# An Exploration of Representation Learning and Sequential Modeling Approaches for Supervised Topic Classification in Job Advertisements

**Clemens Westrup**

**School of Science**

Thesis submitted for examination for the degree of Master of Science in Technology.
Helsinki 04.10.2016

**Thesis supervisor:**

Prof. Aristides Gionis

**Thesis advisor:**

Ph.D. Michael Mathioudakis

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Clemens Westrup

Title: An Exploration of Representation Learning and Sequential Modeling Approaches for Supervised Topic Classification in Job Advertisements

Date: 04.10.2016 Language: English Number of pages: 5+99

Department of Information and Computer Science

Professorship: Machine Learning, Data Mining, and Probabilistic Modeling

Supervisor: Prof. Aristides Gionis

Advisor: Ph.D. Michael Mathioudakis

This thesis applies the explorative double diamond design process borrowed to iteratively frame a research problem applicable in the context of a recruitment web service and then find the best approach to solve it. Thereby the problem focus is laid on multi-class classification, in particular the task of labelling sentences in job advertisements with one of six topics which were found to be covered in every typical job description. A dataset is obtained for evaluation and conventional N-Gram Vector Space models are compared with Representation Learning approaches, notably continuous distributed representations, and Sequential Modeling techniques using Recurrent Neural Networks. Results of the experiments show that the Representation Learning and Sequential Modeling approaches perform on par or better than traditional feature engineering methods and show a promising direction in and beyond research in Computational Linguistics and Natural Language Processing.

Keywords: Natural Language Processing, Computational Linguistics, Representation Learning, Sequential Text Modeling, Text Classification, Job Advertisements

# Preface

This work has been both very challenging and interesting. I learned a huge amount of new things and deepened my knowledge in others. It concludes my studies at Aalto University during which I grew beyond what I not have imagined when I came to Finland. So thank you Aalto for this amazing place and environment that inspired me to pursue a whole Master's degree here, after all I had just planned to stay for a one year exchange. A huge thank you also goes out to my second home Aalto Design Factory and the awesome community which is like a family to me now. Studying and teaching product development broadened my horizon and changed my way of thinking which obviously has also had an impact on this thesis.

I want to thank Aris for providing me with the opportunity to work on this topic, all the incredible support and trust, and Michael for all the discussions, the helpful feedback keeping me on a good path and his patience. I also want to thank Juho for the year of research work which actually taught me how to do research and was an invaluable lesson to keep pursuing, despite things not working out the way you thought more often than they do in reality. And special thanks go out to the people at Sanoma for all the support and inspiration and kindness, especially Mikko, Mika, Sami and Hugo.

Lastly I want to thank my amazing friends here in Finland, back home in Germany as well as all over the world and my beloved family.

Helsinki, 04.10.2016

Clemens Westrup

# Contents

# 1 Introduction

Language is one of the most complex behaviors our species has developed. We humans use it to efficiently transport knowledge between individuals and communicate even the most abstract concepts with it. It takes children years to learn to learn express their thoughts and the subtle nuances of one's language give a glimpse one's cultural environment and upbringing, one's emotional state and one's intellect.

Not surprisingly in the field of Artificial intelligence (AI) and especially Machine Learning (ML) building computer systems with linguistic capabilities and solving language-based problems poses one of the hardest challenges and has motivated decades of research in Computational Linguistics. In fact many of the famous test for universal machine intelligence are based on linguistic capabilities, among them the famous *Turing test* by [Turing, 1950] where the task is for a human judge to determine whether he is having a conversation with a human or a machine in order to determine if the machine can be called intelligent, or the *compression test* proposed by [Mahoney, 1999], where a human's and machine's capability to predict missing words given a context is tested.

This thesis explores the specific task of predicting the semantic structure of job advertisements as a specific example of such a language-based task that turns out to be difficult even for humans to do.

The work was done in close collaboration with the Helsinki-based media and learning company *Sanoma*[1] and the research motivation was thus constantly tied back into real world challenges in the scope of Sanoma's business needs.

## 1.1 Problem Statement

The problem addressed during with this thesis to better understand the structure of job advertisements. In particular job postings typically consist of several parts with a certain function or theme: Usually the company is introduced, the job is described with it's tasks and responsibilities, the requirements for the job are listed, then benefits and offerings by the company are named and the reader is asked to apply in a specified way he or she is interested. Almost all of the text of a job description falls into these categories[2] and the task can thus be posed as predicting a category for each sentence in a job advertisement, that corresponds with this sentence belonging to one of the job ads' parts as described above. This is a challenging problem in itself but can further be used to extract certain functional parts of each job ad, to study a possible correlation between structural patterns and the reach and success

---

[1] "Sanoma is a front running consumer media and learning company in Europe. In Finland and the Netherlands we are the market leading media company with a broad presence across multiple platforms. In Belgium we are among the Top 5. Our main markets in learning are Belgium, Finland, the Netherlands, Poland and Sweden. We entertain, inform, educate and inspire millions of people every day. We employ some 7,500 professional employees operating in Europe.", Source: `http://www.sanoma.com/en/who-we-are`, visited 06.06.2016

[2] Only about 4% of the sentences collected for evaluating the final experiments in this thesis were sorted into the category *other* while the rest falls into either of the categories described. This is described in more detail in Section 2.2.3

of an ad and so forth. The problem therefore can be labelled as *Text Categorization* or *Text Classification* as referred to in the scientific literature.

## 1.2 Need Statement and Motivation

Today's media and education, the basis of Sanoma's core businesses, are undergoing drastic and fundamental transformations that are currently disrupting whole industries. Usage of digital media as a source of information has long surpassed print media. Sanoma's most well-known product, Finland's biggest daily newspaper *Helsingin Sanomat*, lost 6% of its circulation only in 2015[3], while the wide-spread use of social media challenges traditional ways we access information. Similarly in the field of education, with the rise of Massive open online course (MOOCs), traditional learning settings are challenged and the need for advanced techniques for data processing and analysis increases, e.g. to personalize and adapt the learning experience to each individual user and at the same time identify trends across large groups of learners to better meet the needs of education.

Sanoma provides a recruitment platform named *Oikotie Työpaikat*. The service is in direct competition several other international players in the recruitment industry. Through this and other services Sanoma's collects large amounts of user-generated data, offering the potential to be leveraged for machine learning solutions to provide value for their users and innovate and enrich the company's offerings. This was the company's initial motivation for this thesis project — To explore ways to leverage user-generated data to potentially.

From a personal perspective this work was interesting as offered many research possibilities while at the same time being relevant for and inspired by real life applications. The complex nature of Computational Linguistics with its proximity to the general study of machine intelligence and the high interpretability of problems makes poses makes it a fascinating problem domain. This presented a great challenge to learn balancing scientific research objectives and yet exploring potential business and user needs, learn on new fronts and deepen the knowledge in others and apply it to real data.

## 1.3 Related work

The problem of Text Classification (TC), also known as Text Categorization, is one of the various challenges in the thriving fields of Computational Linguistics (CL) and Natural Language Processing (NLP). These areas of research have started out as theoretically challenging but rather marginally popular fields between Artificial intelligence (AI) and formal linguistics and have since exploded in terms of interest with their applications being deployed currently at large scale to practically all kinds of consumer products such as smart phones and home entertainment systems as well as digital services like social networks, automatic translation engines and conversational agents. Today there are textbooks dedicated specifically to this area, such as [Manning and Schütze, 1999], [Jurafsky and Martin, 2014]

---

[3] Source: http://www.digitalnewsreport.org/survey/2016/finland-2016/, visited 27.07.2016

and [Clark et al., 2013] There is also much overlap to the field of Information Retrieval (IR) that rose to popularity during the late 1980's due to the Internet starting to become a mainstream medium (see [Manning et al., 2008], [Leskovec et al., 2014] and the classic work by [Rijsbergen, 1979]).

### Syntactics, Semantics, and Pragmatics

The field of Natural Language Processing has undergone different trends of modeling approaches to tackle its challenges. [Cambria and White, 2014] identify these as three main trend curves focusing on *Syntactics*, *Semantics*, and *Pragmatics*. According to the authors the first trend of syntax-centered NLP is still prevalent and is based on algorithms that more or less directly operate on the words found in the processed texts. The second trend operates on the semantics of text, thus being able to potentially tackle more challenging problems by addressing increasingly subtle notions of meaning and context. According to the author these types of approaches are still in the early phase but at the verge of being adapted by a broader audience of researchers and practitioners in the field. The last trend curve of pragmatics regards the yet more complex issue of modeling inherent narratives in language where so far only pioneering work has been done but which, according to the authors, will lead to tackling problems of natural language understanding.

### The Text Classification Problem

The problem of Text Classification (TC) specifically has seen many evolving fashions of approaches and more or less loosely follows the three trend curves described above. TC has been of immense interest for several decades now due to the exponentially increasing amounts of text data being recorded in forms of e.g. user generated content through social networks and through the increased digitalization of our daily lives. Its applications reach from document filtering, automated metadata generation such as language classification to automatic email labeling, spam identification and sentiment detection, amongst others.

Text Classification is nowadays covered in standard works of Information Retrieval and Machine Learning, such as [Manning et al., 2008]. As [Sebastiani, 2002] points out it is important to mention that the term *automatic text classification* has also been used referring to different problems: "Aside from (i) the automatic assignment of documents to a predefined set of categories [. . . ], the term has also been used to mean (ii) the automatic identification of such a set of categories (e.g., [Borko and Bernick, 1963]), or (iii) the automatic identification of such a set of categories and the grouping of documents under them (e.g., [Merkl, 1998]), a task usually called text clustering, or (iv) any activity of placing text items into groups, a task that has thus both TC and text clustering as particular instances [Manning and Schütze, 1999]"

### Early Work and Syntactic Approaches

Early approaches towards solving Text Classification tasks in the 1980's were in often based on expert systems consisting of sets of manually created logical rules, deciding upon the category of a text segment if a certain formula applies. As discussed by [Sebastiani, 2002] the biggest downside is known as the *knowledge acquisition bottleneck* which refers to the fact that each rule has to be manually created. In the 1980's machine Machine Learning (ML) became more common where a *classifier* automatically learns the attributes of the data and its association with the given data labels using a model that allows to then infer the categories for unseen data. This setting is called Supervised Learning as labels for the data are given and predicted.

ML approaches have since been developed in countless variations. As described earlier syntax-based approaches are still very common and most follow a classic pipeline of *feature extraction* or *indexing* of documents followed by *inductive construction* of a classifier that is lastly evaluated by a measure of effectiveness. A very successful approach to feature extraction has been the introduction of N-Gram based models, also called *bag-of-words* models, which that are based on word-cooccurrence frequencies of the terms that are present in the data (see Section 4.1.1 for an introduction). As [Mikolov, 2012] explains "the most significant advantages of models based on n-gram statistics are speed (probabilities of n-grams are stored in precomputed tables), reliability coming from simplicity, and generality (models can be applied to any domain or language effortlessly, as long as there exists some training data). N-gram models are today still considered as state of the art not because there are no better techniques, but because those better techniques are computationally much more complex, and provide just marginal improvements, not critical for success of given application." The limitation of such models is that their statistics are directly based on word co-occurrences and thus exponentially increase in size as the desired context to be captured is expanded, making them infeasible to adapt to longer text sequences.

### Semantic Approaches and Representation Learning

Recently approaches from the field of *Representation Learning*, have found their way into and gained tremendous traction in the NLP research community. Traditional techniques such as N-gram models, now also referred to as Feature Engineering approaches, use prior domain or expert knowledge in order to build data representations that are effective in combination with a classifier. But as [Bengio et al., 2013] point out this is a laborious and time-consuming task that only exposes the weakness from a Machine Learning point of view by not automatically learning such representations — a challenge representation learning aims to address. Beyond that, with regards to Computational Linguistics (CL) problems, most Feature Engineering techniques do not capture well the semantics of language.

One example particularly popular method called *word2vec* was introduced by [Mikolov et al., 2013a] and learns word representation vectors, so-called *word embeddings*, through a context-prediction task. This produces extremely rich representations capturing many subtleties in the layers of meaning of words[Mikolov et al., 2013b].

Another approach related to Representation Learning under the name of Transfer Learning has been proposed by [Do and Ng, 2006]. Similarly, in attempts to find more expressive ways of modeling semantics of language several breakthrough achievements in NLP have been made using various Deep Learning based methods — a related trend that the field has picked up. [Collobert et al., 2011] succesfully applied a Convolutional Neural Network (CNN) architecture to a number of standard problems in NLP with the intention of minimizing the domain knowledge introduced: “The design of this system was determined by our desire to avoid task-specific engineering as much as possible. Instead we rely on very large unlabeled datasets and let the training algorithm discover internal representations that prove useful for all the tasks of interest.”. Later [Kim, 2014] improved the method slightly, improving on state-of-the-art results in 4 out of 7 of these standard problems. Numerous related work was done building on similar ideas, such as using a CNN on character level resolution for TC [Zhang et al., 2015], Multitask Learning and Semi-Supervised Learning to improve the generalization of the shared tasks[Collobert and Weston, 2008] and using Recurrent Neural Networks (RNNs) [Liu et al., 2016]. Just recently facebook research released a tool called *fastText*[4] for TC with a focus fast run-time while still achieving close to state-of-the-art results. The related publications draw heavily on previous work on word embeddings and similar techniques (see [Joulin et al., 2016] and [Bojanowski et al., 2016]).

### Further Related Work and Focus of This Thesis

Numerous other classes of approaches exist and have been explored with varying popularity such as TC using String Kernels [Lodhi et al., 2002], Expectation Maximization (EM) ([Nigam et al., 2000] and [McCallum, 1999]), Self Organizing Map (SOM) [Merkl, 1998] and using a Conditional Random Field (CRF) for sequence labeing [Lafferty et al., 2001]. Also problem areas from the field of Unsupervised Learning are related with techniques such as Latent Dirichlet Allocation (LDA) [Blei et al., 2003] for Topic Modeling where labels for text documents are found without providing a Ground Truth beforehand. However this work specifically focuses on comparing traditional syntactic approaches to more recent semantic ones, focusing on ideas from Representation Learning and Deep Learning, in particular RNNs.

## 1.4 Research Scope and Objectives

This thesis was initiated as a research project for Sanoma’s recruitment platform Oikotie Työpaikat with the intention of exploring interesting and novel ways to use the various data generated through the use of this service. The research objectives were stated as follows:

> Find an application of data mining / machine learning to the customer-generated data on the recruitment platform Oikotie Työpaikat which has the potential of bringing value to the user of the platform and is

---

[4] The tool is publicly available on GitHub: `https://github.com/facebookresearch/fastText`

technically feasible in the scope of a master's thesis. Further define and investigate a research problem that is essential to this application by researching literature and previous work on similar problems trying different approaches based on the literature using the results and learnings to create an improved approach.

In order to meet these objectives a predefined development process was applied that will be explained in the next section.

## 1.5 Methodological Approach

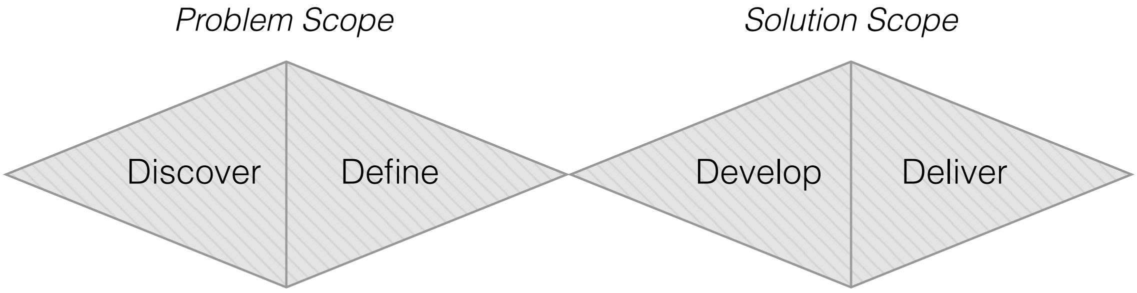


Figure 1: Design process for this thesis, adapted from the *Double Diamond Process* developed by the British Design Council (see [British Design Council, 2007])

The development process for this thesis project was adapted from the Double Diamond design process developed by the by the British Design Council in 2005 (see [British Design Council, 2007]). A design process is "the specific series of events, actions or methods by which a procedure or set of procedures are followed, in order to achieve an intended purpose, goal or outcome." [Best, 2006]. The Double Diamond process consists of iterative, explorative learning phases where first a problem worth solving is found within the *Problem Scope* and afterwards the best solution is formed in the *Solution Scope*. Both scopes are navigated through a so-called divergence phase where the space explored for finding possibilities followed by a convergence phase where the options are reduced and combined. These phases, as illustrated in Figure 1, are called *Discovery Phase* and *Definition Phase* in the Problem Scope and *Development Phase* and *Delivery Phase* in the Solution Scope and briefly described next:

**Discovery Phase** In the Discovery Phase opportunities for problems worth solving are evaluated. Often their potential is measured in terms of economic impact (financial viability), attractiveness to and impact on users (desirability) and the level of technological challenge (feasibility). These aspects require different ways of testing and as the priority in this work lays on primarily on evaluating the technical challenge the main tools were prototyping, technical benchmarking and literature research.

**Definition Phase** The aim of the Definition Phase is to compile the learnings from the Discovery Phase and find the problem that has the most potential and will be tackled. This is done by iteratively reframing the problem and testing the implications of this definition. Thus the final problem definition is often not simply selected but developed through a series of steps of refining the problem, testing and learning.

**Development Phase** When the final problem definition has been set the Development Phase aims to produce a wide variety of potential approaches to solving this problem. Here existing approaches are benchmarked and tested, and combined with new ideas, again through a *test, learn, rescope* cycle. With regards to this work literature research was combined with testing of various methods and evaluating there performance to learn which approaches work best and how they could be improved, combined or built upon.

**Delivery Phase** During the Delivery Phase the best possible solution is chosen through refinement of the different approaches that were evaluated. At the end there stands a result of the whole process, in this case a comprehensive evaluation and conclusion on the approaches.

## 1.6 Results

This thesis shows that recent approaches from Representation Learning and Sequential Modeling techniques are competitive and even yield better performance for text classification than traditional N-Gram based models that are still considered state-of-the art and widely applied. This is confirms the assumption that data-driven approaches are a promising direction for Computational Linguistics (CL) and Natural Language Processing (NLP) and beyond as they are more informed by the actual structure in the data than by model decisions of the algorithm designer using knowledge about the problem domain. Further sequential modeling of text on a character level is shown to yield competitive results while being even more domain-agnostic as the only knowledge provided is the difference between characters.

# 2 Exploration

The first part of this thesis project was dedicated to the exploration of the problem space. Driven by the Research Scope and Objectives (see Section 1.4) the goal of this phase was twofold: At once to find an approachable yet challenging research problem in the context of data from a recruitment platform and secondly to simultaneously evaluate the potential to create business value through an enhanced user experience or new use cases for the service Oikotie Työpaikat. The exploration is divided into a discovery phase where the problem scope diverges and a definition phase where it converges as described in Section 1.5. In the following sections we will look at the process and results of these phases in detail.

## 2.1 Discovery Phase: Exploring the Problem Scope

During the discovery phase the problem space was first opened up through exploration of user needs and initial field research to better understand the problem context. Additionally literature research and benchmarking of approaches helped getting a picture of related scientific work. It should be noted though that in-depth user research was kept at an absolute minimum due to time constraints and the scientific focus of this thesis.

Initially a set of interviews with stakeholders from Product Management to Marketing and Engineering working on the product as well as potential users helped obtaining a good understanding about the service offerings of a recruitment platform and the current trends in the industry. Exploring the various data that the platform generates, an important learning was that large parts of the data are in form of written language and much of it in a more or less unstructured way.

Exploration was therefore focused on language related problems. Literature research revealed a strong interest from the research community in tasks related language modeling using neural networks and related connectionist approaches that have gained traction recently through the latest advances of Deep Learning. Several methods were then tested as quick prototypes. A side-effect was the (re-)realization that real data is often noisy and inconsistent. In addition ideas for focusing the problem search were evaluated, ranging from practical ideas such as predicting trends in the job market to wilder ones such as digitalizing the complete Sanoma archive to analyze how written language evolves over time in order to predict when a text was written or to identify trending terms or writing styles from a certain period of time.

Through further experimentation, research and discussions, the problem of utilizing the implicit knowledge in the text body of job listings was deemed a good fit for further focus. It was both an interesting and challenging research problem and could potentially be of practical use, e.g. when identifying the requirements for a job inside this text. This decision set the course for the definition phase to begin with increased focus on narrowing explored set of problems to a single problem definition.

## 2.2 Definition Phase: Framing the Problem

The definition phase of the design process is characterized by explorative convergence of the problem space through iterative reframing and refinement of the problem definition, hand in hand with further experimentation and learning within the new scope. In the case of this research project it is marked by three main decision points where the problem definition was reformulated. These three iterations of converging towards a final problem definition are described next in terms of rationale for the chosen direction, the experimental approach to learn within this new scope and the results and learnings obtained.

### 2.2.1 Inferring Structure of Job Advertisements

**Rationale** During the discovery phase it became apparent that a there is large amounts of unstructured and implicit knowledge in the data of job advertisements. Specifically large portions of the data are natural language in relatively free form. One example for this which also makes up a significant amount of data is the text body of each job advertisement. This part, just like the content of a job listing in print media, contains all the information about a job position that a company wants to communicate to a potential applicant.

The contained information can be useful to provide better services, e.g. by inferring requirements for a job from the text in order to show more relevant listings to the user. Another potential application of Machine Learning to this data is to predict popularity within certain target audiences for job advertisements, for instance to help recruiters design better job listings. In many cases it is useful for these applications to first better have a better representation of the structure and content of the job ad. Thus learning the structure of job advertisements was set as a subject of further study.

**Experimental Approach** To learn how humans interpret structure and content of job ads and what they consider a good structural representation of the information contained a set of experiments was carried out. Five participants from various professional background were asked to highlight sections of the text in a job advertisement and to label these sections by describing what the section is about. Afterwards they were asked to restructure the job ad into a bullet-point list based on these highlighted sections.

No specific metrics were used as this experiment was purely explorative to gain a basic understanding how terms like *structure* and *content* or *topics* are interpreted and applied given the job listing samples. Each participant was provided with a Google Docs document including one random job advertisement in English and the tasks that were formulated as follows:

> Cheers for helping out with this little experiment for my thesis! My aim here is to find out how people would structure job ads to help find the relevant information faster.
>
> You'll work with the job advertisement below. Your task is the following:

> 1. First tag the job advertisement below into parts. Mark sections of it and use a word or two to categorize this section using the comment tool. You can tag the ad into sections however you want and even make the sections overlap. The goal here is to tag the content of the job ad that you think belongs to different categories, properties or topics. To tag the text, use the comment tool like this: “We’re looking for a Quality Buzzword Engineer.”
> 2. Now fill the bullet-point list in the last section of this document with the tagged sections/categories/topics you found. It should roughly contain the same information as the ad but in a structured way using your tags. You can use sub-points categories if you want.
>
> In total try to spend no more than 30 minutes on this. Don’t worry about it too much, the key is to just do it how it makes sense to you.

A full example of this task as it was provided for the participants can be found in the Appendix in Section A.4.

**Results** After initially expressing difficulties with the open and ambiguous task participants were provided with help by providing analogous examples (e.g. to label the description of a car, which parts describe the components while others talk about the owner etc.). While this helped it was still perceived as a very difficult task as it leaves lots of room for interpretation. The outcomes were quite various, especially along the following spectra:

- Generality versus specificity: A section about the language requirements was labelled as *Requirements* by one participant and a similar section as *language requirements*
- Content versus structure: A heading was described in one instance as *Job description* and a very similar one by another participant as *Job description section indicator*
- Objectivity versus judgement: Some sections were judged by their content, e.g. as *Empty words*

Also some participants built a nested structure in the second part of the task while others formulated rather wide areas to group the information in a flat list of topics.

**Learnings and Conclusions** One meaningful learning was that also explorative experiments need to be constrained and well-defined as posing the question so openly lead to great difficulties understanding and performing it and to diffuse outcomes. However it gave some initial insight in possible ways to capture and structure this type of texts in job advertisements and sparked discussions for the following research. Thus the problem of structuring job advertisements was set up for refinement through an experiment of larger scale.

### 2.2.2 Supervised Multi-label Paragraph Classification

**Rationale** Following the learnings about how participants interpreted the structure of job listings the problem was divided into three sub-problems:

1. Finding sections of text that form semantic units
2. Identify a label that communicates the function or meaning of these units
3. Infer a hierarchy over the labels and therefore also over the structure of the sections

As each of these problems is a challenge of its own the focus was laid on finding the labels of sections while fixing the sections to be paragraphs and ignoring the possibility of a hierarchy of the labels. It seemed a valuable outcome to be able to classify text segments and feasible to evaluate. This implied another important decision in terms of problem structure: The task was set up to be a Supervised Learning problem. As mentioned in Related work (Section 1.3), Topic Modeling and related fields are very relevant to this kind of research problem which are Unsupervised Learning approaches. While certainly a possibility to frame the problem in that way, the rationale behind the choice of a supervised problem setting was that it makes the task much easier to systematically evaluate as expected outcomes are given. Further it was of potential interest in the context of the service that specific topics could identified that are known to be existent in the data as well as possible which seemingly could lead to more accurate results when taking a supervised approach.

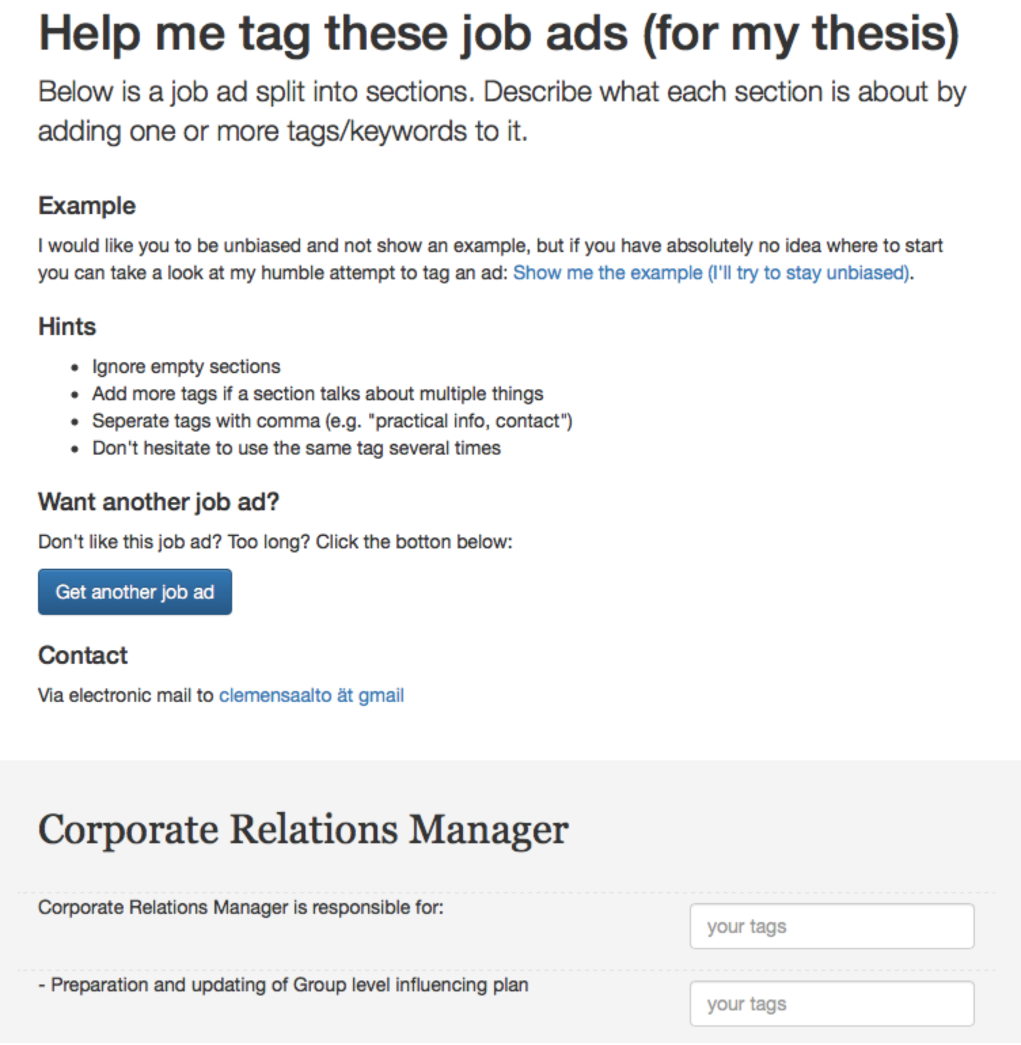
Based on these constraints a new experiment was set up where paragraphs were labelled by participants through a purpose-built web interface. The the aim was to collect a larger sample of data that would be better quantifiable, less biased and more representative through a higher number of participants, and to use this data for deepening the understanding of the problem and to evaluate the feasibility of tackling it with first proof-of-concept prototype experiments.

