Hashcode Representations of Natural Language for Relation Extraction

by

Sahil Garg

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 2019

Acknowledgements

This thesis is dedicated to my friend, Salil Garg. Consider it your thesis, my friend, you shall remain in our hearts as fresh as alive as we are, and shall be immortal through this work after the ones who knew you would be gone.

I would start from the beginning when the seed of passion for research was sown. It was in my school days, I developed scientific curiosity, passion and rebelness to try out something new, defying the known. Many helped me in the beginning of this journey, naming a few, my father, Pawan Garg, my teachers, Kiran, Sushil, Surinder, friends, Sahil, Anand, Mandeep, Pritpal, Manisha, and my dear brother, Sandeep Singla. One simple advice, that we tend to forget in research, I was bestowed from my organic chemistry tutor, Masood Khan, was: "always start from basics when approaching a tough problem". This advice has been the guiding principle for all these years, trying to build basic intuitions about a problem on hand, leading to simpler algorithms. During undergrad days, I had the privilege of making friends, Vivek, Salil, Tansy, Tarun, Sunny, Gaurav, Arun, Pooja, Himani, Jaskanwal, who listened to my silly ideas on new algorithms, encouraged me to keep on enjoying the magical field of computer science, stood by me in my tough days. This passion for exploring the space for small algorithmic innovations kept on even when developing softwares for real world problems in the role of software engineer, under guidance of awesome mentors, Satie, Atul, Nitin, Harman, Krish. A very special thanks to Amit Verma, Nitin Gupta, and Srinivasa Reddy for the confidence in me. You remain role models for me to work hard, enjoying the journey as well as a goal, and most importantly caring about colleagues like a family, being humble.

The formal journey for research started back in 2011, from the initial guidance of Vishal Bansal, Siddharth Eswaran, and the help of dear friends, Rishi Garg and Vivek Chauhan who made the introductions. I could not have contributed any formal piece of work to the research literature if Prof. Amarjeet Singh would not have accepted to guide me in IIIT Delhi. He saw the potential in me, boosted me with confidence about my abilities to solve difficult scientific problems. It is from Prof. Singh, and from our collaborator, Prof. Fabio Ramos, I learned the value of patience in research, the courage to stand by your work, genuine caring for junior researchers. It took me a while to appreciate the advice of Prof. Singh to pursue problem driven research for making a real world impact. Prof. Singh, I remain your student for the lifetime regardless the nature of being an independent researcher you instilled in me. Thanks to Nipun, Sneihil, Shikha, Samy, Kuldeep, Himanshu, Samarth, Manoj, Tejas, Manaswi, Milan, for making my days in IIIT D almost as wonderful as the PhD journey.

My PhD journey started back in 2013 summer. A very special thanks to Prof. Milind Tambe and Prof. Nora Ayanian for their confidence in my research abilities, and thanks to Manish, Yundi, and Jun for guiding me as senior students. I found the most suitable environment for research as a graduate student under the guidance of Prof. Aram Galstyan and Prof. Greg Ver Steeg. Under the official mentorship of Prof. Galstyan, I got the opportunities and support for doing real world problem driven research while ensuring to develop algorithmic techniques with strong theoretical basis. Intuitions, hundreds of hours of brain storming, many many criticisms, extensive error analysis, have been some of the key aspects of the nontangible guide book to accomplish this thesis work. The credit goes to my awesome adviser, Prof. Galstyan.

I cherish and appreciate the friendships born during the PhD program, for the great times shared discussing and living, research, life and what not, to name a few, Kyle, Palash, Debarun, Shuyang, Rashik, Dave, Dan, Neal, Sarik, Yun, Shushan, Shankar, Abhilash, Shankar, Ashok, Mehrnoosh, Sami, Serban, Rob, Nina, Nazgol, Tozammel, Yilei, Nitin, Jitin, Sayan, Ramesh, Vinod, Pashyo, Sandeep, Diana, Hrayr, Linhong, Ransalu, Bortik, Will, Ankit, Ayush, Ekraam. The greatest fortune in PhD life, was meeting a humble friend, whose light shined deep into my heart and soul, changing the whole perspective towards

life, becoming more bold and unattached, yet putting in more energies into work like a play, not worrying about fruits. I don't know what PhD life would have been like if I had not the privilege of enjoying old friendships of Sahil, Sawan, Ankit, Sanju, Pinkoo, Rahul, Tansy, Tarun, Sunny, Priyank, Deepika, Deepanshu, and my silly cousin sister who doesn't like to talk to me anymore. Lots of love for my mother, Kamlesh Rani, my sister, Priyanka Garg, and brother in law, Manit Garg, and my cousins, nephews, and nieces.

The guidance of Prof. Kevin Knight, Prof. Daniel Marcu, and Prof. Satish Kumar Thittamaranahalli, was immensely helpful for the thesis. Many thanks to the thesis proposal committee members, Prof. Gaurav Sukhatme, Prof. Shrikanth Narayanan, Prof. Xiang Ren, Prof. Kevin Knight, Prof. Aram Galstyan (chair), and the thesis committee members, Prof. Roger Georges Ghanem, Prof. Kevin Knight, Prof. Greg Ver Steeg, Prof. Aram Galstyan (chair), and Dr. Irina Rish, and Lizsl De Leon who helped me throughout the PhD program so that I could keep the focus on research & studies, shielded from the details of administrative affairs.

One of the many great aspects of working under the guidance of my adviser, Prof. Aram Galstyan, has been the freedom to collaborate within and outside USC. This led to collaborations in IBM T. J. Watson Research Center, specifically with Dr. Irina Rish, Dr. Guillermo Cecchi, on many topics including the thesis work and related works on representation learning. The experience of working with the two brilliant researchers, has been invaluable. It was also a lot of fun to interact with many other researchers from the center, and spending two summers there as an intern.

Table of Contents

Abstract

This thesis studies the problem of identifying and extracting relationships between biological entities from the text of scientific papers. For the relation extraction task, state-of-the-art performance has been achieved by classification methods based on convolution kernels which facilitate sophisticated reasoning on natural language text using structural similarities between sentences and/or their parse trees. Despite their success, however, kernel-based methods are difficult to customize and computationally expensive to scale to large datasets. In this thesis, the first problem is addressed by proposing a nonstationary extension to the conventional convolution kernels for improved expressiveness and flexibility. For scalability, I propose to employ kernelized locality sensitive hashcodes as explicit representations of natural language structures, which can be used as feature-vector inputs to arbitrary classification methods. For optimizing the representations, a theoretically justified method is proposed that is based on approximate and efficient maximization of the mutual information between the hashcodes and the class labels. I have evaluated the proposed approach on multiple biomedical relation extraction datasets, and have observed significant and robust improvements in accuracy over state-of-the-art classifiers, along with drastic orders-of-magnitude speedup compared to conventional kernel methods.

Finally, in this thesis, a nearly-unsupervised framework is introduced for learning kernel- or neural-hashcode representations. In this framework, an information-theoretic objective is defined which leverages both labeled and unlabeled data points for fine-grained optimization of each hash function, and propose a greedy algorithm for maximizing that objective. This novel learning paradigm is beneficial for building hashcode representations generalizing from a training set to a test set. I have conducted a thorough experimental evaluation on the relation extraction datasets, and demonstrate that the proposed extension leads

to superior accuracies with respect to state-of-the-art supervised and semi-supervised approaches, such as variational autoencoders and adversarial neural networks. An added benefit of the proposed representation learning technique is that it is easily parallelizable, interpretable, and owing to its generality, applicable to a wide range of NLP problems.

Chapter 1

Introduction

Recent advances in genomics and proteomics have significantly accelerated the rate of uncovering and accumulating new biomedical knowledge. Most of this knowledge is available only via scientific publications, which necessitates the development of automated and semi-automated tools for extracting useful biomedical information from unstructured text. In particular, the task of identifying biological entities and their relations from scientific papers has attracted significant attention in the past several years (Rios et al., 2018; Peng and Lu, 2017; Hsieh et al., 2017; Kavuluru et al., 2017; Rao et al., 2017; Hahn and Surdeanu, 2015; Tikk et al., 2010; Airola et al., 2008; Krallinger et al., 2008; Hakenberg et al., 2008; Hunter et al., 2008; Mooney and Bunescu, 2005; Bunescu et al., 2005; McDonald et al., 2005), especially because of its potential impact on developing personalized cancer treatments (Downward, 2003; Vogelstein and Kinzler, 2004; Cohen, 2015; Rzhetsky, 2016; Valenzuela-Escárcega et al., 2017).

1.1 Task: Biomedical Relation Extraction

Consider the sentence *"As a result, mutant Ras proteins accumulate with elevated GTP-bound proportion"*, which describes a "binding" relation (interaction) between a protein "Ras" and a small-molecule "GTP". We want to extract this relation. As per the literature on biomedical relation extraction, one of the preliminary steps for the extraction of relations from a text sentence is to parse the sentence into a syntactic or semantic graph, and to perform the extraction from the graph itself.

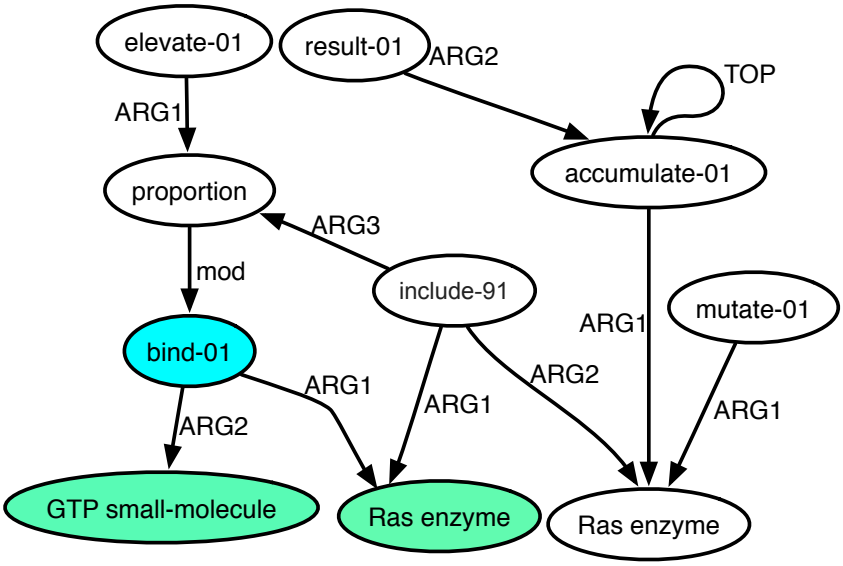
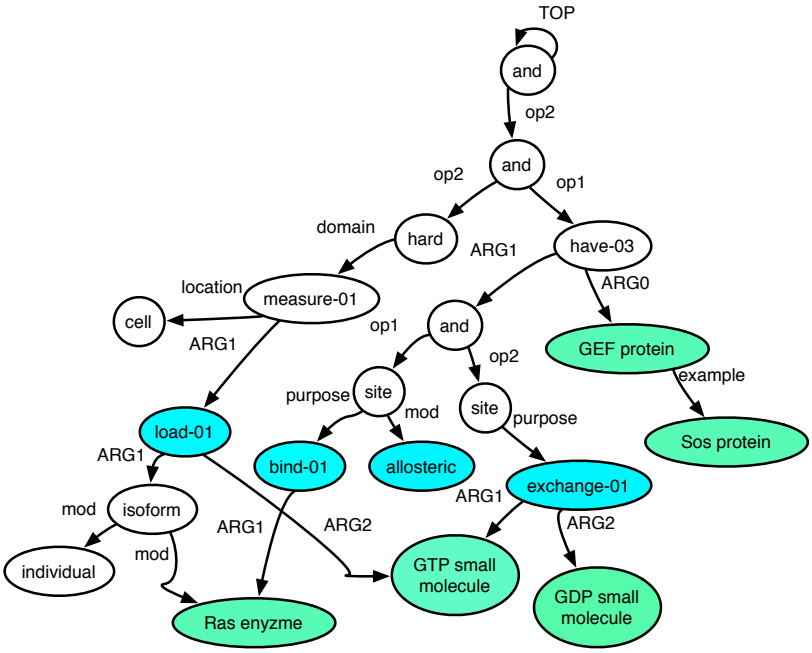


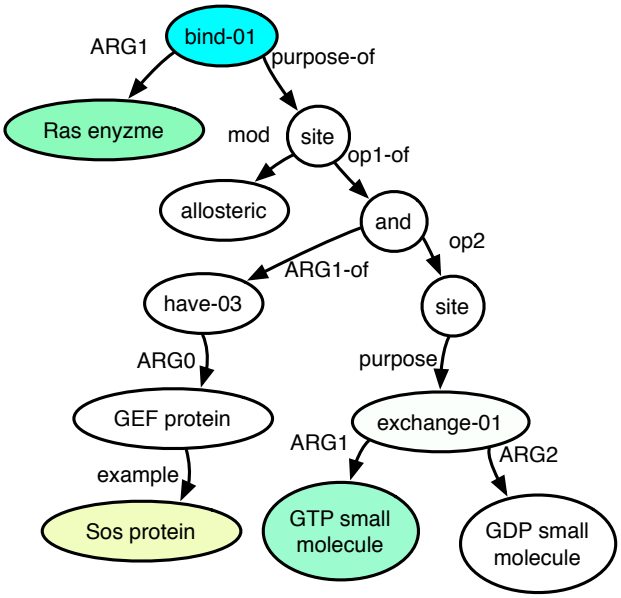
Figure 1.1: AMR of text "*As a result, mutant Ras proteins accumulate with elevated GTP-bound proportion.*"; the information regarding a hypothesized relation, "*Ras binds to GTP*", is localized to the colored sub-graph. The subgraph can be used as structural features to infer if the candidate relation is correct (valid) or incorrect (invalid).

Figure 1.1 depicts a manual Abstract Meaning Representation (AMR, a semantic representation) of the above mentioned sentence. The relation "RAS-binds-GTP" is extracted from the highlighted subgraph under the "bind" node. In the subgraph, relationship between the interaction node "bind-01" and the entity nodes, "Ras" and "GTP", is defined through two edges, "ARG1" and "ARG2" respectively.

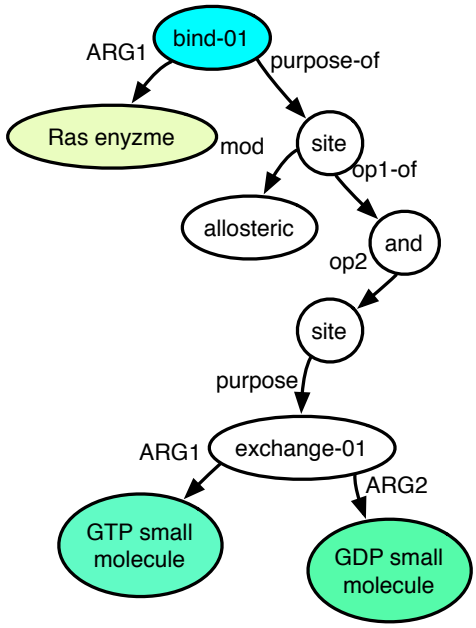
For a deeper understanding of the task, see Figure 1.2. Given an AMR graph, as in Figure 1.2(a), we first identify potential entity nodes (proteins, molecules, etc) and interaction nodes (bind, activate, etc). Next, we consider all permutations to generate a set of potential relations. For each candidate relation, we extract the corresponding shortest path subgraph, as shown in Figure 1.2(b) and 1.2(c). In order to classify a relation, as correct or incorrect, its corresponding subgraph can be classified as a proxy; same applies to the labeled candidates in a train set. Also, from a subgraph, a sequence can be generated by traversals between the root node and the leaf nodes if using classifiers operating on sequences. In general, the biomedical relation extraction task is formulated as of classification of natural language structures, such as sequences, graphs, etc. Note that this thesis doesn't delve in to the problem of named entity recognition,



(a) AMR Graph



(b) correct (positive label)



(c) incorrect (negative label)

Figure 1.2: In 1.2(a), a sentence is parsed into a semantic graph. Then the two candidate hypothesis about different biomolecular interactions (structured prediction candidates) are generated automatically. According to the text, a valid hypothesis is that *Sos catalyzes binding between Ras and GTP*, while the alternative hypothesis *Ras catalyzes binding between GTP and GDP* is false; each of those hypotheses corresponds to one of the post-processed subgraphs shown in 1.2(b) and 1.2(c), respectively.

or the problem of syntactic/semantic parsing. I use existing tools from the literature for these upstream (challenging) tasks in the pipeline.

1.2 Convolution Kernels for Relation Extraction

An important family of classifiers that operate on discrete structures is based on convolution kernels, originally proposed by (Haussler, 1999), for computing similarity between discrete structures of any type and later extended for specific structures such as strings, sequences/paths, trees (Collins and Duffy, 2001; Zelenko et al., 2003; Mooney and Bunescu, 2005; Shi et al., 2009). For the relation extraction task, convolution kernel based classifiers have demonstrated state-of-the-art performance (Chang et al., 2016; Tikk et al., 2010; Miwa et al., 2009; Airola et al., 2008).

1.2.1 Shortcomings of Convolution Kernels

Despite the success and intuitive appeal of convolution kernel-based methods in various NLP tasks, including biomedical relation extraction, there are two important issues limiting their practicality in real-world applications (Collins and Duffy, 2001; Moschitti, 2006; Tikk et al., 2010; Qian and Zhou, 2012; Srivastava et al., 2013; Hovy et al., 2013; Filice et al., 2015; Tymoshenko et al., 2016).

1.2.1.1 Lack of Flexibility in Kernels

First, convolution kernels are not *flexible* enough to adequately model rich natural language representations, as they typically depend only on a few tunable parameters. This inherent rigidity prevents kernel-based methods from properly adapting to a given task, as opposed to, for instance, neural networks that typically have millions of parameters that are learned from data and show state-of-the-art results for a number of NLP problems (Collobert and Weston, 2008; Sundermeyer et al., 2012; Chen and Manning, 2014; Sutskever et al., 2014; Kalchbrenner et al., 2014; Luong et al., 2015; Kumar et al., 2016).

1.2.1.2 Prohibitive Kernel Compute Cost

The second major issue with convolution kernels is that the traditional kernel-trick based methods can suffer from relatively high computational costs, since computing kernel similarities between two natural language structures (graphs, paths, sequences, etc.) can be an expensive operation. Furthermore, to build a Support Vector Machine (SVM), or a k-Nearest Neighbor (kNN) classifier or a Gaussian Process (GP) classifier from N training examples, one needs to compute kernel similarities between $O(N^2)$ pairs of training points (Yu et al., 2002; Cortes and Vapnik, 1995; Schölkopf and Smola, 2002), which can be prohibitively expensive for large N (Moschitti, 2006; Rahimi and Recht, 2008; Pighin and Moschitti, 2009; Zanzotto and Dell'Arciprete, 2012; Severyn and Moschitti, 2013; Felix et al., 2016).

I attempt to resolve the two shortcoming of convolution kernels by proposing novel models and algorithms, relying upon two key concepts: (a) nonstationary kernels (Paciorek and Schervish, 2003; Snelson et al., 2003; Le et al., 2005; Assael et al., 2014), and (b) locality sensitive hashing (Indyk and Motwani, 1998; Kulis and Grauman, 2009; Joly and Buisson, 2011). These two concepts are known in computer science, but not explored for convolution kernels previously.

1.2.2 Key Concepts

1.2.2.1 Nonstationary Kernels

In this thesis, a nonstationary extension to the conventional convolution kernels is proposed, introducing a novel, task-dependent parameterization of the kernel similarity function for better expressiveness and flexibility. Those parameters, which need to be inferred from the data, are defined in a way that allows the model to ignore substructures irrelevant for a given task when computing kernel similarity.

1.2.2.2 Locality Sensitive Hashing

To address the scalability issue, I propose to build implicit or explicit representations of data points using kernelized locality sensitive hashcodes, usable with classifiers, such as kNN, Random Forest, etc. The required number of kernel computations is $O(N)$, i.e. linear in the number of training examples. When using the hashcodes as implicit representations for kNN classification, the technique of locality sensitive hashing serves only as an approximation. On the other hand, as a core contribution of this thesis, I also propose to employ hashcodes as *explicit* representations of natural language so that one can use this efficient kernel functions based technique with a wider range of classification models, for the goal of obtaining superior inference accuracies, rather than approximations, along with the advantage of scalability.

1.2.3 Locality Sensitive Hashcodes as Explicit Representations

From the above, we understand that locality sensitive hashing technique can be leveraged for efficient training of classifiers operating on kNN-graphs. However, the question is whether scalable kernel locality-sensitive hashing approaches can be *generalized to a wider range of classifiers*. Considering this, I propose a principled approach for building *explicit representations* for structured data, as opposed to implicit ones employed in prior kNN-graph-based approaches, *by using random subspaces* of kernelized locality sensitive hashcodes. For learning locality sensitive hashcode representations, I propose a *theoretically justified and computationally efficient method to optimize the hashing model* with respect to: (1) the kernel function parameters and (2) a reference set of examples w.r.t. which kernel similarities of data samples are computed for obtaining their hashcodes. This approach maximizes *an approximation of* mutual information between hashcodes of NLP structures and their class labels. The main advantage of the proposed representation learning technique is that it can be used with arbitrary classification methods, besides kNN, such as Random Forests (RF) (Ho, 1995; Breiman, 2001). Additional parameters, resulting from the proposed non-stationary extension, can also be learned by maximizing the mutual information approximation.

1.2.4 Nearly Unsupervised Hashcode Representations

Finally, I also consider a semi-supervised setting to learn hashcode representations. For this settings, I introduce an information-theoretic algorithm, which is *nearly unsupervised* as I propose to use only the knowledge of which set an example comes from, a training set or a test set, along with the example itself, whereas the actual class labels of examples (from a training set) are input only to the final supervised-classifier, such as an RF, which takes input of the learned hashcodes as representation (feature) vectors of examples along with their class labels. I introduce multiple concepts for fine-grained optimization of hash functions, employed in the algorithm that constructs hash functions greedily one by one. In supervised settings, fine-grained (greedy) optimization of hash functions could lead to overfitting whereas, in my proposed nearly-unsupervised framework, it allows flexibility for explicitly maximizing the generalization capabilities of hash functions.

1.3 Thesis Overview

The structure of the thesis is organized as below.

- In Chapter 2, I provide a problem statement along with some background on convolution kernels, and then demonstrate that Abstract Meaning Representations (AMR) significantly improve the accuracy of a convolution kernel based relation extraction system when compared to a baseline that relies solely on surface- and syntax-based features. I also propose an approach, for inference of relations over sets of multiple sentences (documents), or for concurrent exploitation of automatically induced AMR (semantic) and dependency structure (syntactic) representations.

- In Chapter 3, I propose a generalization of convolution kernels, with a nonstationary model, for better expressibility of natural languages. For a scalable learning of the parameters introduced with the model, I propose a novel algorithm that leverages stochastic sampling on k-nearest neighbor graphs, along with approximations based on locality-sensitive hashing.

- In Chapter 4, I introduce a core contribution of this thesis, proposing to use random subspaces of kernelized locality sensitive hashcodes for efficiently constructing an explicit representation of natural language structures suitable for general classification methods. Further, I propose a principled method for optimizing the kernelized locality sensitive hashing model for classification problems by maximizing an approximation of mutual information between the hashcodes (feature vectors) and the class labels.

- In Chapter 5, I extend the hashcode representations based approach with a *nearly-unsupervised* learning framework for *fine grained optimization* of each hash function so as to building hashcode representations generalizing from a training set to a test set.

- Besides the related works discussed specific to the chapters, I discuss more related works in Chapter 6, and then conclude the thesis in Chapter 7.

It is worth noting that Chapter 2 to 5 are written such that any of those are readable independent of each other.

Chapter 2

Extracting Biomolecular Interactions Using Semantic Parsing of Biomedical Text

Despite the recent progress, current methods for biomedical knowledge extraction suffer from a number of important shortcomings. First of all, existing methods rely heavily on shallow analysis techniques that severely limit their scope. For instance, most existing approaches focus on whether there is an interaction between a pair of proteins while ignoring the interaction types (Airola et al., 2008; Mooney and Bunescu, 2005), whereas other more advanced approaches cover only a small subset of all possible interaction types (Hunter et al., 2008; McDonald et al., 2005; Demir et al., 2010). Second, most existing methods focus on single-sentence extraction, which makes them very susceptible to noise. And finally, owing to the enormous diversity of research topics in biomedical literature and the high cost of data annotation, there is often significant mismatch between training and testing corpora, which reflects poorly on generalization ability of existing methods (Tikk et al., 2010).

In this chapter, I present a novel algorithm for extracting biomolecular interactions from unstructured text that addresses the above challenges. Contrary to the previous works, the extraction task considered here is less restricted and spans a much more diverse corpus of biomedical articles. These more realistic settings present some important technical problems for which I provide explicit solutions.

The specific contributions of the thesis, which are introduced in this chapter (Garg et al., 2016), are as follows:

- I propose a convolution graph-kernel based algorithm for extracting biomolecular interactions from Abstract Meaning Representation, or AMR. To the best of my knowledge, this is the first attempt of using deep semantic parsing for biomedical knowledge extraction task.

- I provide a multi-sentence generalization of the algorithm by defining *Graph Distribution Kernels* (GDK), which enables us to perform document-level extraction.

- I suggest a hybrid extraction method that utilizes both AMRs and syntactic parses given by Stanford Dependency Graphs (SDGs). Toward this goal, I develop a linear algebraic formulation for *learning vector space embedding of edge labels* in AMRs and SDGs to define similarity measures between AMRs and SDGs.

I conduct an exhaustive empirical evaluation of the proposed extraction system on 45+ research articles on cancer (approximately 3k sentences), containing approximately 20,000 positive-negative labeled biomolecular interactions.[1] The results indicate that the joint extraction method that leverages both AMRs and SDGs parses significantly improves the extraction accuracy, and is more robust to mismatch between training and test conditions.

2.1 Problem Statement

Consider the sentence *"As a result, mutant Ras proteins accumulate with elevated GTP-bound proportion"*, which describes a "binding" interaction between a protein "Ras" and a small-molecule "GTP". We want to extract this interaction.

In our representation, which is motivated by BioPAX (Demir et al., 2010), an *interaction* refers to either i) an entity effecting state change of another entity; or ii) an entity binding/dissociating with another entity to form/break a complex while, optionally, also influenced by a third entity. An entity can be of any type existent in a bio pathway, such as protein, complex, enzyme, etc, although here we refer to an entity of all valid types simply as a protein. The change in state of an entity or binding type is simply termed as

[1] The code and the data are available at `https://github.com/sgarg87/big_mech_isi_gg`.

State change: inhibit, phosphorylate, signal, activate, transcript, regulate, apoptose, express, translocate, degrade, carboxymethylate, depalmitoylate, acetylate, nitrosylate, farnesylate, methylate, glycosylate, hydroxylate, ribosylate, sumoylate, ubiquitinate.
Bind: bind, heterodimerize, homodimerize, dissociate.

Table 2.1: Interaction type examples

"interaction type" in this work. In some cases, entities are capable of changing their state on their own or bind to an instance of its own (self-interaction). Such special cases are also included. Some examples of interaction types are shown in Table 2.1.

Below I describe my approach for extracting above-defined interactions from natural language parses of sentences in a research document.

2.2 Extracting Interactions from an AMR

2.2.1 AMR Biomedical Corpus

Abstract Meaning Representation, or AMR, is a semantic annotation of single/multiple sentences (Banarescu et al., 2013). In contrast to syntactic parses, in AMR, entities are identified, typed and their semantic roles are annotated. AMR maps different syntactic constructs to same conceptual term. For instance, "binding", "bound", "bond" correspond to the same concept "bind-01". Because one AMR representation subsumes multiple syntactic representations, it is hypothesized that AMRs have higher utility for extracting biomedical interactions.

An English-to-AMR parser (Pust et al., 2015b) is trained on two manually annotated corpora: i) a corpus of *17k general domain sentences* including newswire and web text as published by the Linguistic Data Consortium; and ii) *3.4k systems biology sentences*, including in-domain PubMedCentral papers and the BEL BioCreative corpus. As part of building the bio-specific AMR corpus, the PropBank-based framesets used in AMR are extended by 45 bio-specific frames such as "phosphorylate-01", "immunoblot-01" and the list of AMR standard named entities is also extended by 15 types such as "enzyme", "pathway".

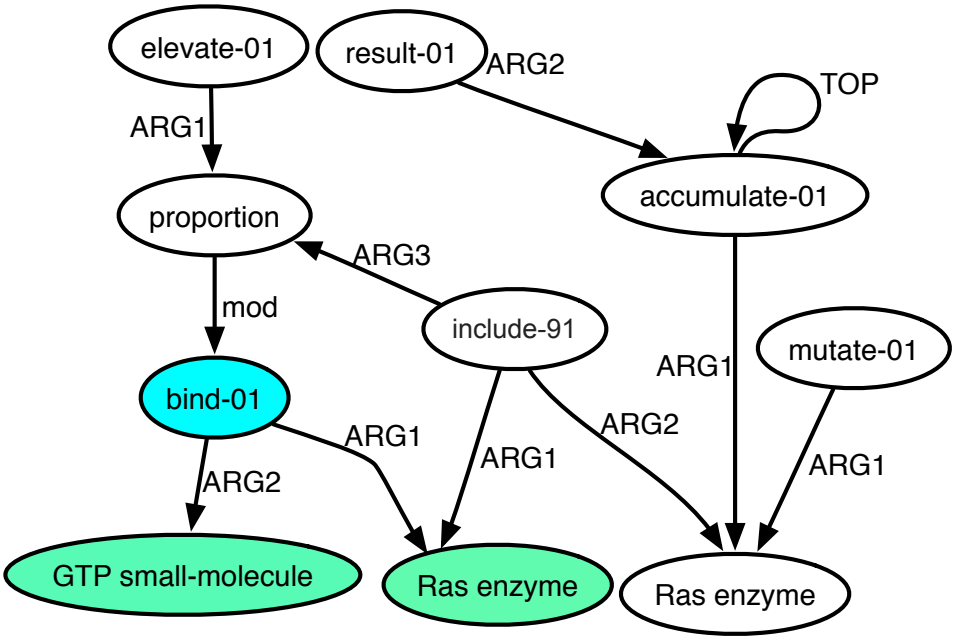


Figure 2.1: AMR of text "*As a result, mutant Ras proteins accumulate with elevated GTP-bound proportion.*"; interaction "*Ras binds to GTP*" is extracted from the colored sub-graph.

It is important to note that these extensions are not specific to biomolecular interactions, and cover more general cancer biology concepts.

2.2.2 Extracting Interactions

Figure 2.1 depicts a manual AMR annotation of a sentence, which has two highlighted entity nodes with labels "RAS" and "GTP". These nodes also have entity type annotations, "enzyme" and "small-molecule" respectively; the concept node with a node label "bind-01" corresponds to an interaction type "binding" (from the "GTP-bound" in the text). The interaction "RAS-binds-GTP" is extracted from the highlighted subgraph under the "bind" node. In the subgraph, relationship between the interaction node "bind-01" and the entity nodes, "Ras" and "GTP", is defined through two edges with edge labels "ARG1" and "ARG2" respectively. Additionally, in the subgraph, we assign roles "interaction-type", "protein", "protein" to the nodes "bind-01", "Ras", "GTP" respectively (roles presented with different colors in the subgraph).

Given an AMR graph, as in Figure 2.1, we first identify potential entity nodes (proteins, molecules, etc) and interaction nodes (bind, activate, etc). Next, we consider all permutations to generate a set of potential interactions according to the format defined above. For each candidate interaction, we extract the corresponding shortest path subgraph. We then project the subgraph to a tree structure with the interaction node as root and also possibly the protein nodes (entities involved in the interaction) as leaves.[2]

Our training set consists of tuples $\{G_i^a, I_i, l_i\}_{i=1}^n$, where G_i^a is an AMR subgraph constructed such that it can represent an extracted candidate interaction I_i with interaction node as root and proteins nodes as leaves typically; and $l = \{0, 1\}$ is a binary label indicating whether this subgraph contains I_i or not. Given a training set, and a new sample AMR subgraph G_*^a for interaction I_*, we would like to infer whether I_* is valid or not. I address this problem by developing a graph-kernel based approach.

2.2.3 Semantic Embedding Based Graph Kernel

I propose an extension of the *contiguous subtree kernel* (Zelenko et al., 2003; Culotta and Sorensen, 2004) for mapping the extracted subgraphs (tree structure) to an implicit feature space. Originally, this kernel uses an identity function on two node labels when calculating the similarity between those two nodes. I instead propose to use vector space embedding of the node labels (Clark, 2014; Mikolov et al., 2013), and then define a sparse RBF kernel on the node label vectors. Similar extensions of convolution kernels have been been suggested previously (Mehdad et al., 2010; Srivastava et al., 2013).

