean distances of the vectorized documents themselves.

Taking a look at the clusters of each method there is no obvious relation between the documents that were clustered together besides the fact that they are all in the general field of machine learning which is known beforehand. The features that are calculated to have the highest importance are single-word terms such as "computer", "data", "machine" and "learning". These don't give any insight into the contents of the documents apart from what is already known. This doesn't even provide the answer to the main project question as these terms are too broad to give any significantly detailed insight into what specific topics are talked about the most.

Secondly, the Word2vec model did improve on the Tfidf results by being able to calculate the most representative terms and whole documents per cluster by calculating the centroid's position per cluster and checking which document vectors and term vectors are closest to that location. This improved the quality of results slightly as the representative terms of some clusters give a slight insight into what the contents of the cluster are, for example, cluster 8 contains the terms "social", "projects" and "world" and reviewing the three most representative documents in that cluster there are some references to the term social like "social security" and "social sciences" in two of the three documents. This also achieves a 100% parsing rate due to the same reason as the Tfidf method as this model comes from a reputable library called "Gensim". Calculating the precision and recall

scores for these methods is irrelevant due to the fact that the clusters don't have a defined topic they represent as usually the representative words per cluster are meaningless. Some examples of these words are "please", "number", "email" and so on. These don't give the cluster meaning therefore the "classification" the method does is also meaningless.

5.2 Database of job details

The regular expressions that located and extracted the data from the model's responses managed to get the data of each variable as a single sentence string mostly with no noticeable imperfections. However, it did not work all the time and the data collection run over the emails with only the search term "phd" in their title had a failure rate of 6.5%. This means that an error occurred during the processing of one of the job ads the model returned as there could be multiple job positions being advertised per email. Therefore for most of the emails that an error occurred in I still managed to extract the information about the ads that were processed before the error. These errors were usually the result of the model returning the information about the job it found in the email in the incorrect format or just adding another job after a real job it found and writing "n/a" next to it as shown in figure 5.1.

```
Job(1):  
- Job Topic: PhD positions in the area of "Constructing Explainability"  
- Duration: n/a  
- Institution: Ludwig-Maximilians-University (LMU) Munich, Germany  
- Supervisor: Chair of Artificial Intelligence and Machine Learning  
- Salary: funded by the German Research Foundation (DFG) as part of the Collaborative Research Center (CRC/TRR 318)  
  
Job(2): n/a  
-----
```

Figure 5.1: Failed parsing example.

This isn't an issue though as it still manages to extract the data about job 1 but doesn't add any more information after that as it fails in job 2 which doesn't matter because there is no more information there to be extracted. Even if this wasn't the case and all the emails containing errors didn't return any information I still would've surpassed my parsing rate goal of over 80% by a long margin.

Some problems with the regular expressions were the fact that sometimes the model will include a note about the job listing after presenting the

information I requested figure 5.2 presents an example of this behaviour. This resulted in this note being accounted for as part of the salary variable which isn't a major issue as the salary isn't being treated as a number but rather as a string because there are many cases where the salary isn't included and in a lot of the ones where it is it's not a set number but rather a range therefore it isn't wise to have the salary field as integers.

```
" n/a

Salary: Note: Multiple positions are available.
```

Figure 5.2: Salary field including a note example.

Another imperfection I found after reviewing the data in the database is that there are multiple entries in the database where all the fields only contain "n/a". This is due to the model interpreting the number of places available in a job listing as multiple jobs and presenting that information as multiple job entries with the information about the topic, duration and such only being present in the first job summary it returns. One such response message that showcased this behaviour is shown in figure 5.3.

```
Job(1):
Topic: PhD in wearable computing
Duration: 36 months
Institution: One of five top-level univer:
ly, Romania, or Spain
Supervisor: n/a
Salary: Fully funded for 36-month period

Job(2):
Topic: n/a
Duration: n/a
Institution: n/a
Supervisor: n/a
Salary: n/a

Job(3):
Topic: n/a
Duration: n/a
Institution: n/a
Supervisor: n/a
Salary: n/a

Job(4):
Topic: n/a
Duration: n/a
Institution: n/a
Supervisor: n/a
Salary: n/a

Job(5):
Topic: n/a
Duration: n/a
Institution: n/a
Supervisor: n/a
Salary: n/a
```

Figure 5.3: Response with multiple null jobs.

