Similarity and explanation for dynamic telecommunication engineer support.

MARTIN, K.

2021

Similarity and Explanation for Dynamic Telecommunication Engineer Support

Kyle Martin

A thesis submitted in partial fulfilment of the requirements of
Robert Gordon University
for the degree of Doctor of Philosophy

This research was carried out in collaboration with British Telecommunications (BT).

May 2021

Supervisor Prof. Nirmalie Wiratunga

Abstract

Understanding similarity between different examples is a crucial aspect of Case-Based Reasoning (CBR) systems, but learning representations optimised for similarity comparisons can be difficult. CBR systems typically rely on separate algorithms to learn representations for cases and to compare those representations, as symbolised by the vocabulary and similarity knowledge containers respectively. Deep Metric Learners (DMLs) are a branch of deep learning architectures which learn a representation optimised for similarity comparison by leveraging direct case comparisons during training. In this thesis we explore the symbiotic relationship between these two fields of research. Firstly we examine what can be learned from traditional CBR research to improve the training of DMLs through training strategies. We then examine how DMLs can fill the traditionally separate roles of the vocabulary and similarity knowledge containers. We perform this exploration on the real-world problem of experience transfer between experts and non-experts on service provisioning for telecommunication organisations. This problem is also revealing about the requirements for practical applications to be explainable to their intended user group. With that in mind, we conclude this thesis with work towards the development of an explanation framework designed to explain the recommendations of similarity-based classifiers. We support this practical contribution with an exploration of similarity knowledge to support autonomous measurement of explanation quality.

Keywords: Similarity, Deep Learning, Deep Metric Learning, Case Based Reasoning

Acknowledgements

I would like to take this opportunity to offer my deepest gratitude to my supervisor and mentor, Nirmalie Wiratunga. As a little birdy once told me, without being taken under wing, you can never learn to fly. You have been a mentor to me in every way imaginable, and I would not have made it halfway without you.

I would also like to thank BT for funding this work. In particular I must thank Anne Liret, who was my constant Skype companion for these last few years of my life. Much of Chapter 5 and 6 of this thesis is the result of many a Monday afternoon meeting. A special thank you also to Gilbert Owusu and Mathias Kern, for their commitment to fostering the success of this work, and the work of many other PhD students collaborating with BT.

I would like to thank my friends and colleagues in the RGU AI Research Group. In particular, Pamela Johnston, Anjana Wijekoon, Stewart Massie, David Corsar, Chamath Palihawadana, and Ikechukwu Nkisi-Orji. You have been a bastion of support and inspiration.

Finally and most importantly, I must thank my family and friends for everything. My beloved Amy, who has listened to my worries while many a film played in the background. My wonderful mum, Suzanne, who has done more for me than anyone else. My dad, Andrew, for many a cinema trip and game of Sunday golf. My brother Shaun and sister Chloe. My fantastic uncles Richard, Ryan and Russell and cousins, Dominic, Ollie and Jasper. Finally, to my Grandma Elizabeth and Grandpa Edward, who I am certain is crying into a tea-towel somewhere about now.

Declaration

I confirm that the work contained in this PhD project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed: Kyle Martin Date: 31/05/21

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Human experience composes one of the more difficult areas to manage in an organisational memory [1], yet sharing and managing expertise is a necessary aspect of knowledge management [2]. However, knowledge transfer between expert-level users is a difficult task [3]. When seeking to automatise such an interaction, it often means that a user is directed towards another expert user, rather than presented with specific information itself. For example, in [4] the authors built the PWC Connection Machine, a machine learning-based system which supported personnel struggling with a specific problem by recommending colleagues who had experienced similar problems, or knowledgeable trained experts. The proposed system did not offer an answer to the problem itself, but instead used the description of the problem to identify the most suitable person to resolve it. This is because the information sources that store experience are necessarily complex, as experiential content is difficult to elicit and therefore difficult to query effectively [5]. As a result, developing a knowledge model to cover all relevant aspects from historical experience would be infeasible. However, sufficiently understanding the similarity between work elements in expertise-reliant sectors could present an opportunity to improve knowledge transfer between users [4]. A methodology such as Case-Based Reasoning could be suitable to leverage this intuition while being easily explainable to users [6, 7].

Case-Based Reasoning (CBR) is a methodology founded on the paradigm that 'similar problems have similar solutions' [8, 9, 10]. The intuition behind this methodology has its roots in human psychology; as people, we learn from our previous experiences to inform future decision-making [9, 10, 11]. This means that the solutions produced by a CBR system often closely mirror human judgements in how they are approached. As a result, when implemented as a component of a machine learning system CBR is often pointed to as an easily explainable technique [6, 7, 12].

CBR is reliant on having records of known past experiences and their solutions ('cases') stored within a structure which can be easily queried (the 'case-base'). When a query is input, the case-base is searched to find the most similar cases to the query. Solutions of retrieved cases can then either be reused in their original form or adapted to suggest a solution which is appropriate for the query case [8, 9]. The query, a record of the proposed solution, and knowledge of its outcome (i.e. was the solution successful) can then be stored to contribute towards further decision-making. Thus the CBR methodology can be modelled as a four-stage process of retrieve (similar historical cases), reuse (the solutions of these cases if possible), revise (these solutions to meet the needs of the new query) and retain (the query case and its solution to improve the case base) called 'the CBR cycle' [8].

In a practical implementation of the CBR methodology, system functionality can be modelled through the interactions between four knowledge containers - vocabulary knowledge, similarity knowledge, adaptation knowledge and case-base knowledge [13]. Each of these containers are a method of formalising the level of knowledge captured by the system: the vocabulary container is indicative of the knowledge captured in the representation of case features; the similarity container is a formalisation of the knowledge in case or feature comparison; the adaptation container represents the known methods or rules to adapt a solution to meet the needs of another known case; and the case-base knowledge container summarises the extent of known cases in the case-base. These containers can be seen as another method of describing specific implementation details of the CBR cycle, whereby the vocabulary and similarity containers provide functionality for the system to retrieve similar cases, the adaptation container controls the process of solution revision and the number of cases captured within the case-base will impact whether retrieved solutions are suitable to be reused and where revised solutions should be retained. Typically, these containers are disjoint. For example, algorithms which support similarity calculations do not contribute towards the vocabulary container. This presents an issue, because in the case of representing human experience it can be expensive to construct a specialised representation and then receive feedback to understand how these should be compared for similarity calculations. It would be desirable for specifically the case vocabulary and similarity function to be learned simultaneously (meaning filling the vocabulary and similarity knowledge containers at the same time).

Deep metric learners are a branch of neural network architectures (including the Siamese Neural Network [14, 15] and Triplet Network [16]) which use similarity knowledge between input examples to improve representation and create a latent space optimised for similarity-based return [15, 17, 18]. They receive multiple examples as input simultaneously to develop embeddings which are optimised based on an objective. This

objective is defined by a 'matching criteria' - a principle which identifies whether two examples are similar or not. DMLs are interesting because they combine both representation learning and similarity functions into a single algorithm. This creates potential for DMLs to be integrated within a modern CBR framework. In particular, it offers a potential means to learn to learn how best to represent records of human experience such that the data is optimised for integration with a system which could enable knowledge transfer.

Furthermore, due to its reliance on reuse of existing cases, CBR allows explanations of its decision-making to be easily formed using comparisons (i.e. by "matching"), or by leveraging knowledge within the case-base itself [7, 12, 19]. The similarity and vocabulary knowledge containers within CBR present a good opportunity to structure explanation types to better meet the explanation needs of different users [20]. This is important in real-world circumstances where many different user groups frequently interact with the system [21, 22]. Beyond this, CBR's utility in the field of explanation hints at the capability of evaluating the quality of explanations autonomously using similarity metrics.

1.1 Research Hypothesis

We believe that DMLs are well adapted to support the task of transfer of experience within expertise-driven domains. The similarity between the CBR and DMLs suggest that the latter should be capable of emulating successful deployment of CBR systems for this task. Beyond this, DMLs have a key advantage in that they are capable of learning to represent complex data structures, whereas CBR's inability to do this (without an expensive knowledge engineering stage) could present a prohibitive barrier to development. We highlight the ability to optimise learned representations using similarity knowledge as an indicator that DMLs present an opportunity to fulfill the traditionally knowledge containers of 'vocabulary' and 'similarity' within a CBR system.

From another perspective, due consideration of previous research in traditional machine learning algorithms highlights an opportunity to improve the training efficiency of DMLs. Specifically we highlight techniques such as boosting in meta-learning, or clustering in CBR, which leverage locality knowledge to build a contextual awareness of the feature space and contribute towards better overall performance of the system. Given DMLs reliance on building representations based on direct comparisons of examples, we suggest that a training strategy which incorporates awareness of the distribution of examples within a locality could lead to faster network convergence.

Finally, given that similarity-based methods such as CBR are interpretable as they

mirror human decision-making, it seems intuitive to suggest that the output of DMLs could also be explained in the same manner. We therefore believe that despite being deep learning architectures whereby the features learned are not necessarily understandable to humans, DMLs' reliance on similarity knowledge presents an opportunity to explain their decisions. As part of this, we suggest that similarity-based explanation methods are well placed to support multiple user groups with escalating requirements from an explanation. This is because the granularity of the similarity comparison can be adjusted to suit the expertise of the user (i.e feature-feature comparisons for experts, case-case comparisons for non-experts, etc). We therefore posit that DMLs can be explained by leveraging similarity knowledge, and the explanation can be targeted to meet the needs of different user groups.

To summarise, we hypothesise that similarities between CBR and DMLs present an opportunity for integration where both methods will benefit. Specifically we anticipate that training of DML architectures can be improved by considering clustering research from other machine learning techniques, and that DMLs present an opportunity to combine the similarity and vocabulary knowledge containers as a component of a CBR system. Furthermore, as DMLs are fundamentally similarity-based architectures we believe that their output can be explained effectively in situations where multiple user groups of varying domain expertise are using the system.

To ensure systematic investigation of these claims, we deconstruct the hypothesis into the following research questions:

1. How can techniques from traditional machine learning methods (such as CBR and meta-learning) be incorporated into strategies to improve training efficiency of DMLs?

2. How effective are DMLs at fulfilling the traditionally separate roles of the 'vocabulary' and 'similarity' knowledge containers in the context of transfer of experience between experts and non-experts of telecommunications engineering?

3. How can we explain the output of similarity-based architectures (including DMLs) intended to support user groups of varying domain expertise, and how can we autonomously evaluate the quality of produced explanations?

Each question underpins one aspect of the research we have performed towards proving our hypothesis. For each research question, we dedicate a chapter in this thesis to describe our work towards answering that question. Therefore, in this thesis we present three original and novel contributions towards research in the areas of CBR, Deep Metric Learning and ecplainability:

1. We introduce several training strategies for DMLs which are inspired by research in meta-learning, curriculum learning and CBR. Experiments on public datasets from multiple domains illustrate that the proposed strategies improve training efficiency of DML architectures.

2. We compare methods of developing a similarity model for transfer of experience using free-text data sources. Our findings demonstrate that DMLs can learn to produce representations optimised for similarity calculations which offer clear improvement over dense representations gained from word embeddings, but require refinement to outperform statistical methods.

3. We describe the development of an explainability framework based upon one of our use-cases, and assess the quality of these explanations using novel autonomous evaluation methods and user feedback. The results highlight the practical utility of a hierarchical explanation framework.

For each of these primary contributions, we also explore a number of secondary contributions in the respective chapters. These secondary contributions will inspect and discuss in greater granularity specific advancements we have made in each of the research fields which are explored through our primary contributions. Our hypothesis, research questions and contributions are summarised in Figure 1.1.

1.2 Methodology

We evaluate the contributions of this thesis on two real-world use cases considering the transfer of experience between expert and non-expert personnel within a telecommunications organisation. The goal of these use cases was to build a system which was capable of enabling both engineers and desk-based agents to leverage the experience of other field engineers to better inform their decision-making. In the first use case, we target recommendation of additional information to support risk management based on previous tasks, while in the second use case we aim to support transfer of experience between desk-based agents and engineers. We evaluate these use cases on novel real-world datasets gathered from engineer notes in the telecommunication domain, which act as our records of experience. Finally, we also demonstrate a general framework to develop explanations using these notes as a foundation.

Both of the use cases explored in this thesis rely on textual data as input in the form of telecommunication engineer notes. In use case 1 we attempt to elicit specific information from engineer notes to recommend additional information, while in use case 2 we aim to recommend the appropriate task intervention for a desk-based agent using the notes. We consider CBR as a promising methodology to achieve these aims, and

Hypothesis

We hypothesise that similarities between Case-Based Reasoning and Deep Metric Learners present an opportunity for integration where both methods will benefit. Specifically we anticipate that training of DML architectures can be improved by considering clustering research from other machine learning techniques, and that DMLs present an opportunity to combine the similarity and vocabulary knowledge containers as a component of a CBR system. Furthermore, as DMLs are fundamentally similarity-based architectures we believe that their output can be explained effectively in situations where multiple user groups of varying domain expertise are using the system.

	Chapter 4	Chapter 5	Chapter 6
Research Question	How can techniques from traditional machine learning methods (such as CBR and meta-learning) be incorporated into strategies to improve training efficiency of DMLs?	How effective are DMLs at fulfilling the traditionally separate roles of the 'vocabulary' and 'similarity' knowledge containers in the context of transfer of experience between experts and non-experts of telecommunications engineering?	How can we explain the output of similarity-based architectures (including DMLs) intended to support user groups of varying domain expertise, and how can we autonomously evaluate the quality of produced explanations?
Primary Contribution	We introduce several training strategies for DMLs which are inspired by research in meta-learning, curriculum learning and CBR. Experiments on public datasets from multiple domains illustrate that the proposed strategies improve training efficiency of DML architectures.	We compare methods of developing a similarity model for transfer of experience using free-text data sources. Our findings demonstrate that DMLs can learn to produce representations optimised for similarity calculations which offer clear improvement over dense representations gained from word embeddings, but require refinement to outperform statistical methods.	We describe the development of an explainability framework based upon one of our use-cases, and assess the quality of these explanations using novel autonomous evaluation methods and user feedback. The results highlight the practical utility of a hierarchical explanation framework.
Secondary Contribution	1. Taking inspiration from current research into optimising the training of DMLs and historical research into meta-learners, we introduce two methods for informed pair selection (DYNE and DYNEE) that optimise pair creation by leveraging the concepts of exploration and exploitation. 2. Encouraged by the results of recent work in curriculum learning we introduce a pair complexity heuristic for ordering that draws on knowledge about the neighbourhood properties of pairs. 3. Building on the limitations of our pair selection strategies, and motivated by techniques from CBR, we present an incremental locality-sensitive batching strategy for triplets (LSB) which allows the batching to evolve alongside example representations over the course of training.	1. We examine our ability to learn task similarity using expert-written documents (engineers' notes) provided by a telecommunication organisation. 2. We introduce two real-world use cases to highlight the real world applicability of the proposed methods. The first use case examines recommendation of additional information to perform dynamic decision support for engineers in the field. The second use case examines the transfer of experience between expert and non-expert personnel within the telecommunications work sector. We demonstrate how both of these use cases are achievable by learning similarity models empowered by DMLs. 3. We perform a short comparative study of developing representations from expert-written documents for similarity-based return on the basis of their accuracy on two simple classification tasks from our use cases.	1. We outline the development and implementation of a modular explainability framework and detail several of its sample modules as applied to the real-world problem of supporting desk-based planning agents in the telecommunications engineering domain. 2. We perform a qualitative evaluation to understand user opinion on the quality of provided explanations with feedback from two user groups of different levels of expertise. The results indicate that the judgement of what forms a good explanation changes based on domain-expertise: experts preferred explanations to mirror their reasoning, while non-experts emphasised task performance. 3. We explore the correlation between the quality of an explanation and similarity knowledge within the latent space using two novel metrics: Meet-in-the-Middle (MITM) and Trust Your Neighbours (TYN). Our results highlight that similarity is a promising starting point to model the quality of explanation.

Figure 1.1: The hypothesis of this thesis as broken down into research questions. We answer each question through our contributions in the corresponding chapters.

so in both use cases our goal is to develop a model capable of assessing the similarity between notes. We have nominated deep metric learning algorithms for this purpose. DMLs, like most machine learning algorithms, require that text is converted into a structured or numerical representation to elicit specific knowledge for algorithms to leverage. Therefore it is relevant for us to examine methods of text representation, and in this thesis we consider both statistical methods (term-frequency / inverse-document-frequency) and learned methods (Word2Vec) to develop representations for text before input to DML algorithms. In this way, we assess the real-world impact of our research. We discuss this further in Chapter 5.

Discussions with real users highlight several pragmatic truths about deploying machine learning algorithms for use in the field. In particular, we highlight the requirements for explanation of decisions made by these algorithms. Previous attempts to use engineering notes as a source for skills-based matching of telecommunications engineers with tasks [23] within our industry partner had been met with resistance when deployed, largely due to lack of user understanding. Therefore we take a different route to provisioning explanations. We use co-creation with real users to understand their needs from explanation, which informs the development of a hierarchical explanation framework with increasing levels of explanation complexity and context awareness. Feedback from real users enables us to propose methods of autonomously evaluating the quality of explanations, and measure their correlation. In this manner, our methodology for approaching explanation of decisions is to work with users to identify good explanations first, and then develop metrics for evaluation quality second. We discuss this in further detail in Chapter 6.

Before exploring the practical aspects of deploying our research, it is important to ensure its theoretical validity. We propose novel training strategies to improve the training of DMLs, with the intention to apply these to our use cases. However we are aware that we cannot share the data which informs our exploration, as it is proprietary to our company partner. Therefore, we evaluate the proposed strategies on a range of datasets and problem domains. The goal of approaching the thesis in this manner is to benchmark our methods against public datasets and enable reproducibility in accordance with best research practice. We provide further details the development and evaluation of our proposed training strategies for DMLs in Chapter 4. To this end we provide an introduction to each of the datasets used in this chapter here.

Image Classification

MNIST [24] is a handwriting recognition dataset comprised of 70,000 greyscale images of handwritten single-digit numbers, divided into a training set of 60,000 images and

a testing set of 10,000 images. Images are 28×28 pixels and have one of ten classes (the numbers zero to nine). Classes are equally represented throughout the dataset (i.e. each class has 5,000 examples in the training set and 1,000 examples in the test set).

CIFAR10 [25] is an object recognition dataset comprised of 60,000 colour images, divided into a training set of 50,000 images and a test set of 10,000 images. Each image is 32×32 pixels in size and features one of ten distinct objects that are used to identify its class label (airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck). These classes are equally represented throughout the dataset (i.e. each class has 5,000 examples in the training set and 1,000 examples in the test set).

STL-10 [26] is an object recognition dataset comprised of 13,000 labelled and 100,000 unlabelled colour images extracted from ImageNet. We only utilise the labelled images in our experiments, which are divided evenly between 10 classes (airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck). The images are 96×96 pixels in size which is substantially larger than examples from CIFAR10 or MNIST and making it a challenging benchmark to test the scalability of our proposed methods, as well as much closer to the size of commercial images.

Human Activity Recognition

The **SelfBACK** [27][1] dataset features time series data collected from 34 users performing 9 different activities over a short period of time (lying, sitting, standing, walk-slow, walk-med, walk-fast, jogging, upstairs, and downstairs). Data was collected by mounting a tri-axial accelerometer on the thigh and right-hand wrist of participants at a sampling rate of 100Hz. Within this thesis we use a subset of the full SelfBACK dataset, where we combine the classes walk-fast, walk-med and walk-slow into a single class (walk) and we remove the class lying. As a result, we have a six-class classification problem, divided between two data sources. We refer to these as SelfBACK-Wrist and SelfBACK-Thigh in our experiments.

Text Classification

The Large Movie Review Dataset [28] is comprised of 50,000 labeled film reviews scraped from the Internet Movie Database (**IMDB**). Polarized reviews have been extracted and labeled as either 'positive' (where a review score is greater than 6) or

[1]The SelfBACK project is funded by European Union's H2020 research and innovation programme under grant agreement No. 689043. More details available: http://www.selfback.eu. The SelfBACK dataset associated with this paper is publicly accessible from https://github.com/selfback/activity-recognition

'negative' (where a review score is lower than 4) to create a binary sentiment analysis task. Though the dataset also contains a large number of unlabeled reviews, we did not use these in any experiments. We selected the IMDB dataset as its boundary is naturally complex due to the presence of both concept complexity and subjective judgment.

The **Reuters** dataset [29] is a document classification dataset comprised of structured newswire articles. We used the ModApte subset of the Reuters-21578 benchmark, which contains 11,228 documents each given one of 46 labels. Reuters was selected as it is particularly challenging for minibatch approaches, given its inherent data imbalance.

1.3 Chapter List

This thesis is structured in the following manner.

Chapter 2: Related Work. In this chapter we contextualize the contributions of this thesis by looking at inspirational and related work. In particular, we explore knowledge containers as a method of formalising the knowledge captured within a CBR system. Additionally, we present the roots of deep metric learning and efforts to optimise their training procedures. This allows us to identify overlaps between the functionality provided by DMLs and the responsibility of individual algorithms within the knowledge containers. Finally, we discuss the wider field of explanation of machine learning algorithms, focusing specifically on methods which utilise similarity and some of the gaps in literature in this regard.

Chapter 3: Technical Aspects of Machine Learning. This chapter introduces the fundamental technical work required to understand the contributions of this thesis. We provide an introduction to the underlying concepts of similarity-based return, representation learning and deep learning in general. Finally, we discuss in detail the mathematical basis for the deep metric learning algorithms used throughout this research and demonstrate how they are linked to CBR and Deep Learning algorithms respectively.

Chapter 4: Similarity Knowledge for Training Deep Metric Learners. In this chapter we present our foundational work towards the advancement of training DMLs by incorporating lessons from CBR research. We focus on the matching-based Siamese Neural Network, and present several novel training strategies using pair-mining approaches inspired by boosting in meta-learners. Our results show that similarity-based training schemes have potential, but experiments on four datasets demonstrate the expense of using similarity knowledge is prohibitive to more complex datasets and architectures. Motivated by these findings, we are inspired to develop a similarity-based

training strategy for more complex DMLs, such as Triplet Networks, which is robust to large or complex datasets and architectures. The result is Locality-Sensitive Batching (LSB), a method which leverages approximate-Nearest Neighbour (a-NN) mechanisms, in particular Locality-Sensitive Hashing (LSH), to incorporate similarity knowledge into a training strategy for triplet networks. Our results over five different datasets demonstrate the performance improvements offered by LSB in different domains.

Chapter 5: Similarity Knowledge for Transfer of Experience. Here we contextualise the contributions of this thesis through two real-world use cases taken from the domain of telecommunications. The overall goal of these use cases is to support decision-making by allowing personnel to leverage the previous experiences of domain experts. In the first use case we present a method of recommending additional information to support field engineers to complete complex work tasks on telecommunications equipment. This use case demonstrates experience transfer between domain experts. In the second use case we present a case-based recommender system to support desk-based planning agents to retrieve actionable knowledge from engineer updates in the form of notes. This use case aims to highlight experience transfer between domain experts (engineers) and domain non-experts (planning agents). Both of the applications are reliant on similarity models learned from a textual information source provided by engineers to describe their daily routine. This allows us to contextualise our contributions to training DMLs in Chapter 4 by applying them in practice. Beyond that, it highlights the practical considerations for putting these algorithms into use in the real-world, motivating our contributions towards explainable similarity-based machine learning architectures in Chapter 6.

Chapter 6: Similarity Knowledge to Support Explanation Explanation mechanisms for intelligent systems are typically designed to respond to specific user needs, yet in practice these systems tend to have a wide variety of users. This can present a challenge to organisations looking to satisfy the explanation needs of different groups using an individual system. In this chapter we present an explainability framework formed of a catalogue of explanation methods, and designed to integrate with a range of projects within a telecommunications organisation. Explainability methods are split into low-level explanations and high-level explanations for increasing levels of contextual support in their explanations. We motivate this framework using the specific case-study of explaining the conclusions of field network engineering experts to non-technical planning staff and evaluate our results using feedback from two distinct user groups; domain-expert telecommunication engineers and non-expert desk agents. We also present and investigate two metrics designed to model the quality of explanations - Meet-In-The-Middle (MITM) and Trust-Your-Neighbours (TYN). Our analysis of these

metrics offers new insights into the use of similarity knowledge for the evaluation of explanations.

Chapter 7 Conclusion. In this chapter, we summarise the outcomes of each chapter and how they evidence the contributions we have discussed above. We offer some conclusions, and finish this thesis with some ideas for future work to continue pursuing these research avenues.

Chapter 2

Related Work

This chapter provides contextualisation on the contributions of this thesis by examining relevant machine learning methodologies to enable transfer of experience between expert and non-expert personnel. This acts as a backdrop for our real world use case in this thesis, using the real problem of providing decision support within the field of provisioning services for telecommunication organisation

Firstly we examine work surrounding Case-Based Reasoning, paying particular attention to the foundational concept of knowledge containers, with an emphasis on the similarity and vocabulary containers. Following this we examine deep metric learning research in detail, examining their composition and strategies to improve their performance. In particular, we highlight links between CBR and DML research which offer a platform for improvements in both methodologies. We will build on this relationship in Chapters 4 and 5, where we demonstrate how DMLs can support delivery of CBR components, and how lessons from CBR research can lead to improvements for the training of DMLs. Finally, we provide an introduction to the growing work in explainability. Based on the literature, we define the key concepts and terminology within explainability research. We focus on the need for an explanation from a user's perspective, and the ramifications that this has on methods to evaluate the quality of an explanation. We conclude with a discussion on the explanatory qualities of CBR and DML architectures. In particular we highlight how explanation fits alongside the knowledge containers from CBR research. This supports our contribution presented in Chapter 6.

2.1 Case-Based Reasoning (CBR)

CBR offers a structured method to develop suitable solutions for novel new problems by leveraging information from a case-base of previously encountered cases. Each of these cases describes a specific problem that was encountered (i.e. customer classification for direct marketing [30], or a client requiring a recommendation for teaching materials to access [31]), the corresponding solution to that problem (i.e. a class label, or the most highly ranked recommendation) and the outcome of that solution (was it correct, or did the user accept the recommendation) [11]. The case-base is accessed and maintained through a four-stage process of retrieve, reuse, revise and retain called 'the CBR cycle' [8, 32]. In a typical example, when a new problem is presented to a CBR system as a query, it will first search its case-base to identify the most similar cases to the query (retrieve). If the problem is sufficiently similar, then the solution of the retrieved problem may be suggested as an answer to the query (reuse). More commonly, the solution will require some adaptation to make it more suitable to answer the current situation (revise). Pending acceptance from the user, a record of the new problem-solution pair will then be stored in the case-base to inform future iterations (retain).

From the example above, it can be observed that implementation of specific functions will be important at several points throughout a CBR system. Aspects such as how problems can be uniformly described, how they can be consistently and fairly compared, or how to guide adaptation so a realistic solution is generated, are important practical considerations. These have been formalised in the literature as 'knowledge containers' [13].

We can model a CBR system as the interaction between four knowledge containers - vocabulary knowledge, similarity knowledge, adaptation knowledge and the knowledge present in its case-base [13]. These containers act as a means to model the available knowledge within a CBR system. The vocabulary container is indicative of the knowledge which is inherent in the representation of case features, while the similarity container formalises the knowledge of how these features can be compared to judge which cases are similar [33]. The adaptation knowledge container quantifies the knowledge which is locked within the revision stage of the CBR cycle, commonly examining the alignment between the query and most similar solution from the case base [34]. Finally, the case-base knowledge is representative of the unique scenarios and problem types which are covered within the stored cases. For example, we can use case-base knowledge to judge areas of low representation for case acquisition [35].

To concretize the description of the relationship between the CBR Cycle and knowledge

containers, consider the following example of a simple theoretical CBR system. The system is designed to offer recommendations for a recipe given knowledge of available ingredients and a user's personal likes and dislikes of ingredients. A case is formed of a problem component (a finite list of available ingredients and an identical list describing whether the user likes those ingredients), a solution component (a recipe for a meal) and an outcome (whether the user enjoyed the meal). This case structure and associated knowledge forms the basis of the **vocabulary** container, while the existing historical cases already captured within the system forms the **case-base** container. Cases can be compared using a similarity table which has been provisioned by a domain expert that describes the similarity of each ingredient to every other ingredient on the basis of flavour (forming the **similarity** knowledge container). When a new user wishes to have a recipe recommended, they will create a query formed of only the problem component (i.e. the ingredients they have available and their personal likes/dislikes). The CBR system will compare the query to every case in its case-base using the similarity table and *retrieve* the most similar historical case. If the problem component of the retrieved case is an identical match to the query (i.e. the available ingredients and personal likes/dislikes are identical), then its solution will be *reused*. If the problem component does not match, the system will *revise* the retrieved solution using the similarity table to replace unavailable or disliked ingredients with available or liked ingredients (forming the basis of **adaptation** knowledge). The adapted recipe will then be recommended to the user, who can feedback on the recommendation. The new case (formed of the original query, the adapted solution, and the outcome gained from user feedback) is then *retained* in the case-base.

An interesting characteristic of knowledge containers is the co-called 'knowledge trade-off' which exists between them. It is rare for a CBR system to have all four knowledge containers fully developed, due to the expense of procuring knowledge to fill these containers or the complexity involved in improving them[36]. Instead, it is significantly more common for one or multiple of the containers to be lacking, while other containers are more fully developed and capable of maintaining performance of the system as a whole [37]. For example, a case-base reasoning system which focuses on similarity knowledge may have less capacity for adaptation, but a combination of robust similarity knowledge and a large case-base will still ensure satisfactory performance [4, 37].

As a result, functionality for each of these containers is provided by separate algorithms. For example, the similarity container is usually filled with local similarity knowledge - in a textual CBR system this may leverage semantic similarity knowledge gained from an ontology [38] while its vocabulary container utilises complex text features. Although the similarity container will operate on the contents of the vocabulary container, that

is not to say that their functionality is tied together within the same algorithm. An interesting aspect of deep metric learners, which we will discuss in greater detail in the upcoming section, is that they present an opportunity to cover multiple knowledge containers in a single algorithm.

2.1.1 Vocabulary Knowledge

Unlike many other traditional machine learning methods, CBR does not require an explicit domain model. In its place there is an emphasis on identifying recurring elements common across the scenarios or data points that make up the real-world objective which is trying to be solved [11]. The recurring elements are collected together under whether they describe (1) the problem, (2) the solution or (3) the outcome of the solution when applied to that problem. Each unique problem-solution-outcome grouping is described as a 'case', while the recurring attributes common across cases are called 'features'. Each feature depicts a data point that describes one specific aspect of a case. To illustrate this description with an example, consider a CBR system designed to categorise data describing bears into their species. The features within the problem component of the case would be a list of attributes describing the bear (i.e. fur colour: white, height: 200cm, etc) while the solution component would contain the class label (i.e species: polar bear), and the outcome would describe whether whether the bear was categorised correctly or incorrectly (i.e correct classification). The vocabulary knowledge container is primarily concerned with the selection and representation of these features [13].

In simple CBR systems, a case is represented as a collection of key-value pair features derived from a structured data source [39]. However, in many real-world situations these structured features are unavailable and the vocabulary to represent a case must be learned from unstructured data [40, 41]. This is described as learning a representation of the case, as different models and learning parameters will allow different representations of the same data to be learned. Representation learning is a vast field, so we focus our attention on only the area which is most relevant to our use case with telecommunication engineers. The notes detailing complex task information are recorded in text, and so in this work we consider approaches to learn a representation for text documents. Typically machine learning algorithms are incapable of accepting raw text as meaningful input. Instead, the text must be pre-processed in order for the algorithm to better leverage the complex knowledge structures captured in text. The process of converting a textual data source (be it a passage of text, a single document, or an entire corpus) into a numerical vector representation is collectively referred to as learning representations for text. By representing the document as a vector, quantitative functions can be performed upon qualitative data to facilitate information extraction by autonomous systems. Text representation methods can be broadly

categorised into two types, statistical methods and learned methods.

Statistic-based methods learn a representation for documents within a corpus by comparing quantifiable features. Individual methods achieve this by examining the presence/absence of key terms in a document (i.e. binary vectors) or comparing the frequency of terms within a document to how common they are across the corpus (i.e. tf-idf). Some recent work in this field directly incorporates aspects such as compactness of terms and how early in a document the word first appears [42]. The representations which are learned as a result of using a statistical method will be similar if the documents have a similar distribution of words, and dissimilar otherwise [43]. Perhaps the most popular statistical method, Term frequency-inverse document frequency (tf-idf) is a statistical measure designed to quantify as a real value the importance of a set of given terms within the context of a document in a corpus [44]. The value for each term is calculated by dividing the frequency of its usage within a document over the number of documents which contain the term within the corpus [45]. Therefore, each feature of a document vector is a value which represents an individual word from the corpus vocabulary and so vectors can be very sparse. As a result, tf-idf becomes steadily less effective in large corpora with a varied vocabulary.

Learned methods of text representation develop a representation for a document by modelling latent knowledge captured with the text, and are closely associated with idea of word embeddings. Word embeddings are a term used to generally describe a family of learned methods which use neural architectures to model syntactical and semantic knowledge in passages of text [46, 47, 48, 49]. Word embeddings generally operate upon the assumption that words which are similar in meaning will occur in similar contexts. These assumptions refer to semantic relatedness between words, which is a broader concept than semantic similarity. Semantic similarity is primarily concerned with 'likeness' relations, such as the relations between synonyms. Semantic relatedness on the other hand, concerns meronymy and associative relations (such as the relationship between the words 'pencil' and 'paper') as well as semantic similarity. Semantic relatedness is powerful as it allows the modeling of association between concepts, and the measurement of similarity between concepts in a vocabulary.

As an example of a learned method, Word2Vec is a deep learning algorithm which develops a vector representation of a document based upon the co-occurrence of keywords within a pre-defined range dubbed 'the sliding window' [46, 47]. Word2Vec uses contextual knowledge from the measurement of word co-occurrence to capture latent syntactical features and develop word embeddings. The result is a feature space where words that have similar contexts exist close together. What makes Word2Vec unique

amongst similar algorithms of this nature, is the fact that the vector describing the document produced as output will be optimised based upon the semantic relatedness of the keywords, so that the model can preserve the linear regularities that exist between words [46]. Additionally, longer passages of text can be represented by combining the embeddings for individual words. In its simplest form, this can be achieved by taking the average of word2vec embeddings for terms in a document. However, more complex methods of learning from word embeddings also exist. Perhaps the most popular method is learning a vector to represent the desired passage of text (i.e. paragraph or document) in parallel to learning word vectors [50, 51]. The process of building a representation of a document from Word2Vec embeddings is known as Doc2Vec [50].

2.1.2 Similarity Knowledge

The concept of similarity, and understanding what makes two cases similar, is a central underpinning of the CBR methodology. Many researchers point to similarity as the single most important aspect of a CBR system [9, 52].

Much like the work in case representation described in the above discussion of the vocabulary knowledge container, popular methods of learning similarity information between data points have previously been reliant on engineered features and hand-crafted metrics, such as taxonomies [53]. This is particularly true in systems designed for domain-specific or complex problems. While some work has examined the automated learning of similarity metrics using hand-crafted or elicited features [54], CBR is a methodology which is generally adopted to avoid the expensive process of explicit knowledge modelling. Increasingly we witness this transition towards global similarity metrics. Global similarity metrics consider the full representation of a case in order to perform its similarity calculations, informed either by amalgamating a breakdown of local feature-based similarities [55, 56], or by transforming cases into numerical vectors to allow distance comparisons as a proxy for similarity [57, 58, 59]. In this thesis, we focus on this latter group of vector-based similarity metrics, as they are closely tied with the learned representations described above.

Vector-based similarity comparisons can quickly become computationally expensive, particularly in large or complex case-bases. Often these reflect real-world use cases where the number of cases within the case-base, or features within a single case, are too large to perform efficient back-to-back comparisons. This problem is often referred to as the curse of dimensionality [60]. There have been concentrated research efforts with the intent of reducing the complexity of this process [61, 62, 63, 64, 65]. Specifically we highlight work which leverages similarity knowledge to cluster the case-base and reduce

the complexity of case retrieval. Examples include cluster-based retrieval from large-scale case-bases of image [61], text [62] and even simulation data [63]. Furthermore, research in this field has also established that the coverage knowledge generated by clustering approaches can be exploited for sample selection. For example, in [66] the authors use clustering methods to identify the most important cases for labeling from an unlabeled set. However, despite the performance gain it can provide, identifying suitable clusters can be a very expensive process. A number of methods have evolved to estimate suitable cluster compositions while maintaining computation efficiency. These methods are called approximate-Nearest Neighbour algorithms (a-NN).

A-NNs are a set of techniques to inexpensively perform neighbourhood computations on large sets of examples. The goal of these algorithms is to approximate similarity calculations while maintaining low computational cost. They offer a means to extract similar examples from a case-base at a fraction of the cost of brute-force nearest neighbour methods such as kNN, with the drawback of usually being less accurate [64, 65]. A specific example is Locality-Sensitive Hashing (LSH). LSH is a data independent a-NN method to economically estimate nearest neighbour computations by randomly dividing the feature space into distinct areas known as 'buckets', which preserve locality knowledge from the original space [67]. When a query is presented, it is indexed into a bucket. Similarity metrics are therefore performed only between a query and the contents of the relevant bucket to establish similarity knowledge in that neighbourhood. The trade off between computational efficiency and accuracy is the decided by the number of projections into the space. Too many projections will result in sparsely populated buckets and less likelihood of their contents being representative of the true distribution of examples within a particular locality. This will have a knock-on effect on the quality of the clusters and the accuracy of the similarity-based return. Meanwhile, fewer projections will mean that localities will be ill-defined, vulnerable to outlying examples, and offer less improvement in computational efficiency. Due to their impact on performance, several works have examined moving from random projections towards data-driven or lattice-based hashing [68].

There are other methods which aim to reduce the complexity of similarity calculations. For example, several works propose the use of representative cases to characterize groups of similar cases, thereby enabling similarity comparisons only to be performed against the representative of that group [69, 70]. Though works suggest different mechanisms and names (i.e. exemplary [69], footprint [70], prototype [71]) for the process, the overall goal of each algorithm is broadly aligned; identify cluster representatives on the basis of finding cases whose features characterise the knowledge captured within a specific locality of the space. The reduction in the cost of similarity comparisons

enabled by identifying representative cases has enabled the use of CBR methodology even in very complex fields such as medical systems [71, 72] and workflow model management [73]. An interesting aspect of these approaches is that they mirror the roots of CBR, which is based on the concept of human beings learning from previous experience. Just as people learn from specific past experiences, they also apply judgement from a summation of all relevant previous scenarios in their decision-making [74]. This link could empower prototypical approaches to be more explainable, as literature suggests methods which mirror human decision-making are easier to explain or justify. We discuss explainability of CBR and similarity return based systems in greater depth in Section 2.3.

2.2 Deep Metric Learning

We look to deep metric learning algorithms as an opportunity to bridge the gap between the similarity and vocabulary knowledge containers within a CBR system. Deep Metric Learners (DMLs) are a branch of neural network architectures (including the Siamese Neural Network [14, 15] and Triplet Network [16]) which use similarity knowledge between input examples to improve representation and create a latent space optimised for similarity-based return [15, 17, 18]. They receive multiple examples as input simultaneously to develop embeddings which are optimised based on an objective. This objective is defined by a 'matching criteria' - a principle which identifies whether two examples are similar or not.

The matching criterion is central to learning in DMLs to identify similar examples in a dataset. Since the expected outcome of training a DML is to have instances deemed similar to be mapped closer together, the matching criterion is a crucial consideration in ensuring that network training reflects the objectives of the problem at hand. Typically, class information will be used for the matching criteria [75, 76], and DMLs have been to shown to be robust to scenarios where class information is limited [17, 77] or there are so many classes as to not be useful [18, 78]. Even in situations where class information is completely unavailable, DMLs can be trained provided that another matching criteria can be identified. For this reason, recent work has demonstrated these networks can perform effectively even when working with extremely limited training data, such as in a one-shot learning environment [17]. Due to this they have demonstrated application in areas where fine-grained similarity knowledge is important, such as face verification [18] and similar text retrieval [79].

Deep Metric Learning algorithms can be divided into three components: (1) the architecture of the neural network itself; (2) the loss function which controls metric learning; and finally (3) the selection of relevant samples to refine network parameters [80] (see

Figure 2.1). In the following subsections we will discuss each of these aspects in more detail.

2.2.1 Deep Metric Learning Architectures

The essence of deep metric learning is to learn a representation of the input data guided by direct comparisons of examples to build an understanding of the relationships between cases. In CBR terms, this could be described as leveraging similarity knowledge to improve the vocabulary container. To achieve this, learning requires comparisons of multiple examples to judge similarity and the network architecture must facilitate these comparisons. This means that the network must be capable of receiving as input multiple examples, and also comparing them in some manner. A number of network architectures have evolved around this methodology [14, 16, 77].

The Siamese Neural Network (SNN) is a deep learning architecture which trains upon pairs of input data to learn a metric space in which training instances can be placed. The expectation of paired learning is that the learned space can better represent salient relationships between pairs which can then be better captured in a latent space. Originally used in binary classification tasks such as signature verification [14] and face recognition [15], SNNs have recently been generalised to multi-class classification [17]. As SNNs are metric learners that develop a new representation of the original data, in a classification setting they require a non-parametric learner (such as k-NN) to perform the explicit classification.

The Triplet Network (TN) is a deep metric learner which learns from three examples concurrently (an anchor, positive and negative example respectively), giving the network its namesake [16]. Throughout training, the network learns to minimise the distance between an anchor and its associated positive example while maximising the distance between an anchor and its associated negative example [16]. Their capability on this task has translated into strong performance in areas such as face recognition/re-identification [18, 81, 82] or image-based search [78]. Unlike SNNs, TNs were designed to be supported by a similarity-based return component [16] and cannot perform classification tasks on their own. This means that TNs are very capable of establishing an effective basis for similarity-based return on multi-class problems [83].

Matching Networks (MNs) [77] are unique in that they can be used flexibly as either a classifier or a DML. Originally designed for few-shot classification problems, MNs learn to match a query case to members of a support set which contains both matching and non-matching cases. They therefore learn a representation which is optimised for similarity comparisons (much like the two DMLs above). MNs can be trivially adapted to

remove class reliance learning by considering altering the matching criterion of the support set. More recently, MN was exploited successfully to achieve personalised HAR[84] and Open-ended HAR [85] where they successfully utilise support set to enforce personal traits of human activities.

