Structured Prediction using Voted Conditional Random Fields

Link Prediction in Knowledge Bases

Adam Dahlgren

Adam Dahlgren
Spring 2017
Degree Project in Computing Science Engineering, 30 ECTS Credits
Supervisor: Johanna Björklund
Examiner: Henrik Björklund
Master of Science Programme in Computing Science and Engineering, 300 ECTS Credits

Abstract

Knowledge bases are useful in the validation of automatically extracted information, and for hypothesis selection during the extraction process. Building knowledge bases is a difficult task and the process is bound to miss facts. Therefore, the existence of facts can be estimated using link prediction, i.e., by solving the structured prediction problem.

It has been shown that combining directly observable features with latent features increases performance. Observable features include, e.g., the presence of another chain of facts leading to the same endpoint. Latent features include, e.g, properties that are not modelled by facts on the form subject-predicate-object, such as being a good actor. Observable graph features are modelled using the Path Ranking Algorithm, and latent features using the bilinear RESCAL model. Voted Conditional Random Fields can be used to combine feature families while taking into account their complexity to minimize the risk of training a poor predictor. We propose a combined model fusing these theories together with a complexity analysis of the feature families used. In addition, two simple feature families are constructed to model neighborhood properties.

The model we propose captures useful features for link prediction, but needs further evaluation to guarantee efficient learning. Finally, suggestions for experiments and other feature families are given.

Acknowledgements

I would like to thank the Foundations of Language Processing research group. Dr. Johanna Björklund for her guidance, feedback and inspiration. Anna Jonsson for her supporting conversations. The rest for all potluck lunches, enlightening discussions and social events.

To my family, for all their support and interest. To my parents, without whom this would have been terribly difficult.

To my colleagues at the department for the things they have taught me, and to all my students for making things interesting.

Thanks to Olle, Niklas and Oskar for their friendship and their company. To Niklas for also being the constant rubber-duck. To everyone whom I have shared have voluntary work with, doing what we love. Finally, to anyone responsible for providing any of all the distractions along the way making everyday life extraordinary.

Contents

1 Introduction

The Web has grown to an almost incomprehensable size, with 2.5 quintillion bytes of data created every day [1]. In 2015, 100 hours of video were uploaded to Youtube every minute [2]. Large parts of this data is unstructured, such as video or plain text. An effort towards structurizing this is the semantic web, where technologies such as RDF extends the linking structure of the Web as a triple connecting two things via a relationship [3, 4]. There is data which is published in a structured format, for example infoboxes on Wikipedia. These can be used as a basis for building an RDF database [5], commonly referred to as a knowledge base. A knowledge base is a structured knowledge representation where relationships between objects represent facts. This is usually ternary relations between entities and predicates [6], e.g. the SPO (subject, predicate, object) triple (*Spock*, *characterIn*, *Star Trek*) denoting that Spock is a character in Star Trek. However, the vast majority of data is unstructured and therefore the need to extract information from, e.g., video and text becomes important. This is one of the challenges in, e.g., Big Data – the task of mining information from multiple (large) data sets. One problem of such information extraction from Web sources is noisy data of poor quality, where missspelled words or missing parts of audio can confuse extractors. This can introduce errors not only for the misspelled word but for the whole sentence. Therefore, it is meaningful to complement natural language processing and other media analysis tools with structural and contextual information, as outlined in [7]. Googles Knowledge Vault is one such example, where information extractors are combined with structural information of known facts to achieve almost a tripling of high confidence facts compared to only relying on the information extractors [8]. Systems that utilize knowledge bases are commonly referred to as knowledge-based systems.

In order to attain structural or contextual information, it is necessary to represent previous knowledge. An example of this kind is the above-mentioned knowledge bases, where previously known facts are stored in RDF. Examples of this include YAGO [9], Freebase [10], DBpedia [5] and Google Knowledge Graph [11]. To evaluate extracted information based on the structural information, it is necessary to perform a link prediction. A link prediction in this context is expressing the likelihood of the relationship between two entities, or calculating the probability that such a relationship exists [6]. The distinction between these two tasks is made since the probability is not always given directly by a model. However, it can usually be calculated as a separate step.

The problem of link prediction belongs to the family of supervised learning problems called structured prediction [12]. Structured prediction overlaps with the problems solved by Relational Machine Learning [13]. Relational Machine Learning, or Statistical Relational Learning, studies the statistical modeling and analysis of objects and their relation to each other. Some of the main tasks of RML include relationship and property predictions, and object clustering based on relationships.

One common critisism of machine learning algorithms is that they usually have a low transparancy as to their inner workings. One goal of this thesis is to give an insight into how these algorithms can be formally analysed to counter this notion.

1.1 Goals and Related Work

The aim of this thesis is to model link prediction in knowledge bases as a VCRF problem. This is summarized in the following goals

Goals	
Goal 1	Provide insight into the theoretical aspects of link prediction.
Goal 2	Evaluate different categories of models used for link prediction.
Goal 3	Model link prediction as a problem solvable with VCRF.

The motivation behind **Goal 3** is to investigate whether the results of Cortes et al.[14] can give similar improvements for other problems. It is important to note that empirical experiments are outside the scope of this study, so **Goal 3** consists in assessing the feasibility of modeling link prediction for VCRF. The proposed model can then be evaluated as future work.

The other two goals are motivated by the results of Cortes et al. in combination with those of Nickel et al. [15] where mixing models with different strengths improve classifier performance. This aligns well with VCRF, as their main result is theoretical tools for mixing different models with regards to their relative complexity.

The reason to study knowledge bases has been thoroughly motivated by semantic web and big data. The focus in this thesis therefore lies on giving a walkthrough of some concrete examples of how knowledge bases can be utilized. Here Google's Knowledge Vault is considered related work as parts of the proposed model is based on their findings.

1.2 Outline

This thesis is outlined as follows:

Chapter 1 introduces the topic of this thesis, motivates why this is interesting and summarizes the goals and related work. Chapter 2 provides preliminary theoretical background necessary to understand structured prediction in general and the VCRF algorithm in particular. Chapter 3 gives a description of how link prediction is performed in knowledge bases, specifically diving into latent and graph features, together with a walkthrough of the VCRF algorithm. Chapter 4 proposes a structured prediction model for link prediction in knowledge bases that is adapted to the VCRF problem formulation. Chapter 4 also gives a brief analysis of the suggested feature family complexity. Chapter 5 discusses problems with the proposed model and the relation between the feature families the model is based on. Chapter 6 concludes briefly and propose topics for further investigation, such as other feature models of interesting.

2 Theory

This chapter contains an overview of some of the theoretical basis for this thesis. It is divided into three main blocks; General structured prediction, learning theory and Conditional Random Fields. The first section gives an introduction to the general formulation of structured prediction problems and how these can be reasoned about. The section on learning theory provides much of the theoretical background necessary to approach the Voted Conditional Random Fields algorithm. The final section on Conditional Random Fields gives an overview of CRF as a concept and how they can be used to perform structured prediction. This will be used as a basis for understanding the VCRF algorithm and how it can be used to perform link prediction.

2.1 Structured Prediction

Structured prediction is a collection of supervised learning techniques dealing with problems where the output contains structure, not only the input. The primary example of structured prediction is that of part-of-speech tagging - the task of annotating words in a sentence with their function (name, verb et c.). A short example of POS tagging is shown in Figure 1.

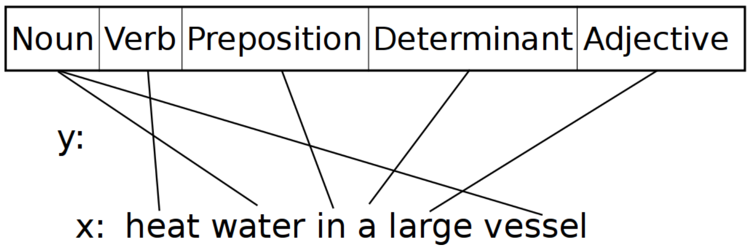

Figure 1: An example of POS tagging as a structured prediction problem. Given a sentence, each word is tagged with its role in a sentence.

Supervised learning in general can be formulated for an input space $\mathcal{X}$ and output space $\mathcal{Y}$ as

Definition 1 (Supervised learning)
Given a sample $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $|S| = N$, a hypothesis function $\boldsymbol{h} : \mathcal{X} \to \mathcal{Y}$ can be trained on S to label input x_i with output y_i. Often $\boldsymbol{h}$ can be formulated using a scoring function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. A label y^* given input x can now be computed with the hypothesis function defined as

$$y^* = \boldsymbol{h}(x) = \arg\max_{y \in \mathcal{Y}} f(x, y) \tag{2.1}$$

The function $\boldsymbol{h}$ is commonly referred to as a classifier*.*

The training points in sample S are drawn i.i.d. from some probability distribution $\mathcal{D}$. It is assumed that when the hypothesis $\boldsymbol{h}$ is used on new data, for example during evaluation, that this data is chosen according to the same distribution $\mathcal{D}$ independently from the other input samples.

The aim of supervised learning is to solve the problem presented in Equation 2.1. Given an input from the input space $\mathcal{X}$ and an output space $\mathcal{Y}$, the task is to find the most compatible y based on some scoring or feature function f. The generalization of structured prediction is that the outputspace can be decomposed into substructures as $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_l$, where $\mathcal{Y}_k$ is the set of possible lables for the substructure k. Other supervised learning problems such as linear or logistic regression and Support Vector Machines outputs scalar values where only the structure of the input is considered.

The input x could be a sentence or an image and the output a part-of-speech tagging or image segmentation. The common denominator is that the input and output space can be arbitrarily complex. Such a multidimensional dataset calls for sophisticated algorithmic approaches. As a result, many approaches for structured prediction algorithms rely heavily on learning theory results.

Performing supervised learning also entails encoding features of x that can be used for learning relations between x and y. In structured prediction, these features are also based on y. These features are encoded in a *feature vector*. In the case of most learning algorithms presented in this thesis, learning then becomes finding optimal weights in a *feature weight vector*. The feature weight vector encodes the importance of each individual feature.

There are many things apart from the intrinsic structures of $\mathcal{X}$ and $\mathcal{Y}$ to consider in order to get a high quality classifier. Many of these are studied in learning theory, such as the distribution from which the sample is chosen contra the true distribution over the space $\mathcal{X} \times \mathcal{Y}$.

Structured prediction in knowledge bases

In the context of knowledge bases, structured prediction can be categorized into three common tasks; Link prediction, entity resolution and link-based clustering [6]. *Entity resolution* is the problem of determine whether two entities refer to the same real world object. In Figure 2, this problem could manifest as an additional fact *(Harrison Ford, starredIn, Star Wars)* where the entity resolution would identify *Ford* and *Harrison Ford* as the same person. *Link-based clustering* is known as community detection in social network sciences. This entails grouping entities based not only on entity features but also their link similarity.

Focus in this thesis lies on the link prediction problem, as this is the most prominent task in aiding, e.g., information extraction. The example in Figure 2 shows why link prediction is important in the context of information extraction. Say that the sentence *Leonard Nimoy, one of the stars in the old Star Trek-series, [...]* is extracted from a web source. Given the knowledge base in Figure 2, it is possible to check the semantic correctness directly against the edge *(Nimoy, starredIn, Star Trek)*. However, the sentence *Harrison Ford, who starred in many of the Star Wars movies, [...]*, cannot be checked without a link prediction being performed to classify the relationship between Ford and Star Wars.

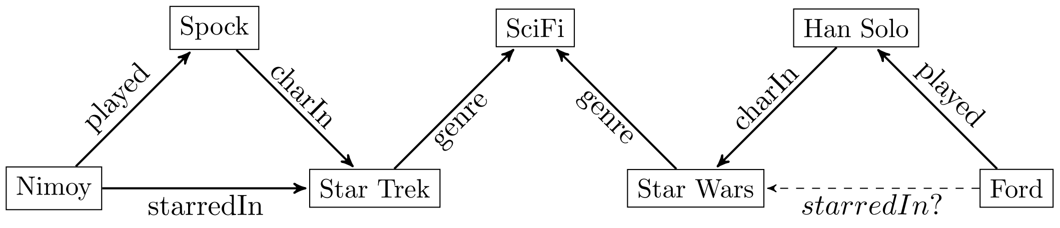

Figure 2: Directed labeled graph constructed from a simple knowledge base with facts on the form (e_a, r, e_b), tuples describing relationships between two entities, in this case about two actors of popular SciFi-movies. This example shows the tuple $(Ford, starredIn, Star\ Wars)$ missing, illustrating the problem of link prediction based on structural similarities.