**Crowd-sourced Data Collection** To collect the data a tool was build, consisting of a Node.js server using MongoDB as a database and communicating via a JSON REST API with a simple website front-end using the Mustache template engine. The used data were job advertisements from the Oikotie Työpaikat job recruitment platform. The full dataset consists of 118.780 job ads published between September 01, 2012 and December 01, 2015. Using the tool langdetect, 9928 of these job advertisements were identified to be written in English language. This corresponds to approximately 8.4% of the dataset, as opposed 75% of postings that are written in Finnish. These English job listings were then split into paragraphs with a simple rule-based approach that can be seen in the software package (see Appendix, Section A.1 for a description and a link to the source code on GitHub)[5].

---

[5] see `https://github.com/cle-ment/thesis-tagger/blob/master/pre-processing.ipynb` for the specific preprocessing procedure of the job ad data.

The exact task given to the participants was: *“Describe what each section is about by adding one or more tags/keywords to it”*. Participants were shown a job ad that was split into paragraphs and besides each paragraph was a text field to enter one or more tags as Figure 2 shows.



Help me tag these job ads (for my thesis)

Below is a job ad split into sections. Describe what each section is about by adding one or more tags/keywords to it.

**Example**

I would like you to be unbiased and not show an example, but if you have absolutely no idea where to start you can take a look at my humble attempt to tag an ad: Show me the example (I'll try to stay unbiased).

**Hints**

- Ignore empty sections
- Add more tags if a section talks about multiple things
- Seperate tags with comma (e.g. "practical info, contact")
- Don't hesitate to use the same tag several times

**Want another job ad?**

Don't like this job ad? Too long? Click the botton below:

**Contact**

Via electronic mail to clemensaalto ät gmail

Corporate Relations Manager

Corporate Relations Manager is responsible for:

- Preparation and updating of Group level influencing plan

Figure 2: Screen capture of the interface of the tagging tool

In a first step the tool was only shown to 3 participants to get immediate feedback if the user interface had flaws and whether the task was understood. Based on this feedback the tool was improved by providing an example for the participants and then tested with a slightly larger group of 12 persons. After correcting a few minor details in the user interface a public link was then shared via social media and other chNNels to as many people as possible. A few days later the tool was then also shared internally within Sanoma where it was set up as a competition to tag the most possible job ads, increasing participation significantly. In total 91 job ads were tagged, resulting in 379 tagged text sections and 358 tags.

**Initial Experiments** To test whether automatic prediction of the obtained labels for paragraphs was possible a small prototype was build. Since every paragraph was potentially assigned multiple labels this first experiment was framed as a Multi-Label Classification problem where the absence or presence of each label is predicted for a paragraph. For classification a TF.IDF weighted Unigram model was applied that uses the frequencies of words associated with a label to create mappings for these labels in a common vector space. Using this vector space we can then apply various classification algorithms with the assumption that closeness of labelled data in the vector space translates to similarity of the associated objects in the real world domain (see Section 4.1.1 for details on these types of models).

Using a k Nearest Neighbors (kNN) approach the $k$ most probable labels were predicted and was called a success if the true labels intersected with the predicted ones. Using 10-fold Cross-Validation this lead to a success rate of 0.32 for $k = 5$, 0.24 for $k = 3$ and 0.1 for $k = 1$, while random predictions constantly performed a success rate of 0. This can be regarded as rather poor performance if the task is to predict the most relevant label. However it has to be considering that the task for assigning the labels was very loosely defined and the sample of data was rather small, both leading to extremely sparse data. Most labels were only used once or twice so that their data pool was only one or two paragraphs of text. While there was much room for improvement, this experiment also showed that even with sparse data learning structure with regards to the problem is possible.

**Structuring the Data** Inspection of the collected labels showed that many labels correlated, being synonyms, different versions of spelling the same term or were simply in terms of meaning. To capture this structure and try to disambiguate labels the data was clustered using two different approaches, once algorithmically and once manually. First algorithmic clustering was carried out with using Lloyds Algorithm for K-means Clustering. Effectiveness of the clustering was measured using the Silhouette Score while testing different numbers of clusters $k$. The Silhouette Score did not indicate a clear minimum at any point. Further visualization of the resulting structure showed seemingly arbitrary changes in the assignment to clusters varying the number of clusters and thus no reliable grouping could be extracted. Given the sparsity of the data the high variance of the outcomes were not unexpected however.

As a next attempt the data was sorted manually into a hierarchy. The grouping of labels was first done with a manual "Chinese Restaurant Process", i.e. the first label defined the first group and each successive label in the list was either added to one of the existing groups due to similarity or a new group was created. From this analysis different types of groups emerged, some of which overlapped in meaning. Thus as a second approach a top down process helped identify groups in a similar domain of meaning and which were largely complimentary and non-overlapping. Figure 3 shows a high-level result of this grouping. The data was then clustered into done the top-level nodes in Figure 3: *Summary: Short introduction*, *Person specification (Who you are)*, *Job specification (What you give)*, *Company (Who we are)*, *Next steps* and *Other*. The full results the manual top-down clustering can be seen in the Appendix, Section A.5 in JSON format.

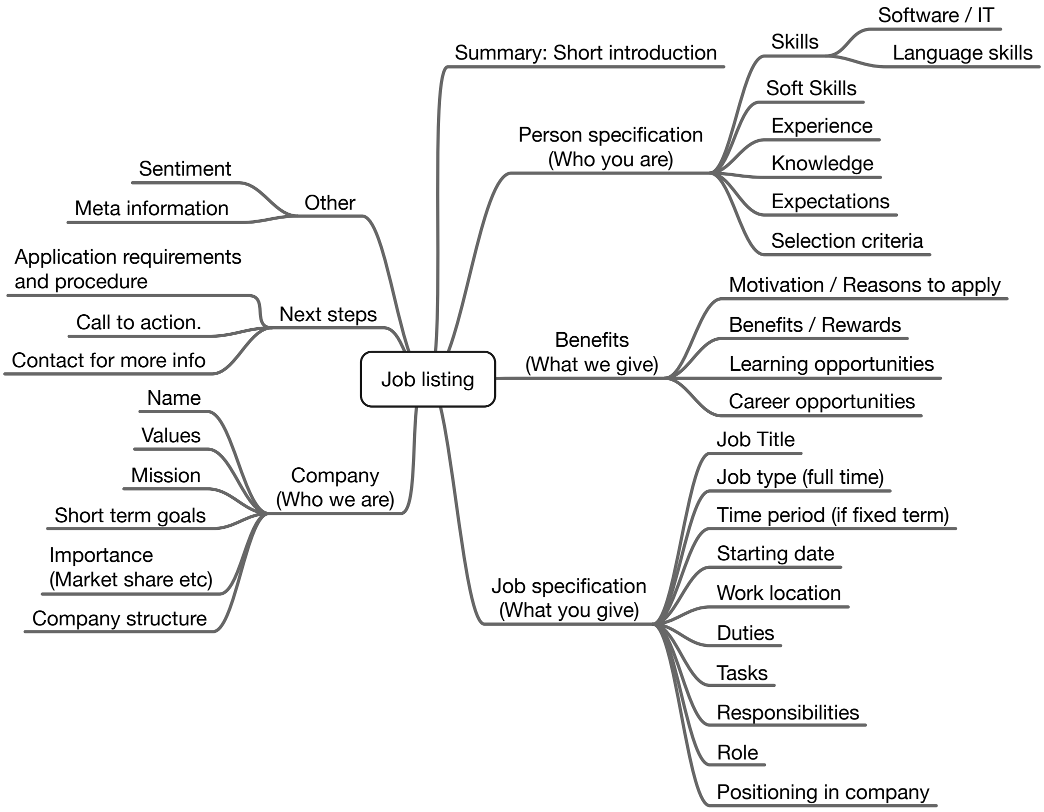


Figure 3: Structure of job listings inferred from the labels given by participants.

**Classification Experiments on Clustered Data** After clustering the data further classification experiments were carried out, this time framed as a Multi-Class Classification problem, i.e. assuming that classes are distinct. A wider array of classifiers was used on the data, including most of the algorithms described in Section 4.2: Logistic Regression, Decision Tree, Naive Bayes, SVM, kNN, Random Forest and a Neural Network with a single hidden layer. Performance was measured in $F1$ Score since at this point it was easier to use and while biased has similar properties to Matthews Correlation Coefficient (MCC) (see Section 4.4 for more information on the different performance metrics). All methods performed considerably well with an F1 test score of 0.55 - 0.60 (except kNN which only achieved 0.30). The label *Summary: Short introduction* received on average 10% lower scores across all classifiers.

**Learnings and Conclusions** The results using the clustered data gave confirmation to continue refining the problem towards this direction. The experiments showed that it is possible to learn enough structure even with a sparse and noisy dataset at hand to predict the topics of paragraphs to a certain extent. Further it showed that the grouping that was imposed manually based on exploring the collected data lead to separability meaning that the supervised setting with explicit labels is a valid choice to model the problem of identifying structure in job advertisements.

As mentioned above, an exception with regards to separability was the class *Summary: Short introduction* whose data points were often classified as belonging to other classes. A closer look at the actual text in this category revealed that this behavior was not an inability to pick up the characteristics of this class by the algorithm but in fact most examples could indeed have been easily assigned to one of the other classes. This observation also lead to the realization that this class is a grouping based on the position within the job advertisement while all other class labels express the content or topics of the paragraph. Thus the label was removed, leading to the six labels which were used in the next stage.

Another important observation when exploring the data was that often paragraphs could be separated further with regards to the six labels chosen. Thus the separation of text into paragraphs was too coarse-grained. This led to the decision to split the data on sentence-level in the next phase.

### 2.2.3 Multi-class Sentence Classification

**Rationale** Towards the end of the definition phase — and with it the exploration phase of the project — the aim was to converge towards a final problem definition that would be the focus of research for the rest of the project where the best solution approach would be evaluated. The paragraph classification experiments and their refinements proved a promising direction. For exploration the problem was previously left open to interpretation on purpose, it now needed to be reframed as a well-defined, constrained and meaningful problem definition. This was to ensure that the problem solved was relevant and modeled the problem domain in a realistic way. Further conventions from related work with regards to the problem formalization were adapted to show the relation and possible differences and put the results into perspective from a academic point of view.

With the learnings from the previous phase the problem was formulated as *sentence-level* Multi-Class Classification. The classes used were *benefits, candidate, company, job, nextsteps, other* and were assumed to be distinct. The semantics behind these classes came from the previous experiments and are explained in Section 3.3 in more detail.

**Crowd-sourced Data Collection via Crowdflower** In order to evaluate approaches to the problem task a labelled dataset of sentences was needed. For this purpose the Mikrotasking service CrowdFlower was chosen over its competitor platform Amazon Mechanical Turk due to the simple practical reason that the latter required the paying party to be an american resident. The source of data was the same dataset as in the previous experiments — a dataset of English job advertisements from the service Oikotie Työpaikat. Here 400 job english listings were selected at random and converted into a set of 10670 english sentences using the *Punkt Sentence Tokenizer*[6] of the Natural Language Toolkit (NLTK). Afterwards the sentences were converted from HTML into raw text using the Beautiful Soup package.

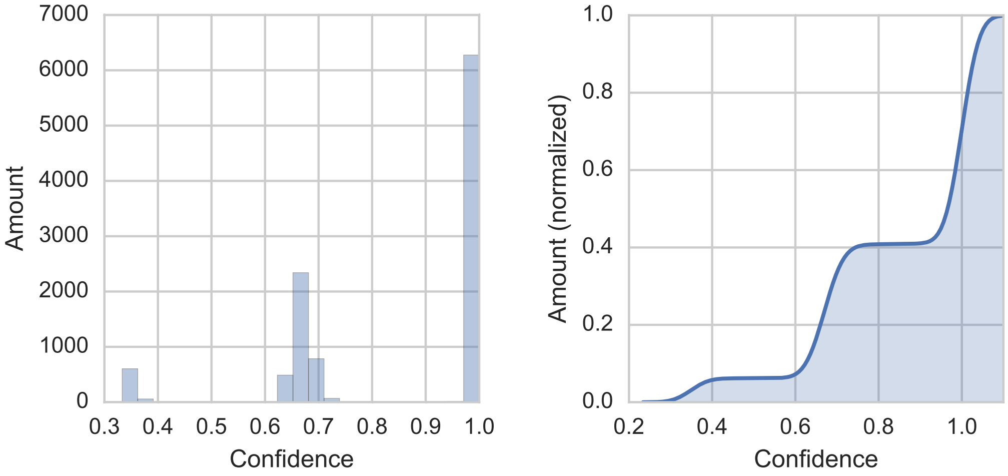
---

[6] See `http://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.punkt`

The task setup was improved over several rounds of pilot testing with a few hundred rows of data. The main learnings were in terms of posing fair test questions, i.e. questions to test participants on pre-labelled data, and further in the formulation of instructions to make the task easily comprehensible for the participants. For the final data collection each sentence was labeled thrice by different participants. This was done to assess the confidence of results in terms agreement between participants.

confirmed that the task was well-defined and solvable for humans to a good degree

The resulting dataset contained a total of 9948 sentences (93.23% of the data) with an inter-subject agreement of over 60%, meaning that at least two of three participants agreed on the label. Further only 4.74% of the sentences were labelled as *other*. Figure 4 shows the distribution of the confidence regarding the labels.



(a) Confidence (b) Cumulative Confidence

Figure 4: Amount of label judgements versus label confidence based on agreement

**First Experiments and Conclusions** To confirm that the dataset was a good representation of the problem and offered opportunity for learning algorithms to predict on unseen data a series of fast experiments was carried out. For this purpose a small set of algorithms was used including Logistic Regression and Naive Bayes using N-grams (see Section 4.1). Results showed a MCC test score of over 0.6 and F1 Scores of around over 0.65 which was. This was an improvement compared to the multi-class setup for paragraph prediction in the previous section (0.55 - 0.60 F1 score). It is important to keep in mind that this is not a fair comparison as the dataset was now consisted of sentences instead of paragraphs and was much larger. Nevertheless these first Experiments and Results along with the fact that only a small portion of data was labelled as *other* gave were a confirmation that a well-defined scope and setting for further research was now found, finalizing the problem definition phase. Furthermore, as discussed in the discovery phase in Section 2.1 there was potential for several application business use cases which are beyond the scope of this thesis, ensuring that the research objectives could be met.

# 3 Problem Definition

This section will formally and mathematically define the research problem that was chosen as a focus for this thesis. First the more general problem known in the machine learning literature as *Text Classification* is defined, then the research problem of this thesis termed *Multi-class Sentence Classification* is formalized as a special case of text classification. Finally the dataset used as a basis to evaluate problem is described in detail.

## 3.1 Context: Definition of Text Classification

Text classification, also known as text categorization, is the task of predicting a *mapping* $\tilde{\Phi} : \mathcal{D} \times \mathcal{C} \to \{True, False\}$ between a set of *documents* $\mathcal{D}$ and a set of *classes* or *categories* $\mathcal{C}$ using a model function $\Phi : \mathcal{D} \times \mathcal{C} \to \{True, False\}$. We thus aim to predict as well as possible the documents associated with each category or vice versa the categories that each document belongs into. The former is called *category-pivoted* text classification whereas the latter is referred to as *document-pivoted* text classification — the setting that we shall focus on from here onwards. The mapping $\tilde{\Phi}$ can be represented as a bipartite graph between the set of documents $\mathcal{D}$ and the set of categories $\mathcal{C}$ as shown in Figure 5. In this representation vertices in the graph represent a $True$ value in the mapping, indicating that the document and category are associated with each other, while missing vertices indicate that they are not which corresponds to a $False$ value in the mapping.

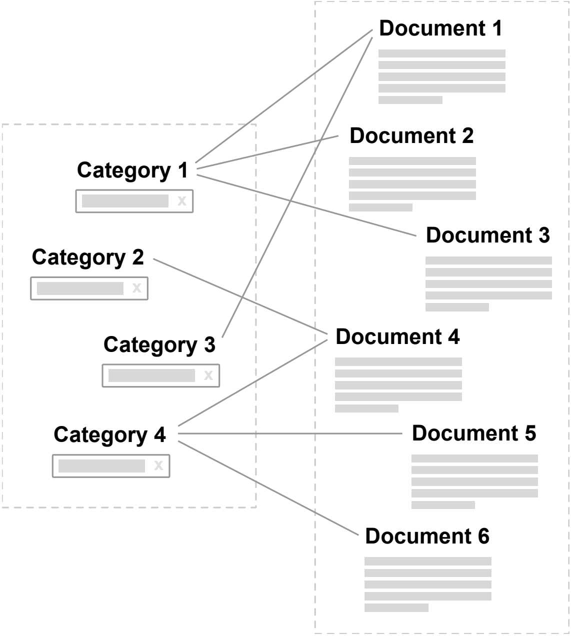


Figure 5: Text classification visualized as a bipartite graph. Here the multi-label setting is shown where no additional constraints are enforced on the problem and hence each document can be assigned to multiple categories

Categories $\mathcal{C}$ are given as symbolic labels and documents $\mathcal{D}$ as sequences of text with variable length. It is usually assumed that no additional information such as metadata or other *exogenous knowledge* is available on neither labels nor documents. As the survey on automatic text classification by [Sebastiani, 2002] points, out a consequence of relying solely on *endogenous knowledge*, especially the semantics of a text, is that there is no objective ground truth to this task in most settings since semantics are a *subjective* notion: “This is exemplified by the phenomenon of inter-indexer inconsistency [Cleverdon, 1984]: when two human experts decide whether to classify document $d_j$ under category $c_i$, they may disagree, and this in fact happens with relatively high frequency. A news article on Clinton attending Dizzy Gillespie's funeral could be filed under Politics, or under Jazz, or under both, or even under neither, depending on the subjective judgment of the expert."

Additional constraints can be imposed on the problem to adapt it for different application scenarios. Firstly text classification can be either framed as *single-label* classification where each document is assigned to only one single category or *multi-label* classification where an assignment to several categories or also no category is possible. The single-label case can be further separated into *dichotomous* or *binary* classification where the presence or absence of only a single class is predicted and *multi-class* classification where one of multiple, mutually exclusive classes is predicted for each document. Multi-class classification can thus be seen as multi-label classification with the additional constraint of classes being mutually exclusive. If labels are assumed to be statistically independent multi-label classification can also be reformulated as $|\mathcal{C}|$ individual binary classification problems, potentially allowing much simpler modeling at the cost of introducing inductive bias through a strong assumption.

## 3.2 Problem Formalism: Multi-class Sentence Classification

The prediction task in the scope of this thesis is *Multi-class Sentence Classification* which shall be formulated as a special case of Text Classification defined above. Specifically the goal is to perform *document-pivoted Multi-class Text Classification* where the documents $\mathcal{D}$ are the set of all sentences $S$ drawn *uniformly and independently* at random from a set of job advertisements $\mathcal{J}$, and the classes $\mathcal{C}$ are set to be the following set of mutually exclusive labels $\mathcal{L} = \{\text{benefits}, \text{candidate}, \text{company}, \text{job}, \text{nextsteps}, \text{other}\}$. We thus allow for no knowledge to be used regarding the origin of a sentence $s_i$, meaning that the prediction of each sentence is independent must assume the same prior information. The task can hence be expressed as predicting true label $l_i$ for a sentence $s_i$, i.e. finding the mapping $\widetilde{\Phi} : \mathcal{S} \to \mathcal{L}$.

## 3.3 Dataset: Labelled Sentences from Job Advertisements

The performance of approaching the research problem will be evaluated using a dataset that was designed and created specifically for the purpose of this work. For detailed information on the process and design decisions involved please refer to Section 2.2 (Definition Phase: Framing the Problem). Here the key characteristics of the data will be shown.

The data from this set stems from job advertisements created by companies using the Sanoma's service Oikotie Työpaikat in Finland. Each job posting contains many fields of associated metadata such as the title of the job posting, the time of creation and many more. Of these only the *job description* was chosen as a basis for this task since — as pointed out previously in Section 3.1 — the task of interest was pure text classification without exogenous knowledge with the motivation to better understand unstructured data such as the job description which does not follow a specific format.

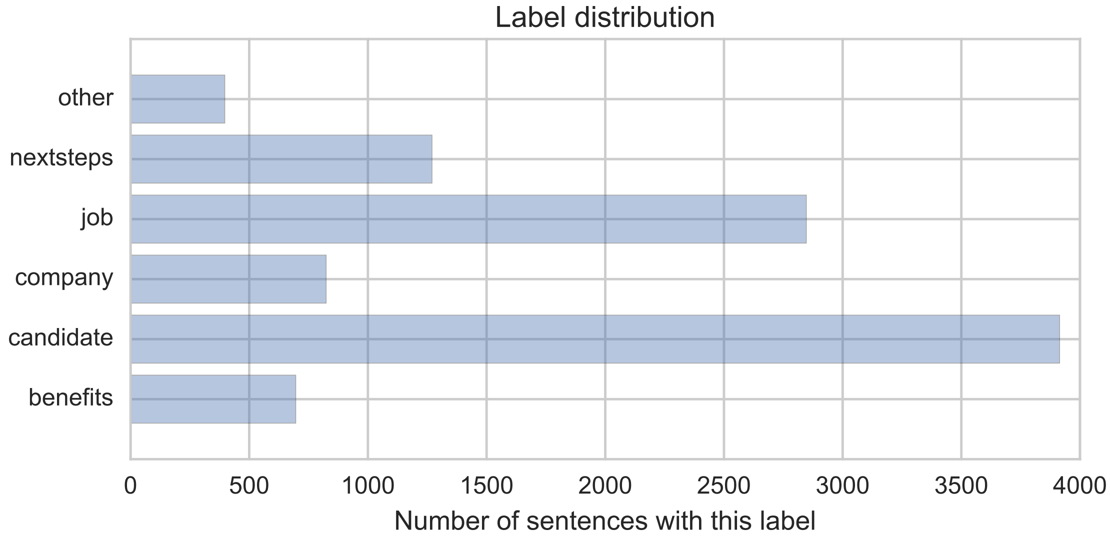


Figure 6: Distribution of labels in sentence data

The dataset was collected using the Crowdsourcing service CrowdFlower as described in detail in Section 2.2.3.

The resulting dataset consists of 9948 labelled sentences totaling to an amount of 121,119 words and 807,984 characters. To each sentence one of the following labels is assigned: *benefits, candidate, company, job, nextsteps, other.* Table 1 shows the meaning of each class and an example of a sentences associated with it. The resulting distribution of the labels is strongly skewed towards higher prevalence of several labels as shown in Figure 6.

## 3.4 Performance Metric: Matthews Correlation Coefficient

As a metric for predictive performance Matthews Correlation Coefficient (MCC) was chosen which is described in detail in Section 4.4 along with other performance metrics. This metric is used relatively rarely used in the machine learning literature as opposed to e.g. the F1 score that is common in the IR literature where ignoring True Negatives can be tolerated. As an example we do not generally care if a search engine predicts correctly all the billions of website we don't want to see for a search query as long as it retrieves enough relevant ones. However, with the dataset at hand a metric was needed that measures prediction reliably and without bias even in case of a strongly skewed distribution of labels. Stratified sampling from the dataset to achieve a balanced distribution was not an option since the dataset was too small. Thus MCC was selected which fulfills these criteria and additionally is easy to interpret: It is a correlation score between -1 and 1, denoting anti-correlation and correlation respectively.

| Label | Description | Example Sentence |
|---|---|---|
| benefits | **Benefits**: Rewards, opportunities, reasons to apply, … | And we like to think we'll give you an enjoyable and inspiring place to spend your working day. |
| candidate | **Candidate requirements**: Requirements, skills, experience, education, personality, … | To succeed in this position it is essential to have strong experience of at least 5 years in international business development and/or international B2B sales and marketing. |
| company | **Company information**: Company name, story, mission, structure, market share, … | Progman is software house specialising in the development of software for Building Services. |
| job | **Job description**: Role, responsibilities, location of work, type of employment, … | Your main objective is to maximize Core Fleet, Dealer B2B, municipality and governmental orders and sales for PC and LCV range to local Fleets and businesses within defined geographical area. |
| nextsteps | **Next steps**: Call to action, application procedure, contact, further information, … | 040 75 67 316, Mon-Fri 10-14, or peas@temp-team.fi. |
| other | **Other**: Does not fit into the above categories | URS' |

Table 1: Labels used for the task with their description that was given for the participants labelling the data and a sentence example for each label.

# 4 Methods

Text classification and related problems have been of interest for research for decades, as mentioned in the Related work section (1.3). Therefore, numerous approaches have been proposed to tackle this task. This section introduces the reader to two general types of approaches: first the method of using Vector Space Models for classification where each text is represented as a vector through transformations of the data and second the technique of modeling text as a sequence and applying algorithms that are specifically well-suited for learning temporal patterns of sequential signals.

This section will be slightly more technical than the previous ones where needed to formally describe the techniques. Thus familiarity with fundamental concepts from Machine Learning, Function Approximation and Optimization, Probability Theory, Information and Decision Theory, and a general knowledge of Linear Algebra is assumed. However, the approaches are also introduced to some degree on a conceptual and intuitional level so that the basic ideas can be understood without this knowledge. The technically inclined reader on the other hand is encouraged to follow the suggested references on the techniques for further study and in-depth coverage that was beyond the scope of this work.

## 4.1 Vector Space Models

Vector Space Models are a popular approach of modeling real world objects that expresses their characteristics and structure through a numerical representation. Specifically this representation assigns real-valued fixed-size vectors to each object, i.e. to each text document in the context of text classification. This means that our objects exist in a common vector space $\mathcal{V}$.

To be a meaningful representation of data that allows to learn patterns this vector space $\mathcal{V}$ follows the contiguity hypothesis: “Documents in the same class form a contiguous region and regions of different classes do not overlap.” [Manning et al., 2008, Chapter 14, p. 289]. This means the similarity of objects is expressed through their distance in the vector space. Further we often assume that the dimensions of a vector encode certain properties or latent factors of the objects or data we are modeling. This adds the notion of sub-vector distance where objects can be close and thus similar to each other with respect to only certain dimensions in the vector space; a property that encodes more fine-grained structure in the data and can be exploited by classification algorithms.

A concrete example for illustration is the Unigram model, the simplest form of N-Gram models, which will be introduced in the next section: each vector encodes word frequencies of words in an associated document, i.e. the vector is basically just a histogram of the word counts in this document. Looking at the histograms for several documents in the dataset can already reveal certain properties visually such as topics. If for instance lots of words regarding politics accumulate in one document whereas another document shows high frequencies of sports related terms.