One of the contributions of this thesis aims to improve the training of DML architectures using inspiration from traditional CBR research. With that in mind, we discuss specifics of network formation and mathematical justification of training SNNs and TNs in Chapter 3. Although the ideas behind the contributions could be applied with some adaptation to MNs, we do not explore this in detail. Instead, we opted to examine the impact the contributions had on a real-world case study in our telecommunications use case.

2.2.2 Metric-Based Loss Functions

Metric-based loss functions aim to develop a representation for the input data such that similarity computations are optimised [15]. This is unlike typical deep learning loss functions, where the goal of learning is to develop a representation optimised for classification (i.e. network output is a probability distribution indicating class membership for a single example) [86]. As a result, metric-based loss functions have a tendency to rely on direct comparisons between examples [15, 16], though some have managed to maintain the emphasis on classification and integrate this with a comparison-based architecture [77].

The earliest metric-based loss function, contrastive loss [15] is credited with kick-starting the field of deep metric learning research. Informed by a pairwise comparison of two input examples, contrastive loss effectively forces a network to learn a binary classification. However, instead of classes, pair members are classified on the basis of whether they meet the matching criteria or not. This requires all input pairs into a network to receive a pair label, indicating whether they are matching or not. Matching pairs will generate loss if their constituent members are too dissimilar, while non-matching pairs will generate loss if they are too similar (as decided by thresholding using a margin value). Though simple, the loss was effective and ensured that the pair-based SNN architecture was competitive in many complex tasks [15, 17].

More recent innovations on pair-dependant metric-based loss functions saw the birth of pairwise loss [87]. The original contrastive loss was not robust to intra-class (or intra-cluster) variance of examples, as it forced all 'similar' examples to occupy the exact same area of the space. Pairwise loss added a boundary parameter to allow to improve differentiation between examples within the same class and better reflect real-world problems. Inspired by the SNNs capability in one-shot learning, a pairwise

loss optimised for transfer learning has since been introduced [88]. Results on both one-shot learning and transfer learning tasks are indicative of strong performance and more possible work in this domain.

While pair-dependant metric-based loss functions are powerful, they present several issues. Chief among these is that the objective of most of these functions is to learn a binary classification of whether a pair is matching or not [15, 87, 88]. It would be desirable for the objective of the loss function to be optimisation of the similarity space itself. Additionally, pairwise comparisons are limited in their granularity. Comparing a greater number of examples would better inform output of the loss function using more knowledge of the latent space. It was from these insights that triplet loss was born [16].

Triplet loss considers three examples - an anchor, a positive example and a negative example - simultaneously. The goal of training with the triplet loss function is to ensure that the anchor-positive example pair is more similar than the anchor-negative pair plus a threshold margin [16]. This resolves many of the problems with pair-dependent loss functions. The greater number of examples utilises more knowledge from the feature space to speed up training, and the objective of the function is more aligned with developing a latent space optimised for similarity. Additionally, because the loss function only requires the anchor-positive pair to be more similar than the anchor-negative pair and not identical, intra-class variance is maintained [87].

There have been several recent advancements in triplet loss. Despite the original function's improved ability to handle intra-class variance, the authors in [89] propose this is still not satisfactory in domains such as person re-identification, where it is difficult to cluster similar examples due to dissimilarities between images (such as changing backgrounds or angles). They propose an adaptation much like pairwise loss, utilising a boundary to allow even greater distribution within a single class. Though their results seem to demonstrate this is an improvement over original triplet loss, intuition would suggest clusters are much more distributed over the space. This likely inhibits its usefulness in problems outwith its specific domain. Other approaches have examined the nature of the comparison between triplet members, with authors suggesting angular comparisons [90], or integrating sample selection into the loss function itself [91, 92, 93].

Interestingly, work in [94] shows a disturbing pattern for many of the recent innovations to metric-based loss functions. In this work, the authors cite a lack of experimental consistency as a factor for seeming enormous improvements in DML research over the last several years. To evidence this, they present a uniform comparison of 13 different loss functions from across the literature. Their findings indicate minimal

improvements beyond the original contrastive and triplet losses respectively. Overall their work demonstrates a lack of consistency across research in this field, and seemingly refutes much of the claims about progress. With these results in mind, in this thesis we use contrastive and triplet loss for our experiments with SNN and TN respectively. Also, to ensure reproducibility of our findings and inline with the recommendations made towards improving the consistency of evaluating metric learning algorithms in [94], we use five cross-fold validation in all of our experiments.

2.2.3 DML Optimisation with Sampling

As DMLs are trained upon multiple examples simultaneously, there is an additional dimension to training which is not present in conventional deep learning architectures - sample selection. Though convergence of DMLs can be achieved through creation of random pairs/triplets, recent work has demonstrated that a training strategy which optimises triplet creation can improve training efficiency [18, 78]. It is important to consider which samples are most suitable to be input together in order to maximise training efficiency, as examples which produce no return in the loss function will not progress network training towards convergence. As the networks approach optima, this becomes increasingly problematic, as it is more likely that randomly selected samples will not meet loss function conditions.

Sample selection for DMLs can be broadly split into two related but distinct steps - batch selection and sample mining. However, inspired by work the area of Curriculum Learning [95], we also propose that sample ordering should be considered as an aspect of sample selection. This is discussed further in Chapter 4.

Sample Mining

Employing informed selection for training data is increasingly gaining attention for deep learning architectures. Both optimising for batch size and the order in which training examples are processed have shown to be effective for achieving performance improvements [95, 96]. Typically network loss is exploited to create ranking heuristics with significant speed-up gains observed when processing harder examples first. Training on an increasing ratio of 'hard' samples can be seen as adopting an 'exploitation' strategy where focus is maintained on known 'hard' problems. It is interesting to note that meta-learning strategies, such as boosting, do precisely this with weak learners; whereby model learning is focused on examples that were incorrectly solved previously [97]. This technique has demonstrated considerable success in areas such as transfer learning [98] and object detection [99]. However such a strategy alone in the context of informed sample selection can be detrimental, if 'exploration' of the space

of possible problems is ignored [96]. We study how both exploration and exploitation strategies can be utilised for informed pair selection. Specifically we consider budgeted learning scenarios associated with learning an embedding function [100], where it has been shown that picking more suitable examples will return greater results in circumstances where labeled data is limited [101].

Paired examples in relation to triplet networks (TNs) [16] help learn useful feature embeddings (representations) by distance comparisons [18, 78]. Like SNNs, the goal of training is to develop an embedding function which minimises distance between the positive examples and the query examples, while maximising the distance between the negative example and the query. Unlike with SNNs, TNs form a triplet instance from a negative and positive pair given a query. Heuristics that are static (neither exploratory or exploitative) based on initial similarity (relevance) alone were found to perform poorly [78]. Using heuristics that continually update to reflect the triplets the network is likely to find difficult in the next iteration, such as exploiting according to the loss value, was found to give superior performance [18]. However, loss information is not available from the start of training so the network must complete an initial 'dry run' to retrieve this information. In our work we consider how heuristics that utilise similarity knowledge calculated from the most recent network embedding can contribute towards formulating a more dynamic ranking heuristic for training examples.

Other research has shown that sampling is incredibly important in the field of deep metric learning [102]. As the number of triplet candidates increases near-cubically with the number of examples, it is not feasible to train on all possible combinations. Furthermore, in many situations not every triplet is valuable. Therefore there is much work targeting the optimisation of training triplet networks through sample selection via triplet mining [18, 78, 81]. In [78], the authors use a deep similarity ranking to guide triplet formation for use in learning image similarity. Using a calculated image relevance, they suggest that a relevant but non-matching image should be selected as the negative example and a non-relevant but matching example as the positive example for an anchor image. The authors of [18] expanded upon this idea and removed the concept of relying on an external ranking to decide relevant triplets. They selected pairs by calculating their loss value to pre-emptively identify their input to the network. They observed that triplets which produced the maximum amount of loss (the 'hardest' triplets) actually caused training to destabilise and network convergence took longer. Instead, focusing on triplets where the distance between the anchor and the negative was greater than the anchor and the positive, but less than the margin (the 'semi-hard' triplets) were more effective for training.

Batch Selection

The sample mining approaches discussed above operate on a subset of the full training set, which is selected randomly from the full distribution, to make computations cheaper. This is consistent with other examples in the literature [81, 82], where authors apply an active learning approach after a subset of the training set has been extracted. We argue that triplet selection actually begins with the selection of that subset, rather than the mining within. Although mining is an important concern, the best pair or triplet cannot be selected if one of the components is not within that initial subset. Identifying this subset is an important aspect of training a triplet network in its own right.

We refer to this as 'batch selection of input candidates', or batch selection from hereon. We use this term to describe the phase in a training strategy which extracts a batch of candidate examples from the full training set which can then have sample mining approaches applied to select suitable pairs/triplets. During implementation, this batch could be seen as a replacement for the standard minibatch associated with stochastic gradient descent in conventional deep learning architectures. Like the minibatch applied to non-DML architectures, network weights could be updated after the examples from the batch have been passed through the network. However, unlike typical deep learning architectures which only receive a single instance as input at a time, after batching DMLs require the examples to be grouped into pairs, triplets or subsets before they can be input to the network. Therefore, batching is not the final step before examples are input to the network.

We are aware of only limited work which directly targets batch selection of instances before input to the network [103]. In this work, the authors process images of faces through a classifier to improve representations, before passing those learned representations to a k-means algorithm to perform clustering for initial batch selection. However, this approach presents several disadvantages. Most importantly, it is a learned clustering method with corresponding overhead and dependence upon access to labelled data. This is problematic, as triplet networks perform best in situations where labelled data is scarce or totally unavailable. The process is also not iterative as clustering is performed on the output of the classification model at the start of training and remains static. Ideally, locality-based minibatching should exploit the latest network output to ensure its batches are relevant to the network at that point in training.

Sample Ordering

Recent work in training of deep learning architectures has also identified that the order in which examples are input to the network can have a substantial impact on training [95, 104].

Curriculum Learning (CL) is the concept of introducing examples to a network in a meaningful order, most often by difficulty from 'easy' to 'hard'. The idea is that by ranking so that the network is initially exposed to simpler examples and then gradually introduced to more complex examples, the network will converge faster [95]. Though a simple concept, CL has demonstrated excellent generalisability, showing success in areas such as motif finding, noun phrase conference [104] and multi-task learning [105]. Research has also shown that self-paced learning, where the ordering of examples is based on feedback from the network itself (dynamic), rather than a (static) curriculum set by a teacher [104], results in model improvements. In this way, the order in which examples are presented to the network is continuously updated, such that the curriculum presented at the start and that presented at the end of training may be vastly different.

Promising results in this field are already leading to the development of novel sample mining approaches which incorporate curriculum learning into their strategy. For example, in [106], the author's propose a difficulty-based curriculum for the ordering of pairs to be input to an SNN on the basis of whether pair member distance is close to a pre-defined margin value. The intuition is regardless whether a pair is positive or negative, the most complex pairs to differentiate will have a distance between pair members which is close to the threshold. The results obtained indicate that this provides a positive impact on training, but a more thorough evaluation would be required. In [107] the authors are also inspired by curriculum learning to improve training of a neural network on the task of human attribute analysis. As the datasets for this problem are highly imbalanced, they propose a two-level curriculum. The first level uses curriculum learning to order sample mined examples on the basis of data imbalance (i.e. classes with many examples are put in to the network first, while classes with fewer examples are input later) at each epoch. The second level gradually alters the focus of the loss function from representation learning (using an adaptation of triplet loss) towards classification over the course of training. The authors argue this makes the network better placed to learn robust similarity-based representations at the beginning of training, while ensuring high classification accuracy at the end of training, which their results support.

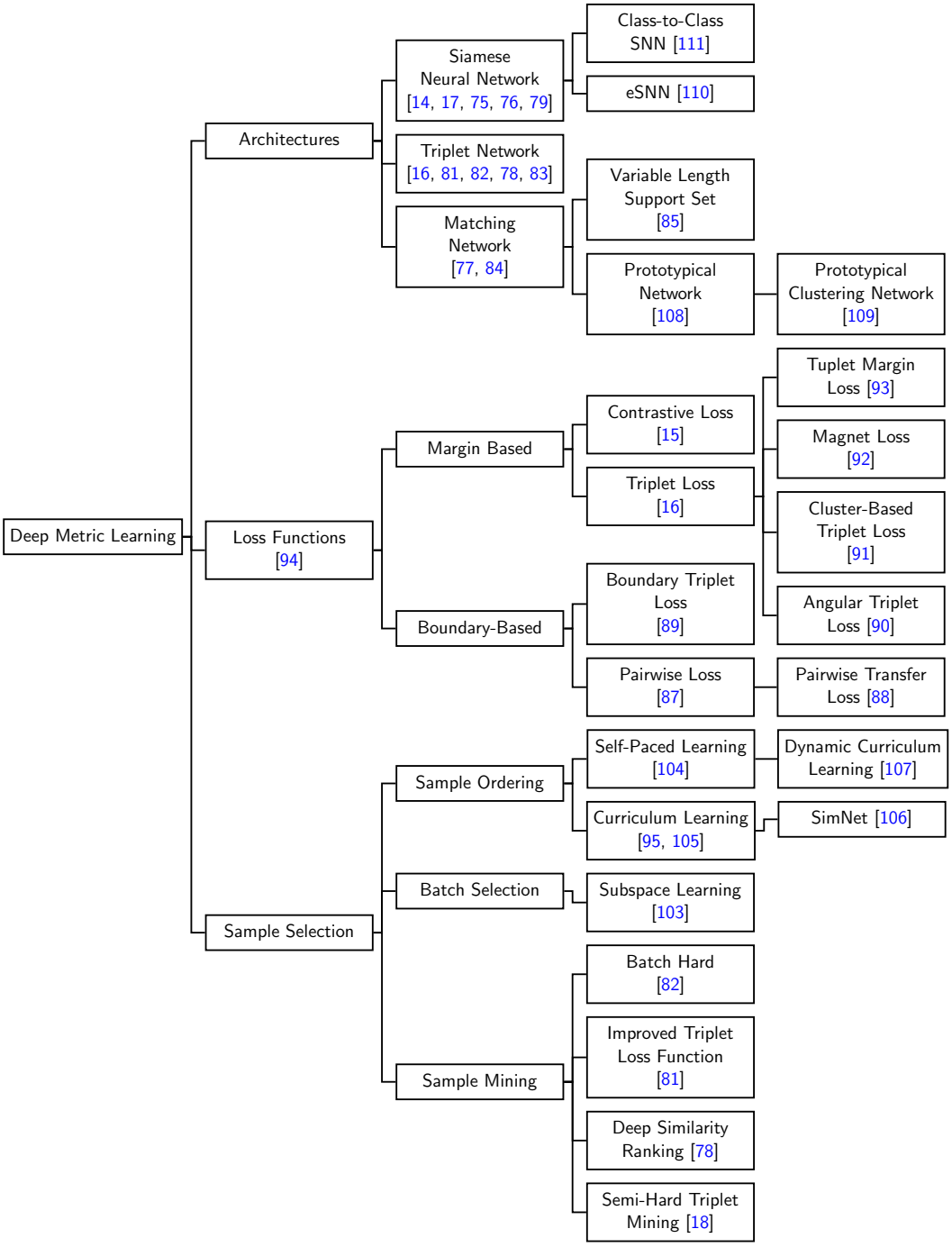


Figure 2.1: Taxonomy of DML Literature

2.2.4 DMLs and CBR

The synergy between deep metric learners and case-based reasoning is demonstrated by growing literature on the subject [41, 84, 110, 111]. In particular, we can observe growing work which uses DMLs as a means to fulfil the roles of both the vocabulary and similarity knowledge containers. For example, in [110] the authors use SNNs to learn a novel similarity function between data, giving rise to an augmented version which they describe as an extended Siamese Neural Network (eSNN). What is interesting in this work is that the network learns the vocabulary (by learning a representation for the input data) and similarity (by optimising that representation for retrieval) containers simultaneously. In a separate work, the authors in [84] use a matching network to achieve a similar outcome. Here they draw the comparison of matching network to a parametric form of k-nearest neighbour and explicitly highlight how their approach replaces conventional vector-based similarity metrics. In yet another example, the authors alter the training of an SNN to instead learn the similarities and differences between classes [111]. This is interesting because the trade off means that the so-called Class-to-Class SNNs contribute less to the similarity container (as the learned representation is not as well optimised for retrieval), but in its place contributes slightly more to the explanatory capability of the network.

We can therefore observe a very clear connection between the knowledge containers in CBR research and the role of DMLs. DMLs are suitable to fulfill the role of components within separate knowledge containers in a single algorithm. In particular, we highlight the vocabulary and similarity containers, where they are capable of covering the tasks usually performed by feature engineering or local similarity measuring algorithms. In addition, much like there is a well acknowledged trade-off between knowledge containers [37], so too can we see this trade-off within alterations to DMLs [111]. The evidence points to a clear synergy between both the CBR and DML methodologies which can be most easily described using knowledge containers. Although we are not the first to highlight relationship between CBR and DML, to our knowledge we are the first to structure this comparison using knowledge container research from CBR.

Another interesting avenue of research is the growing work that looks to take lessons from traditional machine learning methods and apply them to improve the training of deep metric learners. Prototypical Networks [108] are an adaptation of the matching network which incorporates lessons from prototype cases in CBR (see section 2.1.2). The model creates a prototype (by averaging over similar elements in the support set) for each class in the support set, then behaves as a one-shot learning matching network model. Within the experiments presented by [108], the prototypical network outperformed the original matching network through simple adoption of a relevant technique

from the CBR field. An updated of the network, Prototypical Clustering Networks, was proposed to be robust to intra-class variance by using multiple prototypes for each class within the subset [109]. In their results the authors note that increasing the number of clusters considered in the support set offers a dramatic improvement relative to the original prototypical network. This mirrors the description of LSH (see section 2.1.2), where we discussed the trade off considered when deciding the number of projections for LSH.

Similarly, there has been some previous work in using clustering techniques to inform network training by altering the triplet loss function [91, 92]. However, these methods typically require *a priori* knowledge [91], or offer reduced flexibility to incorporate other training methods (such as hard sample mining) because the clustering mechanism is tightly coupled to loss calculation [92]. For example, in [91], the authors build a hierarchical class-level tree to define inter class similarity, which can be consumed by a hierarchical variation of triplet loss. This requires a pre-processing step to compute the tree, meaning *a priori* knowledge of the data or a pre-trained network is required. It would be desirable to have a solution with no previous knowledge requirements, and which could function as part of an ecosystem of training methods. This is indicative that the links between CBR and DMLs extend beyond simple reuse of algorithms and strategies, and hints at deeper research avenues within this area.

2.3 Explainability

Considering the relationship that exists between CBR and DMLs is centered around the notion of similarity, and evidence supporting CBR as an explainable methodology [7, 12], it seems intuitive that DMLs should also be more explainable with similar methods. The term 'explainability' refers to the level at which a machine learning algorithm is comprehensible to its intended user base [21]. Though explainability of systems has been a subject of much works since relatively early in machine learning research [9, 7, 112] recent developments have resulted in explosive popularity of this field [21, 22, 113, 114, 115]. This is largely due to growing social and ethical responsibilities being faced by organisations to ensure that decisions made by their intelligent systems are explainable. These responsibilities are supported by European legislation which dictates 'an individual's right to an explanation' and ensures that organisations are held accountable for the decisions made by these systems [116].

Central to the idea of explainability is the concept of mental modeling. A mental model can be summarised as a user's conceptual understanding of an intelligent system; in short, it is everything they (a) know and (b) believe about a system [114]. An explanation need is created when something happens within the system that breaks a user's

mental model or forces it to adapt. The explanation need is therefore a formalisation of the gap that has been identified in a user's mental model [20], while an explanation is then the artefact which must be created in order to fill that gap [21]. To give an example, consider the average person's mental model of a car. The vast majority of people will understand that a car allows them to move quickly from place to place and that this is enabled by the burning of petrol so that the car can move forward. When something breaks this status-quo (i.e. there is petrol in the car but it does not move forward) a gap in the mental model is usually highlighted. Despite knowing that a car requires petrol to move, the average person may not know the inner mechanisms of the car and so cannot comprehend why the car does not work. Their explanation need becomes"why does my car not drive even though there is petrol in it?". Thus they request an explanation from someone with a more complete mental model (such as a mechanic).

A mental model can be influenced in various different ways. One way in which our beliefs about the world are altered is through observation of truths that do not adhere to those beliefs, and this is the governing principle of transparent algorithms [12, 21, 115]. If we can clearly see and comprehend the decision-making process of an intelligent model then this is usually sufficient to convince us of its effectiveness and clarify why an outcome was reached. This school of thought has lead to classification of algorithms on the basis of their transparency: black-box algorithms, where the decision-making is fully opaque; grey-box algorithms, where the process is clear enough that additional information can be used to interpret missing information; and white-box algorithms, where the decision-making process and all needed information to support that process is clearly visible and understandable [115, 117]. For a comparison of terminology, see Table 2.1 to clarify the vocabulary we will use in the discussions of explainability throughout this thesis.

Often the transparency of a system is used to complement other methods to offer more well-rounded explanation. For example, the authors in [12] propose five goals that an explainable system should be able to achieve: it should be (1) transparent; (2) able to justify its decisions; (3) support the user in their ability to understand and conceptualise necessary features; and (4) ensure that the approach adopted by the system is relevant to the problem. These aspects should (5) support the user in their ability to learn both about the system and the problem domain [12]. An interesting anecdote about this perspective is that it would suggest the complimentary methods that are required to ensure explainability of a system are not totally unlike the knowledge containers in CBR. We explore this in greater detail in Chapter 6.

Terminology	Reference	Definition
Explanation	[21, 115]	A generated artefact designed to improve a user's understanding of a system's decision-making or output.
Explanation Need	[20]	A formalisation of the user's comprehension gap which creates the requirement for an explanation.
Mental Model	[114]	A formalisation of the user's mental image of an algorithm and their understanding of how it works.
White-Box Algorithm	[115, 117]	A fully transparent algorithm whose decision-making process is obvious and clearly understandable.
Grey-Box Algorithm	[117]	An algorithm whose decision-making process is visible, but this does not lead to full understanding of the model. Generally a property of interpretable models, as opposed to transparent models.
Black-Box Algorithm	[21, 115, 117]	An algorithm whose decision-making process is inaccessible or opaque.
Transparency	[12, 21, 115]	The property of an algorithm whose decision-making process is clearly visible and accessible.
Interpretability	[21, 115]	The capacity to which an algorithm's decision-making can be worked out (or interpreted) based on available information.
Justification	[7, 12, 115]	The capacity to which a system can justify its decision-making.
Conceptualisation	[12]	The extent to which a user can grasp the underlying concepts behind the features which make up a case.
Relevance	[12]	The applicability of a provided explanation (i.e. how aligned is an explanation with the explanation need).
Learning	[12]	The extent to which a system supports a user to improve their own individual knowledge and enables learning.

Table 2.1: Explanation Terminology

2.3.1 User-Focused Explanation

At an individual level, a user's need for an explanation is often characterised by a discrepancy between the formal objectives of the learned model and its practical application [21]. In practice this can occur when the expectations of a system's user group do not match up with one another, or their mental model is not aligned with the system's output. A common cause for this is when individuals' capabilities and expertise are not considered [118]. In [119] for example, the authors highlight that understanding the relationship between the needs of a technical expert and the needs of a non-technical user is of fundamental importance for the success of a deployed industrial application. It is often the misalignment of objectives between these two parties, or the inability to effectively transfer information between them, that leads to costly errors. In domains such as telecommunications engineering where the technical experts heavily rely upon their non-technical counterparts for administrative and logistical purposes, it is vital that a clear and understandable flow of information is maintained between the two groups.

In practice, we rarely intend for a system to be used by a single user in isolation. Recent work has demonstrated that it is important to be aware of the multiple stakeholders who are likely to require an explanation of an intelligent system [22, 119, 114, 120]. Each user of an intelligent application approaches the system with an individual context which defines their need for an explanation [120]. Some researchers argue that part of the responsibility of an explainable intelligent system is to enable fairness of its use throughout its user base [113]. It follows that no single explanation method is therefore suitable to answer every possible need from every possible stakeholder. This intuition inspired us to develop a catalogue of explanation methods, allowing users to utilise the most suitable combination of explanation mechanisms to meet their individual needs.

Developing a suite of explainability methods aimed to satisfy multiple user groups presents a number of challenges. Foremost among these is identification of explanation methods which would be suitable for the different user groups we may encounter. There have been several works in the literature which have attempted to group stakeholders by explanation need. In [22] the authors suggest that users of an intelligent application can be divided into three groups (novice users, domain experts and AI experts), each with distinct explanation needs. While AI experts are usually satisfied by global explanations describing how the learned model operates, novice users and experts within individual domains are more likely to require local explanations contextualised by specific input-output examples. Despite this similarity in need, there remains a wide gap between these latter groups in regards to their contextual domain knowledge. This divide is noticeable within our context when comparing field engineers with desk-based agents.

Similarly, there is existing work which suggests that personalizing the explanation to meet the needs of individuals within larger groups will result in improved user experience [121, 122]. In [121] the authors present evidence that an individual's personality significantly correlates with the number and type of explanation which improve their receptiveness to the explanations of recommender model decisions. Their results also highlighted that the format through which an explanation was provided (visualisation or text-based) impacts a user's receptiveness to the explanation. While they found that users universally preferred text-based explanation methods over visualisations regardless of individual personality, they did not examine a broader scope of explanation formats. Intuition would suggest that a study which tested a broader range of explanation formats (incorporating for example, statistical explanations) or examining different use cases (where the original authors focused on movie recommendation using a recommender system) may result in findings where personalization of explanation format is also scenario dependant.

Interestingly, in [122] the authors find that while personalization of explanations may cause increased user satisfaction, it can actually be detrimental to explanation effectiveness. They judge effectiveness of an explanation on the criteria that it supports users to make a knowledgeable decision. They propose an experiment to assess effectiveness of explanation based on whether a user is more or less willing to accept the most appropriate recommended item (as identified by the recommender model being explained) after being exposed to some explanations. Their results demonstrated that non-personalized explanations supported users to do this more often, while personalized explanations lead to better user satisfaction with the explanation itself. This suggests that an applied system which implemented personalization of explanations would have to ensure that task performance was not negatively effected.

For our use case of supporting telecommunication engineers, we find it appropriate to start by building a understanding of the explanation needs of user groups, specifically the expert engineers and non-expert desk-based agents who support them. In doing so, we can learn the explicit differences in explanation need between these two groups. This could present a strong foundation for the framework whereby the nuances and needs of individual users could be met with finer-grained control of explanation through personalization.

2.3.2 Evaluating Explanations

A further challenge is the evaluation of explanations intended for multiple stakeholders. How to evaluate explanations, the need-for and usefulness-of which are fundamentally

subjective to an individual user's context [114, 123], is generally considered a user-centric area of machine learning research. For that reason, much of the work in this field has aimed to formalise qualitative measures of human understanding into quantitative metrics of system performance [12, 22, 117, 124].

For example, in [117], the authors suggest the explanation quality of a system can be appraised through a mix of subjective and objective measurements. These measurements are: user satisfaction (e.g. the clarity and utility of the explanation); mental modeling (e.g. the ability to understand individual decisions and identify strengths and weaknesses of the machine learning model); task performance (e.g. whether user ability to complete the task is improved by using the system); trust assessment (e.g. whether the system is trustable); and correctability (e.g. the user can rectify incorrect decisions). A mix of subjective and objective metrics allows developers to measure user opinion of explanation quality (through user satisfaction, mental model and trust assessment), as well as determine whether the explanations actually approve on practice in an applied environment (via task performance and correctability).

Though attempts to empirically evaluate explanation methods without user feedback are growing more common, these metrics typically rely on justifications that explanation has improved algorithmic performance or comparisons against model-agnostic and/or interpretable model baselines [22, 125, 126]. We are less aware of methods which attempt to assess explanation quality by exploiting similarity information. In this respect, we suggest our work in modelling the relationship between a query, its neighbour set in a latent space and the retrieved explanation, is relatively unique.

2.3.3 CBR as an Explainable Methodology

Case-based Reasoning is often cited as a grey-box algorithm, where transparency and justification of system outcomes are borne from the fact decision-making is based on comparisons to previous cases [7, 127]. This is a stark comparison to black-box algorithms like deep learners, where attempting to clarify the process which has lead to a particular decision is extremely difficult. This is in no small part due to the manner in which it reflects a human's mechanism for learning from experience [9, 11], making it a recognisable method of explanation for many people.

This statement is challenged by [12], who states that decision-making of a CBR system can be significantly less transparent if the presented solution is not clearly aligned with the query case, which is arguably the most common scenario in which a mental model is disrupted. In such circumstances, the similarity calculations which had lead to a particular recommendation can be displayed, presenting an opportunity for justification. However, if the user's conceptualisation of the model is inaccurate, then this will

interfere with the transparency of the decision and force the user to re-evaluate their mental model. Given the expectation that scenarios will occur where the user is asked to collate information to interpret decision-making, it could therefore be argued that CBR is closer to an interpretable model than a fully transparent algorithm.

Even this relatively straightforward example highlights the inter-dependency of different explanatory components and demonstrates an interesting contrast to the knowledge containers in CBR. Though not among the four knowledge containers proposed by [13], there is a clear relationship between the containers and explanation. Some works have used the knowledge container model to align a user's explanation need with the knowledge stored in a particular container [7, 12, 19]. For example, the case-base container is often highlighted as a useful tool for explanation, as it is straightforward to demonstrate the outcome of other known scenarios in comparison to the current query [7, 127]. This means it is well placed for justification of decisions, but only poorly equipped to explain underlying concepts behind case features. These questions (which typically target improving the user's ability to conceptualise the ideas behind case composition, such as "what does this feature mean" or "how does case x represent problem y") are directly linked with the the contents of the vocabulary container [12]. However, the vocabulary container may struggle to demonstrate the relevance of features (beyond obvious overlaps) to decision-making at any specific point in time, as this is a task better handled by the similarity container. In turn, justification of what makes these features similar is perhaps better demonstrated by extracting high-level analytics (such as parallel co-ordinate graphs [128]) using knowledge from the case-base.

However, unlike the knowledge containers in CBR, the principles of explanation usually contribute to better explanatory capabilities of a system when they are all fully formed. Since an explanation need is very tied to the certain explanatory methods, others cannot usually be used to circumvent lack of another method - a user who does not understand the concepts of case composition will likely not be satisfied with an explanation formed from justification for example.

From the perspective of explanation, DMLs are perhaps more similar to CBR than any other deep learning architecture. Much like how similar cases can be used as a point of comparison to explain the output of a CBR system, the same could be achieved with DMLs [111]. This is of course domain dependent - if the user does not understand the features of the comparative example as raw input, then this becomes significantly more challenging. This is because the new representation learned by DML will not be understandable to the user, as its an abstract of the original input. This will impact a user's ability to conceptualise what each of the output features mean. Justification of system decisions is still possible using case-by-case similarity comparisons with knowledge from

the case-base, while the similarity knowledge is still pertinent for demonstrating the relevance of two cases to one another. A key difference is the vocabulary container, which is poorly placed to support conceptualisation. Overall it indicates DMLs could fit the CBR methodology's knowledge containers, and that these containers could be useful to model the explanatory capabilities of the systems.

2.4 Conclusion

To summarise the findings from our review of the literature, we have discussed the fundamentals of CBR and the knowledge container model. We have provided a brief introduction to the vocabulary container as a means to conceptualise the process of identifying case structure and features. We highlight specifically approaches for developing representations for text cases, including statistical and learned methods. Furthermore, we have discussed the similarity container to represent the process of case comparison. Overall our examination of the literature has suggested that these knowledge containers are implemented separately, despite the overlap between them. We will explores this further in the next chapter, where we discuss implementation details for several of the algorithms.

DMLs could provide a means to bridge the gap between the vocabulary and similarity knowledge containers. In this chapter we have introduced the concept of deep metric learning, and detailed how DMLs are split into three related components: (a) a neural architecture capable of receiving multiple simultaneous inputs; (b) a metric-based loss function; and crucially (c) a training strategy to optimise DMLs through sampling. Our exploration of this area has identified close links between DMLs and CBR, which we intend to explore further with our contributions in Chapters 4 and 5. In the next chapter we will demonstrate how DMLs leverage the learning of non-linear feature combinations to learn a latent space optimised for similarity computations.

Finally, we have reviewed literature surrounding explanation of machine learning architectures. Our findings indicate the similarity-based methods, including CBR and DMLs, are well pre-disposed towards explanation. This is because they mimic the way in which humans themselves approach problems (i.e. by comparing the current situation to relevant past experiences). Overall we believe this is indicative that similarity also offers a good research avenue to explore evaluating the quality of explanations. We explore this further in Chapter 6.

Chapter 3

Technical Aspects of Machine Learning

In the previous chapter, we presented a review of literature examining similarity for pre-processing, classification and explanation. In this chapter, we will describe the algorithmic implementation details which will be used throughout the thesis. In particular, we will discuss both case-based reasoning and deep learning in detail, with the latter emphasising the technical aspects of deep metric learning algorithms.

3.1 Case-Based Reasoning (CBR)

CBR is a methodology, not a specific technology [9, 11]. It describes a group of techniques which are co-ordinated in their objectives by the knowledge containers. In the following subsections, we will discuss the methods we have adopted for the vocabulary and similarity containers. These are necessary because DML architectures are incapable of learning a representation directly from the raw input text. The adaptation container is out of the scope of this thesis, so we do not explore relevant algorithms here.

3.1.1 Vocabulary Knowledge

Machine learning methods typically struggle to work directly with raw text. Instead, this text must be converted into a numerical vector representation, which can then be used as input to machine learning algorithms. Although there are a number of ways to perform this process, in this thesis we examine learning representations using statistical and learned approaches specifically. In the following subsections we will discuss how representations based on the statistical measures can be practically implemented using

term-frequency / inverse-document-frequency. Then we will examine how word embeddings which leverage semantic relatedness knowledge by training on word co-occurrence can be learned using Word2Vec.

Statistical Approach - tf-idf

As stated in Section 2.1.1, term-frequency / inverse-document-frequency (tf-idf) enables the numerical representation of a document through quantification of the relative uniqueness of terms within that document when compared with the rest of the corpus. The intuition here is that terms which appear frequently within a document are integral to the overall meaning of the document. These terms should help to uniquely identify the document among others. However, terms which frequently appear throughout the corpus are less likely to uniquely identify any individual document. The trade-off between terms which define document meaning and how novel these terms are within the corpus as a whole is captured in the term-frequency and inverse-document-frequency components of the function respectively.

The tf-idf score of a term is calculated using Equations 3.1, 3.2 and 3.3.

$$tf(t,d) = f_{t,d} \tag{3.1}$$

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in D\}|} \tag{3.2}$$

$$tfidf = tf \cdot idf \tag{3.3}$$

Where t is any given term, d is a document within a corpus D (such that $d \in D$) and N is the total number of documents within the corpus.

Firstly, the tf score of each term t in a document d is captured. This is (in its simplest form) a count of the number of times that the term appears in that document. Then the idf score of each term is calculated by taking the logarithmic of the total number of documents N divided by the number of documents which contain the term ($d \in D : t \in D$). The resulting idf score will be large if the term appears in few documents, and small if the term appears in many documents. It is also worth noting that while the tf score must be calculated for each term in each document, the idf score will be the same for a specific term across all documents. Finally, the tf and idf scores are multiplied to give a tf-idf score for that term in regards to that document. The scores for each term within a document are compiled into a vector to create a representation for that document as a whole.

The representations gained through tf-idf have a score per term which is featured in the corpus. Though this can be controlled by considering only the n top scoring or most frequent terms across the corpus, the representations learned by this method are still very sparse. Furthermore, they are generally not robust to large corpora with very varied vocabulary. This method tends to perform better in domains with small and confined vocabularies, such as terminology-focused technical work. This means that tf-idf should be well placed to support representation learning for the textual notes maintained by telecommunications engineers in our use case.

Learned Approach - Word Embeddings

Word embeddings are vectors which store values describing the latent or semantic features of a word in relation to a corpus. Typically these word embeddings are learned by training using knowledge of word co-occurrence in the training data. As a result, the learned word embeddings exist in a feature space which is optimised such that semantically related words are closer together then semantically unrelated words. This allows algebraic functions to be performed on the vectors to return related words - the most famous example of this being: vector('Queen') = vector('King') - vector('Man') + vector('Woman'). Furthermore, the representations learned maintain these properties even when combined to create representations for longer passages of text (such as sentences or documents), while still being comparable because the same dimensionality is maintained. The representation of documents learned using the word embeddings is therefore theoretically optimised for further similarity computation or measurement because the training stage incorporates the principles of semantic relatedness.

In this thesis, we explore word embeddings more deeply through the Word2Vec algorithm (see Section 2.1.1). Word2Vec is comprised of a simple neural network architecture with a single hidden layer. As input, the algorithm receives the entire training corpus converted into one-hot encoding and segmented into 'sliding windows' of an adjustable length of sequential text, such that one window is a single input to the network. Windows are described as sliding because they sequentially consider potentially overlapping portions of a sentence. Each of these windows contains a 'target term', which is the term whose semantic features are being learned during that iteration of the network, as well as the contextual terms before and after the target term. Importantly, because a term is likely to appear multiple times within a document, target terms feature in multiple windows beside a range of context terms, which helps the network to learn an overview of how corpus vocabulary is related. Dividing the sliding window into target terms and context means that there are two distinct training strategies available; Continuous Bag of Words (CBOW), where the goal of training is to learn a hidden layer which can be used to predict the target term, or Skip-Gram,

where the goal is to predict the context words using the target term. After training, it is the hidden state (i.e. the transformation of an input term created by the hidden layer) which is used as the vector representation of the term.

Continuous Bag of Words (CBOW): The goal of training a CBOW model is to develop representations which are useful to predict a target term given knowledge of its context. Given a sequence of training words $(w_1, w_2, ..., w_T)$, the objective of the model can be represented as maximising the log probability of:

$$\frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-c}, ...w_{t-1}, w_{t+1}, ..., w_{t+c}) \tag{3.4}$$

where T is the size of the training corpus, while t is the index for the target term from within that corpus, such that $t \in T$. Each w_t exists within a window of contextual terms, the limits of which are given by $-c$ (for words preceding w_t) and c (for words following w_t) respectively.

To provide a step-by-step breakdown of how the objective function for CBOW models produces loss, consider the following steps:

- We iterate through each term w_t in a corpus T ($\sum_{t=1}^{T}$).
- For each term we calculate the logarithmic probability of that term appearing given our knowledge of contextual terms ($\log p(w_t|w_{t-c}, ...w_{t-1}, w_{t+1}, ..., w_{t+c})$.
- Finally, the probabilities are summed and we divide by T to normalise the output ($\frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-c}, ...w_{t-1}, w_{t+1}, ..., w_{t+c})$).

Skip-Gram: The goal of training a skip-gram model is to develop representations which are useful to predict the contextual terms surrounding the target terms. Given a sequence of training words $(w_1, w_2, ..., w_T)$, the objective of the model can be represented as maximising the log probability of:

$$\frac{1}{T}\sum_{t=1}^{T} \sum_{-c<=j<=c,\ j\neq 0} \log p(w_j|w_t) \tag{3.5}$$

Given that in the skip-gram model we are trying to predict the context of w_t, we can use j to represent the index of the contextual term we are trying to predict within the bounds of $-c$ and c. Note that j cannot equal 0, because this would be the index of the target term w_t.

This function produces loss for training the system in the following manner:

- We iterate through each term w_t in a corpus T ($\sum_{t=1}^{T}$).

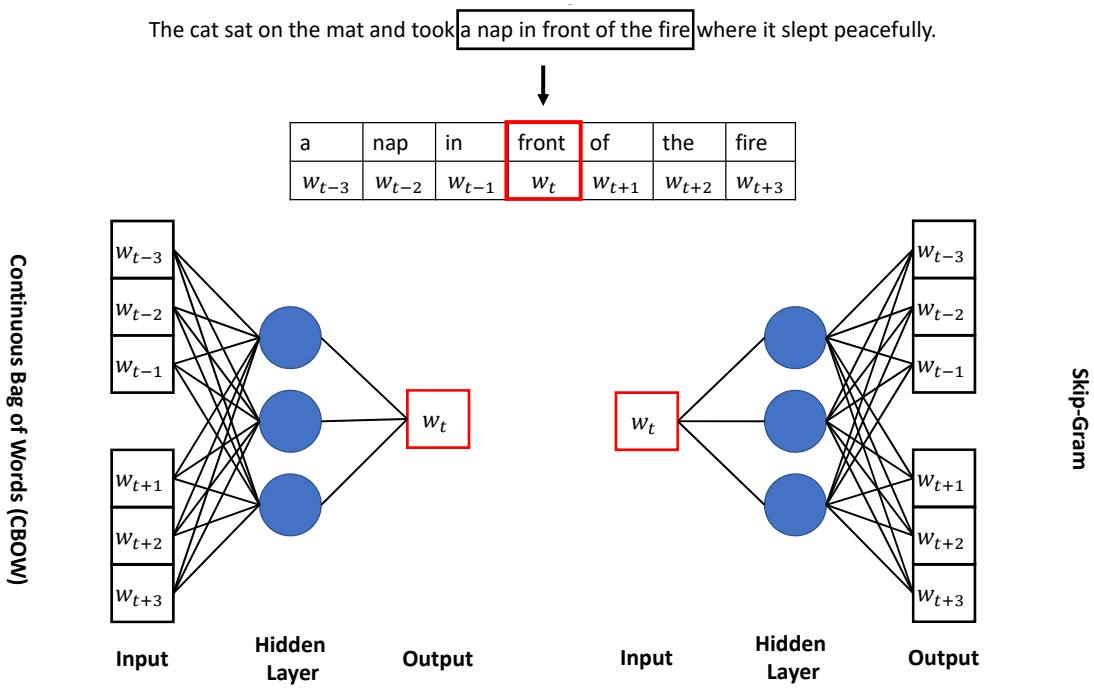


Figure 3.1: CBOW and Skip-Gram Word2Vec Models (adapted from [46])

- For each term, we find the contextual terms within the bounds of $-c$ and c ($\sum_{-c<=j<=c,\ j\neq 0}$).
- For each context term w_j we calculate the logarithmic probability of that term appearing in the context of the target term, w_t ($\log p(w_j|w_t)$).
- Finally, the probabilities of the contextual terms for all target terms are summed and we divide by T to normalise the output ($\frac{1}{T}\sum_{t=1}^{T}\sum_{-c<=j<=c,\ j\neq 0}\log p(w_j|w_t)$).