There are different approaches on how to interpret a missing fact. The *open world assumption* (OWA) is done by the Semantic Web (and by extension, RDF). The OWA interprets a missing triple as the truthness of its existence as unknown. The *closed world assumption* simple assumes that the missing triple indicates that the fact is false. For training purposes, it is usually assumed that a knowledge base is locally complete. This *local closed world assumption* (LCWA) allows a training algorithm to assume that any triples not seen are false. Otherwise, when predicting a relationship would involve predicting every other possible relationship around as well, which quickly becomes intractable.

Inference

Link prediction belongs to what is known as knowledge inference, i.e., deducing new facts from previous knowledge. Knowledge inference can often be computationally difficult, e.g., NP-hard for general graph structures [16]. Inference usually involve some form of parameter estimation, e.g., feature weights. Equation 2.1 can be viewed as an inference procedure.

2.2 Learning Theory

The research field of learning theory studies the design and analysis of machine learning algorithms. The field can be divided into two subfields; computational and statistical learning theory. Computational learning theory provides mathematical models of learning to study the predictive power and computational efficiency of learning algorithms over a hypothesis space. Statistical learning theory is concerned with finding bounds on how well learning algorithms can perform given a hypothesis space. These fields give a theoretical basis for better understanding the inner workings of machine learning algorithms and problems. Central to the theoretical foundation of machine learning is PAC learning, or probably approximately correct learning, a result from learning theory. For an problem to be PAC learnable by an algorithm, it must be true that a hypothesis can be found that has a high probability to have a low error in classifying unseen data.

Hypothesis space

The concept of hypothesis spaces was briefly mentioned in Section 2.1. A hypothesis function $h \in \mathcal{H}$ is a mapping from $\mathcal{X}$ to $\mathcal{Y}$. Intuitively, the goal of learning is to find a h that gives the approximation of the true relationship between $\mathcal{X}$ and $\mathcal{Y}$. In the case of linear regression, a hypothesis space is defined when modelling a specific problem and could for example be the set of all linear functions on the form $y = k_1 1 + k_2 x_1 + \cdots + k_m x_m + c$ for some m given by the model.

Learning a hypothesis function usually entails some form of optimization problem. In order to formulate an objective function, it is necessary to define some way to measure the cost of an erroneous classification. Now, the task becomes minimizing such a loss function.

Definition 2 (Loss function)
Given an output space $\mathcal{Y}$, a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ *measures the dissimilarity of two elements in $\mathcal{Y}$. L is commonly assumed to be* definite, *i.e., that $L(y, y') = 0$ iff. $y = y'$.*

The loss function is problem specific, but there are a couple of common classes of loss functions, including the 0-1, Hamming, square, hinge and logistic loss function. Some of these are shown in Figure 3. They all distinguish elements in $\mathcal{Y}$ with different range and properties. The 0-1 loss function is an indicator function, defined as:

$$L(y, y') = \begin{cases} 0, & \text{if } y = y' \\ 1, & \text{if } y \neq y' \end{cases} \tag{2.2}$$

The Hamming loss function calculates the Hamming distance by $L(y, y') = \frac{1}{l} \sum_{k=1}^{l} 1_{y_k \neq y'_k}$, i.e., a measurement of how many substructures y and y' differ by. This decompositional property is important in structured prediction when choosing a loss function[17]. Usually, there is a natural candidate for such a loss function, e.g., such as the edit-distance for natural language applications.

Only knowing the amount of loss a classifier has on given training data does not necessarly give much information on the quality of the classifier. When formulating a classification problem, it is necessary to define some other measurement of the classifier performance. One way is to train classifiers and evaluate their performance experimentally, but this is a cumbersome task as it is generally resource intense and hard to do if a new algorithm is deviced.

Therefore, when a classifier is trained, we introduce the notions of *generalization error* and *empirical error*. During the training phase, the empirical error captures how well the classifier fits the training data. It is necessary for this error to be sufficiently small to allow convergence but large enough to not overfit the classifier. The generalization error denotes how well the classifier adapts to data not part of the training set. The goal is to minimize the generalization error. However, the relationship between these errors and classifier over-/underfitting leads to the complex task of finding a balance between the two.

Figure 4 shows how a larger empirical error of a linear classifier can give a smaller generalization error than a high-degree polynomial classifier with perfect fit to the training data. This is a good example of how overfitting a classifier during training can yield poor classifiers. Similarly, if the data set in Figure 4 was generated from

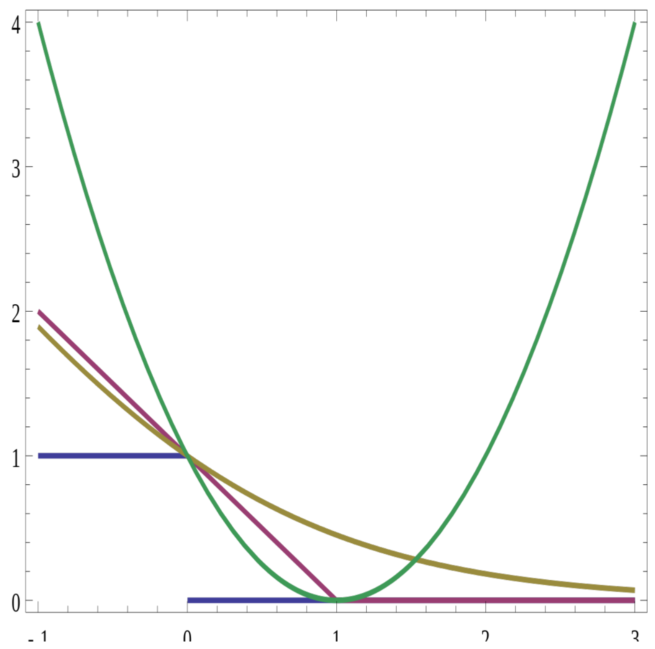

Figure 3: Examples of loss functions commonly used in learning scenarios. Blue is a 0-1 indicator function. Green is a square loss function. Purple is a hinge loss function. Yellow is a logistic loss function. Image distributed under GNU FDL Version 1.2 [18].

a more complex function family, the linear classifier would most likely suffer from underfitting. Both the empirical and generalization error would be much higher and the linear classifier would not even be able to approximate the training data. Extending on this idea, it is clear that the relationship between the function class-complexity and the empirical and generalization error could give insights into how to train a classifier. To do this, we introduce the concept of risk minimization.

Risk minimization

The concept of risk is tightly connected to reasoning about classifier performance. Risk can be measured at different stages of a learning scenario, but the *true risk* is what is important. The true risk denotes the risk of missclassification over the whole $\mathcal{X} \times \mathcal{Y}$.

Definition 3 (Risk)
Given a loss function L and a classifier function $f : \mathcal{X} \to \mathcal{Y}$, the risk of f is defined as the expected loss of f over all points $x \in \mathcal{X}$

$$R(f) := E_{(x,y)\sim\mathcal{D}}(L(y, f(x))) \tag{2.3}$$

where $\mathcal{D}$ is some fixed probability distribution.

In statistical learning theory two assumptions are made on $\mathcal{D}$[19], which has been touched upon earlier but summarized by the following:

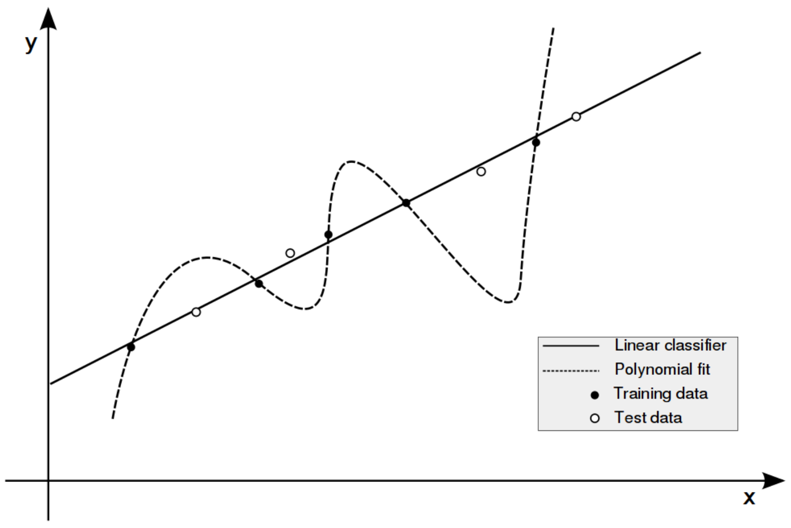

Figure 4: An example of how larger empirical error can be acceptable in favor of a lower generalization error. The linear classifier does not fit the test data perfectly, a high-degree polynomial would give a much smaller training error. However, when unseen data points are introduced, the linear classifier outperforms the more complex classifier by far.

Assumption 1 There exists a fixed probability distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$, i.e., the underlying structure of $\mathcal{X} \times \mathcal{Y}$ does not change.

Assumption 2 Training data is sampled independently from $\mathcal{D}$ (i.i.d. sampling).

No assumptions are made on the form of the probability distribution $\mathcal{D}$, and $\mathcal{D}$ is unknown at the time of learning. Making a useful estimate of $\mathcal{D}$ based on data is seldom realistic. This means that computing the true risk of a function f is not possible. However, an upper bound can be found based on the empirical risk combined with an analysis of the complexity of a classifier function. Intuitively, the empirical risk counts the ratio of training points missclassified by a classifier [20]. For a 0-1 loss function, this is the number of missclassifications over the number of classifications.

Definition 4 (Empirical risk)
Given some training data $S = (x_1, y_1), ..., (x_m, y_m)$ and a loss function L, the empirical risk *of classifier f is defined as*

$$R_{emp}(f) := \frac{1}{m} \sum_{i=1}^{m} L(y_i, f(x_i)). \tag{2.4}$$

This is sometimes written as $\hat{R}(f)$.

Figure 5 shows the impact the complexity of the function class to classify affects the relationship between the empirical error and the generalization error of a classifier.

Considering a function space $\mathcal{F}$ of classifiers, it is possible to choose the most promising classifier according to some training data. In an ideal situation where the true risk is known, a perfect classifier $\boldsymbol{f}$ could be computed as

$$\boldsymbol{f} = \arg\min_{f \in \mathcal{F}} R(f) \tag{2.5}$$

The optimization problem in Equation 2.5 is known as *risk minimization.* A naïve approximation of such a solution during the training of a classifier is using the empirical risk. Choosing a classifier using this procedure is commonly refered to as *empirical risk minimization.* This is equivalent to, e.g., the least-squares method but defined as a general concept. It is generally a bad idea to just choose the hypothesis that minimizes the empirical error [13], as was shown in Figure 4.

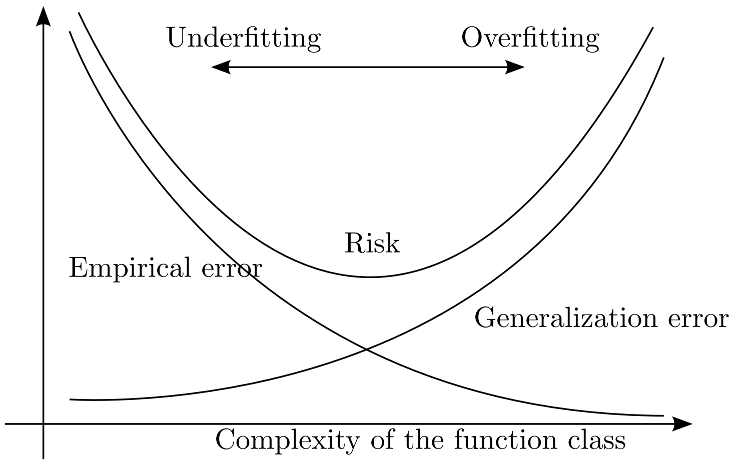

Figure 5: The relationship between the complexity of a classifier function and the estimation and generalization errors [19].

Studying the empirical error in itself does not necessarily give much information on the generalization error, which is what really matters when building a good classifier. Therefore, Vapnik and Chervonenkis introduced the principle of *structural risk minimization.* Structural risk minimization uses the empirical risk $R_{emp}(f)$ together with a measurement of the complexity of the classifier function family to provide an upper bound on the true risk of f. In their work they introduced the Vapnik-Chervonenkis (VC) dimension as a complexity measurement to capture this. This thesis will not use VC dimension, but it gives a nice introduction to the concept of measures used in structural risk minimization.

The VC dimension is an integer denoting the capability of a family of functions to separate labeled data. If 2D points with binary labels are to be classified by a linear classifier, the capability of such a classifier translates to the minimum number of data points that it always can separate properly. Figure 6 shows a simple example where three non-collinear points always are separable regardless of their labeling, whereas four points can be labeled such that a plane classifier is needed. A set that is always separable is called a *shattering set* and the VC dimension of a function class $\mathcal{F}$ is the largest integer such that there exists a subset of the input space which is shattered by $\mathcal{F}$. We denote this by $VC(\mathcal{F})$.