Consider two graphs G_i and G_j rooted at nodes $G_i.r$ and $G_j.r$, respectively, and let $G_i.c$ and $G_j.c$ be the children nodes of the corresponding root nodes. Then the kernel between G_i and G_j is defined as follows:

$$K(G_i, G_j) = \begin{cases} 0 & \text{if } k(i,j) = 0 \\ k(i,j) + K_c(G_i.c, G_j.c) & \text{otherwise} \end{cases},$$

[2]This can be done via so called inverse edge labels; see (Banarescu et al., 2013, section 3).

where $k(i,j) \equiv k(G_i.r, G_j.r)$ is the similarity between the root nodes, whereas $K_c(G_i.c, G_j.c)$ is the recursive part of the kernel that measures the similarity of the children subgraphs. Furthermore, the similarity between root nodes x and y is defined as follows:

$$\begin{aligned} k(x,y) &= k_w(x,y)^2(k_w(x,y)^2 + k_e(x,y) + k_r(x,y)) \\ k_w(x,y) &= \exp((\boldsymbol{w}_x^T \boldsymbol{w}_y - 1)/\beta)((\boldsymbol{w}_x^T \boldsymbol{w}_y - \alpha)/(1-\alpha))_+ \\ k_e(x,y) &= \mathbb{I}(e_x = e_y),\ k_r(x,y) = \mathbb{I}(r_x = r_y)\ . \end{aligned} \tag{2.1}$$

Here $(\cdot)_+$ denotes the positive part; $\mathbb{I}(\cdot)$ is the indicator function; $\boldsymbol{w}_x, \boldsymbol{w}_y$ are unit vector embeddings of node labels;[3] e_x, e_y represent edge labels (label of an edge from a node's parent to it is the node's edge label); r_x, r_y are roles of nodes (such as protein, catalyst, concept, interaction-type); α is a threshold parameter on the cosine similarity ($\boldsymbol{w}_x^T \boldsymbol{w}_y$) to control sparsity (Gneiting, 2002); and β is the bandwidth.

The recursive part of the kernel, K_c, is defined as follows:

$$K_c(G_i.c, G_j.c) = \sum_{\boldsymbol{i},\boldsymbol{j}: l(\boldsymbol{i}) = l(\boldsymbol{j})} \lambda^{l(\boldsymbol{i})} \sum_{s=1,\cdots,l(\boldsymbol{i})} K(G_i[\boldsymbol{i}[s]], G_j[\boldsymbol{j}[s]]) \prod_{s=1,\cdots,l(\boldsymbol{i})} k(G_i[\boldsymbol{i}[s]].r, G_j[\boldsymbol{j}[s]].r),$$

where $\boldsymbol{i}, \boldsymbol{j}$ are contiguous children subsequences under the respective root nodes $G_i.r, G_j.r$; $\lambda \in (0,1)$ is a tuning parameter; and $l(\boldsymbol{i})$ is the length of sequence $\boldsymbol{i} = i_1, \cdots, i_l$; $G_i[\boldsymbol{i}[s]]$ is a sub-tree rooted at $\boldsymbol{i}[s]$ index child node of $G_i.r$. Here, I propose to sort children of a node based on the corresponding edge labels. This helps in distinguishing between two mirror image trees.

This extension is a valid kernel function ((Zelenko et al., 2003, Theorem 3, p. 1090)). Next, I generalize the dynamic programming approach of (Zelenko et al., 2003) for efficient calculation of this extended kernel.

[3] Learned using word2vec software (Mikolov et al., 2013) on over one million PubMed articles.

2.2.3.1 Dynamic Programming for Computing Convolution Graph Kernel

In the convolution kernel presented above, the main computational cost is due to comparison of children sub-sequences. Since different children sub-sequences of a given root node partially overlap with each other, one can use dynamic programming to avoid redundant computations, thus reducing the cost. Toward this goal, one can use the following decomposition of the kernel K_c :

$$K_c(G_i.c, G_j.c) = \sum_{p,q} C_{p,q},$$

where $C_{p,q}$ refers to the similarity between the sub-sequences starting at indices p, q respectively in $G_i.c$ and $G_j.c$.

To calculate $C_{p,q}$ via dynamic programming, let us introduce

$$L_{p,q} = \max_{l} \left(\prod_{s=0}^{l} k(G_i[\boldsymbol{i}[p+s]].r, G_j[\boldsymbol{j}[q+s]].r) \neq 0 \right).$$

Furthermore, let us denote $k_{p,q} = k(G_i[\boldsymbol{i}[p]].r, G_j[\boldsymbol{j}[q]].r)$, and $K_{p,q} = K(G_i[\boldsymbol{i}[p]], G_j[\boldsymbol{j}[q]])$. We then evaluate $C_{p,q}$ in a recursive manner using the following equations.

$$L_{p,q} = \begin{cases} 0 & \text{if } k_{p,q} = 0 \\ L_{p+1,q+1} + 1 & \text{otherwise} \end{cases} \tag{2.2}$$

$$C_{p,q} = \begin{cases} 0 & \text{if } k_{p,q} = 0 \\ \frac{\lambda(1-\lambda^{L(p,q)})}{1-\lambda} K_{p,q} k_{p,q} + \lambda C_{p+1,q+1} & \text{otherwise} \end{cases} \tag{2.3}$$

$$L_{m+1,n+1} = 0, L_{m+1,n} = 0, L_{m,n+1} = 0$$
$$C_{m+1,n+1} = 0, C_{m+1,n} = 0, C_{m,n+1} = 0\ , \tag{2.4}$$

where m, n are number of children under the root nodes $G_i.r$ and $G_j.r$ respectively.

Note that for graphs with cycles, the above dynamic program can be transformed into a linear program.

There are a couple of practical considerations during the kernel computations. First of all, the kernel depends on two tunable parameters λ and α. Intuitively, decreasing λ discounts the contributions of longer child sub-sequences. The parameter α, on the other hand, controls the tradeoff between computational cost and accuracy. Based on some prior tuning I found that our results are not very sensitive to the parameters. In the experiments below in this chapter, I set $\lambda = 0.99$ and $\alpha = 0.4$. Also, consistent with previous studies, I normalize the graph kernel.

2.3 Graph Distribution Kernel- GDK

Often an interaction is mentioned more than once in the same research paper, which justifies a document-level extraction, where one combines evidence from multiple sentences. The prevailing approach to document-level extraction is to first perform inference at sentence level, and then combine those inferences using some type of an aggregation function for a final document-level inference (Skounakis and Craven, 2003; Bunescu et al., 2006). For instance, in (Bunescu et al., 2006), the inference with the maximum score is chosen. I term this baseline approach as "Maximum Score Inference", or MSI. Here I advocate a different approach, where one uses the evidences from multiple sentences jointly, for a collective inference.

Let us assume an interaction I_m is supported by k_m sentences, and let $\{G_{m1}, \cdots, G_{mk_m}\}$ be the set of relevant AMR subgraphs extracted from those sentences. We can view the elements of this set as samples from some distribution over the graphs, which, with a slight abuse of notation, we denote as $\mathcal{G}_m$. Consider now interactions $I_1, \cdots, I_p$, and let $\mathcal{G}_1, \cdots, \mathcal{G}_p$ be graph distributions representing these interactions.

The graph distribution kernel (GDK), $\mathcal{K}(\mathcal{G}_i, \mathcal{G}_j)$, for a pair $\mathcal{G}_i, \mathcal{G}_j$ is defined as follows:

$$\mathcal{K}(\mathcal{G}_i, \mathcal{G}_j) = \exp(-\mathcal{D}_{mm}(\mathcal{G}_i, \mathcal{G}_j));$$
$$\mathcal{D}_{mm}(\mathcal{G}_i, \mathcal{G}_j) = \sum_{r,s=1}^{k_i} \frac{K(G_{ir}, G_{is})}{k_i^2} + \sum_{r,s=1}^{k_j} \frac{K(G_{jr}, G_{js})}{k_j^2} - 2 \sum_{r,s=1}^{k_i,k_j} \frac{K(G_{ir}, G_{js})}{k_i k_j}$$

Here $\mathcal{D}_{mm}$ is the *Maximum Mean Discrepancy* (MMD), a valid l_2 norm, between a pair of distributions $\mathcal{G}_i, \mathcal{G}_j$ (Gretton et al., 2012); $K(.,.)$ is the graph kernel defined in Section 2.2.3 (though, not restricted to this specific kernel). As the term suggests, *maximum mean discrepancy represents the discrepancy between the mean of graph kernel features (features implied by kernels) in samples of distributions $\mathcal{G}_i$ and $\mathcal{G}_j$. Now, since $\mathcal{D}_{mm}$ is the l_2 norm on the mean feature vectors, $\mathcal{K}(\mathcal{G}_p, \mathcal{G}_q)$ is a valid kernel function.*

We note that MMD metric has attracted a considerable attention in the machine learning community recently (Gretton et al., 2012; Kim and Pineau, 2013; Pan et al., 2008; Borgwardt et al., 2006). For our purpose, we prefer using this divergence metric over others (such as KL-D divergence) for the following reasons: i) $\mathcal{D}_{mm}(.,.)$ is a "kernel trick" based formulation, nicely fitting with our settings since we do not have explicit features representation of the graphs but only kernel density on the graph samples. Same is true for KL-D estimation with kernel density method. ii) Empirical estimate of $\mathcal{D}_{mm}(.,.)$ is a valid l_2 norm distance. Therefore, it is straightforward to derive the graph distribution kernel $\mathcal{K}(\mathcal{G}_i, \mathcal{G}_j)$ from $\mathcal{D}_{mm}(\mathcal{G}_i, \mathcal{G}_j)$ using a function such as RBF. This is not true for divergence metrics such as KL-D, Renyi (Sutherland et al., 2012); iii) It is suitable for compactly supported distributions (small number of samples) whereas methods, such as k-nearest neighbor estimation of KL-D, are not suitable if the number of samples in a distribution is too small (Wang et al., 2009); iv) I have seen the most consistent results in our extraction experiments using this metric as opposed to the others.

For the above mentioned reasons, here I focus on MMD as our primary metric for computing similarities between graph distributions. The proposed GDK framework, however, is very general and not limited to a specific metric. Next, I briefly describe two other metrics which can be used with GDK.

2.3.1 GDK with Kullback-Leibler Divergence

While MMD represents maximum discrepancy between the mean features of two distributions, the Kullback-Leibler divergence (KL-D) is a more comprehensive (and fundamental) measure of distance between two distributions.[4] For defining kernel $\mathcal{K}_{KL}$ in terms of KL-D, however, we have two challenges. First of all, KL-D is not a symmetric function. This problem can be addressed by using a symmetric version of the distance in the RBF kernel,

$$\mathcal{K}_{KL}(\mathcal{G}_i, \mathcal{G}_j) = \exp(-[\mathcal{D}_{KL}(\mathcal{G}_i||\mathcal{G}_j) + \mathcal{D}_{KL}(\mathcal{G}_j||\mathcal{G}_i)])$$

where $\mathcal{D}_{KL}(\mathcal{G}_i||\mathcal{G}_j)$ is the KL distance of the distribution $\mathcal{G}_i$ w.r.t. the distribution $\mathcal{G}_j$. And second, even the symmetric combination of the divergences is not a valid Euclidian distance. Hence, $\mathcal{K}_{KL}$ is not guaranteed to be a positive semi-definite function. This issue can be dealt in a practical manner as nicely discussed in (Sutherland et al., 2012). Namely, having computed the Gram matrix using $\mathcal{K}_{KL}$, we can project it onto a positive semi-definite one by using linear algebraic techniques, e.g., by discarding negative eigenvalues from the spectrum.

Since we do not know the true divergence, I approximate it with its empirical estimate from the data, $\mathcal{D}_{KL}(\mathcal{G}_i||\mathcal{G}_j) \approx \hat{\mathcal{D}}_{KL}(\mathcal{G}_i||\mathcal{G}_j)$. While there are different approaches for estimating divergences from samples (Wang et al., 2009), here I use kernel density estimator as shown below:

$$\hat{\mathcal{D}}_{KL}(\mathcal{G}_i||\mathcal{G}_j) = \frac{1}{k_i}\sum_{r=1}^{k_i} \log \frac{\frac{1}{k_i}\sum_{s=1}^{k_i} K(G_{ir}, G_{is})}{\frac{1}{k_j}\sum_{s=1}^{k_j} K(G_{ir}, G_{js})}$$

[4] Recall that the KL divergence between distributions p and q is defined as $\mathcal{D}_{KL}(p||q) = \mathbb{E}_{p(x)}[\log \frac{p(x)}{q(x)}]$.

2.3.2 GDK with Cross Kernels

Another simple way to evaluate similarity between two distributions is to take the mean of cross-kernel similarities between the corresponding two sample sets:

$$\mathcal{K}(\mathcal{G}_i, \mathcal{G}_j) = \sum_{r,s=1}^{k_i,k_j} \frac{K(G_{ir}, G_{js})}{k_i k_j}$$

Note that this metric looks quite similar to the MMD. As demonstrated in the experiments discussed later in this chapter, however, MMD does better, presumably because it accounts for the mean kernel similarity between samples of the same distribution.

Having defined the graph distribution kernel-GDK, $\mathcal{K}(.,.)$, our revised training set consists of tuples $\{\mathcal{G}_i, I_i, l_i\}_{i=1}^n$ with $G_{i1}^a, \cdots, G_{ik_i}^a$ sample sub-graphs in $\mathcal{G}_i$. For inferring an interaction I_*, we evaluate GDK between a test distribution $\mathcal{G}_*$ and the train distributions $\{\mathcal{G}_1, \cdots, \mathcal{G}_n\}$, from their corresponding sample sets. Then, one can apply any "kernel trick" based classifier.

2.4 Cross Representation Similarity

In the previous section, I proposed a novel algorithm for document-level extraction of interactions from AMRs. Looking forward, we will see in the experiments (Section 4.4) that AMRs yield better extraction accuracy compared to SDGs. This result suggests that using deep semantic features is very useful for the extraction task. On the other hand, the accuracy of semantic (AMR) parsing is not as good as the accuracy of shallow parsers like SDGs (Pust et al., 2015b; Flanigan et al., 2014; Wang et al., 2015b; Andreas et al., 2013; Chen and Manning, 2014). Thus, one can ask whether the joint use of semantic (AMRs) and syntactic (SDGs) parses can improve extraction accuracy further.

Abstract Meaning Representation

```
(a / activate-01
   :ARG0 (s / protein :name (n1 / name :op1 "RAS"))
   :ARG1 (s / protein :name (n2 / name :op1 "B-RAF"))
```

Stanford Typed Dependency

```
nsubj(activates-2, RAS-1)
root(ROOT-0, activates-2)
acomp(activates-2, B-RAF-3)
```

Figure 2.2: AMR and SDG parses of "*RAS activates B-RAF.*"

There are some intuitive observations that justify the joint approach: i) shallow syntactic parses may be sufficient for correctly extracting a subset of interactions; ii) semantic parsers might make mistakes that are avoidable in syntactic ones. For instance, in machine translation based semantic parsers (Pust et al., 2015b; Andreas et al., 2013), hallucinating phrasal translations may introduce an interaction/protein in a parse that is non-existent in true semantics; iii) overfit of syntactic/semantic parsers can vary from each other in a test corpus depending upon the data used in their independent trainings.

In this setting, in each evidence sentence, a candidate interaction I_i is represented by a tuple $\Sigma_i = \{G_i^a, G_i^s\}$ of sub-graphs G_i^a and G_i^s which are constructed from AMR and SDG parses of a sentence respectively. Our problem is to classify the interaction jointly on features of both sub-graphs. This can be further extended for the use of multiple evidence sentences. I now argue that the graph-kernel framework outlined above can be applied to this setting as well, with some modifications.

Let Σ_i and Σ_j be two sets of points. To apply the framework above, we need a valid kernel $K(\Sigma_i, \Sigma_j)$ defined on the joint space. One way of defining this kernel would be using similarity measures between AMRs and SDGs separately, and then combining them e.g., via linear combination. However, here I advocate a different approach, where we *flatten* the joint representation. Each candidate interaction is represented as a set of two points in the same space. This projection is a valid operation as long as we have a similarity measure between G_i^a and G_i^s (correlation between the two original dimensions). This is rather problematic since AMRs and SDGs have non-overlapping edge labels (although the space of node

labels of both representations coincide). To address this issue, for inducing this similarity measure, I next develop the approach for edge-label vector space embedding.

Let us understand what I mean by vector space embedding of edge-labels. In Figure 2.2, we have an AMR and a SDG parse of "RAS activates B-RAF". "ARG0" in the AMR and "nsubj" in SDG are conveying that "RAS" is a catalyst of the interaction "activation"; "ARG1" and "acomp" are meaning that "B-RAF" is activated. In this sentence, "ARG0" and "nsubj" are playing the same role though their higher dimensional roles, across a diversity set of sentences, would vary. Along these lines, I propose to *embed these high dimensional roles in a vector space*, termed as "edge label vectors".

2.4.1 Consistency Equations for Edge Vectors

I now describe our unsupervised algorithm that learns vector space embedding of edge labels. The algorithm works by imposing linear consistency conditions on the word vector embeddings of node labels. While I describe the algorithm using AMRs, it is directly applicable to SDGs as well.

2.4.1.1 Linear Algebraic Formulation

In my formulation, I first learn subspace embedding of edge labels (edge label matrices) and then transform it into vectors by flattening. Let us see the AMR in Figure 2.2 again. We already have word vectors embedding for terms "activate", "RAS", "B-RAF", denoted as $\boldsymbol{w}_{activate}$, $\boldsymbol{w}_{ras}$, $\boldsymbol{w}_{braf}$ respectively; a word vector $\boldsymbol{w}_i \in \mathbb{R}^{m\times 1}$. Let embedding for edge labels "ARG0" and "ARG1" be $\boldsymbol{A}_{arg0}$, $\boldsymbol{A}_{arg1}$; $\boldsymbol{A}_i \in \mathbb{R}^{m\times m}$. In this AMR, I define following linear algebraic equations.

$$\boldsymbol{w}_{activate} = \boldsymbol{A}_{arg0}^T \boldsymbol{w}_{ras}, \boldsymbol{w}_{activate} = \boldsymbol{A}_{arg1}^T \boldsymbol{w}_{braf}$$

$$\boldsymbol{A}_{arg0}^T \boldsymbol{w}_{ras} = \boldsymbol{A}_{arg1}^T \boldsymbol{w}_{braf}$$

The edge label matrices $\boldsymbol{A}_{arg0}^T$, $\boldsymbol{A}_{arg1}^T$ are linear transformations on the word vectors $\boldsymbol{w}_{ras}$, $\boldsymbol{w}_{braf}$, establishing linear consistencies between the word vectors along the edges. One can define such a set of

equations in each parent-children nodes sub-graph in a given set of manually annotated AMRs (and so applies to SDGs independent of AMRs). Along these lines, for a pair of edge labels i, j in AMRs, we have generalized equations as below.

$$\boldsymbol{Y}_i = \boldsymbol{X}_i \boldsymbol{A}_i,\ \boldsymbol{Y}_j = \boldsymbol{X}_j \boldsymbol{A}_j,\ \boldsymbol{Z}_i^{ij} \boldsymbol{A}_i = \boldsymbol{Z}_j^{ij} \boldsymbol{A}_j$$

Here $\boldsymbol{A}_i, \boldsymbol{A}_j$ are edge labels matrices. Considering n_i occurrences of edge labels i, we correspondingly have word vectors from the n_i child node labels stacked as rows in matrix $\boldsymbol{X}_i \in \mathbb{R}^{n_i \times m}$; and $\boldsymbol{Y}_i \in \mathbb{R}^{n_i \times m}$ from the parent node labels. There would be a subset of instances, $n_{ij} <= n_i, n_j$ where edge labels i and j has same parent node (occurrence of pairwise relationship between i and j). This gives $\boldsymbol{Z}_i^{ij} \in \mathbb{R}^{n_{ij} \times m}$ and $\boldsymbol{Z}_j^{ij} \in \mathbb{R}^{n_{ij} \times m}$, subsets of word vectors in $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ respectively (along rows). Along these lines, neighborhood of edge label i is defined to be: $\mathcal{N}(i) : j \in \mathcal{N}(i)\ s.t.\ n_{ij} > 0$. From the above pairwise linear consistencies, we derive linear dependencies of an $\boldsymbol{A}_i$ with its neighbors $\boldsymbol{A}_j : j \in \mathcal{N}(i)$, while also applying least square approximation.

$$\boldsymbol{X}_i^T \boldsymbol{Y}_i + \sum_{j \in \mathcal{N}(i)} {\boldsymbol{Z}_i^{ij}}^T \boldsymbol{Z}_j^{ij} \boldsymbol{A}_j = (\boldsymbol{X}_i^T \boldsymbol{X}_i + \sum_{j \in \mathcal{N}(i)} {\boldsymbol{Z}_i^{ij}}^T \boldsymbol{Z}_i^{ij}) \boldsymbol{A}_i$$

Exploiting the block structure in the linear program, I propose an algorithm that is a variant of "Gauss-Seidel" method (Demmel, 1997; Niethammer et al., 1984).

Algorithm 1. *(a) Initialize:*

$$\boldsymbol{A}_i^{(0)} = (\boldsymbol{X}_i^T \boldsymbol{X}_i)^{-1} \boldsymbol{X}_i^T \boldsymbol{Y}_i.$$

(b) Iteratively update $\boldsymbol{A}_i^{(t+1)}$ until convergence:

$$\boldsymbol{A}_i^{(t+1)} = \boldsymbol{B}[\boldsymbol{X}_i^T \boldsymbol{Y}_i + \sum_{j \in \mathcal{N}(i)} {\boldsymbol{Z}_i^{ij}}^T \boldsymbol{Z}_j^{ij} \boldsymbol{A}_j^{(t)}]$$

$$\boldsymbol{B} = [\boldsymbol{X}_i^T \boldsymbol{X}_i + \sum_{j \in \mathcal{N}(i)} {\boldsymbol{Z}_i^{ij}}^T \boldsymbol{Z}_i^{ij}]^{-1}$$

(c) Set the inverse edge label matrices:

$$\boldsymbol{A}_{i_{inv}} = \boldsymbol{A}_i^{-1}.$$

Theorem 6.2 in (Demmel, 1997)[p. 287, chapter 6] states that the Gauss-Seidel method converges if the linear transformation matrix in a linear program is strictly row diagonal dominant (Niethammer et al., 1984). In the above formulation, diagonal blocks dominate the non-diagonal ones row-wise. Thus, Algorithm 1 should converge to an optimum.

Using Algorithm 1, I learned edge label matrices in AMRs and SDGs independently on corresponding AMRs and SDGs annotations from 2500 bio-sentences (high accuracy auto-parse for SDGs). Convergence was fast for both AMRs and SDGs (log error drops from 10.14 to 10.02 for AMRs, and from 30 to approx. 10 for SDGs).

Next, I flatten an edge label matrix $\boldsymbol{A}_i \in \mathbb{R}^{m \times m}$ to a corresponding edge label vector $\boldsymbol{e}_i \in \mathbb{R}^{m^2 \times 1}$, and then redefine $k_e(x, y)$ in (2.1) using the sparse RBF kernel.

$$k_e(x, y) = \exp\left((\boldsymbol{e}_x^T \boldsymbol{e}_y - 1)/\beta\right) \left((\boldsymbol{e}_x^T \boldsymbol{e}_y - \alpha)/(1 - \alpha)\right)_+$$

This redefinition enables to define kernel similarity between AMRs and SDGs. One can either use my original formulation where a single AMR/SDG sub-graph is classified using training sub-graphs from both AMRs and SDGs, and then the inference with maximum score-MSI (Bunescu et al., 2006) is chosen. Another option, preferable, is to consider the set $\{G_i^a, G_i^s\}$ as samples of a graph distribution $\mathcal{G}_i$ representing an interaction I_i. Generalizing it further, $\mathcal{G}_i$ has samples set $\{G_{i1}^a, \cdots, G_{ik_i^a}^a, G_{i1}^s, \cdots, G_{ik_i^s}^s\}$, containing k_i^a, k_i^s number of sub-graphs in AMRs and SDGs respectively from multiple sentences in a document, all for classifying I_i. With this graph distribution representation, one can apply my GDK from Section 2.3 and then infer using a "kernel trick" based classifier. This final formulation gives the best results in our experiments discussed next.

2.5 Experimental Evaluation

I evaluated the proposed algorithm on two data sets.

PubMed45 Dataset

This dataset has 400 manual and 3k auto parses of AMRs (and 3.4k auto parses of SDGs);[5] AMRs auto-parses are from 45 PubMed articles on cancer. From the 3.4k AMRs, I extract 25k subgraphs representing 20k interactions (valid/invalid); same applies to SDGs. This is the primary data for the evaluation.

I found that for both AMR and SGD based methods, a part of the extraction error can be attributed to poor recognition of named entities. To minimize this effect, and to isolate errors that are specific to the extraction methods themselves, I follow the footsteps of the previous studies, and take a filtered subset of the interactions (approx. 10k out of 20k). I refer to this data subset as "PubMed45" and the super set as "PubMed45-ERN" (for entity recognition noise).

AIMed Dataset

This is a publicly available dataset[6], which contains about 2000 sentences from 225 abstracts. In contrast to PubMed45, this dataset is very limited as it describes only whether a given pair of proteins interact or not, without specifying the interaction type. Nevertheless, I find it useful to include this dataset in the evaluation since it enables us to compare the results with other reported methods.

2.5.1 Evaluation Settings

In a typical evaluation scenario, validation is performed by random sub-sampling of labeled interactions (at sentence level) for a test subset, and using the rest as a training set. This sentence-level validation approach is not always appropriate for extracting protein interactions (Tikk et al., 2010), since interactions from a single/multiple sentences in a document can be correlated. Such correlations can lead to information leakage between training and test sets (artificial match, not encountered in real settings). For instance,

[5] not the same 2.5k sentences used in learning edge label vectors
[6] `http://corpora.informatik.hu-berlin.de`

Methods	PubMed45-ERN	PubMed45	AIMed
SDG (SLI)	0.25 ± 0.16 $(0.42, 0.29)$	0.32 ± 0.18 $(0.50, 0.35)$	0.27 ± 0.12 $(0.54, 0.22)$
AMR (SLI)	0.33 ± 0.16 $(0.33, 0.45)$	0.45 ± 0.25 $(0.58, 0.43)$	0.39 ± 0.05 $(0.53, 0.33)$
SDG (MSI)	0.24 ± 0.14 $(0.39, 0.28)$	0.33 ± 0.17 $(0.50, 0.34)$	0.39 ± 0.09 $(0.51, 0.38)$
AMR (MSI)	0.32 ± 0.14 $(0.30, 0.45)$	0.45 ± 0.24 $(0.56, 0.44)$	0.51 ± 0.11 $(0.49, 0.56)$
SDG (GDK)	0.25 ± 0.16 $(0.33, 0.31)$	0.38 ± 0.15 $(0.32, 0.61)$	0.47 ± 0.08 $(0.41, 0.58)$
AMR (GDK)	0.35 ± 0.16 $(0.31, 0.51)$	0.51 ± 0.23 $(0.59, 0.49)$	0.51 ± 0.11 $(0.43, 0.65)$
AMR-SDG (MSI)	0.33 ± 0.18 $(0.29, 0.54)$	0.47 ± 0.24 $(0.50, 0.53)$	$\mathbf{0.55 \pm 0.09}$ $(0.46, 0.73)$
AMR-SDG (GDK)	$\mathbf{0.38 \pm 0.16}$ $(0.33, 0.55)$	$\mathbf{0.57 \pm 0.23}$ $(0.63, 0.54)$	0.52 ± 0.09 $(0.43, 0.67)$
Data Statistics			
Positive ratio	0.07 ± 0.04	0.19 ± 0.14	0.37 ± 0.11
Train-Test Div.	0.014 ± 0.019	0.041 ± 0.069	0.005 ± 0.002

Table 2.2: F1 score statistics. "SLI" is sentence level inference; "MSI" refers to maximum score inference at document level; "GDK" denotes Graph distribution kernel based inference at document level. Precision, recall statistics are presented as (mean-precision, mean-recall) tuples.

in (Mooney and Bunescu, 2005), the reported F1 score from the random validation in the AIMed data is approx. 0.5. My algorithm, even using SDGs, gives 0.66 F1 score in those settings. However, the performance drops significantly when an independent test document is processed. Therefore, for a realistic evaluation, I divide data sets at documents level into approx. 10 subsets such that there is minimal match between a subset, chosen as test set, and the rest of sub sets used for training a kernel classifier. In the PubMed45 data sets, the 45 articles are clustered into 11 subsets by clustering PubMed-Ids (training data also includes gold annotations). In AIMed, abstracts are clustered into 10 subsets on abstract-ids. In each of 25 independent test runs (5 for AIMed data) on a single test subset, 80% interactions are randomly sub sampled from the test subset and same percent from the train data.

For the classification, I use the LIBSVM implementation of Kernel Support Vector Machines (Chang and Lin, 2011) with the sklearn python wrapper [7]. Specifically, I used settings { probability=True, $C = 1$, class_weight=auto }.

2.5.2 Evaluation Results

I categorize all methods evaluated below as follows: i) *Sentence Level Inference*-SLI; [8] ii) document level using *Maximum Score Inference*-MSI (Bunescu et al., 2006); and iii) document-level inference on all the subgraphs using the *Graph Distribution Kernel* (GDK). In each of the categories, AMRs, SDGs are used independently, and then jointly. Edge label vectors are used only when AMRs and SDGs are jointly used, referred as "AMR-SDG".

Table 3.2(a) shows the F1 score statistics for all the experiments. In addition, the mean of precision and recall values are presented as (precision, recall) tuples in the same table. For most of the following discussion, I focus on F1 scores only to keep the exposition simple.

Before going into detailed discussion of the results, I make the following two observations. First, we can see that, in all methods (including our GDK and baselines), we obtain much better accuracy using AMRs compared to SDGs. This result is remarkable, especially taking into account the fact that the accuracy of semantic parsing is still significantly lower when compared to syntactic parsing. And second, observe that the overall accuracy numbers are considerably lower for the PubMed45-ERN data, compared to the filtered data PubMed45.

Let us focus on document-level extraction using MSI. We do not see much improvement in numbers compared to SLI for our PubMed45 data. On the other hand, even this simple MSI technique works for the restricted extraction settings in the AIMed data. MSI works for AIMed data probably because there are multiple sub-graph evidences with varying interaction types (root node in subgraphs), even in a single sentence, all representing same protein-protein pair interaction. This high number of evidences at document level, should give a boost in performance even using MSI.

[7] `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

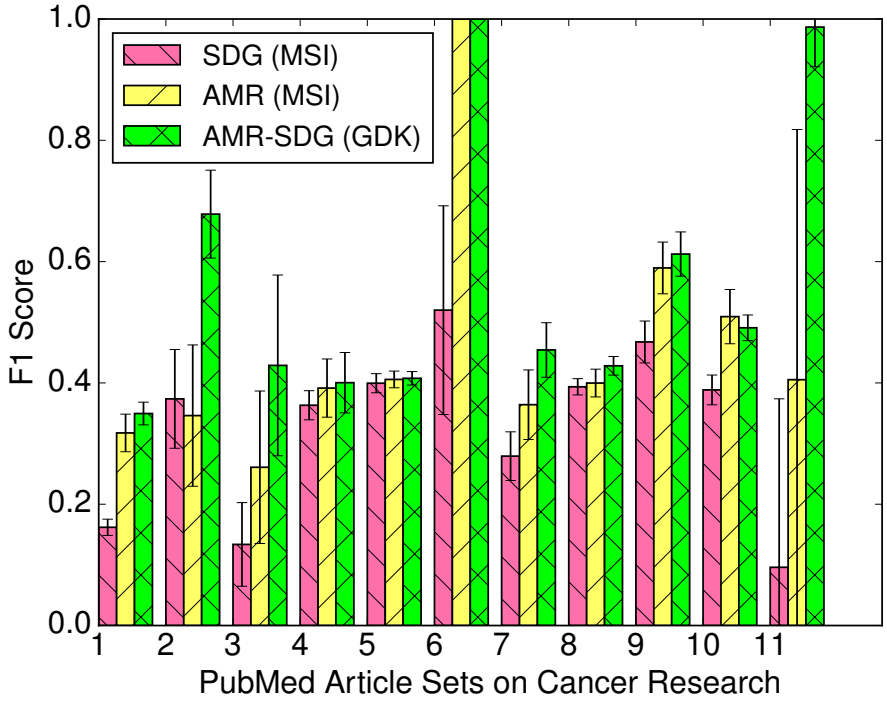
[8] Note that even for the sentence level inference, the training/test division is done on document level.