3.1.2 Similarity Knowledge

Retrieval within a case-base is fundamentally reliant on having knowledge of what makes cases similar. As we have noted in our literature review (See Section 3.1.2), this means that similarity knowledge is a central underpinning of the CBR system. As we are focusing on learning representations for text, we turn our attention to vector-based similarity functions where no hand-crafted feature knowledge is required. In this section we provide an introduction to vector-based similarity functions, and explain how they support the k-nearest neighbour algorithm to classify a query by comparing it to a neighbour set of related cases. We also detail LSH, an approximate-nearest neighbour algorithm which reduces the complexity of neighbour comparisons with the drawback of slightly impacting accuracy.

Vector-Based Similarity Functions

Vector-based similarity functions refer to a family of methods which are designed to measure the similarity between numerical vectors. These functions are typically based upon mathematical proofs. Vector-based similarity functions do not rely on domain-specific knowledge models, and are therefore equally applicable to any numerical vector. Vector-based similarity functions assume that all case features have equal weighting, and that cases are consistently ordered so that features maintain the same index in the feature vector. As a result, these functions are often applied to learned case representations where a knowledge model is unavailable or prohibitively expensive to produce (such as that of our use case). However, they can equally be applied as a local similarity function contributing towards a global similarity aggregate. This is typically done in domains where a case is formed of structured (such as feature-value pairs) and unstructured (free-text, images, etc) data.

In this thesis we consider two vector-based similarity functions: euclidean distance and cosine similarity.

Euclidean Distance: is a function to measure the straight-line distance between two points in a multi-dimensional space. This can be represented as:

$$d(a, b) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2} \tag{3.6}$$

Where the function $d()$ represents Euclidean distance, and a and b represent two vectors of matching length N defining co-ordinates in a multi-dimensional space. The function iterates over each vectors to find the difference between the values at each index i. The differences are squared (to guarantee a positive value), then summed. The square root of the total is then taken, to transition back from squared distance into a normal distance metric. In this manner, euclidean distance is highly dependant on the magnitude of each of the compared vectors.

The output of the function is a distance metric - a value which will be high if the vectors are different, and low if the vectors are similar. Additionally, it will be unbounded; identical vectors will have a distance of 0, while very different vectors can have a potentially infinite distance between them. For similarity calculations, it would be desirable if relationship between vectors could be represented as a bounded value. This can be achieved by $sim = \frac{1}{1+d(a,b)}$. The output of this operation will be a value between 0 and 1, where 1 is representative of identical vectors. It is worth highlighting that dissimilar vectors will only tend towards 0, and never reach it, due to the nature of the conversion.

Cosine Similarity: is a function to measure the similarity of two vectors based on their relative orientation. Unlike Euclidean distance, cosine similarity anticipates that the magnitude of the compared vectors is dissimilar (though the dimensionality should still be equal). Therefore, capturing a straight-line distance is an unfair comparison. Instead, measuring the angle of the orientation difference between vectors is more accurate as a measure of similarity. This can be captured by:

$$cos\ sim(a, b) = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{3.7}$$

Applying the cosine rule, we can calculate the angle between two vectors by taking the quotient of their dot product divided by the product of their lengths. To find the dot product of vectors a and b we calculate the sum of the products of the corresponding values at each index, giving us $\sum_{i=1}^{N} a_i b_i$. Additionally, the length of a vector is defined by the square root of the sum of its squared co-ordinates. We can therefore calculate the product of the two vectors lengths by applying $\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}$. The output of this function will give a value bounded between -1 (representing that the vectors are exactly opposite) and 1 (representing that the vectors are identical). However, the output of the function is bounded to between 0 and 1 in situations where the features of the compared vectors are all positive. Typically, text representations exist within the positive space because the learned features are non-negative. For example, in representations learned by tf-idf cannot be negative, as both term frequency and inverse document frequency rely on the presence of the term at least once in the corpus.

In this manner, we can calculate the similarity of two vectors in a multi-dimensional space on the basis of either the straight-line distance between them, or the angle between their orientation. This demonstrates how we can calculate the similarity between cases whose features are represented by a vector of values. Next we will examine how the similarity values can be used to rank candidate cases for return given a query case.

k-Nearest Neighbour

k-Nearest Neighbour (kNN) is an algorithm designed to classify a query case by considering the class labels of similar cases in the feature space. The intuition is that similar cases will have a similar distribution of features, and are therefore likely to have the same class label. In this manner kNN can be viewed as a microcosm of the mentality that governs CBR as a methodology; that similar problems have similar solutions.

As an instance-based non-parametric learner, the kNN algorithm is not 'trained' through the input of labelled data. Instead, the algorithm requires a dataset of labelled cases to which it can refer to continuously. The algorithm classifies a query

based upon a (potentially weighted) vote of the k most similar examples from this set of labelled data. The value k is an integer to threshold the number of neighbours to consider during the voting process. The similarity between cases is calculated using a similarity function, such as the Euclidean distance or cosine similarity functions which are described above. Weighted variations of kNN ensure that the most similar cases will have more weight during voting. The overall process of kNN is demonstrated in Algorithm 1.

Algorithm 1: k-Nearest Neighbour Algorithm

kNN: $kNN(q, \mathcal{X}, k)$
for x *in* $\mathcal{X}$ **do**
  $S := S.append(sim(q, x))$
end
$\mathcal{X} := \mathcal{X}.sort(X, S, <)$
for $i = 1 \dots k$ **do**
  $NN := NN.append(y(x_i))$
end
$q_y := vote(NN)$
return q_y

In Algorithm 1, q is a query case, $\mathcal{X}$ is a set of labelled cases whose labels can be accessed with the function $y()$ and k is an integer detailing the number of neighbour votes to consider. Firstly, the algorithm calculates the similarity between q and every example $x \in \mathcal{X}$. This is used to sort $\mathcal{X}$ in descending order of similarity to the query. Finally, the algorithm extracts the labels from k most similar cases from the sorted $\mathcal{X}$ to create the nearest neighbour set NN. The $vote()$ function then performs a count of the number of times a label appears in order to produce a label q_y for the query. Conventionally, this vote is a simple majority weighted vote, where the class label is decided by the class with the most representatives in the neighbour set. This can also be adapted to similarity-weighted voting, where more similar neighbours have more input to the classification decision.

Although effective in any domain where the problem can be represented as a vector representation, due to the need to iterate over the training set for every query, kNN can quickly become computationally expensive. The complexity of kNN can be represented as $O(ndk)$, where n represents the number of examples in the training set, d represents the dimensionality of the vectors being compared and k is the number of neighbours considered for classification. In real-world settings, where it is not unusual for large quantities of complex data to be generated daily, kNN can be prohibitively expensive. For this reason, there has been much research into reducing the expensiveness of the algorithm by approximate-Nearest Neighbour algorithms. In the next section we explore

one such algorithm, Locality Sensitive Hashing (LSH) in greater detail.

Locality-Sensitive Hashing (LSH)

LSH defines a family of algorithms which use locality-sensitive hashing functions to cluster information These hashing functions are described as locality-sensitive because there is high probability that similar instances share the same function, but low probability that dissimilar instances share that function:

$$P\big(h(x_i) == h(x_j)\big) \begin{cases} \text{High, if } D_W(x_i, x_j) \text{ is Low} \\ \text{Low, if } D_W(x_i, x_j) \text{ is High} \end{cases} \tag{3.8}$$

In Equation 3.8, $P()$ is a probability function, $h()$ is a hashing function and x_i and x_j are arbitrary examples from within a dataset $\mathcal{X}$. By hashing the space in this manner the complexity of identifying an example's locality becomes sub-linear.

To achieve this in its simplest form, LSH uses random projections to hash the feature space. This method divides the feature space into separate 'buckets' by partitioning the space using a configurable number of random divisions called 'projections'. Every example in the dataset is indexed into a bucket and empty buckets are discarded. To formalise this process, let us consider a random projection, v, with the same dimensionality as x. Since there is more than one projection into the space, v belongs to an ordered set V. To index each example in a dataset, $x \in \mathcal{X}$, we identify the relevant bucket by calculating a hash key, H, formed from a series of binary values, h_i, which are indicative of that example's relationship to each projection. Effectively, this process identifies 'which side of the projection' the example inhabits within the space and are calculated by thresholding a dot product comparison:

$$h_i = \begin{cases} 0, \text{ if } x \cdot v_i < 0 \\ 1, \text{ if } x \cdot v_i >= 0 \end{cases} \qquad \forall v_i \in V \tag{3.9}$$

H is then the ordered concatenation of each h_i it contains, allowing it to act as an identifier for a specific bucket. As the indexing system is based upon a similarity comparison, the buckets preserve locality information from the original distribution of the space and there is a high probability that similar examples are allocated to the same bucket (supporting the declaration in Equation 3.8). These buckets are therefore well-placed to quickly identify the locality of an instance, allowing nearest neighbour calculations to only be performed between occupants of the same bucket. This is significantly computationally faster and cheaper than brute-force neighbour calculations.

3.2 Neural Networks and Deep Learning

As DMLs are neural in nature, in this section we provide an introduction to the general concepts of neural networks and deep learning. This will clarify that training of a neural network allows learning of non-linear feature combinations to create an abstract representation of the original input data. This is a fundamental advantage of DMLs over representation learning methods which operate on feature reduction, such as Principle Component Analysis, and so it is necessary we provide a general introduction to the mathematical foundation of the concept.

Neural networks are a collection of biologically inspired machine learning algorithms. As parametric representation learners, neural networks receive large amounts of input data to 'learn' a transformative function that will convert input data into a representation which is more effective for classification or regression tasks (note that in this work, we focus on classification due to the nature of our use cases). Though motivated by the neural networking systems which exist within the brain, neural networks are more closely related to calculus graphs and rely on backpropagation (known in mathematics as reverse-mode differentiation) throughout training to learn to approximate linear or non-linear functions. The importance of deep learning architectures (from the perspective of this thesis) is the ability to leverage non-linear activations to generate a representation of the input data which is optimised to meet a given objective. These aspects are very important when we move on to discuss the capability of DMLs, and so we provide a working introduction to them here.

3.2.1 Neural Networks

With a few exceptions, neural networks are considered a supervised learning method, as their development requires a set of example data (the training set) where the desired output is already known. Shallow networks are composed of three layers - an input layer, a 'hidden' layer and an output layer - each of which are composed of a number of neural nodes which are linked to every node on the previous layer by a set of weights and biases. Deep neural networks have multiple hidden layers, allowing them to learn deep functions (hence their namesake). Data is input to the neural network and is fed through in sequence, with each layer of the network reliant upon the output of the preceding layer. As each node of the input layer represents an individual feature from the input data, this means that every feature has an impact on all nodes in the hidden and output layers. This is important, because it means each node can leverage combinations of input features from previous layers to feed into its value. Therefore, each layer of the network creates a representation of the input data built upon a learned combination of the raw features. The learned representation is flexible, as it is does

not suffer from the reliance upon task-specific rules or algorithms which often plagues manually-coded features.

To formalise learning in a neural network, let us consider a dataset of cases, $x \in \mathcal{X}$. Each case is comprised of a numerical vector of feature values such that, $x = (x^{f1}, x^{f2}, ..., x^{fm})$, where m denotes number of features. When input to a neural network, each of these features occupies a single node in the input layer. Each node in the input layer is connected to every node in the subsequent hidden layer h by a set of weights $w \in W$. Each weight is a learned parameter which controls the level of contribution of each feature of the input case to each node of the hidden layer, meaning W is a matrix of dimensions $m \times |h|$. Therefore we can weight each feature of the input by taking the dot product of x and W. Each of the weighted input features are then summed with a set of learned bias values B, a vector of bias of size $|h|$ such that there is a bias variable for each hidden node. Finally, a derivable activation function $a()$ is applied to the output of that summation to allow non-linear combination of inputs. It is important that the activation is derivable, to allow training to occur using backpropagation, and non-linear, as this allows the network to approximate non-linear relationships between features in the data. We summarise this for the hidden layer overall as:

$$h(x) = a(x \cdot W + B) \tag{3.10}$$

The output of the hidden layer is a new vector representation of the original input data in which each feature is developed from a non-linear combination of input data features. The representation can then be passed to the next layer in the network architecture, which may be another hidden layer or an output layer.

When passed to a subsequent layer, the function becomes nested. This means that features from the previous layer are fed into a subsequent layer, allowing the features learned at the previous layer to be combined together and transformed (with non-linear activation functions) to create a new representation. This process can be repeated multiple times as desired to learn a representation of the data which is built upon combinations of combinations of features. We can represent this as:

$$h^{l+1}(x) = a^{l+1}(h^l(x) \cdot W^{l+1} + B^{l+1}) \tag{3.11}$$

We can see that the transformation of the data at the subsequent hidden layer, h^{l+1}, is obtained by obtaining the dot product of the previous layer, h^l, and the weight matrix for the new layer, W^{l+1}, then summing the biases, B^{l+1}, before finally applying the activation function, a^{l+1}. In such a manner, the function can become repeated multiple

times. Hence the neural network becomes 'deep', giving rise to the title deep learning. This is valuable because it means the model can learn to leverage information which is only available by making multiple combinations of features, which is typically difficult to build into hand-crafted knowledge models. Furthermore, the trainable weights within the neural network model adapt as feature combinations are learned.

At the output layer, the data must be converted to a format that is appropriate to meet network objectives. For classification, this requires that the input data be attributed to one of multiple distinct groupings called 'classes'. To achieve this, the network calculates a probability distribution which is indicative of class membership. Calculating the probability distribution requires *a priori* knowledge of the number of classes in the training data, meaning that conventional deep learning and neural network architectures are incapable of handling problems with unknown classes. The distribution is typically calculated using a softmax function:

$$Softmax(x) = \frac{e^{x_i}}{\sum_j^{|Y|} e^{x_j}} \tag{3.12}$$

The output of the softmax function is a vector of the same dimensionality as the size of the set of unique items in the label set, $|Y|$, and the values of which sum to a total of 1. A label is decided for the input data by the highest probability value.

Considering our analysis, we can represent the transformation of input data to predict a label as the function $\theta()$:

$$\theta(x) = Softmax(h^n(x)) \tag{3.13}$$

where n is the number of hidden layers in the network.

3.2.2 Training a Neural Network

Given our understanding of how data is transformed throughout the network, we will now discuss how the network is trained to achieve its objective. Here the term 'training' refers to the update of network weights and bias values as informed by the data. We can decompose the concept of training into three distinct steps:

- Identifying the objective of the model, and formalising how well network output meets this objective using a loss function.
- Using information provided by the loss function to calculate the gradient of the error using backpropagation.
- Updating the weights and biases of the network to reduce the error using an

optimizer function.

A neural network aims to meet an 'objective' by learning to model a function using knowledge elicited from input data. The purpose of a loss function is to quantify the difference between network output and expected output. We can describe the value obtained as output from this function as 'loss' or 'error'. Training then becomes a matter of reducing the error produced by a loss function until it converges to a global optima, at which point we know that we have modelled the function to the best of the capabilities of the network architecture. Since the purpose of deep learning is to approximate an unknown function, it is unlikely (though not impossible) that we will reach perfect performance such that this loss value will be 0.

In classification, the purpose of the network is to attribute the input data to one of multiple classes. As we have intimated above, we can calculate the probability of the data belonging to one of these classes using a softmax function. The output of this function will give us a vector which sums to a total of 1 where we can use the index with the highest value as an indicator of class membership. Ideally the output of this would be a one-hot encoded vector where the probability for the correct classification is 1 and the rest of the values are 0. Therefore, the error becomes the difference between the real one-hot encoded class label, $y(x)$, and the probability distribution as predicted by the neural network, $\theta(x)$. We can calculate this using Mean Squared Error (MSE):

$$L(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} \big(\theta(x_i) - y(x_i)\big)^2 \tag{3.14}$$

where N is the number of items in the training set. From this we can quantify the extent to which the network meets its objective. This knowledge will allow is to calculate the gradient of the error for each individual weight using backpropagation.

While we have used a straightforward example in this description, it is worth emphasising that the loss function is a measurement of how well the network meets its objective. As a result, the loss function should complement the objective of the network, meaning that there have been a number of different loss functions developed to meet the needs of different objectives. In Section 3.3 we examine two of the most popular loss functions for deep metric learning in more detail and discuss how they support the objective of clustering similar cases.

Given an understanding of how much a network deviates from its objective (in the form of loss), we can now 'train' the network to reduce this deviation. This is achieved through data-driven alteration of the weights and biases (collectively referred to as 'weights' henceforth) that exist between the hidden layers of the network. To do this

we need to understand the contribution of each individual weight to the total loss value. We can identify this by calculating the gradient of the loss function with respect to each weight. Because the activation functions are derivable, we can capture this information using backpropagation, an algorithm which relies heavily on the chain rule from calculus. After we have identified the gradient of the loss function with respect to each weight, we can use an optimiser function to adjust the weights using gradient descent.

3.3 Deep Metric Learning

Using our knowledge of deep learning as detailed in the last section, we now explore the unique traits of several DML architectures, including Siamese Neural Networks and Triplet Networks. We provide an in-depth exploration of each network and describe in detail how each architecture and metric-based loss function contributes to learning of a latent space optimised for similarity calculations.

Though individual architectures possess distinct nuances, there are several themes which are consistent across DMLs. With this in mind, let us introduce some general notation used throughout this thesis. Let $\mathcal{X}$ be a set of labelled cases, such that for $x \in \mathcal{X}$, the function $y(x)$ returns the class label, y, of case x. In the context of this paper, we will define matching cases as those which have the same class while non-matching cases will have differing classes. The embedding function θ is an appropriate parameterisation of any function used to create the vectorised representation of a given x, while the function D_w represents an arbitrary metric function to measure the distance between two vector representations.

3.3.1 Siamese Neural Networks

Siamese Neural Networks (SNN) are deep metric learners which receive pairs of cases as input to learn a matching task. The SNN architecture consists of two identical embedding functions, which usually take the form of neural networks. The output of these embedding functions, $\theta()$, are feature embeddings for each member of the input pair. During training it is these embeddings that are used for any distance computations, thereby ensuring iterative model refinement through. Input pairs are labelled as either matching or non-matching respectively and the network is trained to differentiate between matching and non-matching pairs. Correspondingly, the objective of training is to minimise the distance between the generated embeddings for matching pair members while maximising the distance between embeddings for non-matching pair members. Thus the overall goal of the network is the development of a space optimised for similarity-based return.

To achieve this goal, each training pair, $p \in \mathcal{P}$, consists of two cases from the training set, $p = (\hat{x}, \ddot{x})$. Whether the pair is matching or non-matching is governed by the relationship of the pivot case, $\hat{x}$, to the passive case, $\ddot{x}$ and motivated by a matching criteria. Though this matching criteria can in effect be anything (such as the presence/absence of certain features, expert annotation or cluster membership), in its simplest form, the pair's relationship class is established by comparing class labels of its members (i.e. $y(\hat{x})$ with $y(\ddot{x})$). For this we use function $Y(p)$, which returns p's relationship class label, such that $Y(p) = 0$ to symbolise a matching pair when $y(\hat{x}) = y(\ddot{x})$, and $Y(p) = 1$ to symbolise a non-matching pair when $y(\hat{x}) \neq y(\ddot{x})$.

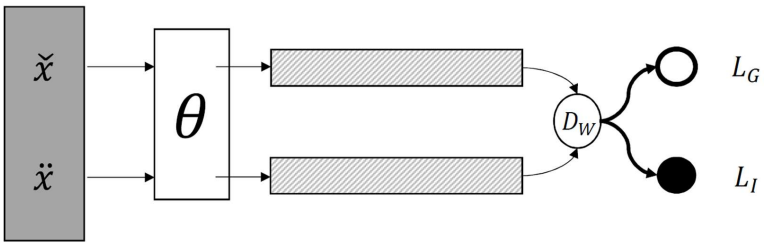


Figure 3.2: Siamese Neural Network Architecture

During training the network develops a multi-dimensional embedding based upon the input training pairs, $\mathcal{P}$. This is facilitated by having the shared layers; essentially these layers enable the SNN to generate an embedding for each member of a pair (see Figure 3.2). Thereafter members can be compared using a distance metric, D_W, which influences the computation of the loss. The loss function formulates the pair prediction error on the basis of matching and not matching error predictions, since it is distance-based the loss also causes the network to directly learn a similarity metric. In the following subsections we will examine this distance-based loss in more detail.

SNNs were originally designed to use 'contrastive loss', as introduced in [14]. Contrastive loss is calculated by summing the results of the individual loss formulas for genuine (matching) and impostor (not matching) pairs. Genuine pairs are penalized by loss L_G for being too far apart, while negative pairs are penalized by L_I if their distance falls within the given margin value. We can observe this in Figure 3.2 and Equations 3.15, 3.16, 3.17.

$$L_G = (1 - Y_A) \cdot {D_W}^2 \tag{3.15}$$

$$L_I = Y_A \cdot (max(0,\ \alpha - D_W))^2 \tag{3.16}$$

$$L = L_G + L_I \tag{3.17}$$

The result is that distance between constituents of genuine pairs are minimised over the course of training whilst ensuring that impostor pairs maintain at least a set margin of α distance apart. The similarity metric is therefore directly learned by the network, as it is implicitly defined by the loss function.

Variable	**Definition**
$\hat{x}$	One half of the input pair. We describe it as the pivot instance, as the value of Y_a is decided by comparing $\ddot{x}$ to it
$\ddot{x}$	One half of the input pair. We describe it as the passive instance, as the value of Y_a is decided by comparing it to $\hat{x}$
L	The loss function which is passed to the network
L_G	The loss component specifically for genuine pairs
L_I	The loss component specifically for impostor pairs
Y_a	The variable which represents the true matching status of the pair - 0 means genuine, while 1 means impostor
D_w	A function to calculate the distance between $\theta(\hat{x})$ and $\theta(x')$
$\theta()$	A parametric embedding function, usually in the form of a neural network
α	The margin value which must exist between impostor pairs

Table 3.1: Details of Contrastive Loss

Taking a closer look, contrastive loss is the comprised of the summation of two loss functions, but these loss functions only activate when the appropriate pair match is presented. The function L_G will only return a value when a genuine pair is presented to the network (otherwise it will return zero), while the function L_I will only return a value when the an impostor pair is presented. This functionality is controlled by the value y_a. Since y_a is set to either 0 (if the pair is genuine) or 1 (if the pair is impostor) a specific loss function can be called for each type of pair.

The loss function for genuine pairs (L_G) only activates when a genuine pair is called. This is controlled by $(1-y_a)$. If y_a represents the relationship of an impostor pair, then it becomes equal to 1. This results in the function becoming $0({D_w}^2)$ which produces a zero for return. If y_a is equal to 0, then the equation simply becomes ${D_w}^2$, which is the distance between $\theta(\hat{x})$ and $\theta(\ddot{x})$ squared. If the distance between instances is large, then the loss itself is large and weights are more influenced to change. If the distance between instances is small then the generated loss value is low so weights receive less update. The result is that the pivot and passive instance are decreasingly penalised until they occupy the exact same space.

The loss function for impostor pairs (L_I) only activates when an impostor pair is called,

again controlled by the variable y_a. If y_a represents the relationship of a genuine pair (i.e. it is equal to 0), then the function will return 0. In the impostor loss function, if the distance between pair members is larger than the specified margin, then subtracting it from the margin α will result in 0 being returned, as controlled by $max(0, \alpha - D_w)$. Note that the $max()$ function ensures that only a loss value which is greater than or equal to zero will be generated. If the distance between pair members is smaller than the margin, then this will result in a value between 0 and α, which will be returned. As a result, the maximum loss which can be returned by an impostor pair is the value of α, which will be returned if the passive and pivot instance are identical. This is an interesting contrast to the genuine loss, which can create a potentially infinite loss value dependent on the distance between pair members (see Figure 3.3 for a visualisation).

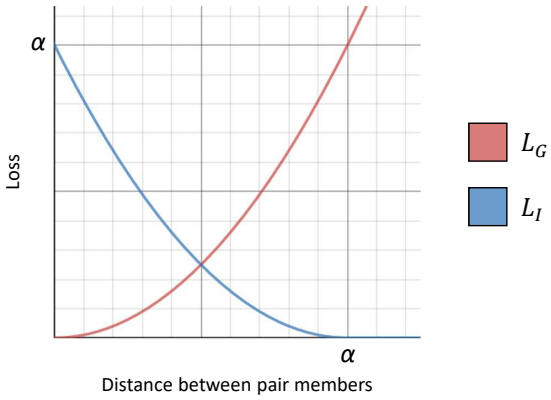


Figure 3.3: Graph demonstrating contrastive loss

Pair Classification as a Proxy for Representation Goodness

As stated above, an SNN is trained to make predictions about a pair's label - i.e. matching or non-matching. We can use this as one method to evaluate representation goodness. At test time the SNN can obtain a predicted pair label by comparing D_W with the margin threshold α. If D_w is less than α then the network judges the pair to be genuine, otherwise it is classified as an impostor. The question then becomes how this can be applied to unseen test data. For classification problems, a label for an unseen test example, x_q, can be obtained by pairing it with a representative training example from each class, $(\bar{x_i}, ..., \bar{x_m})$, where m is the number of classes. In problems without class knowledge, the representatives can be selected from each cluster.

However, this poses another question in how class/cluster representatives should be selected. Intra-class variation means that selecting an unsuitable representative may result in an incorrect classification. To avoid this, representatives can be selected based upon prototypical instances within a class/cluster. We can then use a distance weighted

voting algorithm to determine the classification of a query example from its nearest class prototype neighbour ($\bar{x}_i$). Note that only genuine pairings with $\bar{x}_i$ contribute to the classification vote:

$$vote(c_i) = \sum_{j=1}^{m} w_j * [Y(c_i, y(\bar{x}_j))\ (1 - Y(\bar{x}_j, x_q))] \tag{3.18}$$

$$w_j = \frac{1}{\sqrt{(\sum_i |f_\theta^i(x_q) - f_\theta^i(\bar{x}_j)|^2)}} \tag{3.19}$$

The classification with the highest vote is deemed to be the classification of x_q. The weighting is based on Euclidean distance between examples using their feature embeddings from the network.

3.3.2 Triplet Networks

Triplet networks (TN) are DMLs which learn from three input cases simultaneously. Together described as a triplet, these inputs are the anchor case (x^a), a positive case (x^+) and a negative case (x^-). The anchor case acts as a point of comparison, meaning that the positive and negative cases are dictated by their relationship to the anchor (i.e. matching and not matching respectively). Similar to the SNN, the objective during training is to minimise the distance between an anchor and its associated positive case and maximise the distance between an anchor and its associated negative case. However, considering three cases at once ensures that update of weights is more focused. This is because the SNN is learning based on only one aspect at any given time (e.g. either pair members are alike, or not), meaning that more pairs are required to build the full picture. Considering three cases at once allows the triplet network to better understand the context of the anchor case.

A triplet network is comprised of three identical embedding functions, each of which creates an embedding for an input (see Figure 4.6) before the error is calculated using triplet loss:

$$L = D_W(\theta(x^a), \theta(x^+)) - D_W(\theta(x^a), \theta(x^-)) + \alpha \tag{3.20}$$

Like contrastive loss (see Equations 3.15 - 3.17), triplet loss is a distance based function. The formula will generate a loss value in situations where the distance between the anchor case and the negative case, $D_W(\theta(x^a), \theta(x^-))$, is less than the distance between

the anchor case and the positive case, $D_W(\theta(x^a), \theta(x^+))$. The network is therefore penalised until matching cases are closer than non-matching cases. A minimum boundary between non-matching cases is enforced by the margin α.

Variable	**Definition**
x^a	The anchor example. During training this instance should meet the matching criteria for x^+ and not for x^-.
x^+	The positive example. During training this instance should meet the matching criteria for x^a and not for x^-.
x^-	The negative example. During training this instance should meet the matching criteria for neither x^a nor x^+.
D_w	A function to calculate the distance between two examples.
$\theta()$	A parametric embedding function, usually in the form of a neural network
α	The margin value which must exist between impostor pairs

Table 3.2: Details of Triplet Loss

Unlike in contrastive loss, triplet loss considers three output embeddings simultaneously to inform its comparisons. The anchor example is compared to both the positive example and the negative example in $D_W(\theta(x^a), \theta(x^+))$ and $D_W(\theta(x^a), \theta(x^-))$ respectively. This means the network from two pair comparisons at the same time, allowing the loss function to better incorporate knowledge of the space. Furthermore, an important distinction between contrastive and triplet loss is that while the former relies on an explicit matching label, this information is only implicitly captured in the triplet (in the form of the positive and negative examples). Contrastive loss is fully reliant on this pair label to generate loss (where y_a controls which aspect of the loss function is activated). The combination of considering only a single pair and basing the comparison on a binary value mean that networks trained with contrastive loss learn a matching function. However, in triplet loss because two pairs are being compared (anchor-positive and anchor-negative), the function is enabled to consider relative similarity of the pairings. Matching examples are not forced to the same point - only to be more similar than non-matching examples (plus a threshold margin value). This means that the loss function is significantly more robust to intra-class variance. Furthermore, the objective of the function is more closely aligned with similarity-based return. Overall, the differences between the functions are indicative that contrastive loss is more useful in settings where matching is required, while triplet is more useful in circumstances where fine-grained similarity knowledge is important.

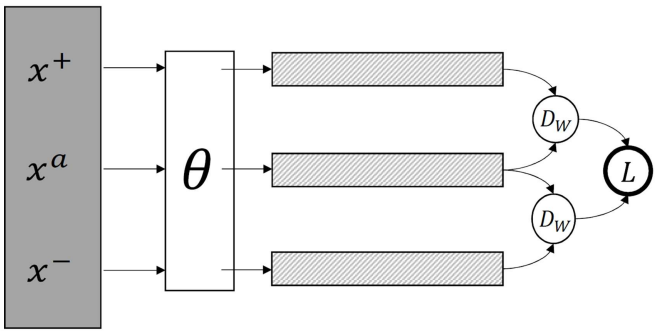


Figure 3.4: Triplet Network Architecture

3.4 Conclusion

To summarise our observations from this chapter, we have discussed specific implementation and mathematical details for several algorithms which support the vocabulary and similarity knowledge containers in CBR. The purpose of this is to highlight that CBR is a methodology supported by a range of algorithms, the properties of which can be broadly categorised using the knowledge container model. From our exploration of several example algorithms, it is clear that the vocabulary and similarity knowledge containers are traditionally fulfilled by algorithms which have little no overlap between one another, thereby preventing knowledge from either container from impacting the algorithm within the other. For example, and specifically of interest to this thesis, knowledge of case relationships play no role in the representation which is learned to characterise case features. We highlight DMLs could offer a bridge between the vocabulary and similarity containers specifically, as the representations they learn are guided by similarity. This is because DMLs are deep learning architectures which learn a representation optimised for similarity comparisons by considering the relationships between input cases during training.

Training of DMLs is enabled by the neural network architecture, which is able to learn non-linear combinations of input features to develop a new representation. DMLs are characterised by combination of (1) a network architecture capable of comparing multiple examples simultaneously and (2) a metric-based loss function which will utilise the knowledge between input cases to guide formation of network parameters. As our investigation in this Chapter and in Section 3.3 has emphasised, the output of these loss functions are fully dependent on the similarity between input examples; ergo, to allow the loss function to actually update the parameters within the neural network, samples must be selected such that the comparison will generate loss. This demonstration identifies the third crucial component of a DML architecture - a training strategy. The training strategy will ensure that the combination of input examples (be they pair,

triplet or a subset of examples) will actually generate loss to contribute to network development. Intuition would suggest a training strategy should incorporate similarity knowledge between examples, as has often been explored in the literature [18, 96, 102]. Such a strategy would need to target complex areas of the space while maintaining knowledge of the overall distribution of examples. We discuss our work towards the development of two strategies for optimising the training of DMLs in Chapter 4.

Chapter 4

Similarity Knowledge for Training Deep Metric Learners

We are motivated by our findings in Chapter 2 and Chapter 3 to develop several training strategies for DMLs which leverage traditional techniques from CBR and meta-learning. In doing so, over the course of this chapter we aim to demonstrate the impact that traditional methods can have to improve the training of DMLs. Firstly, we examine meta-learning research in Boosting to explore the feature space and exploit complex knowledge. Our initial findings indicate that utilising similarity information is promising for training Siamese Neural Networks. However our proposed sample selection method is computationally expensive and unsuitable for large or complex datasets. Once again taking inspiration from traditional machine learning techniques, we are inspired by methods to reduce the complexity of similarity comparisons to develop a training strategy suitable for Triplet Networks. Incorporating Approximate-Nearest Neighbour (a-NNs), we introduce a locality-sensitive batching strategy, which uses the locality of examples to create batches as an alternative to the commonly adopted randomly minibatching. Our results demonstrate this method to offer better performance on three image and two text classification tasks with statistical significance. Importantly most of these gains are incrementally realised with as little as 25% of the training iterations, and are computationally inexpensive in comparison to our initial strategy.

This chapter is structured as follows. In Section 4.1 we state the primary contribution which is addressed by this chapter, and how this is broken down into a number of secondary contributions. In Section 4.2 we present several novel sample selection strategies for pair-based architectures such as the Siamese Neural Network. This section is further broken down into the following subsections: in Section 4.2.1 we formalise

pair creation and its role in SNN learning before we introduce the concepts of pair ordering and informed pair selection employed in our work; in Section 4.2.2 we present a comparative study of the proposed methods with results on the MNIST, IMDB and SelfBACK datasets appearing in Section 4.2.3.

We reflect on our results and the limitations of the proposed strategies in Section 4.3, which allows us to identify several key areas of improvement. We identify an avenue forward in Section 4.4, where we present a novel batching strategy for Triplet Networks which is based on the ideals of similarity we had pursued previously. This section is divided into a number of subsections to structure our contribution: in Section 4.4.1 we review the triplet network architecture and identify several different training strategies for our evaluation. In Section 12 we present our method for creating locality-sensitive triplets. In Section 4.4.2 we layout the details of our evaluation while in Section 4.4.3 we discuss the results of our experiments and our accuracy on several datasets. Finally in Section 4.5 we provide some conclusions.

4.1 Research Question and Contributions

In this chapter we explore the hypothesis that training efficiency of DMLs can be improved by incorporating techniques from traditional methods of machine learning, such as meta-learning and CBR. With that in mind, the research question we aim to answer in this chapter is:

- How can techniques from traditional machine learning methods (such as CBR and meta-learning) be incorporated into strategies to improve training efficiency of DMLs?

In our work towards answering this research question, we highlight the primary contribution of this chapter. We introduce several training strategies for DMLs which are inspired by research in meta-learning, curriculum learning and CBR. Experiments on public datasets from multiple domains illustrate that the proposed strategies improve training efficiency of DML architectures. This contribution can be further divided into a number of secondary contributions:

1. Taking inspiration from current research into optimising the training of DMLs and historical research into meta-learners, we introduce two methods for informed pair selection (DYNE and DYNEE) that optimise pair creation by leveraging the concepts of exploration and exploitation.

2. Encouraged by the results of recent work in curriculum learning we introduce a

pair complexity heuristic for ordering that draws on knowledge about the neighbourhood properties of pairs.[1].

3. Building on the limitations of our pair selection strategies, and motivated by techniques from CBR (see Section 2.1.2), we present an incremental locality-sensitive batching strategy for triplets (LSB) which allows the batching to evolve alongside example representations over the course of training.

4.2 Siamese Neural Networks

The Siamese Neural Network (SNN) is a deep learning architecture which trains upon pairs of input data to learn a metric space in which training instances can be placed. The expectation of paired learning is that the learned space can better represent salient relationships between pairs which can then be better captured in Euclidean space. Originally used in binary classification tasks such as signature verification [14] and face recognition [15], SNNs have recently been generalised to multi-class classification [17].

Central to SNN learning is the use of similarity knowledge to gauge closeness of data points such that it provides a meaningful matching criterion. The expected outcome of SNN training is to have instances deemed similar to be mapped closer together. As SNNs are metric learners that develop a new representation of the original data, in a classification setting they require a non-parametric learner (such as k-NN) to perform the explicit classification. However, a benefit of SNNs is that they do not require full class knowledge during training, as they are learning on the basis of whether pairs meet the matching criteria, not whether they belong to a specific label. For this reason, recent work has demonstrated these networks can perform effectively even when working with extremely limited training data, such as in a one-shot learning environment [17].

Surprisingly, pair selection and ordering have been given little attention in SNNs, despite showing promising results in the closely-related Triplet Networks (TNs) [18, 78]. Pair selection and ordering directly informs the data relationships that the network will learn and be trained upon, thereby directly influencing the metric developed by the network. With our review of the literature in mind (see Section 2.2.3), it is our view that these strategies play an important role in both improving training time and ensuring high overall performance of SNNs. Pair selection and presentation order are likely to be particularly relevant in data-budgeted scenarios where there are only a small number of annotated examples.

In this section we discuss the importance of pair selection strategies and their effect

[1]The code associated with this contribution is publicly accessible at https://github.com/RGU-AI/Informed-Pair-Selection

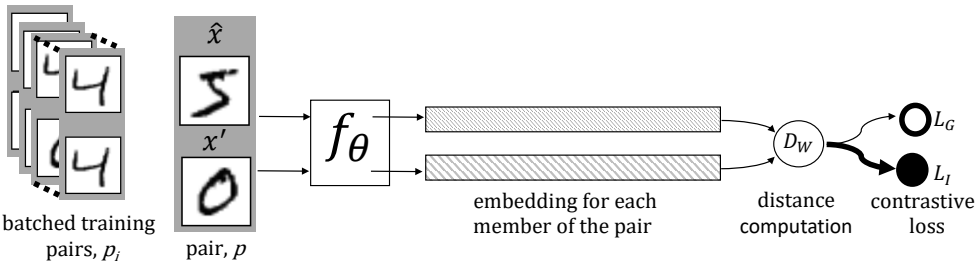


Figure 4.1: The SNN learning process

on training. We are of the opinion that only by understanding the impact of sample selection in multiple-input networks can we build upon these ideas for application towards recent advancements in deep metric learners. For this reason we start with SNNs. Although we limit our scope to SNNs, the contribution from this section is applicable to other neural network architectures that learn from multiple examples - particularly triplet [16] and matching networks [77].

4.2.1 Training with Pairs in an SNN

The SNN architecture consists of two neural networks that share identical weights and are joined at one or more layers [14]. SNNs receive pairs of examples as input during both training and testing to develop similarity knowledge at an object-to-object level.

We summarise the notation used in this chapter to assist presentation of the different pair creation strategies. For an in-depth discussion of the parameters, please see Table 3.1. Let $\mathcal{X}$ be a set of labeled examples, such that example, $x \in \mathcal{X}$ and $y(x)$ is a function that returns the class label, y, of x. $\mathcal{P}$ is a set of pairs $(p_1, ...p_n)$ that form the paired training batches for input to the SNN (see Figure 4.1). Each training pair, $p \in \mathcal{P}$, consists of a pair of examples, $p = (\hat{x}, x')$, where $\hat{x}$ is a pivot example whose relationship to passive example x' dictates whether the pair is of class genuine, or impostor. Here the pair's relationship class is easily established by comparing class labels $y(\hat{x})$ with $y(x')$. For this we use function, $Y(p)$ or $Y(\langle \hat{x}, x' \rangle)$, which returns, p's relationship class label, such that $Y(p) = 0$ when $y(\hat{x}) = y(x')$, and $Y(p) = 1$ when $y(\hat{x}) \neq y(x')$. Typically $\hat{x}$, x' are randomly selected, to form the genuine and impostor relationship pairs for training.

During training the network develops a multi-dimensional embedding based upon the input training pairs, $\mathcal{P}$. This is facilitated by having the shared layers; essentially these layers enable the SNN to generate an embedding for each member of a pair (see Figure 4.1). Thereafter members can be compared using a distance metric, D_W, which influences the computation of the two loss components: loss due to pairs being

further apart when they should not be, L_G; and loss due to pairs being too close when they should be further apart, L_I. Contrastive loss, L (as in Equation 4.3), is commonly used to guide the sub-network weight update for model learning by combining these two losses - genuine L_G and impostor L_I [15]. It essentially formulates the pair prediction error on the basis of genuine and impostor error predictions. The use of both genuine and impostor error means that the similarity metric can be directly learned by the network through the comparison of the actual pair label Y_A (equal to 0 for genuine and 1 for impostor pairs) and the distance, Euclidean or otherwise, between pair members, D_W.

This means that distance between constituents of genuine pairs are minimised over the course of training, whilst ensuring that impostor pairs maintain at least a set margin of M distance apart.

$$L_G = (1 - Y_A) \cdot {D_W}^2 \tag{4.1}$$

$$L_I = Y_A \cdot (max(0,\ \alpha - D_W))^2 \tag{4.2}$$

$$L = L_G + L_I \tag{4.3}$$

The output of the identical neural networks (or 'sub-networks') form feature embeddings, f_θ, for each member of the input pair. During training it is these embeddings that are used for any distance computations, thereby ensuring iterative model refinement through contrastive loss based back propagation.

In the following sections we introduce pair creation strategies for SNN training, with strategies that are informed by knowledge about areas of difficulty (exploitation) and strategies that balance this with the need for problem space coverage (exploration).

Explorative Pair Selection

It is important to note that $\mathcal{P}$ only represents a small subset of all possible pairs which can be obtained by exhaustively pairing all examples in the training set (the total size of which would be $|\mathcal{X}|^2$). The result is that $\mathcal{P}$ gives a narrow view of instance relationships. We can improve this by initiating multiple pair selection sessions throughout training. Doing so allows us to explore the relationships between examples more thoroughly. Specifically instead of a static $\mathcal{P}$, we can create a $\mathcal{P}$ for each cycle of training, where a cycle will consist of a set number of training epochs.