5 Results & Evaluation

As a way to fix the second issue, I queried the dataset using the following query: **`db.DataAll.find({$and: [{ Topic: {$regex : "n/a"}},{ Duration: {$regex : "n/a"}},{ Institution: {$regex : "n/a"}},{ Supervisor: {$regex : "n/a"}},{ Salary: {$regex : "n/a"}}]})`**. This query located all the entries in the database which contained the string "n/a" in all of their fields. I then took the output of that query and deleted all of those entries in an effort to clean the dataset. The percentage of records that matched the query and therefore deleted was merely 3% which means there weren't that many null entries to begin with but removing them still results in a higher quality dataset. The total number of records after this removal of the null entries totalled 1284 records with most of the records including information about the topic of the job it represents as well as the institution offering that position. Around a third of the records were missing information about the supervisor, almost half not including the duration of the position and around two-thirds omitting salary information as visualised in the graph below.

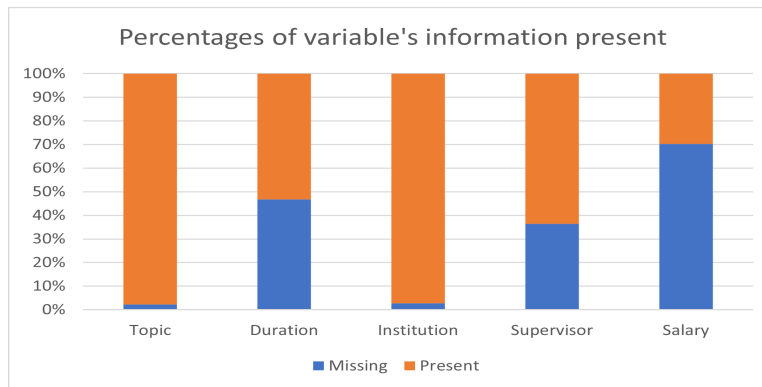


Figure 5.4: Missing information percentages.

One of my goals for this project was to get the answers to certain queries about jobs in the machine learning field that describe their state such as which university is offering the most positions currently for example. Having all the data in the mongoDB database makes it possible to easily construct a pipeline of instructions to get the answer to these queries. To get the answer to the example query I just mentioned I would need to use a pipeline such as this: **`db.collection.aggregate([{$group: { _id: "$Institution", count: { $sum: 1 } } },{$sort: { count: -1 }},{ $limit: 1 }])`**. This will return a list of documents in format { "_id" : "institution name", "count" : 999 } ordered from most occurring to least which gives the user the information they asked for. After running this query in the database, the most occurring institutions were: ADAPT Centre, University of Oslo, Edge Hill University and CYENS Centre of Excellence. The user can now take this information and search and filter the database using each institution to browse what jobs each offer if they cared to know more details about them.

5.3 Keyword occurrences

The main goal of this project was to find the topics in the field of machine learning that are attracting the most interest currently. I achieved this goal by using keyword extraction and counting the most occurring keywords which are the topics of each job. Extracting the keywords from the body of each email resulted in the topic counts shown in figure 5.5.