The next sections will introduce two model approaches of this type with their properties and common variations. First, N-gram models will be described, which are in their core based on word (co-)occurrence frequencies. Then, a more recent approach from the field of Representation Learning will be discussed where word and document representations are automatically learned based through context prediction.

### 4.1.1 N-gram Models

N-gram language models are based on co-occurrences of word or character sequences, generally referred to as N-grams or $k$-shingles in the Data Mining literature [Leskovec et al., 2014, Chapter 3.2, p. 72]. Formally an N-gram is defined as a sequence of $n$ items, each of which consist of $n$ characters or words, effectively used to capture sub-sequences of text. Common choices are N-grams of size 1, 2 or 3 — called *unigrams*, *bigrams* and *trigrams* respectively — and the definition can be extended to using a window size $[w_{\min}, w_{\max}]$, employing all combinations of N-grams in this interval.

N-grams are usually used to create a vector-space model by representing each document in a dataset as a *bag-of-words* or *bag-of-N-grams* vector so that each dimension of the vector represents statistics about the corresponding N-gram. Specifically, a common way to compute the word count vectors for a document is the following:

$$\mathrm{TF}_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{1}$$

Where $f_{ij}$ is "the *frequency* (number of occurences) of a term (word) $i$ in document $j$" and $\mathrm{TF}_{if}$ is the *term frequency*, i.e. "$f_{ij}$ normalized by dividing it by the maximum number of occurrences of any term [. . . ] in the same document" [Leskovec et al., 2014, Chapter 1.3.1, p. 8].

As this approach has been studied for decades there is quite an extensive amount of variants and thus hyper-parameters to tune. The most important ones will be explained in the following sections:

**Words vs. Characters** The first choice when building an N-gram language model is to use characters or words as the atomic units. However in practice there are less characters than words in a dataset, but to capture expressive substrings usually larger N-gram window sizes or ranges have to be chosen, which leads to a combinatorial explosion. In case of word-based models on the other hand the maximal size of the feature space is the size of the vocabulary $\mathcal{V}$ in the case of unigrams or $V^k$ in case of $k$-grams.

**Stop words** For creating N-gram models, so-called stop word lists are often used which are lists of frequent words that will be excluded as they do not carry much meaning [Leskovec et al., 2014, Chapter 1.3.1, p. 7]. The stop-word list used in these experiments is the standard list used for the Scikit-learn framework [Pedregosa et al., 2011] is available online by the University of Glasgow Information Retrieval group[7].

---

[7] `http://www.gla.ac.uk/schools/computing/research/researchoverview/informationretrieval/`. The full stop word list can be found in the Appendix in Section A.3.

**N-gram range** The N-gram range, also known as window size or shingle size, refers to combinations of the atomic units of the model (words or characters) and defines an upper and lower limit for these combinations. For example a range of $[1,1]$ specifies a unigram model, $[2,2]$ a bigram model and $[1,2]$ a combination of both including all unigrams and all bigrams. A larger range allows the model to capture an increasing amount of word order and context. Thus it also leads to a combinatorial explosion in terms of feature space.

**Vector size** The vector size imposes an upper limit to the vector size and therefore the number of N-grams that can be encoded in the feature space. Commonly this approach simply uses the words with the highest frequency to reduce the vector size from the full length — the size of the vocabulary — to the desired size.

**TF.IDF weighting** A common extension to using word-counts is to weight the term frequencies by the so-called inverse document frequency, i.e. the inverse of the frequency of an term or N-gram in all documents. This method is commonly referred to as *TF.IDF*. Specifically, the inverse document frequency is defined as $\mathrm{IDF}_i = \log_2(N/n_i)$, where logarithmic smoothing is applied. The TF.IDF value for a term or N-gram is then computed as $\mathrm{TF}_{ij} \times \mathrm{IDF}_i$.

**Sublinear TF scaling** As [Manning et al., 2008, Chapter 6.4.1, p. 126] suggests "[it] seems unlikely that twenty occurrences of a term in a document truly carry twenty times the significance of a single occurrence". Hence a common variant is *sublinear scaling* where we down-weigh the increase in term importance by applying a logarithmic function to it, resulting in the sub-linear term frequency $\mathrm{subTF}_{ij}$:

$$\mathrm{subTF}_{ij} = \left\{ \begin{array}{ll} 1 + \log \mathrm{TF}_{ij} & \mathrm{TF}_{ij} > 0 \\ 0 & \text{otherwise} \end{array} \right\}$$

**Normalization** Often the term vectors are globally normalized using the $L_1$ or $L_2$ norm to remove the effect of statistical differences between the terms.

There are, of course, various other variants and modifications to the N-gram model, but within the scope of this thesis only the most notable ones were introduced and will be used for experiments later. For further material on this subject refer for example to [Manning et al., 2008].

Today N-gram models are still in wide use and considered as state of the art "not because there are no better techniques, but because those better techniques are computationally much more complex, and provide just marginal improvements" [Mikolov, 2012, p. 17]. As [Mikolov, 2012] points out further "[the] most important weakness is that the number of possible n-grams increases exponentially with the length of the context, preventing these models to effectively capture longer context patterns. This is especially painful if large amounts of training data is available, and most of the patterns from the training data cannot be effectively represented by n-grams and cannot be thus discovered during training. The idea of using neural network based LMs [Language Models] is based on this observation. This approach tries to overcome the exponential increase of parameters by sharing parameters among similar events, and thus no longer require exact matching of the history H.” [Mikolov, 2012, p. 17]

### 4.1.2 Distributed Continuous Representations

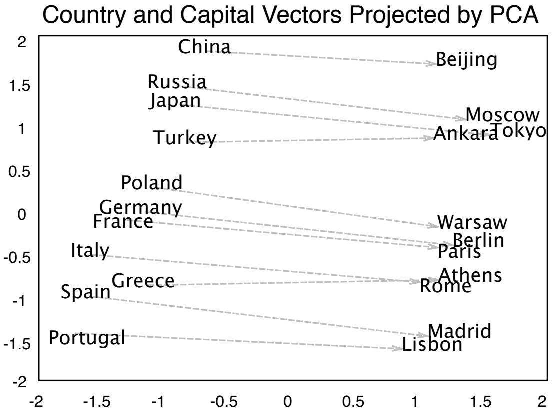


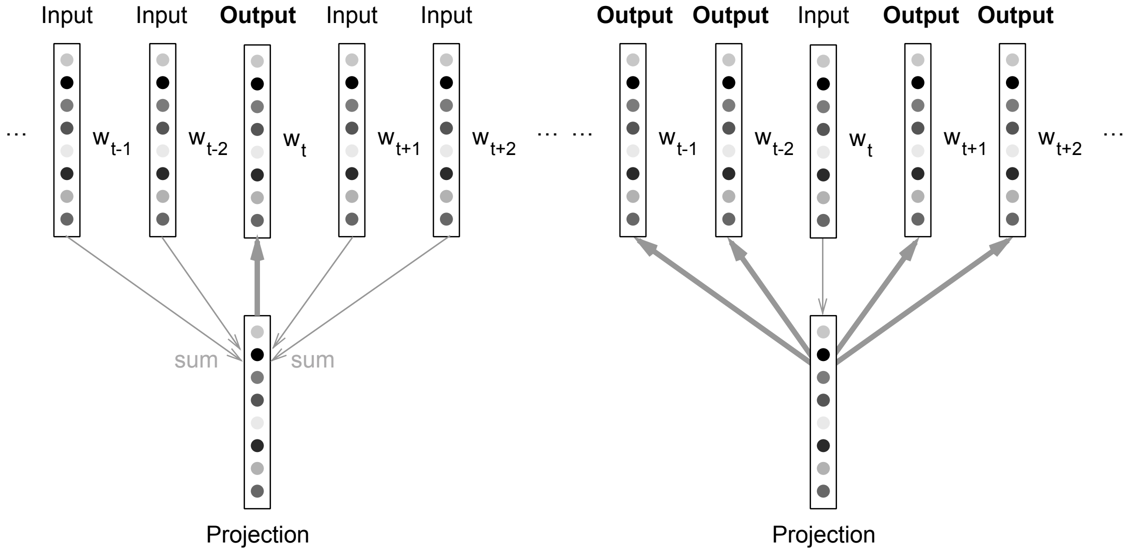
Figure 7: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. (Adapted from [Mikolov et al., 2013c])

To overcome the shortcomings of popular language models such as the N-gram model mentioned above, much recent work has focused on the study of so-called distributed language models. One branch of research that gained significant attention is the work on Neural Network based Language models (NNLMs), popularized largely through the work of T. Mikolov and his software realization of such a model dubbed *word2vec* with interest coming not only from the academic community. Mikolov’s work builds on ideas introduced in [Bengio and Bengio, 2000] where a neural network based model was proposed for modeling high-dimensional discrete data, which was then applied to the domain of language modeling in [Bengio et al., 2003]. The idea of learned distributed representations goes back to [Hinton, 1986]. Following the description in [Bengio and Bengio, 2000], the approach is as follows:

1. Associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. Express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence
3. Learn simultaneously the *word feature vectors* and the parameters of that *probability function.*

To achieve this, a feedforward neural network model is trained to learn the *word feature vectors* or *word embeddings*. As input a sequence of $n$ words is given, each encoded using one-hot encoding or one-of-$V$ encoding where the corresponding indicator vectors for each word have the size of the vocabulary $V$. The input word vectors are then projected linearly into a projection layer of significantly lower dimensionality $D$, using a global projection matrix for across all words, and concatenated, forming the input of size $D \times N$ to a hidden layer of size $H$. The hidden layer then feeds non-linearly into the output layer that is again of size $V$, modeling the probability distribution for a word given its context $P(w_t \mid w_{t-n}, \dots, w_{t-2}, w_{t-1})$.

**Simplified Continuous Models.** [Mikolov et al., 2013a] then introduced two simplified models, removing the hidden layer and only using a projection layer, with shared weights for all words. The Continuous Bag-of-Words Model (CBOW) named model is trained to predict the current word $w_t$ given the $k$ words around it. The naming comes from the fact that the word order does not influence the projection as the word vectors are summed or averaged. The Continuous Skip-gram Model works the other way around, predicting the most likely $k$ words around a given word $w_t$. Figure 8 illustrates both models.



(a) Continuous Bag-of-Words Model
(b) Continuous Skip-gram Model

Figure 8: Architectures for learning continouus distributed word vectors, adapted from [Mikolov et al., 2013a]

These models have been shown to outperform the state-of-the-art N-gram models on various tasks (see e.g. [Bengio et al., 2003] or [Mikolov, 2012]). An interesting outcome of this research is the fact that these *word vectors* capture many interesting and often subtle semantic regularities which can be exploited explicitly in an algebraic manner. When trained on an extensive dataset, one can perform calculations as $v(Paris) - v(France) + v(Germany)$ and the closest vector to the result turns out to be $v(Berlin)$ where $v(\cdot)$ denotes the *word vector* of a word. Figure 7 shows a PCA projection of Skip-gram trained vectors of countries and their capital cities.

A notable alternative to these models was developed by [Pennington et al., 2014]. In their model called *GloVe*, which stands for global vectors, they construct a vector space model with similar properties as the models introduced above, which instead relies on the global word-word co-occurrence counts. This method thus operates directly in the co-occurrence statistics of the corpus compared to the Neural Network based methods that "fail[. . . ] to take advantage of the vast amount of repetition in the data" [Pennington et al., 2014].

There have been various extensions and variants to the Neural Network based language models especially, including architectures based on Recurrent Neural Networks (see [Mikolov, 2012]). Some of the most important variations will are discussed in the following section as they were evaluated in the experiments:

**Hierarchical Softmax** The architectures proposed in [Bengio and Bengio, 2000], [Bengio et al., 2003] and follow-up work use a *softmax* activation function at the output layer in order to obtain valid probabilities for each word to be predicted:

$$\mathrm{softmax}(\mathbf{x}_j) = \frac{\exp(\mathbf{x}_j)}{\sum_k \exp(\mathbf{x}_k)} \tag{2}$$

*Hierarchical Softmax* uses a binary tree to encode the output which leads to an efficient approximation of the full softmax and speeds up on training and inference. Details can be found in [Mikolov et al., 2013b].

**Negative Sampling** Another technique applied by [Mikolov et al., 2013b] *Negative Sampling* which is a simplified version of Noise Contrastive Estimation (NCE) introduced by [Gutmann and Hyvärinen, 2012]. Based on the insight that a good model should be able separate noise from signal, this method mixes samples from a noise distribution into the signal to be learned, in this case random words that are not in the context window, which is shown to approximately maximize the log probability of the softmax. Free parameters of this technique are the number of negative samples $k$ per data sample and the noise distribution $P_n(w)$

**Sub-sampling of Frequent Words** As the difference between frequent and infrequent words in large corpora can be huge and the frequent words often do not carry as much meaning, in [Mikolov et al., 2013b] a simple sub-sampling technique is used to counter this imbalance by discarding words with a probability computed as follows:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_f)}} \tag{3}$$

with $f(w_i)$ denoting the frequency of word $w_i$ and $t$ denoting a threshold. [Mikolov et al., 2013b] state that this method, while chosen heuristically, "accelerates learning and even significantly improves the accuracy of the learned vectors of the rare words".

### Distributed representations for documents

The models explained above are defined on words as the atomic unit. Therefore several ways have been proposed to extend these to sequences of words in order to obtain a vector space of sentences or documents. A few of these will be briefly outlined here:

**Compositional Models and Bag-of-Means** Several approaches of composing document vectors out of word vectors have been studied in [Mitchell and Lapata, 2010]. One of the most successful variants evaluated was weighted averaging of word embeddings, a technique also referred to as *Bag-of-Means*. However this approach "loses the word order in the same way as the standard bag-of-words models do." [Le and Mikolov, 2014]. This intuitive property was confirmed by [Zhang et al., 2015] where the method consistently performed poorest in comparison to other approaches on a variety of tasks.

**Paragraph Vectors** In [Le and Mikolov, 2014] a different approach is shown that builds on the same idea as the original word2vec model. In particular two approaches proposed. The first model, dubbed *Distributed Memory (PV-DM)* is similar to the continuous Bag-of-Words model as a word is predicted given the words in the context, but additionally a paragraph token is used as an input to the model. To the algorithm this simply acts as another word index but it identifies the paragraph the words stem from and serves as a context memory for this paragraph. The second approach is called *Distributed Bag-Of-Words (PV-DBOW)* and does not use word prediction at all. Instead words are sampled from a paragraph and the associated paragraph vector has to be learned to predict these words as well as possible. This model is therefore more similar to the Skip-Gram model by [**?**] as the several words are predicted using a single learned representation. At inference time both models need to construct new Paragraph Vectors for unseen paragraphs using a slightly varied learning procedure (see [Le and Mikolov, 2014]).

## 4.2 Methods For Classification With Vector Space Models

Many classical machine learning algorithms work on data in a fixed-dimensional vector format and can thus be applied to a vector space model as described in the previous section. Here a set of well-known algorithms will be briefly introduced, each representing a family of approaches with different model assumptions. It should be pointed out that there exist a plethora of other models and algorithms that can be used. The ones chosen here though are widely considered standard methods and have proven successful in the past for a wide range of problems.

### 4.2.1 Generalized Linear Models

Generalized Linear Models are an extension of linear discriminant functions. Whereas discriminant functions and ordinary linear regression models produce linear decision boundaries in the input space (*linear-response model*), Generalized Linear Models apply a non-linear *link function* or *basis function* to the input, making it possible for the model to achieve non-linear decision boundaries in the original input space while corresponding to a linear decision boundary in the feature space.

The most prominent algorithm in this class is *Logistic Regression* which uses the *logistic sigmoid* function as a basis function:

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{4}$$

This yields posterior class probabilities of:

$$p(\mathcal{C}_n \mid \phi) = y(\phi) = \sigma(w^T \phi) \tag{5}$$

where $\mathcal{C}_n$ denotes class $n$, $w$ is the learned *weight vector* with the free parameters of the model, $\phi$ is the *feature vector* of inputs transformed through the basis function $\sigma$ and $y$ is the model function that we use to infer class predictions from our data:

$$y_n = \sigma(a_n) \qquad \text{where} \quad a_n = \mathbf{w}^T \phi_n \tag{6}$$

Using a the maximum likelihood method we can now find a set of parameters for our model function $y$ that gives us the predictions with the highest likelihood. For a dataset $\{\phi_n, t_n\}$, where $t_m \in 0, 1$ and $\phi_n = \phi(x_n)$, with $n = 1, ..., N$:

$$p(\mathbf{t}|w) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n} \tag{7}$$

where $\mathbf{t} = (t_1, \ldots, t_N)^T$ and $y_n = p(C_i \mid \phi)$. We can now define an error function by taking the negative logarithm of the likelihood, which is called the *cross-entropy* error function:

$$E((w)) = -\ln p(\mathbf{t}|w) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{8}$$

When taking the gradient we with respect to $\mathbf{w}$ we obtain:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n) \phi_n \tag{9}$$

From here we can apply an optimization procedure such as gradient descent to obtain a weight vector $\mathbf{w}$ that minimizes the error function above. A deeper treatment of the Logistic Regression and Generalized Linear Models can be found in many introductory textbooks such as [Bishop, 2006, Chapter 4.3.2, p. 205].

### 4.2.2 Bayesian Classifiers

Bayesian Classifiers are a family of probabilistic models derived from Bayes' Theorem, which itself is a simple application of the rule conditional probability:

$$p(A \mid B) = \frac{P(B \mid A) P(A)}{P(B)} \tag{10}$$

Where $P(A)$ is the probability of A and $P(A \mid B)$ conditional probability of A given B. In the Bayesian interpretation probabilities represent a degree of belief that changes under the observation of evidence. The following form is usually referred to as Bayes' rule and expresses this interpretation:

$$\underbrace{p(A \mid B)}_{posterior} \propto \underbrace{P(B \mid A)}_{likelihood} \underbrace{P(A)}_{prior} \tag{11}$$

where $P(B)$ is marginalized out as a normalization constant of the probabilities. This simple rule forms the basis of a whole subfield of Bayesian Inference (see e.g. the textbook [Barber, 2012] for a good coverage of the topic).

The most known Bayesian classification algorithm is the *Naive Bayes* classifier, a simple class-conditional generative model which builds on the assumption that all features are conditionally independent given the classes $\mathcal{C}_n$. For each class $c_j$ we can predict it's probability given an input vector $\mathbf{x}$ as:

$$p(\mathbf{x}) = p(c_j) \prod_{i=D}^{D} p(x_i \mid c_j) \tag{12}$$

We can then use Bayes' rule to form a classifier for each class:

$$p(c_j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid c_j)p(c_j)}{p(\mathbf{x})} = \frac{p(\mathbf{x} \mid c_j)p(c_j)}{\sum_c p(\mathbf{x} \mid \mathbf{c}p(c))} \tag{13}$$

We can then choose the class with the highest class probability as our prediction. Naive Bayes is a simple instance of a more general class of models generally referred to as graphical models, a family of techniques which are covered e.g. in [Barber, 2012] or [Bishop, 2006].

### 4.2.3 Tree Based Methods

A simple but widely used classification as well as regression method are *Decision Trees* which work by recursively partitioning the input space into dichotomous subregions, producing a binary decision boundary. At inference time the resulting tree can then be used to produce an output by navigating through a path from its root to a leaf, answering a binary questions at each node to determine which child node to query next. Figure 9 shows the visualization of an exemplified such a tree[8]. Every node represents a threshold in the input space that the input data is compared against, corresponding to a decision such as "Is the elevation of this building's elevation higher than or equal to 91.9 feet?". If the answer is "Yes" the left child node is queried next, otherwise the right. This procedure is repeated until a leaf with a final answer is reached.

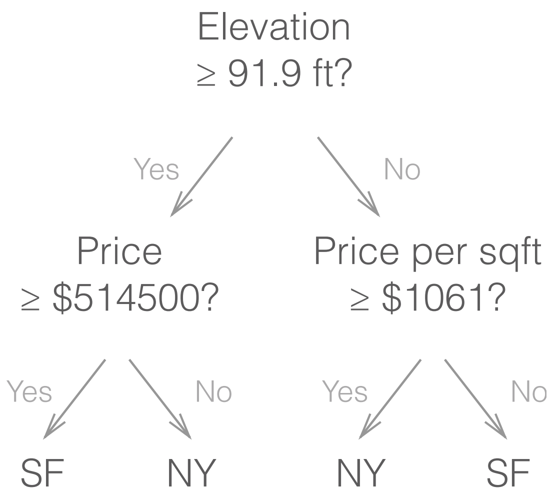


Figure 9: A simple Decision Tree to determine whether a building is located in New York (NY) or San Francisco (SF).

---

[8] Adapted from R2D3's "visual introduction to Machine Learning" at `http://www.r2d3.us/visual-intro-to-machine-learning-part-1/`

The technique was first introduced by [Breiman et al., 1984] under the term *Classification And Regression Tree (CART)* but exists in many variants. Learning such a model involves specifying the structure of the tree, including which input variable to query at each node in the graph and what each of the nodes' threshold values are. Since commonly it is computationally infeasible to generate and evaluate all possible decision trees for a given problem, a greedy procedure is usually chosen. We start with determining a first split in the data that represents the root node and grow the tree one node at a time, each node in the tree being determined by choosing splits that minimize an error measure.

Let $p_{\tau k}$ be the proportion of data points in region $R_\tau$ that are assigned to class $k$. Then we define the error as the negative cross-entropy:

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} \ln p_{\tau k} \tag{14}$$

A common alternative for the classification error is the *Gini index*:

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k}(1 - p_{\tau k}) \tag{15}$$

Both measures vanish for $p_{\tau k} = 0$ and $p_{\tau k} = 1$ and are maximal at $p_{\tau k} = 0.5$, thus rewarding regions with a high proportion of data points assigned to one class. As [Bishop, 2006, Chapter 14.4, p. 664] notices "cross entropy and the Gini index are better measures than misclassification rate for growing the trees because they are more sensitive to the node probabilities. Also, unlike misclassification rate, they are differentiable and hence better suited to gradient based optimization methods".

Decision trees are often chosen because they introduce little inductive model bias, however one has to be aware that this leads to high variance due to the Bias-Variance Dilemma. The algorithm is also prone to overfitting which is often countered by pruning, using a criterion that balances error with model complexity, such as:

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda|T| \tag{16}$$

Where $\lambda$ is a regularization parameter and $|T|$ maximal depth of the tree, i.e. the longest path in the tree. Pruning of the tree happens after training a full tree as evidence shows that sometimes several splits occur that do not significantly reduce the error, followed by one that does [Bishop, 2006, Chapter 14.4, p. 664].

### 4.2.4 Ensemble Methods

Another approach to machine learning algorithms are so-called Ensemble Methods. These methods use different strategies to combine a set of classifiers, often employing techniques of Randomized Algorithms. There is a variety of such algorithms including Bagging, Boosting and Voting methods which are covered in most of the popular introductory books to machine learning, e.g. [Bishop, 2006].

The Random Forests learning algorithm is a popular Ensemble Method introduced by [Breiman, 2001]. It is based on the concept of Decision Trees explained above, but instead of a single decision tree a set of trees, i.e. a forest, is grown on sets of bootstrapped samples from the input space. Specifically the algorithm grows $B$ random trees by repeating the following steps:

1. A *bootstrapped sample* $S_b$ of data points is generated by drawing samples uniformly at random with replacement from the original data $D$. The sample is of smaller size than the dataset and is thus based on a subset of the input data. Notice that data can be sampled multiple times.

2. A decision tree $T_b$ is trained on the bootstrapped data sample and is usually pruned at a certain limit of nodes.

Then the resulting forest of trees $T_1, \ldots, T_b$ is used as an ensemble of classifiers using a majority vote over the predictions:

$$C(X) = \frac{1}{B} \operatorname*{argmax}_i \sum_{b=1}^{B} \mathcal{I}(y_{T_b} = i) \tag{17}$$

Where $\mathcal{I}(\cdot)$ is the indicator function. Other voting schemes have been proposed as well , such as informing the weights through statistical significance testing.

### 4.2.5 Instance-based Methods

Another common class of learning algorithms are *instance-based learning methods*, also known as *example-based methods* or *memory-based learning methods*. These methods directly use the given training examples and compare the new, unseen examples with the training data at inference time, thus constructing the hypothesis directly from the data.

One of the most prominent instance-based classification methods is the kNN algorithm which simply makes a class prediction based on the majority class amongst its $k$ nearest neighbors in the input space. This type of learning algorithm is a

The choice for the parameter $k$ depends on the type of problem and is often determined empirically via hyper-parameter optimization or based on heuristics. The kNN algorithm is highly prone to the curse of dimensionality as when a metric such as the Euclidean Distance for the kNN algorithm to determine the nearest neighbors to the query vector. As the dimensionality grows the pairwise distance between the samples becomes almost equal. Thus often Feature Extraction or Dimensionality Reduction techniques are employed before applying the kNN algorithm.

### 4.2.6 Kernel Methods

Kernel Methods are a set of instance-based methods (see Section 4.2.5), as they construct the hypothesis and make predictions using the training data. Similar to other learning algorithms, kernel methods make use of transformations from an input space $\mathcal{X}$ into a another space $\mathcal{V}$. If the transformation is non-linear this allows a learning algorithm to learn a linear decision boundary in the projected space $\mathcal{V}$ which then corresponds to a non-linear decision boundary in the original space $\mathcal{X}$.

Kernel methods build on this idea and enable us to completely avoid the explicit mapping from the input space $\mathcal{X}$ to the transformed space $\mathcal{V}$ using the so-called *kernel trick*: Using the fact that certain functions $k(x, x')$ in a space $\mathcal{X}$ can be expressed as an inner product in another space $\mathcal{V}$, we can directly operate on the scalar dot products which can be interpreted as distances or similarities between the data. A function adhering to this property is called a *kernel function* and a variety of such functions have been proposed, with extensions beyond transformations on real-valued vectors to kernel functions on complex objects such as graphs (see e.g. [Shervashidze et al., 2011] for an interesting example of a graph kernel method).

A particularly successful classification method using kernels is the Support Vector Machine (SVM) algorithm introduced by the name of *Support Vector Networks* by [Cortes and Vapnik, 1995]. Originally it was formulated as a linear classifier in [Vapnik, 1982] and as such belongs to the family of generalized linear models introduced in Section 4.2.1. This corresponds to using a so-called linear kernel as $k(x, y) = x^T y$. The SVM belongs to the class of *maximum margin classifiers*: Figuratively speaking and assuming linear separability and dichotomous classification, the decision boundary is trained to be maximally distant to the closest data points of each class. We can formalize the approach as follows:

We are given data points $x_1, \ldots, x_N$ and target values $t_1, \ldots, t_N$ for these points where $t_n \in -1, 1$ and $x_n \in \mathbb{R}^d$. We will use a linear model of the form:

$$y(x) = w^T \phi(x) + b \tag{18}$$

where $\phi$ denotes a transformation function. Then the perpendicular distance of a point $x$ to the decision hyperplane is given by

$$\frac{|y(x)}{\|w\|} \tag{19}$$

Since we want a hyperplane where all data points are correctly classified the decision boundary takes the form

$$\frac{t_n y(x_n)}{\|w\|} = \frac{t_n(w^T \phi(x_n) + b)}{\|w\|} \tag{20}$$

We now want to find the parameters $b$ and $w$ so that we maximize the margin which is given by the closest points to the decision boundary. This can be formulated as

$$\underset{w,b}{\operatorname{argmax}} \left\{ \frac{1}{\|w\|} \min_n [t_n(w^T \phi(x_n) + b)] \right\} \tag{21}$$

We can define the margin being one by scaling the input space which leads to the constraint that each point is either on the margin with distance 1 or further away:

$$t_n(w^T \phi(x_n) + b) \geq 1 \qquad n = 1, \ldots, N \tag{22}$$

We are thus given a constrained optimization problem which can be solved using Lagrange multipliers $a_n \geq 0$ giving:

$$L(w, b, a) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n\{t_n(w^T\phi(x_n) + b) - 1\} \tag{23}$$

Taking the derivatives of $L$ with respect to $w$ and $b$ yields:

$$w = \sum_{n=1}^{N} a_n t_n \phi(x_n) \tag{24}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{25}$$

Using (24) and 25, we can rewrite (23) as:

$$\widetilde{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \tag{26}$$

where $a$ is subject to the constrains

$$a_n \geq 0 \qquad n = 1, \ldots, N \tag{27}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{28}$$

and the kernel function being defined as $k(x, x') = \phi(x)^T\phi(x')$.