Next, I consider document-level extraction using the proposed GDK method with the MMD metric. Comparing against the baseline SLI, we see a significant improvement for all data sets and in both AMRs and SDGs (although the improvement in PubMed45-ERN is relatively small). The effect of the noise in entity recognition can be a possible reason why GDK does not work so well in this data compared to the other two data sets. Here, we also see that: a) GDK method performs better than the document level baseline MSI; and b) AMRs perform better than SDGs with GDK method also.

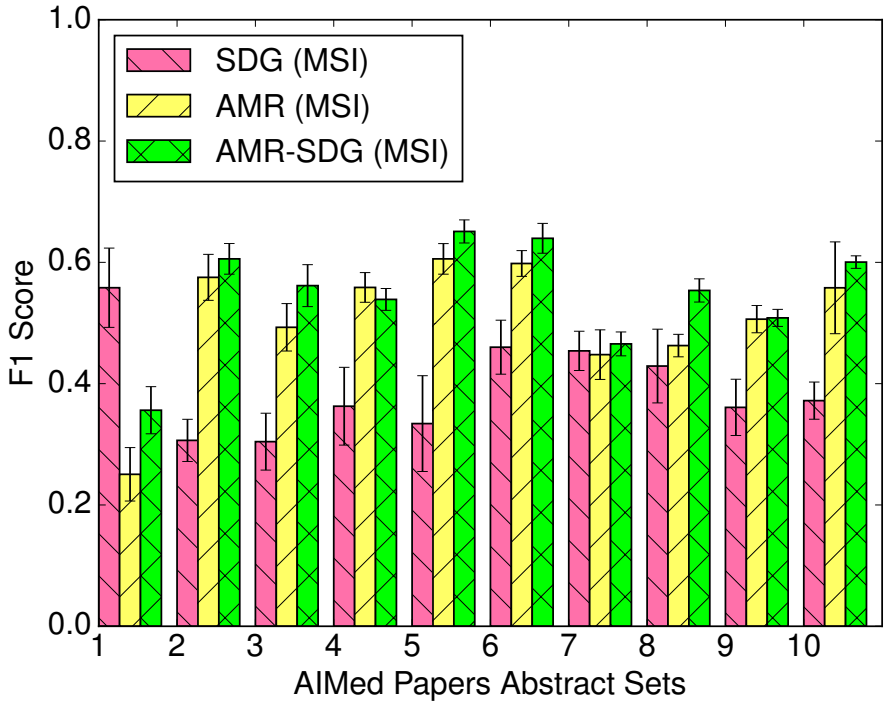
Let us now consider the results of extraction using both AMRs and SDGs jointly. Here I evaluate MSI and GDK, both using our edge label vectors. My primary observation here is that the joint inference using both AMRs and SDGs improves the extraction accuracy across all datasets. Furthremore, in both PubMed45 datasets, the proposed GDK method is a more suitable choice for the joint inference on AMRs and SDGs. As we can see, comparing to GDK for AMRs only, F1 points increment from 0.35 to 0.38 for the PubMed45-ERN data, and from 0.51 to 0.57 for the PubMed45 data. For the AIMed dataset, on the other hand, the best result (F1 score of 0.55) is obtained when one uses the baseline MSI for the joint inference on AMRs and SDGs.

To get more insights, I now consider (mean-precision, mean-recall) tuples shown in the Table 3.2(a). The general trend is that the AMRs lead to higher recall compared to the SDGs. In the PubMed45-ERN data set, this increase in the recall is at cost of a drop in the precision values. Since the entity types are noisy in this data set, this drop in the precision numbers is not completely surprising (note that the F1 scores still increase). With the use of the GDK method in the same data set, however, the precision drop (SDGs to AMRs) becomes negligible, while the recall still increases significantly. In the data set PubMed45 (the one without noise in the entity types), both the precision and recall are generally higher for the AMRs compared to the SDGs. Again, there is an exception for the GDK approach, for which the recall decreases slightly. However, the corresponding precision almost doubles.

For a more fine-grained comparison between the methods, I plot F1 score for each individual test set in Figure 2.3. Here, I compare the baselines, “AMR (MSI)”, “SDG (MSI)” against the “AMR-SDG (GDK)” in PubMed45 data set (and, “AMR-SDG (MSI)” for “AIMed” dataset). We see a general trend, across



(a) PubMed45



(b) AIMed

Figure 2.3: Comparison of extraction accuracy (F1 score)

	MMD	**KL-D**	**CK**
SDG	0.25 ± 0.16 $(0.33, 0.31)$	0.21 ± 0.17 $(0.59, 0.21)$	0.26 ± 0.13 $(0.29, 0.38)$
AMR	0.35 ± 0.16 $(0.31, 0.51)$	0.37 ± 0.17 $(0.50, 0.41)$	0.29 ± 0.13 $(0.28, 0.39)$

Table 2.3: Comparison of F1 scores for different divergence metrics used with GDK. The evaluation is on PubMed45-ERN dataset. "KL-D" and "CK" stand for Kullback-Leibler divergence and Cross Kernels, respectively.

all test subsets, of AMRs being more accurate than SDGs and the joint use of two improving even upon AMRs. Though, there are some exceptions where the difference is marginal between the three. From a cross checking, I find that such exceptions occur when there is relatively more information leakage between train-test, i.e. less train-test divergence. I use Maximum Mean Discrepancy-MMD for evaluating this train-test divergence (originally used for defining GDK in Section 2.3. I find that our GDK technique is more suitable when $MMD > 0.01$ (MMD is normalized metric for a normalized graph kernel).

The results for the GDK method described above are specific to the MMD metric. I also evaluated GDK using two other metrics (KL-D and cross kernels), specifically on "PubMed45-ERN" dataset, as presented in Table 2.3. Here, as in Table 3.2(a), I present (mean-precision, mean-recall) tuples too. We can see that MMD and KL-D metrics, both, perform equally well for AMR whereas MMD does better in case of SDG. CK (cross kernels), which is a relatively naive approach, also performs reasonably well, although for the AMRs it performs worse compared to MMD and KL-D. For the precision and recall numbers in the Table 2.3, we see similar trends as reported in Table 3.2(a). We observe that the recall numbers increase for the AMRs compared to the SDGs (the metric CK is an exception with negligible increase). Also, comparing KL-D against MMD, we see the former favors (significantly) higher precision, albeit at the expense of lower recall values.

2.6 Related Work

There have been different lines of work for extracting protein extractions. Pattern-matching based systems (either manual or semi-automated) usually yield high precision but low recall (Hunter et al., 2008; Krallinger et al., 2008; Hakenberg et al., 2008; Hahn and Surdeanu, 2015). Kernel-based methods based on various convolution kernels have also been developed for the extraction task (Chang et al., 2016; Tikk et al., 2010; Miwa et al., 2009; Airola et al., 2008; Mooney and Bunescu, 2005). Some approaches work on string rather than parses (Mooney and Bunescu, 2005). The above mentioned works either rely on text or its shallow parses, none using semantic parsing for the extraction task. Also, most works consider only protein-protein interactions while ignoring interaction types. Some recent works used distant supervision to obtain a large data set of protein-protein pairs for their experiments (Mallory et al., 2015).

Document-level extraction has been explored in the past (Skounakis and Craven, 2003; Bunescu et al., 2006). These works classify at sentence level and then combine the inferences whereas we propose to infer jointly on all the sentences at document level.

Previously, the idea of linear relational embedding has been explored in (Paccanaro and Hinton, 2000), where triples of concepts and relation types between those concepts are (jointly) embedded in some latent space. Neural networks have also been employed for joint embedding (Bordes et al., 2014). Here I advocate for a factored embedding where concepts (node labels) are embedded first using plain text, and then relations (edge labels) are embedded in a linear sub-space.

2.7 Chapter Summary

In summary, I have developed and validated a method for extracting biomolecular interactions that, for the first time, uses *deep semantic parses* of biomedical text (AMRs). I have presented a novel algorithm, which relies on *Graph Distribution Kernels* (GDK) for document-level extraction of interactions from a set of AMRs in a document. GDK can operate on both AMR and SDG parses of sentences jointly. The rationale behind this hybrid approach is that while neither parsing is perfect, their combination can yield

superior results. Indeed, the experimental results suggest that the proposed approach outperforms the baselines, especially in the practically relevant scenario when there is a noticeable mismatch between the training and test sets.

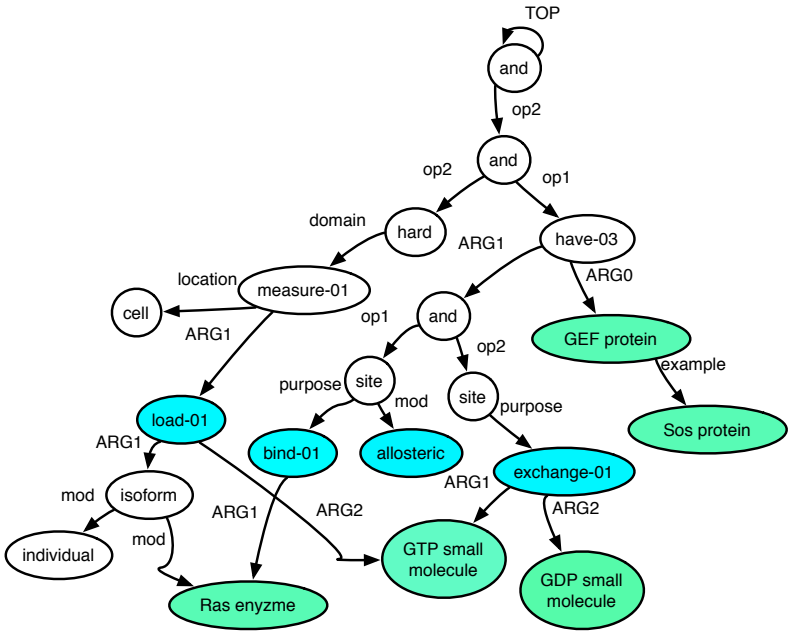
To facilitate the joint approach, I have proposed a novel edge vector space embedding method to assess similarity between different types of parses. I believe this notion of edge-similarly is quite general and will have applicability for a wider class of problems involving graph kernels. As a future work, one can validate this framework on a number of problems such as improving accuracy in AMRs parsing with SDGs.

Chapter 3

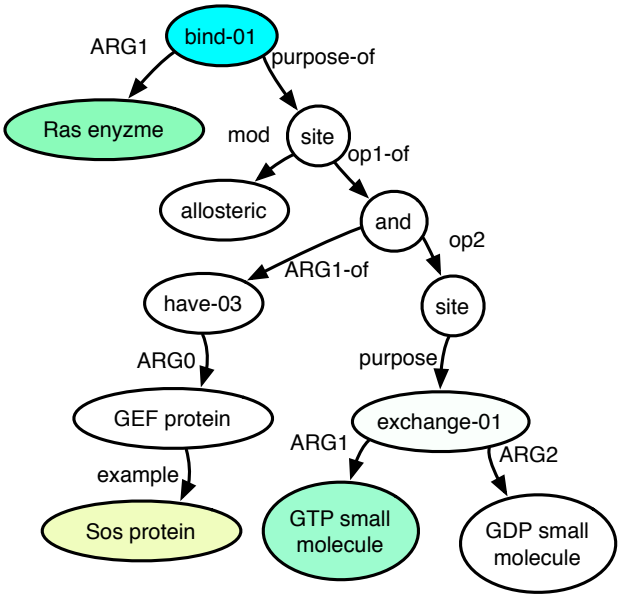
Stochastic Learning of Nonstationary Kernels for Natural Language Modeling

Despite the success of convolution kernel-based methods in various NLP tasks (Collins and Duffy, 2001; Moschitti, 2006; Tikk et al., 2010; Qian and Zhou, 2012; Srivastava et al., 2013; Hovy et al., 2013; Filice et al., 2015; Tymoshenko et al., 2016), there are two important issues limiting their practicality in real-world applications. First, convolution kernels are not *flexible* enough to adequately model rich natural language representations, as they typically depend only on a few tunable parameters. This inherent rigidity prevents kernel-based methods from properly adapting to a given task, as opposed to, for instance, neural networks that typically have millions of parameters that are learned from data and show state-of-the-art results for a number of NLP problems (Collobert and Weston, 2008; Sundermeyer et al., 2012; Chen and Manning, 2014; Sutskever et al., 2014; Kalchbrenner et al., 2014; Luong et al., 2015; Kumar et al., 2016). The second major issue with convolution kernels is the high computational cost, for both learning and inference, as computing kernel similarity between a pair of discrete structures costs polynomial time in its size. For instance, classifying a new data point based on N labeled examples requires calculation of N pairwise similarities, which might be prohibitively expensive for many real-world problems.

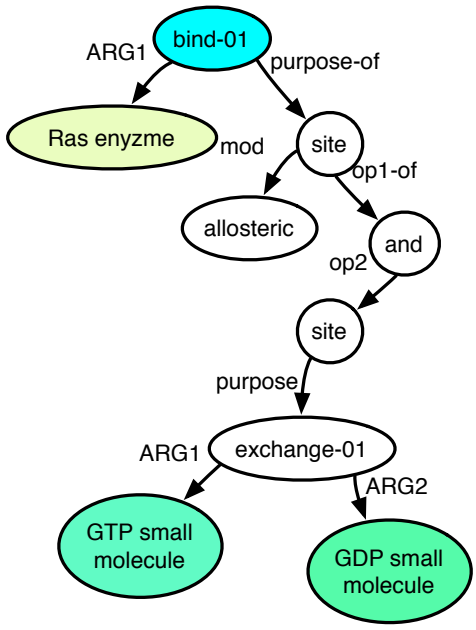
I address the first problem by proposing a nonstationary extension to the conventional convolution kernels, by introducing a novel, task-dependent parameterization of the kernel similarity function for better expressiveness and flexibility. Those parameters, which need to be inferred from the data, are defined in a



(a) AMR Graph



(b) correct (positive label)



(c) incorrect (negative label)

Figure 3.1: In 3.1(a), a sentence is parsed into a semantic graph. Then the two candidate hypothesis about different biomolecular interactions (structured prediction candidates) are generated automatically. According to the text, a valid hypothesis is that *Sos catalyzes binding between Ras and GTP*, while the alternative hypothesis *Ras catalyzes binding between GTP and GDP* is false; each of those hypotheses corresponds to one of the post-processed subgraphs shown in 3.1(b) and 3.1(c), respectively.

way that allows the model to ignore substructures irrelevant for a given task when computing kernel similarity. For example, in Table 3.1, the screened-out tuples are ignored when computing path kernels. This model should be highly relevant to problems where a structure is large in size, and only some sub-structures are relevant for a given task; for instance, in biomedical domain, explaining a biological phenomenon may require either long sentences or multiple sentences.

To address the scalability issue, I introduce an efficient stochastic algorithm for learning the parameters of the convolution kernels. Generally speaking, exact learning of those parameters would require constructing an $N \times N$ Gram matrix of pairwise similarities between the labeled data points, which become computationally expensive for large N. Instead, here I propose an appropriate loss function defined over a k-NN graph,[1] an approach similar to that used for distance metric learning (Weinberger et al., 2006; Weinberger and Saul, 2009). Kernel-based locality-sensitive hashing (LSH) allows computing a k-NN graph approximately,[2] with as low as $O(N^{1.5})$ (Kulis and Grauman, 2012). Since k-NN graphs are built several times during learning, hashing may not be enough for scalable learning, and may therefore necessitate the use of the introduced stochastic-sampling on k-NN graphs in the proposed algorithm.

The main contributions of this chapter (Garg et al., 2018), are as follows:

(i) I propose *a nonstationary extension to the conventional kernels to achieve better expressiveness* and flexibility for general NLP tasks;

(ii) I introduce *an efficient stochastic algorithm for learning the parameters* of the convolution kernel, optionally using any kernel-LSH;

(iii) I validate the proposed approach in experiments, for the relation extraction task, and *find a significant increase in accuracies* across multiple datasets containing 100k (unique) labeled discrete structures.

In addition to the above contributions, another novelty is the use of kernel-LSH in NLP, although we note that (non-kernel) LSH has been used for NLP tasks such as documents retrieval, machine translation, etc (Bawa et al., 2005; Li et al., 2014; Wurzer et al., 2015; Shi and Knight, 2017).

[1] In this work, k-NNs give better accuracy than SVMs.
[2] Known to work in computer vision, but unexplored in NLP.

Positive labeled: ~~(protein, example) (protein, arg0)~~ (have, arg1-of) (and, op1-of) (site, purpose-of) (bind) (bind) (arg1, enzyme) (bind) (purpose-of, site) (op1-of, and) ~~(op2, site)~~ (purpose, exchange) (arg1, small-molecule)
Negative labeled: (enyzme, arg1) (bind) (bind) (purpose-of, site) (op1-of, and) ~~(op2, site)~~ (purpose, exchange) (arg1, small-molecule) (bind) (purpose-of, site) (op1-of, and) ~~(op2, site) (arg2, small-molecule)~~

Table 3.1: The sequences of tuples are obtained from traversing the subgraphs, in Figure 3.1(b) and 3.1(c), respectively.

3.1 Problem Statement

In Figure 3.1, a text sentence "Furthermore, GEFs such as Sos have allosteric sites for Ras binding as well as sites for GDP/GTP exchange, and it is hard to measure GTP loading on individual Ras isoforms in cells", is parsed into an Abstract Meaning Representation (AMR) graph (Banarescu et al., 2013); we use the translation based AMR parser proposed in (Pust et al., 2015a). We extract structured information about biomolecular interactions, each involving the interaction type, and two or three participants with a catalyst or domain role, from the AMR graph.

In the AMR graph, the blue nodes are for potential interaction types, and the green nodes are potential participants. Based on the list of interaction types, and participants, multiple candidate bio-molecular interactions are generated automatically; these are referred as the candidates for structured prediction, each one to be binary classified. As per the meaning of the source text, a candidate is manually annotated as positive (correct) or negative (incorrect), for training/test purposes.

As in the previous chapter, for a given candidate, a corresponding subgraph is extracted from the AMR, and post-processed such that the interaction-type becomes the root node in the subgraph, and the participants become leaf nodes; see Figure 3.1(b), 3.1(c). In the subgraphs, yellow color is for the catalyst role and green for the domain role. Thus, in order to classify a candidate bio-molecular interaction, its corresponding subgraph can be classified as a proxy; same applies to the labeled candidates in a train set. Also, from a subgraph, a sequence can be generated by traversals between the root node and the leaf nodes.

In general, we are interested in classification of natural language structures such as sequences and graphs.

3.2 Convolution Kernels and Locality Sensitive Hashing for k-NN Approximations

Many NLP tasks involve a classification problem where, given a set of discrete structures $\boldsymbol{S}$ and the associated class labels $\boldsymbol{y}$, $(\boldsymbol{S}, \boldsymbol{y}) = \{S_i, y_i\}_{i=1}^{N}$, the goal is infer the correct label y_* for a test structure S_*. My approach for the problem is convolution kernel based k-NN classifiers.

3.2.1 Convolution Kernels

Convolution kernels belong to a class of kernels that compute similarity between discrete structures (Haussler, 1999; Collins and Duffy, 2001). In essence, convolution kernel similarity function $K(S_i, S_j)$ between two discrete structures S_i and S_j, is defined in terms of function $k(.,.)$ that characterizes similarity between a pair of tuples or labels.

Path/Subsequence Kernels

Let S_i and S_j be two sequences of tuples, as in Table 3.1. (Mooney and Bunescu, 2005) define the kernel as:

$$K(S_i, S_j) = \sum_{\boldsymbol{i},\boldsymbol{j}:|\boldsymbol{i}|=|\boldsymbol{j}|} \prod_{k=1}^{|\boldsymbol{i}|} k(S_i(\boldsymbol{i}_k), S_j(\boldsymbol{j}_k))\lambda^{l(\boldsymbol{i})+l(\boldsymbol{j})}.$$

Here, $k(S_i(\boldsymbol{i}_k), S_j(\boldsymbol{j}_k))$ is the similarity between the k_{th} tuples in the subsequences $\boldsymbol{i}$ and $\boldsymbol{j}$, of equal length; $l(.)$ is the actual length of a subsequence in the corresponding sequence, i.e., the difference between the end index and start index (subsequences do not have to be contiguous); $\lambda \in (0, 1)$ is used to penalize the long subsequences.

Tree Kernels

In (Zelenko et al., 2003), the convolution kernel defines the similarity between two graphs/trees T_i and T_j, for example between the trees in Figure 3.1(b) and 3.1(c), as:

$$K(T_i, T_j) = k(T_i.r, T_j.r)(k(T_i.r, T_j.r) + \sum_{\boldsymbol{i},\boldsymbol{j}:l(\boldsymbol{i})=l(\boldsymbol{j})} \lambda^{l(\boldsymbol{i})} \sum_{s=1,\cdots,l(\boldsymbol{i})} K(T_i[\boldsymbol{i}[s]], T_j[\boldsymbol{j}[s]]) \prod_{s=1,\cdots,l(\boldsymbol{i})} k(T_i[\boldsymbol{i}[s]].r, T_j[\boldsymbol{j}[s]].r)).$$

Here $\boldsymbol{i}, \boldsymbol{j}$ are child subsequences under the root nodes $T_i.r, T_j.r$ and $\lambda \in (0, 1)$ as shown above; $\boldsymbol{i} = (i_1, \cdots, i_l)$ and $T_i[\boldsymbol{i}[s]]$ are subtrees rooted at the $\boldsymbol{i}[s]_{th}$ child node of $T_i.r$.

For both kernels above, dynamic programming is used for efficient computation.

3.2.2 Hashing for Constructing k-NN Graphs

A k-NN classifier finds the k-nearest neighbors of a data point per a given kernel function, and then infers its class label from the class labels of the neighbors.

In computer vision domain, a well-known technique is to approximate a kernel-k-NN graph using kernel-locality-sensitive hashing (kernel-LSH or KLSH). A smaller subset of data points, $\boldsymbol{S}^R$, of size M, is randomly sub-selected from the dataset $\boldsymbol{S}$ of size N, with M as low as $N^{\frac{1}{2}}$. For computing a hash code of H binary bits of a data point, its kernel similarity is computed w.r.t. the data points in the set $\boldsymbol{S}^R$, as an input to any kernel-LSH algorithm; a binary hashcode corresponds to a hash bucket. In the final step, the nearest neighbors of a data point are found by computing its kernel similarity w.r.t. the data points in the same hash bucket, and optionally the neighboring ones.

In the following, we brief on two KLSH approaches below. One of the approaches is proposed by (Kulis and Grauman, 2009), and the another one I develop as a *minor contribution* of this thesis.

KLSH by (Kulis and Grauman, 2009)

One approach to building a binary hash function is to randomly sample a linear-hyperplane in the feature

Algorithm 1 Random kNN based Algorithm for Computing KLSH Codes.

Require: Kernel similarity vector $\boldsymbol{k}$ of size M, computed for a structure S; sub-sampling parameter, α, for randomizing hash function; the number of hash functions, H. `% Random subsets, fixed across structure inputs`

for $j = 1 \rightarrow H$ **do**
  $\boldsymbol{r}^h_{j_1} \leftarrow$ randomSubset(M, α)
  $\boldsymbol{r}^h_{j_2} \leftarrow$ randomSubset(M, α)
$\boldsymbol{c} \leftarrow \boldsymbol{0}$
for $j = 1 \rightarrow H$ **do**
  $\boldsymbol{k}_{j_1} \leftarrow \boldsymbol{k}(\boldsymbol{r}^h_{j_1})$
  $\boldsymbol{k}_{j_2} \leftarrow \boldsymbol{k}(\boldsymbol{r}^h_{j_2})$
  if max($\boldsymbol{k}_{j_1}$) > max($\boldsymbol{k}_{j_2}$) **then**
    $\boldsymbol{c}(j) \leftarrow 0$
  else
    $\boldsymbol{c}(j) \leftarrow 1$
return $\boldsymbol{c}$

space implied from the kernel functions; for $\boldsymbol{S}^R$ of size sublinear in the size of dataset $\boldsymbol{S}$, one can obtain only an approximation for the random sampling of linear hyperplane (Kulis and Grauman, 2012). This approach has limitations in terms of computational cost and numerical instability. The number of data points in different buckets can be highly uneven with this hash function, as per the experiments discussed later in this chapter, leading to an increase in kernel computations.

Random kNN based KLSH

As a minor contribution, I build a simple technique, for kernel based binary hash functions. The main advantages of this hashing approach are, less computational cost as well as the use of non-linear boundaries instead of linear ones (in the kernel implied feature space).

To build a random binary hash function $h_j(.)$, we randomly sample two subsets, $\boldsymbol{S}^1_l, \boldsymbol{S}^2_l$, from $\boldsymbol{S}^R$. Having $\boldsymbol{S}^1_l, \boldsymbol{S}^2_l$, and a given data point S_i, the first nearest neighbor of S_i is found in both of the subsets using the convolution kernel similarities. Depending on which of the two data points is the closest to S_i, a binary bit value is obtained. See the pseudo code in Algorithm 1. In the experiments, I find this hash function as accurate as the above, yet much cheaper in kernel computations.

The number of data points across different buckets is relatively uniform with this KLSH algorithm, compared to the one in (Kulis and Grauman, 2009), since kNN models are highly data driven. This keeps

the number of kernel computations low, when finding nearest neighbors in neighboring buckets. Also, unlike their algorithm, this algorithm doesn't require computing a kernel matrix between the data points in $\boldsymbol{S}^R$.

This hashing approach is similar to the one, based on SVMs, proposed in (Joly and Buisson, 2011).

3.3 Nonstationary Convolution Kernels

I propose a generic approach to extend any convolution kernel to a non-stationary one, for higher expressibility and generalization.

Definition 1 (Stationary kernel (Genton, 2001))**.** *A* stationary *kernel, between vectors $\boldsymbol{w}_i, \boldsymbol{w}_j \in \mathbb{R}^d$, is the one which is translation invariant:*

$$k(\boldsymbol{w}_i, \boldsymbol{w}_j) = k^S(\boldsymbol{w}_i - \boldsymbol{w}_j),$$

that means, it depends only upon the lag vector between $\boldsymbol{w}_i$ and $\boldsymbol{w}_j$, and not the data points themselves.

For NLP context, stationarity in convolution kernels is formalized in Theorem 1.

Theorem 1. *A convolution kernel $K(.,.)$, that is itself a function of the kernel $k(.,.)$, is stationary if $k(.,.)$ is stationary.*

Proof Sketch. Suppose we have a vocabulary set, $\{l_1, \cdots, l_p, \cdots, l_{2p}\}$, and we randomly generate a set of discrete structures $\mathcal{S} = \{S_1, \cdots, S_N\}$, using $l_1, \cdots, l_p$. For kernel $k(.,.)$, that defines similarity between a pair of labels, consider a case of stationarity, $k(l_i, l_j) = k(l_{i+p}, l_j) = k(l_i, l_{j+p}); i, j \in \{1, \cdots, p\}$, where its value is invariant w.r.t. to the translation of a label l_i to l_{i+p}. In the structures, replacing labels $l_1, \cdots, l_p$ with $l_{p+1}, \cdots, l_{2p}$ respectively, we obtain a set of new structures $\bar{\mathcal{S}} = \{\bar{S}_1, \cdots, \bar{S}_N\}$. Using a convolution kernel $K(.,.)$, as a function of $k(.,.)$, we obtain same (kernel) Gram matrix on the set $\bar{\mathcal{S}}$ as for $\mathcal{S}$. Thus $K(.,.)$ is also invariant w.r.t. the translation of structures set $\mathcal{S}$ to $\bar{\mathcal{S}}$, hence a stationary kernel (Def. 1). □

Some examples of kernels $k(.,.)$ are:

$$k(i,j) = \mathbb{I}(i=j),\ k(i,j) = \exp(-||\boldsymbol{w}_i - \boldsymbol{w}_j||_2^2),$$
$$k(i,j) = \exp(\boldsymbol{w}_i^T \boldsymbol{w}_j - 1).$$

Herein, i, j are words (node/edge labels), and $\boldsymbol{w}_i$, $\boldsymbol{w}_j$ are word vectors respectively; $\mathbb{I}(.)$ is an indicator function. The first two kernels are stationary. Whereas the third one is nonstationary mathematically (yet not expressive enough, as such), though it relies only on the cosine similarity between the word vectors, not the word-vectors or words themselves. This approach is generic enough for not only extending a stationary convolution kernel, but a non-stationary convolution kernel as well, accounting for the cases as the latter one, *for a higher flexibility and generalization*.

Theorem 2. *Any convolution kernel $K(.,.)$, can be extended to a valid nonstationary convolution kernel $K^{NS}(.,.)$, by extending the stationary/nonstationary kernel function $k(.,.)$ to $k^{NS}(.,.)$ as in (3.1), with a deterministic function $\sigma(.)$ of any form.*

$$k^{NS}(i,j) = \sigma(i)k(i,j)\sigma(j) \tag{3.1}$$

Proof Sketch. The extended kernel function $k^{NS}(i,j)$ is a valid kernel (Rasmussen and Williams, 2006, p. 95), and therefore, $K^{NS}(.,.)$, as a function of $k^{NS}(.,.)$, is also a valid convolution kernel. For establishing the nonstationarity property, following the proof of Theorem 1, if using $K^{NS}(.,.)$, we obtain a (kernel) Gram matrix on the set $\bar{\mathcal{S}}$ that is different from the set $\mathcal{S}$ because $\sigma(l_i) \neq \sigma(l_{i+p})\ \forall i \in \{1, \cdots, p\}$ for an arbitrary selection of $\sigma(.)$. Therefore $K^{NS}(.,.)$ is not invariant w.r.t. the translation of set $\mathcal{S}$ to $\bar{\mathcal{S}}$, hence a nonstationary kernel (Def. 1). □

In natural language modeling problems, since $k(.,.)$ operates on labels (or a tuple of labels), $\sigma(.)$ can be thought of as 1-D embedding of the vocabulary for labels. For $\sigma(.) \in \mathbb{R}_{\geq 0}$, it is like the strength of a label in the context of computing the convolution kernel similarity. *For instance, if $\sigma(i) = 0$, it means*

that the label i should be completely ignored when computing a convolution kernel similarity of a discrete structure (tree, path, etc.), that contains the (node or edge) label i, w.r.t. another discrete structure, or itself. Thus, we see that these additional parameters, introduced with the nonstationary model, allow convolution kernels to be expressive enough to decide if some substructures in a discrete structure should be ignored. One can also define $\sigma(.)$ on the vector space representation of the labels (word vectors), like $\sigma(\boldsymbol{w}_i)$ instead of $\sigma(i)$.

3.3.1 Nonstationarity for Skipping Sub-Structures

Suppose we are interested in computing the convolution kernel similarity between the paths of tuples —as in Table 3.1 or between the two trees in Figure 3.1(b), 3.1(c). In both cases, the basic kernel function, $k(.,.)$ operates on a pair of tuples. The nonstationary kernel on tuples can be defined as:

$$k^{NS}((e_i, n_i),(e_j, n_j)) = \sigma_{e_i} k_e(e_i, e_j)\sigma_{e_j} k_n(n_i, n_j).$$

The similarity between two tuples (e_i, n_i) and (e_j, n_j) is defined as the product of kernels on the edge labels e_i, e_j and the node labels n_i, n_j. In the experiments, the following functions are considered:

$$k_e(e_i, e_j) = \mathbb{I}(e_i = e_j), \;\; k_n(n_i, n_j) = \mathbb{I}(n_i = n_j),$$

$$k_n(n_i, n_j) = \exp(\boldsymbol{w}_{n_i}^T \boldsymbol{w}_{n_j} - 1)((\boldsymbol{w}_{n_i}^T \boldsymbol{w}_{n_j} - \gamma)/(1-\gamma))_+.$$

Herein, $(.)_+$ denotes the positive part; $\gamma \in (-1, 1)$ is a sparsity parameter.

One can consider the nonstationary parameters to be binary valued, i.e., $\sigma \in \{0, 1\}$ (I suggest to keep binary values only, as a measure for robustness to over-fitting, as well as for keeping the optimization simple).

In the expression above, non-stationarity parameters are local to edge labels (referred as semantic labels in this chapter), which come from a language representation (predefined, and small in vocabulary),

such as Abstract Meaning Representations (Banarescu et al., 2012), Stanford Dependencies (Chen and Manning, 2014), etc. Defining the σ parameters local w.r.t. the edge labels (semantic labels), corresponds to attention on the semantics in the context of convolution kernel computation. For instance, as shown in Table 3.1, I find that it is more optimal to skip tuples, containing either of the semantic labels *arg2, arg0, op2, example* when computing convolution kernel similarity, as $\sigma = 0$ for all these labels in the learned model for the task .[3] Thus, the proposed nonstationary approach allows decisions on what to attend in a linguistic representation of a natural language, and what not to attend. This is very intuitive, as it is close to how a human mind processes natural language.