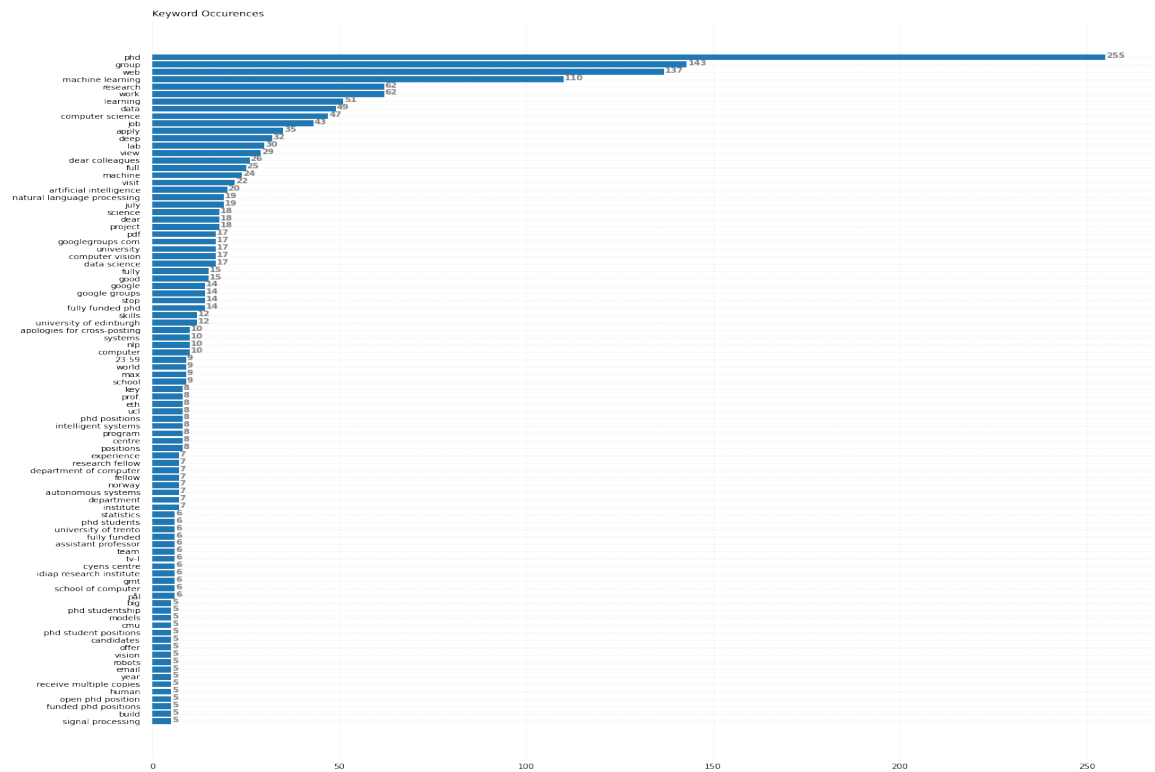


Figure 5.5: Keyword occurrences counts from email body.

Examining these results brings light to a couple of issues with this implementation. Firstly, most of the keywords are not actually representative of any job position with keywords such as "group", which is the second most occurring keyword, and "dear colleagues". Secondly, many of the extracted keywords have a very low statistical significance value (< 0.0001). This issue could've been a result of trying to extract too many keywords from the text which didn't contain that many keywords to be extracted in the first place which I believe to be the cause of this problem. Luckily these issues have easy fixes, for the first problem it can be mostly resolved by trimming the beginning and end of the body which contain these repeated terms therefore not having them be extracted, and for the second issue, I can simply decrease the number of keywords I attempt to extract from each email.

5 Results & Evaluation

Instead of implementing these fixes to try to get a better result, I opted to change the text from which I'm extracting the keywords from. Rather than extracting the keywords from the body of the email, I parsed the "Topic" variable of each job advertisement to the keyword extractor. I chose to implement this change because the topic description is a richer text in terms of topic keywords as well as being much shorter than the email body which allows the keyword extractor to be able to assign higher significance values to the words that are most representative of the topic as they are the main focus of the topic description. Having implemented this change I also needed to change the hyper-parameters of the extractor itself. I tested different combinations of n-gram lengths, numbers of keywords to be extracted and duplication tolerances and concluded that 3, 0.1 and 2 are the most optimal numbers for those parameters respectively with the second-best results coming from mostly the same parameters but with 2 n-gram length instead of 3 as shown in figure A.3 (in the appendix).