Predictions on new data can then be made evaluating the sign in (18) where we can substitute $w$ using (24):

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b \tag{29}$$

Here due to the constraints for every data point either $a_n = 0$ or $t_n y(x_n) = 1$ so that all only points on the margin will remain in the sum and these are hence called *support vectors.*

There are many extensions to the model with the most important one being the introduction of *slack variables* by [Cortes and Vapnik, 1995] in order to handle overlapping class distribution in the input space. Books dedicated to the topic such as [Shawe-Taylor and Cristianini, 2004] offer a thorough introduction into Kernel Methods and again the topic is covered in most introductory books to machine learning.

### 4.2.7 Neural Networks

The Neural Network (NN) algorithm, often simply referred to as a *Neural Network*, draws its inspiration and name form the way information processing in the brain is understood. Its biological analogy are neurons, nerve cells that can receive signals from other neurons through their input coannections called dendrites. A neuron can than react to this input by firing a signal to other neurons itself and this reacting depends on the strength of the incoming coannections which are adapted as an organism learns over time.

Modeling this behavior, a neural network is composed of multiple nodes, also referred to as *neurons*. Each node can receive several signals from other nodes, combine these signals by applying an *activation function* on their weighted sum and output a single signal as the result. The algorithm learns by adapting each single weight of this weighted sum for every node-to-node coannection.

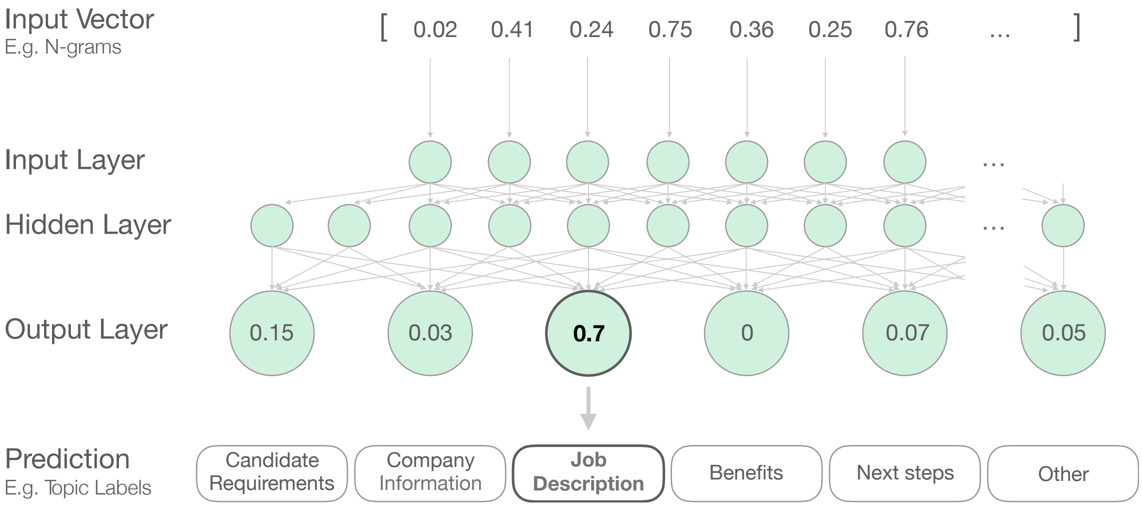


Figure 10: Conceptual visualization of a Feed-forward Neural Network

The most common form, known as *Feed-forward Neural Network*, is a network where all nodes of successive layers are coannected and each node in the network passes on its output signal to all nodes in the next layer until the transformed signal reaches the last layer. This layer represents the output signal or prediction and is thus called the *output layer*. The first layer is called *input layer* and is simply fed with real-valued vectors, e.g. as produced by a Vector Space method as described in Section 4.1. All layers in between are called *hidden layers*. An exemplified illustration of such a feed-forward neural network with one hidden layer can be seen in Figure 10.

For a single node the computation can formally be expressed as follows. We first compute the weighted sum of the inputs of the node:

$$a_j = \sum_i w_{ji} z_i \tag{30}$$

where $a_j$ is the weighted sum for node $j$, $w_{ji}$ is the weight for an input from node $i$ to node $j$ and $z_i$ is the output of node $i$. Then we apply the non-linear activation function $h(\cdot)$ to $a_j$ to receive the nodes output $z_j$:

$$z_j = h(a_j) \tag{31}$$

For training the network it is essential that the activation function $h$ is differentiable. This is because we use partial derivatives of the error function with regards to each weight to obtain the error this weight is produced and update it accordingly. We can achieve this by making use of the chain rule for partial derivatives which, given the error in one layer, lets us derive the errors for the previous layer. This procedure is called *backpropagation* since the error is propagated backwards through the network and will be described as follows.

First, using the chain rule, we can write the derivative of the error $E_n$ with respect to an incoming weight $w_{ji}$ as:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \tag{32}$$

where we use (30) and where $\delta_j$ denotes the *error* at node $j$, defined as:

$$\delta_j = \frac{\partial E_n}{\partial a_j} \tag{33}$$

Then for each output node $j$ given the sum-of-squares error we can directly calculate $\delta_j$ as follows:

$$\delta_j = y_j - t_j \tag{34}$$

where $y_j$ denotes the computed output of node $j$ and $t_j$ the expected output, also called the *target* value.

For hidden nodes we again apply the chain rule which yields

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = h'(a_j) \sum_k w_{kj} \delta_k \tag{35}$$

by making use of (33), (30) and (31) and the fact that the error in node $j$ is a result of the errors in the nodes $k$ in the following layer in direction of the output.

Starting with the $\delta$ of the output nodes we can recursively apply (35) to compute all $\delta$'s in the network which then allows to compute the derivatives with regards to each weight using (32).

1. We first perform a feed-forward pass, i.e. starting from the input nodes we propagate the signal through the network by successive application of weighted sum in (30) and the activation function in (31) until we receive output.
2. Then we compute the $\delta_k$ of the error for each output node with (34).
3. Next we can compute the $\delta$'s by backpropagating the errors using (35).
4. Lastly we can compute the derivatives for all weights using (32) and adapt the weights by a learning rate to reduce the error.

The study of Neural Networks comprises whole books and has been a subject of research for decades with strong fluctuations in popularity. Recently there has been much interest in this technique again, especially regarding in Deep Neural Networks, i.e. networks with many hidden layers where the network can learn layers of data representations with increasing abstraction. Various developments in research and technology, such as the available of massive parallel computational resources through Graphical Processing Units (GPUs) and learnings e.g. on the importance of initialization of the nodes in a network, have led to huge success in various application domains. Further, there are various extensions to the learning procedure such as regularization methods and alternative network architectures such as the Convolutional Neural Network (CNN). It is beyond the scope of this work to cover these topics in depth and the interested reader shall be referred to any standard machine learning literature for an introduction to the topic, such as [Bishop, 2006] from which this section's description was adapted. For a more recent treatment on the topic and on Deep Learning in particular [Bengio et al., 2015] and [Cho, 2014] offer a great overview. Various developments in research and technology, such as the available of massive parallel computational resources through GPUs and learnings e.g. on the importance of initialization of the nodes in a network, have led to huge success in various application domains. Further there are various extensions to the learning procedure such as regularization methods and alternative network architectures such as the CNN.

## 4.3 Sequential Models For Text Classification

In Section 4.1 we covered an approach to text classification where each text segment to be classified was first transformed into a fixed size vector representation and fed into a classifier afterwards. This approach treats each text segment as a document and assumes it belongs to a certain class.

An alternative approach is to model text as a sequential signal produced over time, similar to a time-series. This assumes strong dependencies between the current time step and previous time steps in the signal. We can take this view with different levels of granularity: The most fine-grained representation of text are characters so text can just be seen as a sequence of characters, but also using words or sentences as the atomic unit would be possible. As the space of possible letters in most western languages is quite limited we can directly use these as states that our signal can take. In the case of words or longer sequences we would either have a huge space of states where similarity of words would be disregarded or we could use a vector space representations for words or sequences.

There are various algorithmic approaches to model time-series and sequences of evolving states such as Linear Dynamic Systems or Hidden Markov Models. However many of these methods tend to “forget fast”, i.e. they are unable to model long-term dependencies. An extreme case are HMM's where the next state only depends on the current state. Alternatively we will look at Recurrent Neural Networks in particular which are attractive for their ability to model long-term dependencies.

### Recurrent Neural Networks

Recurrent Neural Networks are networks with loops, meaning that in contrast to Feedforward Neural Networks they can have connections feedback connections from a neuron's output back into it's input. This allows the network to pass information between time steps. Conceptually this can be thought of as a neural network copied $T$ times along the depth axis where the output of nodes $k_t$ at time step $t$ is connected to the input of the same node in the successive copy $k_{t+1}$ at time step $t+1$. This process of turning a network with feedback loops into a three dimensional feedforward network with connections between the time steps is called *unrolling*.

It also provides a way for training the network where the standard back-propagation algorithm for Feedforward Neural Networks (see Section 4.2.7) can be extended to incorporate these dependencies. This procedure is then called *Back-propagtion through time (BPTT)* and was introduced by [Werbos, 1988] and others independently. A visualization of such an unrolled network for a single node is shown in Figure 11.

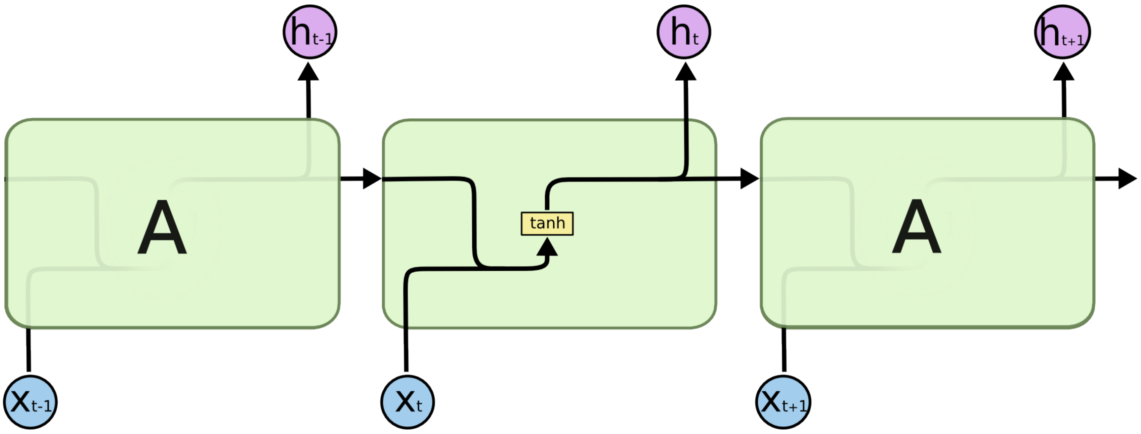


Figure 11: An unit in a recursive neural network, unrolled over time. Source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

### Long Short-Term Memory (LSTM) Networks

While Recurrent Neural Networks can model longer dependencies than e.g. Hidden Markov Models in theory, long-term dependencies are often still difficult to learn for such models. The reason is that when training the network with back-propagation, the gradient error signals are multiplied over many time steps, weakening the signal. This weakness is known as the *vanishing gradient problem* which was formalized and studied in depth by [Hochreiter, 1991]. The opposite effect is known as *exploding gradients* where the gradient signal is so large that learning can diverge. Based on these insights [Hochreiter and Schmidhuber, 1997] then introduced a new recurrent network design specifically to overcome this shortcoming. This network architecture is called Long-Term Short Memory Network (LSTM) and has seen tremendous interest recently due to its effectiveness for learning sequential and temporal structure. In many popular application domains in machine learning it achieves state-of-the-art results, including sequence to sequence language translation, handwriting recognition and generative sequence modeling (see e.g. [Greff et al., 2015] where popular variants of the architecture are compared and applied to a set of problems). Similar to the developments in Deep Learning, much of the recent successes have just recently been made possible with technological advancement of concurrent computational hardware, specifically GPUs, as recurrent neural networks are very expensive to train in terms of computational resources due to their complexity.

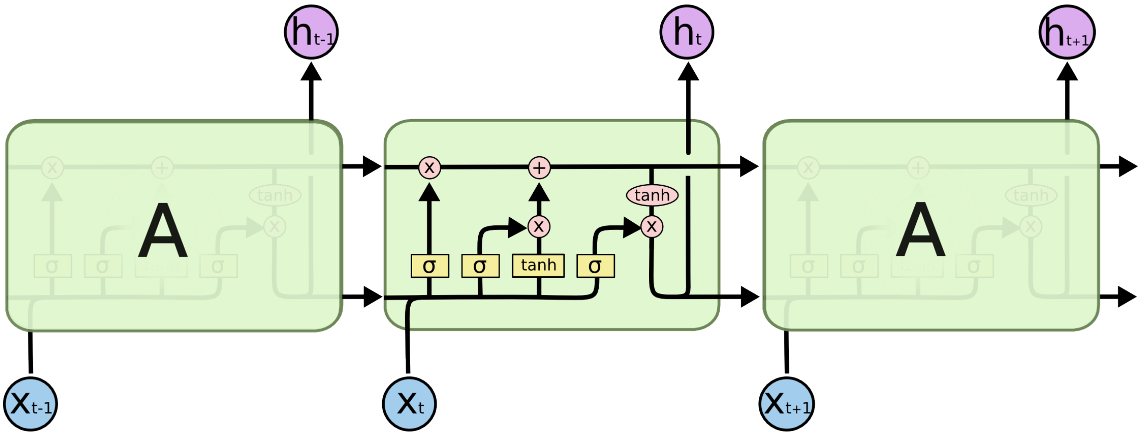


Figure 12: An unrolled LSTM node, showing the architecture. Source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

The specific architecture can be seen in Figure 12, again unrolled over time as above. A node in an LSTM network introduces a few modifications to a standard cell in a traditional RNN. The current cell state is passed through to the next time step by default. This signal can be intercepted by a number of so-called *gates* depending on the previous cell state, the previous cell output and the current data input. The first of such interactions is the *forget gate* that was in fact a later addition to the model introduced in [Gers et al., 1999] but is now considered part of the standard LSTM model. The forget gate is a sigmoid function receiving the previous cell output and the current data input and can influence how much of the cell state is kept or “forgotten”. Next, the *input gate* with a sigmoid function determines which values will be updated in the cell and another tanh activation node selects candidates for the update based on the previous cell output, replacing the current cell state. Lastly the cell outputs the updated cell state and the output based on what the input gate let through.

As mentioned earlier there are many variants, extensions and simplifications of this model. The previously mentioned study by [Greff et al., 2015] offers a good overview as well as a comparative study by [Zaremba, 2015]. For a comprehensive introduction to Recurrent Neural Networks the reader shall be referred to [Graves, 2012].

## 4.4 Evaluation Metrics

In this section the basics of evaluating classification models for the given problem will be laid out. First the different evaluation schemes and their advantages or disadvantages are explained in the dichotomous case where only one class is to be predicted in terms of being active or not. Then these are generalized to the multi-label case where $K$ mutually exclusive classes are given.

### Binary Classification

In the binary case of classification we are given a single class $k$ and a set of labelled data points $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where targets $y_i \in \{0, 1\}$ encode whether a data point $x_i$ belongs the class $c$ or not. The task is then to achieve correct classification of new data points without knowing the true label via a model function or predictor $f(\cdot)$.

To evaluate such a predictor it is useful to present the results in the form of a contingency table as shown in Table 2, because it gives valuable insights about the performance of the predictor. The table shows the proportion of data points that belong to the class (RP) or not (RN) and were predicted correctly (TP) or incorrectly (FN), as well as the number of data samples that do not belong to the class (RN) and were falsely predicted to be in the class (FP) or correctly predicted to not be in the class (TN), and the same proportions for the positively (PP) and negatively (PN) predicted cases with respect to the true assignments to the data. N refers to the total amount of data points.

| | Real Positives (RP) | Real Negatives (RN) |
|---:|:---:|:---:|
| Predicted Positives (PP) | True Positives (TP) | False Positives (FP) |
| Predicted Negatives (PN) | False Negatives (FN) | True Negatives (TN) |

Table 2: Contingency table for binary classification

**Accuracy** An intuitive choice towards classification is to simply ask which data points were correctly classified to belong to the class or not. In terms of the contingency table above the ratio of (TP + FP)/(N), commonly referred to as the “accuracy” of the classifier.

This choice can give a good intuition and it does capture the effectiveness on both true positives as well as true negatives, but it is strongly influenced by bias of the true and predicted class distribution (known as prevalence RP/N and label bias) as pointed out by [Powers, 2011]. For example given a population of 900 positive and 100 negative examples, a predictor that simply always chooses a positive assignment can achieve accuracy of 90% while it obviously is not a great predictor.

“There is a good reason why accuracy is not an appropriate measure for information retrieval problems. In almost all circumstances, the data is extremely skewed: normally over 99.9% of the documents are in the nonrelevant category. A system tuned to maximize accuracy can appear to perform well by simply deeming all documents nonrelevant to all queries. Even if the system is quite good, trying to label some documents as relevant will almost always lead to a high rate of false positives. However, labeling all documents as nonrelevant is completely unsatisfying to an information retrieval system user. "[Manning et al., 2008, Chapter 8.3, p. 155]

**Precision, Recall and F1 Score** In the field of Information Retrieval it is common practice to measure the effectiveness of a predictive system in terms of its precision and recall. The precision of such system is "the proportion of retrieved material that is actually relevant" whereas the recall measures "proportion of relevant material actually retrieved in answer to a search request" [Rijsbergen, 1979]. Formally these two measures are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{36}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{37}$$

As both, high precision and recall, are important for robust information retrieval system. They are typically combined into a single measure such as the F-measure, also referred to as F-score. The F-score is the weighted harmonic mean between precision and recall, derived from the measure of effectiveness proposed in [Rijsbergen, 1979]. The most common form is the $F_1$ score where precision and recall are assigned equal weight:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{38}$$

The $F_1$ score has the advantage of its intuitive interpretability as both precision and recall are well understood measures and, analogous to recall, precision and accuracy, as it lives in the range $[0, 1]$, giving a single number that can express the effectiveness of the system in terms of percentage.

The F1 score is widely used in the field of Machine Learning and Data Mining and thus it is an important measure to consider for comparing the results to outcomes of prior publications by others. It is however important to point out that any version of the F-measure is a biased score as it "ignores TN which can vary freely without affecting the statistic" [Powers, 2011]. This can affect the evaluation of a classifier when the class distribution is skewed (prevalence) or the classifier develops a bias towards certain classes (label bias), motivating the use of unbiased measures in these cases, such as the ones described next.

**Informedness, Markedness and Matthews Correlation Coefficient** [Powers, 2011] introduces unbiased analogue measures to Recall and Precision, called "Informedness" and "Markedness" respectively. As [Powers, 2011] lays out, "Informedness quantifies how informed a predictor is for the specified condition, and specifies the probability that a prediction is informed in relation to the condition (versus chance).":

$$\begin{aligned} \text{Informedness} &= \text{Recall} + \text{Inverse Recall} - 1 \\ &= \frac{\text{TP}}{\text{RP}} + \frac{\text{TN}}{\text{RN}} - 1 \end{aligned} \tag{39}$$

Further he defines: "Markedness quantifies how marked a condition is for the specified predictor, and specifies the probability that a condition is marked by the predictor (versus chance)."

$$\begin{aligned}\text{Markedness} &= \text{Precision} + \text{Inverse Precision} - 1\\ &= \frac{\text{TP}}{\text{PP}} + \frac{\text{TN}}{\text{PN}} - 1\end{aligned} \tag{40}$$

Based on Informedness and Markedness we can then see that *Matthews Correlation Coefficient* $r_G$, first proposed by [Matthews, 1975], is a score that balances these two measures:

$$\begin{aligned} r_G &= \pm\sqrt{\text{Informedness} \cdot \text{Markedness}}\\ &= \frac{(\text{TP}\cdot\text{TN} - \text{FP}\cdot\text{FN})}{(\text{TP}+\text{FN})(\text{FP}+\text{TN})(\text{TP}+\text{FP})(\text{FN}+\text{TN})}\end{aligned} \tag{41}$$

Matthews Correlation Coefficient can thus be used as unbiased alternative to the F-measure and offers a similar ease of interpretability as it ranges from -1 to 1, the former indicating a negative correlation or adverse estimation and the latter indicating a perfect prediction, while a coefficient of 0 reflects chance.

**Cross-Entropy** Another common way to evaluate classifiers is the *cross-entropy* loss function:

$$\mathbb{H}(p,q) = -\sum_{n}^{N} p_n \log q_n \tag{42}$$

where $p$ and $q$ are discrete probability distributions. The *cross-entropy* can be derived from the *KL-divergence* as in [Murphy, 2012, Chapter 2.8.2, p. 57]:

$$\begin{aligned}\mathbb{KL}(p,q) &= \sum_{n}^{N} p_n \log \frac{p_n}{q_n}\\ &= \sum_{n}^{N} p_n \log p_n - \sum_{n}^{N} p_n \log q_n\\ &= -\mathbb{H}(p) + \mathbb{H}(p,q)\end{aligned} \tag{43}$$

where $\mathbb{H}(p)$ is the regular entropy, i.e. the lower bound on the number of bits needed to transmit the state of a random variable (as in [Shannon, 2001]), and $\mathbb{H}(p,q)$ is the cross-entropy, i.e. "the average number of bits needed to encode data coming from a source distribution $p$ when we use model $q$ to define our codebook" [Murphy, 2012, Chapter 2.8.2, p. 57].

In the case of binary classification we can rewrite the cross-entropy into the following error or loss function of the learned weight vector:

$$E(\mathbf{w}) = -\log p(\mathbf{T} \mid \mathbf{w}) = -\sum_{n=1}^{N} t_n \log y_n + (1-t_n)\log(1-y_n) \tag{44}$$

where $y_n$ denotes $y(x_n, \mathbf{w})$, the predicted output for datapoint $x_n$, $t_n$ denotes the $n$-th true label and **w** denotes the trained weight vector of the model, as in [Bishop, 2006, Chapter 4.3.2, p. 205 ]. This form is also known as the *log loss* and it is commonly used with generalized linear models and NNs (see e.g. [Bishop, 2006, Chapter 4.3.2, p. 205 ] and [Alpaydin, 2014, Chapter 10.7, p. 251 ]).

Thus, cross-entropy is a measure which is well-motivated from a information-theoretic perspective. On the downside it does not have an upper bound which makes it hard to interpret, as compared other scores that fall into $[0, 1]$ or similar intervals.

### Multi-class Classification

Multi-class classification refers to a generalization of the binary case where we aim to predict for each datapoint $wowx_i$ one of $K$ labels for the classes at hand. The target space $\mathcal{Y}$ can be represented with each $y_i \in \{0, 1\}^k$, known as *one-hot encoding*, where each target is $c$-dimensional vector. Alternatively we can encode the targets as categorical variables $y_i \in c_1, c_2, \ldots, c_k$. The contingency table from the binary case can be extended as in table 3, which is then commonly known as Confusion Matrix or Error Matrix.

| | Real Class 1 | Real Class 2 | ... | Real Class $k$ |
|---|---|---|---|---|
| Predicted Class 1 | ... | ... | | ... |
| Predicted Class 2 | ... | ... | | ... |
| ... | | | | |
| Predicted Class $k$ | ... | ... | | ... |

Table 3: Contingency table for $k$ classes, also referred to as Confusion Matrix

**Averaging for Multi-class Recall, Precision and F1-Score** By definition, Recall, Precision and thus also the F-measure are defined for the dichotomous classification case, however they can be extended towards multiple classes by averaging. Two common methods are described in [Manning et al., 2008, Chapter 13.6, p. 280]: “Macroaveraging computes a simple average over classes. Microaveraging pools per-document decisions across classes, and then computes an effectiveness measure on the pooled contingency table.” It is important to note that “macroaveraging gives equal weight to each class, whereas microaveraging gives equal weight to each per-document classification decision. Because the F1 measure ignores true negatives and its magnitude is mostly determined by the number of true positives, large classes dominate small classes in microaveraging.” [Manning et al., 2008, Chapter 13.6, p. 280]. Formally these averaging schemes can be defined as follows, with R denoting the Recall and P the Precision.

$$\mathrm{R}_{\text{micro}} = \frac{\sum_{k=1}^{K} \mathrm{TP}_k}{\sum_{k=1}^{K} \mathrm{TP}_k + \mathrm{FN}_k} \qquad \mathrm{R}_{\text{macro}} = \frac{\sum_{k=1}^{K} \mathrm{R}_k}{K} \tag{45}$$

$$\mathrm{P}_{\text{micro}} = \frac{\sum_{k=1}^{K} \mathrm{TP}_k}{\sum_{k=1}^{K} \mathrm{TP}_k + \mathrm{FP}_k} \qquad \mathrm{P}_{\text{macro}} = \frac{\sum_{k=1}^{K} \mathrm{P}_k}{K} \tag{46}$$

And respectively:

$$\mathrm{F}_{1\text{micro}} = 2 \cdot \frac{\mathrm{P}_{\text{micro}} \cdot \mathrm{R}_{\text{micro}}}{\mathrm{P}_{\text{micro}} + \mathrm{R}_{\text{micro}}} \qquad \mathrm{F}_{1\text{macro}} = 2 \cdot \frac{\mathrm{P}_{\text{macro}} \cdot \mathrm{R}_{\text{macro}}}{\mathrm{P}_{\text{macro}} + \mathrm{R}_{\text{macro}}} \tag{47}$$

**Matthews Correlation Coefficient for K classes** [Gorodkin, 2004] introduced a way to extend Matthews Correlation Coefficient to the multi-class case using a generalization of Pearson's Correlation Coefficient. The coefficient is then defined as:

$$R_k = \frac{\mathrm{COV}(X, Y)}{\sqrt{\mathrm{COV}(X, X)\ \mathrm{COV}(Y, Y)}} \tag{48}$$

Where COV is the covariance function:

$$\mathrm{COV}(X, Y) = \sum_{k=1}^{K} w_k \mathrm{COV}(X_k, Y_k) \tag{49}$$

$$= \frac{1}{K} \sum_{n=1}^{N} \sum_{k=1}^{K} (X_{nk} - \overline{X_k})(Y_{nk} - \overline{Y_k}) \tag{50}$$

Similar extensions have been proposed, such as the Confusion Entropy (CEN) as described in [Jurman and Furlanello, 2012]. The article concludes:

> Confusion Entropy [. . . ] is probably the finest measure and it shows an extremely high level of discriminancy even between very similar confusion matrices. However, this feature is not always welcomed, because it makes the interpre- tation of its value quite harder, expecially when considering sit- uations that are naturally very similar (e.g, all the cases with MCC=0). Moreover, CEN may show erratic behaviour in the binary case.
>
> In this spirit, the Matthews Correlation Coefficient is a good compromise between reaching a reasonable discriminancy degree among different cases, and the need for the practitioner of a easily interpretable value expressing the type of misclassification associated to the chosen classifier on the given task. We showed here that there is a strong linear relation between CEN and a logarithmic function of MCC regardless of the dimension of the considered problem. Furthermore, MCC behaviour is totally consistent also for the binary case.
>
> This given, we can suggest MCC as the best off-the-shelf evaluating tool for general purpose tasks, while more subtle measures such as CEN should be reserved for specific topic where more refined discrimination is crucial.

Thus Matthews Correlation Coefficient is the preferred measure when possible.