Note that, even in the traditional convolution kernels, matching substructures are found while skipping over non-matching elements. Our approach of skipping tuples, as proposed above, allows skipping tuples even if those tuples are matching when computing similarity between two structures. This aspect of skipping tuples is more explicit in ignoring sub-structures irrelevant for a given task, unlike the standard skipping over non-matching tuples; the latter is complementary to ours.

3.4 Stochastic Learning of Kernel Parameters

In this section I discuss the learning of kernel parameters for classification with k-Nearest Neighbor models.

3.4.1 k-NN based Loss Function

In the above, I introduced a non-stationary extension of convolution kernel similarity by introducing a task-dependent parameterization of the kernel function. In this section I describe a novel approach for efficiently learning those parameters in the context of a binary classification problem, especially suitable for k-NN classifiers. This setup is highly relevant for various structured inference problems in natural language

[3] arg0, arg2 are not ignored if duplicate data points are kept.

Algorithm 2 Stochastic-subsampling based minimization of the loss, defined on k-NN, for learning a convolution kernel.

Require: Training data set $\boldsymbol{S} = \{S_1, \cdots, S_N\}$; the number of samples to select randomly, β; the number of trials for optimizing a parameter value, α; the convolution kernel function $K(.,.;\boldsymbol{\sigma})$; the semantic-labels $\boldsymbol{L} = \{l_1, \cdots, l_p\}$; k for the global k-NN graph.

% indices of data points, $\boldsymbol{c}_i$, containing a semantic label, l_i

$\boldsymbol{c}_1, \cdots, \boldsymbol{c}_p \leftarrow getDataIndices(\boldsymbol{S}, \boldsymbol{L})$

$\boldsymbol{\sigma} = \{\sigma_1, \cdots, \sigma_p = 1\}$ % parameters, sorted per the frequency of the labels in descending order

% global k-NN graph for intelligent sampling

$\boldsymbol{G} \leftarrow computeNNGraph(\boldsymbol{S}, k, K, \boldsymbol{\sigma})$

% optimize each t_{th} parameter, iteratively

for $t = 1$ to p **do**

  for $j = 1$ to α **do**

    $\boldsymbol{r} \leftarrow randomSubset(\boldsymbol{c}_t, \beta)$ % β random samples

    $\boldsymbol{n} \leftarrow getNN_{\rightarrow}(\boldsymbol{G}, \boldsymbol{r}, k)$ % neighbors of $\boldsymbol{r}$

    $\bar{\boldsymbol{n}} \leftarrow getNN_{\leftarrow}(\boldsymbol{G}, \boldsymbol{r}, k)$ % $\boldsymbol{r}$ as neighbors

    $\boldsymbol{nn} \leftarrow getNN_{\rightarrow}(\boldsymbol{G}, \boldsymbol{n} \cup \bar{\boldsymbol{n}}, 1)$ % neighbors of $\boldsymbol{n} \cup \bar{\boldsymbol{n}}$

    $\boldsymbol{a} \leftarrow \boldsymbol{r} \cup \boldsymbol{n} \cup \bar{\boldsymbol{n}} \cup \boldsymbol{nn}$ % $\boldsymbol{a}$ for loss estimates

    for $v = \{0, 1\}$ **do**

      $\sigma_t = v$ % value v for the t_{th} parameter

      % noisy loss estimate for $K(.,.;\boldsymbol{\sigma})$

      $\boldsymbol{G}^v_{tj} \leftarrow computeNNGraph(\boldsymbol{S}_{\boldsymbol{a}}, 1, K, \boldsymbol{\sigma})$ % 1-NN graph on the subset $\boldsymbol{a}$ for loss estimate

      $\mathcal{L}^v_{tj} \leftarrow computeLoss(\boldsymbol{G}^v_{tj})$ % loss on $\boldsymbol{G}^v_{tj}$

  $\sigma_t \leftarrow \text{argmin}_v \sum_{j=1}^{\alpha} \mathcal{L}^v_{tj}$

return $\boldsymbol{\sigma} = \{\sigma_1, \cdots, \sigma_p\}$ % learned $\boldsymbol{\sigma}$, in $K(.,.;\boldsymbol{\sigma})$

modeling, where a hypothesized structured inference from a sentence is classified as correct/incorrect. Note that the loss-function proposed below can be easily extended to multi-class scenarios.

Loss Function 1. *Consider N number of training examples—say, sequences of tuples $S_1, \cdots, S_N$ with corresponding binary labels $y_1, \cdots, y_N \in \{0, 1\}$. A convolution kernel $K(., .; \boldsymbol{\sigma})$ with parameters $\boldsymbol{\sigma}$ can be learned by minimizing the loss function,*

$$\mathcal{L} = \sum_{i=1}^{N} K(S_i, S_{1nn(i)}; \boldsymbol{\sigma}) \mathbb{I}(y_i \neq y_{1nn(i)}) - \sum_{i=1}^{N} K(S_i, S_{1nn(i)}; \boldsymbol{\sigma}) \mathbb{I}(y_i = y_{1nn(i)} = 1).$$

For a sequence S_i, we correspondingly have the first nearest neighbor of S_i within the training data set itself, i.e., $S_{1nn(i)}$, with a binary label $y_{1nn(i)}$; the function $1nn(.)$ comes from kernel-based 1-NN graph construction, optionally using kernel-LSH.

If the binary label of S_i, y_i, does not match with the binary label of $S_{1nn(i)}$, $y_{1nn(i)}$, then there is loss, equal to the convolution kernel similarity $K(S_i, S_{1nn(i)}; \boldsymbol{\sigma})$. In the loss function, I also encourage higher kernel similarity between a sequence and its first nearest neighbor if both are of positive (binary) labels by using the negative loss term, i.e., reward. The latter should make the kernel function robust w.r.t. noise in data, acting more like a regularization term.

The loss function is extensible for $k > 1$, though I argue that it should be sufficient to keep $k = 1$ during the learning,[4] as the loss function is computed using a small subset of training dataset with our stochastic approximation-based algorithm (discussed next).[5]

Note that the loss function is similar to the ones used for distance metric learning problems (Weinberger and Saul, 2009).

3.4.2 Stochastic Subsampling Algorithm

The loss function defined above involves calculating $O(N^2)$ pairwise similarities between N data points, which is computationally prohibitive for large N. Instead, we would like to approximate the loss function

[4] $k = 1$ in the loss function makes LSH efficient, too.

[5] A bagged 1-NN classifier is optimal, as a k-NN model with an optimal value of k (Biau et al., 2010).

Table 3.2: F1 score (precision, recall), on the positive class. The non-stationary path kernel (NPK), trained on a subset of the "All" dataset with our hashing function. For inference, $k = 8$ ($k = 4$ for PubMed45 since it is a small data set) in k-NN based classifiers, with hashing used in both approaches "Path Kernel" (PK) and "Nonstationary Path Kernel" (NPK). I perform five inference trials, for the cases of 80%, 50%, 30% data sub-selection.

(a) RkNN Hashing Technique

Data sets	PK	**NPK**
BioNLP (100%)	42.4 (43.1, 41.8)	**47.2** (**53.7**, 42.2)
BioNLP (80%)	41.0 ± 0.5 (40.7, 41.4)	$\mathbf{43.9 \pm 1.3}$ (**47.7**, 40.7)
PubMed45 (100%)	38.7 (37.8, 39.6)	**44.7** (**45.8**, **43.7**)
PubMed45 (80%)	36.3 ± 0.9 (35.7, 36.8)	$\mathbf{41.8 \pm 1.1}$ (**42.4**, **41.3**)
All (100%)	55.0 (52.5, 57.8)	**58.4** (**55.0**, **62.2**)
All (50%)	50.9 ± 0.6 (49.5, 52.4)	$\mathbf{54.1} \pm 0.7$ (50.0, **59.0**)
All (30%)	48.6 ± 1.2 (46.7, 50.6)	$\mathbf{51.5} \pm 0.9$ (48.4, **55.1**)

(b) Hashing technique of Kulis & Grauman.

Data sets	PK	**NPK**
BioNLP (100%)	44.8 (45.7, 44.0)	**46.8** (**52.7**, 42.1)
BioNLP (80%)	42.0 ± 1.0 (41.7, 42.2)	$\mathbf{44.6 \pm 1.5}$ (**49.3**, 40.9)
PubMed45 (100%)	38.6 (39.1, 38.1)	**43.5** (**44.9**, **42.2**)
PubMed45 (80%)	35.5 ± 0.8 (35.6, 35.4)	$\mathbf{41.7 \pm 1.1}$ (**42.0**, **41.4**)
All (100%)	54.2 (52.6, 55.9)	**57.9** (**54.6**, **61.7**)
All (50%)	52.2 ± 0.4 (50.6, 53.9)	$\mathbf{54.3} \pm 0.6$ (50.9, **58.2**)
All (30%)	50.3 ± 0.6 (48.7, 52.1)	51.6 ± 0.8 (48.2, 55.6)

by using a small subset of the training data. The naive strategy of random sampling would correspond to taking a random subset of nodes in the k-NN graph, and then considering the neighborhood edges only between those nodes. Unfortunately, uniform random sampling of nodes (data points) would be a crude approximation for the loss function as it would significantly alter (or even completely destroy) local neighborhood structure for each selected data point. Instead, here I propose to include the neighboring nodes of a randomly selected node in the subset of data used for computing a loss estimate. Thus, the problem is how to select the local neighborhoods, which depend upon the kernel function itself that we are optimizing. This is accomplished as described below.

First, before optimizing the parameters, we compute the nearest neighborhood graph for all the training data points with a locality-sensitive hashing technique and by using the initialized convolution kernel function itself; we refer to this as a global k-NN graph, used for building the local neighborhoods approximately.

Now we sample a small subset of data from the training set in a purely random fashion, as the inducing points. Then we look for the training data which are neighbors of these random selections, or have these random selections themselves as their neighbors in the k-NN graph; optionally, we also try to include the first nearest neighbors of the neighbors. Putting all these data points together into a subset, including the original subset of random selections, we obtain our final subset of the training dataset—what I refer to as randomly sampled local neighborhoods. We compute a 1-NN graph on this subset of the dataset itself to compute a noisy estimate of the loss function. As we see, the neighbors (in both directions) of an inducing point (randomly selected) establish the local neighborhood around it. Even though, in the loss function, we are interested only in a small neighborhood around a point, i.e., defined from the first nearest neighbor, we keep k reasonably high in the above global k-NN graph and correspondingly obtain k neighbors of the inducing point from both directions. This is because the higher value of k ($k = 4$ in the experiments, though robust w.r.t. small changes) should make the approach robust to changes in the neighborhood during the learning (as parameter values change), and also increase the chances for the neighbors of an inducing point to be neighbors of each other (for less noise in the loss function estimation).

Table 3.3: F1 score on the positive class. Same settings as for Table 3.2(a).

(a) NPK with lower sampling rates (β)

Data sets	$\beta = 550$	$\beta = 100$	$\beta = 50$	$\beta = 25$
BioNLP	**47.2** (53.7, 42.2)	**47.3** (52.5, 43.0)	**47.1** (52.4, 42.8)	**48.4** (50.9, 46.0)
PubMed45	**44.7** (45.8, 43.7)	**42.7** (45.2, 40.4)	**43.0** (42.2, 43.8)	**41.9** (41.8, 42.0)

(b) TK ($k = 1$)

Data sets	TK	**NTK**
BioNLP	38.5 (37.6, 39.4)	**41.8** (42.0, 41.5)
PubMed45	33.4 (33.0, 33.8)	**37.8** (36.6, 39.0)

(c) With word2vec

Data sets	PK	**NPK**
BioNLP	40.8 (41.5, 40.2)	**43.7** (44.9, 42.5)
PubMed45	35.0 (34.0, 36.1)	**36.5** (37.7, 35.3)

(d) Nonunique

Data sets	PK	**NPK**
BioNLP	60.2 (62.2, 58.3)	**62.4** (**65.2**, 59.8)
PubMed45	36.6 (44.1, 31.3)	**40.5** (45.9, **36.2**)

(e) Individual Training

Data sets	PK	NPK
BioNLP	41.8 (43.0, 40.7)	**44.5** (44.0, **45.0**)
PubMed45	38.1 (39.0, 37.3)	**42.8** (**42.9**, **42.7**)

3.4.2.1 Pseudocode for the Algorithm

The pseudocode for optimizing the semantic-label- based binary parameters is presented in Algorithm 2. Since a parameter σ_t is defined locally w.r.t. a semantic label l_t, we obtain the indices of data points ($\boldsymbol{c}_t$), in the training set $\boldsymbol{S}$, that contain the semantic label; these data should be immediately relevant for the optimization of the parameter as the superset of the inducing points (see line 1 in Algorithm 2). All the binary parameters, $\boldsymbol{\sigma} = \{\sigma_1, \cdots, \sigma_p\}$, are initialized to value one, which corresponds to equal attention to all types of semantics in the beginning, i.e., the baseline convolution kernel (line 2). During the optimization, as we would learn zero values for some of the parameters, the kernel function would drift away from the baseline kernel and would become more non-stationary, accounting for only a subset of the semantic labels in its computations.

Next, after the initialization of the binary parameters, we compute the global k-NN graph on the training data set $\boldsymbol{S}$, using the kernel $K(.,.;\boldsymbol{\sigma})$ with the parameters $\boldsymbol{\sigma}$ (line 3).

For learning these binary parameters, $\boldsymbol{\sigma}$, I find that a simple iterative procedure does a good job (line 4), though the algorithm should be easily extensible for more sophisticated techniques such as MCMC sampling. For optimizing a t_{th} parameter, σ_t, we perform α number of trials (line 5).

In each trial, a subset of the training set, with indices $\boldsymbol{a}$, is obtained using our sampling technique (lines 6-10), starting with β number of randomly selected inducing points. The next step within the trial is to compute the loss function (line 14) for each possible value of σ_t (i.e., $\{0, 1\}$) by computing a 1-NN graph on the subset of samples $\boldsymbol{S_a}$ (line 13), and then obtaining the value of σ_t with a minimum sum of the noisy loss estimates (line 15).

In the algorithm, the global k-NN graph (line 3), and the 1-NN graphs (line 13) can be computed with the kernel-LSH, as done in the experiments; same applies to the inference phase after learning the parameters.

Next, I discuss the empirical results.

Table 3.4: F1 score on the positive class. Same settings as for Table 3.2(a).

(a) $\beta = 25$, multiple runs of the algorithm for NPK

Data sets	PK	$iter = 1$	$iter = 3$	$iter = 6$	$iter = 9$
BioNLP	42.4 (43.1, 41.8)	46.4 (51.7, 42.0)	44.0 (49.8, 39.4)	47.2 (53.7, 42.2)	47.4 (51.0, 44.2)
PubMed45	38.7 (37.8, 39.6)	41.9 (42.7, 41.1)	41.9 (44.7, 39.4)	44.7 (45.8, 43.7)	41.1 (41.6, 40.7)

(b) Pr- Pure Random.

Data sets	PK	$\boldsymbol{\beta = 100}$	$\beta = 1000$ (PR)	$\boldsymbol{\beta = 550}$	$\beta = 5000$ (PR)
BioNLP	42.4 (43.1, 41.8)	**47.3** (52.5, 43.0)	42.4 (43.1, 41.8)	**47.2** (53.7, 42.2)	43.7 (48.3, 39.9)
PubMed45	38.7 (37.8, 39.6)	**42.7** (45.2, 40.4)	40.1 (40.1, 40.0)	**44.7** (45.8, 43.7)	41.8 (44.7, 39.3)

3.5 Empirical Evaluation

For the evaluation of the proposed nonstationary convolution kernels, I consider the relation extraction task as described in Section 5.1.

Relation Extraction from Semantic Graphs

In reference to Figure 3.1, each sentence is parsed into an Abstract Meaning Representation (AMR) graph (Pust et al., 2015a), and from the list of entity nodes and interaction type nodes in the graph, a set of candidate biomolecular interactions are generated—each involving up to three entities with different roles, and an interaction type. Each candidate structure (a biomolecular interaction in this case), as a hypothesized inference, is classified as positive/negative, either using the corresponding subgraph as features (as in Figure 3.1(b), 3.1(c)) with tree kernels, or a subgraph can be post-processed into a path (see Table 3.1) if using path kernels.

Data Sets

I use three datasets, including two public ones, PubMed45-ERN (Garg et al., 2016) and BioNLP (2009,

11,13) (Kim et al., 2009, 2011; Nédellec et al., 2013), and an unpublished one; I refer to PubMed45-ERN simply as PubMed45 in this chapter though PubMed45 refers to a subset of PubMed45-ERN in the previous chapter. Putting all the datasets together, referred as "All" dataset, we have approximately 100k unique data points (20k and 30k from the PubMed45 and BioNLP datasets respectively), with uniqueness defined on the path tuples to avoid artificial redundancies in the evaluation, *although this decreases the overall accuracy numbers as canonical examples are counted only once*. I also show results for non-unique paths, in Section 3.5.1.2, which can be compared to other published results on the datasets.

Settings for the Learning of the Proposed Model Parameters

In the evaluation setup, I randomly divide the full dataset into two halves, and use the first half (with approximately N=50k samples) to learn the proposed nonstationary convolution kernel with the proposed Algorithm 2; I set $\beta = 550$, $\alpha = 10$, and learn σ for the top 50% frequent semantic (edge) labels, i.e., 61 binary parameters. Whenever computing k-NN graphs, I use kernel-LSH (H=14), using the proposed hashing function (RkNN) in Section 3.2. [6]

For building a k-NN graph with hashing in the learning algorithm, I keep M (defined in Section 3.2.2) approximately equal to the square root of the number of nodes in k-NN; for the inference purposes, I keep M=100 fixed across all the datasets.

In the convolution kernels, $\lambda = 0.8$, which is set from preliminary tunings, as well as the use of the learning algorithm itself. For $\beta = 550$, i.e., the primary configuration on the sample size when learning the model on all the data sets together, it takes approximately 12 hours on a 16-core machine to optimize the parameters (a kernel matrix computing is parallelized); I also try $\beta = 25$, taking an hour for learning the model.

[6] H=14 is simply from the observation that it should be a little less than the $\log_2(.)$ of the data set size so that a constant number of data points land in a hash bucket.

Table 3.5: Loss function convergence.

Models	$\beta = 200$ (25)	$\beta = 500$ (25)	$\beta = 1000$ (25)	All data points
PK	7.7e-3±4.9e-3	8.0e-3±3.2e-3	6.2e-3±1.5e-3	4.5e-3
NPK	3.9e-3±6.5e-3	2.3e-3±4.4e-3	2.0e-3±3.2e-3	2e-3

3.5.1 Evaluation Results

The convolution path kernel with the 61 binary parameters, referred to as Nonstationary Path Kernel (**NPK**) and optimized on the training subset from the All dataset, is evaluated w.r.t. the standard (conventional) path kernel (**PK**), across the individual public datasets PubMed45 and BioNLP, as well as the All dataset. For the evaluation of the two kernels, using k-NN classifier with hashing, I split a given data set randomly into two equal parts, with the first half for building a k-NN classifier that is tested on the second half. I have multiple inference evaluations with varying percentages of data sub-sampled from each of the two parts (5 trials).

The results are presented in Table 3.2(a), in terms of classification accuracy (F1-score, precision, recall) for the positive class. We see significant improvement in accuracy numbers over the baseline standard path kernel, with the improvements even more significant for the two public datasets. We obtained this improvement just with the addition of the 61 binary parameters to the baseline kernel, corresponding to NPK. It is even more interesting to note that only 20 of the 61 parameters have zero values learned—thereby solely contributing to the improvements. Since the model is trained on all the datasets put together, it is further interesting to note that the learned model performs well for the individual public datasets, despite distribution changes across data sets in terms of positive ratios as well as language itself.

Further, I verify if the same model parameters, learned with my RkNN hashing technique (see Section 3.2), generalize to the hashing technique of (Kulis and Grauman, 2012). In Table 3.2(b), we show the classification accuracies with their hashing technique. While there is not much difference between accuracy numbers across the two choices for hashing, we do see that NPK outperforms PK, even with this hashing technique.

3.5.1.1 Varying Parameter & Kernel Settings

Lower Sampling Rates

Keeping the same setup as that used for producing the results in Table 3.2(a), but lowering the parameter β, in Algorithm 2,[7] I obtain results shown in Table 3.3(a). The accuracy numbers decrease marginally with lower values of β, down to $\beta = 25$, that correspond to smaller subsets of data during the learning. This leads to more noisy estimates of the loss function, but lesser computational cost. It should be noted that all of these nonstationary models learned with high stochasticity, outperform the standard path kernel (in Table 3.2(a)) significantly.

Tree Kernels

Using the same setup for learning (a single model trained on all the data sets) as well as inference, I evaluate the proposed approach for tree kernels, with the results presented in Table 3.3(b). The non-stationary tree kernels (**NTK**) outperform the standard tree kernels (**TK**), though the overall numbers are not as good as with the path kernels.[8] Also, it is interesting that $k = 1$ is optimal in the inference for tree kernels, possibly because the dimension of the features space implied from tree kernels is very high.

Use of Word Vectors in Path Kernels

The proposed approach is generic enough for applicability to convolution kernels using word vectors. I compare NPK approach w.r.t. PK,[9] both using word vectors, and find that the nonstationary model helps in this scenario as well; see Table 3.3(c).[10]

[7] For $\beta \leq 50$, $\beta = 100$, we use H=8, H=10 respectively.

[8] One possible reason for the overall low numbers with tree kernels is this: In Table 3.1(b), if we exchange the colors of the entity nodes, and so the respective roles, the hypothesized interaction changes, becomes invalid from valid, while the tree remains very similar; the path would change significantly, as obtained from tree-traversal as per the node colors.

[9] Sparsity parameters, γ, is tuned to value 0.6.

[10] The overall accuracy numbers are less when using word vectors, partially because there is no domain mismatch in terms of vocabulary when splitting the set of unique path tuples randomly into training-test sets (in such scenarios, there is not much difference in accuracy, whether we use word vectors or not, be it our path kernels or the baseline path kernels).

Table 3.6: F1 score, on the positive class w.r.t. other classifiers. Same settings as for Table 3.2(a).

Data sets	PK-k-NN	**NPK-kNN-H**	PK-SVM	LSTM	Conv. LSTM
BioNLP	**47.3** (54.6, **41.7**)	**47.2** (53.7, **42.2**)	38.3 (61.5, 27.8)	46.0 (65.7, 35.4)	44.1 (63.8, 33.7)
PubMed45	**45.3** (51.5, **40.4**)	**44.7** (45.8, **43.7**)	40.2 (53.0, 32.5)	34.3 (47.8, 26.7)	40.9 (47.0, 36.2)

3.5.1.2 Non-Unique Path Tuples

In Table 3.3(d), I present experimental results where duplicate path tuples are kept in the datasets ($\beta = 130$, i.e., 0.2% of training data). For the BioNLP data set, I use the BioNLP-2013-development subset for testing, and the rest for building the k-NN classifiers. For the PubMed dataset, I use six papers for testing (approx. 10% data points), and the rest for building the classifier. For both datasets, NPK give better accuracy than PK. These numbers, in Table 3.3(d), can also be compared to the previously published evaluation numbers on the datasets.

3.5.1.3 Analysis of the Learning Algorithm

Analysis of the Loss Function

In Table 3.5 I present the loss (rather than the F1 score) of the standard path kernel model, as well as the nonstationary path kernel model ($\beta = 100$). For computing the loss of a given (already trained) model, I use the same sampling technique as proposed for learning, with $\beta = 200, 500, 1000$ corresponding to the different columns in the table, and also compute the loss on all the training data (with no sampling). The numbers in the last column "All data" validate that the loss function value is reduced with our nonstationary model parameters compared to the standard path kernel, from $4.5e-3$ to $2e-3$. The loss function variation w.r.t. the sampling rate β establishes on empirical convergence of the approximate loss computed with the sampling technique, w.r.t. to the loss computed on all the data.

Multiple Runs of the Algorithm for Learning

I also perform the experiment of running the algorithm multiple times wherein, after each iteration, the

global k-NN graph is recomputed as per the change in the kernel function ($\beta = 25$); the models learned across different iterations are evaluated as presented in Table 3.4(a). While all the models, learned across different iterations, have a much better accuracy compared to the standard path kernel (baseline), the accuracy doesn't seem to always increase with more iterations, partly due to the high noise from the low sampling rate. This should mean that the learning is not highly sensitive to the fine grained structure of the global k-NN graph, while the value of β plays more significant role, as we get much better accuracy for $\beta = 550$ above, even with a single iteration, than the model obtained from 10 iterations of learning with $\beta = 25$.

Experiments with Pure Random Sampling

While keeping the same nonstationary model, a possible baseline approach for the learning could be sampling the data in a pure random fashion rather than using the proposed sampling technique; see Table 3.4(b). For this baseline, we sample only the β inducing points in Algorithm 2, and not the neighbors (suffix "(PR)"). Clearly, the strategy of just pure random sampling doesn't work well.

3.5.1.4 Individual Models Learned on Datasets

For a further evaluation, I also train the model individually for each of the two public datasets, PubMed45 and BioNLP. The experimental settings are the same, using 50% random selections for training, and the rest for inference. The value for parameter β is 3% of the training subset size (i.e., approximately, $\beta = 300$, $\beta = 450$ for the PubMed45 and BioNLP training subsets respectively); I set, H=10, as the training data set size is smaller than above, while using the proposed hashing technique (RkNN) in both learning and inference tasks. The results are presented in Table 3.3(e). Although NPK outperforms the standard path kernel more significantly for the PubMed45 dataset, the improvements are less than what we achieved when training the model on all the datasets together.

3.5.1.5 Comparison with Other Classifiers

Finally, I compare the proposed nonstationary path kernel-based k-NN classifier using the kernel-LSH (the same classifier as "NPK" in Table 3.2(a)), denoted as "NPK-kNN-H" here, w.r.t. other kernel-based classifiers, as well as neural network baselines; see Table 3.6. Other kernel-based classifiers, all using the standard path kernels, include k-NN without using hashing ("PK-k-NN", k=4), and SVM ("PK-SVM", C=1). All these classifiers are expensive to compute as they require full Gram matrices, with $O(N^2)$ number of kernel computations. The hashing-based version of"PK-k-NN" corresponds to "PK" in Table 3.2(a). Clearly, there is a significant decrease in accuracy for standard path kernels when using hashing. In Table 3.6 [11] I demonstrate that the nonstationary path kernel-based k-NN model ("NPK-kNN-H"), despite the hashing-based approximations, performs as good as the PK-k-NN, and both methods outperform all the other classifiers, including the neural network models LSTM and convolution-LSTM (label frequency-based 32-bit embeddings are used in the LSTM models, with "ReLU" units). The advantage of "NPK-kNN-H" over the "PK-k-NN" is the reduction in kernel computations ($O(N^{1.5})$ from $O(N^2)$), due to hashing.

3.6 Related Work

Nonstationary kernels have been explored for modeling spatiotemporal environmental dynamics or time series relevant to health care, finance, etc., in which the challenge in learning is due to the high number of local parameters, scaling linearly with dataset size (Assael et al., 2014; Le et al., 2005; Paciorek and Schervish, 2003; Snelson et al., 2003; Rasmussen and Ghahramani, 2002; Higdon, 1998; Sampson and Guttorp, 1992). The number of nonstationary parameters in our NLP model is a small constant, though the cost of computing kernels is much higher compared to the kernels relevant for those domains.

In neural language models (Hochreiter and Schmidhuber, 1997; Mikolov et al., 2010; Sundermeyer et al., 2012), it is challenging to remember a long sequence (Mikolov et al., 2014), as well as forgetting

[11] For k-NN, accuracy variation w.r.t. k is minimal, so I do a preliminary tuning using a much smaller subset, whereas for other methods, I either perform validation based tuning, or choose the best accuracy numbers from the parameters space.

some parts of it (Gers et al., 1999; Zhang et al., 2015; Yu et al., 2017), with some success attained in the recent past on these problems. On the other hand, convolution (path or subsequence) kernels, implicitly, try to remember entire sequences (leads to less efficient language modeling if text is noisy), and this work allows skipping some of the tuples in sequences, i.e. forgetting.

The idea of skipping tuples in convolution kernels had been previously explored relying on sub-structure mining algorithms (Suzuki et al., 2004; Suzuki and Isozaki, 2006; Severyn and Moschitti, 2013). Recently, it is proposed to learn weights of sub-structures for regression problems within Gaussian process modeling framework (Beck et al., 2015). In this chapter, I show that my principled approach of non-stationary extension of a convolution kernel leads to an additional parameterization that allows learning the skipping of tuples as a special case. Further, for k-NN classifiers, I propose an efficient stochastic sampling based algorithm, along with an appropriate loss function, for scalable learning of the parameters with hundreds of thousands of data in a training set. In contrast, the previous works do not scale beyond a couple of thousand data in a train set. Moreover, as reported in (Beck et al., 2015), the introduced parameterization can lead to overfitting in their work whereas our stochastic sampling based learning ensures robustness to such issues. Also note that, my approach of skipping tuples through non-stationarity is more explicit in ignoring sub-structures irrelevant for a given task, and complementary to the standard skipping over non-matching tuples that is a common aspect in most of the existing (sparse) convolutions kernels (Zelenko et al., 2003; Mooney and Bunescu, 2005; Tkachenko and Lauw, 2015).

It is also interesting to note that most of the previous works, which explored convolution kernels for various NLP tasks, used Support Vector Machine (SVM) classifiers. The use of a kernel-SVM would have been appropriate if a convolution kernel projects natural language structures into a higher dimensional (kernel implied) features space wherein the structures from different classes were linearly separable. Unfortunately, unlike kernels operating on non-structured inputs, the existing convolution kernels (meant

for structured inputs) are not guarantied to have such properties of higher dimensional mappings. This motivates us to explore k-NN like non-linear classifiers in this work as those have non-linear class boundaries even in the kernel implied feature space, in contrast to SVMs. [12]

3.7 Chapter Summary

In this chapter, I proposed a novel nonstationary extension of convolution kernels by introducing a data-driven parameterization of the kernel similarity function. The extended kernels have better flexibility and expressibility of language representations, compared to conventional convolution kernels used in natural language tasks. I validated the proposed approach in a set of experiments for the relation extraction task, and observed significant improvement of accuracy numbers over the state-of-the-art methods across several data sets. The learned models are highly interpretable, as the zero values of the parameters correspond to a list of semantic labels and corresponding substructures that are ignored during kernel computation over the semantic graph. I also proposed a tractable learning method based on a stochastic-sampling algorithm, and demonstrated that keeping the sampling rate low has only a moderate adverse impact on accuracy, while yielding significant gains in computational efficiency.

[12] The use of kNN models has been discouraged in high dimensional euclidean spaces, due to the difficulty of learning appropriate distance/similarity functions; however, such difficulties should not arise for structured inputs.

Chapter 4

Learning Kernelized Hashcode Representations

In the previous chapter, I demonstrated that kernelized locality-sensitive hashing (KLSH) (Kulis and Grauman, 2009; Joly and Buisson, 2011) allows to reduce the number of kernel computations to $O(N)$ by providing efficient approximation for constructing kNN graphs. However, KLSH only works with classifiers that operate on kNN graphs. Thus, *the question is whether scalable kernel locality-sensitive hashing approaches can be generalized to a wider range of classifiers*.

One of the core contributions of this thesis is a *principled approach for building explicit representations for structured data*, as opposed to implicit ones employed in prior kNN-graph-based approaches, *by using random subspaces of KLSH codes.* The intuition behind this proposed approach is as follows. If we keep the total number of bits in the KLSH codes of NLP structures relatively large (e.g., 1000 bits), and take many random subsets of bits (e.g., 30 bits each), we can build a large variety of generalized representations corresponding to the subsets, and preserve detailed information present in NLP structures by distributing this information across those representations.[1] The main advantage of the *proposed representation* is that it *can be used with arbitrary classification methods*, besides kNN such as, for example, random forests (RF) (Ho, 1995; Breiman, 2001). Figure 4.2 provides high-level overview of the proposed approach.

[1] Compute cost of KLSH codes is linear in the number of bits (H), with the number of kernel computations fixed w.r.t. H.