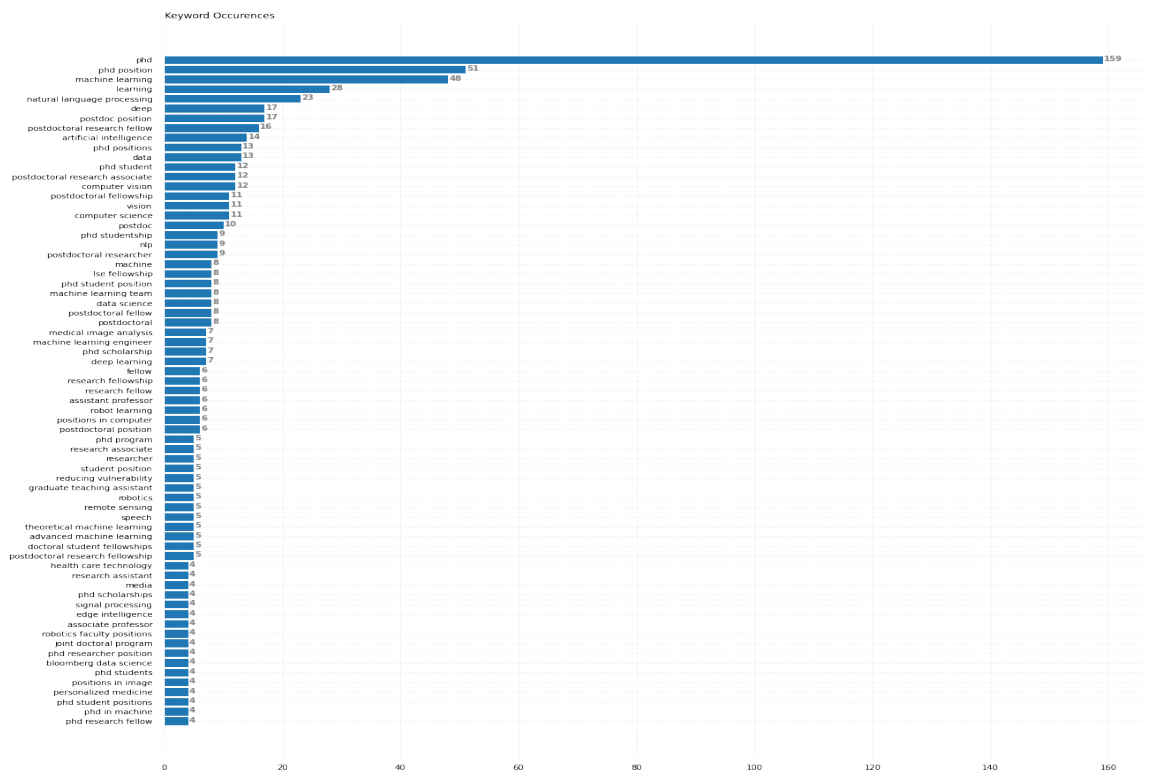


Figure 5.6: Keyword occurrences counts from topic (n-gram = 3).

Analyzing this graph gives some insight into what the corpus of jobs looks like in terms of topics. As you can see the most and second most common keywords (keywords can be phrases of however long the n-gram parameter was set to) are about PhDs. This makes sense as most of the positions in the ML-news group are research-based so are ideal for a PhD thesis. Next, machine learning and learning take the 3rd and 4th positions in the list which also makes sense given the corpus is from a machine learning

Google group. The answer to the main question behind this project, which is what is the topic in machine learning that has the most interest currently, turns out to be natural language processing with 23 occurrences over the topic descriptions. This is further increased as looking a bit further down the list the abbreviation "nlp" is present with 9 occurrences which makes it a total of 32.

5.4 Topic hierarchy and clustering

Finally, after extracting the general topics from the topic descriptions of each job advertisement and providing the model back with the entire list to classify the jobs into these topics this is the tree graph, with the general topics at the top and jobs at the bottom, produced with the results of that classification. The full expanded hierarchy list with the names and count of jobs classified to that specific topic is presented in listing A.1 in the appendix. You may notice that the number of topics in the listing far exceeds the 70 topics I extracted first. The model couldn't avoid falling into the issue I tried to prevent by splitting the topic extraction and classification as it still decided to either alter or add names of topics to the list I provide during the classification.

Topic Tree

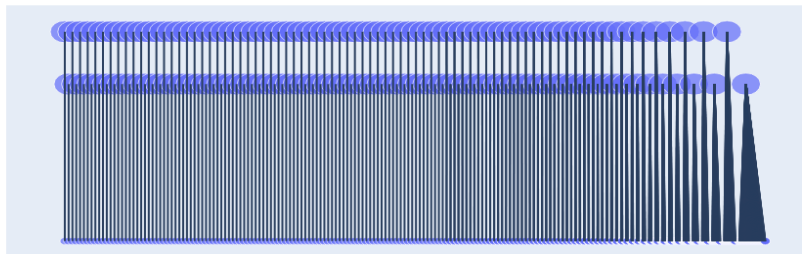


Figure 5.7: Topic hierarchy.

As you can see many topics only link to one or two jobs. This was due to the fact that when the model was extracting the topics it had no awareness of what the previous topics that it extracted were from the last sections of the job list. So it added slight variations to the topics and sometimes combined topics, for example, it combined the machine learning topic with multiple others to create general topics that were two topics which were based on a job topic description that included those two topics hence only having that job in that topic cluster. To analyze the clusters further I zoomed into the end with higher counts per cluster. Figure 5.8 shows the highest

5 Results & Evaluation

cluster counts with figures A.4 and A.5 showing the clusters with lower counts but still a considerable amount.

Topic Tree



Figure 5.8: Topic hierarchy (Highest counts).

As the keyword counts showed, machine learning is the highest count topic with 179 jobs falling under it. In hindsight, I should've added a restriction in the prompt I used to instruct the model doing the topic extraction to not include machine learning as a topic as I already know that all the jobs are in the machine learning field already, therefore, having a machine learning cluster doesn't give us any more useful information about what the topic of the jobs in that cluster is. Contrary to the keywords counts graph, natural language processing did not even make it into the top 10 largest topic clusters. After closer inspection of the clusters, I noticed that multiple jobs contain mentions of NLP in their topic descriptions but that isn't the main focus of the job therefore the model didn't classify them in the NLP cluster as they fit better in other more general topic clusters such as the computer science and artificial intelligence clusters. This then decreases the count of jobs that are mainly focused on natural language processing hence decreasing the cluster count and making it seem like that topic doesn't have as much interest as the others. To read this graph more accurately, the reader needs to understand that these high-count clusters will intuitively be the more general topics and not necessarily be disregarded but should not be compared when concluding how popular a topic is.