**Categorical Cross-Entropy** The *cross-entropy* loss function as defined above in Section 4.4 extends to the multi-class case quite naturally:

$$E(\mathbf{w}_1, \ldots, \mathbf{w}_k) = -\ln p(\mathbf{T} \mid \mathbf{w}_1, \ldots, \mathbf{w}_k) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk} \tag{51}$$

where $y_{nk} = y_k(\phi_n)$, and $\mathbf{T}$ is an $N \times K$ matrix of target variables with elements $t_{nk}$ (see as in [Bishop, 2006, Chapter 4.3.4, p. 209 ]). This form is also referred to as *multi-class log loss* and gives an aggregated loss over all classes.

# 5 Experiments and Results

With the final problem definition and dataset in place a series of experiments were conducted to evaluate the performance of the different approaches explained in Section 4. This part of the thesis lays out these experiments and their results. First the research objectives will be reiterated. Next the experimental setup for each of the methods will be described and the outcomes and observations will be presented.

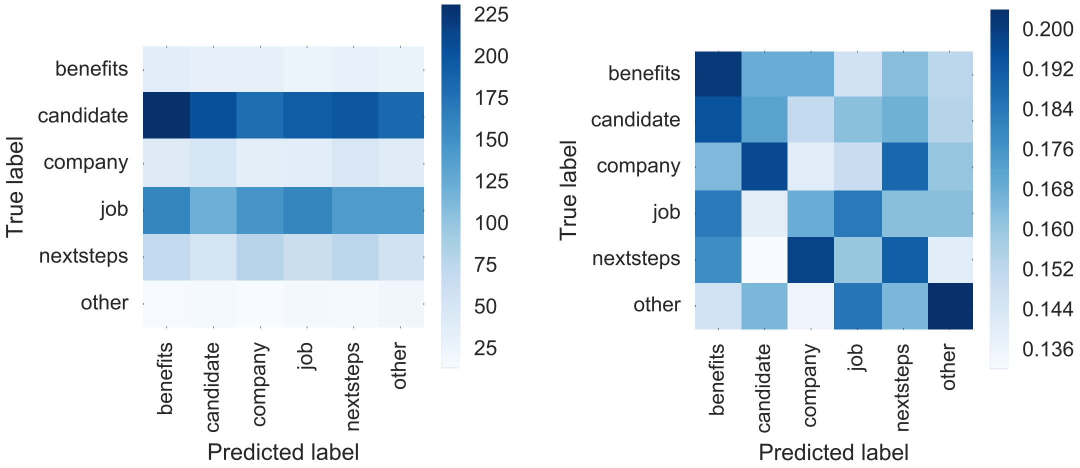
## 5.1 Objectives

The experiments had simple objectives: The goal was to evaluate each method in terms of its effectiveness using a prediction metric well-suited for this problem. The metric used was MCC as described in Section 3.4. Effectiveness in particular meant comparing the algorithms with regards to their overall performance on test data and their ability to generalize well, i.e. to be robust against overfitting.

## 5.2 Baseline

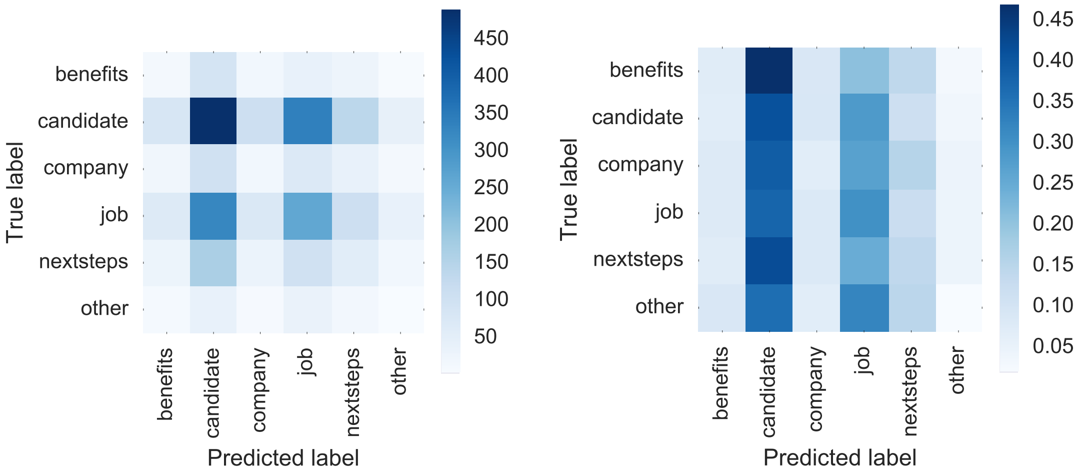
In order to have a reference point for predictive performance that any method should surpass two guessing strategies were used, namely uniform and stratified guessing. Uniform guessing means sampling from a uniform distribution, commonly known as "rolling dice", while stratified guessing refers to an estimator that samples a label for a data point using the observed distribution of labels in the data. Both methods effectively ignore the data itself when producing labels and as such any predictive algorithm should do better.

Averaged over 1000 runs both strategies yielded an MCC score close to zero. Accuracy by comparison was around 0.16 for uniform guessing and 0.26 for stratified guessing. These results are within our expectations and further highlight the rationale for choosing MCC as the principal metric: MCC is zero for uninformed predictions for either strategy whereas the accuracy in the uniform setting corresponds simply to a value of $1/k$ where $k = 6$ is the number of labels. The accuracy in the stratified setting reveals improves by taking advantage of knowledge about the skew in the label distribution. Figure 13 shows the confusion matrices for these baseline variants in absolute and normalized form, where the described properties these guessing strategies can be observed.



(a) Uniform guessing

(b) Uniform guessing (normalized)



(c) Stratified guessing

(d) Stratified guessing (normalized)

Figure 13: Confusion matrices of uniform and stratified guessing strategies.

## 5.3 Classification With Vector Space Models

A popular way to approach text classification and other tasks in natural language processing is to build a model that maps data into a vector space. Distance between data points in this space then translates to similarity between the objects (see Section 4.1). The resulting vector representation of the data can then be fed into various learning algorithms. This section describes the experiments performed to evaluate such approaches.

First the different methods to produce such vector spaces are compared. Several methods were used to generate vectors from the data while limiting dimensionality to 300 for comparability — a heuristically chosen value as performance for the models did not increase significantly beyond it. These vector representations were then compared in terms of performance by using them as input to the simple classification model Logistic Regression. Second, the best performing configurations for each type of method were chosen and a set of various classification techniques were applied to them which were described previously in Section 4.2.

### 5.3.1 N-gram Models

The first class of language models that was investigated for the task of multi-class classification are N-gram models that were explained in Section 4.1.1. N-gram models come in a variety of forms. In these experiments the most common variants were set up as hyper-parameters to the model as listed in Table 4.

| Hyper-Parameter | N-gram Type: Words | N-gram Type: Characters |
|---|---|---|
| N-gram Range (Range) | [1,1], [1,2], [1,3], [2,3], [3,3] | [1,5], [1,10], [5,10], [5,15] |
| Stop Words | English, None | — |
| Vector Size (Size) | 10, 100, 300 | 10, 100, 300 |
| IDF | Yes, No | Yes, No |
| Norm | L1, L2, None | L1, L2, None |
| Sub-linear TF | Yes, No | Yes, No |

Table 4: Parameter search space for word and character level N-gram models

A grid search was performed to test all combinations of configurations within this hyper-parameter space. Each configuration was evaluated with regards to its MCC score using 5-fold cross-validation on the training data with three standard classifiers: Logistic Regression and Naive Bayes and SVM. Table 5 shows the five best results of the exhaustive grid search over the hyper-parameter configurations.

The effects of the different hyper-parameters on performance can be observed based on these results. Regarding the N-gram *type*, words as the atomic unit for N-grams consistently led to better results. This is due to the fact that the search space of combinations of characters is significantly larger than the search space of known words, so model complexity increases exponentially. With regards to the

| Type | Range | Stop words | Size | IDF | Norm | Sub-linear TF | MCC |
|---|---|---|---|---|---|---|---|
| Word | [1,1] | None | 300 | Yes | | Yes | 0.689 |
| Word | [1,1] | None | 300 | Yes | | No | 0.687 |
| Word | [1,1] | None | 300 | No | | Yes | 0.682 |
| Word | [1,1] | None | 300 | No | | No | 0.682 |
| Word | [1,1] | None | 300 | Yes | L2 | Yes | 0.68 |
| Word | [1,1] | None | 300 | No | | Yes | 0.659 |
| Word | [1,1] | None | 300 | No | | No | 0.656 |
| Word | [1,2] | None | 300 | No | | Yes | 0.655 |
| Word | [1,2] | None | 300 | No | | No | 0.655 |
| Word | [1,3] | None | 300 | No | | No | 0.65 |
| Word | [1,1] | None | 300 | Yes | | Yes | 0.689 |
| Word | [1,1] | None | 300 | Yes | | No | 0.689 |
| Word | [1,2] | None | 300 | Yes | | Yes | 0.677 |
| Word | [1,2] | None | 300 | Yes | | No | 0.677 |
| Word | [1,3] | None | 300 | Yes | | Yes | 0.674 |

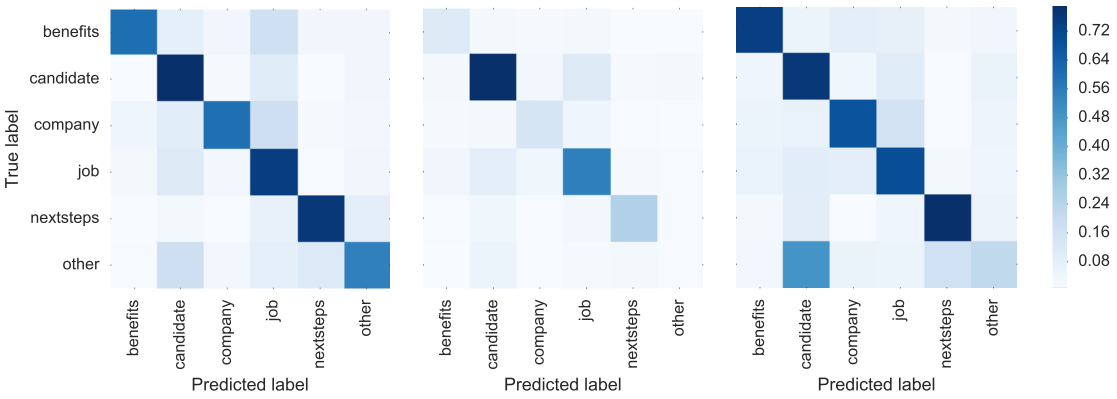
Table 5: Top 5 results of grid search over hyper-parameter space using 5-fold cross-validation on the training set with Logistic Regression (top), Naive Bayes (middle) and SVM (bottom).

*range*, i.e. the interval for possible $N$ in N-grams, there are slight differences to be observed between the three classifiers used, but with all three models the best performance is achieved using Unigrams. Also all of the top results across all classifiers include Unigrams in the model while some extend the range towards bigrams or trigrams. None of the top results of the performed grid searches used *stop words*. This is interesting as using stop-words to remove hand-picked, highly frequent words that do not carry much meaning is common practice. When it comes to the *size* of vectors, for the given settings the highest vector dimensionality of 300 achieves the best performance. There is no consensus between the classifiers on whether or not to weigh the N-gram frequencies by the inverse document frequency (*IDF*, see Section 4.1.1). With respect to the *norm* used, in these experiments normalizing vectors decreased performance. Only in the fifth best performing configuration when using Logistic Regression vectors were normalized, in this case using the L2 norm. Lastly applying sub-linear term frequency scaling (*Sub-linear TF*, see Section 4.1.1) did not seem to affect the results significantly and about half of the top results were obtained using this technique.

Based on these observations the best model for each classifier with regards to MCC validation score was chosen and trained on the whole training data and tested on the test data set to estimate the final performance. Table 6 shows the scores of each classifier using these best N-gram models. It is evident that here logistic regression performs best, achieving the highest MCC score as well as good accuracy.

| | Training | | Test | |
|---|---|---|---|---|
| Classifier | Accuracy | MCC | Accuracy | MCC |
| Logistic Regression | 0.824 | 0.761 | 0.787 | 0.708 |
| Naive Bayes | 0.769 | 0.681 | 0.767 | 0.677 |
| SVM | 0.835 | 0.681 | 0.786 | 0.700 |

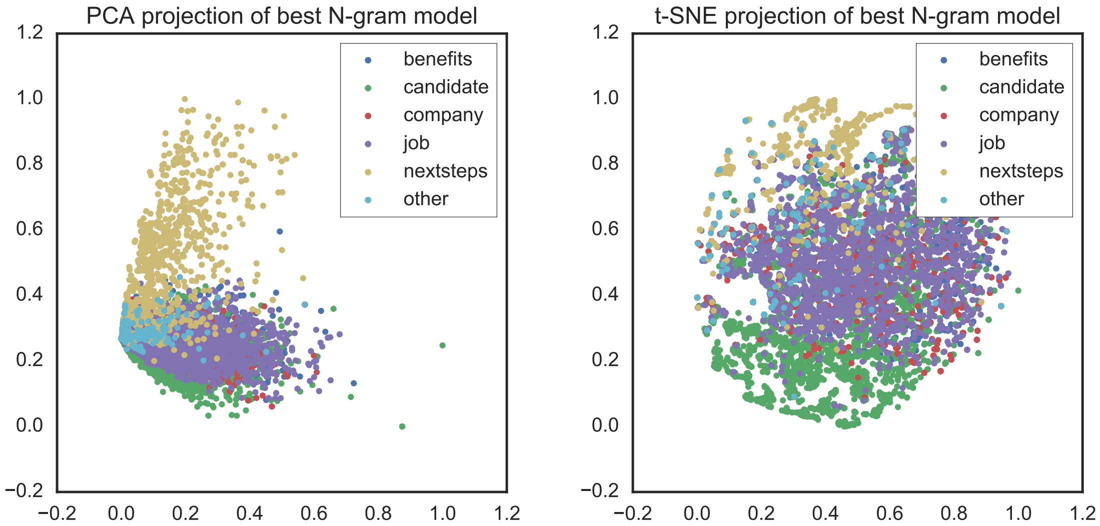
Table 6: Performance of each best N-gram model with Logistic Regression and Naive Bayes on the test data.



(a) Logistic Regression (b) Naive Bayes (c) SVM

Figure 14: Normalized confusion matrices all three classifiers using the best N-gram model found via cross-validated grid search. Both Naive Bayes as well as SVM show label bias towards the prevalent class *candidate*.

To understand the mapping of the data in the resulting vector space visualizations were produced using Principal Components Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE). The results for best-performing N-gram model can be seen in Figure 15. Especially the PCA visualization shows that a high percentage of the data for each class can be separated from other classes even with a linear model.



(a) PCA projection

(b) t-SNE projection

Figure 15: Document vectors produced by the best N-gram model (optimized w.r.t. Logistic Regression) projected onto the first 2 principal components (left) and project using t-SNE projection.

### 5.3.2 Bag-of-Means: An Averaged Word2Vec Model

Next a Bag-of-Means model as described in Section 4.1.2 was evaluated with the same set of classifiers. The model was evaluated on the same test and training data split as used for the N-gram model above. The results are shown in Table 7.

| | Training | | Test | |
|---|---|---|---|---|
| Classifier | Accuracy | MCC | Accuracy | MCC |
| Logistic Regression | 0.797 | 0.722 | 0.784 | 0.702 |
| Naive Bayes | 0.337 | 0.271 | 0.320 | 0.251 |
| SVM | 0.545 | 0.356 | 0.562 | 0.379 |

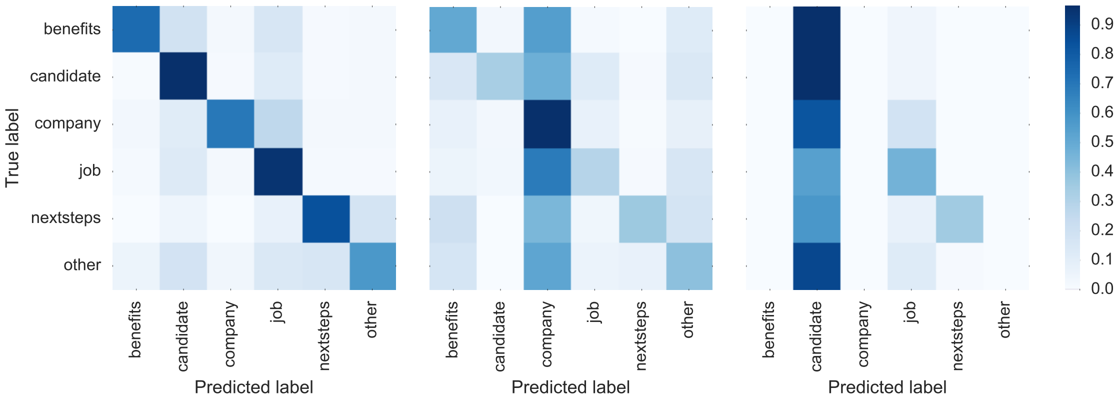
Table 7: Performance base classifiers using the Bag-of-Means model

As a basis, pre-trained word vectors from the Google News dataset[9] were extracted

---

[9] The dataset contains contains 300-dimensional vectors for 3 million words and phrases. The

for the words that occur in the dataset. Then for each sentence in the data a sentence vector was obtained by taking the arithmetic mean over the vectors for all words in the sentence. Labels were again predicted using Logistic Regression, Naive Bayes and SVM. We can see that the model performs well using Logistic Regression and it is almost on par with the best N-gram model. On the other hand the variance in results between the classifiers is drastic, and Naive Bayes' score is 0.451 lower that Logistic Regression in absolute terms. The confusion matrices in Figure 16 reveal strong label bias in the case of Naive Bayes and SVM.



(a) Logistic Regression (b) Naive Bayes (c) SVM

Figure 16: Normalized confusion matrices of all three classifiers using the Bag-of-Means model.

The visualization in Figure 17 shows a projection of the vectors into a 2 dimensional space. We can visually confirm that labels tend to cluster in this space.

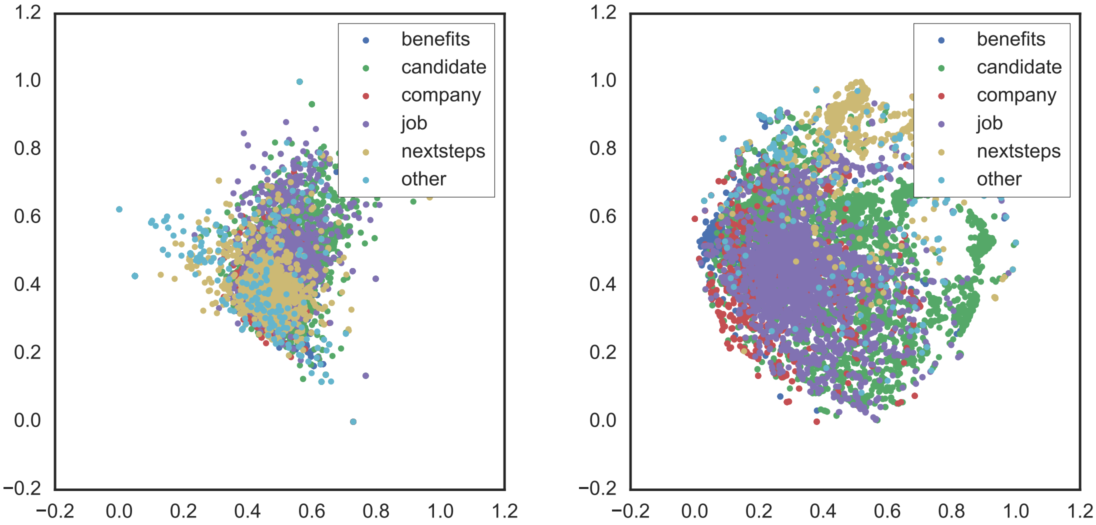
### 5.3.3 Paragraph Vectors using Distributed Representations

Next a vector space model was build using the approach that was proposed by [Le and Mikolov, 2014] and described in detail in Section 4.1.2. There are several variants and hyper-parameters to this model and their effect was experimentally evaluated. As this model is computationally quite expensive a grid search as for the N-gram model above was infeasible. Hence performance effects were measured by varying the hyper-parameters one at a time while keeping the others fixed. In this process Logistic Regression was used with 5-fold cross-validation. The default configuration of hyper-parameters can be seen in Table 8 below.

The results of the evaluation experiments of the hyper-parameters can be seen in Table 9. Similarly to the N-Gram models the *vector size* correlates positively with the performance. Again the highest chosen dimensionality was 300 which yielded the highest test score of 0.608. However the difference to a 100-dimensional model is marginal with 2% absolute loss in performance and a 10-dimensional representation

---

phrases were obtained using a simple data-driven approach described in [Mikolov et al., 2013b]. The dataset can be obtained on the following website: `https://code.google.com/archive/p/word2vec/`



(a) PCA projection

(b) t-SNE projection

Figure 17: Document vectors produced by Bag-of-Means model (optimized w.r.t. Logistic Regression) projected onto the first 2 principal components (left) and projected using t-SNE projection.

| | |
|---|---|
| Vector Size | 100 |
| Sub-sampling threshold | No |
| Hierarchical Softmax | Yes |
| Negative Sampling Value | 3 |
| Window Size | 10 |
| Model Type | PV-DBOW |
| Training Type | Inferred Vectors |

Table 8: Default hyper-parameter configuration for Paragraph Vectors

only leads to an absolute loss of 6% in performance. Further choosing the representation in only two dimensions yields a test score of 0.223. In comparison the best N-gram model achieved an MCC score of 0.151 when limited to two dimensions.

With regards to frequent word *sub-sampling* performance was best without its use. Note that this is in contrast with previous work on word vectors where this setting increased performance (see [Mikolov et al., 2013b]). In contrast when training the model using the PV-DM architecture instead, frequent word sub-sampling increased the training score, though in this mode the test score was significantly lower as we will see below. Using *hierarchical softmax* increased the performance, although only leading to a marginal improvement of around 1% in terms of MCC score. *Negative sampling* generally increased performance. However no trend is apparent as the best performing configurations were a sampling value of 2 and 6 while choosing a value of 4 lead to a slightly lower score. The highest score amongst the tested settings for the

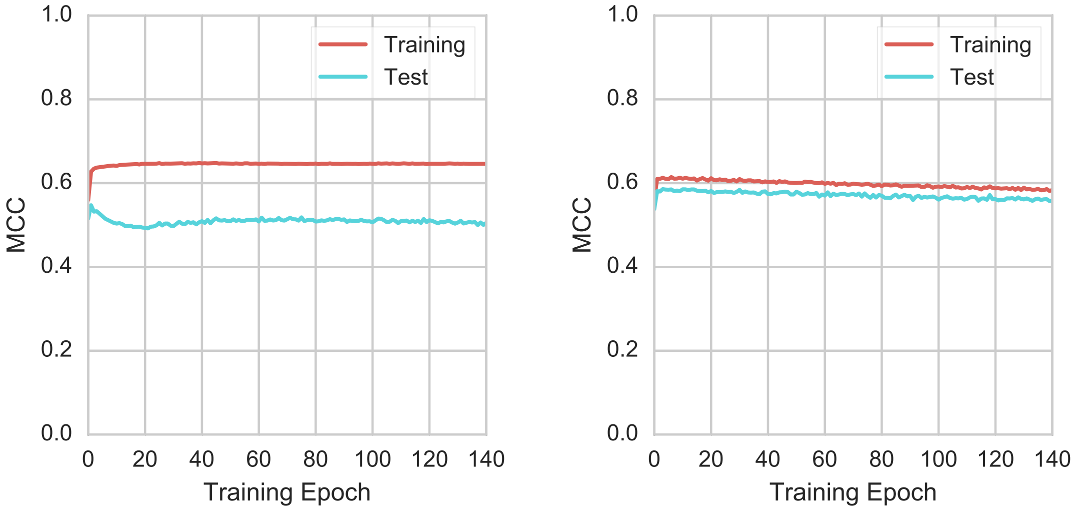
| Hyper-Parameter | Setting | Training Score | Test Score |
|---|---|---|---|
| Vector Size | 2 | 0.229 | 0.223 |
| | 10 | 0.545 | 0.541 |
| | 100 | 0.614 | 0.589 |
| | 300 | 0.648 | **0.608** |
| Sub-sampling Threshold | None | 0.611 | **0.588** |
| | 1e-4 | 0.495 | 0.475 |
| | 1e-5 | 0.328 | 0.303 |
| | 1e-6 | 0.149 | 0.127 |
| Hierarchical Softmax | Not used | 0.600 | 0.578 |
| | Used | 0.613 | **0.586** |
| Negative Sampling Value | 0 | 0.598 | 0.575 |
| | 2 | 0.612 | **0.591** |
| | 4 | 0.612 | 0.587 |
| | 6 | 0.613 | 0.590 |
| Window Size | 5 | 0.611 | 0.586 |
| | 10 | 0.614 | **0.588** |
| | 15 | 0.612 | 0.586 |
| Model Type | PV-DBOW | 0.610 | **0.580** |
| | PV-DM | 0.405 | 0.411 |
| Training Type | Trained Vectors | 0.519 | 0.366 |
| | Inferred Vectors | 0.404 | **0.408** |

Table 9: Test and Training scores measured using MCC with the different hyper-parameter settings. In all configurations only one hyper-parameter was adjusted while keeping the others as shown in Table 8

window size was achieved with a value of 10 while both a window of 5 and a window of 15 words achieved a lower score.

Both *model types* for paragraph vectors proposed in [Le and Mikolov, 2014] were tested, namely the Distributed Bag of Words Model of Paragraph Vectors (PV-DBOW) and the Distributed Memory Model of Paragraph Vectors (PV-DM). The choice of the model type among has a significant influence on the result. As we can see in Table 9 the MCC test score of the PV-DBOW model is 16.9% higher in absolute terms.

The choice of vectors to train the classifier, i.e. the *training type*, can be seen to have a strong impact on the results as well. When using directly the vectors that are trained with the Paragraph Vector model, performance is notably lower than when inferring new vectors for the training data using the inference procedure described in [Le and Mikolov, 2014]. Furthermore when using the model vectors for training the model easily overfits. This effect can be seen in Figure 18 where the training and test scores are shown over 140 iterations of training. While using the model vectors leads to higher training scores the test scores are significantly lower than when inferring vectors for training.



(a) Training using model vectors (b) Training using inferred vectors

Figure 18: Comparison of performance when trained the classifier on the trained vectors of the Paragraph Vector model or when inferring new Paragraph Vector vectors for the training data using the model.

Lastly the best configurations of hyper-parameters of these approaches were compared in another experiment. Table 10 lists the tested settings.

| | |
|---|---|
| Vector Size | 300 |
| Sub-sampling threshold | No |
| Hierarchical Softmax | { No, Yes } |
| Negative Sampling Value | { 2, 6, 10 } |
| Window Size | { 8, 10 } |
| Model Type | PV-DBOW |
| Inference Type | Inferred Vectors |

Table 10: Tested Hyper-parameter configurations to find the best Paragraph Vectors model

While a MCC test score of 0.574 the lowest amongst the candidates the highest score achieved was 0.618. This setting used a window size of 8, a negative sampling value of 10 and hierarchical softmax.

### 5.3.4 Evaluation of Classification Methods

**Experimental Setup** From the previous experiments on vector space models the best performing setting of each type was chosen, specifically:

- **N-Grams**: Unigrams were used with a dimensionality of 300. No stop words filtering and no vector normalization were applied and while using sub-linear TF and IDF.
- **Bag-Of-Means**: As in the experiments above the approach was simply used with an unweighed arithmetic mean of the word vectors in each sentence.
- **Paragraph Vectors**: PV-DBOW word vectors were generated using inference on the training data, a window size of 8, a negative sampling value of 10, hierarchical softmax and no frequent word sub-sampling.