Algorithm 2 lists the steps involved with random creation of a *Explore* pair set, where given n, a call to $\mathcal{P}_{RND}(n)$ assigns n pairs to $\mathcal{P} := \mathcal{P}_{RND}(n)$. Here $\hat{x}$ and its paired members x'_1 and x'_2 are randomly selected from $\mathcal{X}$ with the only condition that the two pairs formed must provide the necessary genuine and impostor representatives; such

Algorithm 2: Algorithm to create the Explore Set

Explore: $\mathcal{P}_{RND}(n)$
P_I, $P_G := \emptyset$
for $i = 1 \ldots n/2$ **do**
    $\hat{x} :=$ rnd_selection($\mathcal{X}$)
    $x'_1 :=$ rnd_selection($\mathcal{X}$) $\wedge\, y(\hat{x}) \neq y(x'_1)$
    $x'_2 :=$ rnd_selection($\mathcal{X}$) $\wedge\, y(\hat{x}) = y(x'_2)$
    $P_I := P_I \cup p(\hat{x}, x'_1)$
    $P_G := P_G \cup p(\hat{x}, x'_2)$
end
Explore $:= P_I \cup P_G$
return *Explore*

that $Y(P_I)$ is 0 and $Y(P_G)$ is 1.

Exploitative Pair Selection

Inspired by uncertainty sampling and boosting we can utilise information that we gain during the previous training cycle to inform pair selection for the next training cycle. Here instead of only exploring the problem space randomly, we integrate an exploitation phase such that pair selection will be guided by sampling in areas found to be 'hard' for the learner. Specifically for each $p_i \in \mathcal{P}$ we use the network's predictions, $Y(p_i)$ and associated loss to rank elements in $\mathcal{P}$. We extract the 'hardest' ranked pairs (i.e. pairs with the highest loss), from which we generate new pairs to form the exploitation set. We represent the ratio of exploit to explore as β. The main idea is to use this ratio to guide pair creation in areas of uncertainty (See Algorithm 3).

Algorithm 3: Algorithm to create the Exploit Set

Exploit: $\mathcal{P}_{NN}(\mathcal{P}')$
P_I, $P_G := \emptyset$
for $p_i \in \mathcal{P}'$ *where* $\mathcal{P}' \subset \mathcal{P}$ *and* $|\mathcal{P}'| = \beta$ **do**
    $p'_i = (NN_1(\hat{x}),\, NN_1(x'))$
    if $Y(p'_i) = 0$ **then**
        $P_G{:=}P_G \cup p'_i$
    end
    if $Y(p'_i) = 1$ **then**
        $P_I := P_I \cup p'_i$
    end
end
Exploit $:= P_I \cup P_G$
return *Exploit*

For each selected pair, we find the nearest neighbour of each member within each pair using function, NN_i. By taking the neighbours of the original difficult pair, we generalise network attention to the complex area of the space without overfitting on specific examples. These neighbours form the basis for a new pair for our training set. It is worth noting here that it is possible to develop the entire training set by using the exploit algorithm and setting β equal to 1. We found this to be detrimental to training, as the network tended to overfit to specific difficult areas, become trapped in a local optima and develop a distorted feature embedding as a result. Hence we suggest using an Explore-Exploit ratio to prevent this.

Explorative and Exploitative Pair Selection

A mixed approach that allows the learner to both explore and exploit requires pair selection that can utilise pairs formed using both strategies from previous subsections. We accomplish this by randomly creating pairs to perform early exploration of the feature space through a 'dry run' of training the network for a small number of epochs (typically ten or less) which helps to initialise network weights. Thereafter we use the ratio β to generate a new set of exploit pairs (as in Algorithm 3); and the rest will consist of a new set of explore pairs (as in Algorithm 2).

Algorithm 4: Algorithm to combine Explore and Exploit Sets

Explore&Exploit: $\mathcal{P}_{HYBRID}(\mathcal{P})$
$b := \beta \cdot |\mathcal{P}|$
for $p_i \in \mathcal{P}$ **do**
  $\mathcal{L}$:= $\mathcal{L}$.**append**($L(\theta, p_i)$)
end
$\mathcal{P}$:= $\mathcal{P}$.**sort**($\mathcal{P}$, $\mathcal{L}$, $<$)
for $i = 1 \ldots b$ **do**
  $\mathcal{P}$**' :=** $\mathcal{P}$**'.add**(p_i)
end
Exploit **:=** $P_{NN}(\mathcal{P}')$
Explore **:=** $P_{RND}(\mathcal{P} \setminus \mathcal{P}')$
return $Explore \cup Exploit$

The loss, L, for each p_i is maintained in $\mathcal{L}$, which is based on current network parameters θ. Pairs are sorted in decreasing order of loss and the top β pairs are used for exploit pair generation and the rest generated through the explore strategy. This process is repeated multiple times during training, as the areas of the feature space that the network will find complex will very likely change as the network 'learns' by refining θ. Note that NN_i's similarity computations are influenced by feature embeddings on the basis of the latest θ - i.e. they are based on activations of the last network layer at the

current point in training.

It is also important to note that the suggested algorithms do not sample from a larger training set than the baseline method. It is merely the way in which instances are paired that changes. For example, in a data-budgeted scenario where only 1% of a dataset is available (such as in one of our evaluations later in this chapter), these algorithms will operate within that budget and will not sample additional data from the training set.

Heuristic Ordering for Self-Paced Learning

To develop a structured ordering method for our pairs, we take inspiration from complexity measures used in neighbourhood analysis for case-based reasoning systems [129]. The basic idea is that an area is considered complex when neighbourhoods of examples are found to be non-homogeneous in terms of their class labels. We adopt this for complexity analysis C for a given pair p (instead of a single example) as in Equation 4.4.

$$C(p) = \frac{\sum_i \sum_j EQ(Y(p), Y(\langle NN_i(\hat{x}), NN_j(x')\rangle))}{\sum_i \sum_j (1 - EQ(Y(p), Y(\langle NN_i(\hat{x}), NN_j(x')\rangle)))} \tag{4.4}$$

The numerator in the complexity ratio counts the number of pairs formed in the neighbourhood that differ from the class of the original pair and denominator counts those that are of the same pair label (i.e. is it an impostor or genuine pair). Here $NN_i(.)$ denotes the ith nearest neighbour of a given example. We create all possible pairs between $\hat{x}$'s and x''s neighbourhoods (see Figure 4.2). For any given pair, function Y returns the pair's class label (0 for genuine and 1 impostor). With self-paced learning we can use any of the pair selection strategies and sort pairs by the complexity metric for model training.

4.2.2 Evaluation

The aim of our experiments is two fold. Firstly, we aim to investigate the effect of incorporating a pair creation method by analysing three variations of pairing strategies: no strategy, a dynamic strategy (exploration) and an informed dynamic strategy (exploration/exploitation). Secondly, we aim to investigate the effect of complexity-based ordering, giving us an unordered and an ordered variation of each strategy and a total of six candidate approaches:

1. **Base**: Pairs are unordered, and are not updated throughout training. As such this is a static, standard paired-training used for SNNs - the baseline.

2. **Base***: As Base but now with pairs ordered .

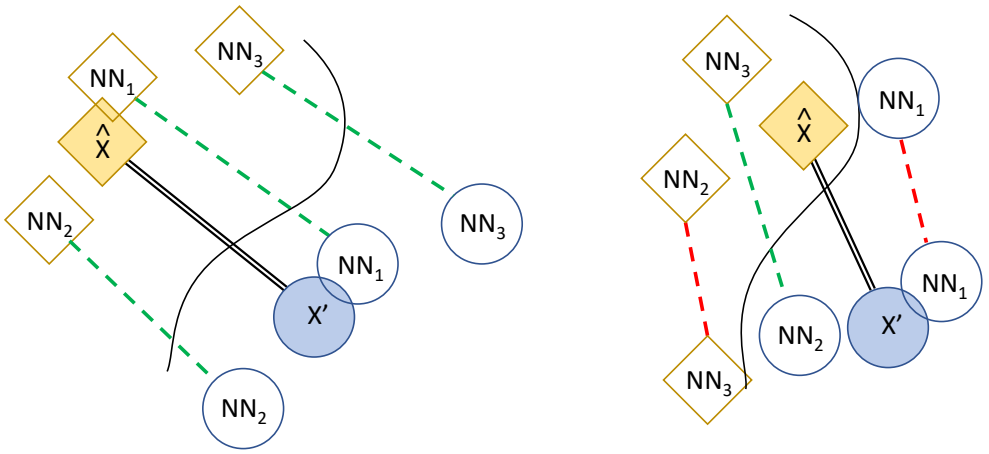


Figure 4.2: Measuring the complexity of pairs using their neighbours, featuring an example of a low complexity pair (Left) and a high complexity pair (Right). Green links represent neighbour relationships which match the original pair relationship, while red links represent neighbour relationships which do not match the original pair.

3. **DynE**: Pairs are unordered, but pair are updated (hence dynamic) using the exploration algorithm throughout training.
4. **DynE***: As DynE but now with pairs ordered.
5. **DynEE**: Pairs are unordered, and are updated (hence dynamic) using the explore and exploit algorithm according to some β ratio.
6. **DynEE***: As DynEE but now with pairs ordered.

We use the '*' postfix to indicate complexity-based ordering over those that have no ordering. We evaluate these algorithms using two different criteria. First, we perform a one-tail t-test to establish statistical significance at a confidence level of 95% on classification accuracy from network output on image classification, sentiment analysis and HAR tasks. Secondly, we examine each algorithm's capacity to learn over time by analysing averaged accuracy on each test set for increasing number of training epochs.

Datasets

Four datasets were used in our evaluation: MNIST, IMDB, SelfBACK-Thigh and SelfBACK-Wrist. For each dataset we divided available examples into the training and test set as identified by their documentation. Then we extracted a budget from the training set, which acted as our total budget for training (i.e. while the training set could be paired in any way, we could not use any examples outwith the budget). We then tested on the full test set for each dataset. In such a manner our evaluation emulated budgeted learning.

For MNIST, we split dataset into 60,000 training and 10,000 test images. We allocated a budget of 1% of the full training set (600 images) and tested on the full test set. For IMDB we adopted a budget of 10% of the training set (2,500 reviews) and tested against the full test set (25,000). Reviews were preprocessed before submission to the network using Word2Vec [46] and averaging the word vectors to form a document vector [50]. This resulted in a single movie review being represented as a vector of 300 features. Within the SelfBACK dataset, for our experiments we remove any users that have less than 60 seconds of recorded data per activity, leaving 24 users. Data was split into 3 second windows, with 900 features per example. Our goal in the SelfBACK dataset is to classify user activity based on minimal information pertaining to them. This is important for personalised HAR model generation to minimise demand on the user for labels [130]. Data is split into training and test sets within each user so that our training set consists of 4 windows (12 seconds) of data for each activity and the test set is the remaining data.

In MNIST and IMDB we performed our evaluation using a 10-fold cross-validation design, while in SelfBACK we divide each user into their own train and test set and average the results from all users. Each algorithm is therefore compared based upon the same initial sample as taken from the dataset, though the way in which this is exploited to form training pairs differ between algorithms.

Network Architecture

For the MNIST and IMDB datasets we used a 3-layer perceptron with a batch size of 16 for each of our sub-networks. We then trained these architectures for 100 epochs. For the SelfBACK dataset, we used a 5-layer convolutional network (as in [27]) with a batch size of 8 for each sub-network. This architecture was trained for 50 epochs to prevent overfitting on the smaller dataset. All architectures used ReLU activations and computations for test classification adopt Euclidean distance on feature embeddings at the similarity layer. Table 4.1 provides information on network hyperparameters.

We found that increasing regularization by decreasing batch size improves convergence speed on all methods, likely due to the limited number of examples. Decreasing the batch size gives DynEE and DynEE* flexibility to extract a relevant exploitation set, and offers more opportunity to update network weights appropriately.

	Sub-network (layers)	Total epochs	Exploitation ratio β
MNIST	MLP (3 Dense)	100	$^{\lvert P\rvert}/_{6}$
IMDB	MLP (3 Dense)	100	$^{\lvert P\rvert}/_{10}$
SelfBACK	Convolutional (3 Conv., 2 Dense)	50	$^{\lvert P\rvert}/_{4}$

Table 4.1: Summary of relevant network hyperparameters

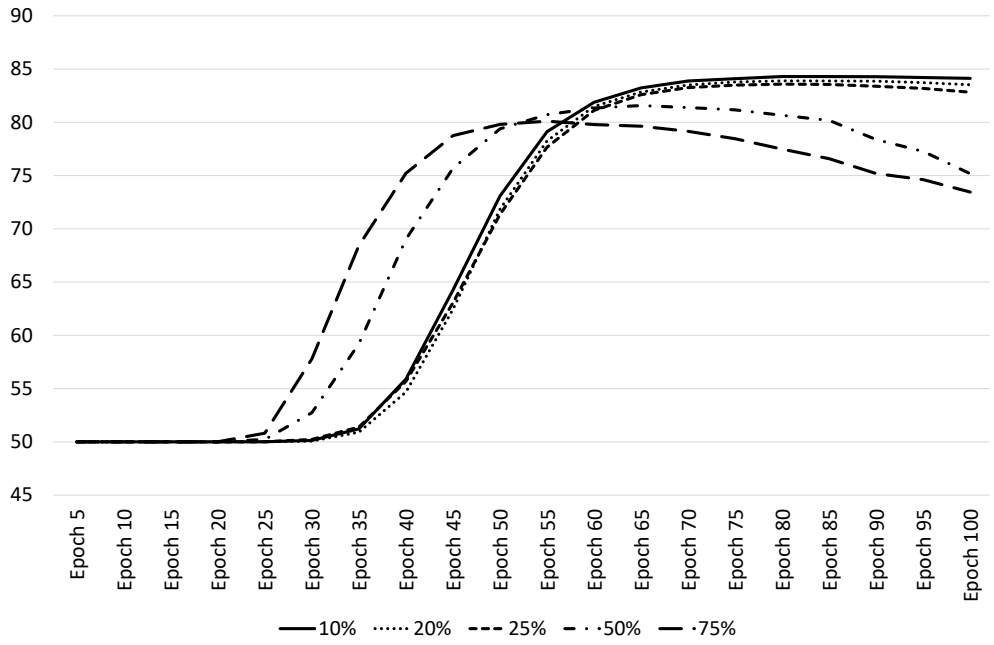


Figure 4.3: Demonstration of increasing levels of exploitation on IMDB dataset.

Ratio of Exploration to Exploitation

We experimented to understand the impact of the ratio of exploration to exploitation, β, for each dataset. As identified above, over-exploiting a given dataset can cause overfitting and have negative effects on accuracy. We therefore sampled various β to determine the optimum for each dataset. We observed that high levels of exploitation are beneficial at the start of training, but cause overfitting towards the end. Intuitively, this suggests that exploitation would function optimally as a decay parameter - something we will explore in future work (see Section 7.1).

We established optimal β for each dataset: $^{\lvert P\rvert}/_{6}$ for MNIST, $^{\lvert P\rvert}/_{10}$ for IMDB and $^{\lvert P\rvert}/_{4}$ for both SelfBACK datasets. This means that for DynEE and DynEE*, at every training cycle, this proportion of the pairs were generated based upon exploiting knowledge from the previous training iterations and the rest of the pairs were sampled according to the explore strategy. We repeat the pair selection process every five epochs

for DynE, DynE*, DynEE and DynEE*.

4.2.3 Results

Our results demonstrate that using a pairing strategy will improve network performance on all of the investigated datasets (see Table 4.2 and Figure 4.4). On every dataset using either DynE or DynEE boasts faster convergence and greater accuracy than the Base method. In one instance (MNIST), DynEE achieves a statistically significant higher optima than any other method. Though complexity-based ordering offers mixed results, we observe that an ordered method (DynE*) ultimately achieves the greatest accuracy on three of the four compared datasets. The results for each dataset appear in Table 4.2 with bold font used to indicate maximum accuracy for a dataset and asterisks indicating statistical significance with 0.95 confidence.

			Our Contributions				
	Training	Base	Base*	DynE	DynE*	DynEE	DynEE*
MNIST	25%	57.50	29.19	61.05	27.04	67.00	29.37
	50%	68.24	52.96	75.16	59.24	81.93*	69.09
	75%	72.03	63.47	80.76	72.78	86.78*	83.12
	100%	75.51	70.74	84.53	82.21	**90.05***	88.92
IMDB	25%	50.28	50.00	50.00	50.18	50.01	50.16
	50%	72.35	78.11	76.87	81.08*	73.08	79.52
	75%	83.66	83.56	84.36	**84.55**	84.10	84.54
	100%	83.95	83.32	84.30	84.26	84.13	84.13
SB - Wrist	20%	31.70	40.92*	31.19	28.62	33.38	31.18
	50%	58.25	64.82*	62.37	60.46	61.54	57.46
	70%	62.39	67.07	66.02	64.58	66.07	65.61
	100%	67.32	68.71	68.72	**70.49**	69.22	67.43
SB - Thigh	20%	40.79	44.46	43.12	43.76	45.31	46.03
	50%	60.91	61.24	60.78	61.30	58.11	58.24
	70%	66.31	66.84	67.59	69.37	65.71	63.71
	100%	71.81	73.89	73.46	**75.20**	74.39	70.63

Table 4.2: Summary of algorithm performance throughout training

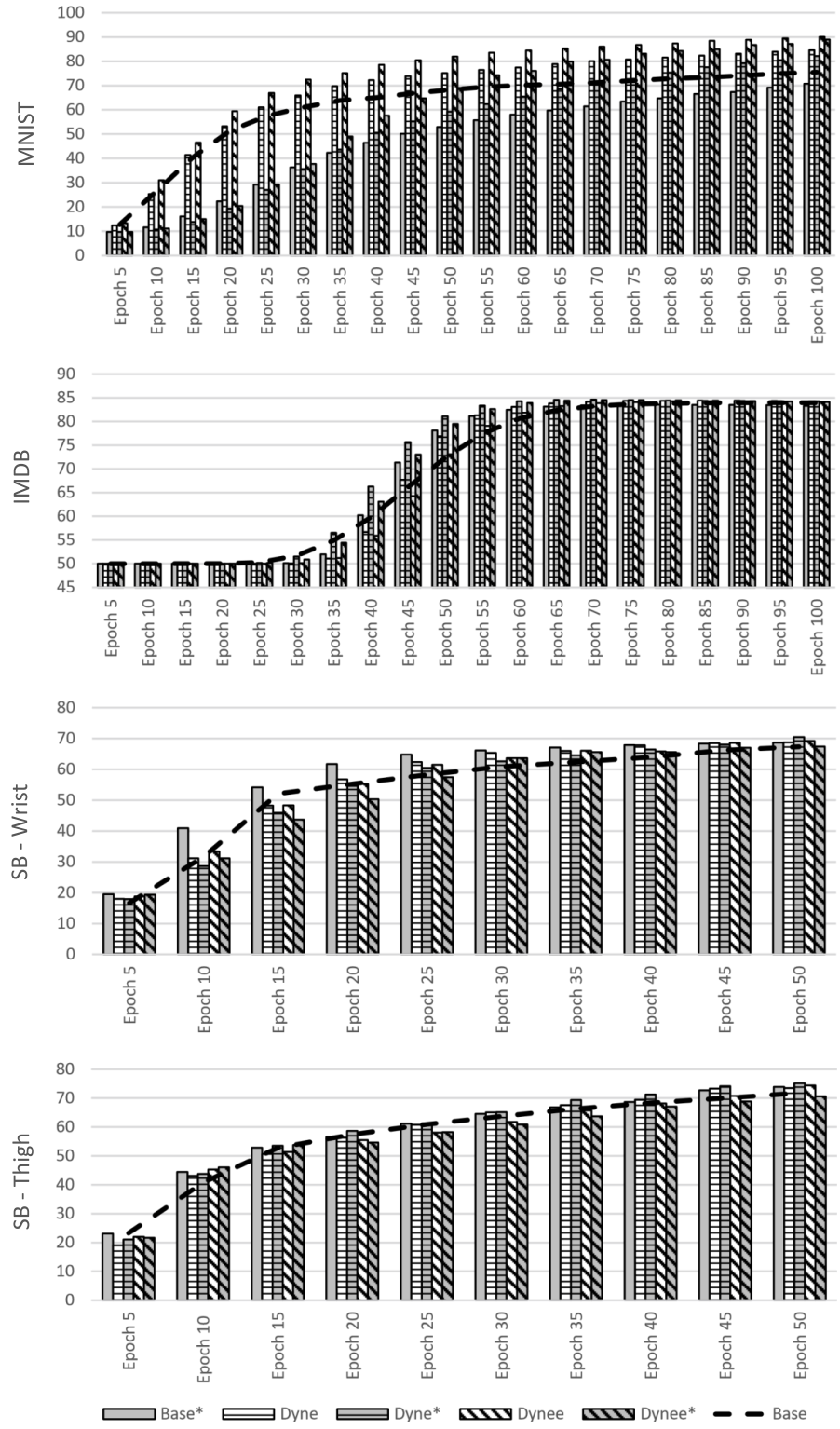


Figure 4.4: Results on the MNIST, IMDB and SelfBACK datasets

Dynamic Informed Pair Selection

On MNIST, we can distinguish that both DynEE and DynE begin to outperform Base from as little as 15 and 20 epochs respectively. The difference between DynEE and Base is statistically significant from epoch 40 onward. In IMDB we observe that DynE and DynEE reach superior performance to that of Base with only 60% and 70% of the training epochs required. Similarly, DynE and DynEE demonstrate faster convergence to optima on the SelfBACK-Wrist dataset, though not on SelfBACK-Thigh. On both SelfBACK datasets, the proposed methods ultimately converge to a higher optima than Base can achieve.

These results suggest that the informed explorative and exploitative pair selection strategies present greater insight into the space than can be achieved through the static pairing method of the baseline. They support our hypothesis that a suitable training strategy can improve the performance of SNNs.

Heuristic Ordering for Self-Paced Learning

Ordering proves most effective in the IMDB dataset, where all ordered methods significantly outperform their unordered counterparts. Both DynE* and DynEE* reach new optima at 65% and 70% of the training epochs required for Base to converge. In MNIST, all ordered methods performed comparably worse than their unordered counterparts. However, only Base* underperforms the baseline and both DynEE* and DynE* outperform Base from epochs 55 and 80 onwards respectively. On SelfBACK-Wrist, Base* demonstrates statistically significant improvements over the Base method until epoch 30 and (though DynE* obtains the best accuracy), converges faster than other methods. On SelfBACK-Thigh, DynE* consistently outperforms the Base method from very early in training and ultimately achives a much superior accuracy.

We take these results as evidence that curriculum learning is effective at dealing with noisy or complex datasets. MNIST has non-complex class boundaries, suggesting that ordering will be less effective. However, both SelfBACK datasets have considerable noise, specifically the wrist dataset, as accelerometers on wrist lead to greater degrees of freedom in movement and hence is more noisy than the thigh. Additionally we observe that ordering performs well on IMDB, which is a sentiment analysis dataset. Different people can have different thresholds for enjoyment and different preferences, meaning they can describe the exact same event with different sentiments. This means there can be considerable complexity at the boundary between positive and negative classes.

In summary, the results demonstrate that a pair selection strategy will improve the performance of an SNN. In every tested dataset, the use of a non-ordered pairing strategy offers faster convergence and enables greater accuracy to be achieved, though selecting the most effective strategy depends on the task. In one domain DynEE even allows the SNN architecture to significantly outperform the baseline in terms of classification accuracy. Furthermore, although ordering is not effective in every scenario, incorporating a sample ordering based upon neighbourhood-complexity analysis can speed up convergence on noisy or complex datasets. Specifically, DynE* operates very effectively in these situations. This is likely because it is performing an ordered exploration of the feature space, which allows it to gradually improve its knowledge to better deal with complex areas.

4.3 Reflections

We have demonstrated the need for a reasoned training strategy for SNNs. We have presented four contributions towards addressing this need, highlighting the importance of a pair selection strategy and noting both an explorative algorithm, explore-exploit algorithm and an ordering method for pairs and have shown on four different datasets that each is suitable in different scenarios. Namely, informed pair selection offers a greater view on the problem space when needed and ordering allows a network to develop a structured training regime which is robust to dataset complexity. Thus we conclude that use of an appropriate pairing strategy will improve network performance, though selecting an optimal strategy is task dependant. Furthermore our results are indicative that a pairing strategy which has its roots in similarity comparisons will be more effective than random pairing. Thus the intuition behind using similarity to inform our thinking seems to be well-founded, and implies that our hypothesis for this chapter is supported.

However, the suggested method has a limitation in the form of its computational complexity, which is prohibitive to its use on large or complex datasets. Let us explore this more deeply by considering the three steps of DynEE (and the fourth step of DynEE*) using a simple complexity analysis:

- Sort the output of the previous epoch based on the loss generated by each pair. Sorting can vary in complexity based on the method used (quicksort, mergesort, heapsort, etc) so in theory this step could be optimized. However, an ideal solution would be to prevent the need for sorting at all.
- Extract the exploitation set and find the most similar pair to each previously

difficult pair. This is the most expensive step in the process as kNN has a computation complexity of $O(nd + kn)$, where n is the size of the set of examples (in this case, the number of examples in the exploit set), d is the operations required for a single distance calculation and k is the number of neighbours. This is because calculating the distance between a query and all items in the training set has a complexity of $O(nd)$, while looping through all items in the training set to return the neighbour set has a complexity of $O(kn)$. In our implementation, because we are only actually interested in identifying the single nearest neighbour (i.e. $k = 1$) the complexity is actually $O(nd+n)$. However we perform this calculation twice (once for each member of the pair, x' and $\hat{x}$ respectively), giving an overall complexity of $O(2(nd + n))$. This is the most expensive step in the algorithm, and the main impediment from expanding to larger datasets. More than that however, it is also an obstacle to applying the algorithm to other deep metric learners. In triplet networks for example, the complexity of the algorithm would become $O(3(nd + n))$, as each member of the triplet would need to be replaced. We can represent this as s, as indicative of the number of samples input to the neural network simultaneously. This gives an overall complexity of $O(s(nd+n))$.

- Create the explore set. This is the cheapest step in the DynEE algorithm, as the complexity of selecting a random partner for each case in our explore set is less than $O(n^2(2(\log_n)))$. Firstly, every case in the explore set must be iterated over, giving a complexity of $O(n)$. For each item, the explore list must be filtered using a Boolean comparison to create the matching and non-matching sets, so the complexity becomes $O(n^2)$. Finally, from each list we select a random partner for the case, which has a complexity of less than $O(2(\log_n))$. It is difficult to surmise the exact complexity, as the size of the matching and non-matching sets are dependent on the distribution of the dataset. However, we know that the number of examples in both sets cannot sum to greater than n. Thus the overall complexity is guaranteed to be less than $O(n^2(2(\log_n)))$. Though this may seem expensive, it is actually the cheapest method of creating a set of pairs, as this complexity would correspond exactly to the baseline for creating random pairs.

- (DynEE* only) Identify the curriculum. In the case of DynEE*, an additional step of the algorithm would require the calculation of loss contribution of each member of the combined explore and exploit sets before passing this to a sorting algorithm. The contribution is particularly expensive to identify, as it requires each pair to be fed through the network to receive the new representation before an accurate loss score could be calculated. The complexity to identify loss contribution is $O(2ph)$, where p is equal to the number of pairs in the training

set and h is equal to the number of operations required for the transformations performed by parameters of the neural network. The pairs and their associated scores could then be sent to a sorting algorithm which will provide an additional level of complexity (though as stated above, this could be optimized).

The total complexity of DynEE and DynEE* is therefore:

$$O(\text{DynEE}) = O(sort + s(nd + n) + n^2(2(\log_n))) \tag{4.5}$$

$$O(\text{DynEE*}) = O(sort + s(nd + n) + n^2(2(\log_n)) + 2ph + sort) \tag{4.6}$$

Where *sort* is representative of the complexity for the sorting algorithm which is used.

From our complexity analysis of DynEE, it is apparent that there are several areas of where computational efficiency could be improved. In particular, it would be desirable to reduce the expensive nearest neighbour calculation which prevents upscale of the algorithm to other deep metric learners with larger simultaneous input (such as triplets or subsets). Additionally, if possible the removal of the initial ranking and sorting of the list to identify the exploitation set should be avoided.

Our investigation into methods to improve the efficiency of the similarity knowledge container in CBR show potential to answer these limitations. In particular, we highlight the approximate-Nearest Neighbour (a-NN) methods which are useful to reduce the cost of brute force nearest neighbour calculations. In the following section, we will discuss how have been inspired by research in CBR to adapt an a-NN method as an inexpensive training strategy for triplet networks.

4.4 Triplet Networks

Triplet networks are deep metric learners which learn to optimise a feature space using similarity knowledge gained from training on triplets of data simultaneously. The architecture relies on the triplet loss function to optimise its weights based upon the distance between triplet members. Composition of input triplets therefore directly impacts the quality of the learned representations, meaning that a training scheme which optimises their formation is crucial. However, an exhaustive search for the best triplets is prohibitive unless the search for triplets is confined to smaller training regions or batches. Accordingly, current triplet mining approaches use informed selection applied only to a random minibatch, but the resulting view fails to exploit areas of complexity in the feature space.

In Section 2.2.3 we discussed the different aspects to consider when developing a training strategy for deep metric learners. To summarise our discussions, often these training strategies make use of random minibatches extracted from the training set to offset the complexity of utilising the full set. For example, in [18], the authors 'mine' optimal triplets for network training from within this minibatch by identifying what they describe as semi-hard combinations - i.e. triplets which produce sufficiently large loss to improve weight formation without causing oscillation. Though mining triplets from minibatches offers reduced complexity to methods which target the full training set, it has a key disadvantage. While random minibatches allow an overview of the distribution of the training set, they offer no additional measures to target complex areas such as class boundaries. This is particularly important for triplet networks, because (as with other deep metric learners) their loss is distance-based. With that and our hypothesis for this chapter in mind, in this thesis we suggest convergence of DML architectures can be achieved faster by considering the locality of examples to inform the creation of minibatches.

In this section we highlight the importance of optimising batch selection before triplet mining approaches are applied. To this end, we propose a novel algorithm, Locality-Sensitive Batching (LSB), which uses locality sensitive methods to focus on example clusters as a substitute for random minibatches for a starting point of further triplet mining. This method can provide the necessary focus on complex class boundary areas to improve training efficiency. Furthermore, as indicated by research in curriculum learning [95], in particular self-paced learning [104], it would be desirable for any batch selection method to be relevant to the network's current parameter set. Therefore the method proposed in this chapter uses the latest network output to inform its clustering. The result is that the batches of input data created by LSB are based upon an up-to-date representation of the latent space, and as such faithfully represent current 'difficult' potential triplets for network input. Though locality-sensitive methods can be more expensive than random minibatching, this can be offset by adopting Approximate-Nearest Neighbour (a-NN) methods. In this work we suggest Locality Sensitive Hashing (LSH).

Our findings demonstrate that different batching strategies offer different insights into the space. Training on the full space using a brute force method allows a triplet network to understand the entire distribution of examples, but using the extent of available knowledge quickly becomes expensive. Minibatched strategies provide a randomly sampled overview of the space, but omit potentially useful information about complex areas. On the other hand locality-sensitive batches offer comprehensive focus on a region in the space. However one needs to be aware that focusing in this manner

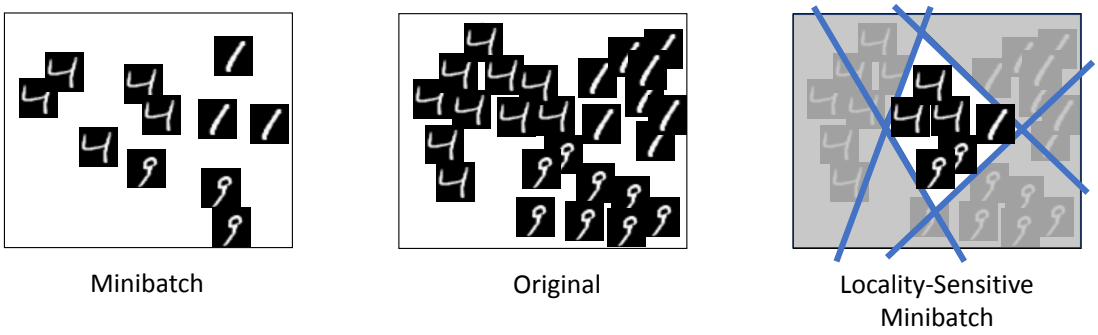


Figure 4.5: Visualisation of the different insights offered into the feature space by different training schemes. Left (Minibatch) is created by randomly sampling the original distribution. Right (Locality-Sensitive Minibatch) is created by applying LSB to the original distribution.

can be detrimental unless an understanding of the full space is also maintained (see Figure 4.5).

4.4.1 Training with Triplets in a TN

Triplet networks are deep metric learners which learn from three input examples simultaneously. These inputs are the anchor example (x^a), a positive example (x^+) and a negative example (x^-), which together are described as a triplet. The anchor example acts as a point of comparison, meaning that the positive and negative examples are dictated by their relationship to the anchor (i.e. matching and not matching respectively). The goal of training is to create a space optimised for similarity-based return by minimising the distance between an anchor and its associated positive example while maximising the distance between an anchor and its associated negative example.

A triplet network is comprised of three identical 'sub-networks' (see Figure 4.6). Typically a deep learning architecture, which can be as shallow or as deep as necessary. Each sub-network creates an embedding for one input (i.e. an individual member of the triplet) before the error is calculated using triplet loss.

Let us summarise the notation used in this section. For a detailed breakdown, please see Table 3.2. Let $\mathcal{X}$ be a set of labeled examples, such that example, $x \in \mathcal{X}$ and $y(x)$ is a function that returns the class label, y, of x. In the context of this chapter, we will define matching examples as those which have the same class ($y(x^+) = y(x^a)$) while non-matching examples will have differing classes ($y(x^-) \neq y(x^a)$). The embedding function θ is an appropriate parameterisation of any one of the identical sub-networks creating the vectorised representation of a given x. We can then represent triplet loss

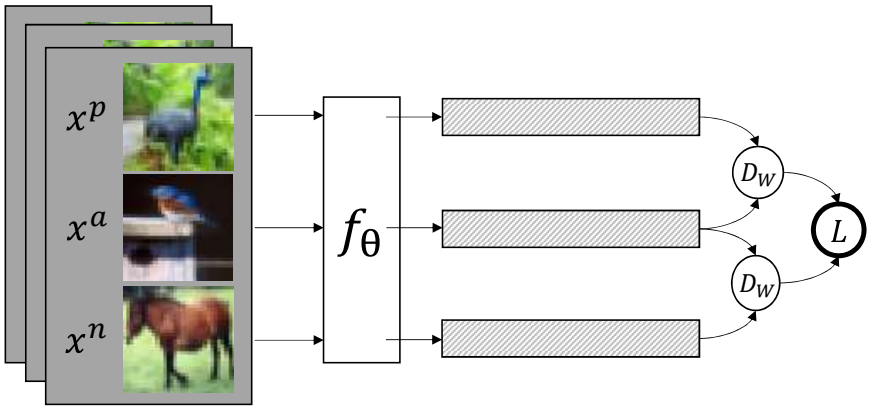


Figure 4.6: Batched triplet network training on the CIFAR10 dataset. The representations learned by each sub-network for each input image are improved over time by using knowledge around the relationship between inputs during training.

L as so:

$$L = max(0, (D_W(\theta(x^a), \theta(x^+)) - D_W(\theta(x^a), \theta(x^-)) + \alpha)) \tag{4.7}$$

Where D_W is a function to calculate the distance between two embeddings and α is the margin which must exist between an anchor and negative example. This formula will generate a loss value in situations where the anchor example is closer to the negative example than it is to the positive example. The network is therefore penalised until similar cases are placed closer together in the feature space. The $max()$ function ensures that only loss values greater than zero impact network weights.

However, there are some issues with this. As the network approaches convergence, random formation of triplets has an increased likelihood to provide triplets which will generate a loss of zero. This is because the feature space will be approaching optima. The result is the network will train for increasing periods of time with decreasing improvements to its weights; hence the importance of sample selection and training optimisation. Simply put, we want to form triplets which will maximise loss for the improvement of the network and allow it to converge towards optima more quickly.

Creating Triplets

Let us first identify a baseline algorithm for randomly creating triplets from the full training set, where $\mathcal{T}()$ is a function to create a triplet.

In Algorithm 5, $is\_matching()$ is a function which separates $\mathcal{X}$ (or any subset) into two sets, $\mathcal{P}os$ and $\mathcal{N}eg$, based on each member's relationship to a given anchor example x^a

Algorithm 5: Create random triplets from full training set.

Random: $\mathcal{X}_{RND}(n)$
for x^a *in* $\mathcal{X}$ **do**
    $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, \mathcal{X})$
    x^+ := rnd_selection($\mathcal{P}os$)
    x^- := rnd_selection($\mathcal{N}eg$)
    $\mathcal{T}_i := \mathcal{T}(x^a, x^+, x^-)$
    $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
end
return $\mathcal{T}$

(see Algorithm 6). Members of $\mathcal{P}os$ have a matching class label with x^a and $\mathcal{N}eg$ have a non-matching class label. Note that this function could be adapted to be non-class reliant (i.e. by altering the if statement in the $is\_matching function$ to consider cluster membership or presence of a particular feature in its comparison of x_a and x_i).

Algorithm 6: Extract the set of matching and non-matching examples from $\mathcal{X}$.

Matching: $is\_matching(x^a, \mathcal{X})$
$\mathcal{P}os = \emptyset$
$\mathcal{N}eg = \emptyset$
for $x_i \ldots \mathcal{X}$ **do**
    if $y(x_i) = y(x^a)$ **then**
        $\mathcal{P}os = \mathcal{P}os \cup x_i$
    else
        $\mathcal{N}eg = \mathcal{N}eg \cup x_i$
end
return $\mathcal{P}os, \mathcal{N}eg$

Algorithm 5 is relatively inexpensive to perform, but as mentioned above, there is no guarantee that the created triplets will result in any loss for the network. This is a problem which gets worse over the course of training as the optimal representation of the space is approached.

To counter these issues, there must be some concept of identifying triplets which are meaningful for training - an informed approach. However, checking every example is too expensive. Hence the importance of extracting minibatches. By examining mini-batches of the data at a time, the complexity of an informed approach is considerably reduced. The question then becomes how best to identify subsets of the data which lend themselves to minibatches. While many active learning approaches are applied on randomly sampled batches to get an overview of the space, based on findings from our literature review (see Section 2.2.3), we suggest that the batching method is an important design consideration, and by selecting the appropriate method significant

improvements can be made.

Creating Triplets from a Random Minibatch (MR)

Hereon, we use the term 'minibatch' to refer to any subset of $\mathcal{X}$. Minibatches which are representative of the original space can be easily created using Algorithm 7. This can be trivially adapted to ensure stratification in cases of class imbalance. In Algorithm 7 m is a minibatch of examples from $\mathcal{X}$, such that $m \subset \mathcal{X}$. $\mathcal{M}$ is then the complete set of minibatches and the function $\mathcal{M}()$ creates a set of minibatches from within $\mathcal{X}$.

Algorithm 7: Develop random triplets from a minibatch.

Random Minibatch: $\mathcal{X}_{MR}(n)$
$\mathcal{M} = \mathcal{M}(\mathcal{X})$
for $m_i \ldots \mathcal{M}$ **do**
  for x^a *in* m_i **do**
    $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, m_i)$
    $x^+ :=$ rnd_selection($\mathcal{P}os$)
    $x^- :=$ rnd_selection($\mathcal{N}eg$)
    $\mathcal{T}_i := \mathcal{T}(x^a, x^+, x^-)$
    $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
  end
end
return $\mathcal{T}$

Though training using minibatches of examples in this way reduces the potential number of triplets (thereby reducing pairwise similarity computations and complexity), it does not provide any focus on complex areas of the feature space. This is because the random selection of the minibatch allows it to be representative of the data distribution as a whole, without allowing any room for specific localised knowledge of the feature space. Clustering methods offer potential to fill this gap, but are difficult to justify due to their high initial resource requirements. In the next section, we will describe how we adopted methods from locality-sensitive hashing to inform the creation of clusters.

Locality-Sensitive Batching (LSB)

It is clear that the greater the loss generated by a given input, the greater its contribution to the network weights (hence the intuition behind [18]). Because triplet networks utilise a distance-based loss, the contribution of a given triplet is decided by the distance between its constituent members. With that in mind, the most appropriate triplet for a given anchor is likely to exist within the same locality. In this section, we detail how we adapt Locality-Sensitive Hashing (LSH) to develop locality-sensitive batches for network training.

In the literature, LSH is often used to reduce complexity for similarity comparisons (see Section 3.1.2). However, each bucket can also effectively be considered as a cluster. We have found that the these clusters offer a good alternative form of batch selection to random minibatching with only trivial adaptation (see Lines 2-18 of Algorithm 8). Furthermore, we ensure that the information from clustering is up to date by basing our locality informed clusters on the latest network output, θ. The intuition is that this will allow the network to maintain focus on complex areas which are most relevant to its current parameters. Hence, on the first epoch of the network the input data is split into batches by using LSH on the original data representation (i.e. $LSH(\mathcal{X})$), while in all subsequent epochs input data is batched by using LSH on the network output (i.e. $LSH(\theta(\mathcal{X}))$). We make this distinction as the network is initialised with random weights and so performing LSH on its output before any training has occurred would not produce meaningful batches.

In Algorithm 8, LSH is a function to extract a set of locality-sensitive buckets from $\mathcal{X}$ and b is an individual bucket from within $\mathcal{B}$, such that $b \in \mathcal{B}$. Finally, $pure()$ is a function which returns True if the selected bucket contains only a single class (or False otherwise) and R is an empty set which is eventually populated with anchor cases which exist in pure clusters or as the sole member of a cluster. Naturally, as the network converges we expect the number of impure buckets to decrease (see Figure 4.7).

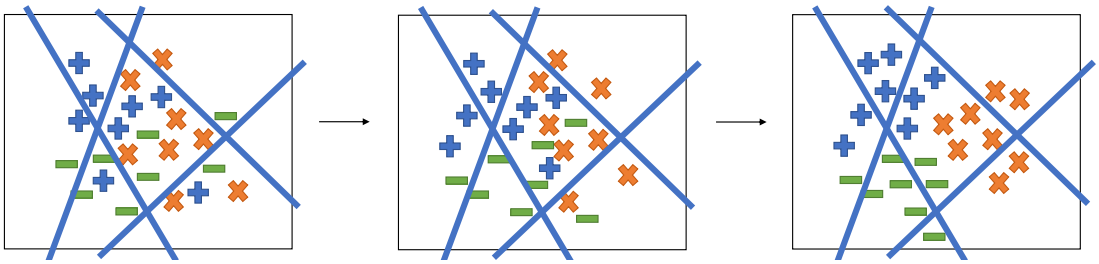


Figure 4.7: Distribution of examples in buckets throughout training. As training progresses the number of impure buckets decreases.