Definition 5 now gives an upper bound on the true risk of $\mathcal{F}$.

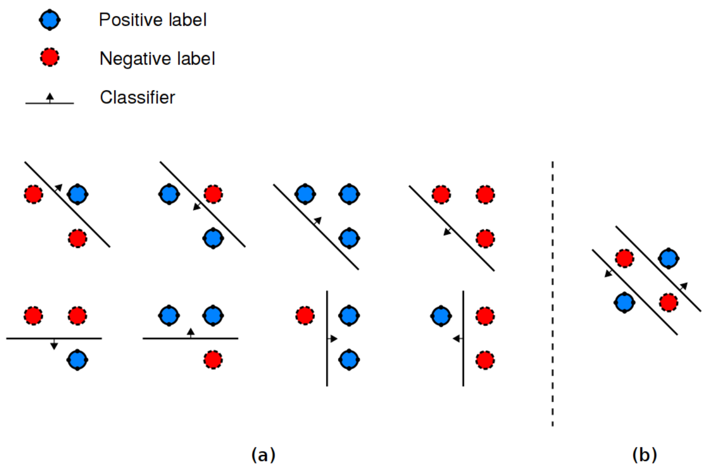

Figure 6: Example to illustrate VC dimension. A linear classifier is used to separate two types of data, red and blue points. The VC dimension of this example is 3 since the classifier can always separate three points regardless of the label ordering, but four points have configurations which can not be separated properly.

Definition 5 (VC dimension generalization bound)
For all $f \in \mathcal{F}$, with probability at least $1 - \delta$:

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{h(\log(2n/h) + 1) - \log(\delta/4)}{n}}. \tag{2.6}$$

where $h = VC(\mathcal{F})$ and n is the sample size.

The bound given by Equation 2.7 only depends on the structure of the function class, i.e., the underlying probability distribution is not taken into account. Another measurement that depend on this distribution is the *Rademacher complexity*, which as a result usually gives a better generalization bound [20].

Rademacher Complexity

It is not guaranteed that a problem with an infinite hypothesis set allows for efficient learning. Therefore, it is necessary to define a complexity notion to reason about this. One such tool is Rademacher complexity, which looks at the richness of a family of functions and to which degree a hypothesis set can fit random noise [21].

Similarly to the generalization bound defined using VC dimensions in Equation 2.7, a generalization bound using Rademacher complexity can be constructed.

Definition 6 (Rademacher complexity generalization bound)
For all $f \in \mathcal{F}$, with probability at least $1 - \delta$:

$$R(f) \leq R_{emp}(f) + 2\mathfrak{R}(\mathcal{F}) + \sqrt{\frac{\log(1/\delta)}{2n}} \tag{2.7}$$

where n is the sample size and $\mathfrak{R}(\mathcal{F})$ the Rademacher complexity of the function class $\mathcal{F}$.

Since the Rademacher complexity of a function class depends on the underlying distribution, it is also dependent on samples to be computable. As a result, the first step is to compute the *empirical* Rademacher complexity. This is done using *Rademacher variables*, essentially independent uniform random variables that take the values in $-1, +1$ with equal probability to simulate how well the function class fits a random labelling of data.

Definition 7 (Empirical Rademacher complexity)
Let G be a family of functions mapping from Z to $[a, b]$ and $S = (z_1, ..., z_m)$ a fixed sample of size m with elements in Z. Then, the empirical Rademacher complexity *of G with respect to the sample S is defined as:*

$$\hat{\mathfrak{R}}_S(G) = \mathop{\mathbb{E}}_{\sigma}\left[\sup_{g \in G} \frac{1}{m} \sum_{i=1}^{m} \sigma_i g(z_i)\right], \tag{2.8}$$

where $\sigma = (\sigma_1, ..., \sigma_m)^\top$, with σ_is independent uniform random variables taking values in $\{-1, +1\}$. The random variables σ_i are called Rademacher variables*.*

Using this definition it is now possible to define the Rademacher complexity using all possible samples of a given size drawn with a given distribution D. This translates into for each sample choosing a function $f \in \mathcal{F}$ that fits best to the random labelings of the sample. Now, taking the expectation over both the data and the random labels, high Rademacher complexity denotes that the function family can fit random labeling well [20]. Definition 8 formalizes the relationship between the empirical and true Rademacher complexity [21].

Definition 8 (Rademacher complexity)
Let D denote the distribution according to which samples are drawn. For any integer $m \geq 1$, the Rademacher complexity *of G is the expectation of the empirical Rademacher complexity over all samples of size m drawn according to D:*

$$\mathfrak{R}_m(G) = \mathop{\mathbb{E}}_{S \sim D^m}[\hat{\mathfrak{R}}_S(G)]. \tag{2.9}$$

A pattern shared by the VC dimension and Rademacher complexity is that the upper bound on $R(\mathcal{F})$ depends on the empirical risk, some capacity of $\mathcal{F}$ and a confidence term [20]. This is summarized in Equation 2.10.

$$R(F) \leq R_{emp}(f) + capacity(\mathcal{F}) + confidence(\delta) \tag{2.10}$$

An alternative approach to formulating an upper bound as in Equation 2.10, is by calculating the regularized risk

$$R_{reg}(f) = R_{emp}(f) + \lambda\Omega(f). \tag{2.11}$$

The function $\Omega(f)$ is called the regularizer, used to penalize classifier functions with high complexity. In order to balance the empirical risk and the regularization term, a weight parameter λ is used to be able to tweak this balance [20].

2.3 Conditional Random Fields

Conditional random fields were presented by Lafferty et al. in 2001 [22]. They presented an alternative to Hidden Markov Models (HMMs) for segmenting and labeling of sequence data. One of the advantages over HMMs is the relaxation of independence assumptions necessary for HMMs to allow tractable inference [22, 23]. Conditional Random Fields essentially associates an undirected graphical structure with a conditional distribution $P(y|x)$, with X being a random variable over data to be labeled and Y being a random variable over the label sequences [22].

Definition 9 (Conditional Random Field)
Let $G = (V, E, F)$ be a factor graph with $V = X \cup Y$ denoting the set of variable vertices, $F = \{\Psi_A\}$ the set of factor vertices and $E = \{\langle v, f\rangle | v \in V, f \in F\}$ the set of edges between variable and factor vertices. Then the conditional distribution can be defined as

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z} \prod_{\Psi_A \in F} \Psi_A(\boldsymbol{x}_A, \boldsymbol{y}_a). \tag{2.12}$$

where Z denotes the normalization factor such that p sums to 1,

$$Z = \sum_{x,y} \prod_{\Psi_A \in F} \Psi_A(\boldsymbol{x}_A, \boldsymbol{y}_a). \tag{2.13}$$

A factor here can intuitively be thought as capturing the relation between all variables in a clique in the underlying model **??**.

In Definition 9, the factors Ψ_A measures how well the subsets $\boldsymbol{x}_A \subseteq X$ and $\boldsymbol{y}_A \subseteq Y$ fit together. The sets $\boldsymbol{x}_A$ and $\boldsymbol{y}_A$ are containing the variable nodes that are conditionally dependent according to some distribution. In Figure 7, Ψ_3 would have as input $\boldsymbol{x}_3 = \{x_2, x_3\}$ and $\boldsymbol{y}_3 = \{y_3\}$. This means that the computation of the probability $p(\boldsymbol{x}|\boldsymbol{y})$ can be computed efficiently with a factorization if the factor functions are efficiently computable. The probability $p(\boldsymbol{y}|\boldsymbol{x})$ would then be factorized as $p(y_1, y_2, y_3|x_1, x_2, x_3) = \Psi_1(x_1, x_2, x_3)\Psi_2(x_1, y_1)\Psi_3(x_2, x_3, y_3)\Psi_4(x_3, y_1, y_2)$ and thus encode the conditional probabilities between all variables according to the structure of the graph. One common approach is to model Ψ_A using feature functions $f \in \mathcal{F}$,

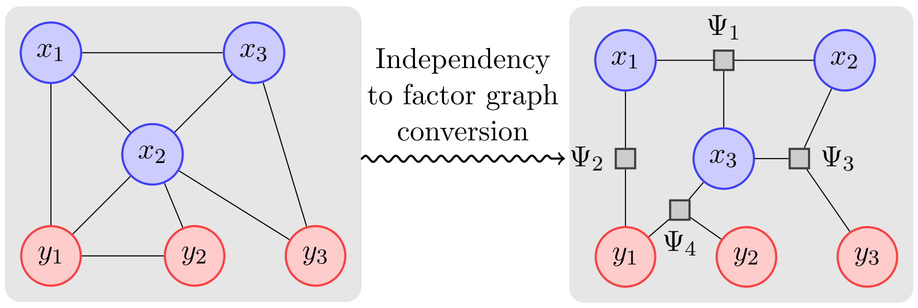

Figure 7: Example CRF given the underlying undirected graphical model. Represents a factorization of the probability $p(\boldsymbol{y}|\boldsymbol{x})$.

where $\mathcal{F}$ is some family of functions [24]. The factors can now be written as

$$\Psi_A(\boldsymbol{x}_A, \boldsymbol{y}_A) = \prod_{\Psi_A \in F} \exp\left\{\sum_k \lambda_{Ak} f_{Ak}(\boldsymbol{y}_A, \boldsymbol{x}_A)\right\}. \tag{2.14}$$

where the exponential function ensures a non-negative value.

Equation 2.14 introduces two new concepts; *feature functions* and *feature weights*. Feature functions captures the domain features mapping them to real values. One common way to model features is to use indicator functions, i.e., binary-valued functions $f_i(x, y) \in \{0, 1\}$. An example of this would be that of feature functions in *Part-Of-Speech tagging*, the problem of annotating the words in a sentence with their gramatical function. In POS tagging, such a feature function could be,

$$f_i(x_k, y_k) = \begin{cases} 1 & \text{if } y_k = \textbf{name} \text{ and } x_k = \textbf{Musk} \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Equation 2.15 shows a feature function that is active if the the kth word is the name $Musk$.

Feature weights are introduced as a parameter vector $\boldsymbol{\lambda}$ of scalars λ_i used as weight for the feature function f_i. Whenever feature f_i is active, the feature weight λ_i is used to increase or decrease the impact this f_i will have on the whole classification.

Parameter Estimation and Inference in CRFs

The parameters $\boldsymbol{\lambda}$ can either be engineered using domain knowledge or learned from training data [25]. Learning these parameters is called *parameter estimation* and is a core concept in making CRFs effective. Given a estimated parameters $\boldsymbol{\lambda}$, the probability function will now depend on the parameter vector; $p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\lambda})$. Estimating these parameters can be done by, e.g., using maximum likelihood (ML) or maximum a posteriori probability (MAP) estimation using standard numerical optimization methods such as gradient descent or Newton's method[22]. ML and MAP both use observed facts to derive what stochastic process could generate them. Observe that this parameter vector can be extended to include other parameters for different models as well. Usually, the notation separates between the feature weight vector and all parameters, using $\boldsymbol{\theta}$ to denote the latter with $\boldsymbol{\lambda} \in \boldsymbol{\theta}$.

Parameter estimation is one part of the larger task of *inference*, i.e., predicting the output, e.g., as formulated in Equation 2.1. However, such inference is usually intractable for general graphs. In order to compute this efficiently either the CRF must possess nice structural properties or an approximative algorithm must be used. CRFs with a tree structure are typical examples of the former, while particle-based methods can be used as a basis for approximative algorithms [26, 16].

3 Background

A knowledge base can contain millions of facts on the form *(Obama, presidentOf, USA)*. The Semantic Web introduced the Resource Description Framework (RDF), which is usually used to encode such knowledge [4]. RDF allows for type hierarchies and data linkage, e.g. by defining entity classes and relationship types [3]. Recent efforts towards storing such factual information have resulted in a number of publically available knowledge bases. Table 1 shows some of them, including their sizes. Many of these are built upon semi-structured information extracted from e.g. Wikipedia (such as DBpedia and Freebase). Google provides their own knowledge base partially built on Freebase, Google Knowledge Graph [11]. Google's Knowledge Graph is central in many of the company's applications. Its most prominent usage is the addition of semantic information to the Google search engine, but Knowledge Graph also helps with automatic query completion and allows virtual assistants to answer natural-language questions [11].

Table 1 Examples of existing knowledge bases and their size. Data from [6].