Despite that, the robotics cluster comes in as the 3rd highest count cluster in the graph, which is interesting because the keyword counts only had 5 occurrences. To test the homogeneity and completeness of these clusters I used the v-measure score to calculate the average between these two measures. This calculation requires a ground truth of the clusters to compare to therefore I had to create the "true clusters". In my case, there will never be a definitive ground truth as a job description can be classified into multiple clusters if I chose to have the clusters more general rather than specific. As a workaround, I sampled some job descriptions from each cluster that contains 5 or more jobs and shuffled them around, and

presented them with 4 randomly chosen topics as well as the topic the model classified the job as to a couple of unbiased third parties and had them choose which topic they would classify that job into. I chose to only present 5 topics in total to avoid overwhelming the person classifying as a human's short-term memory can only contain 7 things (plus or minus 2) at a time [14]. This gave me some interpretations of what the ground truth should be and used them all and calculated the average V measure over all of them. This average was calculated to be 0.83, with the highest score achievable being 1. This result means that the clusters are both relatively homogeneous and complete. I followed the testing method outlined in section 3.2 of this report to calculate the precision and recall scores for these clusters. Going over 5 emails manually and cross-examining the topics I extracted versus the topics the model extracted I calculated the precision score to be the number of true positives (5) over the number of all extracted topics (6) to total a score of 0.83. Recall, calculated as the number of true positives (5) over the sum of the true positives and false negatives (5), to total a perfect score of 1 as it managed to extract all the relevant topics from all the jobs advertised in the emails.

Going off of those results, I can answer the main project question using either the clustering method or the keyword occurrences method as the clusters were proven to be meaningful and qualitatively "good" in terms of homogeneity and completeness and the keywords cannot be incorrect as those words and phrases must've appeared at least as many times as shown in the graph therefore is an accurate measure of popularity as the counts of them cannot be any less than what is shown.

6 Conclusion & Further Work

6.1 Summary

The model GPT-3.5 outperformed all other embedding techniques in being able to correctly identify important phrases and sentences in the emails that relate to the topic of the jobs they advertise. This is largely due to the fact that this model utilizes much more advanced techniques and architectures than just calculating importance based on word frequency or the primitive embedding that the Word2vec model does. Achieving a parsing rate of well over the desired 80% as well as near-perfect precision and recall scores, this method of automatically extracting topics from emails worked exceptionally well and also managed to extract meta-information such as the institution behind the jobs about the jobs in each email when they were present. This not only allowed the main project question to be answered but also provided more information about the jobs in that topic so that the user has a better idea of what the job is.

Overall, this project is a success and could be utilized by researchers in finding new jobs as well as just exploring their field to figure out what topics interest the industry most at the moment. The information in the database, the keyword occurrences graphs, and the topic hierarchy I managed to create will undoubtedly inform the user about the state of the research in their fields, therefore, helping them in making informed decisions about their career path moving forward based on the information.

6.2 Further work

Improving on this work can be done in multiple ways. One such aspect that can be improved on is the prompts used to instruct the model on what it should do with the emails provided. Designing a more detailed and concise prompt will increase the quality of the responses and decrease the cost per API call as the prompt tokens count into the total token count of the API call. Another possible change that might result in improved responses is changing the actual model used to a newer model made by

6 Conclusion & Further Work

the same or different company. At the time of writing this paper, multiple other NLP models exist by numerous leading companies in the industry such as Google with Bard, Nvidia's Megatron-LM [15], and Meta's OPT-IML [16]. Using another available model that has a much larger size and therefore capabilities than the GPT-3.5 I used for this project might allow more information to be extracted from the emails which will allow the user to have access to more information giving further insight into the job market.

This project was mainly focused on whether or not it's possible to automate the extraction of jobs from emails and to what extent the quality of the information extracted is. Some sort of web interface to this data could be developed to allow a user to possibly upload their own corpus of emails to extract the data from and display all the resulting information in a clear and intuitive manner. That will allow this work to be continuously relevant as the data can be updated whenever the user pleases as well as adding to the potential for this work to act like a job-searching platform.

A Appendix

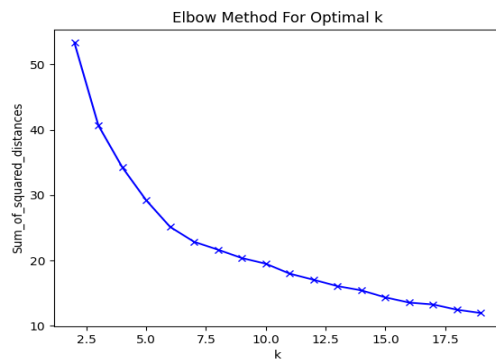


Figure A.1: Optimal number of clusters (elbow method).