As buckets are created using locality knowledge, feeding triplets into the network can enforce sequential learning. This in turn can be problematic, because the implicit curriculum could be non-optimal. With that in mind, we need to randomise the order of the triplets to allow an understanding of the overall distribution of the space. We do this in two ways. Firstly, if we fail to identify a cluster for a given example (or if the cluster identified is 'pure'), we randomly combine it with other examples where a cluster could not be identified to create a triplet. Secondly, we input the triplets we have gained from our buckets to the network in a random order (see Lines 19-25 of Algorithm 8).

Algorithm 8: Develop random triplets from a bucket.

Locality-Sensitive Batching: $\mathcal{X}_{LSB}(n)$
if *epoch = 1* **then**
  $\mathcal{B} = \text{LSH}(\mathcal{X})$
else
  $\mathcal{B} = LSH(\theta(\mathcal{X}))$
$R = \emptyset$
for $b_i \ldots \mathcal{B}$ **do**
  if *not pure(b_i)* **then**
    for x^a *in* b_i **do**
      $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, b_i)$
      $x^+ :=$ rnd_selection($\mathcal{P}os$)
      $x^- :=$ rnd_selection($\mathcal{N}eg$)
      $\mathcal{T}_i := \mathcal{T}(x^a, x^+, x^-)$
      $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
    end
  else
    $R := R \cup b_i$
end
for $x^a \ldots R$ **do**
  $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, R)$
  $x^+ :=$ rnd_selection($\mathcal{P}os$)
  $x^- :=$ rnd_selection($\mathcal{N}eg$)
  $\mathcal{T}_i := \mathcal{T}(x^a, x^+, x^-)$
  $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
end
$shuffle(\mathcal{T})$
return $\mathcal{T}$

4.4.2 Evaluation

In this section, we offer details of our evaluation of the proposed method. We perform an empirical comparison of the representations gained from each training scheme, using k-NN accuracy as a proxy for representation goodness. We evaluate this using two different criteria. Firstly we perform a one-tail t-test to establish statistical significance at a confidence level of 95% on classification accuracy from network output. Secondly, we examine each algorithms' capacity to learn over time by comparing averaged accuracy on each test set for increasing number of training epochs. This is important because improvements that LSB offers are likely to be in the form of training efficiency.

The goal of these experiments is to analyse the difference that each batching method offers as a starting point for further active learning approaches. Accordingly we identify two batching strategies for comparison:

Dataset	Sub-Network (Layers)	Epochs	Minibatches	Batch Size	Projections
MNIST	MLP (3 Dense)	10	1000	50	18
CIFAR10	CNN (4 Conv., 2 Dense)	100	1000	40	18
STL-10	CNN (4 Conv., 2 Dense)	100	250	40	6
IMDB	MLP (3 Dense)	20	1000	40	12
REUTERS	MLP (2 Dense)	10	90	138	5

Table 4.3: Summary of relevant network hyperparameters

1. **Minibatched Random** (MR): Triplets are randomly generated from within a minibatch of the training set. Minibatches are distinct and contain non-overlapping examples. This algorithm will act as our baseline for comparison.

2. **LSB Random** (LSB): Triplets are randomly generated from within a local neighbourhood of each anchor case in the training set. These neighbourhoods are distinct and contain non-overlapping examples. Anchors which have a 'pure' neighbourhood are randomly combined to create triplets.

In both instances, we ensure that every example in the training set is utlised as an anchor only once per training epoch. This means that there are as many triplets per training epoch as there are examples in the training set. Classification is performed using k-NN, where $k = 3$ and similarity is measured using cosine similarity.

Network Architecture

All architectures used ReLU activations and the Adam optimizer [131] and produced an output representation of the size 128. For all other variables, including number of batches for networks using MR and number of projections for networks using LSB, we implemented an empirical evaluation to identify the best performing hyperparameters for each dataset (see Table 4.3). Note that since projections in LSH are random, the number of buckets can vary between runs. This is because there is potential to create empty buckets which are discarded. Therefore, it is more suitable to maintain the number of projections as constant. Batch sizes were set such that they were a multiple of the number of labels contained in each dataset (i.e in MNIST there are 10 classes, so the batch size was a multiple of 10).

Datasets

We have used several datasets across different problem domains to provide a robust evaluation of the versatility and utility of LSB. The reduced complexity of LSB means

Classification	Dataset	Method	Accuracy throughout Training (%)			
			25%	50%	75%	100%
Image	MNIST	MR	95.98	96.66	96.95	96.99
		LSB	95.81	96.93	97.19	**97.40***
	CIFAR10	MR	57.81	64.41	66.55	66.68
		LSB	62.65*	66.68*	68.44*	**69.02***
	STL-10	MR	50.26	57.79	60.76	**61.67**
		LSB	50.97	61.16*	61.51*	61.63
Text	IMDB	MR	85.86	87.39	87.90	88.04
		LSB	87.75*	87.80	88.30*	**88.33**
	REUTERS	MR	74.42	75.01	76.02	76.47
		LSB	77.13*	78.03*	78.32*	**78.68***

Table 4.4: Summary of algorithm performance throughout training

that it is viable to utilise the entire training set during training, and for us to evaluate on larger and more complex datasets than was previously computationally viable when using the DynEE strategy. We selected 3 popular image classification datasets from the literature (MNIST, CIFAR10 and STL-10). These datasets were selected because of the triplet network's utility in image-based search and to demonstrate our algorithm as applicable in this domain. We did not use data augmentation in any case, as our goal was merely to compare the two batching methods. We also selected two text classification datasets (IMDB and Reuters) to investigate the capability of triplet networks for our intended use case of experience transfer for telecommunication engineers, as this relies upon text data as input.

Text datasets were pre-processed using the Keras library, as was also used for implementing the deep learning architectures throughout this thesis. The vocabulary for each dataset used the 5,000 most common words in the case of IMDB and the 1,000 most common words in the case of Reuters. Out of vocabulary words were discarded. Each example in the IMDB dataset was converted to a vector using tf-idf, while the examples in the Reuters dataset were converted to binary vectors. All parameters were set following an initial empirical evaluation. In all situations, 5-fold cross-validation was used to create distinct train and test sets. This was because we were not aiming for state-of-the-art, but to empirically demonstrate that different batching methods can impact network performance to a statistically significant degree.

4.4.3 Results

The results for each dataset appear in Table 4.4 with bold font used to indicate the highest achieved accuracy for a dataset and asterisks indicating performance which is better than the baseline with statistical significance at 95% confidence. As can be observed, LSB outperforms the baseline on all tested datasets with significance for at least some portion of training. On 3 of the 5 tested datasets (MNIST, CIFAR10 and Reuters), LSB achieves a statistically significant improvement on accuracy at the end of training. On every tested dataset LSB converges to optima faster, approximating the baseline performance with only 50% of the required training or less.

On MNIST, though differences are less pronounced during early training, our approach does converge to a statistically significant higher accuracy. The advantages of LSB can be seen on CIFAR10, where it outperforms MR from very early in training, achieving performance improvements that are statistically significant from 10% of training onwards. Though it would seem from Table 4.4 that the baseline for STL-10 converges to the same accuracy as LSB, we actually converge to optima much earlier in training. By 50 epochs, the accuracy achieved by LSB is already at 61.16%, which is a 4% improvement over MR at the same number of epochs. On the IMDB dataset, we observe that LSB has great benefits very early in training, with less improvements as time goes on. This is because LSB can focus on the complex boundary cases that are difficult to classify. The similar performance achieved is indicative of the difficulty to wholly separate the positive and negative viewpoints in this task. It is interesting to note the superior performance of LSB on the Reuters dataset. MR struggles when faced with many classes or imbalanced data. This is not a problem for LSB, as the method is only concerned with a small neighbourhood of the space. Thus we suggest these results seem indicative that LSB is more suitable in problems with class imbalance.

4.5 Conclusions

To conclude, in this chapter we presented work towards confirming our hypothesis that techniques from traditional machine learning, such as meta-learning and CBR, could improve the training efficiency of DMLs. To this end, we presented contributions through the development of several new training strategies from DMLs, based around exploring and exploiting knowledge of the feature space (DynE and DynEE), heuristic ordering of pairs (Base*, DynE* and DynEE*) and locality-sensitive batching (LSB).

Our initial study focused on the concepts of exploration and exploitation, as popularised by boosting in meta-learning, to develop several training schemes for SNNs. Here we introduced the DynE and DynEE algorithms, where the former explored the

space to identify new pairings to inform network training, and the latter exploited areas of known difficulty to augment this exploration. We also identified a pair heuristic ordering approach that was applicable to both algorithms, based on the concept of curriculum learning. Results of our evaluation of these algorithms demonstrated promising improvements to training efficiency, but highlighted that the approaches were prohibitively expensive to adapt to more complex DML approaches. To answer these limitations, we introduced a locality-sensitive minibatching method, LSB, which utilises locality information to inform selection of minibatches for training a triplet network. The networks trained using LSB obtained better accuracy than random minibatching methods on our evaluation task, suggesting that locality-sensitive minibatches are a better starting point for further active learning approaches. The value added by LSB is a direct indication of how techniques from CBR can improve the efficiency of training DMLs.

In both of the described circumstances, our evaluation utilised knowledge from multiple domains captured in public datasets, and demonstrated that the proposed strategies often improved the training efficiency of the DMLs on which they were applied. In particular, we highlight that the improvements presented by LSB were statistically significant compared to the baseline at some point in training for all tested datasets. Therefore, we consider that the work we have demonstrated in this chapter supports our hypothesis, though we acknowledge that further study would be required to understand whether similar impact could be observed in other domains and datasets.

Chapter 5

Similarity Knowledge for Transfer of Experience

In the previous chapter, we explored the use of similarity to optimise the training of Deep Metric Learners. In this chapter, we build upon our work by applying DMLs as a component of a CBR system on two real-world use cases. These use cases focus on leveraging textual records of experience to provide decision support in complex work domains. Sufficiently understanding the similarity between work elements in these sectors presents an opportunity to improve transfer of experiential content [4] and provide services such as work recommendation [23]. Within the context of this thesis, we highlight field provision of services for telecommunication organisations as a domain to explore this area.

Service provisioning for telecommunication organisations can be very broadly defined as the "making available of resources necessary for a service by allocating those resources in a carrier's network" [132]. In practicality, this means ensuring that the various network components necessary for a customer to receive connectivity are available and operational. This can be a challenging task, as the nature of telecommunications means that the infrastructure for maintaining the network are normally dispersed throughout the service delivery area (which is typically a large geographical region, such as a country). For a telecommunication organisation, it is therefore necessary to have a workforce dedicated to the installation and upkeep of network components in the field to ensure continuous service delivery to customers. It is common within complex services provisioning that the personnel which fulfill this required technical work gradually become highly skilled in their domain. In this thesis specifically, we highlight the telecommunications engineering force whom develop expertise in network equipment installation and repair. To ensure continuous service delivery, they traditionally are

allocated tasks. A task, in this scenario, represents either a time-constrained action to perform on a piece of equipment or an investigation regarding a delayed step of a wider network process. Field engineers record information about the tasks they have completed in free text documents called "notes". These notes form an heterogeneous base of mixed types of information, such as work order, identified problem, failure reason, task progression, task context, and sometimes informal recommendation.

This work is motivated by the need to learn similarity from a user's perspective. We believe that using notes written by engineers themselves as the information source for a similarity metric will ensure that the cases retrieved through similarity-based return are more representative of this point of view. In essence, we wish to achieve a similarity model which is indicative of what a domain expert's own experiences have lead them to believe is the truth. We believe this can be achieved by basing it on experts' notations regarding the subject. Beyond this however, the notes offer potential as a multi-faceted source which can inform a number of decision support systems. The notes are a large semi-structured source of information detailing specific experiences of human experts in the field. Thus, we view this as an opportunity to develop a corporate memory of human experience, improving the effectiveness of engineers in the field and enabling business robustness to the departure of experts from employment.

However, as we have intimated previously, understanding how to represent and compare records of human experience is traditionally a difficult task. Cases using hand-crafted features or a knowledge model solicited from domain experts would be costly to produce and would require wide-scale user involvement to ensure that they were reflective of the majority of opinion. Therefore we have investigated methods of learning representations for textual records. It is our intuition that the representations learned by DMLs should be particularly suitable for this task, as they optimise the learned representation for similarity-based return (as we have explored in Chapter 3, and demonstrated in Chapter 4), which is a key aspect of this task. In this chapter we demonstrate the utility of two case-based retrieval systems which incorporate DMLs in their vocabulary and similarity knowledge containers on two industrial use cases within the domain of decision support for telecommunication engineers. It is our intention that this highlights the real-world application of the research in this thesis.

This chapter is split into the following sections. In Section 5.1 we highlight the primary contribution of this chapter and discuss how it is broken down into a number of secondary contributions. In Section 5.2 we provide an introduction to the problem domain of service provisioning for telecommunication organisations. We use this to structure the description of each of our use cases, in Sections 5.3 and 5.4 respectively. Each use case subsection is further divided into a description of the problem, details

of the evaluation methodology and evidence of the deployed applications. We offer some conclusions and highlight why we believe this work supports our hypothesis in Section 5.5.

5.1 Contributions

In this chapter we explore the hypothesis that DMLs present an opportunity to combine the similarity and vocabulary knowledge containers as a component of a CBR system. Our collaboration with an industrial research partner allows us the opportunity to explore this hypothesis on real-world data captured by domain experts during the daily workings of a telecommunications organisation. Therefore, in this chapter we aim to answer the research question:

- How effective are DMLs at fulfilling the traditionally separate roles of the 'vocabulary' and 'similarity' knowledge containers in the context of transfer of experience between experts and non-experts of telecommunications engineering?

Through our work towards answering this research question, we highlight the primary contribution of this chapter. We compare methods of developing a similarity model for transfer of experience using free-text data sources. Our findings demonstrate that DMLs can learn to produce representations optimised for similarity calculations which offer clear improvement over dense representations gained from word embeddings, but require refinement to outperform statistical methods. To this end, we offer several secondary contributions:

1. Firstly, we examine our ability to learn task similarity using expert-written documents (engineers' notes) provided by a telecommunication organisation.

2. We introduce two real-world use cases to highlight the real world applicability of the proposed methods. The first use case examines recommendation of additional information to perform dynamic decision support for engineers in the field - transfer of experience between experts (see Section 5.3). The second use case examines the transfer of experience between expert and non-expert personnel within the telecommunications work sector (see Section 5.4). We demonstrate how both of these use cases are achievable by learning similarity models empowered by DMLs.

3. We perform a short comparative study of developing representations from expert-written documents for similarity-based return on the basis of their accuracy on two simple classification tasks from our use cases.

5.2 Provision of Services for Telecom Organisations

Within field service provisioning industries there is increasing interest into the empowerment of workers to ensure the right expert knowledge is used at the right level in the decision process. In [23], the scheduling system interactively allocates tasks which empowered telecommunication engineers thanks to a personalised recommendation system that suggested tasks to an engineer based on their history of completed tasks. However, the increasing complexity of tasks lead to situations where engineers struggle to evaluate accurately the required work on tasks which are nearby and within their skill set. The amount of tasks generated every day across all business divisions can be very large - for example, in the telecommunications organisation who provided data for our use case, an average of 2,670 new tasks are generated every day. Time is often wasted by an engineer's need to develop a schedule to optimise his working day, which can further exacerbate the problem they face in finding appropriate work with the correct context information at the right time. This question becomes more critical when the type of services are inherently dynamic, such as when high priority tasks are raised that require an engineer's immediate attention, and require that he must abandon tasks which he might be unable to revisit on time. In a worst case scenario these tasks may miss their deadline, either because an engineer can no longer return to them or potentially because they overlooked them in the first place.

5.2.1 Stakeholders

In this section we briefly describe the roles and responsibilities of the main stakeholder groups impacted by our use cases.

Field Engineers

Field engineers are the group of highly-trained workers who install and maintain the required technologies to support the delivery of telecommunications services to the organisation's customer-base. The groups' chief responsibility is to maintain the distributed telecommunication technologies across the country, meaning that there are different groups of engineers serving different regions within the UK. They are referred to as 'field' engineers, as the work primarily requires installation or maintenance to be performed in the field, and off-site of a central location.

The workforce dedicated to these tasks are specially trained with appropriate skills to complete the work. This includes early on-the-job training in the form of an apprenticeship, and continuous professional development through training courses to advance and improve their capabilities. Engineer achievement of training is stored within a skills matrix, where their professional ability is formalised as a set of 'skills'. Each skill

refers to a specific technical competency that the engineer is able to practically apply. These skills are associated with the tasks that engineers are asked to perform, and it is ensured during task allocation that an engineer does not receive a task they do not have the skills for. However, the telecommunication equipment that the engineers maintain can be impacted by a broad variety of factors. While the skills required for the work can (in most cases) be identified in advance of the task being attempted, there is often a requirement for the engineer to adapt after the task has started. For example, a routine inspection of a generator can transform into a repair task if it is revealed that a specific component is broken. Alternatively, an engineer may be dispatched to resolve connectivity issues for a customer, and this requires diagnosis to identify the root of the problem. As a result, the work is highly reliant on an engineer's previous experiences and capabilities.

The engineer's create textual records of these experiences in engineer notes. These notes are required to document task progression and completion as part of work force auditing. Although they originated as this purpose, we believe these textual records of experience can be useful to inform decision-support systems. As such, they act as the basis for our similarity models in both use cases. We discuss the relationship between tasks and notes, as well as the structure of notes, in Section 5.2.2.

We discuss the development of a system to support engineers to complete a task by providing additional information in our first use case in Section 5.3. This use case demonstrates capability of transferring information between domain experts.

Desk-Based Planning Agents

Planning agents are the desk-based staff who support field engineers in their work by performing administrative duties, identifying incoming tasks, and supporting the completion of tasks. As these staff are not field-based and do not personally survey the tasks up for completion, their decision-making is primarily informed by knowledge provided by engineers. The most explicit form of this information is the notes recorded by engineers to report the progress of each task (described officially as 'Further' notes, but known informally as task updates). Though these agents develop aptitude in understanding some aspects of telecommunication engineering, they are not experts nor do they benefit from the experience or training that technical experts receive. The result is an increased likelihood of human-error and decreased efficiency when they interpret engineer notes, particularly in cases where the notes are complex.

One of the responsibilities of planning agents is to incorporate knowledge sourced from task updates to identify and regulate suitable task intervention or assistance (see Figure 5.1). We describe this process as anticipating the next 'scenario' for a given task.

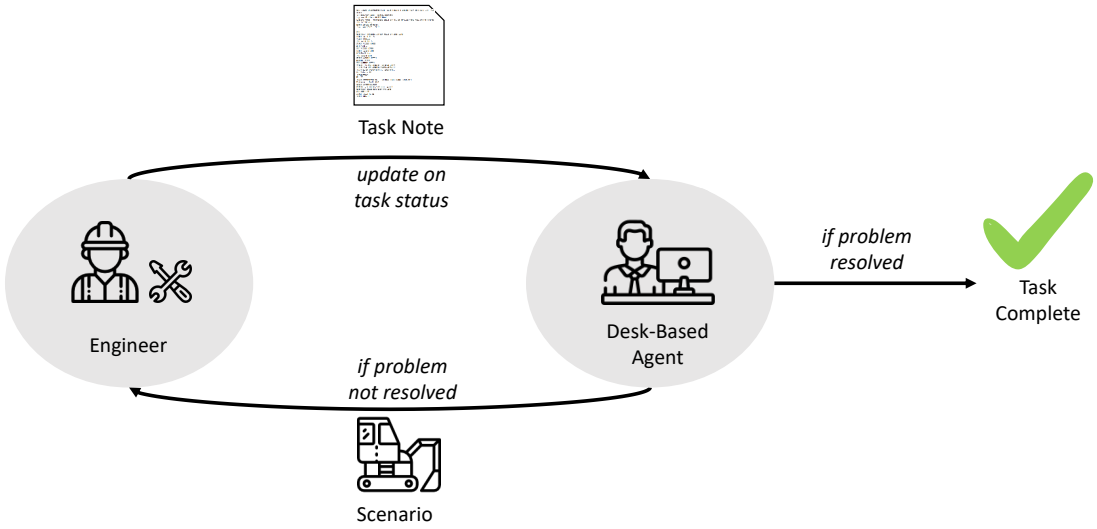


Figure 5.1: Desk-based agents (Non-Experts) supporting engineers (domain experts) to complete their tasks. If a task is completed, then the desk-based agent will sign off on the work. If the task is incomplete, then it is the agent's role to organise a suitable scenario to progress the task.

We discuss the development of a system to support the desk-based staff to anticipate the appropriate scenario using engineer notes as a data source in our second use case in Section 5.4, representing the transfer of experience between experts and non-experts.

Management

Though a record of human experience, the notes generated by engineers also act as an auditing tool. The notes allow management to maintain a record of worker performance, and this can be compared against best practice and company policy. Information from the notes could be combined with data from other sources to perform this work force auditing. Generally however, this procedure would use summary data analysis of the range of sources to check that the worker was meeting various performance targets. However, specific notes can also be used during these discussions, particularly if areas of improvement or further training are identifiable from the notes. In this thesis we focus on the interactions between engineers, as well as the interactions between engineers and desk-based agents, so we consider feedback into these management processes as out of scope. However it is worth highlighting management as a potentially impacted stakeholder group if further work with the notes as a data source is pursued.

5.2.2 Tasks Notes

In this domain, a task typically represents a time-sensitive activity on a piece of equipment (such as maintenance, installation or decommissioning) or interacting with and

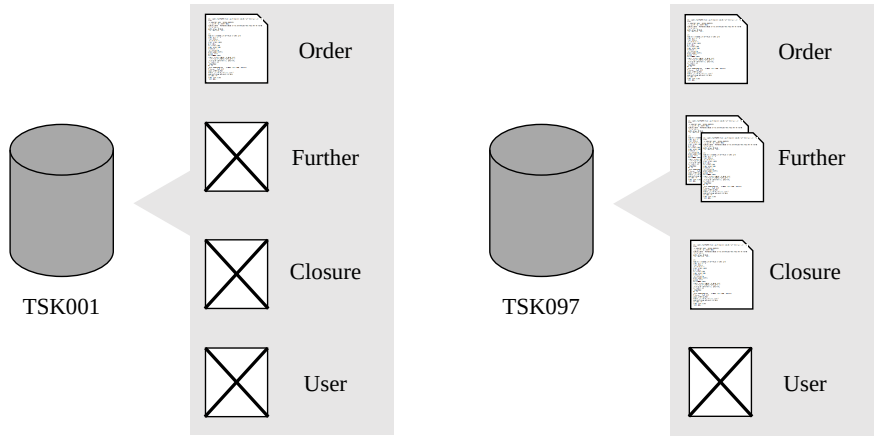


Figure 5.2: The relationship between telecoms tasks and engineer recorded notes.

responding to customer inquiries (both home and business). As part of work force auditing, field engineers record information about the tasks they have completed in text documents called "notes". These notes form a knowledge-base of experiential content and are comprised of rich, heterogeneous information. The notes are categorised based upon their contents. For example, Order notes contain information about the work requirements of a task. Details of task completion are stored in Closure notes, while records of task failure and the reason behind it are stored in Further notes. Lastly, an engineer can enter additional miscellaneous information about a task in its in User notes.

The notes are a semi-structured source of expert information describing a specific task. However, they are made complex by the fact that only certain note types may be present in certain tasks. For example, a task which has never been attempted will only be associated with Order notes, while there may be both Further and Closure notes describing another task if it had been failed at least once before it was successfully completed. Furthermore, a task will often not only be associated with multiple different types of notes, but also multiples of the same note type (i.e. two Further notes if the task has been failed twice). However, certain note types (such as Order notes) will never be duplicated. Figure 5.2 displays the relationship between notes and tasks for two examples. In this Figure, Task TSK001 is an example of a newly created task, where only the Order notes have been generated. Meanwhile, TSK097 is an example of a task that has been attempted twice and failed (notice two Further notes) before being successfully completed (detailed in the Closure notes). In both instances the engineer has declined to add more information in the User notes.

5.3 Use-Case 1: Transfer of Experience Between Experts

In UC1, we aim to develop a system which facilitates transfer of experience between experts. Specifically we aim to achieve this through development of a method to access and prioritize tasks which fall within the engineers' capabilities, experience and are of relevance to the business at that point in time. A recommender system has potential to fill this gap [133], but responses to a fuzzy logic recommender [23] suggested that users resented the lack of clarity behind its recommendations and felt that recommendations of the system were not indicative of their own perspective. The ability to explain a system's recommendation, or display a level of transparency which allows the user to understand the reasoning behind that recommendation, encourages trust between a system and its users [134]. In answer to this, we have built a Case-Based Reasoning (CBR) system with greater transparency and drawing on free text documents recorded by the engineers' themselves to inform its similarity model. In doing so, an engineer's experience is being shared with others to support their decision-making (allowing them to answer the question "based on the experiences of my colleagues, is this task relevant to my skill set and therefore something I can achieve?"). Thus, in this use case we focus on how we utilised the complex information source of engineer notes to develop a similarity model which can act as a basis for additional information recommendation.

Additional information can take many forms and assist an engineer in the field in different ways. We formally view additional information in this use case as the provision of extra knowledge which can contribute to the successful completion of a task. Of particular interest is additional information which may allow a user to pre-emptively identify possible task failure and potentially avoid it. In this manner, we hope to either prevent task failure or 'fail fast' such that minimum resources are wasted on a doomed task.

5.3.1 Use-Case 1: Evaluation

In this use case, we demonstrate a similarity model generated based upon Order notes, which are representative of the original work order calling for the task to be completed. Since an Order note is created at the same time as the original task, this guarantees that a task can be used to query the model immediately upon creation. This is an important component of a timely system, as it would be less useful to query the model after a task has already been allocated, or even failed.

For the purposes of comparison we have created a simple classification task where notes are classified according to one of four work types. The quality of the learned representations are assessed by their performance on this classification task. We extracted

two months of Order notes written by telecommunication engineers between March and April 2018. We filtered out any note which contained less than 50 characters, as we judged them not to be adequately meaningful. This resulted in a dataset of 1,610 notes split into four classes - Cabling (CAB - 227 notes), Jointing (JRT - 789 notes), Overhead (OVH - 503 notes) and Power Testing (PTO - 91 notes). These classes represent the primary required competence which is associated with each note.

Experimental Setup

Although a subject study would be desirable to understand whether the model develops a score which is representative in an expert's opinion, we will empirically evaluate the model using a simple classification task as a proxy. We compare several methods of learning representations for text documents on the basis of their performance on a similarity-based return classification task. This allows us to determine the most suitable method to use as a basis for our model. Specifically, we consider a statistical method (tf-idf), a learned method (doc2vec) to text representation and two deep metric learners (Siamese Neural Network (SNN) and Triplet Network (TN)). In addition, as our research has demonstrated that the performance of a deep metric learner can be enhanced through the selection of a suitable training strategy (see Chapter 4), we also consider an SNN which uses DynE and DynEE sample selection and a TN enhanced with Locality-Sensitive Batching (LSB).

The dataset was split into train and test sets for evaluation using 5-fold cross evaluation. The Doc2Vec feature size was 300 as this was found to be the optimal parameter through empirical evaluation. We used a gridsearch to identify the best combination of features for our tf-idf algorithm. Using the Python Natural Language ToolKit (NLTK) [1] platform, stop words were removed using NLTK's list of English stop words and stemming was performed, and the top 500 most common words were transformed into tf-idf features for each task. Further text pre-processing was not performed as the text contained many examples of domain-specific terminology, which could be lost if aggressive pre-processing was performed. The resulting vectors were very sparse, with most tasks having 10 or fewer non-zero features. Finally, the hyperparameters for the SNN and TN algorithms are presented in Table 5.1.

All sub-networks were composed of multi-layer perceptrons with 3 layers. Networks were intentionally shallow, as the small amount of data provided by the use case meant training vast quantities of weights and biases within a very deep network would be challenging. Each layer had 128 nodes and used ReLU activations. Due to the difference in training speeds and how quickly the networks approach convergence, we trained all

[1] The Python NLTK library, available here: https://www.nltk.org/

Architecture	Sub-Network	Epochs	Minibatches	Batch Size
SNN	MLP (Dense, 3 Layer)	50	81	16
TN	MLP (Dense, 3 Layer)	50	81	16

Table 5.1: Hyperparameters of networks for training on UC1 dataset.

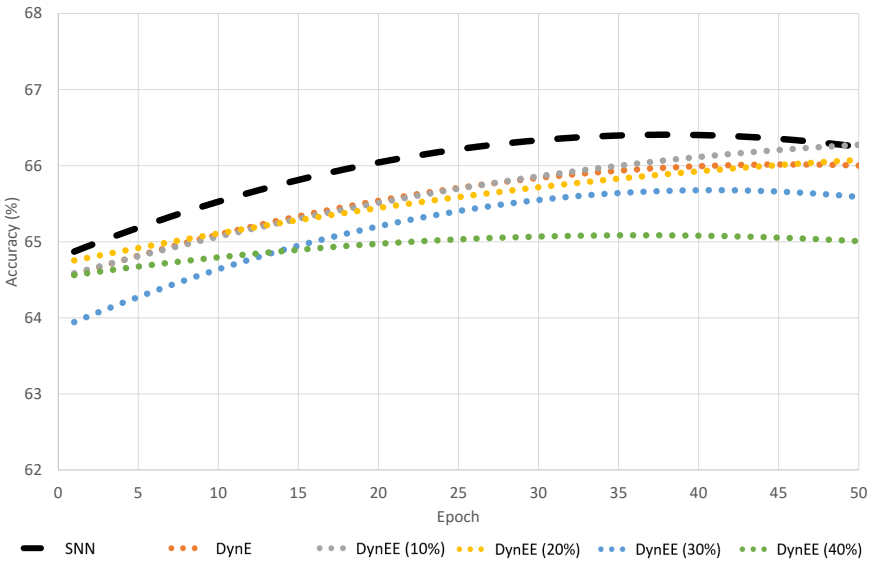
networks for 50 epochs and at each epoch we record accuracy at multiple values of k. This allows us to report in our results accuracy of these k values at the epoch which achieved peak accuracy for the respective architecture and training scheme.

Selecting Parameters for DynEE and LSB

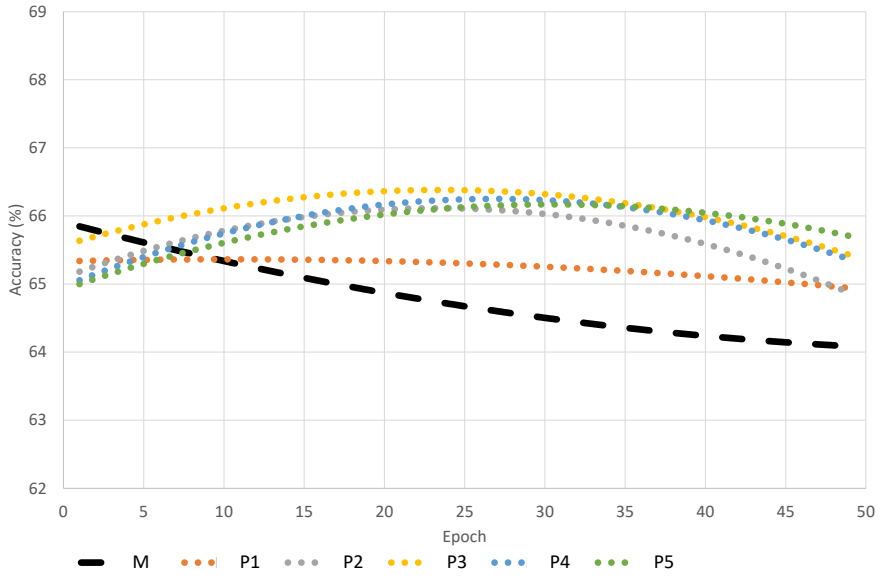
We performed some simple analysis to select the β ratio for DynEE and the number of projections for LSB. We set up a small experiment, examining the accuracy values achieved over the course of 50 training epochs at a k value of 3. For both DynEE and LSB, we then compared a range of parameters for each of the methods, investigating a range of β exploitation ratios between 0 and $^{|P|}/_{2}$ (50%) and projections between 0 and 6 for LSB. Methods were run every epoch to inform the creation of the next epochs training set. We used five-cross fold validation, and plotted polynomial trendlines of the results (see Graph 5.3). This allowed us to quickly identify suitable parameters for each of the methods.

As seen in Graph 5.3a, the trend line indicated that no exploitation would benefit the architecture during early training. However, later training (tending towards epoch 50) would be continue to converge to optima with a small proportion of exploitation knowledge (10 or 20%), while the baseline SNN would begin to overfit. As a result, we selected a β exploitation ratio for DynEE of $^{|P|}/_{10}$ to compare against a baseline SNN with no training strategy. This means that 10% of the training pairs at any one time were formed through exploitation, while the remainder were randomly generated to provide exploration. We anticipated that this was indicative that the SNN with DynEE would take longer to converge than the baseline SNN, but that it would obtain a superior accuracy from better approximating the optima when convergence was reached.

Our comparison of the number of projections for LSB, as seen in Graph 5.3b, was indicative of more promising behaviour. The baseline TN seemed to overfit almost immediately, while increasing projections would train to reach better accuracy before overfitting. This implied the LSB method would be more robust to overfitting, and the idea was supported by the fact that between epochs 20 and 40, the classification accuracy for representations learned by architectures using $p > 1$ were statistically significantly better than the baseline TN (using a one-tailed T-Test and with a confidence



(a) DynEE- exploitation ratio



(b) LSB - Number of projections

Figure 5.3: Trendline analysis of different DynEE and LSB hyperparameters for UC1 dataset on Doc2Vec representations.

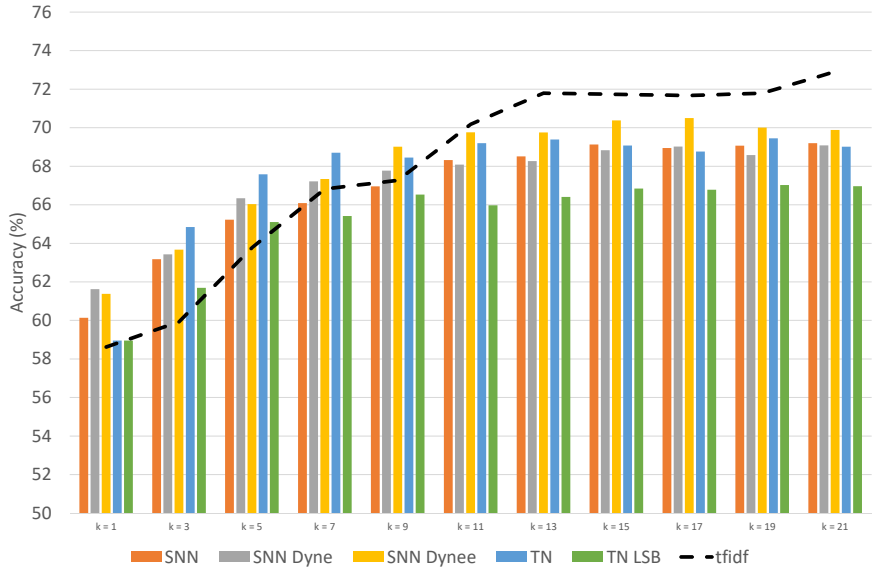
of 95%). Peak accuracy values were obtained in early training using 3 projections to divide the space. However, later training suggested that setting p to 4 or 5 would avoid overfitting for longer. With these results in mind, we selected $p = 5$ as the parameter for our experiments, but we also highlighted $p = 3$ and $p = 4$ for further investigation.

5.3.2 Use-Case 1: Results and Discussion

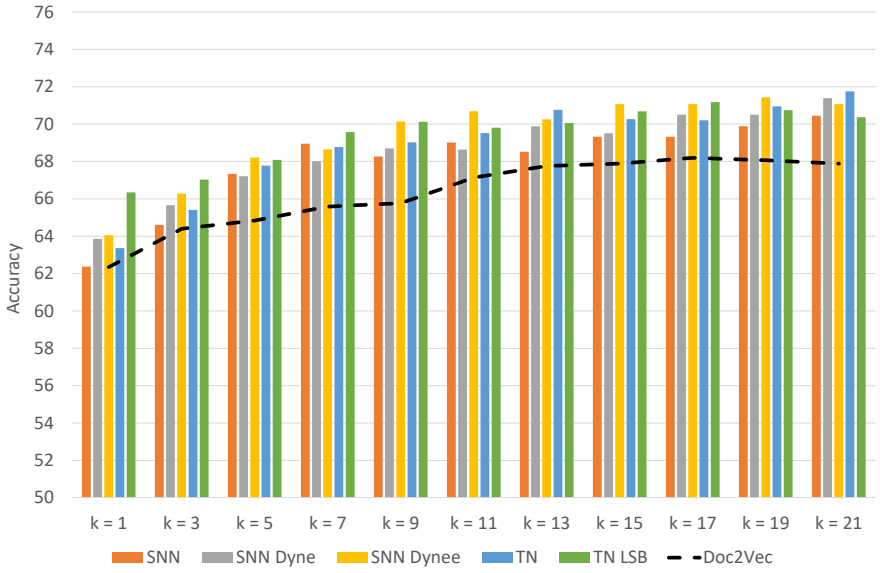
The results are presented in a graphical comparison in Figures 5.4. Performance on this task is generally capped between 71% and 72%, regardless of whether tf-idf or Doc2Vec features are used, implying a difficult task. The lower overall accuracy may be representative that the dataset is not large enough to achieve better accuracy. Overall we can observe that the representations learned by DMLs on Doc2Vec embeddings are better clustered, as evidenced by the improved accuracy at all values of k. The representations learned from tf-idf vectors generally perform better at lower values of k, though the raw data achieves better performance at high values of k. This is likely because of the sparseness of the tf-idf vectors. The notes are typically short, meaning that it is not unusual for tf-idf vectors to have a small number of non-zero indices. When transforming the input with the DML, this could mean it is more difficult to learn the weighting for feature combinations. These combinations could be important, as the classification is likely dependent on the presence/absence of certain key combinations of terms (evidenced by the better performance of tf-idf at high values of k). The Doc2Vec embeddings are dense, avoiding this problem and giving the DMLs more comprehensive input data. However, the raw Doc2Vec representations are limited by a small amount of training data, meaning that they struggle to learn the semantic relatedness between terms within the notes, and therefore between tasks. It is likely that the DMLs therefore are capable of improving Doc2Vec representations as they fill this gap by spending additional training time to learn the similarity between tasks.

SNN Architectures

Overall, results from the baseline SNN and the SNN using the DynE and DynEE training strategies demonstrate good performance on this task. When learned on Doc2Vec embeddings, classification performance on the representations gained from SNN, SNN DynE and SNN DynEE demonstrates noticeable improvement to kNN classification of the raw embeddings at all values of k. While the baseline SNN is comparable at lower values of k (until $k = 10$ for SNN DynE and $k = 7$ for SNN DynEE respectively), we observe that SNN DynE and SNN DynEE leads to improved clustering of the dataset over the baseline, particularly at larger values of k. In fact, SNN DynEE is comparable to the baseline TN and TN LSB for classification accuracy on most cluster sizes. This is likely because of the small size of the dataset; previously we had highlighted dataset size



(a) TF-IDF



(b) Doc2Vec

Figure 5.4: Accuracy of similarity-based return classification on UC1 dataset at increasing values of k, using representations learned with TF-IDF and Doc2Vec.

and a complexity as a barrier for the adoption of DynEE. However, in small datasets such as this it is evident that the training strategy does offer improvement beyond the baseline. This supports our conclusions about the algorithm in Section 4.2.3.

Results on the tf-idf feature vectors are mixed. In small cluster sizes ($k <= 5$), all of the SNN architectures demonstrate noticeable improvement over kNN classification performance on the raw tf-idf vectors, but the baseline outperforms them in larger clusters ($k > 11$). SNN DynEE maintains comparable performance in medium sized clusters ($7 <= k <= 11$), while both SNN and SNN DynE underperform the baseline. As we have intimated above, we suspect this is due to the sparsity of the tf-idf feature vectors. Interestingly, SNN DynEE achieves the best performance of any DML at medium and larger values of k ($k >= 9$). This implies that SNN DynEE is better able to learn to separate difficult parts of the vocabulary, something we will explore in future (see Section 7.1).

TN Architectures

Similar to the SNN architectures, both the TN architectures demonstrate noticeable improvements in classification accuracy over the Doc2Vec embedding baseline. This is particularly evident when the similarity-based return relies on only a single value of k to perform its classification (i.e. $k = 1$), as TN LSB obtains the top accuracy by a wide margin here and is on the cusp of statistical significance. This is a good indication that LSB is capable of mapping similar tasks to very similar regions of the space, which is evidenced by TN LSB's strong performance on classification using small k values ($k <= 9$). It is interesting to note that in larger clusters, the baseline TN architecture started to outperform TN LSB ($k >= 19$). This could be indicative that the number of projections used (5) presented too great a focus on difficult cases, and was warping the learned latent space to fit these complex cases. To investigate, we captured statistics for bucket contents using LSB with 3, 4 and 5 projections during the course of our experiments. We demonstrate our findings in Table 5.2.

We can observe that the number of buckets required is generally large for the raw data. Given that each projection has two sides, the maximum number of buckets that the space can be divided into is 2^p. Since empty buckets are discarded, we do not guarantee that all buckets will have contents. However, tending towards the use of all buckets is a good indicator that data points in the latent space are very spread out. We can observe this on the raw Doc2vec representations, particularly when only using 3 or 4 projections. After even a single training epoch, we notice that the number of required buckets is much less, highlighting that data is being clustered to a smaller region of the space. Furthermore, the average complexity (number of classes) of the buckets tends to

	Projections	@Epoch	No. Buckets	Pure(%)	Complex(%)	Complexity
TF-IDF	3	0	8	0	100	3.98
		1	3.2	0	100	3.63
		10	4.4	0	100	3.40
		25	4.6	4.35	95.65	3.31
		50	4.6	0	100	3.30
	4	0	16	0	100	3.88
		1	3.60	0	100	3.54
		10	7.80	20.51	79.49	3.00
		25	5.40	0	100	3.45
		50	8.6	13.95	86.05	2.72
	5	0	32	0.62	99.38	3.60
		1	6.6	3.03	96.97	3.14
		10	8.4	9.52	90.48	3.08
		25	9.6	10.42	89.58	2.92
		50	11.4	7.02	92.98	2.61
Doc2Vec	3	0	8	10	90	3.08
		1	4.80	0	100	3.57
		10	5.20	7.69	92.31	3.42
		25	7.40	8.11	91.89	3.20
		50	5.6	10.71	89.29	3.17
	4	0	15.8	10.13	89.87	3.10
		1	4.40	0	100	3.51
		10	5.4	16.67	83.33	3.14
		25	7.40	13.51	86.49	3.03
		50	8	11.11	88.89	2.82
	5	0	23	28.70	71.30	2.57
		1	9	11.11	88.89	3.08
		10	9	11.11	88.89	3.18
		25	13	15.38	84.62	2.90
		50	11	18.18	81.82	2.62

Table 5.2: LSB statistics for tf-idf and Doc2Vec experiments on UC1 dataset

increase at the start of training, but decrease with every epoch thereafter, evidencing that similar data points are being clustered together.