	Nr. of entities	Nr. of Relation Types	Nr. of Facts
Freebase	40 M	35000	637 M
Wikidata	18 M	1632	66 M
DBpedia (en)	4.6 M	1367	538 M
YAGO2	9.8 M	114	447 M

Knowledge Vault

A good example of how a knowledge base can improve extractions is Knowledge Vault. Knowledge Vault is a Google research project where structured prediction is used to automatically construct probabilistic knowledge bases. Predicted probabilities were used in combination with extractor confidence to improve NLP fact extraction. Their approach increased the number of high confidence facts from 100 million to 271 million, of which 33 percent were new facts not present in Freebase [8]. An example of their results is shown below, where two pieces of text are analysed to produce a new fact denoting that `Barry Ritcher` attended `University of Wisconsin-Madison`.

```
<Barry Ritcher (/m/02ql38b),
/people/person/edu./edu/edu/institution,
University of Wisconsin-Madison (/m/01yx1b)>
```

This triple was extracted with a confidence of 0.14, based on the two following sources of information:

```
In the fall of 1989, Ritcher accepted a scholarship to the
university of Wisconsin, where he played for four years and
earned numerous individual accolades...

The Polar Caps' cause has been helped by the impact of
knowledgable coaches such as Andringa, Byce and former
UW temmates Chris Tancill and Barry Ritcher.
```

Knowledge Vault computes a prior belief based on its knowledge base by using structured prediction techniques. This knowledge base contains the fact that Barry Ritcher was born and raised in Madison, which increases the prior belief that he also went to school there. Together with the rather low confidence on the extraction (0.14), the final confidence in the extracted triple is 0.61 which is a rather large improvement. They call this knowledge fusion.

IBM's Watson

Other applications of knowledge bases include IBM's work on question answering computer system Watson, used to beat human experts in *Jeopardy!*. In the underlying machinery, Watson used a knowledge base as a part of scoring competing alternatives with a confidence using previous knowledge from e.g. Freebase. Together with other components, this information was used to decide alternative seemed most resonable[27]. Today, Watson is accessible for developers to assist in, e.g., education, IoT, and health. An example of an application in the last area is cancer treatment assistance [28].

3.1 Link Prediction in Knowledge Bases

Building a knowledge base manually is a difficult task, only feasible for small expert systems with very specific use cases, such as in-house company FAQs. Therefore it is necessary to automate the process of both extraction and extension of a knowledge base.

In a recent paper written in collaboration between Google and MIT researchers, Nickel et al. provide a review over the state-of-the-art of relational machine learning for knowledge graphs [6]. Their focus lies on *Statistical Relational Learning*, which roughly is a structured prediction with a probabilistic annotation, i.e., the confidence in the existence of a relationship between two entities. A *knowledge graph* is essentially a knowledge base with a graph structure, sometimes also referred to as a *heterogeneous information network*. Most knowledge bases described in this thesis can be considered knowledge graphs. The authors describe three different approaches to modelling knowledge graphs for relational learning; Latent features, graph features and Markov Random Field models.

Markov Random Field Models

The authors note that this is technically a Conditional Random Field model, and the theory of Section 2.3 can therefore be used as a basis. A MRF just models a probability distribution over random variables, whereas CRF introduces a conditioning on some features x given output y. As shown in Section 2.3, in order to formulate a predictive model, it is necessary to define a set of feature functions. Connected to the concept of MRFs is *Markov Logic Networks*[29]. A MLN is based on a set of logical formulae such that an edge between nodes in the MLN corresponds to the facts occuring in at least one grounded formula F_i. Returning to the actor example in Figure 2 of Section 2.1, one such formula could be

$$F_1 : (p, played, c) \wedge (c, charIn, m) \Rightarrow (p, starredIn, m). \tag{3.1}$$

Using such a formulae set $\mathcal{F} = \{F_i\}_{i=1}^{L}$, by counting the number of true groundings x_c of F_c in Y, the first equation defining CRFs in Definition 9 in Section 2.3 can be written as

$$P(Y|\theta) = \frac{1}{Z} \sum_c \exp(\theta_c x_c). \tag{3.2}$$

with the definition of Z written analogously. Here θ_c denotes the weight of a formula F_c. An example of a grounding of Equation 3.1 could be $p =$ *Ford*, $c =$ *Han Solo*, $m =$ *Star Wars*.

Latent Feature Models

Latent feature models assume that all facts can be explained using latent features, i.e., features that cannot be observed directly. The authors present an example where a latent feature is an entity being a good actor that explains the actor recieving an Oscar, a fact observable in the knowledge graph. Latent features can be modelled using RESCAL, a bilinear relational latent feature model. RESCAL[15] uses pairwise interactions between latent features to explain triples, and can be formulated for a triple y_{ijk} as

$$f_{ijk}^{\texttt{RESCAL}} := \boldsymbol{e}_i^\top \boldsymbol{W}_k \boldsymbol{e}_j = \sum_{a=1}^{H_e} \sum_{b=1}^{H_e} w_{abk} e_{ia} e_{jb} \tag{3.3}$$

where H_e denotes the number of latent features for entities and $\boldsymbol{W}_k \in \mathbb{R}^{H_e \times H_e}$. Each weight $w_{abk} \in \boldsymbol{W}_k$ describes the strength of the interaction between latent features a and b in the k-th relation (i.e. *starredIn*). An example of this interaction is shown in Figure 8.

Equation 3.3 can be formulated using the Kronecker product to achieve a feature vector $\phi_{ij}^{\texttt{RESCAL}}$,

$$f_{ijk}^{\texttt{RESCAL}} := W_k \boldsymbol{e}_i \otimes \boldsymbol{e}_j = W_k \phi_{ij}^{\texttt{RESCAL}}. \tag{3.4}$$

The formulation in Equation 3.4 can be benificial for certain problems, depending on requirements on the model.

One approach to solving the RESCAL problem is by tensor factorization [30]. Tensor factorization is a generalization of matrix factorization to higher-order data, as tensors essentially are multidimensional arrays with some formal requirements [31]. In this setting, a tensor represents a higher-order relationship between latent variables. RESCAL has been shown to outperform other latent feature model approaches, such as neural networks, for link prediction tasks [32].

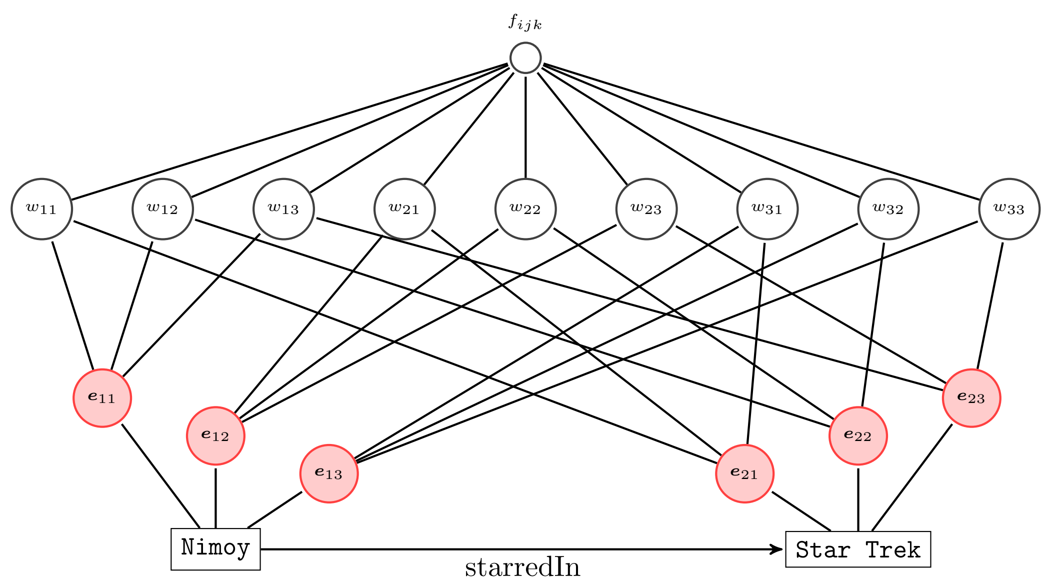

Figure 8: Showing how latent variables of two entities interact for $H_e = 3$ via the weight matrix.

The weights W_k and the latent features $\boldsymbol{e}_i$ are trained simultanously by the RESCAL-ALS algorithm, an alternating least-squares approach. RESCAL-ALS is practically viable as 30-50 iterated updates is usually enough to reach a stable solution [33].

Graph Feature Models

Contrary to latent feature models, a *graph feature model* extracts features that are directly observable as facts in the graph. This could, e.g., be using the facts *(Ford, played, Han Solo)* and *(Han Solo, charIn, Star Wars)* to predict the fact *(Ford, starredIn, Star Wars)*. Graph feature models are used extensively in link prediction for single relation graphs, such as social networks where a relationship between two people indicates friendship [6, 34]. There are several approaches where the idea is that similarity between entities can be derived from paths and random walks of a bounded length. This means that examining what can be reached from an entity given a length parameter, should be enough to measure similarity between two entities. One such approach is the *local random walk*-method [35]. As most knowledge bases are multi-relational, the random walk approach must be extended upon. This is done by the Path Ranking Algorithm, which also uses a bounding length but generalizes to paths on arbitary relations [36].

Path Ranking Algorithm

To extend the random walk idea to multi-relational knowledge graphs, $\pi_L(i, j, k, t)$ denotes a path of length L over a sequence t of relationship between entities $\tau = e_i \overset{r_1}{\to} e_2 \overset{r_2}{\to} \cdots \overset{r_L}{\to} e_j$. As the goal is to use the intermediate path τ to predict the relation k between e_i and e_j, the edge $e_i \overset{r_k}{\to} e_j$ is also required to exist for $\pi_L(i, j, k, t)$ to be valid. The set $\Pi_L(i, j, k)$ denoting all such paths can then be found by enumerating all paths from e_i to e_j. Such an enumeration is only practical when the number of relation types is small, for knowledge bases a more efficient approach is necessary. Lao et al. suggests a random sampling approach where not every relation type is used in generating paths according to a usefulness measure learned during training. Since the generations of such paths now depends on a random sampling, it is possible to compute the probability of following that path. Assuming that an outgoing link is picked uniformly at random, the probability $P(\pi_L(i, j, k, t))$ of a path can be computed recursively using an efficient procedure [36]. Now, using this probability as features, a feature vector can be defined as

$$\phi_{ijk}^{\mathtt{PRA}} = [P(\pi) : \pi \in \Pi_L(i, j, k)] \tag{3.5}$$

The feature vector $\phi_{ijk}^{\mathtt{PRA}}$ can be used directly for each pair of entities to predict the probabilities of each relation k between them using a feature vector $\boldsymbol{w}_k$

$$f_{ijk}^{\mathtt{PRA}} := \boldsymbol{w}_k^\top \phi_{ijk}^{\mathtt{PRA}} \tag{3.6}$$

Using PRA to model features is beneficial since the features correspond directly to Horn clauses, with the addition that a weight (or probability) specifies how predictive the corresponding clause is. The example Horn clause in Equation 3.7 corresponds to the edge prediction problem shown in Figure 2 in Section 2.1.

$$(p, starredIn, m) \leftarrow (p, played, c) \wedge (c, charIn, m). \tag{3.7}$$

Returning to the example of Google's Knowledge Vault previously presented, this is how they calculate their predicted probabilities. Table 2 shows three examples of Horn clauses used to predict which college a person attends.

Table 2 PRA Freebase Knowledge Vault college attendee F1 Precision Recall Weight

Relation Path	**F1**	**Prec**	**Rec**	**Weight**
(draftedBy, school)	0.03	1.0	0.01	2.62
(sibling(s), sibling, education, institution)	0.05	0.55	0.02	1.88
(spouse(s), spouse, education, institution)	0.06	0.41	0.02	1.87
(parents, education, institution)	0.04	0.29	0.02	1.37
(children, education, institution)	0.05	0.21	0.02	1.85
(placeOfBirth, peopleBornHere, education)	0.13	0.1	0.38	6.4
(type, instance, education, institution)	0.05	0.04	0.34	1.74
(profession, peopleWithProf., education, institution)	0.04	0.03	0.33	2.19

According to their model, Table 2 shows, e.g., that a person drafted by a university can accuratly be predicted to also study there but not all students will be found using this path. Their results also suggests that the place of birth is useful information, as the both the $F1$-score and weight are the highest amongst the given paths. The $F1$-score gives the relationship between precision and recall, where a high $F1$-score suggests a better prediction.

The PRA algorithm has been shown to give good prediction performance even for binary features [37]. Since working with floating point numbers usually entails heavier computations, reducing the probabilities to binary values can greatly improve the efficiency of the algorithm. The authors also provide an open source implementation of PRA [38].