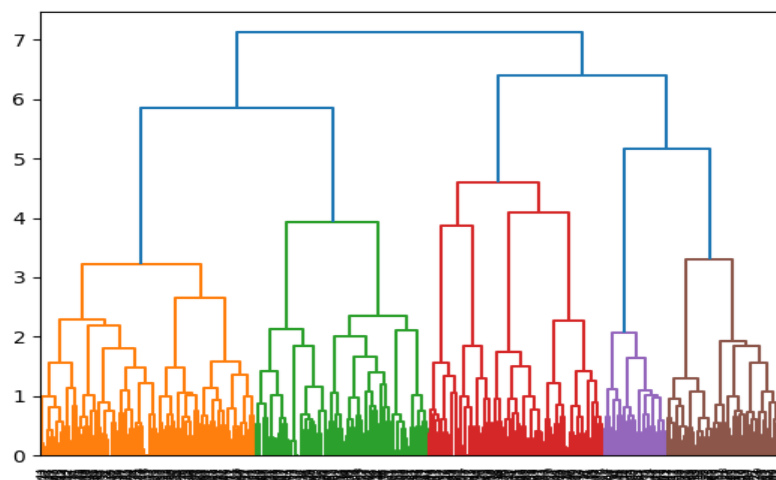


Figure A.2: Dendrogram hierarchy.

A Appendix

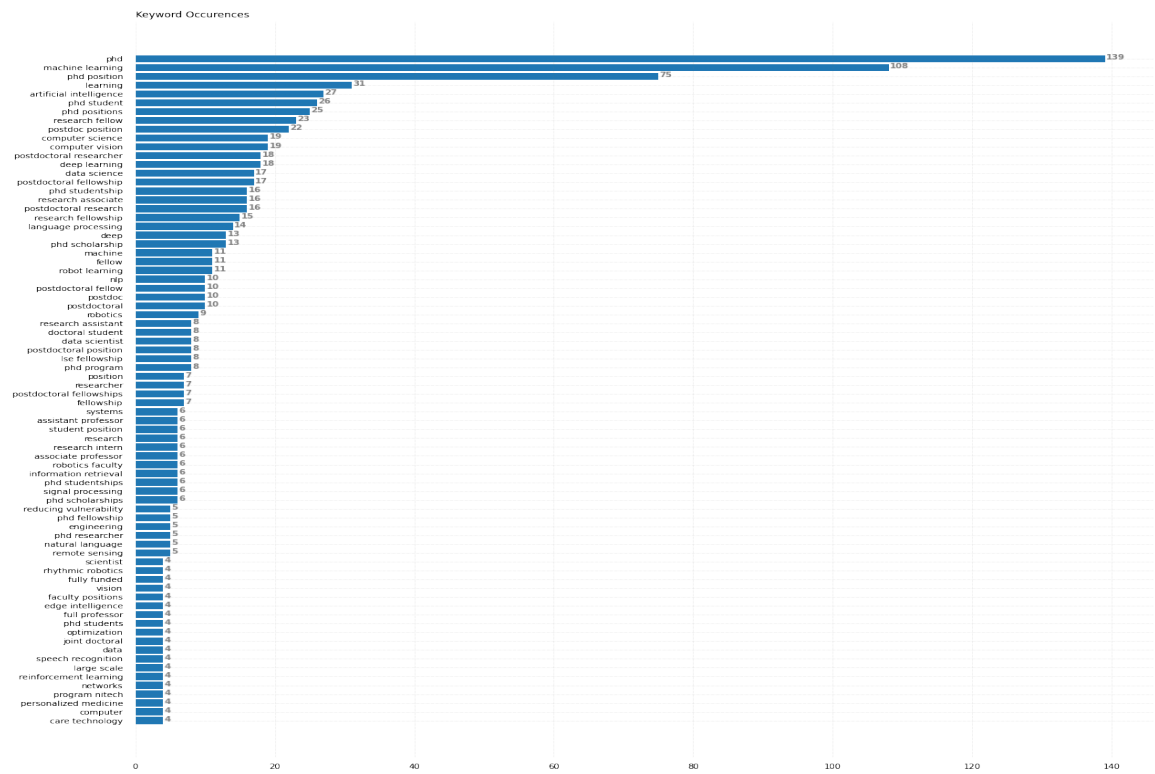


Figure A.3: Keyword occurrences counts from topic (n-gram = 2).