On the tf-idf vectors, we see mixed results. The representations gained from the baseline TN architectures outperform the raw tf-idf vectors when the similarity-based return

task uses low values of k ($3 <= k <= 9$), but underperforms at high values. The TN LSB consistently underperforms the other architectures on this task. This may be because dividing the space into buckets using projections is likely to produce large and extremely complex buckets in sparse spaces. There is evidence to support this idea in Table 5.2. We can observe that even though the raw tf-idf representations were divided into the maximum number of buckets available, these buckets were still highly complex and tended to contain representatives from 3 or 4 classes. This likely meant that TN LSB started training with difficult triplets, which we know from curriculum learning research can lead to stunted network development.

5.3.3 Use-Case 1: Recommending Additional Information

We suggest that based upon the developed similarity model, we can recommend additional information to users with the purpose of supporting their work. Specifically, we will identify the likelihood of potential risk categories to an incoming task and make a recommendation based on knowledge from the notes to counter the risk where appropriate. Though we focus on recommendation of risk information in this work, the same principle can apply to other problems and domains with similarly recorded expertise.

Firstly, we extract a set of risk categories from the Further notes of previously failed tasks. These categories are an abstraction of specific risks that are collected into a single related concept (i.e. both "the customer was not ready" and "the customer was not present" would fall into the Customer risk category). This presented us with six risk categories - Contractor, Customer, Duct Blockage, External Event, Planning and Time. These categories were formed based upon feedback from telecommunication engineering experts and the most common sources of risk. For example, though Duct Blockage is a reasonably specific point of failure, it is a very common one. Equally, while the category External Events covers many different hazards (i.e. dangerous animals, adverse weather, etc), it is much rarer for any individual risk to cause task failure.

We then label each failed task based upon the risk category which caused its failure. Note that any given task may have been failed more than once and so can be associated with multiple labels. Also note that these labels only apply to tasks which have already been failed - we do not generate risk category labels for tasks that were successful on their first attempt or have yet to be attempted (i.e. the tasks that lack Further notes).

To perform the recommendation of risk information, we can then submit a query case to the similarity model to retrieve a return set. However, instead of considering all possible tasks, we only consider tasks which are associated with Further notes (and therefore at least one risk label). We can then perform a vote weighted by similarity to gauge the likelihood of a given risk occurring in the query task (see Figure 5.5). We

perform this vote in the following manner.

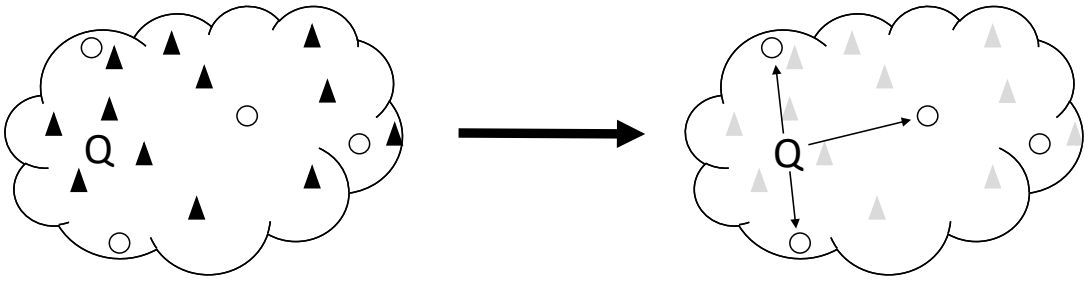


Figure 5.5: Recommending risk information using similarity-weighted vote from learned representations. If the triangles represent successfully completed tasks, while the circles represent tasks that failed at least once, then we can observe only the latter feed into the weighted by similarity vote to recommend risk.

Let us describe an individual risk category as r and a function to induce a score for the risk category of an unseen example $r()$. We will also describe a task as x and our full set of task examples as X. Similarly we will identify a failed task as $\hat{x}$, such that $\hat{x} \in \hat{X}$ and $\hat{X} \subset X$. We can retrieve a label for a given failed task using the function $y(\hat{x})$. We compare a query, q, with its set of nearest failed neighbours, $\hat{x}^{NN}$. To develop a score, each risk category is calculated by the following formula:

$$r(q) = \frac{1}{k} \sum_{i}^{k} sim(x_i^{NN}, q) \cdot s_r \tag{5.1}$$

where k denotes the neighbourhood size parameter and s_i denotes a binary value 'switch' which is set to 1 if x_i^{NN} has previously failed due to the given risk (i.e. $r = y(\hat{x})$), or 0 otherwise. What this means is that a risk category's score is based upon a similarity weighted vote of its nearest failed neighbours. In this manner, we can develop a score of the likelihood for the occurrence of each risk in a given query.

Recommendations to Resolve Risks

Though it is useful to demonstrate the likelihood of individual risks to an engineer, this is not necessarily helpful if they do not understand how to circumvent those risks. Therefore, we also generate a recommendation to answer any sufficiently likely risks. This is achieved by comparing each individual risk category score against a threshold. We have three possible classifications of risk - Low ($r(q) < 30\%$), Medium ($r(q) > 30\%$ and $r(q) < 60\%$) and High ($r(q) > 70\%$). The recommendations themselves are based upon the most common successful solution derived from the Closure notes of previously failed task. For example, most of the Closure notes suggest that many task failures

relating to the Customer label can be avoided by phoning the customer ahead of time. It is worth noting that this will not necessarily prevent the task itself from failing. If the Customer has still not completed necessary pre-work, then the task will fail regardless. It does however offer an opportunity to 'fail fast' (i.e. prevent the engineer wasting time traveling to the customer's location). This in itself will improve productivity, as the engineer will then be free to complete another task.

Although simple, these prototypical risk solutions are easily generalisable within any of the given risk categories. Furthermore, as they are representative of the notes, they draw on what the experts themselves (in this case telecommunication engineers) have commonly found to be a successful approach. Though the recommendations are currently confined to a singular generalisable recommendation for any given risk category, in future work we seek to develop personalised representations based upon the notes associated with the nearest failed neighbours which have since been succeeded based upon Closure notes.

An example of the system is presented in Figure 5.6. Note the text area on the left provides details of the original task, while the window on the right details the scoring across the list of risk categories. The bottom window is used to provide a recommendation - notice that the greatest risk being Contractor is highlighted and the system recommends that the user contact the Contractor in advance to ensure that the work is ready to begin.

Previously Failed Tasks and Progressed Tasks

Field service provisioning is a field where large scale jobs can frequently occur. These jobs are usually broken down into a series of related, often sequential, tasks which we describe as a work chain. Therefore, this environment must be considered when recommending additional risk information. Equally, there is potential for tasks to have been failed by one engineer before being attempted by another. Thus it is necessary for our system to consider at least some evidence of task history in order to make its recommendations.

We adopt a strict stance towards failure in task history for our additional information recommender; if a task, or any of the previous tasks in its work chain, has ever been failed previously, then we make a strong recommendation to counter this risk. The system will still display scoring for other risk categories, but will highly recommend that action be taken to answer this specific risk. Furthermore, it will highlight that this task (or a member of its work chain) has been failed in the past and provide the Further notes regarding the failure. These notes include contact information for the engineer that previously attempted the task, as well as specific information on the failure. This

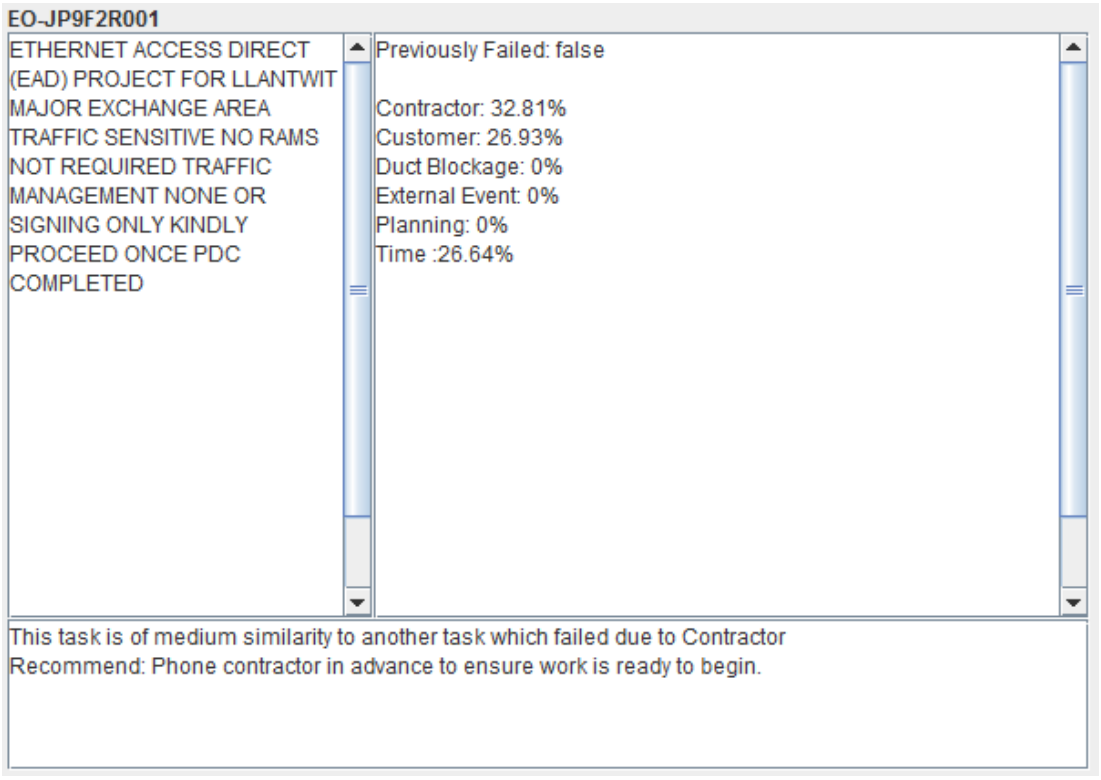


Figure 5.6: Additional information recommendation for a medium risk task

enables the engineer to form a response to the risk or contact the engineer for further information as required.

An example of the recommendation made regarding a previously failed task is shown in Figure 5.7. We can observe that the right text area has been extended to include the Further notes of the previously failure. Note also that the bottom text area highlights which risk category caused this task to fail on its previous attempt. If more than one Further note was associated with this task, then all previously written Further notes would appear in the right text area. Similarly, if a previous task in the work chain had failed, the bottom window would identify this.

5.4 Use-Case 2: Transfer of Experience Between Experts and Non-Experts

In this use case, we explore the transfer of experiential knowledge between expert and non-expert users. To achieve this, we have selected a problem where engineers rely on input from the supporting desk-agents in order to progress towards completion of a task. This allows us to understand how DMLs could perform in a situation of asymmetrical

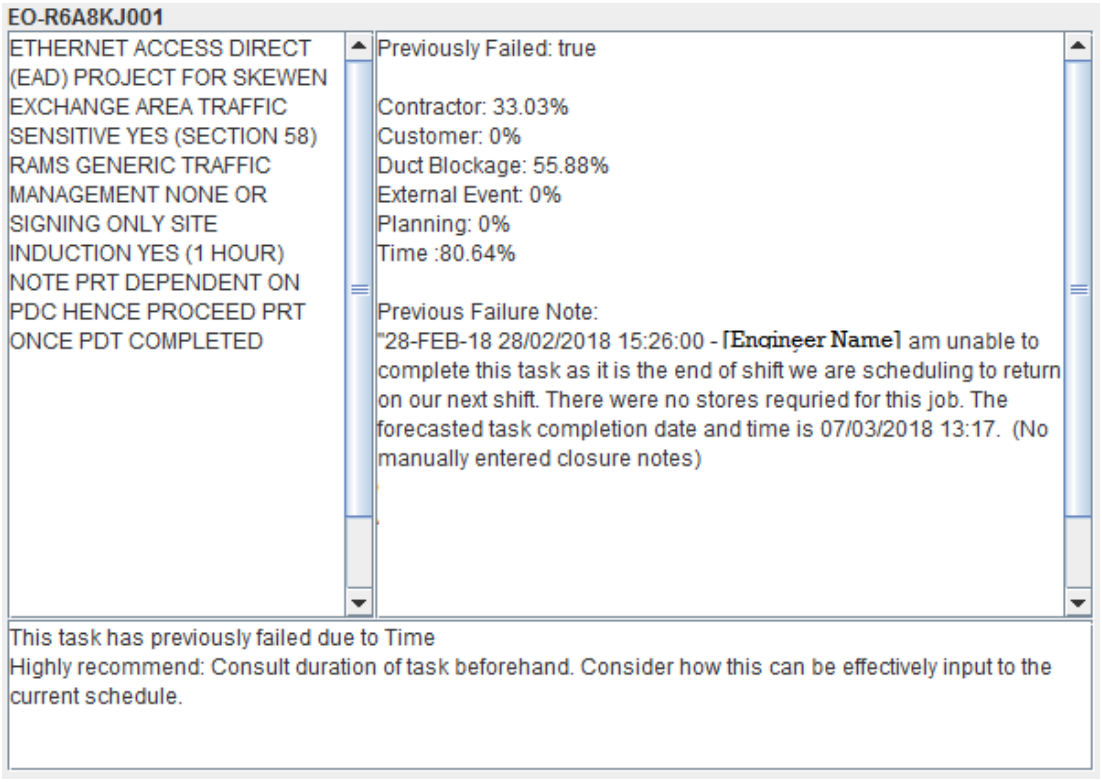


Figure 5.7: Additional information recommendation for a previously failed task

experience levels. Furthermore, this use case is highly revealing about the practical requirement for real-world systems to be explainable, something which we explore in more depth in Chapter 6.

As we have alluded to previously, the maintenance and installation of telecommunication equipment is often a multi-stage process. The treatment of complex orders (such as fibre access installation) requires decomposition into a chain of tasks, together described as an 'order journey'. Each individual task can involve various external dependencies (e.g. traffic management, hoist, and digging) and be subject to hazards or delays. Throughout the journey, planning agents must decide the next action to progress the order on the basis of the textual notes reported by technical engineers. However, understanding these notes can be challenging for non-experts in the field of telecommunication engineering. A recommender system offers means to support the desk-based agents in their work and pave the way for potential automation of some diagnosis operations in future. However, such a system would need to prove its trustworthiness for real-world application through transparent and explainable decision-making. Therefore, to support planning agents we have developed a recommender system to identify the most appropriate scenario for a given query note.

5.4.1 Use-Case 2: Evaluation

In this use case we demonstrate a similarity model based upon Further notes. These Further notes are important, as they are the mechanism by which the engineer can update on the status of a task, and they are also the foundation for a desk-based planning agent's decision-making. We performed an exploratory evaluation to understand the effectiveness of different representation learners on the task of scenario classification using a dataset extracted from our use case.

We extracted 46 days worth of engineering note data, spread between the months of May, August, September and October. In total, we extracted approximately 6,800 task notes over 33 unique scenario types (classes). We then removed any class which contained less than 5 examples. It also became clear that a certain scenario, "No New Action Required" (NNR), was fully reliant on external information and not on the contents of the note. This was because the NNR class was only relevant if a scenario had already been organised for a given task. Based upon feedback gained from co-creation with the user group, we decided to remove this class until the external data source was available. The resulting dataset contained 5,343 notes spread between 29 classes. There was notable class imbalance, with the rarest class containing only 7 notes while the most populated class contained 1,120 (see Table 5.3).

Experimental Setup

Using this dataset, we created a classification task where notes were classified according to one of 29 scenarios (see Table 5.3). The dataset was divided into distinct training and test sets using 5-fold cross-validation. We considered both a statistical (term frequency/inverse document frequency) and a learned (Doc2Vec) method of learning representations. For each of these representations, we performed additional learning using a baseline SNN, an SNN with DynE training strategy and an SNN with SNN using DynEE strategy. We also considered a baseline TN architecture, and a TN which leveraged the LSB training strategy. We focused on similarity-based classification, and adopted kNN for this purpose.

The hyper parameters for both representation learners were optimised using a grid search (an exhaustive search of all combinations of hyper parameters for a given algorithm). In the case of tf-idf, data was pre-processed by removing stop words and stemming words to their root form using the Python NLTK platform. We then considered the 300 most common unigrams (n-gram range of 1) to build a representation. Finally, this output was normalised using cosine normalisation. For Doc2Vec, a window size of 10 was used to identify semantically related words and generated a representation of 300 features. The hyperparameters for the SNN and TN algorithms are presented

Scenario	No. of Examples
Aerial Cable Required	22
ARLLAOH	7
Asset Assurance Required	55
C002 - New Circuit D Side	51
C004 - Plan Do Installation	154
C017 - D-Pole Validation	105
Customer Readiness and Sales Query	333
Customer Access	41
Complete	345
Dig Required	614
Duct Work Required	11
Exchange Equipment Required	18
Faulty E Side	350
Frames Work Required	60
Hazard Indicator	93
Hoist Required	286
Hold Required	127
Line Plant Required	36
Manhole Access Required	25
New Site Required	10
No Access	164
No Dial Tone	28
Out of Time	1120
Planning Required	431
Polling Required	319
Survey Required	32
Track & Locate Required	193
Traffic Management Required	198
Underground Work Required	124
Total	5353

Table 5.3: Number of examples within each class of UC2 dataset.

in Table 5.4.

All sub-networks were composed of multi-layer perceptrons with 3 layers. Each layer had 128 nodes and used ReLU activations. Due to the difference in training speeds and how quickly the networks approach convergence, we trained all networks for 25 epochs

Architecture	Sub-Network	Epochs	Minibatches	Batch Size
SNN	MLP (Dense, 3 Layer)	25	81	16
TN	MLP (Dense, 3 Layer)	25	81	16

Table 5.4: Hyperparameters of networks for training on UC2 dataset.

and at each epoch we record accuracy at multiple values of k. This allows us to report in our results accuracy of these k values at the epoch which achieved peak accuracy for the respective architecture and training scheme.

Finally, similar to UC1 we performed a short empirical analysis to understand the most appropriate parameter settings for DynEE and LSB. As a result of this empirical analysis, we selected a β ratio of 20% and 10% for SNN DynEE architectures trained on tf-idf vectors and Doc2Vec embeddings respectively. We elected to use 5 projections, meaning a maximum allocation of up to 32 buckets, for our implementation of TN LSB across both experiments.

5.4.2 Use-Case 2: Results and Discussion

The results of the experimentation can be seen in Figure 5.8. Tf-idf offered superior performance on this problem when compared to Doc2Vec. We surmise that this is for two reasons. Firstly, Doc2Vec (like other neural network based approaches) commonly requires a large training set to function very effectively. Pre-training of a Doc2Vec model is also not valid here, due to the high usage of unique technical vocabulary in the notes. Secondly, the likely scenario for a given note is highly reliant on the technical vocabulary which is used to describe the work performed as part of the task. This is evidenced by the extremely good performance of tf-idf when only a single nearest neighbour is considered to inform the classification. The text leading to a scenario being recommended is quite specific, however, there is a large variety in the way this is expressed in the notes. As a result, very small clusters are good for recommending a class label for unseen data. However, due to the large number of classes in the dataset, the imbalance between the number of examples within certain classes and the sparsity of tf-idf representations, class boundaries in the latent space are likely to be poorly formed. This explains the diminishing accuracy of similarity-based classification as the size of k increases. Furthermore, the partner company who provided the dataset had previously attempted a simple rule-based token-matching approach, but its performance did not match either of the above methods. This suggests that the additional information that tf-idf offers about term rarity (in the form of its idf portion) is important.

Although none of the DML architectures achieved the performance of the tf-idf with

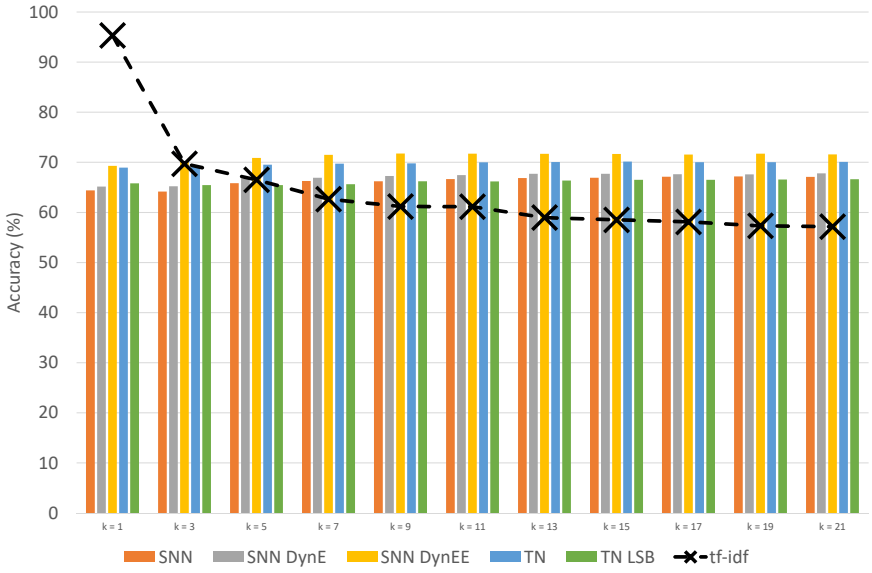
a k value equal to 1, they demonstrated much better clustering of the data at high values of k. The similarity-based classification at $k = 1$ was very likely affected by the transition from sparse vectors to the dense representations learned by DML architectures, making it difficult for DMLs to learn to emulate the strong performance of very small clusters in tf-idf. However, the dense representations meant that DMLs were able to learn relationships between feature groups to make the clusters more robust at scale. As a result, all DML architectures learned to produce representations which noticeably outperformed the raw representations when the neighbourhood considered for classification was larger than $k = 7$.

All DMLs learned to produce representations which outperformed those produced by the Doc2Vec baseline at all values of k. The justification of this is likely that the Doc2Vec struggled to learn to model the problem with the limited number of training examples. The DML architectures effectively offered an additional, targeted, training phase. This training phase was augmented with knowledge of similarity between examples, allowing the DMLs to learn more effective representations for similarity-based classification. However, they were limited by the Doc2Vec algorithm's inability to learn effective relationships between terms in the UC2 dataset, and therefore the results show less improvement over the baseline than those obtained at larger values of k on tf-idf representations.

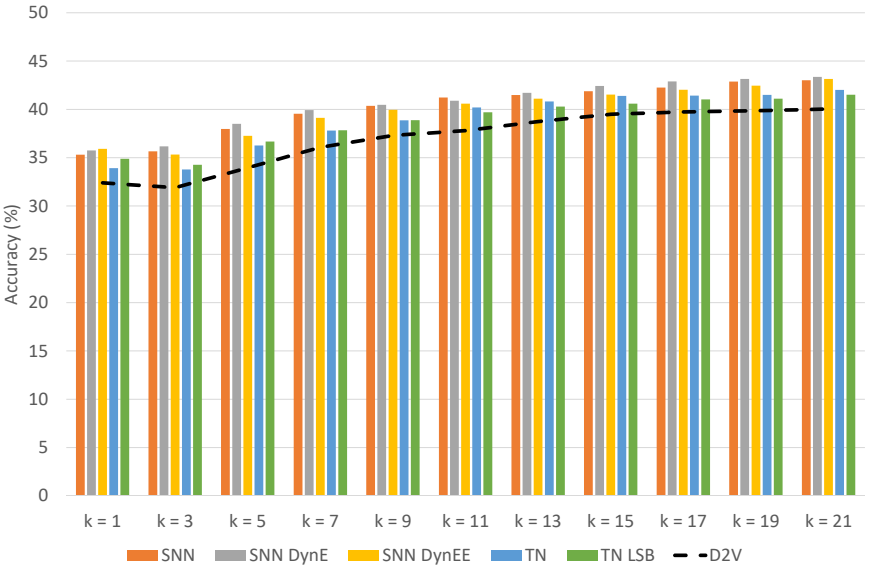
SNN Architectures

All three SNN architectures performed well on the UC2 dataset using either tf-idf or Doc2Vec representations. Generally, the SNN and SNN DynE architectures performed comparably to one another across both tasks. Using tf-idf vectors as input, SNN and SNN DynE underperformed the baseline when small neighbourhood sizes were considered for classification ($k <= 5$), but outperformed the baseline at larger values of k. This is indicative that the latent space learned by the architectures is better structured for larger clusters of examples. On the Doc2Vec representations, all three SNN architectures performed extremely well, outperforming both TN architectures. We take this as indicative that the problem defined in UC2 is closer to a matching problem which is where SNN excels. This is supported by the strong performance of the raw tf-idf vectors when $k = 1$, as this suggests that the classification is reliant on matching vocabulary.

SNN DynEE was the top performing DML across both representations on the UC2 dataset. On the tf-idf representations, the SNN trained with DynEE training scheme achieved results comparable to the baseline from $k = 3$ and outperformed the baseline at all larger k values. At the largest sampled value of k, $k = 21$, SNN DynEE achieved



(a) TF-IDF



(b) Doc2Vec

Figure 5.8: Accuracy of similarity-based return classification on UC2 dataset at increasing values of k, using representations learned with TF-IDF and Doc2Vec.

a peak accuracy of 71.6%, while tf-idf only achieved 57.1%. These vast improvements demonstrate the significantly improved clustering of data achieved by training using similarity information. On the Doc2Vec embeddings, the SNN DynEE performs well when only a single neighbour is considered for the classification. When larger neighbourhoods are considered for classification, SNN DynEE performs comparably with the other SNN architectures, which are the top performers on this task. Overall the results from both representations on UC2 dataset are indicative that the DynEE training strategy is effective for emphasising complex boundaries to support improved training of the SNN architecture.

TN Architectures

The TN architectures generally underpeform our expectations on the UC2 dataset. We would have anticipated that the additional information provided by training on a triplet would cause the architecture to have an overwhelming advantage. However, although the TN architecture performs comparably to the SNN DynEE when using tf-idf vectors as input, it does not achieve superior accuracy and TN LSB is the worst performing architecture. Interestingly, the baseline TN actually converged more rapidly (see Figure 5.9), but the SNN DynEE was able to leverage complexity information to converge to a better optima. The graph also reveals that TN trained using the LSB training strategy started at a much lower accuracy, but showed greater improvement over the course of training. This could suggest that when dividing the feature space into buckets in preparation for sample selection, the buckets were very complex and offered a difficult starting point for training. This is evidenced in Table 5.5, where we can observe that before training (@Epoch 0) the buckets of tf-idf vectors are more complex than those which contain Doc2Vec embeddings. We know from curriculum learning research that starting with 'difficult' examples can be detrimental to training, and this may be the case here.

5.4.3 Use-Case 2: Recommending Scenarios for Incoming Tasks

We have deployed a prototype of the described recommender system applied to our use case and accessible on the company intranet as a web application (see Figure 5.10). The deployed application currently makes use of tf-idf representations supported by kNN to inform its recommendations, due to the effectiveness of this setup as demonstrated by our experiments. When a query is entered, the text is converted to tf-idf representations and classified by kNN to make a recommendation from the known 29 scenario labels to support desk-based agents decision-making. For example, in Figure 5.10 the system has recommended the scenario AER (Aerial Cable Required). In response to

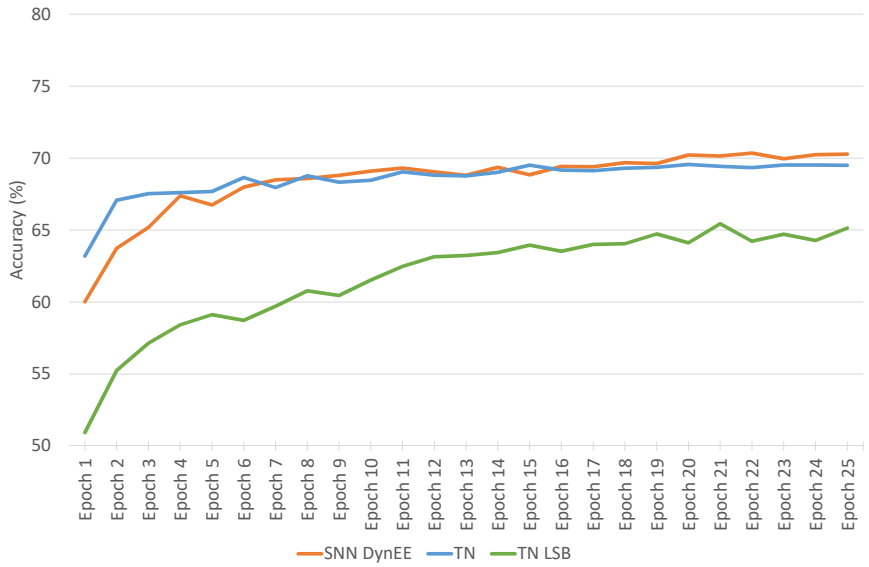


Figure 5.9: Training speed of architectures on UC2 task with tf-idf representations, using accuracy at $k = 3$ as a proxy for convergence.

a recommendation, users may feedback on whether it is 'good' or 'bad' and optionally enter feedback text and a corrected label. In future iterations of the system, this feedback will be used to update the classifier using a feedback engine which has been developed by the company partner. The user can access an ordered list of other potential scenarios by selecting 'Recommending Other Scenarios (ordered per confidence score)'.

5.5 Conclusion

In this chapter we have presented our work towards proving the hypothesis that DMLs present an opportunity to fulfil the vocabulary and similarity knowledge containers within a CBR system. To this end we have introduced two real-world use cases from industry which aim to leverage similarity information to enable transfer of experiential content. Over both of our use cases, we have demonstrated similarity models built upon textual documents written by experts as a source. In UC1 we use our model to facilitate sharing of experience between experts by generating additional information to support experts while in the field. We have focused on the pre-emptive identification of risk categories and build on common solutions from the notes to recommend a possible work around for these risks. In UC2 supported non-expert decision-making by leveraging information provided by experts, in the form of a recommender system to suggest suitable task intervention through scenarios.

Across these use cases we have investigated deep metric learners as a mechanism to

	Projections	@Epoch	No. Buckets	Pure(%)	Complex(%)	Complexity
TF-IDF	3	0	8	0	100	23.93
		1	4	0	100	22.12
		10	6	10	90	13.86
		25	5.8	6.90	93.10	13.79
	4	0	16	0	100	20.38
		1	6.6	6.06	93.94	17.61
		10	9.6	10.42	89.58	12.78
		25	9.2	8.70	91.30	11.51
	5	0	32	0	100	16.76
		1	8	0	100	16.01
		10	7.6	10.53	89.47	12.90
		25	15	10.67	89.33	9.70
Doc2Vec	3	0	8	0	100	21.95
		1	3.2	0	100	21.85
		10	3.8	0	100	22.45
		25	4.4	9.19	90.91	17.40
	4	0	16	0	100	17.98
		1	3.8	5.26	94.74	21.58
		10	5.2	11.54	88.46	18.82
		25	4.2	14.29	85.71	16.48
	5	0	32	0	100	13.76
		1	6	6.67	93.33	19.16
		10	5	4.00	96.00	18.46
		25	7.2	13.89	86.11	16.16

Table 5.5: LSB statistics for tf-idf and Doc2Vec experiments on UC2 dataset

combine the vocabulary and similarity knowledge containers of a case-based reasoning system. This has demonstrated promising results, and highlighted some areas which require further study. In particular, our chosen use cases were much more vocabulary reliant than we had previously anticipated. As a result of this, tf-idf representations performed better than expected. However, DMLs demonstrated capability to improve clustering of the data: in UC1, there was evidence that DMLs supported better clustering of the data in small neighbourhood sizes; while in UC2 the results indicated that DMLs were capable of of improving the complex class boundaries created by the sparse tf-idf representations in the imbalanced dataset. On the other hand, the DMLs demonstrated universal improvement over the Doc2Vec representations on both use

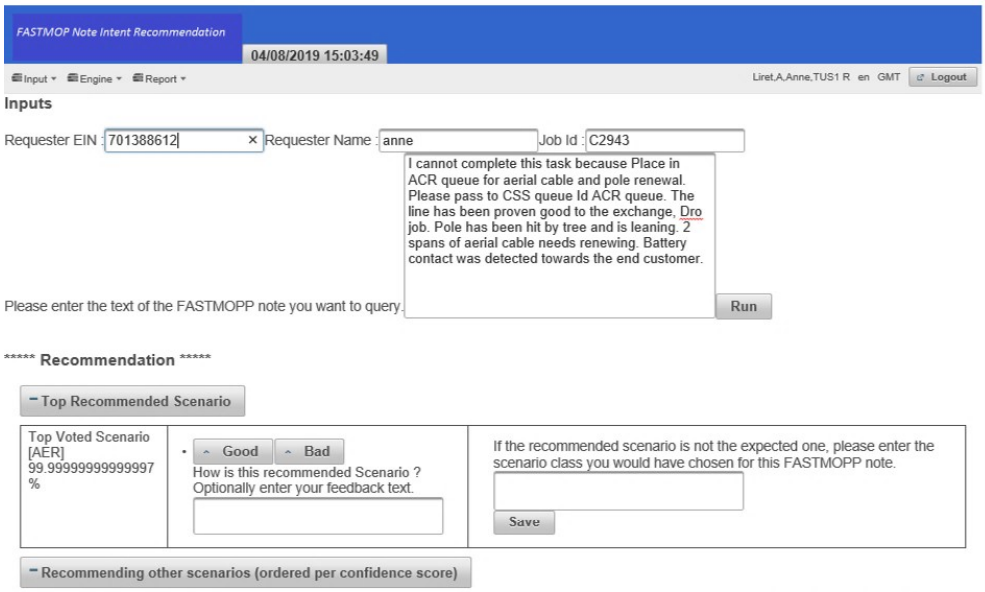


Figure 5.10: The recommender system developed during the research on use case 2. The application allows a user to enter a query note, and recommends an appropriate scenario.

cases. This may be because of the limited training set for the learned method, but is more likely due to the dense representation of the Doc2Vec being a better basis for metric learning. This proposes an interesting avenue of future work, which we will further explore in Section 7.1. Our results therefore suggest that DMLs can learn to produce representations optimised for similarity calculations which offer clear improvement over dense representations gained from word embeddings, but require refinement to outperform statistical methods.

More than this, these use cases have been very revealing about the needs for explainability. Although we were unable to perform a user test with the developed application for UC1, a recurring theme in the feedback from stakeholders throughout development indicated they were keenly interested in why specific risks were highlighted. Throughout our discussions with users in UC2, there was continual interest in supporting desk-based staff to learn on the job by providing explanations for why the recommended scenario was suitable for a given query. This ties to our background knowledge of the area, where it was explicitly discussed that stakeholders were keen to more deeply understand the autonomous model and information they were working with to better inform their working practice. In Chapter 6 we discuss the development of an explainability framework designed to support desk-based staff and engineers by integrating with machine learning systems. We explore its development specifically on the problem associated with UC2.

Chapter 6

Similarity Knowledge to Support Explanation

One of the advantages of similarity-based architectures is that they are generally predisposed towards explanation (see Section 2.3.3). As we highlighted on the use cases within the previous chapter, there is a need at an operational level for users to better understand the systems they are using to achieve superior working performance, nurture trust and ultimately increase productivity [128, 134]. In a real-world case, the quality and benefits of explanation depend on how timely and comprehensively they are produced. However, explanations are typically crafted to respond to specific user needs and specific applications [135, 136, 128]. This practice is both time-consuming and inefficient. We believe that there are overlaps between the requirements of an explanation for different applications. We are therefore motivated to create a general purpose explanation framework which can interface with a broad variety of projects across an organisation to reduce the cost of provisioning an explanation for individual applications.

In this chapter we present a framework formed of three components; a classification engine, an explanation generation engine and a feedback loop to ensure iterative refinement (see Figure 6.1). The framework is modular, allowing the classification engine to be switched with other learned models as necessary. The explanation engine operates upon the classification engine's output (as well as some external knowledge bases), to explain system decision-making. It achieves this by incorporating a catalogue of explainability techniques to provide transparency around system decision-making, and improve user understanding of the source data. Two progressive levels of explanation content have been developed: low-level explanations which provide key insights on the data; and high-level explanations which generate relevant sentence summaries. The

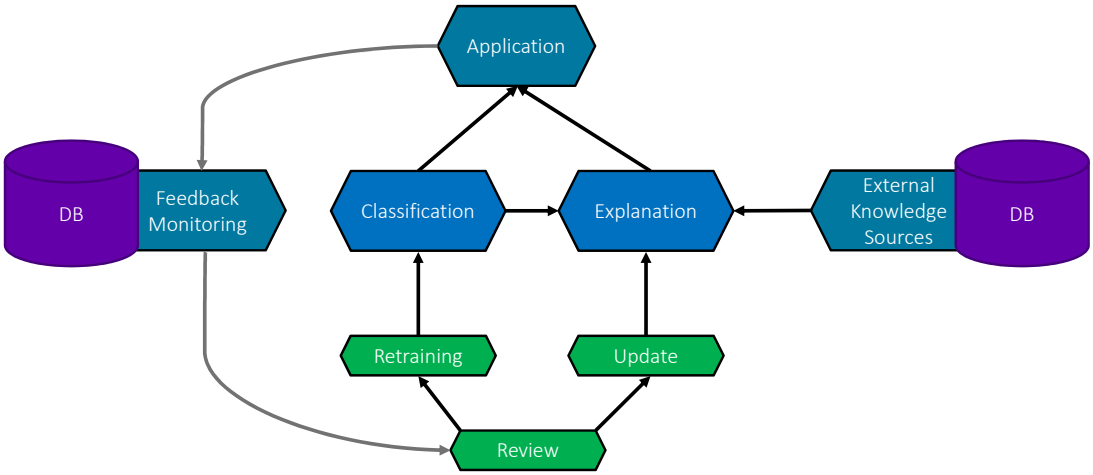


Figure 6.1: A flow diagram of the developed system, displaying its linked components.

progressive approach allows increasing levels of complex, context-aware explanations as users require.

We demonstrate the capabilities of this framework with the real-world use case of improving the transfer of information between telecommunication field engineers and desk-based planning agents (see Section 5.4). This recommender acted as the classification engine to test our framework, and allowed the opportunity to co-create various explanation methods with a real user base. The goal of the system is therefore to identify the appropriate scenario for a desk-based agent given an engineering note and explain why that scenario was selected. Though we demonstrate the application of this model to a specific use case, our method can be adapted to any reasoning task.

Furthermore, we extend our analysis to include an investigation of the relationship between explanation quality and similarity knowledge between a query, its neighbour set and its explanation. To this end, we introduce two novel similarity-based metrics, called Meet-In-The-Middle (MITM) and Trust-Your-Neighbours (TYN) respectively. Using these metrics, we generate some interesting analysis and insight into use of similarity for measuring explanation quality.

6.1 Contributions

In the final contribution chapter of this thesis, we explore explainability of DML architectures. Specifically, we present work towards proving our hypothesis that, as they are fundamentally similarity-based architectures, the output of DML architectures can be explained effectively in situations where multiple user groups of varying domain expertise are using the system. To this end, in this chapter we explore our third and

final research question of this thesis:

- How can we explain the output of similarity-based architectures (including DMLs) intended to support user groups of varying domain expertise, and how can we autonomously evaluate the quality of produced explanations?

Towards answering this question, we present the primary contribution of this chapter. Here, we describe the development of an explainability framework based upon one of our use-cases, and assess the quality of these explanations with feedback from two stakeholder groups of differing levels of expertise from within a telecommunication organisation - (expert) engineers and (non-expert) desk-based agents. Furthermore, we propose two novel autonomous evaluation methods for explanation, and compare their performance on an empirical study using statistical text representation methods and DMLs. The results highlight the practical utility of a hierarchical explanation framework, as well as the value of similarity knowledge as a starting point for research into the evaluation of explanations. We summarise this contribution through a number of novel secondary contributions:

1. We outline the development and implementation of a modular explainability framework and detail several of its sample modules as applied to the real-world problem of supporting desk-based planning agents in the telecommunications engineering domain.

2. We perform a qualitative evaluation to understand user opinion on the quality of provided explanations with feedback from two user groups of different levels of expertise. The results indicate that the judgement of what forms a good explanation changes based on domain-expertise: experts preferred explanations to mirror their reasoning, while non-experts emphasised task performance.

3. We explore the correlation between the quality of an explanation and similarity knowledge within the latent space using two novel metrics: Meet-in-the-Middle (MITM) and Trust Your Neighbours (TYN). Results from an empirical study comparing representations gained from tf-idf, SNN and TN highlight that similarity is a promising starting point to model the quality of explanation.

6.2 Development of Explanation Strategies

Use case 2 (see Section 5.4), offered a platform for co-creation to identify what the users considered important aspects of explanation. Meetings with engineers, desk-based agents and their managerial representatives occurred weekly throughout development of the framework. The results of these sessions revealed that the overwhelming desire from

co-creation participants was that explanations should be counterfactual, supporting findings in [120]. Co-creation participants were specifically interested to understand why a certain scenario was recommended before another, and what made a given note unique among similar notes. Furthermore, there was an acknowledgement throughout co-creation that desk-based agents would require more clarity around their explanations than engineers would, but that this would also be down to an individual. One example cited was that an experienced desk agent may well know more than an apprentice engineer. Therefore, the framework should be flexible to give the level of explanation support that is required.

With the results of this co-creation in mind, in this chapter we present a framework of explainability mechanisms that support a classification engine by explaining its output. The idea is to have multiple levels of explanation support by providing explainablility methods of increasing contextual awareness. In this work, we divide explanation mechanisms into two categories:

- **Low-level explanations methods** allow the user to visualise key information that provide insight to system decision-making and support interpretation.
- **High-level explanation methods** augment one or more low-level explanations with contextual information to enable more comprehensive explanation.