Combining RESCAL and PRA

Although the PRA algorithm can be used to achieve good results on link prediction for Freebase in Knowledge Vault (with a 0.884 area under the receiver operating characteristic (ROC) curve)[8], it has been shown that latent and graph-based models have different strengths [39]. The ROC curve plots the true positive rate against the false positive rate, area (AUC) under measures the ranking quality. Therefore, a natural step forward would be to combine these two approaches. In [30] Nickel et al. show that the RESCAL computation can be sped up significantly if observable features are included. This is achieved as the rank of the tensor factorization can be lowered allowing a lower latent dimensionality. Essentially, RESCAL now only needs to fill in where a graph feature model misses out. Not only does the computational complexity decrease, but this combination allows for higher predictive performance as well.

Based on these findings, a combination [6] of RESCAL and PRA can be formulated as

$$f_{ijk}^{\texttt{RESCAL+PRA}} = \boldsymbol{w}_k^{(1)\top} \phi_{ij}^{\texttt{RESCAL}} + \boldsymbol{w}_k^{(2)\top} \phi_{ijk}^{\texttt{PRA}}. \tag{3.8}$$

3.2 Voted Conditional Random Fields

In their paper *Structured Prediction Theory Based on Factor Graph Complexity*[14], Cortes et al. presents new data-dependent learning guarantees based on theoretical analysis of structured prediction using factor graph complexity. They combine these learning bounds with the principle of Voted Risk Minimization [17] in their design of two new algorithms, *Voted Conditional Random Fields* and *Voted Structured Boosting.*

Factor graph complexity

Cortes et al. define the *empirical factor graph Rademacher complexity* $\hat{\mathfrak{R}}_S^G(\mathcal{H})$, where $\mathcal{H}$ is a hypothesis set for a sample $S = (x_1, ..., x_m)$ and a factor graph G:

$$\hat{\mathfrak{R}}_S^G(\mathcal{H}) = \frac{1}{m} \mathop{\mathbb{E}}_{\epsilon} \left[\sup_{h \in \mathcal{H}} \sum_{i=1}^{m} \sum_{f \in F_i} \sum_{y \in \mathcal{Y}_f} \sqrt{|F_i|} \epsilon_{i,f,y} h_f(x_i, y) \right] \tag{3.9}$$

This denotes the expectation over the set of independent Rademacher variables $\boldsymbol{\epsilon} = (\epsilon_{i,f,y})_{i \in [m], f \in F_i, y \in \mathcal{Y}_f}$. This is an extension of the theory in Section 2.2 and in particular Definition 7. The *factor graph Rademacher complexity* is defined as the standard Rademacher complexity, $\mathfrak{R}_m(G) = \mathop{\mathbb{E}}_{S \sim D^m} [\hat{\mathfrak{R}}_S(G)]$, see Definition 8 in Section 2.2.

With a definition of the Rademacher complexity, it is possible to find a bound on the generalization error. First, it is necessary to define a couple of building blocks. In order to measure the confidence of a hypothesis, the concept of *margin* is introduced for an input/output pair (x, y). This will be helpful in proving the generalization bound as shown in Theorem 1.

Definition 10 (Margin)
The margin *of a hypothesis h at a labeled point $(x, y) \in (\mathcal{X} \times \mathcal{Y})$ is defined as*

$$\rho_h(x, y, y') = \min_{y' \neq y} h(x, y) - h(x, y'). \tag{3.10}$$

Figure 9 illustrates the margin in the case of binary labeling of data points in two dimensions.

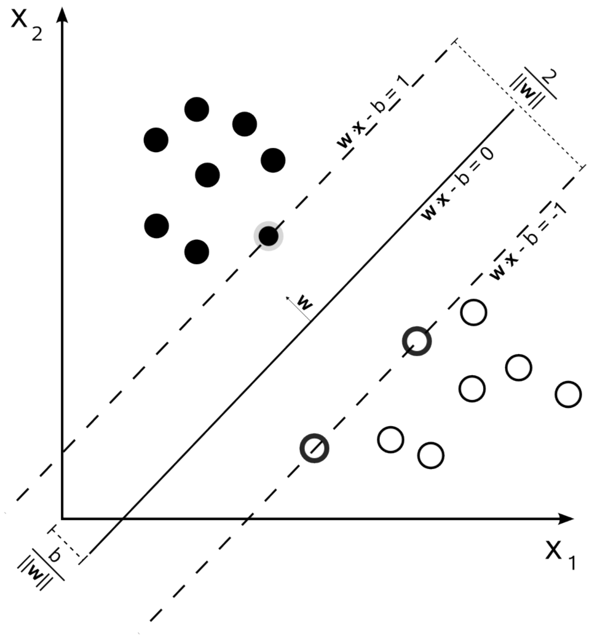

Figure 9: Example of how Support Vector Machines maximizes the margin between labels. Here the margin is the distance from the separating line to the closest data points.

The margin should be interpreted as the distance between a classifications, given input x. In other words, ρ_h measures how distinct classifications h gives. A low margin $\rho_h(x, y)$ signals a low confidence as there are other candidates y' close at hand.The margin is not used explicitly in the remainder of this thesis, however, the generalization bounds provided by Cortes et al. rely on the relationship between the loss function and the margin of h as well as the empirical margin losses defined below:

$$\hat{R}^{add}_{S,\rho}(h) = \mathop{\mathbb{E}}_{(x,y)\sim S}\left[\Phi^*\left(\max_{y'\neq y} \mathtt{L}(y', y) - \frac{1}{\rho}\big[h(x, y) - h(x, y')\big]\right)\right] \tag{3.11}$$

$$\hat{R}^{mult}_{S,\rho}(h) = \mathop{\mathbb{E}}_{(x,y)\sim S}\left[\Phi^*\left(\max_{y'\neq y} \mathtt{L}(y', y)\big(1 - \frac{1}{\rho}\big[h(x, y) - h(x, y')\big]\big)\right)\right] \tag{3.12}$$

where $\Phi^*(r) = \min(M, \max(0, r))$ for all r, with $M = \max_{y,y'} \mathtt{L}(y, y')$.

The empirical margin loss is defined both for multiplicative and additive margins, as choosing a hypothesis from a convex combination of hypothesis families can be expressed in both multiplicative and additive terms [40]. This is a necessary step in defining tightening the generalization bounds, which will be shown later. Theorem 1 is the shows how the empirical margin loss can be used together with the factor graph Rademacher complexity to an upper bound on the generalization error $R(H)$.

Theorem 1
Fix $\rho > 0$. For any $\delta > 0$, with probability at least $1 - \delta$ over the draw of a sample S of size m, the following holds for all $h \in \mathcal{H}$,

$$R(H) \leq R_{\rho}^{add}(h) \leq \hat{R}_{S,\rho}^{add}(h) + \frac{4\sqrt{2}}{\rho}\mathfrak{R}_m^G(\mathcal{H}) + M\sqrt{\frac{\log\frac{1}{\delta}}{2m}},$$

$$R(H) \leq R_{\rho}^{mult}(h) \leq \hat{R}_{S,\rho}^{mult}(h) + \frac{4\sqrt{2}M}{\rho}\mathfrak{R}_m^G(\mathcal{H}) + M\sqrt{\frac{\log\frac{1}{\delta}}{2m}},$$

Making predictions

Using a factor graph G, Definition 11 formulates a scoring function based on the factor nodes of G.

Definition 11 (Scoring function)
Given an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, a scoring function $h : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ *gives a single value measuring how well $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ fit together. For the purpose of this thesis, the standard assumption that $h \in \mathcal{H}$ can be decomposed as a sum is made. This decomposition can be based on the factor graph $G = (V, F, E)$, giving*

$$h(x, y) = \sum_{f \in F} h_f(x, y_f). \tag{3.13}$$

,i.e., a summation of scoring each y_f local to some factor node f.

In order to construct such a scoring function it is necessary to define a scoring function family $\mathcal{H}$ from which h should be chosen, from here on called the *hypothesis set*. Such a scoring function should also be based on some features extracted from the input-output-space $\mathcal{X} \times \mathcal{Y}$. Therefore, it is necessary to define some way of mapping those features. These features are then the basis for scoring a prediction.

Definition 12 (Feature mapping)
A feature mapping Ψ *is a function from $(\mathcal{X} \times \mathcal{Y})$ to $\mathbb{R}^N$ such that Ψ is decomposable over all factors of F, i.e., $\Psi(x, y) = \sum_{f \in F} \Psi_f(x, y_f)$.*

The feature mapping function Ψ can now be used to define the hypothesis set, where a hypothesis is equivalent to the classifier function given by Definition 1 in Section 2.1. In a part-of-speech tagging, one such feature element could be the number of times the word *the* appears labeled as a determiner next to a word labeled as a noun [41]. The dimension N is therefore problem specific and could, e.g., be defined as the number of input features times the number of classes to allow a direct mapping between features and labels [42].

Now, using the feature vector computed by Ψ it is possible to define a hypothesis set from which a classifier can be chosen. Such a classifier is essentially a parameterized version of the feature mapping, weighted to choose how much each feature should influence the classification.

Definition 13 (Hypothesis set)
Given a feature mapping $\Psi : (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}^N$. For any p, the hypothesis set *$\mathcal{H}_p$ is defined as:*

$$\mathcal{H}_p = \{x \mapsto \boldsymbol{w} \cdot \boldsymbol{\Psi}(x, y) : \boldsymbol{w} \in \mathbb{R}^N, \|\boldsymbol{w}\|_p \leq \Lambda_p\}. \tag{3.14}$$

The number N denotes the number of possible features and Λ_p an upper bound on the weight vector, given as a parameter. Such a hypothesis set is labeled linear since it is a linear combination of feature functions Ψ_f.

The hypothesis set defined by Definition 3.14 is used by convex structured prediction algorithms such as structured support vector machines [43], Max-Margin Markov Networks [12] or Conditional Random Fields [22], as outlined by Cortes et al. in [14].

Given a hypothesis set, i.e., a family of scoring functions, a predictor $\mathtt{h}$ can be constructed for any $h \in \mathcal{H}$ by for any $x \in \mathcal{X}$ choosing $\mathtt{h}(x) = \arg\max_{y \in \mathcal{Y}} h(x, y)$. The predictor $\mathtt{h}$ is essentially the classifier function defined by Definition 2.1 in Section 2.1, i.e., a function returning the $y \in \mathcal{Y}$ that gives the largest weighted feature vector output. Table 3 shows a summary of the notation used in this section.

The value p used in Definition 3.14 denotes the vector norm used to bound the feature weights. The results extend to arbitrary p's, but focus in this thesis lies on $p = 1$. Theorem 2 gives an upper bound on $\hat{\mathfrak{R}}^G_m$ for p=$1, 2$, i.e., for bound using the Manhattan and Euclidian norms. This is achieved by using the sparsity of a feature mapping, i.e., how many features are active. Intuitively, the empirical Rademacher complexity should increase if the feature mapping increases the maximum number of active features. Feature mappings using binary indicator functions should be less complex than assigning floating point numbers to many features.

Theorem 2

$$\hat{\mathfrak{R}}^G_S(\mathcal{H}_1) \leq \frac{\Lambda_1 r_\infty}{m}\sqrt{s \log(2N)}, \quad \hat{\mathfrak{R}}^G_S(\mathcal{H}_2) \leq \frac{\Lambda_2 r_2}{m}\sqrt{\sum\nolimits_{i=1}^{m} \sum\nolimits_{f \in F_i} \sum\nolimits_{y \in \mathcal{Y}_f} |F_i|} \tag{3.15}$$

where $r_\infty = \max_{i,f,y} \|\boldsymbol{\Psi}_f(x_i, y)\|_\infty$, $r_2 = \max_{i,f,y} \|F_i\|_2$ *and the sparsity factor* $s = \max_{j \in [i,N]} \sum_{i=1}^{m} \sum_{f \in F_i} \sum_{y \in \mathcal{Y}_f} |F_i| 1_{\boldsymbol{\Psi}_{f,j}(x_i, y) \neq 0}$.

The intuitive description of the variable r_∞ is the maximum value of any feature.

Theorem 2 will later be used to formulate a complexity penalty. The factor graph-based Rademacher complexity can now be used to reason about the capacity of hypothesis families. In particular, the combination of families.