Next a set of established classification methods representative of the algorithm classes described in Section 4.2 were applied to the data and evaluated.

The dataset was then split into 5 folds to perform Cross-Validation. For the results to be reproducible, comparable and to save computational resources, the vector representations of the three types were precomputed for all folds. The full results can be seen in Table 12 while Table 11 shows the classifiers' best results sorted by their test score:

| Classifier | Vector Space | Test Score |
|---|---|---|
| Neural Network | Bag-of-Means | 0.716 |
| Deep Neural Network | Bag-of-Means | 0.715 |
| SVM | N-Grams | 0.713 |
| Logistic Regression | Bag-of-Means | 0.706 |
| Random Forest | N-Grams | 0.701 |
| kNN | Bag-of-Means | 0.685 |
| Naive Bayes | N-Grams | 0.648 |
| Decision Tree | N-Grams | 0.625 |

Table 11: The best results in terms of test score for all classifiers.

**Test Performance** In terms of test performance, the most important measure when considering the effectiveness of a classification method, the best performing candidate is a single hidden layer Neural Network. On the other hand we can clearly see that there is only a small margin between the top scoring methods and the multi-layer Deep Neural Network as well as SVM's perform virtually at the same level. Also Logistic Regression and Random Forests show comparable performance while kNN, Naive Bayes and Decision Trees can be seen to have a larger gap.

| Classifier | Vector Space Model | Training Score | Test Score |
|---|---|---|---|
| Logistic Regression | N-Grams | 0.770 | 0.697 |
| | Bag-of-Means | 0.743 | **0.706** |
| | Paragraph Vectors | 0.709 | 0.643 |
| Decision Tree | N-Grams | 0.924 | **0.625** |
| | Bag-of-Means | 0.938 | 0.455 |
| | Paragraph Vectors | 0.994 | 0.290 |
| Naive Bayes | N-Grams | 0.668 | **0.648** |
| | Bag-of-Means | 0.498 | 0.491 |
| | Paragraph Vectors | 0.603 | 0.573 |
| SVM | N-Grams | 0.882 | **0.713** |
| | Bag-of-Means | — | — |
| | Paragraph Vectors | 0.831 | 0.683 |
| kNN | N-Grams | 0.929 | 0.642 |
| | Bag-of-Means | 0.944 | **0.685** |
| | Paragraph Vectors | 0.993 | 0.579 |
| Random Forest | N-Grams | 0.924 | **0.701** |
| | Bag-of-Means | 0.940 | 0.618 |
| | Paragraph Vectors | 0.985 | 0.509 |
| Neural Network | N-Grams | 0.911 | 0.709 |
| | Bag-of-Means | 0.930 | **0.716** |
| | Paragraph Vectors | 0.975 | 0.665 |
| Deep Neural Network | N-Grams | 0.913 | 0.708 |
| | Bag-of-Means | 0.936 | **0.715** |
| | Paragraph Vectors | 0.989 | 0.679 |

Table 12: Test and Training performance in terms of MCC for the whole set of classifiers. Each classifier was applied to transformed data using the best configuration for each of the three vector space model types. Note that the SVM classifier could not be evaluated in terms of MCC score in combination with the Bag-Of-Means vector space model due to unknown reasons.

Regarding vector space models we can see that the Paragraph Vectors did not achieve highest test performance in combination with any of the classifiers. Four of seven classifiers performed best with the Bag-of-Means model and also make up the highest two results while the N-Gram model worked best in the other cases. Table 12 shows that a set of methods achieve remarkable performance with simple N-Grams. Among them the SVM ranks highest with a MCC test score of 0.713. Conversely in many cases the training score using Paragraph Vectors is highest, including both the Tree-based methods, Neural Network approaches and kNN. Another important observation is that the Bag-of-Means model performs best overall and this is consistent half of the classifiers. The highest test score of 0.716 was achieved by a Neural Network using this model.

**Generalization Performance** Another property of classification techniques is their capability to generalize. An indicator for this ability is the ratio between performance on test and training data, since a classifier that simply "memorizes" training examples might achieve a very high training score but overfits and fails to classify new data well. Table 13 shows a comparison in terms of this ratio between both the classification algorithms as well as the vector space models when averaged over the performance with all algorithms.

| Method | Ratio |
|---:|---|
| Naive Bayes | 0.970 |
| Logistic Regression | 0.950 |
| SVM | 0.808 |
| Neural Network | 0.770 |
| Deep Neural Network | 0.764 |
| Random Forest | 0.759 |
| kNN | 0.726 |
| Decision Tree | 0.676 |
| N-Grams | 0.795 |
| Bag-Of-Means | 0.763 |
| Paragraph Vectors | 0.679 |

Table 13: Test to training score ratio for classifiers (top) using the best performing vector space model and vector space models (bottom) averaged over classifiers.

We can see that Naive Bayes and Logistic Regression achieve a very high ratio signaling good generalization. For the Neural Networks models tested it is apparent that the deep architecture overfits more than the network with a single hidden layer, i.e. it achieves higher training but lower test scores. When comparing the Deep Neural Network to Random Forests using Paragraph Vectors though we can see that while both methods achieve similarly high scores on the training data, Random Forests perform significantly worse on the test data — Random Forests achieve scores of 0.985 (training) and 0.509 (test) as opposed to the scores of 0.989 (training) and 0.679 (test) for the Deep Neural Network. Naive Bayes behaves very differently with each vector space model. When using N-Grams the model shows decent performance but the score drops with the other models, showing a decrease of performance of 7.5% with Paragraph Vectors and 15.7% with Bag-of-Means.

When looking at the vector space models Paragraph Vectors show a test to training score ration which is significantly lower than both other representations with by a margin of 10% in absolute terms. This means that on average the classifiers tend to overfit on this representation. N-grams on the other hand achieve the highest ration in these terms.

**Runtime Complexity** We can see quite significant variations between the computational requirements for the different methods as Table **??** shows. Computation of the N-gram model with a dataset of this size (9948 sentences) can be performed in under one second for all five folds. In comparison training the Paragraph Vector model took around 5,5 hours on the same machine. Note that regarding the Bag-Of-Means model the results are not directly comparable: Although computation of this model was fast, this does not account for the fact that it uses pre-computed word embeddings whose computation is almost as complex as training the Paragraph Vector model and was done with a dataset of a billion words.

| Algorithm | N-Grams | Bag-Of-Means | Paragraph Vectors |
|---|---|---|---|
| Data Transformation | 0.68s | 1.49m | 5.31h |
| Logistic Regression | 32.08s | 2.15m | 6.26m |
| Decision Tree | 30.05s | 4.47m | 4.85m |
| Naive Bayes | 1.33s | 12.29s | 12.07s |
| SVM | 15.53m | 1.30h | 1.60h |
| kNN | 4.24s | 1.16h | 1.23h |
| Random Forest | 14.82m | 4.24h | 5.65h |
| Neural Network | 21.32m | 21.26m | 21.04m |
| Deep Neural Network | 32.84m | 32.65m | 32.22m |

Table 14: Computation time for the experiments. The top row shows the time for generating the feature space models and the rest of the results show the runtime for the different classifiers. The neural network models are separated because they were trained on an Amazon Web Services (AWS) EC2 *g2.2xlarge* GPU instance while the other models were trained on *c4.large* instance using a multi-core CPU. The runtime of all classifiers includedes 5-fold Cross-Validation and grid-search over hyper-parameters for most models. Thus the results are only representative to a certain extent as some models had a smaller hyper-parameter space to be evaluated.

The difference in complexity involved in training the models can also be seen when comparing the computation time of the classifiers with the different vector space models. Here we can see that training and evaluation on 5 folds can be done with an N-Gram model in minutes even with computationally more resource-intensive methods like a Support Vector Machine. Compared to that the runtime of evaluating the Bag-of-Means models takes significantly longer with a varying degree depending on the classifier and this trend potentiates with Paragraph Vectors.

The classifiers with the shortest computation time are the Naive Bayes which can be computed on an N-Gram model in only 1.33 seconds, totaling to around 2 seconds including the construction of the N-Gram vector space, and kNN. Similarly Logistic Regression and Decision Trees can be evaluated in half a minute using N-Grams and the SVM and Random Forest require much more time to compute.

These relations between the classifiers translate to the other vector space models with a few exceptions: The runtime of kNN in the Bag-of-Means and Paragraph Vector spaces becomes much more resource intensive, almost reaching the level needed to compute the SVM model. Naive Bayes keeps computation within a couple of seconds for all models, but at the same time we observe its test score performance decrease with the Bag-of-Means and Paragraph Vector models as mentioned previously.

Both Neural Network models were trained on a GPU instance and as such their computation time shows a very different profile. Despite parallelization of training over GPU cores these methods still have a higher demand on computation time than all other models that were trained on the single core machine when looking at the N-Gram model. On the other hand the time to evaluate these models does not increase with the Bag-of-Means or the Paragraph Vector models as it does for all other methods. In fact we can see a minimal decrease in computation time.

## 5.4 Classification With Sequential Models

**Experimental setup and model choices** Three sequential models were evaluated, each of them based on Long Short-Term Memory Networks (see Section 4.3). All Networks are trained at character level, so instead of indices for words they receive the sentences as a sequence of characters as their input:

- *Simple LSTM*: This model uses a single LSTM layer of dimensionality 256. The time steps of this model are 200, meaning that from each sentences the first 200 letters are used and shorter sequences are appended with padding characters. Then a dropout layer is applied, i.e. a layer which prevents forward propagation of each neuron with a given probability as proposed by [Srivastava et al., 2014] to prevent overfitting. Here a probability of 0.5 was used. The output layer consists of six nodes which encode the labels used for classification and uses a softmax activation function.

- *Stacked LSTM*: The stacked model is similar to the simple model but instead uses three LSTM layers of dimensionality 256, 64 and 32 respectively. The first two layers again return a sequence of the sequence size 200, making the model theoretically capable to learn several levels of abstractions. The last LSTM layer is again connected to the output layer encoding the prediction of labels. Dropout is applied to each LSTM layer with a probability of 0.5.

- *Multi-task LSTM*: The multi-task model is slightly more complex in its setup. It is in principle a generative model that, given a sequence, produces both the next character and the label that is most likely at this point in the sequence. Like the other models it reads sequences, in this case limited to 40 characters. On top it uses two stacked LSTM layers of which the first one returns sequences itself. The layers have a dimensionality of 512 and dropout with a probability of 0.2 is used. The output layer in this is of size $|K| + |L|$ where $|K|$ is the amount of possible characters and $|L|$ denotes the number or labels. At inference time model was used to step through a sentence one character at a time and for each character the label prediction was used so that the whole sentence would be labelled at character level. Then a majority vote over the labels was performed to retrieve the final prediction of the label for the given sentence.

Table 15 shows the results of each model. During training the performance was measured in terms of categorical cross-entropy as the evaluation of MCC was computationally quite expensive at each iteration due to a lack of a distributed implementation so that for evaluation a switch between GPU and CPU would have had to be performed. In the table these results are therefor called Training Loss and Test Loss. At test time the score was measured using MCC.

| Model | Training Loss | Test Loss | Test Score |
|---|---|---|---|
| Simple LSTM | 0.733 | 0.767 | 0.689 |
| Stacked LSTM | 0.674 | 0.791 | **0.707** |
| Multi-task LSTM | — | — | 0.258 |

Table 15: Test and training scores for the LSTM recurrent neural network models. Due to limited computational resources the MCC score was only evaluated at test time while measuring the categorical cross-entropy loss during training and also at test time.

The first two models perform on par with the classification methods using vector space models discussed in the previously. The stacked model achieves the highest score of 0.707 which is comparable to the Logistic Regression model using a Bag-Of-Means representation but slightly worse than the Neural Network models or the SVM. The simple LSTM model performs worse with an absolute difference 1.8% in test performance. The training to test ratios of 0,955 for the Simple LSTM and 0,852 for the stacked LSTM show that the model with one layer tends to overfit much less, despite performing worse overall. The multi-task LSTM did not nearly achieve similar performance and classifies sentences with an MCC test score of 0.258. Training and Test loss in this case were not measured.

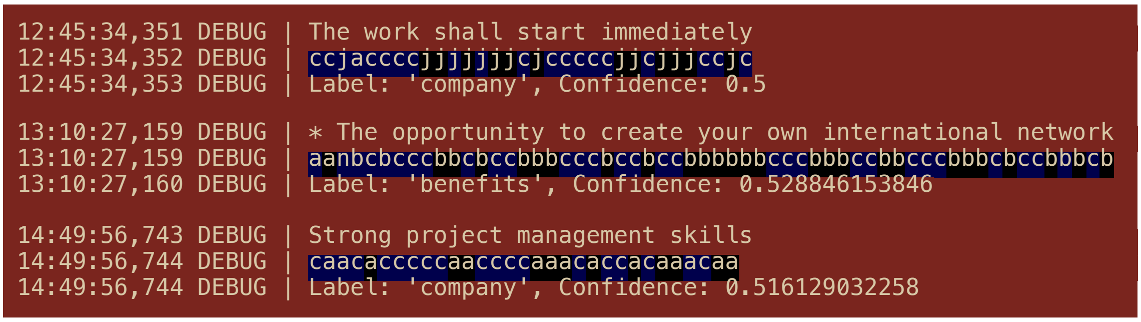
| Model | Runtime |
|---|---|
| Simple LSTM | 7.04h |
| Stacked LSTM | 3.05h |
| Multi-task LSTM | 27.57h |

Table 16: Computation time for the LSTM experiments. These were carried out on a single core instance as opposed to the other Neural Network models above and could be sped up on a GPU.

Table 16 shows the computation time for training and evaluating each of the models. The single LSTM layer network takes more time than the stacked one. It is important to note that these experiments were not carried out on an AWS EC2 instance as the experiments for classification with vector space models. Instead they were computed on a personal machine which was in use so the resulting difference in computation is likely to be an artifact of that. The first two models both ran

60 iterations and the best validation loss during training was reached after 30-40 iterations. The computation time for the Multi-task network takes significantly more time with almost 28 hours.

The multi-task network has the nice property that we can sample from it which is a great tool for introspection. Figure 19 shows three samples of typical output when the network is sequentially processing sentences. For each character a label is produced given the previous characters in the sequence that have already been seen. The brightness of color here encodes the confidence about the label (brighter means higher).



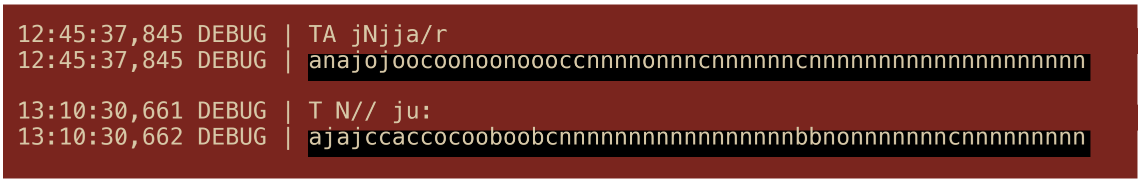
```
12:45:34,351 DEBUG | The work shall start immediately
12:45:34,352 DEBUG | ccjaccccjjjjjjjcjcccccjjcjjjccjc
12:45:34,353 DEBUG | Label: 'company', Confidence: 0.5

13:10:27,159 DEBUG | * The opportunity to create your own international network
13:10:27,159 DEBUG | aanbcbcccbbcbccbbbcccbccbccbbbbbbcccbbbccbbcccbbbcbccbbbcb
13:10:27,160 DEBUG | Label: 'benefits', Confidence: 0.528846153846

14:49:56,743 DEBUG | Strong project management skills
14:49:56,744 DEBUG | caacacccccaaccccaaacaccacaaacaa
14:49:56,744 DEBUG | Label: 'company', Confidence: 0.516129032258
```

Figure 19: Samples of the network sequentially labeling sentences. The brightness of the bluetone behind each character shows the confidence about the prediction, i.e. the probability the networks softmax function outputs it. In these samples the confidence is thus very low (black) or low (dark blue). The labels are: *nextsteps* ($n$), *job* ($j$), *benefits* ($b$), *other* ($o$) and *candidate* ($a$ is used as in applicant since $c$ already encodes company).

Some interesting observations can be made here, e.g. that the word *work* in the first line immediately triggers the network to output the label $j$. In general though the output fluctuates quite much, and the overall confidence which is achieved by a majority vote over the labels is often just over 50%.

Another interesting possibility for getting a better understanding of the model's behavior is to let it generate output on its own. The network is primed with sequence of padding characters which contain no information in the dataset as they only extend sequences to the expected length. Then we create a first character and label and this serves as the input for the next prediction. Thus the network is always provided with a truncated history of its own output and generates more on its basis.

Figure 20 shows such output which tells us that the network did not learn much about language. It is unable to produce output that resembles a word and generally stops using other characters than a whitespace after a few characters. Interestingly it still uses different labels at times even when producing whitespace characters.



```
12:45:37,845 DEBUG | TA jNjja/r
12:45:37,845 DEBUG | anajojoocoonoonoooccnnnnonnncnnnnnncnnnnnnnnnnnnnnnnnnnn

13:10:30,661 DEBUG | T N// ju:
13:10:30,662 DEBUG | ajajccaccocooboobcnnnnnnnnnnnnnnnnnbbnonnnnnnncnnnnnnnnn
```

Figure 20: Samples of the network generating sequences. At each time step the LSTM is to produce a new character and a new label for this character given the output it has already produced.

# 6 Discussion

## 6.1 Vector Space Models Approach

### 6.1.1 N-gram Models

Not without reason are N-Gram Models still so widely used as the experiments in Section 5 confirm. They are easy to understand, straightforward to train using publicly available software libraries, fast to compute and yield good performance for many problems in NLP.

Hyper-parameters have to be adjusted before training the model as they affect the results differently given the problem and also the used classifier. As we saw in Section 5.3.1 the type of N-gram (word versus character-based) and the vector dimensionality have the strongest impact on the results. Other hyper-parameters do not have a substantial influence as for instance stop-words, or do not seem to matter at all in this particular settings such as sub-linear term frequency scaling.

Optimizing for hyper-parameters though is very cheap as well as an N-gram model can be trained and evaluated in a matter of seconds, allowing to exhaustively and automatically explore a wide parameter-space with simple Grid Search. N-Grams thus make for a great choice as a baseline model with good performance and simple usage.

### 6.1.2 Bag-of-Means Model

Composition of feature vectors to produce a feature representation for a group of objects is a known approach (see e.g. [Mitchell and Lapata, 2010] for a study on compositions of word embeddings). However this technique is also seen as an ad-hoc solution with unforeseeable side-effects such as flattening out the significance of complementary patterns as the number of vectors increase. As [Zhang et al., 2015] puts it: “Bag-of-means is a misuse of word2vec [...]” and “such a simple use of a distributed word representation may not give us an advantage to text classification”. Further [Le and Mikolov, 2014] state that “weighted averaging of word vectors [...] loses the word order in the same way as the standard bag-of-words models do”.

Given these previous results and thoughts it is the more surprising how well the Bag-of-Means model performs, achieving the highest performance of all models in combination with a simple Neural Network. This type of model might work well in this specific case for two reasons: First the word vectors that were used had been trained on a humongous dataset, implicit knowledge that the model could leverage while the other vector space models could not. Second the dataset consists of sentences, so short sequences of text where the problem of vectors negating each other’s patterns can be expected to not be as severe.

Regarding computational complexity this model is relatively fast to train if one uses pre-computed word-vectors as done in the experiments. However the training of such corpus of 1 billion word-vectors itself takes enormous time and resources.

### 6.1.3 Paragraph Vectors using Distributed Representations

The Paragraph Vector model is appealing with it's inspiration from word embeddings and has been shown to outperform other text representations, specifically N-gram models (see e.g. [Le and Mikolov, 2014]). However significant effort is needed to achieve good results with this technique and it is especially sensitive to the configuration of its hyper-parameters which are challenging to extensively test due to the computational resources needed.

With regards to the hyper-parameters, the choice the biggest impact on performance is the chosen model architecture. The results on this specific task and dataset show that contrary to the observations in [Le and Mikolov, 2014] the Distributed Bag-Of-Words (PV-DBOW) architecture achieves significantly higher performance (an absolute difference of almost 14% in terms of MCC test score) and is more stable towards overfitting. As in other vector space models the dimensionality of the vector representation naturally has a strong impact on performance, however the with decreasing vector size performance does not suffer as much as N-Grams models do and the model still achieves an MCC test score of 0.223 with only 2 dimensions. This corresponds to a relative loss in performance by a factor of 2/3 while scaling down in dimensionality by a factor of 150. Further adding to the complexity of finding a good model several hyper-parameter interact with each other. For instance sub-sampling of frequent words only helped performance when using the Distributed Memory (PV-DM) model and otherwise significantly harmed it.

An interesting observation is that a number of classifiers tend to overfit using the vectors produced by this model. This could mean that it produces clearer patterns in the representation of documents and a richer representation. It is very likely that this type of model needs a much larger dataset in order to achieve it's highest performance as for many models based on Neural Network architectures. As mentioned earlier the word2vec models in [Mikolov et al., 2013a] were trained on a dataset of a billion words.

### 6.1.4 Comparison of Vector Space Models

Based on the results and learnings regarding the vector space models we can see that the choice about these models for a prediction task depends on a few factors which will be described next along with considerations regarding the models:

- **Size of the dataset:** Especially when training distributed representations large datasets are needed. In comparison the dataset aquired for this thesis was small: For instance the Google News dataset mentioned above and created by [Mikolov et al., 2013b] used 3 million words while the dataset used here contains a total of 121,119 words which is 4% of the size. It is very likely that a bigger dataset is needed to achieve the models' best performance. In the case of a small dataset as given in this work N-Grams still work reliably well.

- **Type of problem:** Here a topic classification problem is given, a task that can be seen as a very objective analysis of the type of information and facts contained. Other tasks like sentiment analysis where a classifier predicts whether a positive or negative sentiment is expressed have to rely on much more subtle notions in the language in order to perform well. While sentiment analysis can be framed as a simple multi-class classification problem as is the case in this work, in order to identify for instance sarcasm slight variations in the word order or formulations can make a big difference while topic analysis can work by simply spotting words or word combinations as an N-Gram model does. In this regard distributed representation models are expected to work better the more subtleties in the language matter to the task as they are shown to be very effective at modeling these (see e.g. [Mikolov et al., 2013c]), given of course enough data is available.

- **Constraints on development time:** Testing, tuning and deploying an N-Gram model is significantly faster than the other models that were investigated. This is due to the fact that computation time is fast as it simply based on occurrence counts. This also makes hyper-parameter optimization feasible using exhaustive grid search. On the other hand given a set of trained word embeddings such as the Google News dataset (see Appendix Section A.2) the computation of the Bag-Of-Means model is as simple as averaging vectors and does not require any tuning (though of course other vector compositions are possible as mentioned in [Mitchell and Lapata, 2010]). Developing a good Paragraph Vector model is possible but takes significant effort as the effects and interaction of the hyper-parameters is quite complex and quick experimentation or exhaustive search is more challenging with the computational demands of this model.
- **Constraints on computational resources:** A minor point to take note of is that there is a significant difference in the need computational resources. If these are limited, e.g. for financial reasons or because the software runs on constrained hardware such as on a phone or an embedded device, training Paragraph Vectors might be infeasible and equally training the Bag-Of-Means model if the word embeddings have to be computed from scratch.

N-Gram models have been subject to research for many years. The results here show that these models still perform very well on small datasets and can make for a great baseline model. Further they are well and easily understood as they their assumptions align with our intuition how texts can be characterized. These assumptions however are also strong ones and their shortcomings such as insensitivity to word order and inability to capture subtle meaning and semantic relationships between words have been shown in previous work.

This perspective makes the models using distributed representations much more appealing as they do not make strong assumptions about the structure of the data except that the occurrence of words in another word's context signifies a relationship between them. We have seen that while not on par with N-Grams, the Paragraph Vector model still achieve good results with a difference of only 3.6% in absolute test performance for a Deep Neural Network. This is especially remarkable given that similar models are usually trained on much larger data. Further in comparison to N-Grams which were designed to N-Gram occurrences as a good indicator of the content of a sentence, the Paragraph Vector model did not have this knowledge or inductive bias built in but had to learn such relationships from scratch by training it to simply predict the missing word(s) given other words from a sentence.

More surprisingly when using well-trained word embeddings a simple averaged model of the word vectors in a sentence, an approach that was stated by previous work as too simple and not performing well (see Section 6.1.2), performed best. With this result, given we have pre-trained word vectors at hand this approach is the most straightforward to take and achieves best results. Note that given a larger dataset Paragraph Vectors might outperform this result and there is reason to believe that this technique performs worse with longer texts as mentioned in Section 6.1.2.

### 6.1.5 Comparison of Classification Methods

The classification techniques evaluated yield strongly varying results in combination with the different vector space models. A key takeaway here is that simple models should always be the starting point as we can see that a Logistic Regression classifier here already produces a very strong baseline which is only surpassed by few methods. Namely these are an the SVM and both Neural Network based algorithms which, despite a far higher level of complexity with regards to model choices such as architecture and hyper-parameters, only lead to a marginal improvement over Logistic Regression.

There seems to be a tendency of certain algorithms to work better with N-Gram models than with distributed representations. This can be observed for the tree-based methods, i.e. Decision Tree and Random Forest, and Naive Bayes. This could be interpreted with regards to these methods' model assumptions: Tree-based methods make exclusive choices on feature dimensions, splitting up the hypothesis space by thresholding the features. This works well for N-Grams as we can expect that there is no strong interaction between the dimensions: Each dimension encodes the (possibly transformed) number of occurrences of a certain pattern. Distributed representations on the other hand can be much more dense as essentially learn basis functions that are composed to show the desired pattern. Thus the more the patterns can be expressed in terms of compositions of these basis functions, the more information the representation can contain as it is more compressed and less redundant. This on the other hand means that simply selecting feature dimensions will not capture the complex activation composition of dimensions and might lead to the tree-based methods to perform poorly. Similarly the Naive Bayes works on the explicit assumption that features are independent and identically distributed. This is one of the reasons the technique works well in combination with N-Grams as features are words or groups of words. However similarly to Tree-based methods the compositional nature of distributed representations and does not align well with Naive Bayes' model assumptions and might be the a cause for their poor performance with these methods.

Similar observations with can be made with regards to runtime: For instance training a kNN model with distributed representations increases the runtime by an order of magnitudes although the vector representations are all of the same size. This is again likely due to these vectors being much more dense as opposed to N-Gram vectors where in the absence of an N-Gram the activation is simply zero. This can be assumed to occur relatively often as specific words with lower overall frequency are preferred for carrying more information about a certain class. On the other hand for some classifiers the distributed representation models do not significantly increase computation time. Decision Trees can only work with the features they are given as opposed to Random Forests which can extensively bootstrap samples for good feature subsets and therefore take much longer time to converge. Logistic Regression and Neural Network models simply take the input as an activation which is exactly how the distributed representations are trained in the first place, and thus do not need much more time or even less in some cases. SVM's like kNN operate on the distance of vectors in the vector space (though not directly if the kernel-trick is applied). This makes them much slower to compute with dense vectors as well as for sparse vectors certain dimensions are automatically omitted (and become zero in the dot product). Therefore also SVM's are much more compute-intensive when using distributed representations.

When choosing a classifier similar considerations as for the vector space model apply. Simple models such as Logistic Regression need far less time to be implemented, tested and tuned and with and lead to good performance. Of course this choice has to be informed by empirical results as simply choosing a method that is easy to implement and understand can lead to very poor performance as for the combination of Decision Trees with a Paragraph Vector representation (MCC test score of 0.290). With more time and effort the limits can of performance can be pushed but as mentioned above the most limiting factor for distributed representations and also for methods like Deep Neural Networks is the size of the dataset used as argued e.g. by [Halevy et al., 2009].