As this is initial work towards an explainability framework, we have constructed two example low level explanation modules, and one high-level explanation module and integrated them within the framework. Furthermore, as inspired by [12], we highlight each goal that a specific module is designed to achieve. It is our eventual goal that the framework is completely modular and will expand so that both 'off the shelf' and novel explanation methods are integrated. To this end, the framework structure we have proposed is intended to be easily extensible. It is outwith the scope of this thesis to develop further explanation methods for integration with this framework, but this could serve as an avenue of future collaboration with our industry partner.

Though the use case we have selected for discussion in this paper is confined to the use of textual data, the goal of the framework is to be data agnostic. The idea is that this will provide a resource for developers within the telecommunication organisation to easily and quickly integrate with their projects. Effective cataloguing (i.e. allowing searching by explanation type, the explanation goal it supports and data type) is key to provisioning a maintainable and accessible framework.

6.2.1 Low-Level Explanations

Low-level explanation methods describe key information directly extracted from the data itself or generated as part of the decision-making process. In the literature these are described as analytic explanations [22].

Confidence Measures

We can establish the confidence of our predictions with the traditional method of using similarity as a proxy [136]. If similarity is sufficiently high, we can be confident that our classification is correct. We base our confidence on the similarity of the nearest neighbour from a given label. Confidence measures can be seen as a form of justifying the decision which has been made by the system.

Word Overlap and Scoring

Scoring features to identify their contribution to algorithmic decision-making is a common trope throughout traditional AI methods [137, 128] and the subject of growing work in modern neural methods [138, 139]. Researchers in this area have identified that it is important for users to understand the differences between a query and its neighbours [128]. With this in mind, we designed this module to promote understanding of the impact that note vocabulary has on system decision-making. The overlap component identifies key terms which appear both in the query and within the neighbour set of a particular label. This enables the user to quickly visualise key similarities or differences between the notes and inform about complementary terms from similar notes in the corpus. The word scoring module then measures the activation of terms to highlight the influence of each term's local similarity on selection of a given neighbour note.

A key aspect of this module is correctability, as it offers the user a simple interface to highlight non-relevant keywords which were included in the explanation, and report relevant key words which were missing. In turn, this allows update of the explanation to improve it for similar users, as part of end-to-end debugging of explanations [140]. Additionally, this method can be extended to cover phrases or embedding-based approaches [135]. Word scoring and identification of overlapping terms is a method of improving the user's ability to understand the underlying concepts of system decision-making and improve interpretability of the process.

6.2.2 High-Level Explanations

While low-level explanations identify key information about the query or recommendation, they are potentially inaccessible to non-expert users. In these scenarios, it

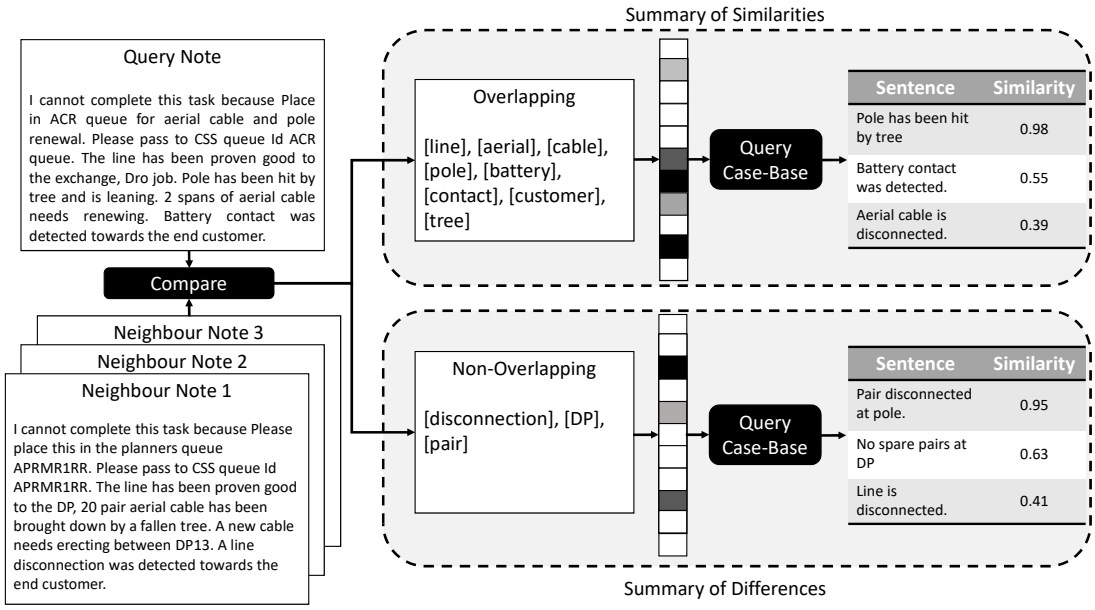


Figure 6.2: Summarisation of similarities/differences between a query note and a set of neighbours.

would be helpful to give the information context by incorporating relevant background knowledge. High-level explanations cover verbal and visual explanations [22], which are generated by building on insights from low-level (analytic) explanations. In this work, we use the example of generating summaries to contextualise similarities and differences between notes based on the output of the 'word overlap and scoring' component.

Summarisation of Similarities/Differences

We consider a method of extractive summarisation to create a verbal explanation of similarities and differences between a query note and its neighbour set. First introduced in [141] as a means to create abstracts for journal papers, extractive summarisation is reliant upon the identification, extraction and combination of content representative sentences to summarise a document. It is applicable in domains where documents share unique technical vocabulary, such as law reports [142] or research papers with similar focus [143]. Our method of summarisation builds upon those mentioned. Given a query and a neighbour note (or set thereof), we are interested in summarising the similarities or differences. This means we are generating a summary from a list of overlapping and non-overlapping terms, as opposed to generating a summary from a full document. Essentially, we are augmenting the technical vocabulary which is highlighted by the low-level 'word overlap and scoring' mechanism and giving context to that information with free text.

From the notes, we generate a case-base of sentences which will act as summaries. Each note is divided into multiple sentences by slicing at natural end points (such as the end of a sentence, or beginning of a new topic). We transform these sentences in the same way as our full dataset, which in the case of the above problem means that their word contents have been stemmed, stop words removed and transformed using tf-idf. When the classification model is queried, we identify overlapping (or non-overlapping) words between the query and its return set. We transform this list with our representation learner to create a vector which is used to query our summarisation case-base and find the most similar sentence to act as a summary of similarities (or differences). This process is demonstrated in Figure 6.2. Words can be weighted using their idf score to emphasise rare terms and we can integrate aspects of query expansion from information retrieval research and augment queries with further information using local context.

This summarisation method produces a sentence in the engineers own words. This is useful for two reasons. Firstly, when engineers use the system it can be reassuring and trust building for them to see the difference clearly in their own words. Secondly, in the instances where non-experts are using the system, the summary of similarities and differences can expose them to language that engineers use in a controlled environment and supported by the other low-level explanation methods. This can improve learning about the original source data. However, autonomously evaluating the quality of explanations gained in this manner is traditionally difficult. In the next section, we discuss our attempt to model the quality of explanations using similarity knowledge.

6.3 Similarity Knowledge for Evaluating Explanations

Relying on user feedback means it is difficult to benchmark the usefulness of explanation methods before they are expanded to other user groups. It would be advantageous if we could identify consistent patterns of what makes a 'good' explanation. Given that the method of classification we have proposed leverages similarity knowledge to inform its decision-making, it seems reasonable to investigate the similarity between examples as a potential indicator of explanation quality. Consider for example, the above proposed methods of explanation. By leveraging knowledge of what makes two examples 'similar' (or indeed, 'dissimilar'), we are able to produce low-level and high-level explanations to help the user understand why these examples are similar. Therefore, intuition suggests that the quality of the explanation is tied to similarity. We therefore propose two novel methods of similarity-based explanation scoring which we introduce as Meet-In-The-Middle (MITM) and Trust-Your-Neighbours (TYN). Both of these metrics aim to quantitatively model the area of 'information need' that is suggested by a query. While MITM explicitly considers the relationship between a query and its neighbour set in

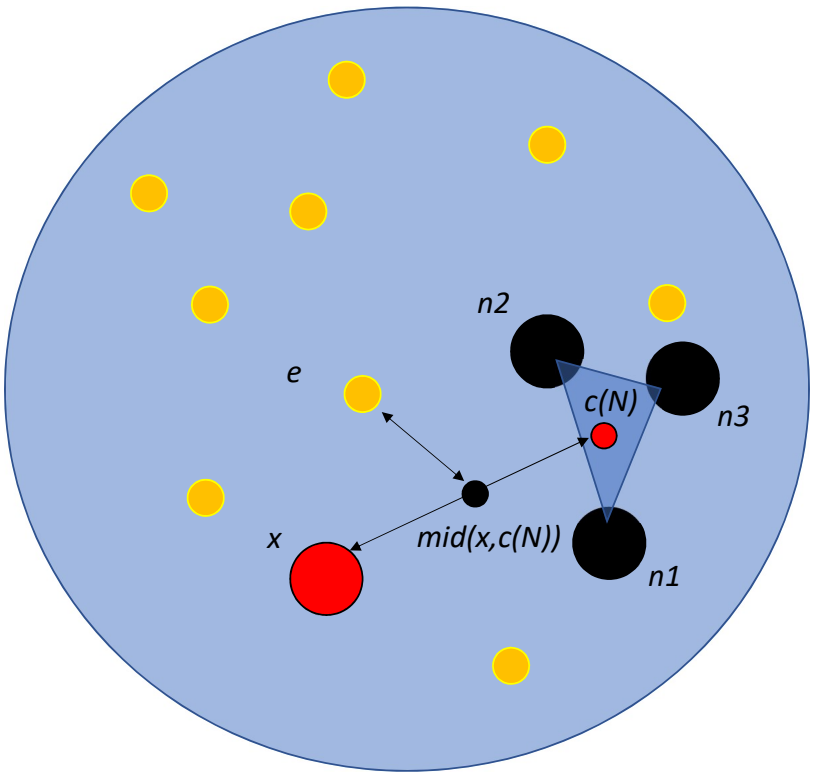


Figure 6.3: Capturing the MITM score

its scoring mechanism, TYN implicitly considers this relationship. We describe both mechanisms in more detail below.

6.3.1 Meet-In-The-Middle (MITM)

The MITM metric attempts to model the user's 'information need' from an explanation to enable scoring of whether the retrieved explanation meets that need. We propose that the information need experienced by a user in a similarity-based system is typically the understanding of the similarities and differences between the user's query and the neighbour set responsible for its classification. We base our approach on the observation that the query note, the notes used to produce the explanation (from summarisation) and the notes used to identify the classification label are all linked. We then take (and discuss through experiments) the assumption that if the meaning of a query note x and a neighbour n1 is understood by the user, then the meaning of the sentences built from the same vocabulary as x and n1 will likely be understood too. This intuition is supported by our findings from co-creation with real users. Therefore, in a latent space the representation for the information need can be hypothesised to exist within the range of values between the representation for a query and the centroid of representations for its neighbour set.

Although better understanding of each individual user is required to pinpoint where exactly within this range the specific information need may lie, we can approximate with a certain degree of accuracy by taking the midpoint to act as a proxy. By doing so, we are explicitly considering the exact point in the space which would summarise the relationship between them. The midpoint can be seen as the point where a note, were it to exist, would contain a perfect blend of the information contained within the query and the neighbour set. Therefore, this midpoint could be seen as the most appropriate summary to describe the similarities/differences between two notes. With that in mind, it is our intuition that the distance between the midpoint and the actual explanation which is retrieved, could act as a metric for the quality of the explanation.

To extract the MITM score M_s for a given query x we firstly we take the centroid of the neighbours $n \in N$ using the function $c()$ (see Equation 6.1). Using the centroid allows us to represent the neighbour set as a single point within the feature space. We can then identify the midpoint between the x and $c(N)$ (Equation 6.2). Finally, we use a distance metric D_W to measure the distance between $mid(x, c(N))$ and the retrieved explanation e (Equation 6.3).

$$c(N) = \frac{\sum_{i=1}^{|N|} n_i}{|N|} \tag{6.1}$$

$$mid(x, N) = \frac{x + c(N)}{2} \tag{6.2}$$

$$M_s = D_W(mid(x, N), e) \tag{6.3}$$

The MITM score M_s for an explanation is therefore captured as a real value. We demonstrate this process graphically in Figure 6.3.

6.3.2 Trust-Your-Neighbours (TYN)

As many of the background research works indicate, explanation knowledge is informed by the relationship between the query and the classification. With that in mind, it would seem foolish to disregard any query information in a potential metric. However, in similarity-based algorithms the neighbour set maintains some query knowledge. This occurs because the neighbour set is identified based upon the query's placement into the feature space. Therefore, query knowledge is implicitly captured. We can use that knowledge to inform the development of another metric - Trust-Your-Neighbours (TYN).

TYN follows an assumption that the information need associated with a query is less

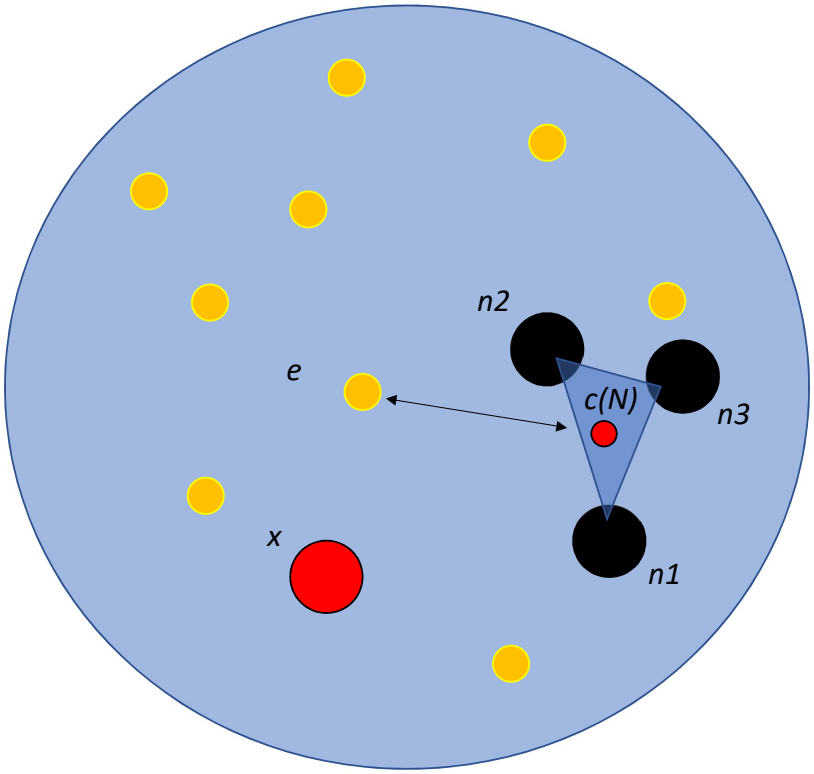


Figure 6.4: Capturing the TYN score

concerned with specific differences between the query and its neighbour set. Instead, the information need is associated with a user's inability to comprehend the region of the space into which the query has been placed. In other words, TYN measures would sit well with the assumption that the user is likely unable to understand why examples in the neighbour set are similar to each other. Therefore, a useful explanation is one which helps the user to understand the relationship between these neighbours.

To extract the TYN score T_s for a given explanation, we make some adaptation to the MITM formula. Similar to MITM, we firstly we take the centroid of the neighbours $n \in N$ using the function $c()$ (see Equation 6.4). We then use a distance metric D_W to measure the distance between the neighbour set centroid $c(N)$ and the retrieved explanation e (Equation 6.5).

$$c(N) = \frac{\sum_{i=1}^{|N|} n_i}{|N|} \tag{6.4}$$

$$T_s = D_W(c(N), e) \tag{6.5}$$

This allows us to capture the TYN score T_s for an explanation as a real value. We demonstrate this process graphically in Figure 6.4.

6.4 Evaluation

Evaluating the quality of explanations is traditionally difficult due to their inherent subjectivity. The needs of different user groups can be very different, which is reflected in their expectations of what an explanation should offer. With this in mind, we evaluate the quality of explanations using qualitative feedback from telecommunication field engineers. Technical experts were selected to identify whether explanations emulated their decision process, as requested during co-creation. We retrieved qualitative feedback on explanation quality from individual engineer comments verbally communicated during a beta test of the software. We also extracted structured feedback from desk-based agents during a pilot test of the software. This allows analysis of results from two distinct user groups and insight into two separate ways in which the system would be used.

Engineers and desk-based agents provided feedback using an expanded version of the application presented in Section 5.4. We applied the explainability framework to our recommender system from this use case, and upgraded the interface to include explanations (see Figure 6.5. The application uses the explainability framework to explain the ordered list of recommendations. We compare the query to each label by aggregating the explanations for each note within the kNN neighbourhood that possesses that label. The confidence of the recommendation of each scenario is based upon the similarity of the nearest neighbour with that label, while direct note comparisons (e.g. the word scoring and overlap) are aggregated using a distance-weighted average. The focus of this explanation is therefore linked to identifying the most suitable scenario to be recommended. This allows the system to demonstrate the similarities and differences between a query and each label by displaying the identified overlapping and non-overlapping terms ordered by their score. A sentence summary (generated by the method defined in Section 6.2.2) then contextualises this information. There is also a mechanism for the user to feedback on the explanation to improve further query note explanation mechanisms. Finally, supporting information is highlighted for the label and each query.

6.4.1 Evaluating the Explanation Framework

We measure the effectiveness of our explanation by applying the model suggested in [117]. The model divides evaluation of an explainable systems into five different headings: user satisfaction (e.g. the clarity and utility of the explanation); mental

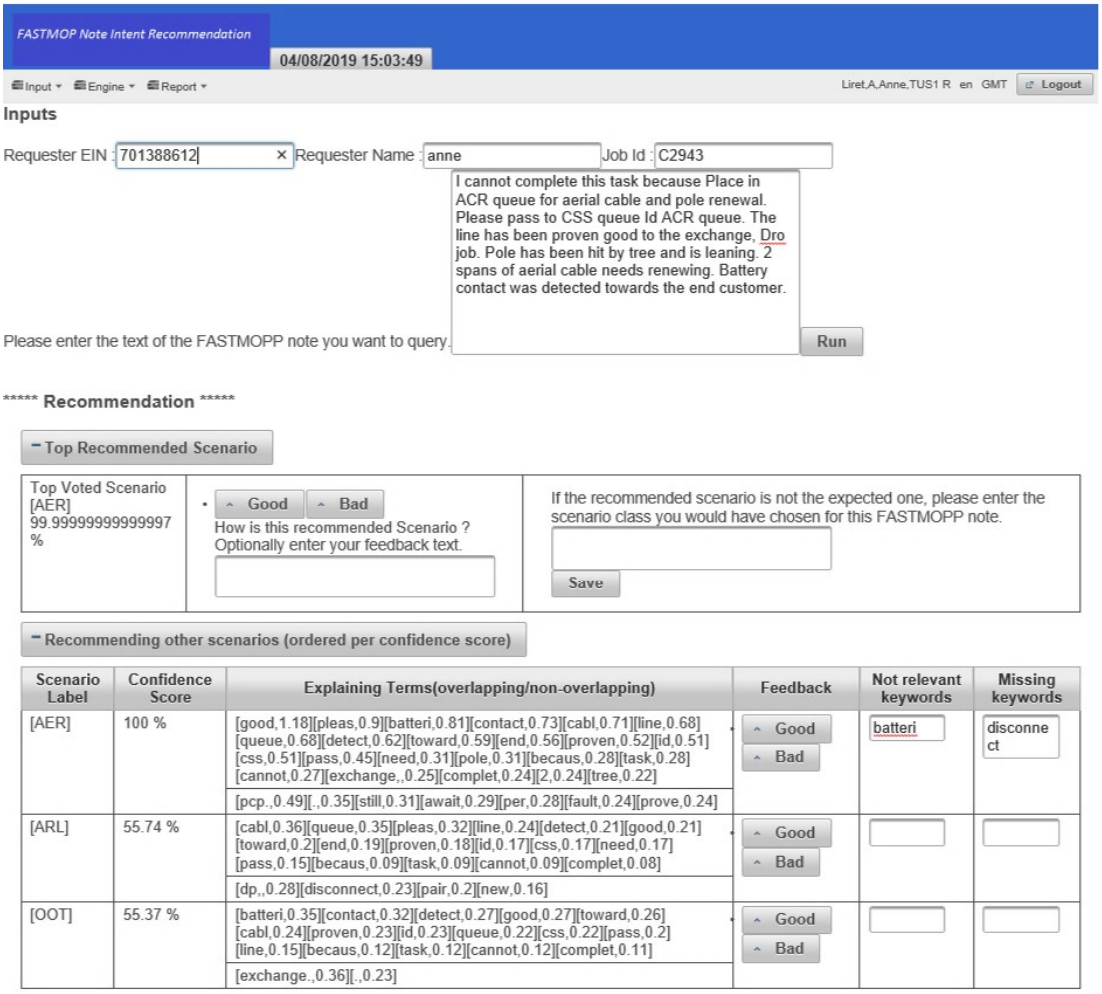


Figure 6.5: The recommender system from UC2 (see Section 5.4, supported by the explainability framework proposed in Section 6.2

model (e.g. the ability to understand individual decisions and identify strengths and weaknesses of the model); task performance (e.g. whether user ability to complete the task is improved by using the system); trust assessment (e.g. whether the system is trustable); and correctability (e.g. the user can rectify incorrect decisions). We examine each of these aspects in turn.

Results and Discussion

In total we observed 23 interactions between engineers and the system, and obtained feedback from desk-based agents for a further 30 interactions. All engineers provided a simple positive/negative score on whether the provided explanation was useful, while all desk-based agents used this score to indicate it supported them in their work. We therefore use results from engineers to measure user satisfaction, and feedback from

Table 6.1: Example of qualitative feedback on explanation quality from field engineers.

Query		Drop Wire already up at front of property but landlord wants the customer drop wire moved to the wall of the flat which is above the flat roof and to drill out where the socket is required is out to the flat roof.
Recommended Scenario (Action)		Out of Time (Re-allocate engineer)
Keywords	Overlap	[flat, 3.33] [roof, 2.76] [wire, 1.43] [drop, 1.18] [abov, 0.7]
	Non-Overlap	[one,0.29] [insid,0.29] [coil,0.28] [side.,0.27] [build,0.27]
Summary	Similarities	Drop wire is already up at front of property.
	Differences	I have fitted the socket inside and left a coil of cable.
Feedback		Roof is rightly highlighted. Fair explanation since engineer faced additional steps on the customer site (drill out)

desk-based agents to measure task performance. This distinction resembles the different scenarios in which we expect the system to be used.

Qualitative feedback was provided by 17 engineers and all 30 of the recorded interactions with desk-based agents. While engineers tended to give a feedback comment per explanation per class, desk-based agents supplied one comment to summarise their feedback on all explanations and classifications for a given scenario. This is likely due to the difference in feedback capture mechanisms. Engineers were given opportunity to share all their thoughts during a closed beta test of the software, so had time to give detailed replies. Desk-based agents on the other hand were piloting the software as part of daily work, and looking to maximise their efficiency. Furthermore, while engineers tended to focus on whether explanations justified each of the top-n recommended scenarios, desk-based agents tended to prioritise the classification of a scenario and mainly commented on whether explanations supported only the correct recommendation. As

a result, feedback from engineers is more granular and descriptive of explanation quality, while desk-based agent feedback is shorter and focuses on task performance. An example of feedback from an engineer (and the explanation case it refers to) can be seen in Table 6.1.[1]

User satisfaction with the system seems reasonably high. Of the 23 interactions with the system, 15 (65%) engineers left positive feedback regarding the explanation quality. In almost all cases (7 of 8 or 87.5%) where negative feedback was provided by engineers, the explanation was associated with an incorrect classification. This suggests that when a user discovers an error in the system decision-making, they are also likely to find a fault in its explanation of that decision. Word matching and scoring was the most popular explanation mechanism, with almost every observed engineer discussing the selected words (both formally as recorded comments and informally with the researcher). Though summaries were observed, they were not discussed in the same level of detail. This is indicative that domain experts require less contextualisation from an explanation to understand it, likely because they can infer their own context. This was an interesting (if somewhat expected) contrast to desk-based agents, who tended to prefer the retrieved sentence summaries of similarities/differences.

In 20 of the 30 recorded interactions (67%) between desk-based agents and the system, the explanation supported or improved their task performance. In 2 of these interactions, the classification was only partially correct, but the explanation supported a correct classification. This was an interesting finding, since one of the failings of the extracted dataset was that it did not demonstrate any examples where multiple scenarios could be recommended simultaneously, whereas this was a possibility in real-life. It was interesting to see that the explanation made the system more robust to this, as the retrieved sentences gave desk-agents more information to support their decision-making. Overall the cases where classification and explanation were dually complement to identify the complete expected result, has contributed to a good level of correctability with the system. In 1 instance the classification of the system was completely incorrect but the retrieved explanation supported the desk-based agent to make the correct classification.

Although both user groups understood that the model was reliant on task note vocabulary, there was a tendency to misunderstand the learned model as simple token matching. As such, engineers often criticised the lack of keywords identified for certain notes, even when they had little or no impact on model decisions. In one example, an engineer stated that ‘leaning’ and ‘tree’ should be highlighted as key words, even

[1]Full details of all feedback cannot be disclosed in this thesis due to the presence of confidential data in several of the comments.

though the word leaning is too generic to be represented by the vocabulary. Similarly, many desk-based agents would report cases of missing key words when these terms had no impact on decision-making. This is indicative that users were able to understand some aspects of system decision-making (e.g. that it was vocabulary-based), but unable to mentally model the entire system. In future work, we aim to improve this (see Section 7.1).

Although not directly related to trusting the explanation itself, something that was interesting to observe was the lack of trust that desk-based agents displayed towards the system in general. Though not recorded in structured feedback, verbal comments to the researcher indicated there was suspicion that the system was being used to audit working procedure, and this generally lowered user engagement. This was exacerbated by the feedback components because agents felt the system was aimed to assess their understanding of the job and to ensure their reasoning processes were appropriate. This was obviously not the case. Still, this feedback demonstrates a good example of how areas of the workforce feel threatened by implementation of smart automatisation, and that having an explanation component does not necessarily resolve those fears.

Our model offers a means for users to submit corrections, which was well received by engineers. Of the 23 interactions with the system, 15 (65%) engineers made use of the feedback system to highlight missing or non-relevant words and phrases. This included both engineers who had left positive feedback about an explanation, and engineers who had left negative feedback about the system, suggesting that partial explanations were able to satisfy the explanation need in some cases, but not others. Several engineers commented that they felt more comfortable with the system due to this feedback component. This suggests that correctability of an explanation is an important consideration when users are deciding whether to trust the system. This may be something that could be resolved by prolonged use of the system, allowing engineers and agents to actually experience how their feedback updates the system. We plan to explore this further in future work.

6.4.2 Similarity Knowledge for Evaluating Explanations

We investigate the relationship between explanation quality and the two proposed metrics, MITM and TYN. The purpose of this investigation is to identify whether these metrics can be used to model explanation quality. Since our explanations are classful (i.e. there is an explanation per recommended class), this is reflected in MITM and TYN. We therefore suggest 2 different metrics across with 3 unique scenarios, giving a total of 6 potential scoring metrics for evaluation of explanation quality. We therefore consider the following six metrics:

1. **MITM-B:** The distance between the midpoint of the original query note and the centroid of nearest neighbours of that class (computed from an average of the queries for *overlapping and non-overlapping keywords*) and the explanation center point (computed from an average of the representations for the returned sentences to summarise *similarity and differences*).

2. **MITM-S:** The distance between the midpoint of the original query and the centroid of nearest neighbours of that class (computed by *only considering overlapping keywords*) and the representation of the sentence used to explain *similarities*.

3. **MITM-D:** The distance between the midpoint of the original query and the centroid of nearest neighbours of that class (computed by *only considering non-overlapping keywords*) and the representation of the sentence used to explain *differences*.

4. **TYN-B:** The distance between the returned explanation (computed from an average of the representations for the returned sentences to summarise *similarity and differences*) and the centroid of nearest neighbours of that class (computed from an average of the queries for *overlapping and non-overlapping keywords*).

5. **TYN-S:** The distance between the returned explanation (computed only from the representations for the returned sentence to summarise *similarity*) and the centroid of nearest neighbours of that class (computed by *only considering overlapping keywords*)

6. **TYN-D:** The distance between the returned explanation (computed only from the representations for the returned sentence to summarise *differences*) and the centroid of nearest neighbours of that class (computed by *only considering non-overlapping keywords*).

We apply the MITM and TYN scoring metrics to three different representations for explanations from the UC2 dataset, including the statistical measure tf-idf and two DMLs (SNN and TN). To obtain representations for the explanations using DMLs, we trained an SNN and a TN using the full UC2 dataset following the results of our experimentation in Section 5.4, using tf-idf vectors as input. We then used the trained architectures to convert the tf-idf representations of the notes into the representation learned by DMLs. The explanations acted as unseen test data (which did not form part of training) and after training were converted into DML representations. Thus this enabled us to compare MITM and TYN against sparse tf-idf representations, and the dense DML representations.

We compared the output of the MITM and TYN metrics with a ground truth for explanation quality presented by telecommunication engineers. To perform this comparison, we obtained the output for the six metrics (MITM-B, MITM-S, MITM-D, TYN-B, TYN-S, TYN-D) for each of the explanations that were generated in response to real engineer queries. We were then able to compare them directly to the binary feedback of explanation usefulness as provided by these engineers. This allowed us to examine the correlation of whether these scores demonstrated any trends for predicting useful or non-useful explanations.

Results and Discussion

Firstly we consider the MITM and TYN scoring metrics when applied to tf-idf representations of the provided explanations. A visualisation of the results can be seen in the graphs in Figure 6.6. For each graph, the y-axis is distance and the x-axis is a unique identifier for each interaction. For readability we have ordered the graphs by increasing score, allowing us to identify situations where a threshold is clearly visible. In this work we use Euclidean distance to calculate similarity between examples. Analysing the results, it would appear that both summary metrics (MITM-B and TYN-B, which combine the query generated for similarities and differences) show a consistent pattern. Explanations which are very similar to the estimated point of information need (the midpoint or centroid for MITM or TYN respectively) tend not to be useful. Instead, most useful explanations (as identified by engineers) tend to exist at least a set distance away from this point. We propose this is because this offers room for 'context' to be built into the explanation. If the explanation is too similar to the exact point of information need, then it does not add the necessary additional information that would allow a user to understand what is changing. This is an interesting observation, as it suggests that explanation requires some additional elements from outside the original query or neighbour set in order to be useful. The finding is supportive of similar outcomes reported in research of counterfactual explanations.

From the results, it also seems that MITM is generally better at modelling the quality of explanations describing differences between the query and its neighbour set, while TYN is more promising for scoring explanation of similarities. Given the differences between the two metrics, this observation suggests that understanding the differences between a query and its neighbour set requires an understanding of how these are linked, which the midpoint in MITM provides. However, understanding the similarity between query and neighbour set is difficult without understanding the region of the space into which the query has been placed. We highlight this as a valuable finding which could help inform the development of explanation quality metrics in future.

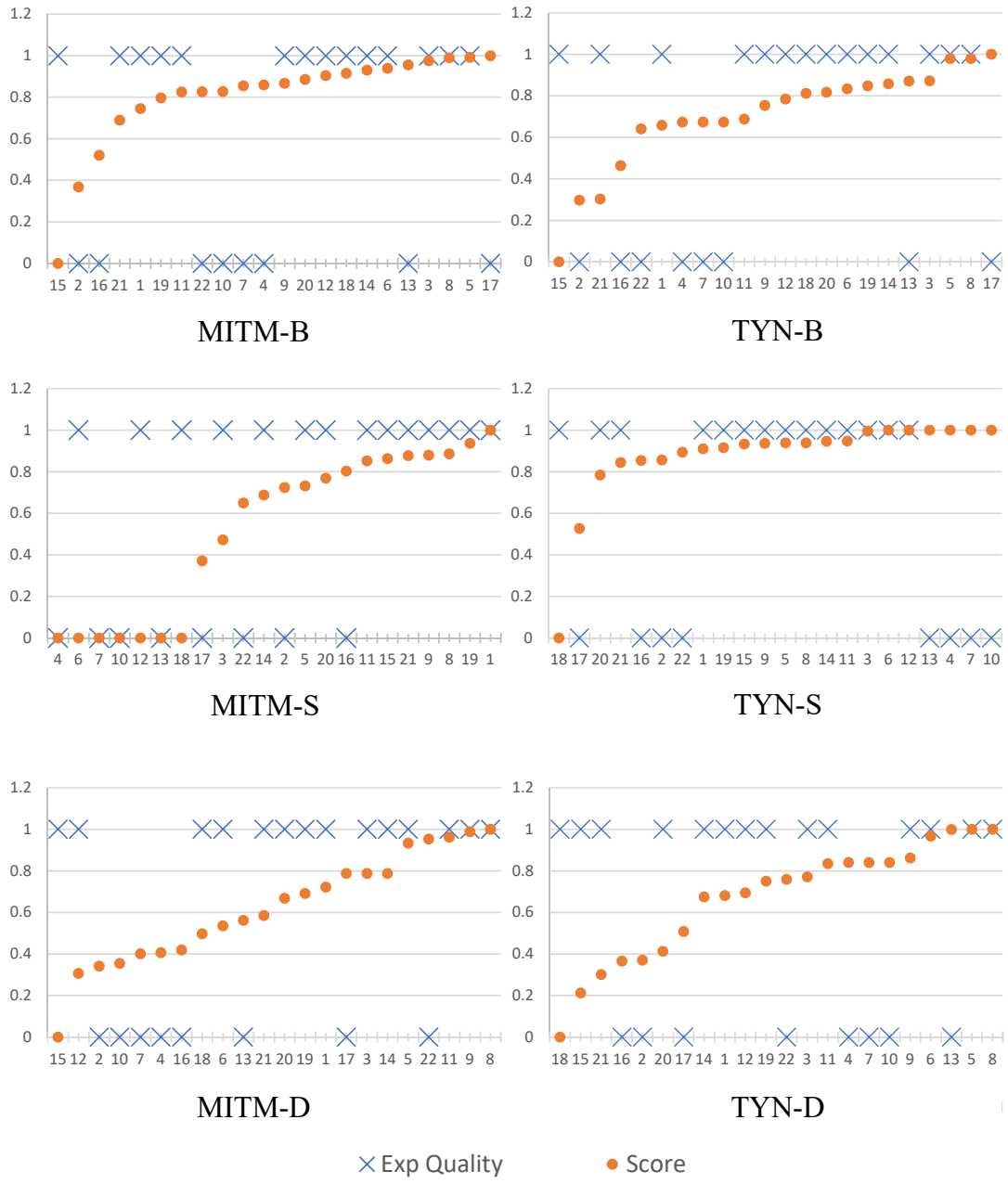


Figure 6.6: Using representations from tf-idf, a comparison of MITM and TYN scoring metrics and correlation with explanation quality.

Examining each interaction in more detail (from Figure 5), interactions 13 and 17 seem to be exceptions to these observations. For interaction 13, the engineer feedback states that the outcome and the explanation was a distinct possibility "but at this stage in the task was more likely to be a risk than a definite outcome". So the explanation was valid, but could not leverage temporal information to better inform its findings - something which could be targeted in future. For interaction 17, both of the summary metrics

(MITM-B and TYN-B) allocate it a very high score. The interaction involved the system attempting to explain an incorrect classification. This suggests that interaction 17 involved a query which was allocated to a sparse region in the latent space, which may offer some reasoning as to why it does not follow general trends of other interactions.

Another outlier is interaction 15, an interaction where a useful explanation was experienced by the user. By almost all metrics this interaction is judged to be very similar to the identified point of information need. For other interactions this has typically been an indicator of non-useful explanation. This suggests that there is room for simply rewording statements and have them act as useful explanation, but the situations in which this is useful will be rarer.

If we consider the results of MITM and TYN as applied to the output of DML architectures, then we notice a considerable change to our insights (See Figures 6.7 and 6.8). For both DML architectures, the summary metrics MITM-B and TYN-B do not seem to demonstrate a consistent pattern. This is indicative that the summary metrics are incompatible with DMLs, potentially because of the way in which these architectures are trained. If we consider how the summary metrics are calculated, the representation for the explanation is formed from a list of overlapping and non-overlapping keywords as identified from the user's original query, and the neighbour set used in its classification. This means we collect information from several different sources. When this is converted to a tf-idf vector, the impact is localised to specific features and the vector is still sparse. Even if it is unlikely for the features to appear together, this means that the impact is relatively minimal as the latent space is also sparse. However when this is applied to DMLs, the combination of features are converted into a dense representation. The learned weight matrices may not have seen similar combinations of features during training, which would have an impact on the output representation. This could explain the lack of pattern in the summary metrics, though we will study this further.

Looking at the representations gained from an SNN, in most circumstances a lower MITM or TYN score is associated with more useful explanations. This is particularly noticeable in MITM-D, where most non-useful explanations obtained a normalised score greater than 0.7, and only a single useful explanation (19) was above this threshold. This outlier may be explained by the fact that explanation 19 was associated with an incorrect classification, where the engineer highlighted that it lacks keywords from the text, and as a result was only 'somewhat useful'. A similar, though less pronounced, relationship is demonstrated in TYN-D. While this pattern is not replicated exactly in the representations gained from TN, we can see indicators that this pattern is beginning to form (particularly once again in MITM-D and TYN-D).

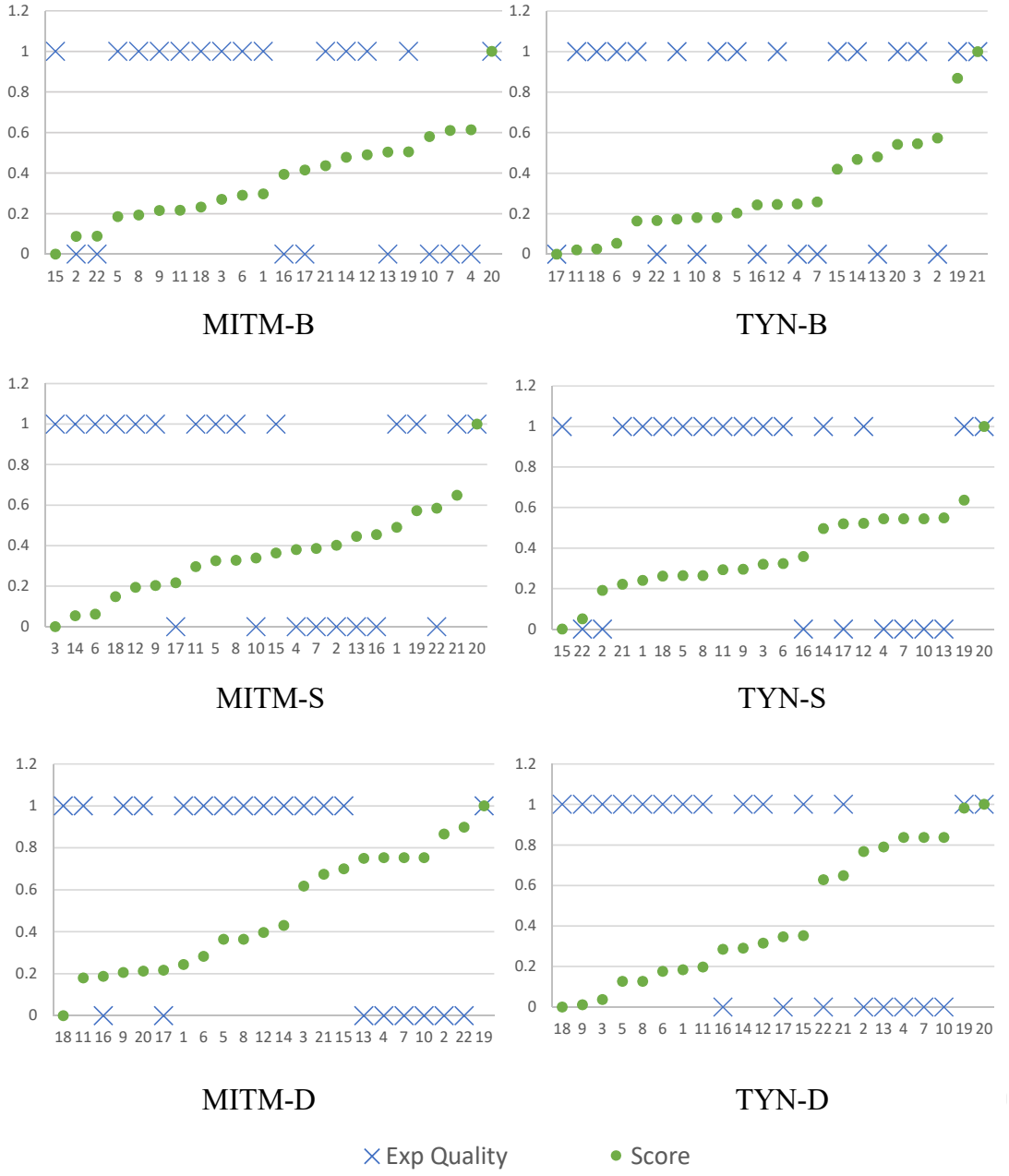


Figure 6.7: Using representations from SNN, a comparison of MITM and TYN scoring metrics and correlation with explanation quality.