Table 3 Explaination of VCRF notation

$\mathcal{X}$	Input space
$\mathcal{Y}$	Output space
N	The length of the joint feature vector given by $\Psi(x, y)$.
$\Psi(x, y)$	Feature mapping from $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}^N$, measuring compatibility of x and y. For a simple model, the joint_feature(x, y) is a vector of size `n_features` $\times$ `n_classes`, which corresponds to one copy of the input features for each possibly class.
$\mathcal{H}_p$	Hypothesis class
$\boldsymbol{w}$	Weight vector
$\hat{\mathfrak{R}}^G_S(\mathcal{H})$	Empirical factor graph Rademacher complexity of hypothesis class $\mathcal{H}$ given a sample S on factor graph G
$\mathfrak{R}^G_m(\mathcal{H})$	Rademacher complexity of hypothesis class $\mathcal{H}$ over samples of size m on factor graph G.
Λ_p	An upper bound on the weight vector. Usually found via cross-validation.
p	The number of function families H_i.
r_k	Complexity penalty of function family H_k.
F_i	Factor graph for element i of sample S
$F(k)$	The factor graph of feature family k.
d_i	The largest number of active features for variables connected to any factor node $f \in F_i$.
$H_1, \ldots, H_p$	p families of functions mapping $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$
$L(y, y')$	Loss function measuring the dissimilarity of two elements in the output space.
Φ_u	Surrogate loss function.
λ, β	Parameters to the VCRF problem.
$\rho_h(x, y)$	Margin function, measuring the distance between a classification y and the second best output y'.

Voted risk minimization

The principle of *voted risk minimization*[17] is that a predictor family `h` can be decomposed into sub-families $H_1, \ldots, H_p$, as illustrated by Figure 10. These subfamilies could, e.g., correspond to different types of features that H is concerned with. In such a case, one subfamily could be representing the part of the feature vector encoding structure in the input with another subfamily encoding structure in the output. The main idea is that using families with rich features (such as is common in, e.g., NLP and computer vision) can increases the risk of overfitting. Voted risk minimization uses mixture weights to balance complex against simple feature families. These weights are adjusted so that more weight is given to simpler families if complex hypotheses are used. This idea is used to derive a Rademacher complexity which explicitly depends on the difference in complexity between subfamilies. The intuition is that an empirical risk minimizing algorithm could distribute its votes amongst the given subfamilies using the complexity penalty to make sure no family gets to much say in the prediction, i.e., leading to overfitting.

The decomposition of H can be formulated as the convex hull over the union

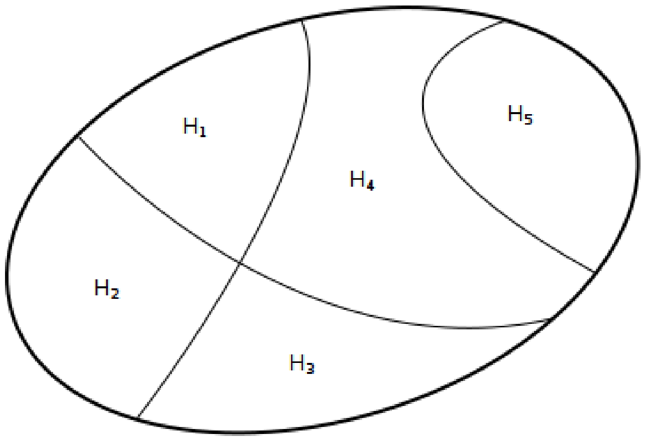

Figure 10: Example of how a hypothesis family can be divided into subfamilies.

of all p families $H_1, \dots, H_p$, i.e., $\mathcal{F} = \text{conv}(\cup_{k=1}^{p} H_k)$. With the ensemble family $\mathcal{F}$, predictor functions $f \in \mathcal{F}$ can be formed as $f = \sum_{t=1}^{T} \alpha_t h_t$, where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_T)$ is in the simplex Δ of the convex hull, and h_t is in H_{k_t} for some $k_t \in [1, p]$, where $t \in [1, T]$, for some T. T gives room for flexibility when composing f. For convenience, the assumption that $\mathfrak{R}_m^G(H_1) \leq \mathfrak{R}_m^G \leq \dots \leq \mathfrak{R}_m^G(H_p)$ is made.

Now that the predictor family H can be decomposed, it is also possible to decompose the feature function $\Psi(x, y)$, giving

$$\Psi = \begin{bmatrix} \Psi_1 \\ \vdots \\ \Psi_p \end{bmatrix}$$

The decomposition of Ψ also means that the feature vector and inherently the feature weight vector w can be decomposed, where

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_k \\ \vdots \\ w_p \end{bmatrix}$$

Here w_k is the feature weight vector associated with the subfamily H_k. Depending on interpretation, $\boldsymbol{w}$ can be a matrix or the concatinated vectors w_k.

Generalization bound on decomposed predictor

In order to generalize the voted risk minimization theory, the empirical margin losses are redefined to include a margin term $\tau \leq 0$, giving:

$$\hat{R}_{S,\rho,\tau}^{add}(h) = \mathop{\mathbb{E}}_{(x,y)\sim S} \left[\Phi^* \left(\max_{y' \neq y} \mathtt{L}(y', y) + \tau - \frac{1}{\rho} \left[h(x, y) - h(x, y') \right] \right) \right] \tag{3.16}$$

$$\hat{R}^{mult}_{S,\rho,\tau}(h) = \mathop{\mathbb{E}}_{(x,y)\sim S}\left[\Phi^*\left(\max_{y'\neq y} \mathtt{L}(y',y)\big(1+\tau-\frac{1}{\rho}\big[h(x,y)-h(x,y')\big]\big)\right)\right] \tag{3.17}$$

With these generalized empirical margin losses, it is now possible to provide a bound on the generalization error based on the ensemble family $\mathcal{F}$.

Theorem 3
Fix $\rho > 0$. For any $\delta > 0$, with probability at least $1-\delta$ over the draw of a sample S of size m, each of the following inequalities holds for all $f \in \mathcal{F}$:

$$R(f) - \hat{R}^{add}_{S,\rho,1}(f) \leq \frac{4\sqrt{2}}{\rho}\sum_{t=1}^{T}\alpha_t \mathfrak{R}^G_m(H_{k_t}) + C(\rho, M, c, m, p),$$

$$R(f) - \hat{R}^{mult}_{S,\rho,1}(f) \leq \frac{4\sqrt{2}M}{\rho}\sum_{t=1}^{T}\alpha_t \mathfrak{R}^G_m(H_{k_t}) + C(\rho, M, c, m, p),$$

where $C(\rho,M,c,m,p) = \frac{2M}{\rho}\sqrt{\frac{\log p}{m}} + 3M\sqrt{\left\lceil \log(\frac{c^2\rho^2 m}{4\log p})\right\rceil \frac{\log p}{m} + \frac{\log\frac{2}{\delta}}{2m}}$. and $|\mathcal{Y}| = c < +\infty$.

Theorem 3 shows that it is possible to bound the generalization error using the empirical margin losses together with a mixture of the factor graph Rademacher complexities of each subfamily in a decomposition and a factor based on the loss function and size of the output space.

The results up until now all provide a basis for an algorithm that utilizes the key ideas of voted risk minimization in a structured prediction setting. However, as the results hold for arbitrary hypothesis families and loss functions, it is necessary to introduce *surrogate loss functions* before formulating this algorithm.

Surrogate loss function

Since the functions h and $L(h(x),y)$ are arbitrary on their domains, it is not guaranteed that the mapping $h \mapsto \mathtt{L}(h(x),y)$ is a convex function (this is typically not the case). If the mapping is not a convex function, optimizing using gradient-descent, or other optimization algorithms, leads to computationally hard problems. Therefore, it is possible to formulate a surrogate loss function which establishes computational tractability by being convex.

Lemma 1 (General surrogate loss function)
For any $u \in \mathbb{R}_+$, let $\Phi_u : \mathbb{R} \to \mathbb{R}$ be an upper bound on $v \mapsto u\mathbf{1}_{v\leq 0}$. Then, the following upper bound holds for any $h \in \mathcal{H}$ and $(x,y,) \in \mathcal{X},\mathcal{Y}$,

$$\mathtt{L}(\mathtt{h}(x),y) \leq \max_{y\neq y'} \Phi_{\mathtt{L}(y',y)}(h(x,y)-h(x,y')). \tag{3.18}$$

In other words, a convex function Φ can be constructed such that it provides an upper bound on the loss function that is usable for optimization.

Choosing the surrogate loss function can now be used to define many of the common structured prediction algorithms, as listed in Table 4. As noted by Cortes et al., investigating other choices of surrogate loss functions can lead to new structured prediction algorithms, some of which they also discuss.

Table 4 Example of surrogate loss functions that define common state-of-the-art structured prediction algorithms.

$\Phi_u(v) = \max(0, u(1-v))$	StructSVM
$\Phi_u(v) = \max(0, u-v)$	Max-Margin Markov Networks
$\Phi_u(v) = \log(1+e^{u-v})$	Conditional Random Fields

VCRF optimization problem

Now all pieces are in place to define a structured prediction algorithm based on the findings presented in this chapter. As with many other machine learning problems, it is possible to encapsule the theory presented in an optimization problem, shown in Equation 3.19. This is achieved by using the surrogate loss function $\Phi_u(v) = \log(1+e^{u-v})$ used to define CRF models and combining this formulation with the voted risk minimization theory. The learning algorithm based on the optimization problem presented in Equation 3.19 is referred to as VCRF.

Given an input space $\mathcal{X}$, output space $\mathcal{Y}$, a sample S of size m, a loss function $\mathtt{L}$, predictor families $H_1, \dots, H_p$ and parameters λ, β, the VCRF algorithm is formulated as the learning algorithm based on the optimization problem 3.19. The goal is to minimize the feature weight vector w over the samples (x_i, y_i) whilst keeping a penality on families H_k based on their factor graph complexity captured and influence of features connected to family p.

Definition 14 (Voted Conditional Random Field)
The Voted Conditional Random Field *algorithm is the procedure which solves the optimization problem*

$$\min_{\boldsymbol{w}} \frac{1}{m} \sum_{i=1}^{m} \log\left(\sum_{y\in\mathcal{Y}} e^{L(y,y_i) - \boldsymbol{w}\cdot(\boldsymbol{\Psi}(x_i,y_i) - \boldsymbol{\Psi}(x_i,y))} \right) + \sum_{k=1}^{p} (\lambda r_k + \beta) \|\boldsymbol{w_k}\|_1, \tag{3.19}$$

with $r_k = r_\infty |F(k)| \sqrt{\log N}$. *The function* $F(k)$ *here denotes the factor graph connected to family* H_k *and* $r_\infty = \max_{i,f,y} \|\Psi_f(x_i, y)\|_\infty$.

The first term tries to find a vector w that maximizes the predictability of classes given all features, essentially the empirical loss term. The second term tries to compensate by weighting size of the feature vector w_k and complexity penalty r_k associated with each family H_k against each other. In other words, a high complexity penalty gives little penalty if few features are considered useful. If a high complexity family H_k has high weights associated with its features, those features will either be toned down or replaced by a family H_j with $r_j < r_k$. Worth noting here is that each r_k can be precomputed as it only depends on the factor graph and size of the feature vector. The form of Equation 3.19 can be compared to that of the risk regularization from Section 2.2 in Equation 2.11.

This optimization problem can be solved using the standard gradient-descent algorithm. As a result, it is necessary to be able to formulate a gradient of Equation 3.19 that can be efficiently computed. Cortes et al. show a general formulation of this gradient, noting that it is exponential in the size of the ensemble family simplex. However, they provide, amongst others, a formulation of the gradient that is computationally efficient given that the loss function is *Markovian.* A Markovian loss function is defined as

$$\mathtt{L}(y, y') = \sum_{t=1}^{l} \mathtt{L}_t(y_{t-p+1}^t, y'^t_{t-p+1}).$$

This property shows a decomposition similar to the features, and essentially limits the loss function L to local interactions for each label in the output given some boundary. In other words, the loss of a label cannot depend on an arbitrary subset of the output. The authors also provide a gradient which is completely agnostic of the loss function, by constructing a weighted finite automaton where states are output labels.

Complexity penalty

Cortes et al. presents an upper bound on the complexity term r_k for linear hypotheses in H_1, bound by $O(\sqrt{\log(N)\max_i |F_i|^2 d_i/m})$, where $d_i = \max_{f\in F_i} |\mathcal{Y}_f|$, i.e., the maximum number of labels associated with the neighbourhood of a factor node f. This bound can be improved upon, e.g., when binary indicator functions are used. Worth noting is that this bound is dependent on the given sample S as well as the feature modelling.

VCRF results on part-of-speech tagging

Cortes et al. also present experimental results on using the VCRF algorithm to perform part-of-speech tagging. They compare VCRF against L_1-regularized CRF on 10 datasets over 8 languages, including English, Chinese, Finnish and a Twitter dataset. Their results show that the VCRF outperforms CRF on 7 out of 10 datasets on both single labels (or tokens) and sentences, with a 5% confidence level using 20% random noise added to the training set. These improvements are of varying sizes, from 22.50 ± 1.57 to 19.82 ± 0.69 for the Tamil sentence error compared with 5.51 ± 0.06 to 5.51 ± 0.04 for English tokens. However, the most interesting result is the average number of features used by VCRF compared to that of CRF. The largest difference is seen in the Basque dataset where VCRF uses 7028 features against CRFs 94712653, a ratio of 0.00007. A sparse feature vector with few active features opens up for optimization in terms of both memory utilization and computational efficiency. As the length of a feature vector can be enormous for certain applications, this is a promising result.