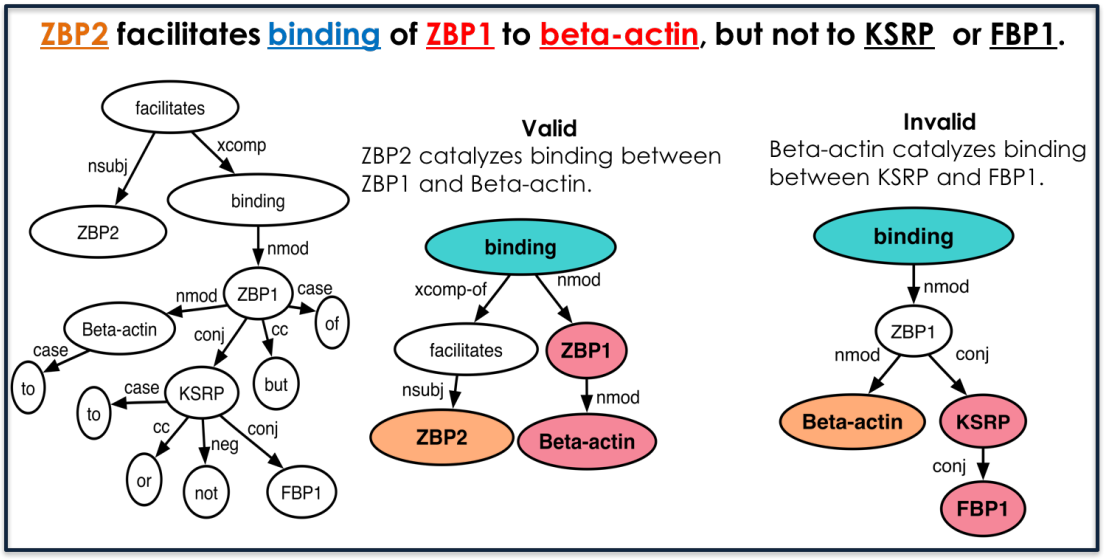


Figure 4.1: On the left, a parse tree of a sentence is shown. In the sentence, the tokens corresponding to bio-entities (proteins, chemicals, etc.) or interaction types are underlined. I highlight the result of extracting one relation from the sentence, using color-coding for its constituents: an interaction type (green) and bio-entities either participating in the interaction (red), or catalyzing it (orange). From two extraction candidates (valid/invalid), I obtain subgraphs from the parse tree, used as structural features for binary classification of the candidates.

Another major contribution of this thesis is a *theoretically justified and computationally efficient method for optimizing the KLSH representation* with respect to: (1) the kernel function parameters and (2) a reference set of examples w.r.t. which kernel similarities of data samples are computed for obtaining their KLSH codes. The proposed optimization method maximizes *an approximation of* mutual information between KLSH codes of NLP structures and their class labels.[2] In addition, the parameters resulting from the non-stationary extension that I introduced in the previous chapter, are also learned by maximizing the mutual information approximation.

I validate the proposed model on the relation extraction task using four publicly available datasets. Significant improvements are observed in F1 scores w.r.t. the state-of-the-art methods, including recurrent neural nets (RNN), convnets (CNN), and other methods, along with large reductions in the computational complexity as compared to the traditional kernel-based classifiers.

[2]See our code here: `github.com/sgarg87/HFR`.

In summary, the contributions of my thesis, discussed in this chapter (Garg et al., 2019), are as follows: (1) I propose an explicit representation learning for structured data based on kernel locality-sensitive hashing (KLSH), and generalize KLSH-based approaches in information extraction to work with arbitrary classifiers; (2) I derive an approximation of mutual information and use it for optimizing the proposed models, including the nonstationary kernel based models; (3) I provide an extensive empirical evaluation demonstrating significant advantages of the approach versus several state-of-art techniques.

4.1 Background

As indicated in Figure 4.1, the relation extraction task is mapped to a classification problem, where each candidate interaction as represented by a corresponding (sub)structure is classified as either valid or invalid.

Let $\boldsymbol{S} = \{S_i\}_{i=1}^N$ be a set of data points representing NLP structures (such as sequences, paths, graphs) with their corresponding class labels, $\boldsymbol{y} = \{y_i\}_{i=1}^N$. Our goal is to infer the class label of a given test data point S_*. Within the kernel-based methods, this is done via a convolution kernel similarity function $K(S_i, S_j; \boldsymbol{\theta})$ defined for any pair of structures S_i and S_j with kernel-parameter $\boldsymbol{\theta}$, augmented with an appropriate kernel-based classifier (Garg et al., 2016; Srivastava et al., 2013; Culotta and Sorensen, 2004; Zelenko et al., 2003; Haussler, 1999).

4.1.1 Kernel Locality-Sensitive Hashing (KLSH)

Previously, Kernel Locality-Sensitive Hashing (KLSH) was used for constructing approximate kernelized k-Nearest Neighbor (kNN) graphs (Joly and Buisson, 2011; Kulis and Grauman, 2009). The key idea of KLSH as an approximate technique for finding the nearest neighbors of a data point is that rather than computing its similarity w.r.t. all other data points in a given set, the kernel similarity function is computed only w.r.t. the data points in the bucket of its hashcode (KLSH code). This approximation works well in practice if the hashing approach is locality sensitive, i.e. data points that are very similar to each other are assigned hashcodes with minimal Hamming distance to each other.

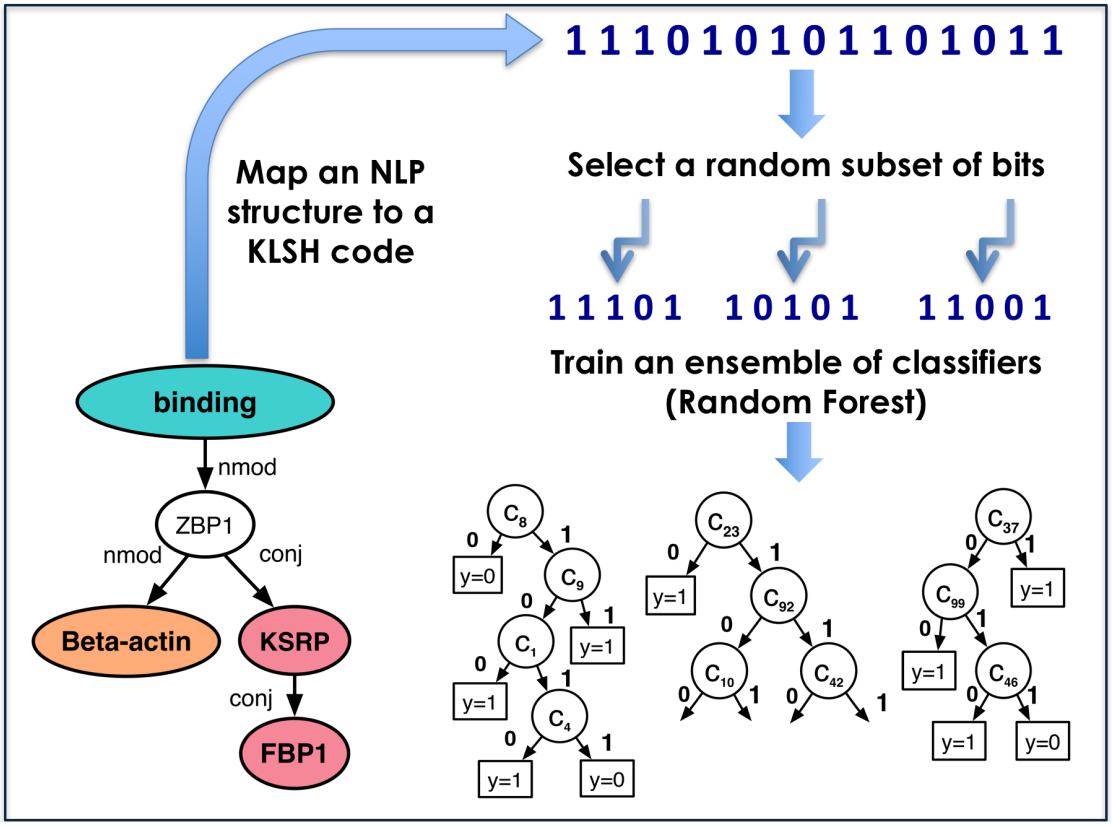


Figure 4.2: On the left, I show a subgraph from Figure 4.1 which has to be classified (assuming binary classification). The subgraph is mapped to a high-dimensional, kernel similarity-based locality-sensitive hashcode ($\boldsymbol{c}$), and use it as a feature vector for an ensemble classifier. For instance, an efficient and intuitive approach is to train a Random Forest on binary kernel-hashcodes; in the figure, the nodes in a decision tree makes decisions simply based on hashcode bit values, where each bit represents presence or absence of some structural pattern in the subgraph.

Herein, I brief on the generic procedure for mapping an arbitrary data point S_i to a binary kernel-hashcode $\boldsymbol{c}_i \in \{0,1\}^H$, using a KLSH technique that relies upon the convolution kernel function $K(.,.;\boldsymbol{\theta})$.

Let us consider a set of data points $\boldsymbol{S}$ that might include both labeled and unlabeled examples. As a first step in constructing the KLSH codes, we select a random subset $\boldsymbol{S}^R \subset \boldsymbol{S}$ of size $|\boldsymbol{S}^R| = M$, which I call a *reference set*; this corresponds to the grey dots in the left-most panel of Figure 4.3. Typically, the size of the reference set is significantly smaller than the size of the whole dataset, $M \ll N$.

Next, let $\boldsymbol{k}_i$ be a real-valued vector of size M, whose j–th component is the kernel similarity between the data point S_i and the j–th element in the reference set, $k_{i,j} = K(S_i, S_j^R; \boldsymbol{\theta})$. Further, let $h_l(\boldsymbol{k}_i)$, $l = 1, \cdots, H$, be a set of H binary valued hash functions that take $\boldsymbol{k}_i$ as an input and map it to binary bits and let $\boldsymbol{h}(\boldsymbol{k}_i) = \{h_l(\boldsymbol{k_i})\}_{l=1}^H$. The kernel hashcode representation is then given as $\boldsymbol{c}_i = \boldsymbol{h}(\boldsymbol{k}_i)$.

I now describe a specific choice of hash functions $h_l(.)$ based on nearest neighbors, called as Random k Nearest Neighbors (RkNN). For a given l, let $\boldsymbol{S}_l^1 \subset \boldsymbol{S}^R$ and $\boldsymbol{S}_l^2 \subset \boldsymbol{S}^R$ be two randomly selected, equal-sized and non-overlapping subsets of $\boldsymbol{S}^R$, $|\boldsymbol{S}_l^1| = |\boldsymbol{S}_l^2| = \alpha$, $\boldsymbol{S}_l^1 \cap \boldsymbol{S}_l^2 = \emptyset$. Those sets are indicated by red and blue dots in Figure 4.3. Furthermore, let $k_{i,l}^1 = \max_{S \in \boldsymbol{S}_l^1} K(S_i, S)$ be the similarity between S_i and its nearest neighbor in $\boldsymbol{S}_l^1$, with $k_{i,l}^2$ defined similarly (indicated by red and blue arrows in Figure 4.3). Then the corresponding hash function is:

$$h_l(\boldsymbol{k}_i) = \begin{cases} 1, & \text{if } k_{i,l}^1 < k_{i,l}^2 \\ 0, & \text{otherwise} \end{cases}. \tag{4.1}$$

Pictorial illustration of this hashing scheme is provided in Figure 4.3, where S_i's nearest neighbors in either subset are indicated by the red and blue arrows. [3] [4]

The same principle of random sub-sampling is applied in KLSH techniques previously proposed in (Kulis and Grauman, 2009; Joly and Buisson, 2011). In (Joly and Buisson, 2011), $h_l(.)$ is built by learning a (random) maximum margin boundary (RMM) that discriminates between the two subsets, $\boldsymbol{S}_l^1$ and $\boldsymbol{S}_l^2$.

[3] Small value of α, i.e. $1 \lll \alpha \ll M$, should ensure that hashcode bits have minimal redundancy w.r.t. each other.
[4] In RkNN, since $\alpha \ggg 1$, $k = 1$ should be optimal (Biau et al., 2010).

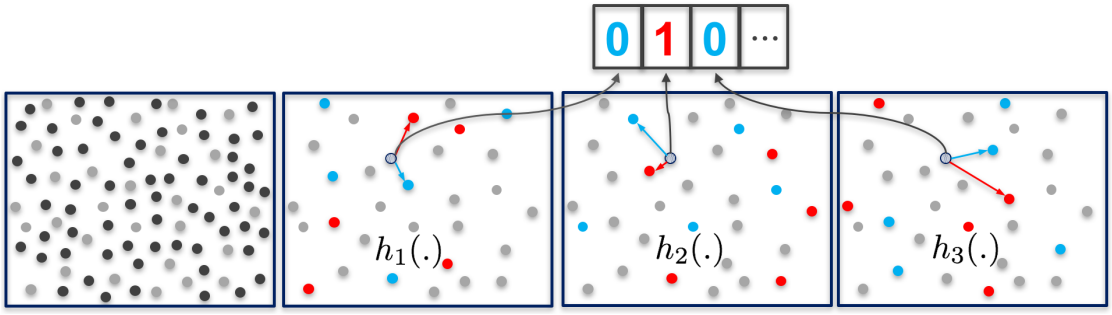


Figure 4.3: An illustration of the KLSH technique, Random K Nearest Neighbors (RkNN). First, one obtains a small subset (gray dots) from a super set of NLP structures as a *reference set* $\boldsymbol{S}^R$ that is used for constructing hash functions. For each hash function, two random subsets from the gray dots are obtained, denoted by red and blue. For a given structure, its kernel-based 1-nearest neighbor is found in both of the subsets as indicated by the arrows. Depending on which of the two 1-NNs–either the red 1-NN or the blue 1-NN—is the nearest to the sample, hash function $h_1(.)$ assigns value zero or one to the sample. The same procedure applies to $h_2(.)$. In this manner, hashcodes are generated with a large number of bits as explicit representations of NLP structures.

In (Kulis and Grauman, 2009), $h_l(.)$ is obtained from $\boldsymbol{S}_l^1 \cup \boldsymbol{S}_l^2$, which is a (approximately) random linear hyperplane in the kernel implied feature space; this is referred to as "Kulis" here.

In summary, I define klsh$(.; \boldsymbol{\theta}, \boldsymbol{S}^R)$ as the function, that is parameterized by $\boldsymbol{\theta}$ and $\boldsymbol{S}^R$, and maps an input data point S_i to its KLSH code $\boldsymbol{c}_i$, using the kernel function $K(., .; \boldsymbol{\theta})$ and the set of hash functions $\boldsymbol{h}(.)$ as subroutines.

$$\boldsymbol{c}_i = \text{klsh}(S_i; \boldsymbol{\theta}, \boldsymbol{S}^R); \quad \boldsymbol{c}_i = \boldsymbol{h}(\boldsymbol{k}_i); \quad k_{i,j} = K(S_i, S_j^R; \boldsymbol{\theta}) \tag{4.2}$$

Next, in Section 4.2, I propose an approach for learning KLSH codes as generalized representations of NLP structures for classification problems.

4.2 KLSH for Representation Learning

I propose a novel use of KLSH where the hashcodes (KLSH codes) can serve as generalized representations (feature vectors) of the data points. Since the KLSH property of being *locality sensitive* (Indyk and

Motwani, 1998) [5] ensures the data points in the neighborhood of (or within the same) hashcode bucket are similar, hashcodes should serve as a good representation of the data points.

In contrast to the use of KLSH for k-NN, after obtaining the hashcodes for data points, this approach ignores the step of computing kernel similarities between data points in the neighboring buckets. In kNN classifiers using KLSH, a small number of hashcode bits (H), corresponding to a small number of hashcode buckets, generate a coarse partition of the feature space—sufficient for approximate computation of a kNN graph. In my representation learning framework, however, hashcodes must extract enough information about class labels from the data points, so we propose to generate longer hashcodes, i.e. $H \gg 1$. It is worthwhile noting that for a *fixed number of kernel computations* M per structure S_i ($|\boldsymbol{S}^R| = M$), a large number of hashcode bits (H) can be generated through the randomization principle with computational cost linear in H.

Unlike regular kernel methods (SVM, kNN, etc.), I propose to use kernels to build an explicit feature space, via KLSH. Referring to Figure 4.3, when using RkNN technique to obtain $\boldsymbol{c}_i$ for S_i, l_{th} hashcode bit, $\boldsymbol{c}_{i,l}$, should correspond to finding a substructure in S_i, that should also be present in its 1-NN from either the set $\boldsymbol{S}^1_l$ or $\boldsymbol{S}^2_l$, depending on the bit value being 0 or 1. Thus, $\boldsymbol{c}_i$ represents finding important substructures in S_i in relation to $\boldsymbol{S}^R$. The same should apply for the other KLSH techniques.

4.2.1 Random Subspaces of Kernel Hashcodes

The next question is how to use the binary-valued representations for building a good classifier.

Intuitively, not all the bits may be matching across the hashcodes of NLP structures in training and test datasets; a single classifier learned on all the hashcode bits may overfit to a training dataset. This is especially relevant for bio-information extraction tasks where there is a high possibility of *mismatch between training and test conditions* (Airola et al., 2008; Garg et al., 2016); for e.g., in biomedical literature, the mismatch can be due to high diversity of research topics, limited data annotations, variations in writing

[5] See a formal definition of locality-sensitive hashing in (Indyk and Motwani, 1998, Definition 7 in Sec. 4.2).

styles including aspects like hedging, etc. So I adopt the approach of building an ensemble of classifiers, with each one built on a random subspace of hashcodes (Zhou, 2012; Ho, 1998).

For building each classifier in an ensemble of R classifiers, η bits are selected randomly from $H \gg 1$ hash bits; for inference on a test NLP structure S_*, I take mean statistics over the inferred probability vectors from each of the classifiers, as it is a standard practice in ensemble approaches. Another way of building an ensemble from subspaces of hashcodes is bagging (Breiman, 1996). If we use a decision tree as a classifier in ensemble, it corresponds to a random forest (Ho, 1995; Breiman, 2001).

It is highly efficient to train a random forest (RF) with a large number of decision trees ($R \gg 1$), even on long *binary* hashcodes ($H \gg 1$), leveraging upon the fact that decision trees can be very efficient to train and test on binary features.

4.3 Supervised Optimization of KLSH

In this section, I propose a framework for optimization of the KLSH codes as generalized representations for a supervised classification task. As described in Section 4.1.1, the mapping of a data point (an NLP structure S) to a KLSH code depends upon the kernel function $K(.,.;\boldsymbol{\theta})$ and the reference set $\boldsymbol{S}^R$ (4.2). So, within this framework, the KLSH codes are optimized via learning the kernel parameters $\boldsymbol{\theta}$, and optionally the reference set $\boldsymbol{S}^R$. One important aspect of the proposed optimization setting is that the parameters under optimization are *shared* between all the hash functions jointly, and are not specific to any of the hash functions.

4.3.1 Mutual Information as an Objective Function

Intuitively, we want to generate KLSH codes that are maximally informative about the class labels. Thus, for optimizing the KLSH codes, a natural objective function is the mutual information (MI) between KLSH codes of $\boldsymbol{S}$ and the class labels, $\mathcal{I}(\boldsymbol{c} : y)$ (Cover and Thomas, 2006).

$$\boldsymbol{\theta}^*, \boldsymbol{S}^{R^*} \leftarrow \underset{\boldsymbol{\theta},\ \boldsymbol{S}^R : \boldsymbol{S}^R \subset \boldsymbol{S}}{\arg\max}\ \mathcal{I}(\boldsymbol{c} : y);\ \boldsymbol{c} = \text{klsh}(S; \boldsymbol{\theta}, \boldsymbol{S}^R) \tag{4.3}$$

The advantage of MI as the objective, being a fundamental measure of dependence between random variables, is that it is generic enough for optimizing KLSH codes as generalized representations (feature vectors) of data points to be used with any classifier. Unfortunately, exact estimates of MI function in high-dimensional settings is an extremely difficult problem due to the curse of dimensionality, with the present estimators having very high sample complexity (Kraskov et al., 2004; Walters-Williams and Li, 2009; Singh and Póczos, 2014; Gao et al., 2015; Han et al., 2015; Wu and Yang, 2016; Belghazi et al., 2018).[6] Instead, here it is *proposed to maximize a novel, computationally efficient, good approximation of the MI function*.

4.3.2 Approximation of Mutual Information

To derive the approximation, I express the mutual information function as, $\mathcal{I}(\boldsymbol{c} : y) = \mathcal{H}(\boldsymbol{c}) - \mathcal{H}(\boldsymbol{c}|y)$, with $\mathcal{H}(.)$ denoting the Shannon entropy. For binary classification, the expression simplifies to:

$$\mathcal{I}(\boldsymbol{c} : y) = \mathcal{H}(\boldsymbol{c}) - p(y = 0)\mathcal{H}(\boldsymbol{c}|y = 0) - p(y = 1)\mathcal{H}(\boldsymbol{c}|y = 1).$$

To compute the mutual information, we need to efficiently compute joint entropy of KLSH code bits, $\mathcal{H}(\boldsymbol{c})$. We *propose a good approximation of* $\mathcal{H}(\boldsymbol{c})$, as described below; same applies for $\mathcal{H}(\boldsymbol{c}|y = 0)$ and $\mathcal{H}(\boldsymbol{c}|y = 1)$.

[6] The sample complexity of an entropy estimator for a discrete variable distribution is characterized in terms of its support size s, and it is proven to be not less than $s/\log(s)$ (Wu and Yang, 2016). Since the support size for hashcodes is exponential in the number of bits, sample complexity would be prohibitively high unless dependence between the hash code bits is exploited.

$$\mathcal{H}(\boldsymbol{c}) = \sum_{l=1}^{H} \mathcal{H}(c_l) - \mathcal{TC}(\boldsymbol{c}) \approx \sum_{l=1}^{H} \mathcal{H}(c_l) - \mathcal{TC}(\boldsymbol{c}; \boldsymbol{z}^*); \tag{4.4}$$

$$\mathcal{TC}(\boldsymbol{c}; \boldsymbol{z}) = \mathcal{TC}(\boldsymbol{c}) - \mathcal{TC}(\boldsymbol{c}|\boldsymbol{z}). \tag{4.5}$$

In Eq. 4.4, the first term is the sum of marginal entropies for the KLSH code bits. Marginal entropies for binary variables can be computed efficiently. Now, let us understand how to compute the second term in the approximation (4.4). Herein, $\mathcal{TC}(\boldsymbol{c}; \boldsymbol{z})$ describes the amount of *Total Correlations (Multi-variate Mutual Information)* [7] within $\boldsymbol{c}$ that can be explained by a latent variables representation $\boldsymbol{z}$.

$$\boldsymbol{z}^* \leftarrow \underset{\boldsymbol{z}:|\boldsymbol{z}|=|\boldsymbol{c}|}{\arg\max}\, \mathcal{TC}(\boldsymbol{c}; \boldsymbol{z}) \tag{4.6}$$

An interesting aspect of the quantity $\mathcal{TC}(\boldsymbol{c}; \boldsymbol{z})$ is that one can compute it efficiently for optimized $\boldsymbol{z}^*$ that explains maximum possible Total Correlations present in $\boldsymbol{c}$, s.t. $\mathcal{TC}(\boldsymbol{c}|\boldsymbol{z}) \approx 0$. In (Ver Steeg and Galstyan, 2014), an unsupervised algorithm called CorEx [8] is proposed for obtaining such latent variables representation. Their algorithm is efficient for binary input variables, demonstrating a low sample complexity even in very high dimensions of input variables. Therefore it is particularly relevant for computing the proposed joint entropy approximation on hashcodes. For practical purposes, the dimension of latent representation $\boldsymbol{z}$ can be kept much smaller than the dimension of KLSH codes, i.e. $|\boldsymbol{z}| \ll H$. This helps to reduce the cost for computing the proposed MI approximation to negligible during the optimization (4.3).

Denoting the joint entropy approximation as $\bar{\mathcal{H}}(\boldsymbol{c})$, we express the approximation of the mutual information as:

[7] "Total correlation" was defined in (Watanabe, 1960).

[8] `https://github.com/gregversteeg/CorEx`

$$\bar{\mathcal{I}}(\boldsymbol{c}:y) = \bar{\mathcal{H}}(\boldsymbol{c}) - p(y=0)\bar{\mathcal{H}}(\boldsymbol{c}|y=0) - p(y=1)\bar{\mathcal{H}}(\boldsymbol{c}|y=1).$$

For computation efficiency as well as robustness w.r.t. overfitting, *I use small random subsets (of size γ) from a training set for stochastic empirical estimates of* $\bar{\mathcal{I}}(\boldsymbol{c}:y)$, motivated by the idea of stochastic gradients (Bottou, 2010). For a slight abuse of notation, when obtaining an empirical estimate of $\bar{\mathcal{I}}(\boldsymbol{c}:y)$ using samples set $\{\boldsymbol{C}, \boldsymbol{y}\}$, I simply denote the estimate as: $\bar{\mathcal{I}}(\boldsymbol{C}:\boldsymbol{y})$. Here it is also interesting to note that computation of $\bar{\mathcal{I}}(\boldsymbol{c}:y)$ is very easy to parallelize since the kernel matrices and hash functions can be computed in parallel.

It is worth noting that in the proposed approximation of the MI, both terms need to be computed. In contrast, in the previously proposed variational lower bounds for MI (Barber and Agakov, 2003; Chalk et al., 2016; Gao et al., 2016; Alemi et al., 2017), MI is expressed as, $\mathcal{I}(\boldsymbol{c}:y) = \mathcal{H}(y) - \mathcal{H}(y|\boldsymbol{c})$, so as to obtain a lower bound simply by upper bounding the conditional entropy term with a cross entropy term while ignoring the first term as a constant. Clearly, *these approaches are not using MI in its true sense*, rather using conditional entropy (or cross entropy) as the objective. Further, the proposed sapproximation of MI also allows semi-supervised learning as the first term is computable even for hashcodes of unlabeled examples.

4.3.3 Algorithms for Optimization

Using the proposed approximate mutual information function as an objective, one can optimize the kernel parameters either using grid search or an MCMC procedure.

For optimizing the reference set $\boldsymbol{S}^R$ (of size M) as a subset of $\boldsymbol{S}$, via maximization of the same objective, I propose a greedy algorithm with pseudo code in Algorithm 3. Initially, $\boldsymbol{S}^R$ is initialized with a random subset of $\boldsymbol{S}$ (line 1). Thereafter, $\bar{\mathcal{I}}(.)$ is maximized greedily, updating one element in $\boldsymbol{S}^R$ in each greedy step (line 3); greedy maximization of MI-like objectives has been successful (Gao et al., 2016;

Algorithm 3 Optimizing Reference Set for KLSH

Require: Train dataset, $\{\boldsymbol{S}, \boldsymbol{y}\}$; size of the reference set, M; β, γ are number of samples from $\boldsymbol{S}$, as candidates for $\boldsymbol{S}^R$, and for computing the objective, respectively.

1: $\boldsymbol{S}^R \leftarrow$ randomSubset($\boldsymbol{S}$, M) % M samples from $\boldsymbol{S}$
2: % optimizing the reference set up to size M greedily
3: **for** $j = 1 \rightarrow M$ **do**
4: $\quad \boldsymbol{S}^{eo}, \boldsymbol{y}^{eo} \leftarrow$ randomSubset($\{\boldsymbol{S}, \boldsymbol{y}\}, \gamma$) % γ samples from $\boldsymbol{S}$ for estimating the objective $\bar{\mathcal{I}}(.)$.
5: $\quad \boldsymbol{S}^{cr} \leftarrow$ randomSubset($\boldsymbol{S}$, β) % β samples from $\boldsymbol{S}$ as choices for selection to $\boldsymbol{S}^R$
6: $\quad$ % iterate over candidates elements for greedy step
7: $\quad$ **for** $q = 1 \rightarrow \beta$ **do**
8: $\quad\quad S_j^R \leftarrow S_q^{cr}$ % S_q^{cr} is a choice for selection to $\boldsymbol{S}^R$
9: $\quad\quad \boldsymbol{c}_i^{eo} \leftarrow$ klsh(S_i^{eo}; $\boldsymbol{\theta}$, $\boldsymbol{S}^R$) $\forall S_i^{eo} \in \boldsymbol{S}^{eo}$ % Eq. 4.2
10: $\quad\quad \boldsymbol{mi}_q \leftarrow \bar{\mathcal{I}}(\boldsymbol{C}^{eo}, \boldsymbol{y}^{eo})$ % estimating objective
11: $\quad S_j^R \leftarrow$ chooseElementWithMaxMI($\boldsymbol{mi}$, $\boldsymbol{S}^{cr}$)
12: **return** $\boldsymbol{S}^R$

Krause et al., 2008). Employing the paradigm of stochastic sampling, for estimating $\bar{\mathcal{I}}(.)$, I randomly sample a small subset of $\boldsymbol{S}$ (of size γ) along with their class labels (line 4). Also, in a single greedy step, I consider only a small random subset of $\boldsymbol{S}$ (of size β) as candidates for selection into $\boldsymbol{S}^R$ (line 5); for $\beta \gg 1$, with high probability, each element in $\boldsymbol{S}$ should be seen as a candidate at least once by the algorithm. Algorithm 3 requires kernel computations of order, $O(\gamma M^2 + \gamma\beta M)$, with β, γ being the sampling size constants; in practice, $M \ll N$. Note that $\boldsymbol{\theta}$ and $\boldsymbol{S}^R$ can be optimized in an iterative manner.

The proposed hashcode representation learning approach in this chapter is directly applicable for non-stationary convolution kernels which I had introduced in the previous chapter. The parameters emerging from a nonstationary extension of a convolution kernel can also be learned by maximizing the same objective, $\bar{\mathcal{I}}(.)$, using the well known Metropolis-Hastings MCMC procedure (Hastings, 1970).

4.4 Experiments

I evaluate the model "KLSH-RF" (kernelized locality-sensitive hashing with random forest) for the biomedical relation extraction task using four public datasets, AIMed, BioInfer, PubMed45, BioNLP, as briefed

below.[9] Figure 4.1 illustrates that the task is formulated as a binary classification of extraction candidates. For evaluation, it is a standard practice to compute precision, recall, and F1 score on the positive class (i.e., identifying valid extractions).

4.4.1 Details on Datasets and Structural Features

AIMed & BioInfer

For AIMed and BioInfer datasets, cross-corpus evaluation has been performed in many previous works (Airola et al., 2008; Tikk et al., 2010; Peng and Lu, 2017; Hsieh et al., 2017). Herein, the task is of identifying pairs of interacting proteins (PPI) in a sentence while ignoring the interaction type. I follow the same evaluation setup, using Stanford Dependency Graph parses of text sentences to obtain undirected shortest paths as structural features for use with a path kernel (PK) to classify protein-protein pairs.

PubMed45 & BioNLP

I use PubMed45 and BioNLP datasets for an extensive evaluation of the proposed model, KLSH-RF (Kim et al., 2009, 2011; Nédellec et al., 2013). Annotations in these datasets are richer in the sense that a bio-molecular interaction can involve up to two participants, along with an optional catalyst, and an interaction type from an unrestricted list. In PubMed45 (BioNLP) dataset, 36% (17%) of the "valid" interactions are such that an interaction must involve two participants and a catalyst. For both datasets, I use abstract meaning representation (AMR) to build subgraph or shortest path-based structural features (Banarescu et al., 2013), for use with graph kernels (GK) or path kernels (PK) respectively, as done in the recent works evaluating these datasets (Garg et al., 2016; Rao et al., 2017). For a fair comparison of the classification models, I use the same bio-AMR parser (Pust et al., 2015a) as in the previous works. As in Chapter 2 (Garg et al., 2016), the PubMed45 dataset is split into 11 subsets for evaluation, at paper level. Keeping one of the subsets for testing, I use the others for training a binary classifier. This procedure is repeated for all 11 subsets in order to obtain the final F1 scores (mean and standard deviation values are

[9] PubMed45 dataset is available here: `github.com/sgarg87/big_mech_isi_gg/tree/master/pubmed45_dataset`; the other three datasets are here: `corpora.informatik.hu-berlin.de`

reported from the numbers for 11 subsets). For BioNLP dataset (Kim et al., 2009, 2011; Nédellec et al., 2013), I use training datasets from years 2009, 2011, 2013 for learning a model, and the development dataset from year 2013 as the test set; the same evaluation setup is followed in (Rao et al., 2017).

In addition to the models previously evaluated on these datasets, I also compare KLSH-RF model to KLSH-kNN (kNN classifier with KSLH approximation).

For PubMed45 and BioNLP datasets, for the lack of evaluations of previous works on these datasets, I perform extensive empirical evaluation myself of competitive neural network models, LSTM, Bi-LSTM, LSTM-CNN, CNN; from fine-grained tuning, for PubMed45 & PubMed45-ERN datasets, the tuned neural architecture was a five-layer network, [8, 16, 32, 16, 8], having 8, 16, 32, 16, and 8 nodes, respectively, in the 1st, 2nd, 3rd, 4th, 5th hidden layers; for BioNLP dataset, the tuned neural architecture was a two layer network, [32, 32].

4.4.2 Parameter Settings

I use GK and PK, both using the same word vectors, with kernel parameter settings same as in (Garg et al., 2016; Mooney and Bunescu, 2005).