## 6.2 Sequential Modeling Approach

When building the RNN models the expectation was that they might be able to achieve a positive correlation score but not much more. This is mostly because these models, especially when stacked, need lots of training data. A nice read on the topic can be found on Andrej Karpathy's blog [10] where different generative LSTM based models were trained to produce english text, or linux source code. Similarly [Graves, 2013] used LSTM Networks to generate wikipedia articles. For these experiments a wikipedia dataset of 100MB was used, and Andrej Karpathy's smallest dataset is 1MB or a million characters which is slightly more than the dataset given here, before splitting it into training and test sets.

Nevertheless the experiments yield results on the same level with a Logistic Regression model paired with Bag-Of-Means or a Neural Network using N-Grams and with an MCC test score of 0.707 are among the higher scoring models in this

---

[10] `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

thesis. This is surprising for at least two reasons: First these models did not have any assumptions about language built in, not even the information what a word is. This means they had to learn a representation of the english language at least to the level that is needed to identify the enough structure to effectively classify sentences — and in the case of the Multi-Learner actual english language one of the objectives was to produce written language. The other reason is that while for the vector spaces methods, especially the Paragraph Vector models, extensive research and experimentation was needed to achieve good results, the LSTM models used here were the result of a few rounds of testing different regularization by modifying the Dropout probability and trying different sizes of hidden layers. The final models are actually fairly small, [Graves, 2013] for instance used a network with an LSTM of 1000 nodes. However it is important to note that in general LSTM models are far more complex in their nature compared to e.g. a Logistic Regression model. While using open software packages that abstract much of the implementation details away and example code for network architectures known to be effective towards certain problems available it would have been significantly more effort to implement and tune these models from scratch.

Is is questionable whether the performance of these models reached their potential. As mentioned before these models work much better with more data, further the architecture was small and with stronger regularization training bigger networks would have been successful. This is especially visible for the Multi-Task Learning Network which clearly did not reach a level where it can produce output resembling english language. While it had to predict both characters and labels, a task that is slightly more involved than only producing the next character given a sequence, its poor performance is likely to be an artifact of either too little data or mistakes setting the training parameters as almost identical architectures were shown to master this task (as can be seen in the blog post mentioned above and [Graves, 2013]).

Computational requirements of RNNs and LSTM Networks in particular are very high as seen in these experiments, although it is important to note that these runtimes do not compare with the other Neural Network models as these were trained on a GPU. Making use of parallel computing capabilities also training recurrent models will be faster by an order of magnitudes.

## 6.3 Comparison of Approaches

Vector Space models are an effective way of tackling problems in CL and NLP as we have seen. It is crucial though to not blindly pick from the wide array of possible representation approaches and classification methods but to choose a vector representation approach whose model assumptions align with the classifier's assumptions to enable the classifier to be able to exploit and leverage the structure exposed by the representation. N-Gram models can work with very little data due their strong inductive bias but are somewhat limited in their expressiveness while distributed representations produce much more dense vectors with strong interaction between dimensions, similar to the idea of basis functions in sparse coding or "self-taught learning" (see e.g. [Raina et al., 2007]).

The idea of treating text as a signal in time and modeling its dynamics without imposing any assumptions about language on the model on the other hand is very appealing and as shown here can lead to comparable results even with small datasets. While the RNN model variants themselves can be quite involved they allow for much less thinking about the characteristics of the problem domain, learning to expose the needed structure themselves. In this sense this approach shows more promise towards higher effectiveness if more data is given.

# 7 Conclusions

This thesis has applied an explorative and iterative product development process to define and then research a machine learning problem and approaches to solve it. The research problem investigated was set to be multi-class text classification on sentence data. Despite being an well-known and simple problem in its nature it is far from being solved and the topic of much recent works, also because of it's wide applicability to various real world problem domains such as sentiment analysis, spam identification and document filtering.

This work investigated a recent trend in CL and NLP, but also more generally in fields across ML towards more general purpose methods which avoid built-in assumptions about the problem domain and are therefore applicable through various domains. We saw in this work that beyond potentially freeing us from the often involved and tedious work of Feature Engineering these methods also perform on par or even better. This is because they inform their data representation choices in a data-driven way and with enough data and a well chosen model design are shown to increasingly outperform hand-designed feature extraction techniques. This also brings us closer to one of the goals of AI by having more world informed knowledge and decisions in intelligent systems instead of providing this knowledge in a curated form.

These trends were greatly accelerated by the recent wave in popularity of Deep Learning methods and more generally connectionist models where ideas from Representation Learning and Unsupervised Feature Learning have been a focus of research for decades, by the growth in large data and its availability and by increased access to massive parallel and distributed computational resources. Currently these trends are only increasing so we can expect more work in this research in this inspiring direction.

## Proposals for Future Research

The work on this thesis has triggered and inspired quite a variety of ideas for further research. Here we will look at a few of these which might be considered as a continuation of this work:

**Structured prediction** When the first problem definition was formulated it was posed as a structured prediction task (see Section 2.2.1). The field of structured prediction is an active field of research in ML and the problem could be treated in such a sense directly from a Machine Learning (ML) perspective, e.g. by trying to not only identify certain high level themes and topics as *requirements* but also specific sub-categories as *language skills* as well as the hierarchy over these.

**Vector Space Trajectories** As described in Section 4.1.2 and shown by [Mikolov et al., 2013c] distributed representation models encode linguistic and semantic relationships through dimensions in the vector space. Mikolov's analogy task shows that even certain algebraic operations allow us to navigate between related concepts within this vector space. Further one can regard a sentence or a text as a trajectory through this word vector space. Taking this perspective leads to various interesting potential directions of research. Trajectory based classification could be applied to classify sentences (an idea somewhat related to sequential modeling of text) or one could try to identify topic change within a text, speaker identification in a conversation or more modeling subtle linguistic notions as humor through variations from an expected trajectory. One should note that these are just (somewhat wild) ideas but the principle could be investigated.

**Verification of assumptions regarding the size of the dataset** The result and discussion section of this work makes an assumption towards Representation Learning and sequential modeling approaches to perform better if more data were provided, while the evidence with given data just slightly better performance. This assumption is of course based on previous work as mentioned but it could be confirmed or falsified by either collection a bigger dataset or by comparing the exact same set of methods against a standard dataset of larger size such as the Reuters-21578 dataset[11].

**Continuation of Multi-Task Generative Model** As shown in the results the Multi-Task Learning LSTM network was not able to produce english words while previous work showed that in a single task setting this is possible. More experimentation would be needed, as it was not clear whether model was even converged in training due to its slow training time. Also, again, the dataset might have been to small for such a task. In several previous work Multi-Task Learning has been shown to improve performance of the shared tasks (see e.g. [Collobert and Weston, 2008]) and thus it would be interesting to test the limits of this approach.

**Combine approaches with CNNs** : CNNs have been shown to be highly effective also in language based problem domains and especially character-level networks have recently attracted attention (see e.g. cite[Zhang et al., 2015]). While evaluating the Neural Network models a convolutional network was actually implemented but led to such poor performance that it was not further investigated. As this model captures patterns invariant of their position in the data it can be expected to align well with text analysis tasks and could be investigated with further effort.

**Test the methods with Finnish data** : For the sake of simplicity and interpretability only english data was analyzed. The vast majority of the dataset at hand though was in Finnish as it stems from a Finnish service. Finnish does not share the same root with english from a linguistic point of view and has a very different structure, e.g. making much more use of suffixes instead of prepositions. Especially word-index based methods such as N-Grams are expected to suffer in performance in this regard while character based methods might work better that “do not know the concept of a word”. As mentioned previously a recent interesting approach in

---

[11] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

this direction was taken by [Bojanowski et al., 2016] by using so-called sub-word information and could be embraced.

# References

[Alpaydin, 2014] Alpaydin, E. (2014). *Introduction to machine learning.* MIT press.

[Barber, 2012] Barber, D. (2012). *Bayesian reasoning and machine learning.* Cambridge University Press.

[Bengio and Bengio, 2000] Bengio, S. and Bengio, Y. (2000). Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks*, 11(3):550–557.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., and Vincent, P. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Bengio et al., 2015] Bengio, Y., Goodfellow, I. J., and Courville, A. (2015). Deep learning. *An MIT Press book in preparation. Draft chapters available at http://www. iro. umontreal. ca/ bengioy/dlbook.*

[Best, 2006] Best, K. (2006). *Design management: managing design strategy, process and implementation.* AVA publishing.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022.

[Bojanowski et al., 2016] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606.*

[Borko and Bernick, 1963] Borko, H. and Bernick, M. (1963). Automatic Document Classification. *J. ACM*, 10(2):151–162.

[Breiman, 2001] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

[Breiman et al., 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and Regression Trees.* The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis.

[British Design Council, 2007] British Design Council (2007). Eleven lessons: Managing design in eleven global companies-desk research report.

[Cambria and White, 2014] Cambria, E. and White, B. (2014). Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]. *IEEE Computational Intelligence Magazine*, 9(2):48–57.

[Cho, 2014] Cho, K. (2014). *Foundations and Advances in Deep Learning.* PhD thesis, Aalto University.

[Clark et al., 2013] Clark, A., Fox, C., and Lappin, S. (2013). *The Handbook of Computational Linguistics and Natural Language Processing.* John Wiley & Sons.

[Cleverdon, 1984] Cleverdon, C. (1984). Optimizing Convenient Online Access to Bibliographic Databases. *Information Services and Use*, 4:37–47.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA. ACM.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3):273–297.

[Do and Ng, 2006] Do, C. B. and Ng, A. Y. (2006). Transfer learning for text classification. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 299–306. MIT Press.

[Gers et al., 1999] Gers, F., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: continual prediction with LSTM. In *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2.

[Gorodkin, 2004] Gorodkin, J. (2004). Comparing two K-category assignments by a K-category correlation coefficient. *Computational Biology and Chemistry*, 28(5–6):367–374.

[Graves, 2012] Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence.* Springer Berlin Heidelberg, Berlin, Heidelberg.

[Graves, 2013] Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs].* arXiv: 1308.0850.

[Greff et al., 2015] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). LSTM: A Search Space Odyssey. *arXiv:1503.04069 [cs].* arXiv: 1503.04069.

[Gutmann and Hyvärinen, 2012] Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. *J. Mach. Learn. Res.*, 13(1):307–361.

[Halevy et al., 2009] Halevy, A., Norvig, P., and Pereira, F. (2009). The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24(2):8–12.

[Hinton, 1986] Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA.

[Hochreiter, 1991] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München*, page 91.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

[Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759.*

[Jurafsky and Martin, 2014] Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing.* Pearson.

[Jurman and Furlanello, 2012] Jurman, G. and Furlanello, C. (2012). A unifying view for performance measures in multi-class prediction. *PLoS ONE*, 7(8):e41882. arXiv: 1008.2908.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882.*

[Kohonen, 1990] Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Departmental Papers (CIS).*

[Le and Mikolov, 2014] Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053.*

[Leskovec et al., 2014] Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets.* Cambridge University Press.

[Liu et al., 2016] Liu, P., Qiu, X., and Huang, X. (2016). Recurrent Neural Network for Text Classification with Multi-Task Learning. *arXiv:1605.05101 [cs].* arXiv: 1605.05101.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text Classification Using String Kernels. *J. Mach. Learn. Res.*, 2:419–444.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

[Mahoney, 1999] Mahoney, M. V. (1999). Text compression as a test for artificial intelligence. In *AAAI/IAAI*, page 970.

[Manning et al., 2008] Manning, C. D., Raghavan, P., Schütze, H., and others (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

[Manning and Schütze, 1999] Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*, volume 999. MIT Press.

[Matthews, 1975] Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451.

[McCallum, 1999] McCallum, A. (1999). Multi-label text classification with a mixture model trained by EM. In *AAAI'99 workshop on text learning*, pages 1–7.

[Merkl, 1998] Merkl, D. (1998). Text classification with self-organizing maps: Some lessons learned. *Neurocomputing*, 21(1–3):61–77.

[Mikolov, 2012] Mikolov, T. (2012). *Statistical Language Models Based on Neural Networks*. PhD thesis, Ph. D. thesis, Brno University of Technology.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. arXiv: 1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

[Mikolov et al., 2013c] Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic Regularities in Continuous Space Word Representations. In *HLT-NAACL*, pages 746–751.

[Mitchell and Lapata, 2010] Mitchell, J. and Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science*, 34(8):1388–1429.

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective.* MIT press.

[Nigam et al., 2000] Nigam, K., Mccallum, A. K., Thrun, S., and Mitchell, T. (2000). Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 39(2-3):103–134.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global Vectors for Word Representation. In *EMNLP*, volume 14, pages 1532–1543.

[Powers, 2011] Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

[Raina et al., 2007] Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught Learning: Transfer Learning from Unlabeled Data. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 759–766, New York, NY, USA. ACM.

[Rijsbergen, 1979] Rijsbergen, C. J. V. (1979). *Information Retrieval.* Butterworth-Heinemann, London ; Boston, 2nd edition edition.

[Sebastiani, 2002] Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Comput. Surv.*, 34(1):1–47.

[Shannon, 2001] Shannon, C. E. (2001). A Mathematical Theory of Communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55.

[Shawe-Taylor and Cristianini, 2004] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis.* Cambridge University Press. Google-Books-ID: 9i0vg12lti4C.

[Shervashidze et al., 2011] Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[Turing, 1950] Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59(236):433–460.

[Vapnik, 1982] Vapnik, V. N. (1982). *Estimation of dependences based on empirical data.* Springer-Verlag. Google-Books-ID: vAnvAAAAMAAJ.

[Werbos, 1988] Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356.

[Zaremba, 2015] Zaremba, W. (2015). An empirical exploration of recurrent network architectures.

[Zhang et al., 2015] Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 649–657. Curran Associates, Inc.

# Acronyms

| | |
|---|---|
| **AI** | Artificial intelligence 2, 3 |
| **API** | Application Programming Interface 12 |
| **AWS** | Amazon Web Services 59, 61 |
| **CL** | Computational Linguistics 3, 5 |
| **CNN** | Convolutional Neural Network 6, 37 |
| **CRF** | Conditional Random Field 6 |
| **EM** | Expectation Maximization 6 |
| **GPU** | Graphical Processing Unit 37, 39, 59, 61 |
| **IR** | Information Retrieval 4, 20 |
| **JSON** | JavaScript Object Notation 12, 14 |
| **kNN** | k Nearest Neighbors 14, 15, 32, 57 |
| **LDA** | Latent Dirichlet Allocation 6, 83 |
| **MCC** | Matthews Correlation Coefficient 15, 17, 20, 46, 48, 49, 53–55, 57, 60, 61, 64 |
| **ML** | Machine Learning 2, 4, 5 |
| **NLP** | Natural Language Processing 3, 5, 6, 63 |
| **NLTK** | Natural Language Toolkit 16 |
| **NN** | Neural Network 35, 43 |
| **PCA** | Principal Components Analysis 51 |
| **REST** | Representational State Transfer 12 |
| **RNN** | Recurrent Neural Network 6, 39 |
| **SOM** | Self Organizing Map 6 |
| **SVM** | Support Vector Machine 33 |
| **t-SNE** | t-Distributed Stochastic Neighbor Embedding 51 |
| **TC** | Text Classification 3–6 |

# Glossary

**Amazon Mechanical Turk**

“Amazon Mechanical Turk (MTurk) is a crowdsourcing Internet marketplace enabling individuals and businesses (known as Requesters) to coordinate the use of human intelligence to perform tasks that computers are currently unable to do”. Source: `https://en.wikipedia.org/wiki/Amazon_Mechanical_Turk`. See also Mikrotasking. 16

**Amazon Web Services**

“Amazon Web Services (AWS), a subsidiary of Amazon.com,[2] offers a suite of cloud-computing services that make up an on-demand computing platform. [...] The most central and best-known of these services arguably include Amazon Elastic Compute Cloud, also known as "EC2", and Amazon Simple Storage Service, also known as "S3".” Source: `https://en.wikipedia.org/wiki/Amazon_Web_Services` 59

**Application Programming Interface**

An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. 74

**Artificial intelligence**

“Artificial intelligence is intelligence exhibited by machines. In computer science, an ideal intelligent machine is a flexible rational agent that perceives its environment and takes actions that maximize its chance of success at some goal. Colloquially, the term artificial intelligence is applied when a machine mimics cognitive functions that humans associate with other human minds, such as learning and problem solving. As machines become increasingly capable, facilities once thought to require intelligence are removed from the definition. For example, optical character recognition is no longer perceived as an exemplar of artificial intelligence having become a routine technology.” Source: `https://en.wikipedia.org/wiki/Artificial_intelligence` 2, 3, 74

**Bagging**

“Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting.” Source: `https://en.wikipedia.org/wiki/Bootstrap_aggregating` 31

**Beautiful Soup**

Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. See `https://www.crummy.com/software/BeautifulSoup/` 16

**Boosting**

"Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance[1] in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones." Source: `https://en.wikipedia.org/wiki/Boosting_(machine_learning)` 31

**Computational Linguistics**

Computational Linguistics is "the scientific study of language from a computational perspective. Computational linguists are interested in providing computational models of various kinds of linguistic phenomena." Source: `http://www.aclweb.org/archive/misc/what.html` 3, 5, 74

**Conditional Random Field**

"Conditional random fields are a probabilistic framework for labeling and segmenting structured data, such as sequences, trees and lattices. The underlying idea is that of defining a conditional probability distribution over label sequences given a particular observation sequence, rather than a joint distribution over both label and observation sequences." Source: `http://www.inference.phy.cam.ac.uk/hmw26/crf/` 6, 74

**Confusion Matrix**

A Confusion Matrix, sometimes referred to as Error Matrix, is a table listing the expected versus actual results of a classification algorithm for each class. It is often visualized to understand the behaviour of an algorithm. 43, 77

**Convolutional Neural Network**

A Convolutional Neural Network is a specific Neural Network architecture which enable the network to potentially be more invariant to certain transformations of patterns in the data. Specifically it is often used in Computer Vision tasks where it can learn patterns invariantly of their rotation and positon in a picture. For more information refer e.g. to [Bengio et al., 2015] 6, 37, 74

**Cross-Validation**

In order to validate or evaluate a machine learning model Cross-Validation is an effictive technique. The data is split into $k$ parts, so-called folds, and then $k$ times on of these folds is used as the Test Set or Validation Set while the other folds together form the Training Set. 14, 56, 59, 78

**CrowdFlower**

"CrowdFlower is a data enrichment, data mining and crowdsourcing company based in the Mission District of San Francisco, California. The company's software as a service platform allows users to access an online workforce of millions of people to clean, label and enrich data. CrowdFlower is typically used by data scientists at academic institutions, start-ups and large enterprises." Source `https://en.wikipedia.org/wiki/CrowdFlower`. See `https://www.crowdflower.com` for the company's official website. 16, 20

**Crowdsourcing**

"Crowdsourcing is the process of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, especially an online community, rather than from employees or suppliers." Source: `https://en.wikipedia.org/wiki/Crowdsourcing`. See also Mikrotasking 20

**Deep Learning**

Deep Learning is a sub-field of Machine Learning which is related to Representation Learning. In particular models are used which can learn several layers of representations of patterns in data at increasing level of abstraction. Many of such methods use Neural Network approaches and recently the field has gained much interest as with enough computational resources and amounts of data state-of-the-art results are being achieved in many application domains. See [Bengio et al., 2015] for an in-depth coverage. 6, 9, 37, 39

**Dimensionality Reduction**

Dimensionality Reduction is the problem of reducing the dimensionality of a vector space and thus transforming the data in this space to a lower dimensionality, while aiming to retain as much of the contained information as possible. See e.g. [Alpaydin, 2014, Chapter 6, p. 105]. 32, 77, 81

**Ensemble Method**

Ensemble Methods are approaches to combine the predictions of several learning models. See Section 4.2.4 for a brief introduction. 31, 32, 84

**Error Matrix**

See Confusion Matrix 43, 76

**Expectation Maximization**

The Expectation Maximization algorithm is a method for finding maximum likelihood solutions for Graphical Models with latent variables. It is often used for optimizing Mixture Models such as the Gaussian Mixture Model for soft clustering 6

**Feature Engineering**

Feature Engineering refers to techniques for constructing features for learning algorithms with the help of expert or domain knowledge about the data. As opposed to that Representation Learning tries to produce features or representations of the data in a purely data-driven manner. 5

**Feature Extraction**

Feature Extraction is a way to generate a more succint feature representation given the features that are used as input for a learning algorithm and as such is strongly related to Dimensionality Reduction. 32

**GitHub**

"GitHub is a web-based Git repository hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project." Source: `https://en.wikipedia.org/wiki/GitHub` 6, 12

**Gold Standard**

see Ground Truth 78

**Google Docs**

"Google Docs, Google Sheets and Google Slides are a word processor, a spreadsheet and a presentation program respectively, all part of a free, web-based software office suite offered by Google within its Google Drive service. The suite allows users to create and edit documents online while collaborating with other users in real-time." Source: `https://en.wikipedia.org/wiki/Google_Docs,_Sheets_and_Slides` 10, 87

**Graphical Processing Unit**

"A graphics processing unit (GPU) [...] is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. [...] Modern GPUs are very efficient at manipulating computer graphics and image processing, and their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms where the processing of large blocks of data is done in parallel." Source: `https://en.wikipedia.org/wiki/Graphics_processing_unit` 37, 74

**Grid Search**

Grid Search describes the systematic exhaustive search over a space of hyper-parameters. Usually this is combined with Cross-Validation in order to test each hyper-parameter combination within reasonable bounds to find the best configuration for a given model or algorithm. 63

**Ground Truth**

The ground truth, also sometimes referred to as Gold Standard, is the known set of labels in Supervised Learning which forms the target for learning and is thus also used for evaluating a learning algorithm. 6, 78, 82, 83

**Information Retrieval**

Information Retrieval is the problem of retrieving relevant information from a often humongous collection of possible resources. For more information on the subject see e.g. [Rijsbergen, 1979] and [Manning et al., 2008] 4, 74

**JavaScript Object Notation**

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language." Source: `http://www.json.org` 74

**k Nearest Neighbors**

The $k$-Nearest Neighbors algorithm is a non-parametric method for classification or regression tasks. See Section 4.2.5 14, 74

**K-means Clustering**

K-means Clustering is the problem to find clusters by partitioning a euclidean space to finding $k$ cluster centroids which are the mean of the points beloning to each cluster. 14, 79

**langdetect**

langdetect is a language detection library ported from Google's tool *language-detection* (`https://code.google.com/p/language-detection/`). See `https://pypi.python.org/pypi/langdetect` 12

**Latent Dirichlet Allocation**

Latent Dirichlet Allocation is a popular algorithm for Topic Modeling proposed by [Blei et al., 2003]. Conceptually it assumes that a document is generated first sampling a set of topics from a distribution of topics and then sampling words from each of these topics distributions of words. Learning topics from documents can then be achieved by figuratively inverting this process and inferring topics from the distributions of words under the same assumption how the document was generated. 6, 74

**Lloyds Algorithm**

Lloyd's algorithm, introduced by [Lloyd, 1982], is an method for the tackling the K-means Clustering problem. See K-means Clustering. 14

**Machine Learning**

Machine Learning is a discipline that aims to design algorithms which can adapt to and generalize from data, using a performance criterion to optimize their performance. For an in-depth introduction into the topic see e.g. [Murphy, 2012], [Bishop, 2006] or [Alpaydin, 2014] 2, 4, 5, 10, 74, 77, 82

**Matthews Correlation Coefficient**

Matthews Correlation Coefficient is an unbiased correlation score which can be used to evaluate the performance of a supervised learning algorithm. This score is covered in detail in Section 4.4. 15, 20, 74

**Mikrotasking**

"Microtasking is the process of splitting a job into its component microwork and distributing this work over the Internet. Since the inception of microwork, many online services have been developed that specialize in different types of microtasking. Most of them rely on a large, voluntary workforce composed of Internet users from around the world." Source: `https://en.wikipedia.org/wiki/Microwork` 16, 75, 77

**MongoDB**

"MongoDB (from humongous) is a free and open-source cross-platform document-oriented database. [. . . ]." Source: `https://en.wikipedia.org/wiki/MongoDB`, Website: `https://www.mongodb.com` 12

**Multi-Class Classification**

Multi-Class Classification refers to the task of assigning data points with classes. As opposed to Multi-Label Classification each data point can only be assigned one class, meaning that classes are assumed to be non-overlapping. 15, 16, 80

**Multi-Label Classification**

Multi-Label Classification is the problem of assigning data points with an one or more classes. This is sometimes also referred to as tagging as the several classes can be assigned to one data point, contrary to Multi-Class Classification 14, 80

**Mustache**

"Mustache is a simple web template system." Source: `https://en.wikipedia.org/wiki/Mustache_(template_system)`, Website: `https://mustache.github.io` 12

**Natural Language Processing**

“Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction. Many challenges in NLP involve: natural language understanding, enabling computers to derive meaning from human or natural language input; and others involve natural language generation.” Source: `https://en.wikipedia.org/wiki/Natural_language_processing` 3, 74

**Natural Language Toolkit**

“NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning and wrappers for industrial-strength NLP libraries.” Source: `http://www.nltk.org` 16, 74

**Neural Network**

A Neural Network, also referred to as Artificial Neural Network, is a learning algorithm inspired by ideas from Neuro-biology on information processing in the brain. It consists of chained function compositions and learns by adapting the weights in these compositions to reduce error. See 4.2.7 for an introduction to the technique. 35, 63, 74, 76, 77, 81

**Node.js**

“Node.js is an open-source, cross-platform runtime environment for developing server-side Web applications.” Source: `https://en.wikipedia.org/wiki/Node.js` `https://nodejs.org/` 12

**Oikotie Työpaikat**

Oikotie Työpaikat is a recruitment platform offered by Sanoma which offers companies the possibilty to post publicly job advertisments for open positions and much more. See `https://tyopaikat.oikotie.fi`. 9, 12, 16, 19

**Principal Components Analysis**

Principal Components Analysis is a Dimensionality Reduction technique. It transforms data into a lower-dimentional space by projecting it onto a lower-dimensional orthogonal basis that is aligned with the highest variance in the data. It is thus often used for data exploration and visualization. See e.g. [Alpaydin, 2014] for a treatment of this method. 51, 74

**Randomized Algorithm**

Randomized algorithms are algorithms that use randomness as part of their logic. Often this is used to trade algorithmic space and time complexity for accuracy in results, using approximations to achieve faster run-time or less space usage. 31

**Recurrent Neural Network**

A Recurrent Neural Network is a type of Neural Network model with feedback loops built in, making it well suitable to model temporal and sequential patterns. See 4.3 for more information. 6, 74

**Representation Learning**

Representation Learning is the problem of “learning representations of the data that make it easier to extract useful information when building classifiers or other predictors” [Bengio et al., 2013]. Often seen as opposed to so-called feature engineering where domain knowledge is used to create a good representation of data for a model to learn, the focus of representation learning is to extract the representation with a data-driven approach by exploiting and finding regularities in the data. See the mentioned paper or [Bengio et al., 2015] for a coverage of the topic. 6, 23, 77

**Representational State Transfer**

“Representational State Transfer (REST) is an architectural style used for web development. [. . . ] RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers.” Source: `https://en.wikipedia.org/wiki/Representational_state_transfer` 74

**Sanoma**

“Sanoma Corporation (Finnish: Sanoma Oyj, formerly SanomaWSOY) is a leading media group in the Nordic countries with operations in over 10 European countries, based in Helsinki. The group is also among the top five European magazine publishers and has a strong position in Finland as well as in Belgium, Croatia, the Czech Republic, Denmark, Estonia, Hungary, Latvia, Lithuania, the Netherlands, Russia, Serbia, Slovakia, and (until 2015) in Ukraine.” Source: `https://en.wikipedia.org/wiki/Sanoma`. See `http://www.sanoma.com` 19, 81

**scikit-learn**

“Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings. Source code, binaries, and documentation can be downloaded from http://scikit-learn.sourceforge.net.” [Pedregosa et al., 2011] 86

**Self Organizing Map**

Self Organizing Map, also called Kohonen Map after its inventor Teuvo Kohonen, is an unsupervised learning technique to create "spatially organized internal representations of various features of input signals and their abstractions." [Kohonen, 1990] 6, 74

**Silhouette Score**

"Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a succinct graphical representation of how well each object lies within its cluster. It was first described by Peter J. Rousseeuw in 1986." Source: `https://en.wikipedia.org/wiki/Silhouette_(clustering)`. 14

**Supervised Learning**

Supervised Learning refers to a set of Machine Learning problems where a Ground Truth is given. That means given a set of tuples $(x_i, t_i)$ where $x_i$ are data points and $t_i$ are targets or labels to an algorithm to predict the targets $t_k$ for unseen data $x_k$. 5, 12, 78, 83

**Support Vector Machine**

Support Vector Machine refers to a popular kernel-based learning algorithm and is described in detail in Section 4.2.6. 33, 74

**t-Distributed Stochastic Neighbor Embedding**

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. For more information visit the authors information website (`https://lvdmaaten.github.io/tsne/`) and see the introductory publication ([Maaten and Hinton, 2008]) 51, 74

**Test Set**

In Supervised Learning the Test Set is the part of the data used for a final evaluation of a model as opposed to the Training Set that is used for training it and the Validation Set used for tuning the algorithm. 76, 83, 84

**Text Classification**

Text Classification refers to the problem of assiging classes or categories to text sections or documents. A good review of the field is given in [Sebastiani, 2002] and Section 3.1 gives a formal definition. 3–5, 74

**Topic Modeling**

Topic Modeling refers to the research problem of identifying the prevalent topics for a set of documents. This problem is usually in terms of Unsupervised Learning since topic Ground Truth exists beforehand. A popular algorithm for Topic Modeling is LDA. 6, 12, 79

**Training Set**

The Training Set is the portion of the data in Supervised Learning which is used to train a model. After training the model can then be tested on the Test Set for a final evaluation of its performance. 76, 83

**Unsupervised Learning**

Unsupervised Learning is the study of algorithms and data modeling techniques to discover and extract patterns in the data. Contrary to Supervised Learning no Ground Truth is given which the algorithm tries to approximate and so rather aims to reveal structure in the data under modeling assumptions. 6, 12, 83

**Validation Set**

When tuning the hyper-parameters of a learning algorithm often the data is split up into a Training Set used to train the algorithm, a Validation Set used to test different configurations and variations of the model and a Test Set to perform the final evaluation. 76, 83

**Voting**

Voting is an Ensemble Method where the predictions of several models are combined, e.g. using weighted averaging, to form a common prediction. See Ensemble Method 31

# A Appendix

## A.1 Source Files and Code

### A Tool for Crowd-Sourced Tagging of Chunked Job Ads

The following repository contains the source code for the tool described in Section 2.2.2: `https://github.com/cle-ment/ma-thesis-crowdsource-text-tagger` It was used to collect labels for paragraphs with the help of volunteers. To use it a server running MongoDB and Node.js is needed. A jupyer notebook for preprocessing data is provided in case other data shall be used.