4 Modelling Link Prediction in Knowledge Bases using VCRF

This chapter proposes a model for link prediction in knowledge bases. This includes modelling a structured prediction problem in general and accompanying this with a couple of feature families. A complexity analysis of these feature families presented according to the VCRF theory presented in Section 3.2. The proposed model is an attempt to combine the idea of balancing hypothesis classes of different complexity with the idea that latent feature and graph feature models complement each other.

4.1 Prediction Problem

There are several ways to model link prediction, but in this thesis the focus lies on the prediction off a single triple $y_{ijk} = (e_i, r_k, e_j)$. The much larger task of extending a knowledge base with missing facts can be done by performing single link predictions for all pairs (e_i, e_j). Now, since the goal is to predict y_{ijk} given a set of such triples, the first observation is that the structural information provided by the input and output should be equal. However, giving a whole knowledge base as input x and expecting a triple y_{ijk} back as output seems rather artificial. It is necessary for the input to specify exactly which entities that are of interest, i.e., the tuple (e_i, e_j). However, there is little structure in these input and output spaces to base a prediction on. In order to include the structural information available in the given knowledge base, this information is utilized implicitly is made available to the feature families to utilize instead. An alternative definition would be that the input would be $x = ((e_i, e_j), N_{KB,L}(e_i, e_j))$, where $N_{KB,L}$ is defined as the subset of triples reachable from e_i and e_j given a length L. Here L would be determined by the parameters used by the feature families, demarcating subset of the knowledge base they need access to in order to identify features. As will be seen below, this could, e.g., be determined by the length of the paths used by the PRA algorithm.

In this chapter, a simple 0-1 loss function comparing triples will be used. There are more complex loss functions to consider, either by looking at pure graph features such as the degrees of entity nodes or by including knowledge about the hierarchical structure within entities.

4.2 Feature Families

There are five feature families presented here; Two based on previous work, PRA and RESCAL, and two simple families based on neighbourhoods. These are built on the feature families $H_L^{\mathtt{PRA}}$,$H^{\mathtt{RESCAL}}$, $H_{k_1}^{\mathtt{ENT}}$ and $H_{k_2}^{\mathtt{REL}}$. Following are the descriptions and complexity analysis of each family. The complexity analysis is based on the complexity bound given in Section 3.2 as $O\left(\sqrt{\log(N) \max_i |F_i|^2 \max_{f \in F_i} |\mathcal{Y}_f|/m}\right)$.

PRA features

The PRA family is based on the PRA algorithm presented in Section 3.1. Given $x = (e_i, e_j)$, this family parameterized on the path length L. The features are binary indicators of sequences of relationships, but can easily be extended to the original probabilistic floating point values (See Section 3.1 for discussion). Equation 4.1 defines this family using P_L, a procedure to enumerate all paths between two entities.

$$H_L^{\texttt{PRA}} = \left\{x \to \mathbf{1}_{p' \in P_L(e_i, e_j)} : p' \in R^L\right\} \tag{4.1}$$

Given $x = (\textit{Ford}, \textit{Star Wars})$ and $y = (\textit{Ford}, \textit{starredIn}, \textit{Star Wars})$, the following feature $h_1 \in H_2^{\texttt{PRA}}$ will be active

$$h_1(x) = \mathbf{1}_{p' = (\textit{Ford}, \textit{Star Wars}, \textit{starredIn})}(x)$$

where p' encodes the relationships on the path via *Han-Solo*, i.e., *(played, charIn)*, as given in Figure **??**. $H_L^{\texttt{PRA}}$ can be defined to include all paths of length $l \leq L$, but here it is defined to use paths of $l = L$. This is similar to how n-grams are defined by Cortes et al.

The factor graph will contain one factor for each path in P_L, since calculating a feature will depend on all r_i of a path. The factors will therefore connect to each r_i. One example would be an actor playing two roles in a movie, e.g., John Cleese playing both the character Sir Lancelot the Brave and the Black Knight in Monty Python and the Holy Grail. This example is shown in Figure 11. Here there will be two paths from Cleese to the movie, resulting in one factor connected to each path.

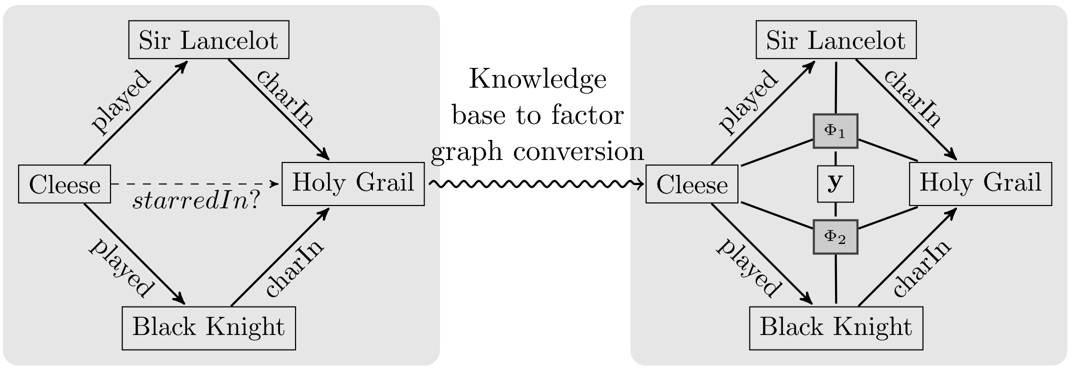

Figure 11: An example of what the PRA feature family factor graph looks like. Here the variable $\boldsymbol{y}$ is used to denote the link prediction between Cleese and Holy Grail.

Since there can be more than one intermediate path between two entities, as with the example of Cleese, there can be more than one feature active. This gives $\max\limits_{f \in F_i} |\mathcal{Y}_f| = |P_L(e_i, e_j)|$. P_L can be computed using the standard breadth-first-search algorithm with goal state e_j, or using the random sampling method suggested by

PRA [15]. The number of paths will also give the number of factor nodes, as $|F_i| = |P_L(e_i, e_j)|$, of the factor graph. The feature vector will contain one entry for each possible path, i.e., one entry for each righthand side of the Horn clauses described in Section 3.1 by, e.g., Equation 3.7. This gives $|R|^L$ number of features.

RESCAL latent features

In order to include latent features, a family based on the RESCAL model is proposed. This is formulated as exactly the feature function of RESCAL:

$$H^{\texttt{RESCAL}} = \phi_{ij}^{RESCAL} \tag{4.2}$$

where the latent features of e_i and e_j are given as parameters to, e.g., the feature function Ψ during evaluation. These parameters are trained in parallel with the weights. Since it is possible to formulate RESCAL training as a stochastic gradient descent problem, this extension to the VCRF problem is possible.

The RESCAL factor graph will only depend on the entities e_i and e_j given by the input x, since the features are calculated using the latent variables $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$. This gives a factor graph size of $|F_i| = 1$. Since the features are all pairwise interactions between the latent variables of e_i and e_j, the maximum number of active features $|\mathcal{Y}_f|$ is going to be all H_e^2 pairwise interactions $\boldsymbol{e}_{ix}\boldsymbol{e}_{jy}$.

The RESCAL features $\phi_{ij}^{\texttt{RESCAL}}$ should be evaluated for each r_k, but this is not encoded by Equation 4.2. Instead, this behaviour will be a result of the interaction between $H^{\texttt{RESCAL}}$and $H_{k_1}^{\texttt{ENT}}$. This gives the feature vector length H_e^2. As will be shown later, the interaction with $H_{k_1}^{\texttt{ENT}}$will effectively set all features not associated with the r_k given by y to 0. In other words, giving only the equation $f_{ijk}^{RESCAL} = W_k^\top \phi_{ij}^{\texttt{RESCAL}}$ influence in scoring the pair (x, y). Each weight matrix W_k is then vectorized to provide a feature weight vector of the same size.

Entity adjacency

The third family is based on the idea that relationships can be predicted by looking at what type of entities surrounds a relationship. Looking at the example in Figure 12 , the *starredIn*-relationship is surrounded by an actor and a movie.

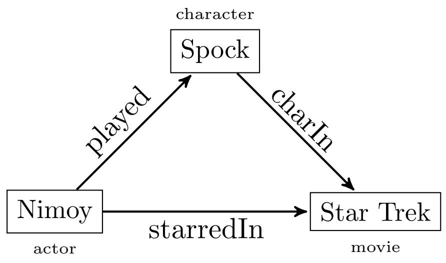

Figure 12: Illustration of entity types that can be used as features. For $k_1 = 1$, $H_{k_1}^{\texttt{ENT}}$activate the features corresponding to *character*, *actor* and *movie.*

For example, connecting a beverage with a movie via *starredIn* could be ruled less likely than via *seenIn*, soley using the types of the entities. This information can be encoded as features, as shown in Equation 4.3.

$$H_{k_1}^{\mathtt{ENT}} = \left\{x \to 1_{t' \in N_e(x,k_1)} : t' \in T_e\right\} \tag{4.3}$$

Allowing a neighbourhood including entities not directly connected by the relationship could be considered a semi-latent modelling, which is why a parameter k_1 is introduced. Equation 4.3 defines this family using a procedure $N_t(x, k_1)$. $N_t(x, k_1)$ computes all entity types reachable from e_i, e_j of x by at most k_1 steps. For $k_1 = 0$, this gives the types of e_i, e_j. T_e denotes the set of all possible entity types, which is also the number of features needed.

The maximum number of active features depend on the number of entities reachable from e_i, e_j. These entities can be computed using a graph exploration algorithm $\mathtt{REACH}_t(k_1)$, returning all types found within k_1 steps. One such algorithm is a modified version of the depth-limited depth-first-search algorithm, remembering all types traversed. The number of factor nodes for $H_{k_1}^{\mathtt{ENT}}$also depends on the size of $\mathtt{REACH}_t(k_1)$.

Relationship adjacency

The idea of entity type neighbourhoods can also be applied to relationship types. The definition of the family $H_{k_2}^{\mathtt{REL}}$is analogue to that of $H_{k_1}^{\mathtt{ENT}}$, where $N_r(y, k_2)$ denotes the set of all relationship types reachable from y in k_2 steps. Here $k_2 = 0$ gives the type of the relationship in y.

$$H_{k_2}^{\mathtt{REL}} = \left\{y \to 1_{r' \in N_r(y,k_2)} : r' \in R\right\} \tag{4.4}$$

The number of active features is bound by $\mathtt{REACH}_r(k_2)$, i.e., the set of all r_i's reachable from y. The number of factor nodes is also $|\mathtt{REACH}_r(k_2)|$. Since there is one feature per relationship type, the number of total features is $|R|$.

4.3 Combined Family

This section contains a proposed joint feature family based on the families given in Section 4.2, as well as an analysis of the complexity factors r_k used in the optimization problem of VCRF (see Equation 3.19).

Table 5 summarizes all terms used to compute the complexity penalty for the proposed feature families. All terms are presented in the description of each feature family, albeit not always with formal notation.

Table 5 Factor graph and feature vector complexity components summarized for all families.

	Max. active features	Factor nodes	Feature vector length
$H_L^{\mathtt{PRA}}$	$\lvert P_L\rvert$	$\lvert P_L\rvert$	$\lvert R\rvert^L$
$H^{\mathtt{RESCAL}}$	H_e^2	1	H_e^2
$H_{k_1}^{\mathtt{ENT}}$	$\lvert\mathtt{REACH}_t(k_1)\rvert$	$\lvert\mathtt{REACH}_t(k_1)\rvert$	$\lvert T_e\rvert$
$H_{k_2}^{\mathtt{REL}}$	$\lvert\mathtt{REACH}_r(k_2)\rvert$	$\lvert\mathtt{REACH}_r(k_2)\rvert$	$\lvert R\rvert$

The complexity penalties for each feature family is formulated in Table 6. The complexities are given as the boundary presented in Section 3.2, $O\big(\sqrt{\log(N)\max_i |F_i|^2 \max_{f\in F_i} |\mathcal{Y}_f|/m}\big)$.

The parameterization of `REACH` and P_L means that the largest factor graph out of the samples is chosen to contribute towards the family complexity penalty.

Table 6 Complexity penalties for each feature family. The functions `REACH` and P_L are parameterized on training samples.