Reference set size, M, doesn't need tuning in the proposed model; there is a trade-off between compute cost and accuracy; by default, I keep $M = 100$. For tuning any other parameters in the proposed model or competitive models, including the choice of a kernel similarity function (PK or GK), I use 10% of training data, sampled randomly, for validation purposes. From a preliminary tuning, I set parameters, $H = 1000$, $R = 250$, $\eta = 30$, $\alpha = 2$, and choose RMM as the KLSH technique from the three choices discussed in Section 4.1.1; same parameter values are used across all the experiments unless mentioned otherwise.

When selecting reference set $\boldsymbol{S}^R$ randomly, I perform 10 trials, and report mean statistics. (Variance across these trials is small, empirically.) The same applies for KLSH-kNN. When optimizing $\boldsymbol{S}^R$ with Algorithm 3, I use β=1000, γ=300 (sampling parameters are easy to tune). I employ 4 cores on an i7 processor, with 16GB memory.

Models	(AIMed, BioInfer)	(BioInfer, AIMed)
SVM_1 (Airola08) SVM_2 (Airola08)	0.25 0.47	0.44 0.47
SVM (Miwa09)	0.53	0.50
SVM (Tikk10)	0.41 (0.67, 0.29)	0.42 (0.27, 0.87)
CNN (Nguyen15-Rios18)	0.37	0.45
Bi-LSTM (Kavuluru17-Rios18)	0.30	0.47
CNN (Peng17)	0.48 (0.40, 0.61)	0.50 (0.40, 0.66)
RNN (Hsieh17)	0.49	0.51
CNN-RevGrad (Ganin16-Rios18)	0.43	0.47
Bi-LSTM-RevGrad (Ganin16-Rios18)	0.40	0.46
Adv-CNN (Rios18)	0.54	0.49
Adv-Bi-LSTM (Rios18)	**0.57**	0.49
KLSH-kNN	0.51 (0.41, 0.68)	0.51 (0.38, 0.80)
KLSH-RF	**0.57** **(0.46, 0.75)**	**0.54** **(0.37, 0.95)**

Table 4.1: Cross-corpus evaluation results for (training, test) pairs of PPI datasets, AIMed and BioInfer datasets. For each model, I report F1 score in the first row corresponding to it. In some of the previous works, precision, recall numbers are not reported; wherever available, I show precision, recall numbers as well, in brackets. Here, "Ganin16-Rios18" means that the model is originally proposed in (Ganin et al., 2016), and evaluated for these datasets by (Rios et al., 2018).

4.4.3 Main Results for KLSH-RF

In the following I compare the simplest version of KLSH-RF model that is optimized by learning the kernel parameters via maximization of the MI approximation, as described in Section 4.3 ($\gamma = 1000$). In summary, KLSH-RF model outperforms state-of-the-art models consistently across the four datasets, along with very significant speedups in training time w.r.t. traditional kernel classifiers.

4.4.3.1 Results for AIMed and BioInfer Datasets

In reference to Table 4.1, KLSH-RF gives an F1 score significantly higher than state-of-the-art kernel-based models (6 pts gain in F1 score w.r.t. KLSH-kNN), and consistently outperforms the neural models.

When using AIMed for training and BioInfer for testing, there is a tie between Adv-Bi-LSTM (Rios et al., 2018) and KLSH-RF. However, KLSH-RF still outperforms their Adv-CNN model by 3 pts; further, the performance of Adv-CNN and Adv-Bi-LSTM is not consistent, giving a low F1 score when training on the BioInfer dataset for testing on AIMed. For the latter setting of AIMed as a test set, we obtain an F1 score improvement by 3 pts w.r.t. the best competitive models, RNN & KLSH-kNN. Overall, the performance of KLSH-RF is more consistent across the two evaluation settings, in comparison to any other competitive model.

The models based on adversarial neural networks (Ganin et al., 2016; Rios et al., 2018), Adv-CNN, Adv-Bi-LSTM, CNN-RevGrad, Bi-LSTM-RevGrad, are learned jointly on labeled training datasets and unlabeled test sets, whereas our model is purely supervised. In contrast to our principled approach, there are also system-level solutions using multiple parses jointly, along with multiple kernels, and knowledge bases (Miwa et al., 2009; Chang et al., 2016). We refrain from comparing KLSH-RF w.r.t. such system level solutions, as it would be an unfair comparison from a modeling perspective.

4.4.3.2 Results for PubMed45 and BioNLP Datasets

A summary of main results is presented in Table 4.2. "PubMed45-ERN" is another version of the PubMed45 dataset from (Garg et al., 2016), with ERN referring to entity recognition noise. Clearly, the proposed model gives F1 scores significantly higher than SVM, LSTM, Bi-LSTM, LSTM-CNN, CNN, and KLSH-kNN model. For PubMed45, PubMed45-ERN, and BioNLP, the F1 score for KLSH-RF is higher by 6 pts, 8 pts, and 3 pts respectively w.r.t. state of the art; KLSH-RF is the most consistent in its performance across the datasets and significantly more scalable than SVM. Note that standard deviations of F1 scores are high for the PubMed45 dataset (and PubMed45-ERN) because of the high variation in distribution of text across the 11 test subsets (the F1 score improvements with the proposed model are statistically significant, p-value=4.4e-8).

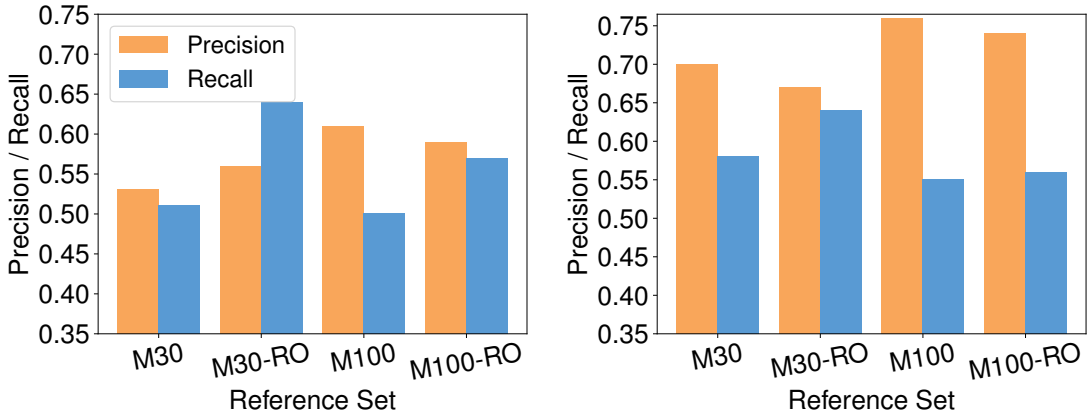
Models	PubMed45	PubMed45-ERN	BioNLP
SVM (Garg16)	0.45 ± 0.25 (0.58, 0.43)	0.33 ± 0.16 (0.33, 0.45)	0.46 (0.35, 0.67)
LSTM (Rao17)	N.A.	N.A.	0.46 (0.51, 0.44)
LSTM	0.30 ± 0.21 (0.38, 0.28)	0.29 ± 0.14 (0.42, 0.33)	0.59 (0.89, 0.44)
Bi-LSTM	0.46 ± 0.26 (0.59, 0.43)	0.37 ± 0.15 (0.45, 0.40)	0.55 (0.92, 0.39)
LSTM-CNN	0.50 ± 0.27 (0.55, 0.50)	0.31 ± 0.17 (0.35, 0.40)	0.60 (0.77, 0.49)
CNN	0.51 ± 0.28 (0.46, 0.46)	0.33 ± 0.18 (0.36, 0.32)	0.60 (0.80, 0.48)
KLSH-kNN	0.46 ± 0.21 (0.44, 0.53)	0.23 ± 0.13 (0.23, 0.29)	0.60 (0.63, 0.57)
KLSH-RF	$\mathbf{0.57 \pm 0.25}$ (**0.63, 0.55**)	$\mathbf{0.45 \pm 0.22}$ (**0.51, 0.52**)	**0.63** (**0.78, 0.53**)

Table 4.2: Evaluation results for PubMed45 and BioNLP datasets. For each model, I report F1 score (mean $\pm$ standard deviation) in the first row corresponding to it, and show mean-precision, mean-recall numbers as well, in brackets. For BioNLP, I don't show standard deviation since there is only one fixed test subset.

For the PubMed45 dataset, there are no previously published results with a neural model (LSTM). The LSTM model of (Rao et al., 2017), proposed specifically for the BioNLP dataset, is not directly applicable for the PubMed45 dataset because the list of interaction types in the latter is unrestricted.

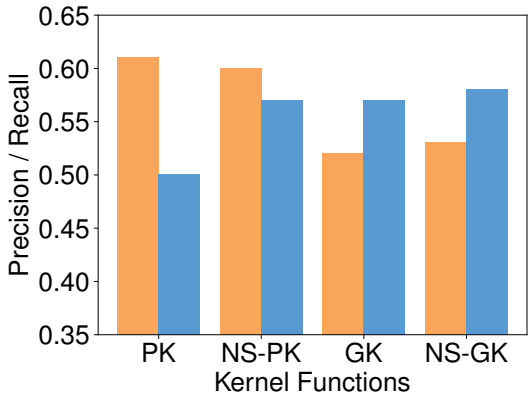
4.4.4 Detailed Analysis of KLSH-RF

While we obtain superior results with the basic KLSH-RF model w.r.t. state-of-the-art methods using just core optimization of the kernel parameters $\boldsymbol{\theta}$, in this subsection I analyze how we can further improve the model. In Figure 4.4 I present our results from optimization of other aspects of the KLSH-RF model: reference set optimization (RO) and non-stationary kernel parameters learning (NS). I report mean values for precision, recall, F1 scores. For these experiments, I focus on PubMed45 and BioNLP datasets.

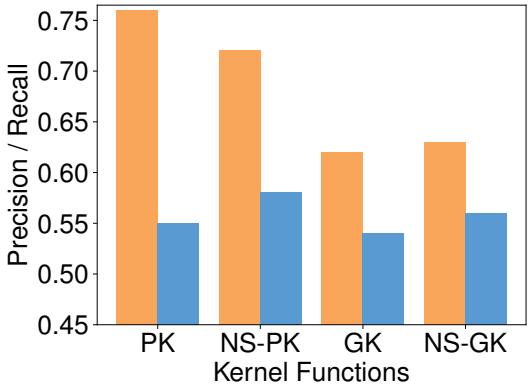


(a) Reference Set Opt. (PubMed45)

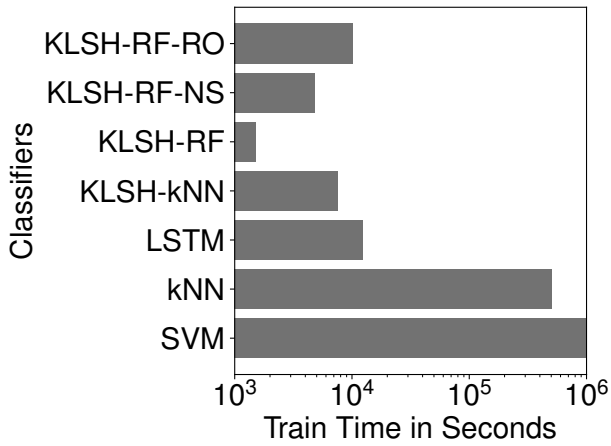
(b) Reference Set Opt. (BioNLP)



(c) NSK Learning (PubMed45)



(d) NSK Learning (BioNLP)



(e) Training Time (BioNLP)

Figure 4.4: Detailed Evaluation of KLSH-RF model, using PubMed45 and BioNLP datasets. Here, orange and blue bars are for precision and recall numbers respectively. "NSK" refers to nonstationary kernel learning; PK & GK denote Path Kernels and Graph Kernels respectively; NS-PK and NS-GK are extensions of PK and GK respectively, with addition of nonstationarity based binary parameters; "M30" represents $\boldsymbol{S}^R$ of size 30 selected randomly, and the suffix "RO" in "M30-RO" refers to optimization of $\boldsymbol{S}^R$ (Reference optimization) in contrast to random selection of $\boldsymbol{S}^R$.

4.4.4.1 Reference Set Optimization

In Figure 4.4(a) and 4.4(b), I analyze the effect of the reference set optimization (RO), in comparison to random selection, and find that the optimization leads to significant increase in recall (7-13 pts) for PubMed dataset along with a marginal increase/decrease in precision (2-3 pts); I used PK for these experiments. For the BioNLP dataset, the improvements are not as significant. Further, as expected, the improvement is more prominent for smaller size of reference set (M). To optimize reference set $\boldsymbol{S}^R$ for $M = 100$, it takes approximately 2 to 3 hours (with $\beta = 1000, \gamma = 300$ in Algorithm 3).

4.4.4.2 Nonstationary Kernel Learning (NSK)

In Figure 4.4(c) and 4.4(d), I compare performance of non-stationary kernels, w.r.t. traditional stationary kernels (M=100). As proposed in Chapter 3, the idea is to extend a convolution kernel (PK or GK) with non-stationarity-based binary parameters (NS-PK or NS-GK), optimized using an MCMC procedure via maximizing the proposed MI approximation based objective ($\gamma = 300$). For the PubMed45 dataset with PK, the advantage of NSK learning is more prominent, leading to high increase in recall (7 pts), and a very small drop in precision (1 pt). Compute time for learning the non-stationarity parameters in KLSH-RF model is less than an hour.

4.4.4.3 Compute Time

Compute times to train all the models are reported in Figure 4.4(e) for the BioNLP dataset; similar time scales apply for other datasets. It is observed that the basic KLSH-RF model has a very low training cost, w.r.t. models like LSTM, KLSH-kNN, etc. (similar analysis applies for inference cost). The extensions of KLSH-RF, KLSH-RF-RO and KLSH-RF-NS, are more expensive yet cheaper to train than LSTM and SVM.

4.5 Related Work

Besides some related work mentioned in the previous sections, this section focuses on relevant state-of-the-art literature in more details.

Other Hashing Techniques

In addition to hashing techniques considered in this thesis, there are other locality-sensitive hashing techniques (Grauman and Fergus, 2013; Zhao et al., 2014; Wang et al., 2017) which are either not kernel based, or they are defined for specific kernels that are not applicable for hashing of NLP structures (Raginsky and Lazebnik, 2009). In deep learning, hashcodes are used for similarity search but classification of objects (Liu et al., 2016).

Hashcodes for Feature Compression

Binary hashing (not KLSH) has been used as an approximate feature compression technique in order to reduce memory and computing costs (Li et al., 2011; Mu et al., 2014). Unlike prior approaches, this work proposes to use hashing as a representation learning (feature extraction) technique.

Using Hashcodes in NLP

In NLP, hashcodes were used only for similarity or nearest neighbor search for words/tokens in various NLP tasks (Goyal et al., 2012; Li et al., 2014; Shi and Knight, 2017); our work is the first to explore kernel-hashing of various NLP structures, rather than just tokens.

Kernel Approximations

Besides the proposed model, there are other kernel-based scalable techniques in the literature, which rely on approximation of a kernel matrix or a kernel function (Williams and Seeger, 2001; Moschitti, 2006; Rahimi and Recht, 2008; Pighin and Moschitti, 2009; Zanzotto and Dell'Arciprete, 2012; Severyn and Moschitti, 2013; Felix et al., 2016). However, those approaches are only used as computationally efficient approximations of the traditional, computationally-expensive kernel-based classifiers; unlike those

approaches, our method is not only computationally more efficient but also yields considerable accuracy improvements.

4.6 Chapter Summary

In this chapter I proposed to use a well-known technique, kernelized locality-sensitive hashing (KLSH), in order to derive feature vectors from natural language structures. More specifically, I proposed to use random subspaces of KLSH codes for building a random forest of decision trees. I find this methodology particularly suitable for modeling natural language structures in supervised settings where there are significant mismatches between the training and the test conditions. Moreover I optimize a KLSH model in the context of classification performed using a random forest, by maximizing an approximation of the mutual information between the KLSH codes (feature vectors) and the class labels. I apply the proposed approach to the difficult task of extracting information about bio-molecular interactions from the semantic or syntactic parsing of scientific papers. Experiments on a wide range of datasets demonstrate the considerable advantages of the novel method.

Chapter 5

Nearly Unsupervised Hashcode Representations

In the previous chapter, kernelized locality sensitive hashcodes based representation learning approach has been proposed that has shown to be the most successful in terms of accuracy and computational efficiency for the task (Garg et al., 2019). In that approach, kernelized (binary) hashcode of an example acts as its representation vector, computed as a function of kernel similarities of the example w.r.t. (only) a constant sized *reference set* of examples, and then the representations vectors are fed into a Random Forest of decision trees as the final classification model.

The choice of the reference set and the parameters of a kernel function are optimized in a purely supervised manner, shared between all the (kernel based) hash functions, whereas an individual hash function is constructed in a *randomized* fashion, serving for randomized semantic categorization of examples (binary semantic feature). It is suggested to obtain thousands of (randomized) semantic features extracted from natural language examples into binary hashcodes, and then making classification decision as per the *features* using hundreds of decision trees, which is the core of the robust classification approach.

Although hashcode representation learning is a very promising direction of research, there is a significant scope for improving upon the supervised approach presented in the previous chapter. Even if we extract thousands of semantic features using the hashing approach, it is difficult to ensure that the features extracted from a training set of examples would generalize to a test set. While the inherent randomness in constructing hash functions from a training set of examples can help towards generalization in the case of

absence of a test set, there should be better alternatives if we do have the knowledge of a test set of examples. What if we construct hash functions in an intelligent manner via exploiting the additional knowledge of unlabeled examples in a test set, performing *fine-grained optimization of each hash function* rather than relying upon randomness, so as to extract semantic features which generalize?

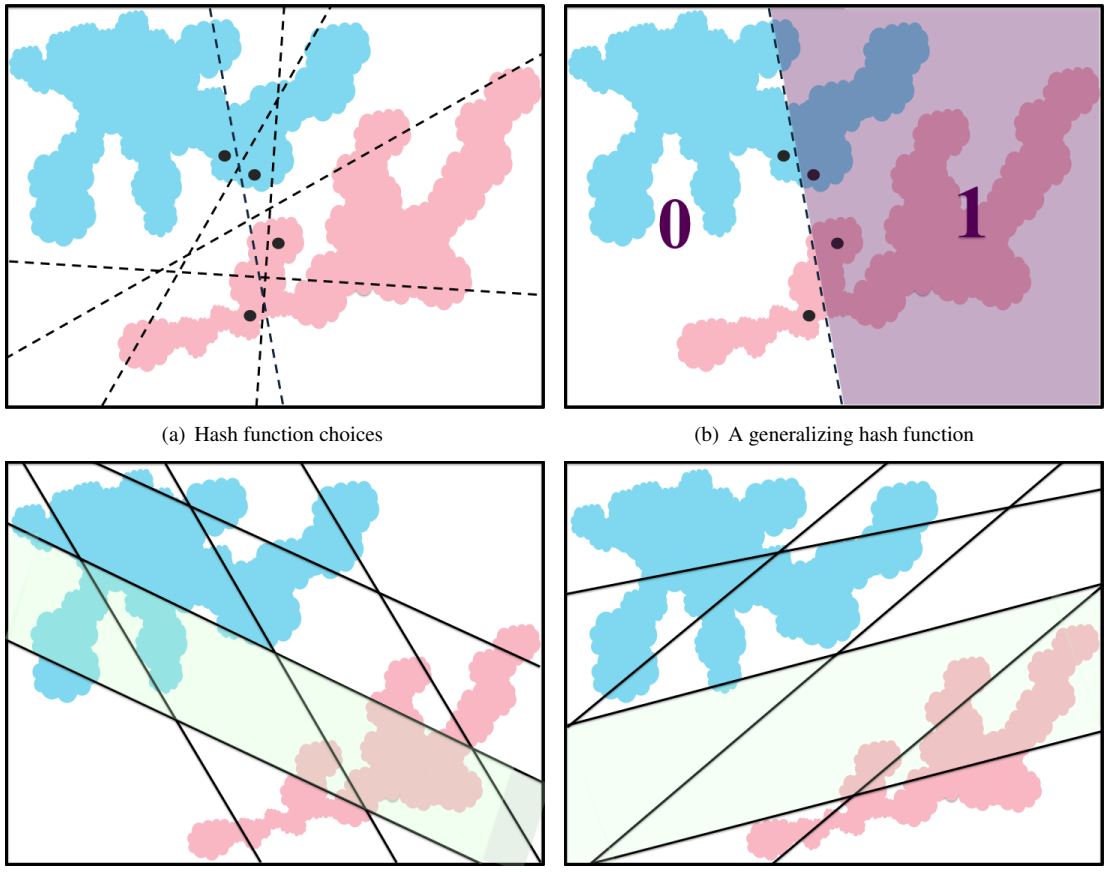
Along these lines, in this chapter, I propose a new framework for learning hashcode representations accomplishing two important (inter-related) extensions w.r.t. the supervised approach presented in the previous chapter:

(a) I propose to use a semi-supervised learning setting, that is *nearly unsupervised* as it is suggested to use only the knowledge of which set an example comes from, a training set or a test set, along with the example itself, whereas the actual class labels of examples (from a training set) are input only to the final supervised-classifier, such as an RF, which takes input of the learned hashcodes as representation (feature) vectors of examples along with their class labels;

(b) I introduce multiple concepts for fine-grained optimization of hash functions, employed in a novel information-theoretic algorithm that constructs hash functions greedily one by one. In supervised settings, fine-grained (greedy) optimization of hash functions could lead to overfitting whereas, in the proposed nearly-unsupervised framework, it allows flexibility for explicitly maximizing the generalization capabilities of hash functions, as I explain in the following, introducing two key ideas for the fine-grained optimization.

Key Ideas for Fine-Grained Optimization of Hash Functions

To understand our *first key idea* for fine-grained optimization of a hash function, see Figure 5.1. In Figure 5.1(a), I demonstrate how a single hash function is constructed from a small set of examples by splitting the set into two subsets, showing many possible splits as dashed lines corresponding to different choices for a hash function. While in the previous works (Garg et al., 2019, 2018; Joly and Buisson, 2011), a split is chosen randomly from all the choices, I propose to *select an optimal split* such that it helps in generalization across training and test sets, i.e., a hash function should assign same value, zero as well as one, to



(a) Hash function choices

(b) A generalizing hash function

(c) Generalizing hash functions

(d) Non-generalizing hash functions

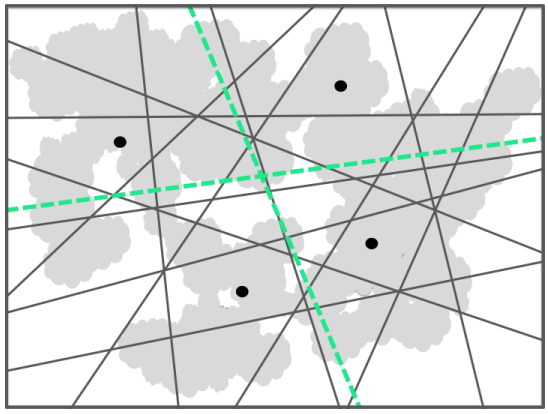
Figure 5.1: In this figure, I illustrate how to construct hash functions which generalize across a training and a test set. Blue and red colors denote a training set and a test set respectively. A hash function is constructed by splitting a very small subset of examples into two parts. In Figure 5.1(a), given the set of four examples selected randomly, there are many choices possible for splitting the set, corresponding to difference choices of hash functions which are denoted with dashed lines. The optimal choice of hash function is shown in Figure 5.1(b) since it generalizes well, assigning same value to many examples from both training & test sets. In Figure 5.1(c) and Figure 5.1(d), I analyze generalization of multiple hash functions jointly. In both the figures, hash functions are represented as solid lines, satisfying the criterion of locality sensitivity; the difference, however, is that the hash functions in Figure 5.1(d), do not generalize well from a training set (blue color) to a test set (red color); the proposed criterion to analyze generalization of multiple hash functions is to see if there are training as well as test examples within a cluster (shaded region); the inter-spaces between the vertical (or horizontal) lines can be thought of as clusters; as we can see, a cluster in Figure 5.1(c) contains examples from both training & test sets, which is not true for many clusters in Figure 5.1(d).

many examples from both training & test sets as shown in Figure 5.1(b). The concept of evaluating generalization of hashcodes can be further extended to multiple hash functions, as illustrated in Figure 5.1(c) and Figure 5.1(d). In Figure 5.1(d), I show six hash functions which produce valid locality sensitive hashcodes but not generalizing across training & test sets, whereas the ones in Figure 5.1(c) do generalize. The criterion I propose to gauze the generalization of hashcodes is to see if there are training as well test examples within a cluster of examples (for instance, the shaded regions in the two figures), with *clusters being defined from a subset of the hash functions*. Thus, for constructing a hash function from a given set of examples, one can optimize upon the choice of a split of the set such that the function, along with the other hash functions, generalize across training & test sets as per the criteria introduced in the above.

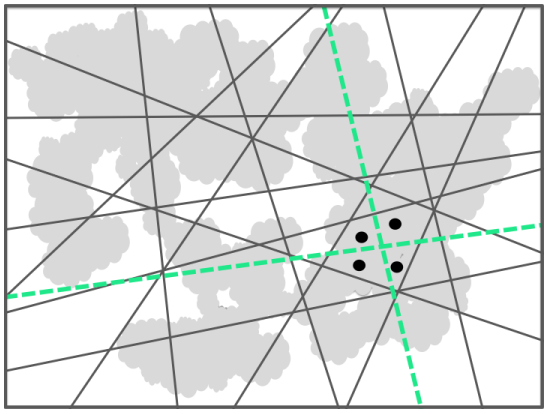
My *second key idea* for fine-grained optimization of a hash function, referring back to Figure 5.1(a), is to *select the set of examples intelligently* that is required for construct a hash function. In the previous works, a hash function is constructed from a set of examples which is sub-sampled randomly from a superset of examples (say a training set). This approach of global sampling may lead to redundancy between hash functions. So I propose that, for constructing some of the latter hash functions in a greedy algorithm, the examples to construct a hash function should be sub-sampled locally from within a cluster so as to capture fine-grained differences between examples, helping reduce redundancies between the functions. See Figure 5.2 for a contrast between hash functions constructed via global sampling vs local sampling; in the figure, we see that global sampling is suitable to capture higher-order differences between examples whereas local sampling is to appreciate fine-grained differences between examples. Moreover, from the perspective of generalization, I suggest to sample from a cluster that contains examples from both training & test sets as shown in Figure 5.2(c).

Summary of Contributions

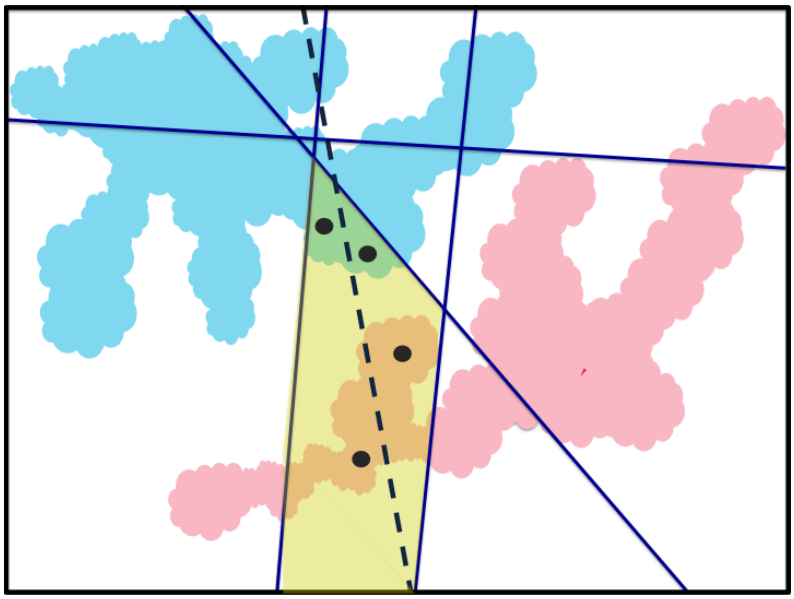
In summary, I make the following contributions in this chapter.



(a) Global sampling



(b) Local sampling from a cluster



(c) Local sampling from a high entropy cluster

Figure 5.2: In all the three figures, dark-gray lines denote hash functions optimized previously, and intersections of the lines give us 2-D cells denoting hashcodes as well as clusters; black dots represent a subset of examples which are used to construct a hash function, and some of the choices to split the subset are shown as green dashed lines. The procedure of sampling the subset varies across the three figures. In Figure 5.2(a), since the four examples from global sampling have already unique hashcodes (2-D cells), the newly constructed hash function (one of the green dashed-lines) adds little information to their representations. In Figure 5.2(b), on the other hand, a hash function constructed from examples, sampled locally from within a cluster, puts some of the examples in the subset into two different cells (hashcodes), so adding more fine-grained information to their representations, hence more advantageous from representation learning perspective. In Figure 5.2(c), training and test examples are denoted with blue and red colors respectively, and the examples to construct a new hash function are sampled locally from within a high entropy cluster, i.e. the one containing examples from both training & test sets.

(i) I extend the recently proposed approach of learning supervised hashcode representations, with a novel nearly-unsupervised learning framework which allows fine-grained optimization of each hash function, one by one in a greedy manner, relying upon the concepts of, (a) *intelligently selecting a small set* of examples to construct a hash function, and also, (b) *informatively splitting* the set, such that hashcode representations generalize across training & test sets.

(ii) For the task of biomedical relation extraction, I evaluate the proposed approach on four public datasets, and obtain significant gain in F1 scores w.r.t. state-of-the-art models including kernel-based approaches as well the ones based on (semi-supervised) adversarial learning of neural networks.

(iii) I show how to employ the nearly unsupervised framework for learning hashcode representations, using not only the traditional kernel similarity functions, but neural networks as well.

5.1 Problem Formulation & Background

In this section, I brief upon the approach of kernelized hashcode representations from the previous chapter (Garg et al., 2019) which has shown state-of-the-art results for bio-medical relation extraction task though the approach, in principle, is applicable for any NLP task involving classification of natural language structures.

We have natural language structures, $\boldsymbol{S} = \{S_i\}_1^N$, such as parse trees, shortest paths, text sentences, etc, with corresponding class labels, $\boldsymbol{y} = \{y_i\}_1^N$. For the examples coming from a training set and a test set, we use notations, $\boldsymbol{S}_T, \boldsymbol{y}_T$, and $\boldsymbol{S}_*, \boldsymbol{y}_*$, respectively. In addition, we define indicator variable, $\boldsymbol{x} = \{x_i\}_{i=1}^N$, for $\boldsymbol{S}$, with $x_i \in \{0, 1\}$ denoting if an example S_i is coming from a test set ($x = 0$, unlabeled example) or a training set ($x = 1$, labeled example). Our goal is infer class labels of the examples, $\boldsymbol{y}_*$, from a test set, $\boldsymbol{S}_*$.

5.1.1 Hashcode Representations

As per the hashcode representation approach, $\boldsymbol{S}$ is mapped to a set of *locality sensitive* hashcodes, $\boldsymbol{C} = \{\boldsymbol{c}_i\}_1^N$, using a set of binary hash functions, i.e. $\boldsymbol{c}_i = \boldsymbol{h}(S_i) = \{h_1(S_i), \cdots, h_H(S_i)\}$. These hashcodes serve as feature vector representations so as to be used with any classification model, for instance, a random forest of decision trees. In the following, I describe the concept of locality sensitivity, discuss a general procedure to learn a set of locality sensitive hash functions.

Locality Sensitivity of Hash Functions

With locality sensitive hash functions, the probability of two examples to be assigned same hashcodes is proportional to their similarity (defined as per any valid similarity/distance function); this also implies that examples that are very similar to each other are assigned hashcodes with minimal Hamming distance to each other, and vica versa (Wang et al., 2014, 2017; Indyk and Motwani, 1998). Locality sensitivity of hash functions serve as the basis for using hashcodes as representations as well as for nearest neighbor search (Garg et al., 2019, 2018; Kulis and Grauman, 2012; Joly and Buisson, 2011). In my approach proposed in Section 5.2, I have an additional use of *locality sensitive hash functions for clustering of examples to facilitate semi-supervised representation learning*.

Basics of Constructing Hash Functions

A locality sensitive hash function, $h_l(.;\boldsymbol{\theta})$, is constructed such that it splits a set of examples, $\boldsymbol{S}_l^R$, into two subsets as shown in Figure 5.1(a), while choosing the set as a small random subset of $\boldsymbol{S}$, i.e. $\boldsymbol{S}_l^R \subset \boldsymbol{S}$ s.t. $|\boldsymbol{S}_l^R| = \alpha \ll N$. In this manner, we can construct a large number of hash functions, $\{h_l(.;\boldsymbol{\theta})\}_{l=1}^H$, from a *reference set*, $\boldsymbol{S}^R = \{\boldsymbol{S}_1^R \cup \cdots \cup \boldsymbol{S}_H^R\}, |\boldsymbol{S}^R| = M \leq \alpha H \ll N$. Although a hash function is constructed from a handful of examples, the parameters, $\boldsymbol{\theta}$, and optionally, the selection of $\boldsymbol{S}^R$ can be optimized using the whole training dataset, $\{\boldsymbol{S}_T, \boldsymbol{y}_T\}$.