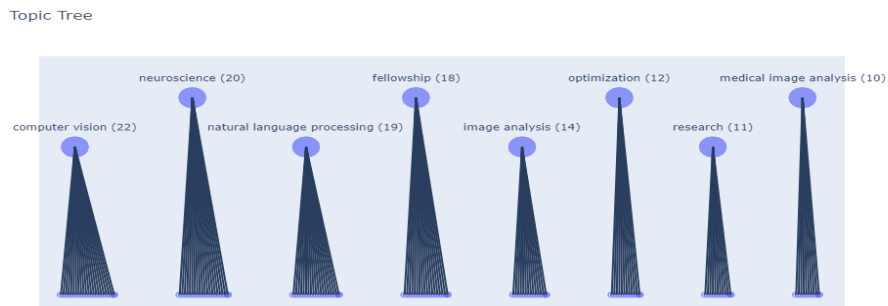


Figure A.4: Topic hierarchy (Medium counts).

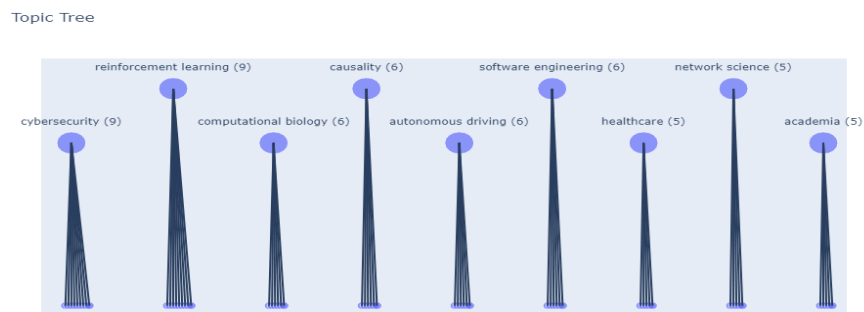


Figure A.5: Topic hierarchy (Lower counts).

Listing A.1: Topic hierarchy

- Artificial Intelligence (78)
 - Machine Learning (179)
 - Deep Learning (27)
 - Reinforcement Learning (9)
 - Fewshot Learning (1)
 - Data-efficient Machine Learning (1)
 - Advanced Machine Learning (4)
 - Federated Learning (1)
 - Self-supervised Learning (1)
 - Gaussian Processes (1)
 - Neural Networks (2)
 - Bayesian Methods (1)
 - Machine Learning and Evolutionary Models (1)
 - Machine Learning and Radiological Waste Characterization (1)
 - Machine Learning and Remote Sensing (1)
 - Machine Learning and Deep Learning (2)
 - Visualization (3)
 - Machine Learning and Theoretical Neuroscience (1)
 - Reinforcement Learning and Embodied AI (1)
 - Reinforcement Learning and Healthcare (1)
 - Machine Learning and Medicine (1)
 - Clinical Informatics (1)
 - Responsible and Transparent AI (1)
 - Artificial Intelligence and Healthcare (1)
 - Artificial Intelligence and Machine Learning (5)
 - Fair Learning (1)
 - Knowledge Graphs and Machine Learning (1)
 - Graphs and Machine Learning (1)
 - Wavelets and Machine Learning (1)
 - Conversational Search and Recommendation (1)
 - Conversational Agents (1)
 - User Modelling and Monitoring in Social Service Robotics (1)
 - Neurosymbolic Modelling (1)
 - Environmental Modelling (2)
 - Audiovisual Speech Enhancement (1)
 - Multimodal Sensing (1)
 - XR Representations (1)
 - Natural Language Processing (19)
 - Haptic Technology (1)
 - Explainable AI (3)

A Appendix

- - - Continual Learning (2)
- - - Social Media (2)
- - - Robotics (56)
 - - - - Social and Cognitive Robotics (1)
 - - - - Autonomous Driving (6)
 - - - - Human-Robot Interaction (3)
 - - - - Machine Learning and Robotics (2)
 - - - - Medical Images and Machine Learning (1)
 - - - - Robotics and Control (1)
 - - - - Sensor Integration (1)
 - - - - Assistive Technology (1)
 - - - - Autonomous Systems (2)
- - - Computer Vision (22)
 - - - - Image Analysis (14)
 - - - - Image Processing (3)
 - - - - Medical Image Analysis (10)
 - - - - Vision and Language (1)
 - - - - Anomaly Detection (2)
 - - - - Remote Sensing (4)
 - - - - Optics (2)
 - - - - Computer Graphics (2)
 - - - - 3D Modelling (1)
 - - - - Image Analysis and Optimization (1)
- - - Data Science (45)
 - - - - Data Mining (1)
 - - - - Data Management (1)
 - - - - Data Engineering (1)
 - - - - Information Retrieval (2)
 - - - - Information Systems (1)
 - - - - Crowdsourcing (1)
 - - - - Knowledge Representation (1)
 - - - - Knowledge Graphs (1)
 - - - - Data Privacy (2)
 - - - - Data Analysis (1)
 - - - - Time Series Modelling (1)
- - - Statistics (23)
 - - - - Statistics and Computer Science (1)
 - - - - Biostatistics (3)
 - - - - Causal Inference (3)
 - - - - Fairness and Transparency (4)
 - - - - Sentiment Analysis (1)
 - - - - Statistics and Machine Learning (1)
- - - Optimization (12)
 - - - - High-Performance Computing (3)
 - - - - Computing and Society (4)