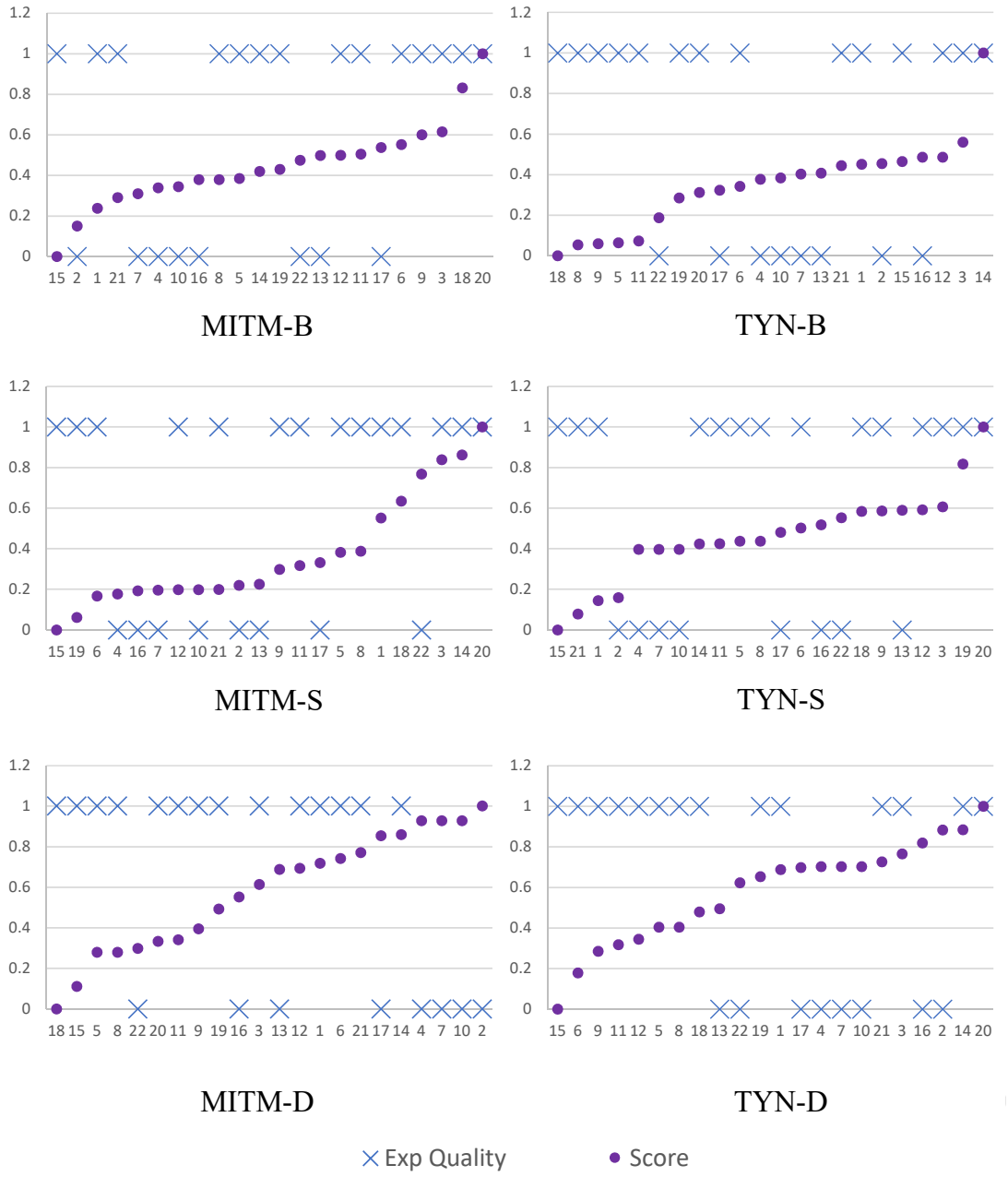


Figure 6.8: Using representations from TN, a comparison of MITM and TYN scoring metrics and correlation with explanation quality.

Overall we take this as indicative that latent spaces which are learned by leveraging similarity knowledge between examples will be more responsive to similarity-based scoring mechanisms. Specifically, the matching knowledge learned by SNN architectures seems to be well aligned with the MITM and TYN metrics we have suggested in this chapter when comparing explanations derived from similarities and differences between the query and the neighbour set used for classification. We plan to pursue this even further in future work, with the goal of deriving a meaningful and accurate method of autonomously evaluating explanations for similarity-driven machine learning algorithms.

6.5 Conclusions

In this chapter, we have explored how we can explain the output of similarity-based architectures, including DMLs, to multiple user groups. To that end, we have described the development of a framework to promote explainability of machine learning methods within a telecommunication organisation. We have motivated and explored the application of this framework to the specific use case of explaining technical engineer notes to non-technical planning personnel. An evaluation of this framework over two distinct user groups, engineers and desk-based agents, demonstrates several key differences between them which impacts how they use the system. In particular it is interesting to note the different ways in which these two groups judge the quality of an explanation. For engineers, it is about whether the explanation follows their reasoning, while desk-based agents are more concerned with whether it supports their work. Overall, we believe the feedback from both stakeholder groups highlights the utility of similarity-based architectures as explainable machine learners for the task of transfer of experience.

Beyond this, we have also investigated the relationship between similarity and explanation quality by introducing the two metrics MITM and TYN. Overall, these metrics seem to indicate that similarity and explanation quality do share a relationship, but that it is quite complex - an explanation cannot simply be described as 'good' if it is within the locality of the query. A deeper understanding of the explanation is required (for example, is it describing similarities or differences between a query and an outcome) is required to understand the most appropriate method to leverage similarity knowledge as a component of a metric. In this regard, further exploration is required to fully understand how we can autonomously evaluate explanations with similarity knowledge.

Chapter 7

Conclusion

In this thesis, we presented our hypothesis that similarities between CBR and DMLs present an opportunity for integration where both methods will benefit. Specifically we anticipate that training of DML architectures can be improved by considering clustering research from other machine learning techniques, and that DMLs present an opportunity to combine the similarity and vocabulary knowledge containers as a component of a CBR system. Furthermore, as DMLs are fundamentally similarity-based architectures we believe that their output can be explained effectively in situations where multiple user groups of varying domain expertise are using the system.

To ensure systematic investigation of these claims, we identified three research questions:

1. How can techniques from traditional machine learning methods (such as CBR and meta-learning) be incorporated into strategies to improve training efficiency of DMLs?

2. How effective are DMLs at fulfilling the traditionally separate roles of the 'vocabulary' and 'similarity' knowledge containers in the context of transfer of experience between experts and non-experts of telecommunications engineering?

3. How can we explain the output of similarity-based architectures (including DMLs) intended to support user groups of varying domain expertise, and how can we autonomously evaluate the quality of produced explanations?

We systematically explored these research questions across Chapters 4, 5 and 6 of this thesis and presented the following contributions:

1. We introduce several training strategies for DMLs which are inspired by research

in meta-learning, curriculum learning and CBR. Experiments on public datasets from multiple domains illustrate that the proposed strategies improve training efficiency of DML architectures.

2. We compare methods of developing a similarity model for transfer of experience using free-text data sources. Our findings demonstrate that DMLs can learn to produce representations optimised for similarity calculations which offer clear improvement over dense representations gained from word embeddings, but require refinement to outperform statistical methods.

3. We describe the development of an explainability framework based upon one of our use-cases, and assess the quality of these explanations using novel autonomous evaluation methods and user feedback. The results highlight the practical utility of a hierarchical explanation framework.

In Chapter 4 we presented several novel training strategies for SNNs and TNs which leveraged previous research in meta-learning and CBR respectively. Initial experimentation with SNNs trained using our proposed Dynamic Exploration (DynE) and Dynamic Explore and Exploit (DynEE) strategies to leverage exploration and exploitation knowledge presented promising results across several public datasets, but expansion to more complex problems and DML algorithms was limited by the high complexity of the DynEE algorithm. This complexity was primarily caused by the expensive nearest neighbour calculations to identify the exploitation set. To resolve this, we were inspired by Locality-Sensitive Hashing (LSH) from CBR research to develop an inexpensive training strategy for TNs. The Locality-Sensitive Batching (LSB) strategy was capable of inexpensively dividing the latent space into a number of buckets to capture the locality of complex regions of the space. By randomly creating triplets from within complex buckets to inform our exploitation knowledge, and joining pure buckets for triplet generation as exploration knowledge, we were able to maintain the concepts of exploration and exploitation whilst reducing the computational cost. Results across several problem domains demonstrate the effectiveness of using similarity-based techniques (such as boosting from meta-learning, and LSH from CBR) to improve the training of DMLs.

In Chapter 5 we applied DMLs to the problem of experience transfer for service provisioning within telecommunications using expert-written task notes as a data source. We divided this problem into two use cases: UC1, where we learned similarity models with the intention of enabling transfer of experience between expert engineers by recommending additional information to support task completion; and UC2, where we created a similarity-based recommender system to enable transfer of experience

from engineers to non-expert desk staff. On both of these use cases we performed a comparative study to understand the best method to represent the data for similarity calculations, comparing tf-idf, Doc2Vec and several DML architectures. Our findings indicated a surprising reliance on specific vocabulary to inform classification, evidenced by the strong performance of tf-idf vectors in both experiments. However, in both use cases we could also observe that DMLs contributed to better clustering of the data, particularly noticeable on dense Doc2Vec representations. We use this as evidence of our second contribution in this thesis, and highlight that DMLs can bridge the gap between the traditionally separate vocabulary and similarity knowledge containers. However, we are also motivated to suggest further work in this area in Section 7.1.

Finally, in Chapter 6 we presented our work towards an explanation framework. The goal of this framework is to interface with a range of machine learning applications within our company partner, but in this work we have focused on developing explanation strategies to support similarity-based architectures such as that in UC2. Motivated by literature around explanation types as an influence, we suggest both low-level explanations (confidence and term overlap) as well as a high-level explanation (extractive summarisation of similarities and differences) which leverage similarity knowledge to inform their explanations. Feedback from both expert engineers and non-expert desk-based staff indicate that the explanations are useful to support their work, highlighting the utility of explaining the output of similarity-based architectures to support stakeholder groups of multiple experience levels interacting with the same machine learning system. Furthermore, we have investigated similarity information as a means to autonomously judge the quality of explanations, and introduced two novel metrics, Meet-In-The-Middle (MITM) and Trust-Your-Neighbours (TYN), for this purpose. Preliminary results on a task with user feedback on the quality of explanations suggests that there is a relationship between similarity and explanation quality. This is particularly noticeable when using representations obtained from SNNs, where our results are indicative that good explanations to explain similarities and differences between a query and its neighbour set generally have lower MITM and TYN scores. We view this as a good sign that the matching knowledge learned by SNNs helps to identify high-quality explanations. We take this as evidence that similarity knowledge is a useful starting point to study autonomous evaluation of explanation quality, but highlight that further work is required to truly understand this relationship. We propose several avenues in Section 7.1.

Overall, we consider the contributions we have made in this thesis as a good indication that our hypothesis is correct. Our findings have demonstrated that there are genuine benefits for both CBR systems and DML architectures when knowledge from one field

of research is applied to the other. For DMLs, we have demonstrated this is the case by applying techniques from meta-learning and CBR to build training strategies which outperform baseline methods. For CBR, we have demonstrated the ability to create a robust similarity model for a real world problem by leveraging DMLs to learn a representation. Finally, we have demonstrated the explainability of decisions produced using similarity-based methods with real users, and highlighted the capability of metrics for autonomous evaluation of explanation quality using similarity knowledge. With these factors in mind, we propose that our hypothesis is supported by the findings of this thesis.

7.1 Future Work

We have identified several avenues for further research from this thesis. We conclude this thesis with some thoughts on each of our contribution chapters and how they can be further developed.

Considering the training strategies we have suggested in Chapter 4, we feel that the results we have obtained on public datasets are good evidence that DML training strategies able to mix exploration of the space and exploitation of complex areas are a promising research avenue. An area we discussed in some length was the balance between exploration and exploitation, and how over emphasis on one of these concepts was detrimental to strategy effectiveness. It would be interesting to explore how the appropriate ratio of exploration and exploitation changes over the course of training through decay parameters. Somewhat tangential information from curriculum learning research highlights that it is ill advised to start training with complex examples, so our intuition is that we would like a training strategy where the focus gradually shifts from exploration towards exploitation. Therefore we very much view our contributions in this chapter as a first step towards better understanding the role of these concepts for training similarity-based architectures in future.

In Chapter 5 we highlighted that we suspected that the sparseness of the tf-idf vectors played some role in their improvements over performance achieved by DMLs in specific neighbourhood sizes. We are inspired by this to consider whether we could develop a strategy which would better enable DMLs to process very sparse vectors. Such a contribution would likely evolve from work in case completion from CBR, once more highlighting the synergy of these these two areas. If we were to pursue this avenue, it would be our goal to enable DMLs to learn similarity between incomplete cases, or cases with a minimal number of non-zero features. Furthermore, we noted that the DML architectures (and the SNN in particular) was better able able to divide complex regions of the space, due to their focus on throughout training on the concept of

'matching'. This implies that DMLs would be more capable in tasks which conventional CBR systems found difficult, such as domains which require aggressive clustering of examples as differences between cases are very small (such as anomaly detection in medical imaging). One manner in which this could be achieved is by developing a training strategy which could leverage knowledge of specific features to identify areas of complexity in the space (potentially in a semi-supervised capacity). In doing so, the exploitation we have proposed in this thesis would become better adapted to fine-grained problems where differences between specific features are important. These contributions would improve DMLs as a viable architecture to fulfill the similarity and knowledge containers of a CBR system.

Finally, we are eager to continue to explore the role of similarity to provision and evaluate explanations of machine learning architecture decisions, as we have begun in Chapter 6. We see a good amount of evidence to support that similarity knowledge is important to achieve this, both from literature and from our experiences over the course of this thesis. In future work, we plan to extend the framework to incorporate explanations which acknowledge sequential and co-occurring scenarios, as these are necessary concepts for full automation. We also aim to apply this framework to further use cases, enabling us to better understand the explanation needs of users from different work types and experience levels. We are also specifically interested in improving users ability to mental model the decision-making process of machine learning systems. In the use case we presented, the produced explanations primarily targeted feature relevance - highlighting to engineers and desk-based agents why the recommended solution was appropriate by highlighting the overlapping and non-overlapping features between the query and its neighbour set. We had originally imagined this would improve their ability to mentally model the system, and by understanding why the system viewed two cases to be similar would be sufficient. However, while the users understood that vocabulary played a role in system decision-making, they failed to understand that these words played only a minor role. Therefore, it would be useful to develop an explanation to holistically describe the similarity-based return process, and prompt users to understand that features exist within context of each other for the purposes of decision-making. We hope this would also improve ability to trust of the system.

Furthermore, we highlight autonomous scoring of explanation as an area of interest. Lack of convenient scientific method to evaluate explanation quality remains a barrier to explainability research. Inherently, similarity-based metrics, and particularly those we have proposed here, assume that the point of explanation need can be modelled in the same feature space as learned by a classifier. This assumption may not hold true in every scenario requiring an explanation - for example, situations where an

explanation is provided in a different data type from the query (i.e. text captioning of an image), or situations where the explanation need cannot be modelled at all (such as when the user's explanation need is caused by the context in which the system is being used). Furthermore, the individual metrics themselves possess limitations. For example, MITM assumes that the user has a good understanding of the query. This can be problematic in situations where the user does not fully understand what they are asking of the system. Similarly, when using TYN the centroid of neighbours may in fact be very similar to the query, which could interfere with the quality of the metric. Lastly, our evaluation highlighted that 'timeliness' of an explanation is an important aspect which is not easily modelled with similarity knowledge. With these factors in mind, it seems sensible to propose that similarity may be useful for modelling aspects of the quality of an explanation as part of a set of features. The idea creating features for an explanation (i.e. aspects of similarity knowledge, temporal information such as the timeliness in which it was provisioned and contextual information about the situation in which an explanation is provided) is appealing, as it would allow more complex analysis of important features to dictatet explanation quality.

Bibliography

[1] Walsh JP, Ungson GR. Organizational Memory. The Academy of Management Review. 1991;16(1):57–91.

[2] Ackerman MS, Pipek V, Wulf V. Sharing expertise: Beyond knowledge management. MIT press; 2003.

[3] Dörner C, Pipek V, Won M. Supporting Expertise Awareness: Finding out What Others Know. In: Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology. CHIMIT '07. ACM; 2007. .

[4] Göker MH, Thompson C, Arajärvi S, Hua K. The PwC Connection Machine: An Adaptive Expertise Provider. In: Roth-Berghofer TR, Göker MH, Güvenir HA, editors. Advances in Case-Based Reasoning. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 549 – 563.

[5] Wright G, Ayton P. Eliciting and modelling expert knowledge. Decision Support Systems. 1987;3(1):13–26.

[6] Doyle D, Tsymbal A, Cunningham P. A review of explanation and explanation in case-based reasoning. Trinity College Dublin, Department of Computer Science; 2003.

[7] Roth-Berghofer TR. Explanations and Case-Based Reasoning: Foundational Issues. In: Funk P, González Calero PA, editors. Advances in Case-Based Reasoning. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004. p. 389 – 403.

[8] Aamodt A, Plaza E. Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications. 1994 Mar;7(1):39 – 59.

[9] Leake DB. Case-Based Reasoning: Experiences, Lessons and Future Directions. Cambridge, MA, USA: MIT Press; 1996.

[10] Slade S. Case-based reasoning: A research paradigm. AI magazine. 1991;12(1):42–42.

[11] Watson I. An introduction to case-based reasoning. In: UK Workshop on Case-Based Reasoning. Springer; 1995. p. 1–16.

[12] Sørmo F, Cassens J, Aamodt A. Explanation in Case-Based Reasoning - Perspectives and Goals. Artificial Intelligence Review. 2005;24(2):109 – 143.

[13] Richter MM, Michael M. Knowledge Containers. Readings in Case-Based Reasoning Morgan Kaufmann Publishers. 2003.

[14] Bromley J, Guyon I, LeCun Y. Signature Verification Using a 'Siamese' Time Delay Neural Network. International Journal of Pattern Recognition and Artificial Intelligence. 1993 August;7(4):669 – 688.

[15] Chopra S, Hadsell R, LeCun Y. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In: Proc. of the 2005 IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition. CVPR '05. Washington, DC, USA: IEEE Computer Society; 2005. p. 539 – 546.

[16] Hoffer E, Ailon N. Deep Metric Learning Using Triplet Network. In: Feragen A, Pelillo M, Loog M, editors. Similarity-Based Pattern Recognition. Cham: Springer International Publishing; 2015. p. 84 – 92.

[17] Koch G, Zemel R, Salakhutdinov R. Siamese Neural Networks for One-Shot Image Recognition. In: Deep Learning Workshop. ICML '15; 2015. .

[18] Schroff F, Kalenichenko D, Philbin J. FaceNet: A Unified Embedding for Face Recognition and Clustering. CoRR. 2015;abs/1503.03832.

[19] Roth-Berghofer TR, Cassens J. Mapping Goals and Kinds of Explanations to the Knowledge Containers of Case-Based Reasoning Systems. In: Muñoz-Ávila H, Ricci F, editors. Case-Based Reasoning Research and Development. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 451–464.

[20] Sørmo F, Cassens J. Explanation goals in case-based reasoning. In: Proceedings of the ECCBR; 2004. p. 165–174.

[21] Lipton ZC. The mythos of model interpretability. arXiv preprint arXiv:160603490. 2016.

[22] Mohseni S, Zarei N, Ragan ED. A Survey of Evaluation Methods and Measures for Interpretable Machine Learning. arXiv preprint arXiv:181111839. 2018.

[23] Mohamed A, Bilgin A, Liret A, Owusu G. Fuzzy Logic Based Personalized Task Recommendation System for Field Services. In: Bramer M., Petridis M. (eds) Artificial Intelligence XXXIV. SGAI 2017.Lecture Notes in Computer Science, vol 10630. Cambridge, UK: Springer, Cham; 2017. p. 300–312.

[24] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86(11):2278–2324.

[25] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. Citeseer; 2009.

[26] Coates A, Ng A, Lee H. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In: Gordon G, Dunson D, DudÃk M, editors. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. vol. 15 of Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR; 2011. p. 215–223.

[27] Sani S, Wiratunga N, Massie S, Cooper K. SELFBACK - Activity Recognition for Self-management of Low Back Pain. In: Research and Development in Intelligent Systems XXXIII. SGAI '16. Cham, Switzerland: Springer Nature; 2016. p. 281 – 294.

[28] Maas AL, Daly RE, Pham PT, Huang D, Ng AY, Potts C. Learning Word Vectors for Sentiment Analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics; 2011. p. 142–150.

[29] The Reuters Collection; 1995. Available from: http://www.research.att.com/~lewis/reuters21578.html.

[30] Chiu C. A case-based customer classification approach for direct marketing. Expert Systems with Applications. 2002;22(2):163–168.

[31] Bousbahi F, Chorfi H. MOOC-Rec: a case based recommender system for MOOCs. Procedia-Social and Behavioral Sciences. 2015;195:1813–1822.

[32] De Mantaras RL, McSherry D, Bridge D, Leake D, Smyth B, Craw S, et al. Retrieval, reuse, revision and retention in case-based reasoning. The Knowledge Engineering Review. 2005;20(3):215–240.

[33] Coyle L, Doyle D, Cunningham P. Representing Similarity for CBR in XML. In: European Conference on Case-Based Reasoning. Springer; 2004. p. 119–127.

[34] Wilke W, Vollrath I, Althoff KD, Bergmann R. A framework for learning adaptation knowledge based on knowledge light approaches. In: Proceedings of the fifth german workshop on case-based reasoning; 1997. p. 235–242.

[35] Leake D, Wilson M. How many cases do you need? assessing and predicting case-base coverage. In: International Conference on Case-Based Reasoning. Springer; 2011. p. 92–106.

[36] Smyth B, Keane MT. Retrieval and adaptation in Déja Vu, a case-based reasoning system for software design. In: Adaptation of Knowledge for Reuse: AAAI Fall Symposium; 1995. p. 228–240.

[37] Ganesan D, Chakraborti S. An empirical study of knowledge tradeoffs in case-based reasoning. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence; 2018. p. 1817–1823.

[38] Sánchez D, Batet M, Isern D, Valls A. Ontology-based semantic similarity: A new feature-based approach. Expert systems with applications. 2012;39(9):7718–7728.

[39] Hayes C, Cunningham P, Doyle M. Distributed cbr using xml. In: Proceedings of the KI-98 Workshop on Intelligent Systems and Electronic Commerce; 1998. .

[40] Wilson DC, Bradshaw S. CBR textuality. Expert Update. 2000;3(1):28–37.

[41] Amin K, Kapetanakis S, Althoff KD, Dengel A, Petridis M. Answering with cases: a CBR approach to deep learning. In: International Conference on Case-Based Reasoning. Springer; 2018. p. 15–27.

[42] Xue X, Zhou Z. Distributional Features for Text Categorization. IEEE Transactions on Knowledge and Data Engineering. 2009;21(3):428–442.

[43] Ferrone L, Zanzotto FM. Symbolic, Distributed and Distributional Representations for Natural Language Processing in the Era of Deep Learning: a Survey. CoRR. 2017;abs/1702.00764. Available from: http://arxiv.org/abs/1702.00764.

[44] Rajaraman A, Ullman JD. Mining of Massive Datasets. Cambridge University Press; 2011.

[45] Ramos J. Using tf-idf to determine word relevance in document queries. In: Proceedings of the first instructional conference on machine learning; 2003. p. 133 – 142.

[46] Mikolov T, Chen K, Corrado G, Dean J. Efficient Estimation of Word Representations in Vector Space. CoRR. 2013;abs/1301.3781.

[47] Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J. Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems; 2013. p. 3111–3119.

[48] Pennington J, Socher R, Manning CD. Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP); 2014. p. 1532–1543.

[49] Bojanowski P, Grave E, Joulin A, Mikolov T. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics. 2017;5:135–146.

[50] Le Q, Mikolov T. Distributed Representations of Sentences and Documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. ICML'14. JMLR.org; 2014. p. II–1188–II–1196.

[51] Lau JH, Baldwin T. An empirical evaluation of doc2vec with practical insights into document embedding generation. arXiv preprint arXiv:160705368. 2016.

[52] Bergmann R. Introduction to case-based reasoning. Centre for Learning Systems and Applications, University of Kaiserslautern; 1998.

[53] Bergmann R. On the use of taxonomies for representing case features and local similarity measures. 1998.

[54] Gabel T. On the use of vocabulary knowledge for learning similarity measures. In: Biennial Conference on Professional Knowledge Management/Wissensmanagement. Springer; 2005. p. 272–283.

[55] Richter MM. Similarity. In: Case-Based Reasoning on Images and Signals. Springer; 2008. p. 25–90.

[56] Recio JA, Díaz-Agudo B, Gómez-Martín MA, Wiratunga N. Extending jCOLIBRI for textual CBR. In: International Conference on Case-Based Reasoning. Springer; 2005. p. 421–435.

[57] Bogaerts S, Leake D. Facilitating CBR for incompletely-described cases: distance metrics for partial problem descriptions. In: European Conference on Case-Based Reasoning. Springer; 2004. p. 62–76.

[58] De S, Chakraborty B. Case-Based Reasoning (CBR)-Based Anemia Severity Detection System (ASDS) Using Machine Learning Algorithm. In: International Conference on Advanced Machine Learning Technologies and Applications. Springer; 2020. p. 621–632.

[59] Gu D, Liang C, Zhao H. A case-based reasoning system based on weighted heterogeneous value distance metric for breast cancer diagnosis. Artificial intelligence in medicine. 2017;77:31–47.

[60] Beyer K, Goldstein J, Ramakrishnan R, Shaft U. When is "nearest neighbor" meaningful? In: International conference on database theory. Springer; 1999. p. 217–235.

[61] Yixin Chen, Wang JZ, Krovetz R. CLUE: cluster-based retrieval of images by unsupervised learning. IEEE Transactions on Image Processing. 2005 Aug;14(8):1187–1201.

[62] Altingovde IS, Ozcan R, Ocalan HC, Can F, Ulusoy O. Large-scale Cluster-based Retrieval Experiments on Turkish Texts. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '07. New York, NY, USA: ACM; 2007. p. 891–892. Available from: http://doi.acm.org/10.1145/1277741.1277961.

[63] Lucca MR, Junior AGL, Freitas EP, Silva LA. A Case-Based Reasoning and Clustering Framework for the Development of Intelligent Agents in Simulation Systems. In: The Thirty-First International Flairs Conference; 2018. .

[64] Muja M, Lowe DG. Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1). 2009;2(331-340):2.

[65] Kulis B, Grauman K. Kernelized locality-sensitive hashing for scalable image search. In: Computer Vision, 2009 IEEE 12th International Conference on. IEEE; 2009. p. 2130–2137.

[66] Wiratunga N, Craw S, Massie S. Index Driven Selective Sampling for CBR. In: Proceedings of the Fifth International Conference on Case-Based Reasoning. Springer; 2003. p. 637–651.

[67] Slaney M, Casey M. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. IEEE Signal processing magazine. 2008;25(2):128–131.

[68] Paulevé L, Jégou H, Amsaleg L. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. Pattern recognition letters. 2010;31(11):1348–1358.

[69] Bareiss ER, Porter BW, Wier CC. Protos: an exemplar-based learning apprentice. International Journal of Man-Machine Studies. 1988;29(5):549 – 561. Available from: http://www.sciencedirect.com/science/article/pii/S0020737388800129.

[70] Smyt B, McKenna E. Footprint-based retrieval. In: International Conference on Case-Based Reasoning. Springer; 1999. p. 343–357.

[71] Gierl L, Bull M, Schmidt R. CBR in Medicine. In: Case-Based Reasoning Technology. Springer; 1998. p. 273–297.

[72] Schmidt R, Gierl L. Experiences with prototype designs and retrieval methods in medical case-based reasoning systems. In: European Workshop on Advances in Case-Based Reasoning. Springer; 1998. p. 370–381.

[73] Madhusudan T, Zhao JL, Marshall B. A case-based reasoning framework for workflow model management. Data & Knowledge Engineering. 2004;50(1):87–115.

[74] Rosch E. Prototype classification and logical classification: The two systems. New trends in conceptual representation: Challenges to Piaget's theory. 1983:73–86.

[75] Berlemont S, Lefebvre G, Duffner S, Garcia C. Siamese neural network based similarity metric for inertial gesture classification and rejection. In: 11th IEEE International Conf. and Workshops on Automatic Face and Gesture Recognition. Washington, DC, USA: IEEE Computer Society; 2015. p. 1–6.

[76] Berlemont S, Lefebvre G, Duffner S, Garcia C. Class-balanced siamese neural networks. Neurocomputing. 2018;273:47–56.

[77] Vinyals O, Blundell C, Lillicrap T, kavukcuoglu k, Wierstra D. Matching Networks for One Shot Learning. In: Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R, editors. Advances in Neural Information Processing Systems 29. Curran Associates, Inc.; 2016. p. 3630–3638.

[78] Wang J, Song Y, Leung T, Rosenberg C, Wang J, Philbin J, et al. Learning Fine-Grained Image Similarity with Deep Ranking. In: Proc. of the 2014 IEEE Conf. on Computer Vision and Pattern Recognition. CVPR '14. Washington, DC, USA: IEEE Computer Society; 2014. p. 1386 – 1393.

[79] Neculoiu P, Versteegh M, Rotaru M. Learning Text Similarity with Siamese Recurrent Networks. In: Rep4NLP@ACL; 2016. .

[80] Kaya M, Bilge HŞ. Deep metric learning: A survey. Symmetry. 2019;11(9):1066.

[81] Liao W, Ying Yang M, Zhan N, Rosenhahn B. Triplet-based deep similarity learning for person re-identification. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 385–393.

[82] Hermans A, Beyer L, Leibe B. In defense of the triplet loss for person re-identification. arXiv preprint arXiv:170307737. 2017.

[83] Liu Y, Huang C. Scene Classification via Triplet Networks. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. 2018 Jan;11(1):220–237.

[84] Sani S, Wiratunga N, Massie S, Cooper K. Personalised Human Activity Recognition Using Matching Networks. In: Cox MT, Funk P, Begum S, editors. Case-Based Reasoning Research and Development. Cham: Springer International Publishing; 2018. p. 339–353.

[85] Wijekoon A, Wiratunga N, Sani S. Zero-shot learning with matching networks for open-ended human activity recognition. 2018.

[86] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT press; 2016.

[87] Li Y, Tian X, Shen X, Tao D. Classification and Representation Joint Learning via Deep Networks. In: IJCAI; 2017. p. 2215–2221.

[88] Yang J, Zou H, Zhou Y, Xie L. Learning gestures from wifi: A siamese recurrent convolutional architecture. IEEE Internet of Things Journal. 2019;6(6):10763–10772.

[89] Cheng D, Gong Y, Zhou S, Wang J, Zheng N. Person re-identification by multi-channel parts-based cnn with improved triplet loss function. In: Proceedings of the iEEE conference on computer vision and pattern recognition; 2016. p. 1335–1344.

[90] Wang J, Zhou F, Wen S, Liu X, Lin Y. Deep metric learning with angular loss. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 2593–2601.

[91] Ge W, Huang W, Dong D, Scott MR. Deep Metric Learning with Hierarchical Triplet Loss. CoRR. 2018;abs/1810.06951. Available from: http://arxiv.org/abs/1810.06951.

[92] Rippel O, Paluri M, Dollar P, Bourdev L. Metric Learning with Adaptive Density Discrimination; 2015.

[93] Yu B, Tao D. Deep metric learning with tuplet margin loss. In: Proceedings of the IEEE International Conference on Computer Vision; 2019. p. 6490–6499.

[94] Musgrave K, Belongie S, Lim SN. A Metric Learning Reality Check; 2020.

[95] Bengio Y, Louradour J, Collobert R, Weston J. Curriculum Learning. In: Proc. of the 26th Annual International Conf. on Machine Learning. ICML '09. New York, NY, USA: ACM; 2009. p. 41 – 48.

[96] Loshchilov I, Hutter F. Online Batch Selection for Faster Training of Neural Networks. In: ICLR Workshops. ICLR '16; 2016. .

[97] Shapire RE. The Boosting Approach to Machine Learning: An Overview. In: Nonlinear Estimation and Classification. Lecture Notes in Statistics, vol 171. New York, NY, USA: Springer; 2003. p. 149 – 172.

[98] Dai W, Yang Q, Xue GR, Yu Y. Boosting for Transfer Learning. In: Proc. of the 24th International Conf. on Machine Learning. ICML '07. New York, NY, USA: ACM; 2007. p. 193 – 200.

[99] Xiao R, Zhu L, Zhang HJ. Boosting chain learning for object detection. In: Proc. of the Ninth International Conf. on Computer Vision. ICCV 2003. Washington, DC, USA: IEEE Computer Society; 2003. p. 709 – 715.

[100] Lizotte DJ, Madani O, Greiner R. Budgeted Learning of Naive-Bayes Classifiers. In: Proc. of the Nineteenth Conf. on Uncertainty in Artificial Intelligence. UAI '03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2003. p. 378 – 385.

[101] Deng K, Zheng Y, Bourke C, Scott S, Masciale J. New Algorithms for Budgeted Learning. Machine Learning. 2013 January;90(1):59 – 90.

[102] Wu CY, Manmatha R, Smola AJ, Krahenbuhl P. Sampling matters in deep embedding learning. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 2840–2848.

[103] Wang C, Zhang X, Lan X. How to train triplet networks with 100k identities? In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 1907–1915.

[104] Kumar MP, Packer B, Koller D. Self-Paced Learning for Latent Variable Models. In: Advances in Neural Information Processing Systems 23. NIPS '10. Red Hook, NY, USA: Curran Associates, Inc.; 2010. p. 1189 – 1197.

[105] Pentina A, Sharmanska V, Lampert CH. Curriculum Learning of Multiple Tasks. In: Proc. of the 2015 IEEE Computer Society Conf. on Computer Vision and Patter Recognition. CVPR '15. Washington, DC, USA: IEEE Computer Society; 2015. p. 5492 – 5500.

[106] Appalaraju S, Chaoji V. Image similarity using Deep CNN and Curriculum Learning. CoRR. 2017;abs/1709.08761. Available from: http://arxiv.org/abs/1709.08761.

[107] Wang Y, Gan W, Yang J, Wu W, Yan J. Dynamic Curriculum Learning for Imbalanced Data Classification. In: The IEEE International Conference on Computer Vision (ICCV); 2019. .

[108] Snell J, Swersky K, Zemel R. Prototypical networks for few-shot learning. In: Advances in Neural Information Processing Systems; 2017. p. 4077–4087.

[109] Prabhu V, Kannan A, Ravuri M, Chablani M, Sontag DA, Amatriain X. Prototypical Clustering Networks for Dermatological Disease Diagnosis. CoRR. 2018;abs/1811.03066. Available from: http://arxiv.org/abs/1811.03066.

[110] Mathisen BM, Aamodt A, Bach K, Langseth H. Learning similarity measures from data. Progress in Artificial Intelligence. 2019 Oct. Available from: https://doi.org/10.1007/s13748-019-00201-2.

[111] Ye X, Leake D, Huibregtse W, Dalkilic M. Class-to-class Siamese Networks: Explaining Classification using Supportive and Contrastive Cases. In: Proc. International Conf. on Case-Based Reasoning. Springer, Cham; 2020. .

[112] Caruana R, Kangarloo H, Dionisio J, Sinha U, Johnson D. Case-based explanation of non-case-based learning methods. In: Proceedings of the AMIA Symposium. American Medical Informatics Association; 1999. p. 212.

[113] Arrieta AB, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion. 2020;58:82–115.

[114] Hoffman RR, Mueller ST, Klein G, Litman J. Metrics for explainable ai: Challenges and prospects. arXiv preprint arXiv:181204608. 2018.

[115] Rosenfeld A, Richardson A. Explainability in human–agent systems. Autonomous Agents and Multi-Agent Systems. 2019;33(6):673–705.

[116] European Parliament and Council. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation);.

[117] Gunning D. Explainable artificial intelligence (xai). Defense Advanced Research Projects Agency (DARPA), nd Web. 2017.

[118] Ras G, van Gerven M, Haselager P. In: Escalante HJ, Escalera S, Guyon I, Baró X, Güçlütürk Y, Güçlü U, et al., editors. Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges. Cham: Springer International Publishing; 2018. p. 19–36.

[119] Nordin I. Expert and non-expert knowledge in medical practice. Medicine, Health Care and Philosophy. 2000 Oct;3(3):295–302.

[120] Miller T. Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence. 2018.

[121] Kouki P, Schaffer J, Pujara J, O'Donovan J, Getoor L. Personalized Explanations for Hybrid Recommender Systems. In: Proceedings of the 24th International Conference on Intelligent User Interfaces. New York, NY, USA: Association for Computing Machinery; 2019. p. 379–390.

[122] Tintarev N, Masthoff J. Evaluating the effectiveness of explanations for recommender systems. User Modeling and User-Adapted Interaction. 2012;22(4-5):399–439.

[123] Muir BM. Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems. Ergonomics. 1994;37(11):1905–1922.

[124] Mueller ST, Hoffman RR, Clancey W, Emrey A, Klein G. Explanation in human-AI systems: A literature meta-review, synopsis of key ideas and publications, and bibliography for explainable AI. arXiv preprint arXiv:190201876. 2019.

[125] Gilpin LH, Bau D, Yuan BZ, Bajwa A, Specter M, Kagal L. Explaining explanations: An overview of interpretability of machine learning. In: 2018 IEEE 5th International Conference on data science and advanced analytics (DSAA). IEEE; 2018. p. 80–89.

[126] Ross AS, Hughes MC, Doshi-Velez F. Right for the right reasons: Training differentiable models by constraining their explanations. arXiv preprint arXiv:170303717. 2017.

[127] Lamy JB, Sekar B, Guezennec G, Bouaud J, Séroussi B. Explainable artificial intelligence for breast cancer: A visual case-based reasoning approach. Artificial intelligence in medicine. 2019;94:42–53.

[128] Massie S, Craw S, Wiratunga W. Visualisation of case-base reasoning for explanation. In: Proceedings of the ECCBR 2004 Workshops; 2004. p. 135–144.

[129] Massie S, Craw S, Wiratunga N. Complexity-guided case discovery for case-based reasoning. In: Proc. of the 20th AAAI Conf. on AI. AAAI Press; 2005. p. 216—221.

[130] Sani S, Wiratunga N, Massie S, Cooper K. kNN Sampling for Personalised Human Activity Recognition. In: Proc. International Conf. on Case-Based Reasoning. Springer, Cham; 2017. .

[131] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014.

[132] Misra K. In: Service Provisioning and Activation. London: Springer London; 2004. p. 49–91. Available from: https://doi.org/10.1007/978-0-85729-400-5_6.

[133] Isinkaye FO, Folajimi YO, Ojokoh BA. Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal. 2015;16(3):261 – 273.

[134] Muhammad K, Lawlor A, Smyth B. On the Pros and Cons of Explanation-Based Ranking. In: Aha DW, Lieber J, editors. Case-Based Reasoning Research and Development. Cham: Springer International Publishing; 2017. p. 227 – 241.

[135] Arras L, Horn F, Montavon G, Müller KR, Samek W. "What is relevant in a text document?": An interpretable machine learning approach. PLOS ONE. 2017 08;12(8):1–23.

[136] Cheetham W. Case-Based Reasoning with Confidence. In: Blanzieri E, Portinale L, editors. Advances in Case-Based Reasoning. Berlin, Heidelberg: Springer Berlin Heidelberg; 2000. p. 15–25.

[137] Lind M, Johansson J, Cooper M. Many-to-Many Relational Parallel Coordinates Displays. In: 2009 13th International Conference on Information Visualisation; 2009. p. 25 – 31.

[138] Fong RC, Vedaldi A. Interpretable explanations of black boxes by meaningful perturbation. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 3429–3437.

[139] Hou YL, Peng J, Hao X, Shen Y, Qian M. Occlusion localization based on convolutional neural networks. In: 2017 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC). IEEE; 2017. p. 1–5.

[140] Kulesza T, Stumpf S, Burnett M, Wong WK, Riche Y, Moore T, et al. Explanatory debugging: Supporting end-user debugging of machine-learned programs. In: 2010 IEEE Symposium on Visual Languages and Human-Centric Computing. IEEE; 2010. p. 41–48.

[141] Luhn HP. The automatic creation of literature abstracts. IBM Journal of research and development. 1958;2(2):159–165.

[142] Hachey B, Grover C. Extractive summarisation of legal texts. Artificial Intelligence and Law. 2006;14(4):305–345.

[143] Collins E, Augenstein I, Riedel S. A supervised approach to extractive summarisation of scientific papers. arXiv preprint arXiv:170603946. 2017.

Appendix A

Support for this Thesis

All contribution chapters for this thesis have been published following peer review from the international research community. The bibliographical information for the relevant papers for each chapter is linked in this short appendix.

Chapter 4

- Martin, K., Wiratunga, N., Massie, S. and Clos, J., 2018, December. Informed Pair Selection for Self-paced Metric Learning in Siamese Neural Networks. In International Conference on Innovative Techniques and Applications of Artificial Intelligence (pp. 34-49). Springer, Cham.
- K. Martin, N. Wiratunga and S. Sani, 2020. "Locality Sensitive Batch Selection for Triplet Networks," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom.

Chapter 5

- Martin, K., Liret, A., Wiratunga, N., Owusu, G. and Kern, M., 2018, December. Risk Information Recommendation for Engineering Workers. In International Conference on Innovative Techniques and Applications of Artificial Intelligence (pp. 311-325). Springer, Cham.
- Martin, K., Liret, A., Wiratunga, N., Owusu, G. and Kern, M., 2019, December. Developing a Catalogue of Explainability Methods to Support Expert and Non-Expert Users. In International Conference on Innovative Techniques and Applications of Artificial Intelligence. Springer, Cham. [**Awarded Best Student Application Paper**]

Chapter 6

- Martin, K., Liret, A., Wiratunga, N., Owusu, G. and Kern, M., 2019, December. Developing a Catalogue of Explainability Methods to Support Expert and Non-Expert Users. In International Conference on Innovative Techniques and Applications of Artificial Intelligence. Springer, Cham. [**Awarded Best Student Application Paper**]

- Martin, K., Liret, A., Wiratunga, N., Owusu, G. and Kern, M., 2021. Evaluating Explainability Methods Intended for Multiple Stakeholders. KI - Künstliche Intelligenz, 1-15.

Other Publications

Other relevant publications achieved throughout this thesis are shown below.

Martin, K., Wijekoon, A. and Wiratunga, N. 2019, September. Human activity recognition with deep metric learners. In Workshop Proceedings of the 27th International Conference on Case-based Reasoning (ICCBR 2019), 8-12 September 2019, Otzenhausen, Germany. CEUR-WS, Aachen.

Martin, K., Wiratunga, N., Sani, S., Massie, S. and Clos, J., 2017, June. A Convolutional Siamese Network for Developing Similarity Knowledge in the SelfBACK Dataset. In Workshop Proceedings of the 25th International Conference on Case-based Reasoning (ICCBR 2017), 26 - 28 June 2017, Trondheim, Norway. (pp. 85-94). CEUR-WS, Aachen.

Martin, K., 2017, June. A Case-Based Real-Time Adaptive Engineer Site Support System. In Workshop Proceedings of the 25th International Conference on Case-based Reasoning (ICCBR 2017), 26 - 28 June 2017, Trondheim, Norway. (pp. 184-188). CEUR-WS, Aachen.