Supervised Learning of Kernelized Hash Functions

While, mathematically, a locality sensitive hash function can be of any form, previously introduced locality sensitive hash functions, which are applicable for hashing of natural language structures, are based on kernel methods (Garg et al., 2019, 2018; Joly and Buisson, 2011), relying upon a convolution kernel similarity function $K(S_i, S_j; \boldsymbol{\theta})$ defined for any pair of structures S_i and S_j with kernel-parameter $\boldsymbol{\theta}$ (Haussler, 1999). To construct $h_l(.)$, a kernel-trick based model, such as kNN, SVM, is fit to $\{\boldsymbol{S}_l^R, \boldsymbol{z}_l\}$, with a randomly sampled binary vector, $\boldsymbol{z}_l \in \{0,1\}^\alpha$, that defines the split of $\boldsymbol{S}_l^R$. For computing hashcode $\boldsymbol{c}_i$ for an example S_i, it requires only M number of convolution-kernel similarities of S_i w.r.t. the examples in $\boldsymbol{S}^R$, which makes this approach highly scalable in compute cost terms.

$$\boldsymbol{\theta}^*, \boldsymbol{S}^{R^*} \leftarrow \underset{\boldsymbol{\theta},\ \boldsymbol{S}^R : \boldsymbol{S}^R \subset \boldsymbol{S}_T}{\arg\max}\ \mathcal{I}(\boldsymbol{c} : y);\ \boldsymbol{c} = \boldsymbol{h}(S; \boldsymbol{\theta}, \boldsymbol{S}^R) \tag{5.1}$$

In the previous chapter, it is proposed to optimize all the hash functions jointly by learning only the *parameters which are shared amongst all the functions*, i.e. learning kernel parameters, $\boldsymbol{\theta}$ and the choice of reference set, $\boldsymbol{S}^R \subset \boldsymbol{S}_T$. This optimization is performed in a supervised manner via maximization of the mutual information between hashcodes of examples and their class labels (5.1), using $\{\boldsymbol{S}_T, \boldsymbol{y}_T\}$ for training.

Limitations of the Supervised Hashcode Learning Approach

My key insight in regards to limitation of the approach for supervised learning of hashcode representations, also mentioned in the previous chapter (Garg et al., 2019), is that, to avoid overfitting, learning is intentionally restricted only to the optimization of shared parameters whereas each hash function is constructed in a randomized manner, i.e. random sub-sampling of a subset, $\boldsymbol{S}_l^R$, and a random split of the subset. On the other hand, in a semi-supervised hashcode learning settings like considered in this chapter, it is possible to extend the optimization from global parameters to fine-grained optimization of hash

functions while avoiding overfitting due to the knowledge of examples from a test set (or a large set of unlabeled examples if considering inductive settings), as I demonstrate in Section 5.2. Further, although it is difficult to characterize mathematically, the inherent randomness in sampling of a subset is constitutive towards preserving locality sensitivity of hash functions. In the light of this important algorithmic nature of the core approach, in the proposed framework here, sampling of a subset is pseudo-random, i.e. random sampling of examples locally from within one of the clusters of the examples in the superset.

Semi-Supervised Settings for Learning Hashcodes

I propose to learn hash functions, $\boldsymbol{h}(.)$, using $\boldsymbol{S} = \{\boldsymbol{S}_T, \boldsymbol{S}_*\}$, $\boldsymbol{x}$, and optionally, $\boldsymbol{y}_T$. Herein, $\boldsymbol{S}_*$ is a test set of examples for which we want to infer class labels $\boldsymbol{y}_*$, corresponding to transductive (semi-supervised) settings. One can also consider inductive (semi-supervised) settings where $\boldsymbol{S}_*$ is a very large set of unlabeled examples. Within the semi-supervised setting, inductive or transductive, if we are not using $\boldsymbol{y}_T$ for learning $\boldsymbol{h}(.)$, but only, $\{\boldsymbol{S}, \boldsymbol{x}\}$, I refer to it as *nearly unsupervised* settings. In the next section, I introduce the framework for learning hash functions in above specified learning settings, to optimize global parameters as well optimizing aspects local to a hash function. After learning hash functions, one can compute hashcodes $\boldsymbol{C}$ for $\boldsymbol{S}$, and train a random forest classifier, or any other classifier, using $\{\boldsymbol{C}_T, \boldsymbol{y}_T\}$ to infer class labels for input of hashcodes, $\boldsymbol{C}_*$, computed for $\boldsymbol{S}_*$. One can also train a semi-supervised classifier, using $\{\boldsymbol{C}, \boldsymbol{x}, \boldsymbol{y}_T\}$.

5.2 Semi-Supervised Hashcode Representations

In this section, I propose a novel algorithm for a fine-grained optimization of the hash functions within the semi-supervised (nearly unsupervised) learning framework, with the objective of learning representations generalizing across training & test sets.

5.2.1 Basic Concepts for Fine-Grained Optimization

In the prior works on kernel-similarity based locality sensitive hashing, the first step for constructing a hash function is to randomly sample a small subset of examples, from a superset of examples, $\boldsymbol{S}$, and in the second step, the subset is split into two parts using a *kernel-trick* based model (Garg et al., 2019, 2018; Joly and Buisson, 2011), serving as the hash function, as described in Section 5.1.

In the following, I introduce basic concepts for improving upon these two key aspects of constructing a hash function, while later, in Section 5.2.2, these concepts are incorporated in a unified manner in a novel *information-theoretic algorithm that greedily optimizes hash functions* one by one.

5.2.1.1 Informative Split of a Set of Examples

In Figure 5.1(a), construction of a hash function is pictorially illustrated, showing multiple possible splits, as dotted lines, of a small set of four examples (black dots). (Note that a hash function is shown to be a linear hyperplane only for simplistic explanations of the basic concepts.) While in the previous works, one of the many choices for splitting the set is chosen randomly, I propose to optimize upon this choice. Intuitively, one should choose a split of the set, corresponding to a hash function, such that it gives a balanced split for the whole set of examples, and it should also generalize across training & test sets. In reference to the figure, one simple way to analyze the generalization of a split (so the hash function) is to see if there are training as well test examples on either side of the dotted line. As per this concept, an optimal split of the set of four examples is shown in Figure 5.1(b).

Referring back to Section 5.1, clearly, this is a combinatorial optimization problem, where we need to choose an optimal choice of $\boldsymbol{z}_l \in \{0,1\}^\alpha$ for set, $\boldsymbol{S}_l^R$, to construct $h_l(.)$. For a small value of α, one can either go through all the possible combinations in a brute force manner, or use Markov Chain Monte Carlo sampling. It is interesting to note that, even though a hash function is constructed from a very small set of examples (of size $\alpha \not\gg 1$), the generalization criterion, formulated in the info-theoretic objective introduced in Section 5.2.2, is computed using all the examples available for the optimization, $\boldsymbol{S}$.

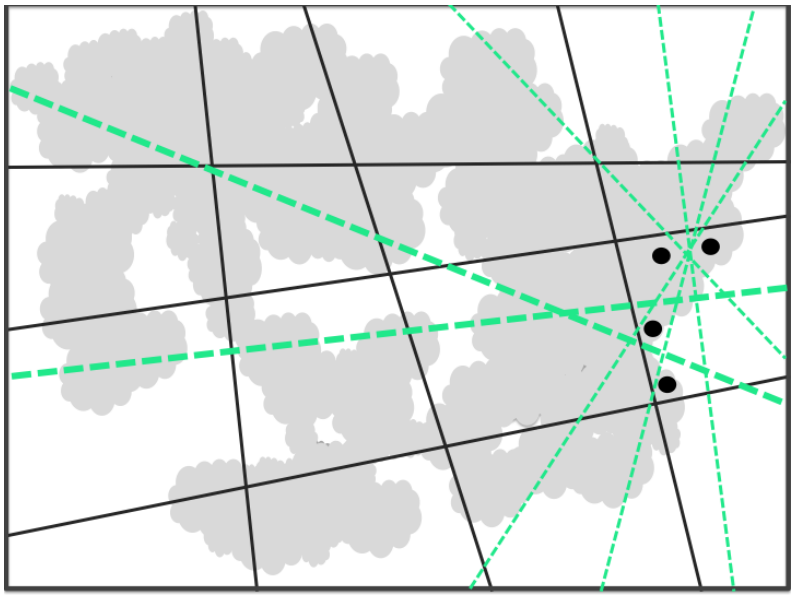


Figure 5.3: Dark-gray lines denote hash functions optimized previously, and gray color represents examples used for the optimization. The intersections of the lines give us 2-D cells, corresponding to hashcodes as well as clusters. For the set of four examples sampled within a cluster, there are many choices to split it (corresponding to hash functions choices), shown with dashed lines. I propose to choose the one which also splits the sets of examples in other (neighboring) clusters in a balanced manner, i.e. having examples in a cluster on either side of the dashed line and cutting through as many clusters as possible. As per this criterion, the thick-green dashed lines are superior choices w.r.t. the thin-green ones.

5.2.1.2 Sampling Examples Locally from a Cluster

Another aspect of constructing a hash function, having a scope for improvement, is sampling of a small subset of examples, $\boldsymbol{S}_l^R \subset \boldsymbol{S}$, that is used to construct a hash function. In the prior works, the selection of such a subset is purely random, i.e. random selection of examples globally from $\boldsymbol{S}$.

In Figure 5.2, I illustrate that, after constructing a small number of hash functions with purely random sub-sampling of the subsets, it is wiser to (randomly) *select examples ($\boldsymbol{S}_l^R$) locally from one of the clusters* of the examples in $\boldsymbol{S}$, rather than sampling globally from $\boldsymbol{S}$; herein, I propose that clustering of all the examples in $\boldsymbol{S}$ can be obtained using the hash functions itself, due to their locality sensitive property. While using a large number of locality sensitive hash functions give us fine-grained representations of examples, *a small subset of the hash functions, of size ζ, defines a valid clustering* of the examples, since examples which are similar to each other should have same *hashcodes serving as cluster labels*.

From this perspective, we can construct first few hash functions from global sampling of examples, what I refer as global hash functions. These global hash functions should serve to provide hashcode representations as well as clusters of examples. Then, using local sampling, we can also construct *local hash functions* to capture more finer details of examples, which may be missed out by global hash functions. As per this concept, we can learn *hierarchical (multi-scale) hashcode representations* of examples, capturing differences between examples from coarser (global hash functions) to finer scales (local hash functions).

Further, I suggest two criteria to choose a cluster for local sampling: (i) a cluster should have a reasonably high number of examples belonging to it; (ii) a cluster should have a balanced ratio of the count of training & test examples so that a new cluster emerging from the split has a high probability of containing training as well test examples, which is desirable from the perspective of having generalized hashcode representations; see Figure 5.2(c).

5.2.1.3 Splitting Neighboring Clusters (Non-redundant Local Hash Functions)

We can also understand the idea of a local hash function as the one which splits a cluster into two parts, indirectly via splitting a set of examples randomly selected from within the cluster. So we can keep on splitting clusters one by one by constructing more and more local hash functions. Though, considering a large number of clusters possible, we would like a hash function to split the neighboring clusters as well, characterizing its non-redundancy w.r.t. the other hash functions; see Figure 5.3 for a pictorial explanation of this concept.

Next I mathematically formalize all the concepts introduced above for fine-grained optimization of hash functions into an information-theoretic objective function.

5.2.2 Information-Theoretic Learning Algorithm

I consider optimizing hash functions greedily, one by one. Referring back to Section 5.1, I define binary random variable x denoting the presence or absence of a class label for an example, S; in other words, it

Algorithm 4 Nearly Unsupervised Hashcode Representation Learning

Require: $\{\boldsymbol{S}, \boldsymbol{x}\}, H, \boldsymbol{\alpha}, \zeta$.

$\boldsymbol{h}(.) \leftarrow \{\}, \boldsymbol{C} \leftarrow \{\}, \boldsymbol{f} \leftarrow \{\}$
`% Greedy step for optimizing hash function,` $h_l(.)$
while $|\boldsymbol{h}(.)| < H$ **do**
  $\alpha_l \leftarrow$ sampleSubsetSize($\boldsymbol{\alpha}$) `% Sample a size for the subset of examples,` $\boldsymbol{S}_l^R$`, to construct a hash function`
  if $|\boldsymbol{h}(.)| < \zeta$ **then**
    $\boldsymbol{S}_l^R \leftarrow$ randomSampleGlobally($\boldsymbol{S}, \alpha_l$) `% Randomly sample examples, globally from` $\boldsymbol{S}$`, for constructing a global hash function.`
  else
    $\boldsymbol{S}_l^R \leftarrow$ randomSampleLocally($\boldsymbol{S}, \boldsymbol{C}, \boldsymbol{x}, \alpha_l, \zeta$) `% Sample examples randomly from, a high entropy cluster, for constructing a local hash function.`
  $h_l(.), f_l \leftarrow$optimizeSplit$\Big(\boldsymbol{S}_l^R, \boldsymbol{S}, \boldsymbol{C}, \boldsymbol{x}\Big)$ `% Optimize split of` $\boldsymbol{S}_l^R$`.`
  $\boldsymbol{c} \leftarrow$ computeHash($\boldsymbol{S}, h_l(.)$)
  $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \boldsymbol{c}, \boldsymbol{h}(.) \leftarrow \boldsymbol{h(.)} \cup h_l(.), \boldsymbol{f} \leftarrow \boldsymbol{f} \cup f_l$.
  $\boldsymbol{h}(.), \boldsymbol{C}, \boldsymbol{f} \leftarrow$ deleteLowInfoFunc($\boldsymbol{h}(.), \boldsymbol{C}, \boldsymbol{f}$) `% delete hash functions from the set with lower objective values`
Return $\boldsymbol{h}(.), \boldsymbol{C}$.

denotes whether an example comes from a training set or a test set. In a greedy step of optimizing a hash function, random variable, $\boldsymbol{c}$, represents the hashcode of an example, S, as per the previously optimized hash functions, $\boldsymbol{h}_{l-1}(.) = \{h_1(.), \cdots, h_{l-1}(.)\}$; $\boldsymbol{c} = \boldsymbol{h}_{l-1}(S)$. Along same lines, c denotes the binary random variable corresponding from the present hash function under optimization, $h_l(.)$; $c = h_l(S)$. I propose to maximize an information-theoretic objective function as below.

$$\underset{h_l()}{\operatorname{argmax}} \mathcal{H}(x, c) - \mathcal{I}(c : \boldsymbol{c}) + \mathcal{H}(x|\boldsymbol{c}); \tag{5.2}$$

$$\boldsymbol{c} = \boldsymbol{h}_{l-1}(S), c = h_l(S)$$

Herein the optimization of a hash function, $h_(.)$, involves: (i) intelligent selection of $\boldsymbol{S}_l^R$, (ii) an informative split of $\boldsymbol{S}_l^R$, i.e. optimizing $\boldsymbol{z}_l$ for $\boldsymbol{S}_l^R$, and (iii) learning of the parameters $\boldsymbol{\theta}$ of a (kernel or neural) model, which is fit to $\{\boldsymbol{S}_l^R, \boldsymbol{z}_l\}$, acting as the hash function.

In the objective function above, maximizing the first term, $\mathcal{H}(x, c)$, i.e. joint entropy on x and c, corresponds to the concept of *informative split* described above in Section 5.2.1, i.e. a hash function

assigning value 0 to labeled (training) as well as unlabeled examples (test), and same applies for value 1; see Figure 5.1. This term is cheap to compute since x and c are both 1-dimensional binary variables.

Referring to the second term in the objective function, mutual information is minimized between the hash function being optimized and the previously optimized hash functions, i.e. $\mathcal{I}(c : \boldsymbol{c})$, so as to ensure minimal redundancies between hash functions. This is related to the concept of constructing a hash function, from examples sampled within a cluster, such that it splits many of the neighboring clusters, as mentioned above in Section 5.2.1; see Figure 5.3. This mutual information function can be computed using the approximation in the previous work by (Garg et al., 2019).

The last quantity in the objective is $\mathcal{H}(x|\boldsymbol{c})$, conditional entropy on x given $\boldsymbol{c}$. Since both quantities, x and $\boldsymbol{c}$ are fixed in the optimization, this term is is not directly effected from the greedy optimization of hash function, $h_l(.)$, as such. Yet, this term has a high significance from the perspective of learning generalized hashcode representations, since maximizing $\mathcal{H}(x|\boldsymbol{c})$ encourages clusters, defined from the hash functions, to contain labeled (training) as well as unlabeled (test) examples. So I propose to maximize this term indirectly via *choosing a cluster informatively, from which to randomly select examples for constructing the hash function*, such that it contains a balanced ratio of the count of training & test examples, i.e. a cluster with high entropy on x, which I refer as a *high entropy cluster*. As explained in Section 5.2.1, the hash function constructed from examples within a cluster splits the clusters itself, and the new clusters emerging from a split of a high entropy cluster should have higher chances to be high entropy clusters themselves, in comparison to the ones emerging from a split of a lower entropy cluster. See Figure 5.2(c) to understand this concept pictorially. Computations in the procedure of characterizing entropies of clusters are cheap, requiring to compute marginal entropy on x for each cluster, and an explicit computation of $H(x|\boldsymbol{c})$ is not required.

It is interesting to observe that the above info-theoretic objective uses only the knowledge of whether an example is labeled or unlabeled (random variable x), while ignoring the actual class label of an example. In this sense, the objective function is *nearly unsupervised*. Here, the emphasis is on fine-grained optimization of each hash function such that hashcode representations generalize across training & test sets. Optionally

one may extend the objective function to include the term, $-\mathcal{H}(y|\boldsymbol{c}, c)$, with y denoting the random variable for a class label.

The above described framework is summarized in Algorithm 4.

Kernel-trick Model or Neural Network as a Hash Function

For optimizing a kernelized hash function given $\boldsymbol{S}_l^R$, $h_l(.)$, when finding an optimal choice of $\boldsymbol{z}_l \in \{0,1\}^\alpha$ for $\boldsymbol{S}_l^R$, we can also optimize kernel parameters, $\boldsymbol{\theta}$, of a kernel-trick based model, which is fit on $\boldsymbol{S}_l^R$ along with a given choice of $\boldsymbol{z}_l$, corresponding to one of the choices for $h_l(.)$. Thus we can learn kernel parameters local to a hash function unlike the previous works. Further, we can fit any (regularized) neural model on $\{\boldsymbol{S}_l^R, \boldsymbol{z}_l\}$, acting as a *neural locality sensitive hash function*. For a random choice of $\boldsymbol{z}_l$ for $\boldsymbol{S}_l^R$, one can expect that the neural hash function may overfit to $\{\boldsymbol{S}_l^R, \boldsymbol{z}_l\}$, so acting as a poor hash function. However it is reasonable to assume that some of the many possible choices for a split of $\boldsymbol{S}_l^R$ should be natural to obtain even with a highly parameterized model, as I observe empirically.

Deleting (Weak) Hash Functions

In the algorithm, I also propose to *delete some of the hash functions* from the set of optimized ones, the ones which have low objective function values w.r.t. the rest. The reason for including this step is to have robustness against an arbitrarily bad choice of *randomly* selected subset, $\boldsymbol{S}_l^R$, from within a chosen cluster. This can be especially useful for neural hashing, because of the possible sensitivity of a neural network to input examples.

Mixture of Values for a Tuning Parameter

Another advantage of the idea of deleting hash functions is that the size of $\boldsymbol{S}_l^R$, i.e. parameter α, doesn't have to be fixed in the algorithm. Deletion of hash functions makes the algorithm robust against a bad choice for the parameter, α, thus allowing us to have a mixture of the values, while avoiding fine-grained tuning of the parameter. Same concept can applied to any other tuning parameter that is local to a hash function.

Parallel Computing

There are many aspects in the algorithm which can be parallelized, such as optimizing a split of a subset to construct a hash function, computing kernel matrices, or neural network learning (on a GPU), or one can even construct multiple hash functions in parallel in each greedy step.

In contrast to the supervised approach proposed in the previous chapter, here, I have proposed a fine-grained greedy optimization of hash functions, while having robustness w.r.t. overfitting, relying upon the nearly unsupervised learning framework which assumes the knowledge of a test set or any other set of unlabeled examples. Next, I demonstrate the applicability of the approach for the highly challenging task of biomedical relation extraction.

5.3 Experiments

For my proposed approach of nearly unsupervised hashcode representations, I use a random forest (RF) as the final supervised-classifier, following the work from the previous chapter (Garg et al., 2019) in which supervised hashcode representations are fed as input feature vectors to an RF (KLSH-RF). I refer to the new kernel based model, introduced in this chapter, as KLSH-NU-RF, and the neural one as NLSH-NU-RF.

For learning nearly-unsupervised hashcode representations (using Algorithm 4) in the proposed models, KLSH-NU-RF & NLSH-NU-RF, I use training as well as test examples without any class labels, but not any other source of unlabeled examples unless specified otherwise.

5.3.1 Dataset Details

For evaluation, I considered the task of biomedical relation extraction using four public datasets, AIMed, BioInfer, PubMed45, BioNLP, as done for KLSH-RF in the previous chapter. While PubMed45, BioNLP datasets represent more challenging aspects of the task, AIMed, BioInfer datasets have relevance because

Models	(AIMed, BioInfer)	(BioInfer, AIMed)
SVM_1 (Airola08)	0.25	0.44
SVM_2 (Airola08)	0.47	0.47
SVM (Miwa09)	0.53	0.50
SVM (Tikk et al., 2010)	0.41 (0.67, 0.29)	0.42 (0.27, 0.87)
CNN (Nguyen15)	0.37	0.45
Bi-LSTM (Kavuluru et al., 2017)	0.30	0.47
CNN (Peng and Lu, 2017)	0.48 (0.40, 0.61)	0.50 (0.40, 0.66)
RNN (Hsieh et al., 2017)	0.49	0.51
CNN-RevGrad (Ganin et al., 2016)	0.43	0.47
Bi-LSTM-RevGrad (Ganin et al., 2016)	0.40	0.46
Adv-CNN (Rios et al., 2018)	0.54	0.49
Adv-Bi-LSTM (Rios et al., 2018)	**0.57**	0.49
KLSH-kNN (Garg et al., 2019)	0.51 (0.41, 0.68)	0.51 (0.38, 0.80)
KLSH-RF (Garg et al., 2019)	**0.57** (0.46, 0.75)	0.54 (0.37, 0.95)
SSL-VAE	0.50 (0.38, 0.72)	0.46 (0.39, 0.57)
KLSH-NU-RF	**0.57** (0.44, 0.81)	**0.57** (0.44, 0.81)
NLSH-NU-RF	0.56 (0.43, 0.80)	0.55 (0.40, 0.84)

Table 5.1: I show results from cross-corpus evaluation for (train, test) pairs of PPI datasets, AIMed and BioInfer. For each model, I report F1 score, and if available, precision, recall scores are also shown in brackets. For the adversarial neural models by (Ganin et al., 2016), evaluation on the datasets was provided by (Rios et al., 2018).

cross corpus evaluations been performed on these two datasets, using kernel as well as neural models including the ones doing (semi-supervised) adversarial training of neural networks using the knowledge of examples in a test set.

For AIMed and BioInfer, cross-corpus evaluations have been performed in many previous works (Airola et al., 2008; Tikk et al., 2010; Peng and Lu, 2017; Hsieh et al., 2017; Rios et al., 2018; Garg et al., 2019). These datasets have annotations on pairs of interacting proteins (PPI) in a sentence while ignoring the interaction type. Following these previous works, for a pair of proteins mentioned in a text sentence from a training or a test set, I obtain an undirected shortest path between the pair in a Stanford dependency parse of the sentence, for which a hashcode representation (feature vector) is obtained.

PubMed45 and BioNLP datasets have been used for extensive evaluations in recent works (Garg et al., 2019, 2018; Rao et al., 2017; Garg et al., 2016). These two datasets consider a relatively more difficult task of inferring interaction between two or more bio-entities mentioned in a sentence, along with the inference of their interaction-roles, and the type of interaction from an unrestricted list. As in the previous works, I use abstract meaning representation (AMR) to obtain shortest path-based structural features (Banarescu et al., 2013; Pust et al., 2015a), which are mapped to hashcodes in my model. PubMed45 dataset has 11 subsets for evaluation, with evaluation performed for each of the subsets as a test set leaving the rest for training. For BioNLP dataset (Kim et al., 2009, 2011; Nédellec et al., 2013), the training set contain annotations from years 2009, 2011, 2013, and the test set contains development set from year 2013. Overall, for a fair comparison of the proposed models w.r.t. the ones introduced in the previous chapter (Garg et al., 2019), I keep same experimental setup as followed in there, for all the four datasets, so as to avoid any bias due to engineering aspects; evaluation metrics for the relation extraction task are, f1 score, precision, recall.

5.3.2 Parameter settings

I use path kernels with word vectors & kernel parameter settings as in the previous works (Garg et al., 2019, 2016). From a preliminary tuning, I set the number of hash functions, $H = 100$, and the number of decision

Models	PubMed45	BioNLP
SVM (Garg et al., 2016)	0.45 ± 0.25 (0.58, 0.43)	0.46 (0.35, 0.67)
LSTM (Rao et al., 2017)	N.A.	0.46 (0.51, 0.44)
LSTM (Garg et al., 2019)	0.30 ± 0.21 (0.38, 0.28)	0.59 (0.89, 0.44)
Bi-LSTM (Garg et al., 2019)	0.46 ± 0.26 (0.59, 0.43)	0.55 (0.92, 0.39)
LSTM-CNN (Garg et al., 2019)	0.50 ± 0.27 (0.55, 0.50)	0.60 (0.77, 0.49)
CNN (Garg et al., 2019)	0.51 ± 0.28 (0.46, 0.46)	0.60 (0.80, 0.48)
KLSH-kNN (Garg et al., 2019)	0.46 ± 0.21 (0.44, 0.53)	0.60 (0.63, 0.57)
KLSH-RF (Garg et al., 2019)	0.57 ± 0.25 (0.63, 0.55)	0.63 (0.78, 0.53)
SSL-VAE	0.40 ± 0.16 (0.33, 0.69)	0.48 (0.43, 0.56)
KLSH-NU-RF	$\mathbf{0.61 \pm 0.23}$ (**0.61, 0.62**)	$\mathbf{0.67}$ (**0.73, 0.61**)
NLSH-NU-RF	0.59 ± 0.23 (0.57, 0.64)	0.63 (0.71, 0.56)

Table 5.2: Evaluation results for PubMed45 and BioNLP datasets. I report F1 score (mean $\pm$ standard deviation), and mean-precision & mean-recall numbers in brackets. For BioNLP, standard deviation numbers are not provided as there is one fixed test subset.

trees in a Random Forest classifier, $R = 100$; these parameters are not sensitive, requiring minimal tuning. For any other parameters which may require fine-grained tuning, I use 10% of training examples, selected randomly, for validation. Within kernel locality sensitive hashing, I choose between Random Maximum Margin (RMM) and Random k-Nearest Neighbors (RkNN) techniques, and for neural locality sensitive hashing, I use a simple 2-layer LSTM model with 8 units per layer. In my nearly unsupervised learning framework, I use subsets of the hash functions, of size 10, to obtain clusters ($\zeta = 10$). I employ 8 cores on an i7 processor, with 32GB memory, for all the computations.

5.3.3 Experiment Results

In summary, the proposed model KLSH-NU-RF that is learned by nearly unsupervised optimization of hashcode representations which are fed as feature vectors into a supervised random forest classifier, significantly outperforms its purely supervised counterpart, KLSH-RF, and also other semi-supervised models, such as, adversarial neural networks, variational autoencoders (Rios et al., 2018; Ganin et al., 2016; Kingma et al., 2014).

In reference to Table 5.1, I first discuss results for the evaluation setting of using AIMed dataset as a test set, and BioInfer as a training set. We observe that our model, KLSH-NU-RF, obtains F1 score, 3 pts higher w.r.t. the most recent baseline, KLSH-RF. In comparison to the recent adversarial neural network approaches, CNN-RevGrad, Bi-LSTM-RevGrad, Adv-CNN, Adv-Bi-LSTM, which are learned in a semi-supervised (tranductive) manner just like our model, we gain 8-11 pts in F1 score. On the other hand, when evaluating on BioInfer dataset as a test set and AIMed as a training set, my model is in tie w.r.t. the adversarial neural model, Adv-Bi-LSTM, though outperforming the other three adversarial models by large margins in F1 score. In comparison to KLSH-RF, we retain same F1 score, while gaining in recall by 6 pts at the cost of losing 2 pts in precision.

For PubMed and BioNLP datasets, there is no prior evaluation of semi-supervised models. Nevertheless, in Table 5.2, we see that my model significantly outperforms the most recent baseline, KLSH-RF, gaining F1 score by 4 pts for both the datasets. These two datasets have high importance to gauge practical relevance of a model for the task of biomedical relation extraction.

In reference to Table 5.1 and 5.2, I also evaluated semi-supervised variational autoencoders (Kingma et al., 2014), for all the four datasets. Despite extensive tuning of these neural models, the F1 scores obtained are significantly low w.r.t. our semi-supervised approach.

The proposed approach is easily extensible for other modeling aspects introduced in the previous chapters, such as non-stationary kernel functions, document level inference, joint use of semantic & syntactic

parses (Garg et al., 2019, 2016), though I have focused only upon analyzing improvements from the principled semi-supervised extension of the technique proposed in the previous chapter that had already been shown to be successful for the task.

It is also noteworthy that, w.r.t. the most recent baseline for the four datasets used for the evaluation above, KLSH-RF, the proposed model varies only in the sense that hashcode representations are learned in a semi-supervised manner not only using examples from a training set but also a test set (discounting the class labels). As for building the final classifier, an RF, both approaches are purely supervised using only the hashcode representations of training examples along with their class labels, while completely ignoring the test set. Therefore it is quite interesting (and perhaps surprising) to observe such drastic improvements in F1 score w.r.t. the baseline, even more so considering the fact that the semi-supervised hashcode learning framework is *nearly unsupervised*, using only the knowledge of which set an example belongs to, a training or a test set, while completely ignoring the actual class labels of examples. Further, note that the number of hash functions used in the previous chapter is 1000 whereas, here, I use only 100 hash functions. Compute time is same as for that model.

Neural Hashing

In reference to Table 5.1 and 5.2, I also evaluate NLSH-NU-RF (neural locality sensitive hashcodes from nearly unsupervised learning), and find it to be performing a little worse than its kernel based counterpart, KLSH-NU-RF, in terms of F1 scores, though it is highly competitive w.r.t. all the other baselines. See Figure 5.4 for a direct contrast between the kernel hashing vs neural hashing, within the proposed semi-supervised framework, across all the four datasets; here, we also observe that neural hashing is significantly superior w.r.t. the best of the neural baselines.

Transductive vs Inductive

In the discussed experiments, I considered transductive settings for learning the proposed models, KLSH-NU-RF & NLSH-NU-RF. For these models, the proposed approach of learning nearly unsupervised hashcode representations is equally applicable for inductive settings. Specifically, for additional

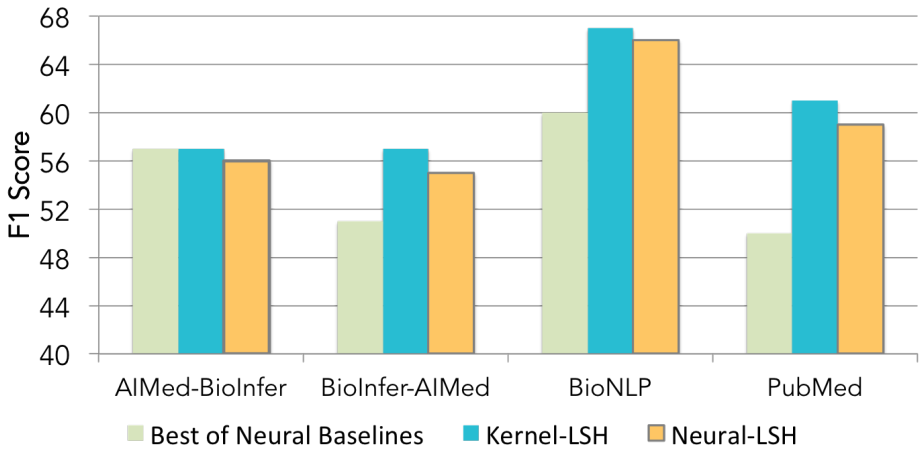


Figure 5.4: Comparison of neural hashing w.r.t. kernel hashing, and the best of neural baselines.

experimental results presented in the following, I consider a random subset of a training dataset itself as a pseudo-test set; 25% of training examples are assigned value 0 for the indicator variable x, and the rest of training examples are assigned value 1; using only these examples as input to Algorithm 4, and not a real test set, I learn hash functions, which are then applied to the test set as well to obtain the hashcode representations for test examples. Since the pseudo-test set is taken as a random subset of a training set, the learned hashcode representations are purely unsupervised. It is also worth noting that, in inductive settings, KLSH-NU-RF model is trained from information lesser than the baseline model, KLSH-RF. In Figure 5.5, I compare results from all the three choices for learning hashcode representations, supervised, transductive, and inductive.[1] In the figure, we observe that both inductive and transductive settings are more favorable w.r.t. the baseline. F1 scores obtained from the inductive setting are on a par with the transductive settings, even though the former one require no information outside a training set unlike the latter one.