### Implementation of Final Experiments

This repository contains all source code needed to run the final experiments which are discussed in Section 5: `https://github.com/cle-ment/ma-thesis-experiments` Unfortunately there was no time to make these completely reproducible from scratch as most experiments were run on AWS instances with all needed libraries setup. The code can be also be run locally though if the python modules are installed and the code to set up the instances is provided if a similar setup is to be achieved.

### LaTeX Source of This Thesis

The source files for this document can be found in the following repository: `https://github.com/cle-ment/ma-thesis-tex`. The source files were made public for others to use them as a template and as a side-effect document the process of writing quite nicely:

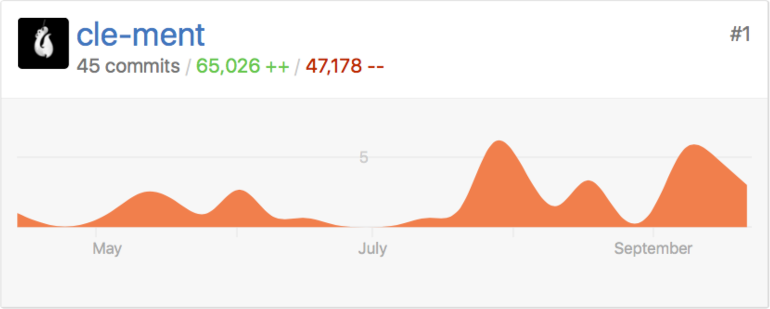


Figure A1: The activity on the thesis repository during the time of writing.

## A.2 Google News Dataset and Word Embeddings

For constructing the Bag-Of-Means model explained in Section 4.1.2, pre-computed word embeddings were used which were created by [Mikolov et al., 2013b] from a dataset from Google News. The dataset contains 300-dimensional vectors for 3 million words and phrases and can be obtained on the following website: `https://code.google.com/archive/p/word2vec/`

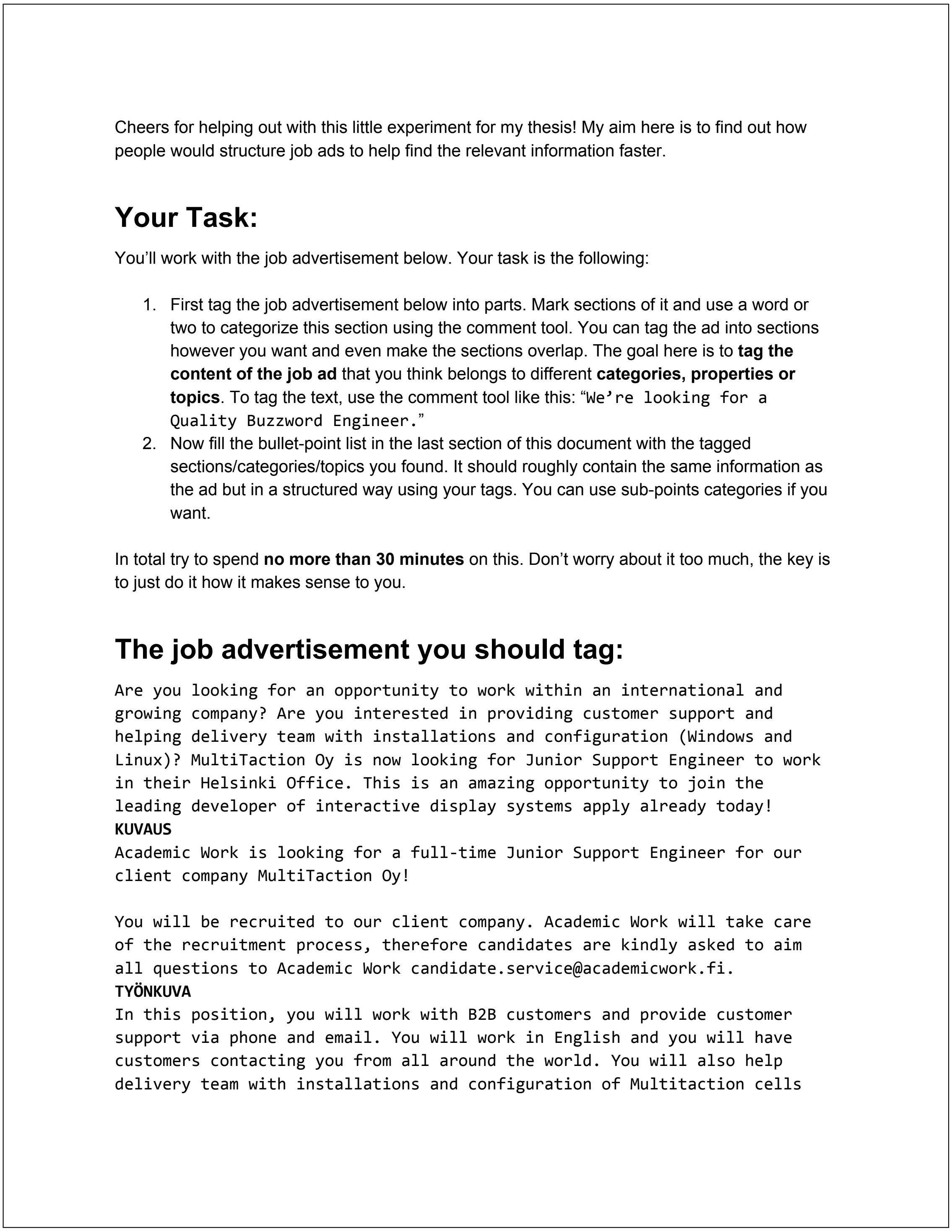
## A.3 Stopwords for N-grams

This following list of English stop words is the one used in the scikit-learn implementation for N-Grams and is taken from the *Glasgow Information Retrieval Group*. The original list can be found at `http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words`.

`a, about, above, across, after, afterwards, again, against, all, almost, alone, along, already, also, although, always, am, among, amongst, amoungst, amount, an, and, another, any, anyhow, anyone, anything, anyway, anywhere, are, around, as, at, back, be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, below, beside, besides, between, beyond, bill, both, bottom, but, by, call, can, cannot, cant, co, computer, con, could, couldnt, cry, de, describe, detail, do, done, down, due, during, each, eg, eight, either, eleven, else, elsewhere, empty, enough, etc, even, ever, every, everyone, everything, everywhere, except, few, fifteen, fify, fill, find, fire, first, five, for, former, formerly, forty, found, four, from, front, full, further, get, give, go, had, has, hasnt, have, he, hence, her, here, hereafter, hereby, herein, hereupon, hers, herself, him, himself, his, how, however, hundred, i, ie, if, in,inc, indeed, interest, into, is, it, its, itself, keep, last, latter, latterly, least, less, ltd, made, many, may, me, meanwhile, might, mill, mine, more, moreover, most, mostly, move, much, must, my, myself, name, namely, neither, never, nevertheless, next, nine, no, nobody, none, noone, nor, not, nothing, now, nowhere, of, off, often, on, once, one, only, onto, or, other, others, otherwise, our, ours, ourselves, out, over, own, part, per, perhaps, please, put, rather, re, same, see, seem, seemed, seeming, seems, serious, several, she, should, show, side, since, sincere, six, sixty, so, some, somehow, someone, something, sometime, sometimes, somewhere, still, such, system, take, ten, than, that, the, their, them, themselves, then, thence, there, thereafter, thereby, therefore, therein, thereupon, these, they, thick, thin, third, this, those, though, three, through, throughout, thru, thus, to, together, too, top, toward, towards, twelve, twenty, two, un, under, until, up, upon, us, very, via, was, we, well, were, what, whatever, when, whence, whenever, where, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, whoever, whole, whom, whose, why, will, with, within, without, would, yet, you, your, yours, yourself, yourselves`

## A.4 Experiment Example: Inferring Structure of Job Advertisements

This section shows an example of the task given to participants during the experiment described in Section 2.2.1. The task was presented in Google Docs, enabling the participants to highlight and comment sections of the text.



Cheers for helping out with this little experiment for my thesis! My aim here is to find out how people would structure job ads to help find the relevant information faster.

### Your Task:

You'll work with the job advertisement below. Your task is the following:

1. First tag the job advertisement below into parts. Mark sections of it and use a word or two to categorize this section using the comment tool. You can tag the ad into sections however you want and even make the sections overlap. The goal here is to **tag the content of the job ad** that you think belongs to different **categories, properties or topics**. To tag the text, use the comment tool like this: "`We're looking for a Quality Buzzword Engineer.`"
2. Now fill the bullet-point list in the last section of this document with the tagged sections/categories/topics you found. It should roughly contain the same information as the ad but in a structured way using your tags. You can use sub-points categories if you want.

In total try to spend **no more than 30 minutes** on this. Don't worry about it too much, the key is to just do it how it makes sense to you.

### The job advertisement you should tag:

Are you looking for an opportunity to work within an international and growing company? Are you interested in providing customer support and helping delivery team with installations and configuration (Windows and Linux)? MultiTaction Oy is now looking for Junior Support Engineer to work in their Helsinki Office. This is an amazing opportunity to join the leading developer of interactive display systems apply already today!

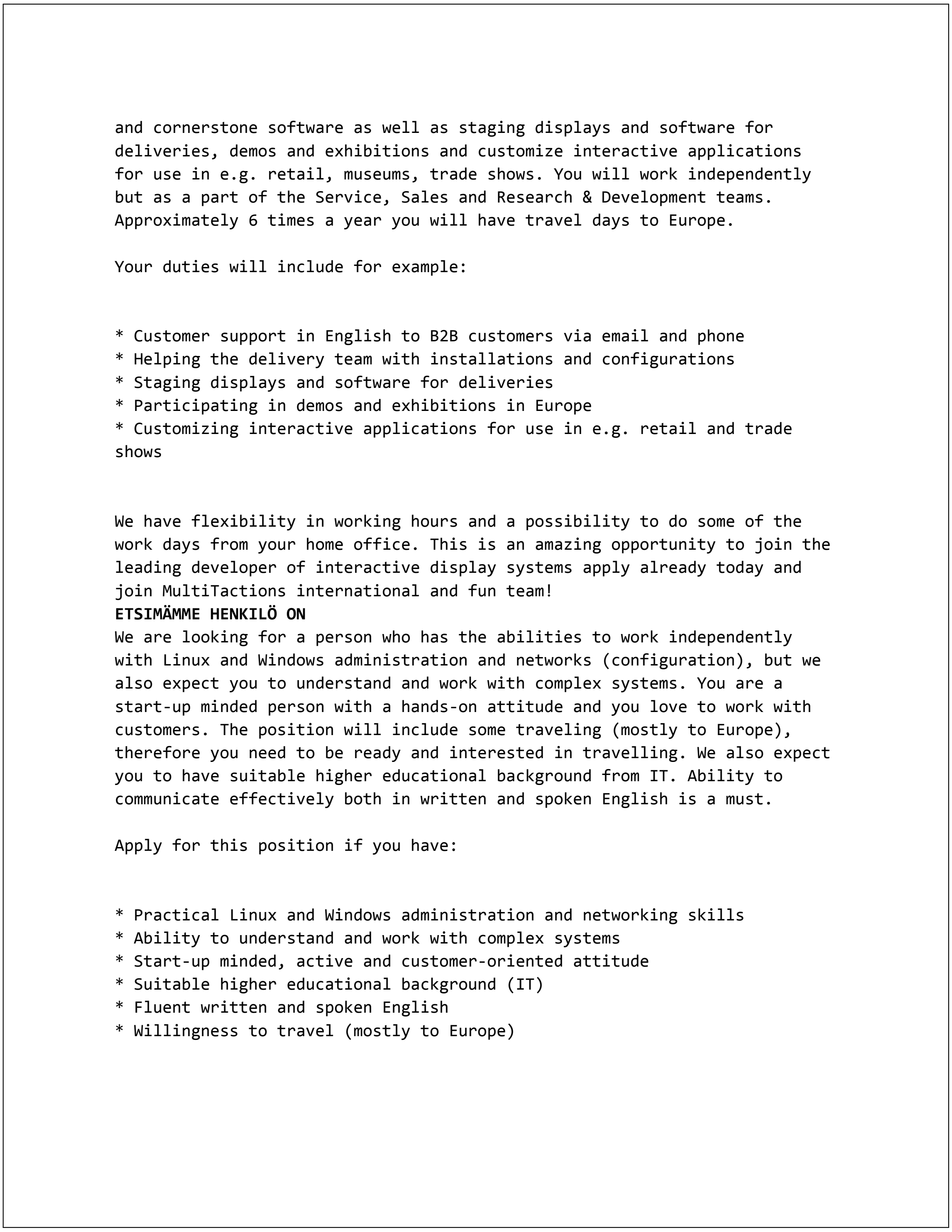
**KUVAUS**

Academic Work is looking for a full-time Junior Support Engineer for our client company MultiTaction Oy!

You will be recruited to our client company. Academic Work will take care of the recruitment process, therefore candidates are kindly asked to aim all questions to Academic Work candidate.service@academicwork.fi.

**TYÖNKUVA**

In this position, you will work with B2B customers and provide customer support via phone and email. You will work in English and you will have customers contacting you from all around the world. You will also help delivery team with installations and configuration of Multitaction cells



and cornerstone software as well as staging displays and software for deliveries, demos and exhibitions and customize interactive applications for use in e.g. retail, museums, trade shows. You will work independently but as a part of the Service, Sales and Research & Development teams. Approximately 6 times a year you will have travel days to Europe.

Your duties will include for example:

* Customer support in English to B2B customers via email and phone
* Helping the delivery team with installations and configurations
* Staging displays and software for deliveries
* Participating in demos and exhibitions in Europe
* Customizing interactive applications for use in e.g. retail and trade shows

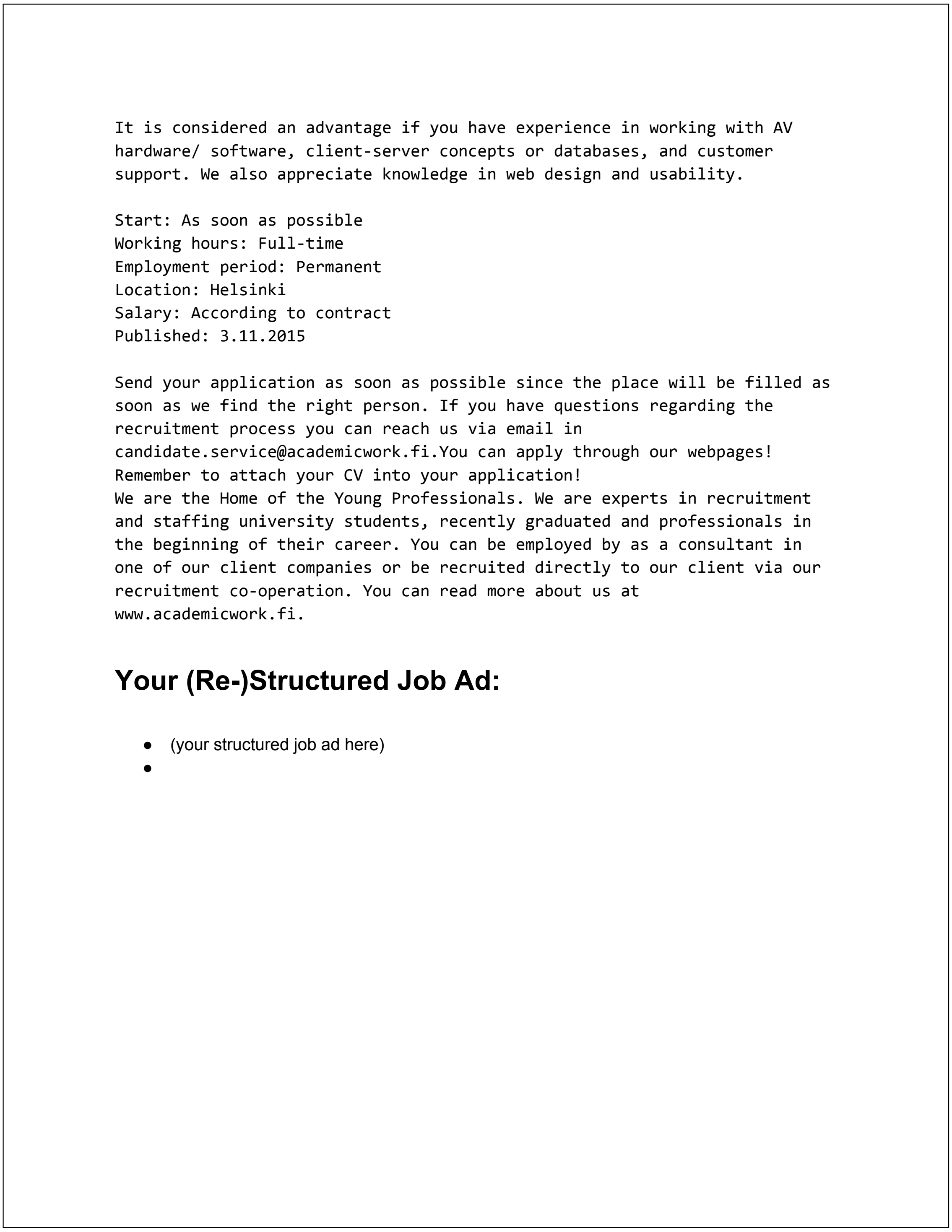
We have flexibility in working hours and a possibility to do some of the work days from your home office. This is an amazing opportunity to join the leading developer of interactive display systems apply already today and join MultiTactions international and fun team!

**ETSIMÄMME HENKILÖ ON**

We are looking for a person who has the abilities to work independently with Linux and Windows administration and networks (configuration), but we also expect you to understand and work with complex systems. You are a start-up minded person with a hands-on attitude and you love to work with customers. The position will include some traveling (mostly to Europe), therefore you need to be ready and interested in travelling. We also expect you to have suitable higher educational background from IT. Ability to communicate effectively both in written and spoken English is a must.

Apply for this position if you have:

* Practical Linux and Windows administration and networking skills
* Ability to understand and work with complex systems
* Start-up minded, active and customer-oriented attitude
* Suitable higher educational background (IT)
* Fluent written and spoken English
* Willingness to travel (mostly to Europe)



It is considered an advantage if you have experience in working with AV hardware/ software, client-server concepts or databases, and customer support. We also appreciate knowledge in web design and usability.

Start: As soon as possible
Working hours: Full-time
Employment period: Permanent
Location: Helsinki
Salary: According to contract
Published: 3.11.2015

Send your application as soon as possible since the place will be filled as soon as we find the right person. If you have questions regarding the recruitment process you can reach us via email in candidate.service@academicwork.fi.You can apply through our webpages! Remember to attach your CV into your application!
We are the Home of the Young Professionals. We are experts in recruitment and staffing university students, recently graduated and professionals in the beginning of their career. You can be employed by as a consultant in one of our client companies or be recruited directly to our client via our recruitment co-operation. You can read more about us at www.academicwork.fi.

## Your (Re-)Structured Job Ad:

- (your structured job ad here)
-

## A.5 Collected Labels in Paragraph Experiment

The following is the full collection of labels collected during the crowd-source paragraph labeling experiment described in Section 2.2.2:

```
{
    "Summary: Short introduction": [
        "intro", "job description", "job position", "description",
        "general", "job info", "general job description",
        "basic info", "job introduction ", "introduction",
        "general task", "general task ", "job",
        "job related field", "announcement", "announcement ",
        "about", "job description ", "overview",
        "position description", "job desription", "position",
        "kuvaus", "high level summary", "summary", "global work",
        "software integrations", "customer experience"

    ],
    "Person specification (Who you are)": [
        "required skill", "required skills", "capabilities",
        "skills", "skill", "requirements", "applicant’s abilities",
        "actual requirements", "wished skills",
        "requirements ", "requirement", "competence",
        "skill requirement", "skill exception",
        "candidate’s requirements", "prerequisite studies",
        "requirement ", "formal requirements", "job requirements",
        "important requirements", "job specific requirements",
        "common requirements", "start of skills list",
        "skills heading", "checklist for requirements ",
        "requirements title", "html5", "user-testing",
        "prototyping", "programming languages", "programming",
        "programming skills", "software development",
        "mobile development", "software developer",
        "language requirement", "languages", "langugage skills",
        "language requirements", "language", "language skill",
        "language skills", "web services integration", "software",
        "mgmt", "agile", "a360", "istqb", "bugtracking tool",
        "tools", "manollo", "clojure", "java", "scala", "design",
        "thread programming", "mobile", "spring", "angularjs",
        "additional skills", "bonus skills", "desired skill",
        "applicant’s personality", "personal", "soft skills",
        "personal skill", "social skills", "characteristics",
        "habits", "character", "attitude", "transaction management",
        "technical skills", "domain skill", "project management",
        "quality assurance", "analysis", "tracking", "testing",
        "communication skill", "travel requirements",
        "traveling requirements", "travel", "expected values",
        "time commitment", "time", "company’s needs", "performance",
        "employee qualities ", "degree", "job specific tag",
        "knwoledge", "education", "knowledge", "capability",
        "expectations", "background", "job challenge",
        "business-minded", "qualifications", "asset",
        "working hours", "detailed info", "innovation",
        "tech", "experience requirement", "experience",
        "integration experience", "previous experience",
        "work experience", "full time", "temps", "job form",
        "job classification ", "extent", "position type",
        "job duration ", "job type", "employment type", "work type"

    ],
    "Benefits (What we give)": [
        "marketing the job", "goal", "selling the job",
        "marketing of the company", "sales pitch", "remuneration",
        "motivation", "self-advertising", "benefits", "reward",
        "treats", "job benefits", "company’s treats",
        "opportunities", "additional job opportunities",
        "incentive", "additional opportunity", "salary", "offering",
        "we offer", "job offer", "compensation", "oppoturnity",
        "what you get", "offer"
    ],
    "Job specification (What you give)": [
        "job title ", "seniormanager", "clinicalcontractcoordinator",
        "search", "ask", "what", "work title", "need", "looking for",
        "title", "job title" "functions", "responsibilities heading",
        "role", "job role", "detailed task", "about the role",
        "duties", "checklist for job description ", "general tasks",
        "job responsibilities", "work summary", "task titles",
        "responsibilites", "main tasks", "tasks", "responsibility",
        "responsibilities", "task", "team", "unit",
        "unit description", "team title", "company branch", "level",
        "asw", "practicalities", "position level", "department",
        "organisational position", "unit of the company",
        "unit of company", "company unit descriptions",
        "location of job", "finlandjob", "location ", "job location",
        "where", "location data", "sijainti", "paikka", "place",
        "office", "job ad", "location", "finland", "duration",
        "time frame", "dates", "starting date", "time period",
        "conditions"
    ],
    "Company (Who we are)": [
        "manpower", "bayer", "bayer2013", "bayerr&d",
            "bayerrecruiting ", "kemira", "reaktor", "company name",
            "tieto", "paf", "company", "company info",
            "introduction to organization", "company description",
            "company ", "employer‘s information", "general company info",
            "history of employer", "company description ",
            "company facts", "organisatio", "employer info",
            "employer description", "website of employer",
            "website contact info", "website", "employer listing ",
            "service offering", "product line",
            "detailed information on productline",
            "company introduction", "intro to company", "who",
            "company presentation", "organization", "ccompany details",
            "company details", "organisation description", "about us",
            "company data", "industry", "field of operations", "retail",
            "business type", "consulting", "employer", "money games",
            "non-discrimination", "equal opportunity", "company vision",
            "company culture", "company policy"
        ],
        "Next steps": [
            "applying", "contract", "selection", "apply", "how to apply",
            "apply info", "application process",
            "application information", "application practicalities ",
            "practical information", "application instructions",
            "how to apply?", "instructions", "application submission",
            "application", "application procedure", "attachments",
            "application details", "opening date", "closing date",
            "deadline ", "deadline for application", "dealine",
            "application deadline", "deadline", "timetable",
            "auxiliary target group", "encouragement", "invitation ",
            "invitation" "employer listing", "more about position",
            "further info", "further information", "more information",
            "futrher information", "lisätieto", "more info",
            "additional information", "link", "contact details",
            "contact person", "contact info", "contact",
            "contact practicalities ", "contact person name ",
            "contact person title ", "contact person mail ",
            "job contact information", "job inquiry email",
            "phone number", "contacts", "contact ddetails",
            "help", "signature ", "contact ", "contact person title",
            "details", "contact information", "recruiter", "recruiting",
            "contact agent", "job agent", "about job agent ",
            "about job agent"
        ],
        "Other": [
            "empy", "-", "empty", "none", "nothing", "empty space",
            "no data", "start of list", "heading", "section title",
            "header", "introductory phrase ", "transition", "subheader",
            "headline", "question", "job id", "number", "job number",
            "keywords", "90% bullshit", "blabla",
            "signal for interview question", "bullshit", "useless info",
            "crap", "information", "info"
        ]
    }
```