Feature family	Complexity penality
$H_L^{\texttt{PRA}}$	$O(\sqrt{L\log(\lvert R\rvert)\max_i(\lvert P_{L,i}\rvert^2\lvert P_{L,i}\rvert)/m})$
$H^{\texttt{RESCAL}}$	$O(\sqrt{2\log(H_e)H_e/m})$
$H_{k_1}^{\texttt{ENT}}$	$O(\sqrt{\log(\lvert T_e\rvert)\max_i(\lvert\texttt{REACH}_{t,i}(k_1)\rvert^2\lvert\texttt{REACH}_{t,i}(k_1)\rvert)/m})$
$H_{k_2}^{\texttt{REL}}$	$O(\sqrt{\log(\lvert R\rvert)\max_i(\lvert\texttt{REACH}_{r,i}(k_2)\rvert^2\lvert\texttt{REACH}_{r,i}(k_2)\rvert)/m})$

The complexities presented in Table 6 all depend on the parameters of each feature family. The complexities of $H_{k_1}^{\texttt{ENT}}$and $H_{k_2}^{\texttt{REL}}$differ on the number of features but otherwise exert the same form. The $H^{\texttt{RESCAL}}$family competes with $H_{k_1}^{\texttt{ENT}}$for the lowest complexity, however the $H^{\texttt{RESCAL}}$complexity only depends on the formulation of the latent variables whereas $H_{k_1}^{\texttt{ENT}}$depends on the connectivity of the given knowledge base. The complexity of $H_L^{\texttt{PRA}}$also depends on the connectivity of the knowledge base. The terms $\max_i(|P_{L,i}|^2|P_{L,i}|)$ of $H_L^{\texttt{PRA}}$and $\max_i(|\texttt{REACH}_{r,i}(k_2)|^2|\texttt{REACH}_{r,i}(k_2)|)$ of $H_{k_2}^{\texttt{REL}}$will most likely be the determining factors in determining which family has the highest complexity. As $\texttt{REACH}_{r,i}(k_2))$ depends on $|R|$, the penality is bound by $O(\sqrt{\log(|R|)|R|^2)/m})$. Since $|P_L|$ depends on R,the maximum degree of an entity in the knowledge base and the length L, this term will grow more rapidly for dense knowledge bases than the complexity of $H_{k_1}^{\texttt{ENT}}$.

The joint feature vector will be all possible combinations of the features from each family H_k. The length of this vector can therefore be written as

$$N = N_{\texttt{PRA}}N_{\texttt{RESCAL}}N_{\texttt{ENT}}N_{\texttt{REL}} \tag{4.5}$$

Using the sizes outlined in Table 5, Equation 4.5 can be rewritten as

$$N = |R|^{(L+1)}|T_e|H_e^2 \tag{4.6}$$

The English part of DBPedia contains 6 million entities from 754 classes with 1379 relationship types [44]. Using the $L = 4$ and 10 latent features, this translate into a vector of size $N \approx 10^{20}$.

5 Discussion

The proposed model for link prediction in knowledge bases shows some interesting properties. This chapter contains a short discussion of these.

5.1 Goals

All goals presented in Section 1.1 of Chapter 1 have been achieved. Introducing of the regularization of feature family based on the family complexity to the link prediction problem is to the best of the authors knowledge a theoretical advancement. The previous approach where latent and graph features where combined to achieve stronger predictions fit nicely into this theoretical framework. This provides insight into the effects of how and why such combinations can improve the quality of the predictions. To some extent the combination of PRA and RESCAL described in Section 3.1 implicitly exploits the difference in feature family complexity. It is however beneficial to model this relationship explicitly, as outlined in this thesis, in order to achieve higher transparancy of the inner workings of machine learning algorithms. This is one step closer to reducing the notion of black box algorithms.

The third goal of modelling link prediction for VCRF is achieved in Chapter 4, and the practicality of the proposed model is discussed in Section 5.2 below.

5.2 Feature Families

The simple models proposed in addition to the established formulations show some promise. However, it is clear that there are issues with the multiplicative combination of feature families. Even if the idea is that VCRF will lower the amount of features active on average to achieve sparsity, the length of the feature vector becomes alarmingly large for general knowledge bases. The dominating component is the number of relationship types. This suggests that the approach taken here is less favorable in the general case but can be advantageous in use case-specific problems where $|R|$ can be limited.

One problem with how the model is formulated is that the RDF type hierarchies seldom provide a straight answer to, e.g., the number of existing entity types. This opens up for both problems and opportunities. It gives room for use case-specific model training with a narrow scope. However, it is problematic since a demarcation of the type hierarch might not be straightforward and does not fully utilize the strengths of linked data and RDF.

The reformulation of RESCAL in Section 4.2 downplays the strength of the RESCAL algorithm. It is unclear how it will perform when the RESCAL-ALS optimization is affected by the regularization of $\boldsymbol{w}_k$.

Over all, the features families formulated in Section 4.2 are rather naïve, resulting in the large feature vectors. However, clever implementations can perhaps overcome some of these issues by, e.g., not storing the whole vector but using a hashtable for each active feature. It is also possible to use less expressive features. Instead of having one feature for each possible path between two entities, the PRA features family could be restricted to one feature for each relationship type found on any such path. This would alleviate the exponential size $|R|^L$, giving a much more resonable size $|R|$. Exploring the effect of such simplifications is something that needs to be expolored with experiments.

Another reason that the feature vector becomes large is that the feature families suffer from a multiplicative combination. It should be investigated whether an additive combination could be possible.

Finally, it is not necessary to include all feature families in the combined family. There is some overlap in what information the feature families encode such as, e.g., the family $H_{k_2}^{\mathtt{REL}}$having active features corresponding to paths given by $H_L^{\mathtt{PRA}}$for some values of L and k_2. Therefore, it is necessary to investigate, both experimentally and theoretically, how this overlap can be (1) modelled and (2) utilized.

6 Conclusions and Future Work

This thesis outlines a combination of highly theoretical and practical results as an approach to model machine learning problems. The research into the dynamics of link prediction in multirelational data sets provides a good introduction to the field. The proposed model captures features relevant to performing predictions. However, there is no guarantee that all of the proposed feature families allows VCRF to training efficiently.

The most important extension to the work presented here is an implementation of the proposed model. This model should be tested on knowledge bases with different properties in order to evaluate the strengths and weaknesses. In order to compare the results with existing solutions, testing on Freebase and YAGO are two options high on the list. The model should also be tested on mono-relational data, to further evaluate the weaknesses described in Chapter 5.

The largest flaw of the work of this thesis is that the formulation and analysis of the model gradient has been left out. The gradient used in gradient descent to solve the VCRF problem has a couple of critera that must be fulfilled to allow efficient optimization. Here a deeper analysis of the feature families is necessary.

The factor graph Rademacher complexity is a powerful tool to analyse structured prediction problems and algorithms. However, as the theory is state-of-the-art, the application of it is not straightforward. Continued work should make an effort towards aligning notation and theory with common formulations and algorithms in structured prediction. Also, continued work should conform more existing structured prediction problems with the VCRF problem formulation. More practical evaluations could provide more insight into many other machine learning problems.

Finally, a more comprehensive investigation into feature families should be performed. There are many other existing feature families for, e.g., latent features that should be evaluated. Looking at the formulation of the templated Markov Logic CRF approach described in Section 3.1, such templating mechanisms can be applied to other formal systems. Especially interesting are formal semantics.

References

[1] IBM, “What is big data?.” http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[2] A. Gandomi and M. Haider, “Beyond the hype: Big data concepts, methods, and analytics,” *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[3] W. W. W. Consortium, “Rdf - semantic web standards,” 2014.

[4] T. Berners-Lee, J. Hendler, O. Lassila, *et al.*, “The semantic web,” *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[5] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, *et al.*, “Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia,” *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[6] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, “A review of relational machine learning for knowledge graphs,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2016.

[7] G. Weikum and M. Theobald, “From information to knowledge: Harvesting entities and relationships from web sources,” in *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS ’10, (New York, NY, USA), pp. 65–76, ACM, 2010.

[8] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, “Knowledge vault: a web-scale approach to probabilistic knowledge fusion,” in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’14, New York, NY, USA - August 24 - 27, 2014* (S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, eds.), pp. 601–610, ACM, 2014.

[9] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: a core of semantic knowledge,” in *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706, ACM, 2007.

[10] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: A collaboratively created graph database for structuring human knowledge,” in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’08, (New York, NY, USA), pp. 1247–1250, ACM, 2008.

[11] A. Singhal, “Introducing the knowledge graph: things, not strings.” https://googleblog.blogspot.se/2012/05/introducing-knowledge-graph-things-not.html, 2012.

[12] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin, “Learning structured prediction models: A large margin approach,” in *Proceedings of the 22Nd International Conference on Machine Learning*, ICML ’05, (New York, NY, USA), pp. 896–903, ACM, 2005.

[13] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning.* Adaptive Computation and Machi, MIT Press, 2007.

[14] C. Cortes, V. Kuznetsov, M. Mohri, and S. Yang, “Structured prediction theory based on factor graph complexity,” in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 2514–2522, Curran Associates, Inc., 2016.

[15] M. Nickel, V. Tresp, and H.-P. Kriegel, “A three-way model for collective learning on multi-relational data,” in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 809–816, ACM, 2011.

[16] C. Sutton and A. McCallum, “An introduction to conditional random fields,” *Found. Trends Mach. Learn.*, vol. 4, pp. 267–373, Apr. 2012.

[17] C. Cortes, P. Goyal, V. Kuznetsov, and M. Mohri, “Kernel extraction via voted risk minimization,” in *Proceedings of the 1st Workshop on Feature Extraction: Modern Questions and Challenges, FE 2015, co-located with the 29th Annual Conference on Neural Information Processing Systems (NIPS 2015), Montreal, Canada, December 11-12, 2015.*, pp. 72–89, 2015.

[18] Wikimedia Commons, “Loss function surrogates,” 2014. File: `Loss_function_surrogates.svg`.

[19] V. N. Vapnik and V. Vapnik, *Statistical learning theory*, vol. 1. Wiley New York, 1998.

[20] U. von Luxburg and B. Schölkopf, “Statistical learning theory: Models, concepts, and results.,” in *Inductive Logic* (D. M. Gabbay, S. Hartmann, and J. Woods, eds.), vol. 10 of *Handbook of the History of Logic*, pp. 651–706, Elsevier, 2011.

[21] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning.* The MIT Press, 2012.

[22] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML ’01, (San Francisco, CA, USA), pp. 282–289, Morgan Kaufmann Publishers Inc., 2001.

[23] H. M. Wallach, “Conditional random fields: An introduction,” 2004.

[24] R. Klinger and K. Tomanek, "Classical probabilistic models and conditional random fields," 2007.

[25] X. Zhu, "Cs838-1 advanced nlp: Conditional random fields," tech. rep., Department of Computer Sciences, The University of Wisconsin-Madison, 2007.

[26] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[27] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty, "Building watson: An overview of the deepqa project," *AI Magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[28] G. Doda, "Manipal hospitals announces national launch of ibm watson for oncology." https://www-03.ibm.com/press/in/en/pressrelease/50290.wss, 2016.

[29] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1, pp. 107–136, 2006.

[30] M. Nickel, X. Jiang, and V. Tresp, "Reducing the rank in relational factorization models by including observable patterns," in *Advances in Neural Information Processing Systems*, pp. 1179–1187, 2014.

[31] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[32] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[33] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing yago: Scalable machine learning for linked data," in *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, (New York, NY, USA), pp. 271–280, ACM, 2012.

[34] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[35] W. Liu and L. Lü, "Link prediction based on local random walk," *EPL (Europhysics Letters)*, vol. 89, no. 5, p. 58007, 2010.

[36] N. Lao, T. Mitchell, and W. W. Cohen, "Random walk inference and learning in a large scale knowledge base," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 529–539, Association for Computational Linguistics, 2011.

[37] M. Gardner, A. Bhowmick, K. Agrawal, and D. Dua, "Experimenting with the path ranking algorithm," 2015.

[38] M. Gardner, "PRA (Path Ranking Algorithm) and SFE (Subgraph Feature Extraction)." https://github.com/matt-gardner/pra.

[39] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 57–66, 2015.

[40] N. Srebro and S. Ben-David, "Learning bounds for support vector machines with learned kernels," in *International Conference on Computational Learning Theory*, pp. 169–183, Springer, 2006.

[41] H. C. Daume, III, *Practical Structured Learning Techniques for Natural Language Processing.* PhD thesis, University of Southern California, Los Angeles, CA, USA, 2006. AAI3337548.

[42] A. Mueller, "User guide - pystruct 0.2.4 documentation," 2013. [Online; Accessed 12-May-2017].

[43] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *Journal of machine learning research*, vol. 6, no. Sep, pp. 1453–1484, 2005.

[44] DBpedia, 2016. Release notes, DBpedia version 2016-04.