[1]Of course, a Random Forest is trained in a supervised manner in all the three settings, and it is only the settings for learning hashcodes which differ.

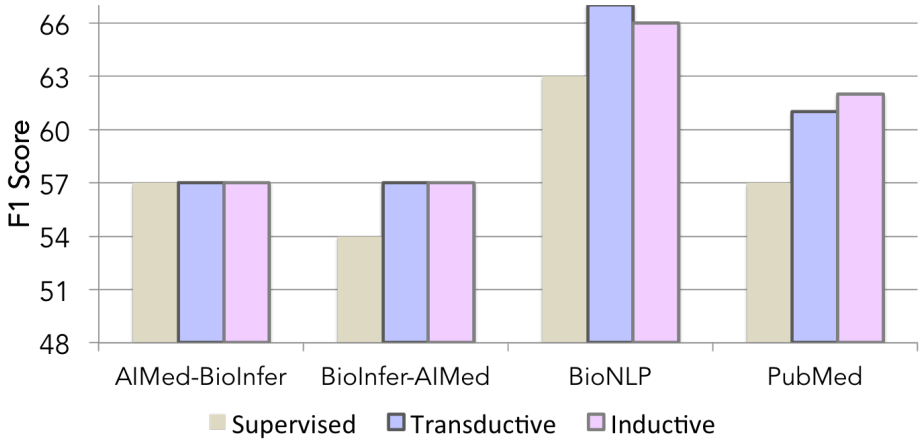


Figure 5.5: Comparison of KLSH-RF across the three kinds of learning settings, supervised, transductive, and inductive.

5.4 Chapter Summary

In this chapter, I proposed a semi-supervised framework for learning of kernelized locality sensitive hashcode representations, a recent technique introduced in the previous chapter of this thesis, that was originally supervised, which has shown state-of-the-art results for the difficult task of biomedical relation extraction on four public datasets. Within the proposed semi-supervised framework, I use the knowledge of test examples, or a random subset of training examples as pseudo-test examples, for fine-grained optimization of hash functions so as to obtain hashcode representations generalizing across training & test sets. The experiment results show significant improvements in accuracy numbers w.r.t. the supervised hashcodes representations approach, as well as semi-supervised neural network models based on adversarial learning or variational autoencoding, for the task of bio-medical relation extraction across all the four datasets. Further, I demonstrated that the generic learning framework can be used for neural networks based locality sensitive hashcode representations, obtaining competitive accuracy numbers w.r.t. all the other baselines.

Chapter 6

Related Work

In the previous chapters, I discussed an extensive list of related works as per the context of particular contributions introduced in each chapter. Here, I summarize on more general related works applicable to this thesis.

Relation Extraction

There have been different lines of work for extracting information on protein interactions. Pattern-matching based systems (either manual or semi-automated) usually yield high precision but low recall (Hunter et al., 2008; Krallinger et al., 2008; Hakenberg et al., 2008; Hahn and Surdeanu, 2015). Kernel-based methods based on various convolution kernels have also been developed for the extraction task (Chang et al., 2016; Tikk et al., 2010; Miwa et al., 2009; Airola et al., 2008; Mooney and Bunescu, 2005). Some approaches work on string rather than parses (Mooney and Bunescu, 2005). Many neural networks based classification models have also been applied for bio-medical relation extraction (Peng and Lu, 2017; Hsieh et al., 2017; Rao et al., 2017; Kavuluru et al., 2017). In (Hsieh et al., 2017), it is shown through cross corpus evaluations that kernel methods are more robust in domain adaptation scenarios compared to the neural language models. Recently, for domain adaptation, neural network models have been explored, such as the ones doing adversarial learning using the knowledge of examples from a test set (Rios et al., 2018; Ganin et al., 2016), or semi-supervised variational autoencoders (Zhang and Lu, 2019; Kingma et al.,

2014). Some recent works used distant supervision to obtain a large data set of protein-protein pairs for their experiments (Mallory et al., 2015; Rao et al., 2017).

Outside the biomedical domain, SemEval-2010 Task 8 (Hendrickx et al., 2009) is one of the popular relation extraction task (Socher et al., 2012; Li et al., 2015; Hashimoto et al., 2015; Xu et al., 2015a,b; Santos et al., 2015; Miwa and Bansal, 2016) wherein accuracy numbers are relatively higher as the relations between entities span across short text, making the task simpler. Some other relation extraction tasks in general domain are more challenging, for which novel algorithms have been proposed based upon concepts such as distance supervision, joint extraction of entities and relations (Ren et al., 2017; Qin et al., 2018; Yuan et al., 2018; Luan et al., 2018; Lin et al., 2019); these contributions are complementary to the ones proposed in this thesis.

Kernel Approximations

Low rank approximations reduce the training/inference cost of a kernel-based classifier, but not the the cost of computing kernels (Williams and Seeger, 2001; Schwaighofer and Tresp, 2002; Lawrence, 2007; Hoang et al., 2015). Also, caching reduces kernel cost, if high redundancies in the substructures (Severyn and Moschitti, 2012). A large body of literature on feature selection is not directly applicable to kernels, but rather only to explicit features (Yu and Liu, 2004; Brown et al., 2012). For reducing kernel cost in inference, *though not in learning*, approximations exist for the subselection of data (Tibshirani, 1996; Das and Kempe, 2011). For k-NN graphs, most of the approximations, including many variants of locality-sensitive hashing (LSH), have been built for explicit features space (Indyk and Motwani, 1998; Heo et al., 2012). Recently, LSH was extended for kernels (Kulis and Grauman, 2012; Raginsky and Lazebnik, 2009). There is another kernel based hashing model that employs a Support Vector Machine to obtain a random maximum margin boundary in the kernel implied feature space for generating each of the hash code bits (Joly and Buisson, 2011).

There are also other scalable techniques in the literature for approximation of a kernel function (Moschitti, 2006; Rahimi and Recht, 2008; Pighin and Moschitti, 2009; Zanzotto and Dell'Arciprete, 2012;

Severyn and Moschitti, 2013; Felix et al., 2016); computational cost reductions from these works are not significant, and not directly applicable for convolution kernels in NLP.

Note that these approaches are only used as computationally efficient approximations of the traditional, computationally-expensive kernel-based classifiers; unlike those approaches, my method is not only computationally more efficient but also yields considerable accuracy improvements.

Structural Features for Relation Extraction

The above mentioned works on relation extraction either rely on text or its shallow parses, none using semantic parsing for the extraction task. In this thesis, I proposed to use Abstract Meaning Representations as semantic parses of text for obtaining semantic structured features as input to the kernel based models (Banarescu et al., 2012, 2013; Pust et al., 2015a; Wang et al., 2015a). Since semantic parsing is an open problem, the semantic features can be highly noisy especially for biomedical domain (Wang et al., 2015a; Vanderwende et al., 2015; Werling et al., 2015; Zhou et al., 2016; Buys and Blunsom, 2017; Konstas et al., 2017). Fortunately, the proposed locality sensitive hashcodes based models in this thesis have the advantage of being robust to such noise in data. In this thesis, for principled empirical comparisons of the techniques proposed across the different chapters, I employed AMR parses of bio-text obtained from the model that was originally released in (Pust et al., 2015a). One could improve the accuracy numbers for my proposed models further by using the most advanced AMR parsers from the present/future works. Though, one practical challenge therein is that it is not straightforward to extend a general domain AMR parser for biomedical domain, partly because for the biomedical domain, an AMR parser relies upon a model for biomedical named entity recognition which is itself an open problem. Recent works for unsupervised neural network learning, such as BERT (Devlin et al., 2018; Radford et al., 2019), may be used to improve semantic/syntactic parsing or even text based features which are input to my proposed models for relation extraction.

Chapter 7

Conclusions

In this thesis, I studied the difficult task of extracting information about bio-molecular interactions from the semantic or syntactic parsing of scientific papers.

One of the primary contributions of this thesis is the idea of using a well-known technique, kernelized locality-sensitive hashing, in order to derive feature vectors from natural language structures. More specifically, I proposed to use random subspaces of hashcodes for building a random forest of decision trees. I find this methodology particularly suitable for modeling natural language structures in settings where there are significant mismatches between the training and the test conditions. Moreover I optimize a hashing model in the context of classification performed using any model, by maximizing an approximation of the mutual information between the hashcodes (feature vectors) and the class labels. I also proposed a novel nonstationary extension of convolution kernels by introducing a data-driven parameterization of the kernel similarity function. The extended kernels have better flexibility and expressibility of language representations, compared to conventional convolution kernels used in natural language tasks. Experiments on multiple bio-medical relation extraction datasets demonstrate the considerable advantages of the proposed approach w.r.t. the state-of-the-art, in terms of significant gain in accuracy numbers as well as scalability.

Finally, I proposed a semi-supervised framework for the learning of kernelized locality sensitive hashcode representations. Within the proposed semi-supervised framework, one uses the knowledge of test

examples for fine-grained optimization of hash functions so as to obtain hashcode representations generalizing across training & test sets. The experiment results show significant improvements in accuracy numbers w.r.t. the proposed supervised hashcode representations approach, as well as semi-supervised neural network models, for the same task of bio-medical relation extraction.

In the future, I plan to explore the applicability of this work for other NLP problems, such as dialog modeling, unsupervised text generation, text classification, etc.

Bibliography

Airola, A., S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski
2008. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC Bioinformatics*.

Alemi, A., I. Fischer, J. Dillon, and K. Murphy
2017. Deep variational information bottleneck. In *Proceedings of the International Conference on Learning Representations*.

Andreas, J., A. Vlachos, and S. Clark
2013. Semantic parsing as machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Assael, J.-A. M., Z. Wang, B. Shahriari, and N. de Freitas
2014. Heteroscedastic treed bayesian optimisation. *arXiv preprint arXiv:1410.7172*.

Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider
2012. Abstract meaning representation (AMR) 1.0 specification. In *Parsing on Freebase from Question-Answer Pairs. In Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider
2013. Abstract meaning representation for sembanking. In *Proceedings of the Linguistic Annotation Workshop and Interoperability with Discourse*.

Barber, D. and F. Agakov
2003. The IM algorithm: a variational approach to information maximization. In *Proceedings of the Neural Information Processing Systems Conference*.

Bawa, M., T. Condie, and P. Ganesan
2005. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the International World Wide Web Conference*.

Beck, D., T. Cohn, C. Hardmeier, and L. Specia
2015. Learning structural kernels for natural language processing. *Transactions of Annual Meeting of the Association for Computational Linguistics*.

Belghazi, M. I., A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm
2018. Mutual information neural estimation. In *Proceedings of the International Conference on Machine Learning*.

Biau, G., F. Cérou, and A. Guyader
2010. On the rate of convergence of the bagged nearest neighbor estimate. *Journal of Machine Learning Research*.

Bordes, A., X. Glorot, J. Weston, and Y. Bengio
2014. A semantic matching energy function for learning with multi-relational data. *Machine Learning*.

Borgwardt, K. M., A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf, and A. J. Smola
2006. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*.

Bottou, L.
2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the International Conference on Computational Statistics*.

Breiman, L.
1996. Bagging predictors. *Machine learning*.

Breiman, L.
2001. Random forests. *Machine learning*.

Brown, G., A. Pocock, M.-J. Zhao, and M. Luján
2012. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *Journal of Machine Learning Research*.

Bunescu, R., R. Ge, R. J. Kate, E. M. Marcotte, R. J. Mooney, A. K. Ramani, and Y. W. Wong
2005. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial Intelligence in Medicine*.

Bunescu, R., R. Mooney, A. Ramani, and E. Marcotte
2006. Integrating co-occurrence statistics with information extraction for robust retrieval of protein interactions from medline. In *Proceedings of the Workshop on Linking Natural Language Processing and Biology: Towards Deeper Biological Literature Analysis*.

Buys, J. and P. Blunsom
2017. Robust incremental neural semantic graph parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Chalk, M., O. Marre, and G. Tkacik
2016. Relevant sparse codes with variational information bottleneck. In *Proceedings of the Neural Information Processing Systems Conference*.

Chang, C.-C. and C.-J. Lin
2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*.

Chang, Y.-C., C.-H. Chu, Y.-C. Su, C. C. Chen, and W.-L. Hsu
2016. PIPE: a protein–protein interaction passage extraction module for biocreative challenge. *Database*.

Chen, D. and C. D. Manning
2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Clark, S.
2014. Vector space models of lexical meaning. *Handbook of Contemporary Semantics*.

Cohen, P. R.
2015. DARPA's big mechanism program. *Physical biology*.

Collins, M. and N. Duffy
2001. Convolution kernels for natural language. In *Proceedings of the Neural Information Processing Systems Conference*.

Collobert, R. and J. Weston
2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the International Conference on Machine Learning*.

Cortes, C. and V. Vapnik
1995. Support vector machine. *Machine learning*.

Cover, T. M. and J. A. Thomas
2006. *Elements of information theory*.

Culotta, A. and J. Sorensen
2004. Dependency tree kernels for relation extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Das, A. and D. Kempe
2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the International Conference on Machine Learning*.

Demir, E., M. P. Cary, S. Paley, K. Fukuda, C. Lemer, I. Vastrik, G. Wu, P. D'Eustachio, C. Schaefer, J. Luciano, et al.
2010. The BioPAX community standard for pathway data sharing. *Nature Biotechnology*.

Demmel, J. W.
1997. *Applied numerical linear algebra*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova
2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Downward, J.
2003. Targeting RAS signalling pathways in cancer therapy. *Nature Reviews Cancer*.

Felix, X. Y., A. T. Suresh, K. M. Choromanski, D. N. Holtmann-Rice, and S. Kumar
2016. Orthogonal random features. In *Proceedings of the Neural Information Processing Systems Conference*.

Filice, S., G. Da San Martino, and A. Moschitti
2015. Structural representations for learning relations between pairs of texts. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith
2014. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky
2016. Domain-adversarial training of neural networks. *Journal of Machine Learning*.

Gao, S., G. Ver Steeg, and A. Galstyan
2015. Efficient estimation of mutual information for strongly dependent variables. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Gao, S., G. Ver Steeg, and A. Galstyan
2016. Variational information maximization for feature selection. In *Proceedings of the Neural Information Processing Systems Conference*.

Garg, S., A. Galstyan, U. Hermjakob, and D. Marcu
2016. Extracting biomolecular interactions using semantic parsing of biomedical text. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Garg, S., A. Galstyan, G. V. Steeg, I. Rish, G. Cecchi, and S. Gao
2019. Kernelized hashcode representations for relation extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Garg, S., G. V. Steeg, and A. Galstyan
2018. Stochastic learning of nonstationary kernels for natural language modeling. *arXiv preprint arXiv:1801.03911*.

Genton, M. G.
2001. Classes of kernels for machine learning: a statistics perspective. *Journal of Machine Learning Research*.

Gers, F. A., J. Schmidhuber, and F. Cummins
1999. Learning to forget: Continual prediction with LSTM.

Gneiting, T.
2002. Compactly supported correlation functions. *Journal of Multivariate Analysis*.

Goyal, A., H. Daumé III, and R. Guerra
2012. Fast large-scale approximate graph construction for NLP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Grauman, K. and R. Fergus
2013. Learning binary hash codes for large-scale image search. *Machine Learning for Computer Vision*.

Gretton, A., K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola
2012. A kernel two-sample test. *Journal of Machine Learning*.

Hahn, M. A. V.-E. G. and P. T. H. M. Surdeanu
2015. A domain-independent rule-based framework for event extraction. In *Proceedings of ACL-IJCNLP*.

Hakenberg, J., C. Plake, L. Royer, H. Strobelt, U. Leser, and M. Schroeder
2008. Gene mention normalization and interaction extraction with context models and sentence motifs. *Genome Biology*.

Han, Y., J. Jiao, and T. Weissman
2015. Adaptive estimation of shannon entropy. In *Proceedings of IEEE International Symposium on Information Theory*.

Hashimoto, K., P. Stenetorp, M. Miwa, and Y. Tsuruoka
2015. Task-oriented learning of word embeddings for semantic relation classification. In *Proceedings of the Conference on Computational Language Learning*.

Hastings, W. K.
1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika*.

Haussler, D.
1999. Convolution kernels on discrete structures. Technical report.

Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz
2009. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*.

Heo, J.-P., Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon
2012. Spherical hashing. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Higdon, D.
1998. A process-convolution approach to modelling temperatures in the north atlantic ocean. *Environmental and Ecological Statistics*.

Ho, T. K.
1995. Random decision forests. In *Proceedings of the International Conference on Document Analysis and Recognition*.

Ho, T. K.
1998. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*.

Hoang, T. N., Q. M. Hoang, and K. H. Low
2015. A unifying framework of anytime sparse gaussian process regression models with stochastic variational inference for big data. In *Proceedings of the International Conference on Machine Learning*.

Hochreiter, S. and J. Schmidhuber
1997. Long short-term memory. *Neural computation*.

Hovy, D., S. Srivastava, S. K. Jauhar, M. Sachan, K. Goyal, H. Li, W. Sanders, and E. Hovy
2013. Identifying metaphorical word use with tree kernels. In *Proceedings of the Workshop on Metaphor in NLP*.

Hsieh, Y.-L., Y.-C. Chang, N.-W. Chang, and W.-L. Hsu
2017. Identifying protein-protein interactions in biomedical literature using recurrent neural networks with long short-term memory. In *Proceedings of the International Joint Conference on Natural Language Processing*.

Hunter, L., Z. Lu, J. Firby, W. A. Baumgartner, H. L. Johnson, P. V. Ogren, and K. B. Cohen
2008. OpenDMAP: an open source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression. *BMC Bioinformatics*.

Indyk, P. and R. Motwani
1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of Annual ACM Symposium on the Theory of Computing*.

Joly, A. and O. Buisson
2011. Random maximum margin hashing. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Kalchbrenner, N., E. Grefenstette, and P. Blunsom
2014. A convolutional neural network for modelling sentences. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Kavuluru, R., A. Rios, and T. Tran
2017. Extracting drug-drug interactions with word and character-level recurrent neural networks. In *IEEE International Conference on Healthcare Informatics*.

Kim, B. and J. Pineau
2013. Maximum mean discrepancy imitation learning. In *Proceedings of Robotic: Science and Systems*.

Kim, J.-D., T. Ohta, S. Pyysalo, Y. Kano, and J. Tsujii
2009. Overview of BioNLP'09 shared task on event extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing: Shared Task*.

Kim, J.-D., S. Pyysalo, T. Ohta, R. Bossy, N. Nguyen, and J. Tsujii
2011. Overview of BioNLP shared task 2011. In *Proceedings of the BioNLP Shared Task Workshop*.

Kingma, D. P., S. Mohamed, D. Jimenez Rezende, and M. Welling
2014. Semi-supervised learning with deep generative models. In *Proceedings of the Neural Information Processing Systems Conference*.

Konstas, I., S. Iyer, M. Yatskar, Y. Choi, and L. Zettlemoyer
2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Krallinger, M., F. Leitner, C. Rodriguez-Penagos, and A. Valencia
2008. Overview of the protein-protein interaction annotation extraction task of BioCreative II. *Genome Biology*.

Kraskov, A., H. Stögbauer, and P. Grassberger
2004. Estimating mutual information. *Physical Review E*.

Krause, A., A. Singh, and C. Guestrin
2008. Near-optimal sensor placements in Gaussian processes: theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*.

Kulis, B. and K. Grauman
2009. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the International Conference on Computer Vision*.

Kulis, B. and K. Grauman
2012. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kumar, A., O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher
2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the International Conference on Machine Learning*.

Lawrence, N. D.
2007. Learning for larger datasets with the gaussian process latent variable model. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Le, Q. V., A. J. Smola, and S. Canu
2005. Heteroscedastic gaussian process regression. In *Proceedings of the International Conference on Machine Learning*.

Li, H., W. Liu, and H. Ji
2014. Two-stage hashing for fast document retrieval. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Li, J., M.-T. Luong, D. Jurafsky, and E. Hovy
2015. When are tree structures necessary for deep learning of representations? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Li, P., A. Shrivastava, J. L. Moore, and A. C. König
2011. Hashing algorithms for large-scale learning. In *Proceedings of the Neural Information Processing Systems Conference*.

Lin, H., J. Yan, M. Qu, and X. Ren
2019. Learning dual retrieval module for semi-supervised relation extraction. In *Proceedings of the International Conference on World Wide Web*.

Liu, H., R. Wang, S. Shan, and X. Chen
2016. Deep supervised hashing for fast image retrieval. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Luan, Y., L. He, M. Ostendorf, and H. Hajishirzi
2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Luong, M.-T., H. Pham, and C. D. Manning
2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Mallory, E. K., C. Zhang, C. Ré, and R. B. Altman
2015. Large-scale extraction of gene interactions from full-text literature using DeepDive. *Bioinformatics*.

McDonald, R., F. Pereira, S. Kulick, S. Winters, Y. Jin, and P. White
2005. Simple algorithms for complex relation extraction with applications to biomedical IE. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Mehdad, Y., A. Moschitti, and F. M. Zanzotto
2010. Syntactic/semantic structures for textual entailment recognition. In *Human Language Technologies: NAACL*.

Mikolov, T., A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato
2014. Learning longer memory in recurrent neural networks. *ICLR Workshop Track*.

Mikolov, T., M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur
2010. Recurrent neural network based language model. In *Proceedings of Interspeech*.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean
2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Neural Information Processing Systems Conference*.

Miwa, M. and M. Bansal
2016. End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Miwa, M., R. Sætre, Y. Miyao, and J. Tsujii
2009. Protein–protein interaction extraction by leveraging multiple kernels and parsers. *International Journal of Medical Informatics*.

Mooney, R. J. and R. C. Bunescu
2005. Subsequence kernels for relation extraction. In *Proceedings of the Neural Information Processing Systems Conference*.

Moschitti, A.
2006. Making tree kernels practical for natural language learning. In *Proceedings of European Chapter of the Association for Computational Linguistics*.

Mu, Y., G. Hua, W. Fan, and S.-F. Chang
2014. Hash-SVM: Scalable kernel machines for large-scale visual classification. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

Nédellec, C., R. Bossy, J.-D. Kim, J.-J. Kim, T. Ohta, S. Pyysalo, and P. Zweigenbaum
2013. Overview of BioNLP shared task 2013. In *Proceedings of the BioNLP Shared Task Workshop*.

Niethammer, W., J. De Pillis, and R. Varga
1984. Convergence of block iterative methods applied to sparse least-squares problems. *Linear Algebra and its Applications*.

Paccanaro, A. and G. E. Hinton
2000. Learning distributed representations by mapping concepts and relations into a linear space. In *Proceedings of the International Conference on Machine Learning*.

Paciorek, C. J. and M. J. Schervish
2003. Nonstationary covariance functions for gaussian process regression. In *Proceedings of the Neural Information Processing Systems Conference*.

Pan, S. J., J. T. Kwok, and Q. Yang
2008. Transfer learning via dimensionality reduction. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Peng, Y. and Z. Lu
2017. Deep learning for extracting protein-protein interactions from biomedical literature. In *Proceedings of BioNLP Workshop*.

Pighin, D. and A. Moschitti
2009. Efficient linearization of tree kernel functions. In *Proceedings of The SIGNLL Conference on Computational Natural Language Learning*.

Pust, M., U. Hermjakob, K. Knight, D. Marcu, and J. May
2015a. Parsing english into abstract meaning representation using syntax-based machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Pust, M., U. Hermjakob, K. Knight, D. Marcu, and J. May
2015b. Using syntax-based machine translation to parse english into abstract meaning representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Qian, L. and G. Zhou
2012. Tree kernel-based protein–protein interaction extraction from biomedical literature. *Journal of Biomedical Informatics*.

Qin, P., X. Weiran, and W. Y. Wang
2018. Robust distant supervision relation extraction via deep reinforcement learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever
2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

Raginsky, M. and S. Lazebnik
2009. Locality-sensitive binary codes from shift-invariant kernels. In *Proceedings of the Neural Information Processing Systems Conference*.

Rahimi, A. and B. Recht
2008. Random features for large-scale kernel machines. In *Proceedings of the Neural Information Processing Systems Conference*.

Rao, S., D. Marcu, K. Knight, and H. D. III
2017. Biomedical event extraction using abstract meaning representation. In *Workshop on Biomedical Natural Language Processing*.

Rasmussen, C. E. and Z. Ghahramani
2002. Infinite mixtures of gaussian process experts. *In Proceedings of the Neural Information Processing Systems Conference*.

Rasmussen, C. E. and C. K. I. Williams
2006. *Gaussian processes for machine learning*.

Ren, X., Z. Wu, W. He, M. Qu, C. R. Voss, H. Ji, T. F. Abdelzaher, and J. Han
2017. Cotype: Joint extraction of typed entities and relations with knowledge bases. In *Proceedings of the International Conference on World Wide Web*.

Rios, A., R. Kavuluru, and Z. Lu
2018. Generalizing biomedical relation classification with neural adversarial domain adaptation. *Bioinformatics*.

Rzhetsky, A.
2016. The big mechanism program: Changing how science is done. In *Proceedings of DAMDID/RCDL*.

Sampson, P. D. and P. Guttorp
1992. Nonparametric estimation of nonstationary spatial covariance structure. *Journal of the American Statistical Association*.

Santos, C. N. d., B. Xiang, and B. Zhou
2015. Classifying relations by ranking with convolutional neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Schölkopf, B. and A. J. Smola
2002. *Learning with kernels: support vector machines, regularization, optimization, and beyond*.

Schwaighofer, A. and V. Tresp
2002. Transductive and inductive methods for approximate gaussian process regression. In *Advances in Neural Information Processing Systems*.

Severyn, A. and A. Moschitti
2012. Fast support vector machines for convolution tree kernels. *Data Mining and Knowledge Discovery*.

Severyn, A. and A. Moschitti
2013. Fast linearization of tree kernels over large-scale data. In *Proceedings of the International Joint Conferences on Artificial Intelligence*.

Shi, Q., J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan
2009. Hash kernels for structured data. *Journal of Machine Learning Research*.

Shi, X. and K. Knight
2017. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Singh, S. and B. Póczos
2014. Generalized exponential concentration inequality for rényi divergence estimation. In *Proceedings of the International Conference on Machine Learning*.

Skounakis, M. and M. Craven
2003. Evidence combination in biomedical natural-language processing. In *Proceedings of BIOKDD*.

Snelson, E., C. E. Rasmussen, and Z. Ghahramani
2003. Warped gaussian processes. In *Proceedings of the Neural Information Processing Systems Conference*.

Socher, R., B. Huval, C. D. Manning, and A. Y. Ng
2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Srivastava, S., D. Hovy, and E. H. Hovy
2013. A walk-based semantically enriched tree kernel over distributed word representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Sundermeyer, M., R. Schlüter, and H. Ney
2012. LSTM neural networks for language modeling. In *Proceedings of Interspeech*.

Sutherland, D. J., L. Xiong, B. Póczos, and J. Schneider
2012. Kernels on sample sets via nonparametric divergence estimates. *arXiv preprint arXiv:1202.0302*.

Sutskever, I., O. Vinyals, and Q. V. Le
2014. Sequence to sequence learning with neural networks. In *Proceedings of the Neural Information Processing Systems Conference*.

Suzuki, J. and H. Isozaki
2006. Sequence and tree kernels with statistical feature mining. In *Proceedings of the Neural Information Processing Systems Conference*.

Suzuki, J., H. Isozaki, and E. Maeda
2004. Convolution kernels with feature selection for natural language processing tasks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Tibshirani, R.
1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society.*

Tikk, D., P. Thomas, P. Palaga, J. Hakenberg, and U. Leser
2010. A comprehensive benchmark of kernel methods to extract protein–protein interactions from literature. *PLoS Computational Biology*.

Tkachenko, M. and H. W. Lauw
2015. A convolution kernel approach to identifying comparisons in text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Tymoshenko, K., D. Bonadiman, and A. Moschitti
2016. Convolutional neural networks vs. convolution kernels: Feature engineering for answer sentence reranking. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Valenzuela-Escárcega, M. A., O. Babur, G. Hahn-Powell, D. Bell, T. Hicks, E. Noriega-Atala, X. Wang, M. Surdeanu, E. Demir, and C. T. Morrison
2017. Large-scale automated reading with reach discovers new cancer driving mechanisms.

Vanderwende, L., A. Menezes, and C. Quirk
2015. An AMR parser for english, french, german, spanish and japanese and a new AMR-annotated corpus. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*.

Ver Steeg, G. and A. Galstyan
2014. Discovering structure in high-dimensional data through correlation explanation. In *Proceedings of the Neural Information Processing Systems Conference*.

Vogelstein, B. and K. W. Kinzler
2004. Cancer genes and the pathways they control. *Nature Medicine*.

Walters-Williams, J. and Y. Li
2009. Estimation of mutual information: A survey. In *Proceedings of the International Conference on Rough Sets and Knowledge Technology*.

Wang, C., N. Xue, and S. Pradhan
2015a. A transition-based algorithm for AMR parsing. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Wang, J., H. T. Shen, J. Song, and J. Ji
2014. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*.

Wang, J., T. Zhang, N. Sebe, H. T. Shen, et al.
2017. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wang, Q., S. R. Kulkarni, and S. Verdú
2009. Divergence estimation for multidimensional densities via-nearest-neighbor distances. *IEEE Transactions on Information Theory*.

Wang, Y., J. Berant, and P. Liang
2015b. Building a semantic parser overnight. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Watanabe, S.
1960. Information theoretical analysis of multivariate correlation. *IBM Journal of Research and Development*.

Weinberger, K. Q., J. Blitzer, and L. Saul
2006. Distance metric learning for large margin nearest neighbor classification. *In Proceedings of the Neural Information Processing Systems Conference*.

Weinberger, K. Q. and L. K. Saul
2009. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*.

Werling, K., G. Angeli, and C. Manning
2015. Robust subgraph generation improves abstract meaning representation parsing. *arXiv preprint arXiv:1506.03139*.

Williams, C. and M. Seeger
2001. Using the Nyström method to speed up kernel machines. In *Proceedings of the Neural Information Processing Systems Conference*.

Wu, Y. and P. Yang
2016. Minimax rates of entropy estimation on large alphabets via best polynomial approximation. *IEEE Transactions on Information Theory*.

Wurzer, D., V. Lavrenko, and M. Osborne
2015. Twitter-scale new event detection via k-term hashing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Xu, K., Y. Feng, S. Huang, and D. Zhao
2015a. Semantic relation classification via convolutional neural networks with simple negative sampling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin
2015b. Classifying relations via long short term memory networks along shortest dependency paths. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Yu, A. W., H. Lee, and Q. V. Le
2017. Learning to skim text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Yu, K., L. Ji, and X. Zhang
2002. Kernel nearest-neighbor algorithm. *Neural Processing Letters*.

Yu, L. and H. Liu
2004. Efficient feature selection via analysis of relevance and redundancy. *The Journal of Machine Learning Research*, 5:1205–1224.

Yuan, Y., L. Liu, S. Tang, Z. Zhang, Y. Zhuang, S. Pu, F. Wu, and X. Ren
2018. Cross-relation cross-bag attention for distantly-supervised relation extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zanzotto, F. and L. Dell'Arciprete
2012. Distributed tree kernels. In *Proceedings of the International Conference on Machine Learning*.

Zelenko, D., C. Aone, and A. Richardella
2003. Kernel methods for relation extraction. *Journal of Machine Learning*.

Zhang, S., H. Jiang, M. Xu, J. Hou, and L.-R. Dai
2015. The fixed-size ordinally-forgetting encoding method for neural network language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Zhang, Y. and Z. Lu
2019. Exploring semi-supervised variational autoencoders for biomedical relation extraction. *Methods*.

Zhao, K., H. Lu, and J. Mei
2014. Locality preserving hashing. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhou, J., F. Xu, H. Uszkoreit, Q. Weiguang, R. Li, and Y. Gu
2016. AMR parsing with an incremental joint model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Zhou, Z.-H.
2012. *Ensemble methods: foundations and algorithms*.