A Appendix

- - - - Leadership (2)
- - - - Decision Making (1)
- - - - Emerging Technologies (1)
- - - - Full Stack Development (1)
- - - Computational Biology (6)
- - - - Medical AI (1)
- - - - Healthcare (5)
- - - - Disease Ecology (1)
- - - - Health (3)
- - - - Human Mobility (1)
- - - - Time Series Modelling (1)
- - - - Bioengineering (4)
- - - - Clinical Informatics (1)
- - - - Computational Neuroscience (2)
- - - - Neuroscience (20)
- - - - Machine Learning and Medicine (1)
- - - - Computational Biology and Machine Learning (1)
- - - Cybersecurity (9)
- - - - Security (2)
- - - - Privacy (2)
- - - - Ethics in AI (2)
- - - - Legal Implications (1)
- - - Computer Science (37)
- - - - Software Engineering (6)
- - - - Engineering (2)
- - - - Networking (5)
- - - - Algorithms (1)
- - - - Mathematics (2)
- - - - Physics (1)
- - - - Logic (1)
- - - - Interdisciplinary (1)
- - - - Systems (1)
- - - - Education (1)
- - - - Humanities (1)
- - - - Mechatronics (1)
- - - - Informatics (1)
- - - - Computational Materials Science (1)
- - - - Atmospheric Science (1)
- - - - Sustainability (1)
- - - - Mentorship Program (1)
- - - - Internship (1)
- - - - Career Development (1)
- - - - Academic Positions (5)
- - - - Fellowship (18)

A Appendix

- - - - Research (11)
- - - - Web Conference (1)
- - - - Doctoral Networks Project Management (1)
- - - - Causality (6)

Bibliography

- [1] M. Aßenmacher, *Multimodal deep learning*, 2023.
- [2] T. Mikolov, *Efficient estimation of word representations in vector space*, 2013.
- [3] M. A. J. P. A. Carpenter, *A theory of reading: From eye fixations to comprehension*, 1980.
- [4] D. Bahdanau, K. Cho and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL].
- [5] A. Vaswani, N. Shazeer, N. Parmar *et al.*, ‘Attention is all you need,’ *Advances in neural information processing systems*, vol. 30, 2017.
- [6] T. B. Brown, B. Mann, N. Ryder *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].
- [7] OpenAI, *Gpt-4 technical report*, 2023. arXiv: 2303.08774 [cs.CL].
- [8] *Parseur*, <https://parseur.com/>, Accessed: 19/01/2023.
- [9] J. Devlin, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019.
- [10] E. Strubell, *Energy and policy considerations for deep learning in nlp*, 2019.
- [11] G. Brockman, M. Murati, P. Welinder and OpenAI, *Openai api*, 2020. [Online]. Available: <https://openai.com/blog/openai-api>.
- [12] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes and A. Jatowt, ‘Yake! keyword extraction from single documents using multiple local features,’ *Information Sciences*, vol. 509, pp. 257–289, 2020, ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2019.09.013>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020025519308588>.
- [13] *Mongodb*, <https://www.mongodb.com/>, Accessed: 15/04/2023.
- [14] G. A. Miller, *The magical number seven, plus or minus two: Some limits on our capacity for processing information*, 1956. [Online]. Available: <http://psychclassics.yorku.ca/Miller/#f1>.
- [15] M. Shoenybi, M. Patwary, R. Puri, P. LeGresley, J. Casper and B. Catanzaro, *Megatron-lm: Training multi-billion parameter language models using model parallelism*, 2020. arXiv: 1909.08053 [cs.CL].

Bibliography

- [16] S. Iyer, X. V. Lin, R. Pasunuru *et al.*, *Opt-impl: Scaling language model instruction meta learning through the lens of generalization*, 2023. arXiv: 2212.12017 [cs.CL].