The Classification Using the Hierarchy and Exclusion Graphs

by

Temirlan Raimbekov

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Master of Science in Applied Mathematics

at the

NAZARBAYEV UNIVERSITY

May 2020

Author . . . . . .
Department of Mathematics
May 9, 2020

Certified by . . . . . .
Rustem Takhanov
Assistant Professor
Thesis Supervisor

Accepted by . . . . . .
Daniel Pugh
Dean, School of Sciences and Humanities

The Classification Using the Hierarchy and Exclusion Graphs

by

Temirlan Raimbekov

Submitted to the Department of Mathematics
on May 9, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Applied Mathematics

Abstract

This thesis first reviews the Conditional Random Fields (CRF) model. Then, we introduce the Hierarchy and Exclusion (HEX) graphs and describe the probabilistic classification model based on these graphs (HEX model). Next, we demonstrate that the HEX model is a special case of the CRF model. This allows us to train the HEX model using the framework of the CRF model. After that, we explain the algorithm for this process that calculates the marginals (Exact Inference algorithm). The main objective of the research was to design the sparsification and densification steps for the exact inference algorithm. We propose algorithms for these steps. Then, we introduce the betting model that is modified HEX model. We calculate marginals for this model using the Exact Inference algorithm without sparsification and densification steps. After that, we perform the same experiments using these steps. Finally, by estimating the execution time for the experiments we demonstrate that using the sparsification and densification steps in the exact inference algorithm boosts its performance.

Thesis Supervisor: Rustem Takhanov
Title: Assistant Professor

Contents

Chapter 1

Introduction

1.1 Conditional Random Fields

According to Nowozin and Lampert [10], conditional random fields model can be described as

$$p(y|x,w) = \frac{1}{Z(x,w)} e^{-E(x,y,w)}$$

where w is unknown weight vector, $E(x,y,w) = \langle w, \phi(x,y) \rangle$, $Z(x,w) = \sum_{y \in \mathcal{Y}} e^{-E(x,y,w)}$. Let us denote the actual conditional label distribution as $d(y|x)$. Then, our main target, during the learning process, is to find weight vector w^* such that $p(y|x,w^*)$ would be as close to this distribution as it can be. Despite the fact that the true distribution is not known, for learning we can use that $d(x,y)$ has an $i.i.d.$ sample set $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\dots,N}$.

First, let us find such w^* that would make $p(y|x,w^*)$ be close to the true distribution as possible. For this, we compare these two distributions using the relative entropy. It is also known as the Kullback-Leibler divergence that is used to measure the difference between two probability distributions [7]. Thus, $\mathrm{KL}(p||d) = \sum_{y \in \mathcal{Y}} d(y|x) \log \frac{d(y|x)}{p(y|x,w)}$. Next, we find this divergence over all $x \in \mathcal{X}$:

$$\mathrm{KL}_{total}(p||d) = \sum_{x \in \mathcal{X}} d(x) \sum_{y \in \mathcal{Y}} d(y|x) \log \frac{d(y|x)}{p(y|x,w)}$$

Now, we want to reduce this divergence, by finding appropriate w^*:

$$w^* = \underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \mathrm{KL}_{total}(p||d) = \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} \sum_{x\in\mathcal{X}} d(x) \sum_{y\in\mathcal{Y}} d(y|x) \log p(y|x,w)$$

$$= \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} \mathbb{E}_{(x,y)\sim d(x,y)}[\log p(y|x,w)].$$

As it was mentioned before, we have no information about the actual distribution $d(x,y)$. This makes direct calculation of w^* impossible. However, we can use sample set $\mathcal{D} = \{(x^n,y^n)\}_{n=1,...,N}$ for its approximation

$$\approx \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} \sum_{(x^n,y^n)\in\mathcal{D}} \log p(y^n|x^n,w)$$

Since sample set $\mathcal{D} = \{(x^n,y^n)\}_{n=1,...,N}$ is $i.i.d.$

$$= \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} \log \prod_{n=1}^{N} p(y^n|x^n,w) = \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} \prod_{n=1}^{N} p(y^n|x^n,w) = \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} p(y^1,...,y^N|x^1,...,x^N,w).$$

This is called the maximum conditional likelihood (MCL). In fact, MCL has a disadvantage. It overfits the model in the case when the dimensionality of weight vector w is greater than the amount of training data. This is because w^* would significantly vary in relation to the training collection $\mathcal{D}$. To prevent this problem. we can process w as a random variable, not as a parameter. Let us consider the probability of w given sample set $\mathcal{D}$

$$p(w|\mathcal{D}) = \frac{p(w)p(y^1,...,y^N|x^1,...,x^N)}{p(y^1,...,y^N|x^1,...,x^N)} \quad \text{(by Bayes' Theorem)}$$

$$= p(w) \prod_{n=1}^{N} \frac{p(y^n|x^n,w)}{p(y^n|x^n)} \quad \text{(since } \mathcal{D} \text{ is } i.i.d.\text{)}$$

Using the *maximum a posteriori* estimation,

$$w^* = \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} p(w|\mathcal{D}) = \underset{w\in\mathbb{R}^d}{\operatorname{argmax}} p(w) \prod_{n=1}^{N} \frac{p(y^n|x^n,w)}{p(y^n|x^n)} = \underset{w\in\mathbb{R}^d}{\operatorname{argmin}} [-\log(p(w) \prod_{n=1}^{N} \frac{p(y^n|x^n,w)}{p(y^n|x^n)})]$$

$$= \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} [-\log p(w) - \sum_{n=1}^{N} \log p(y^n | x^n, w)].$$

It is not possible to calculate prior $p(w)$ from data. Thus, we make an assumption that this prior is Gaussian, and its mean is zero. As a result, $p(w) \propto \exp(-\frac{||w||^2}{2\sigma^2})$. So,
$-\log p(w) = \frac{||w||^2}{2\sigma^2}$. Let us call $\frac{1}{2\sigma^2}$ as a parameter λ. Therefore, $-\log p(w) = \lambda ||w||^2$
Next, let us take negative log-likelihood of $p(w|\mathcal{D})$,

$$\mathcal{L}(w) = -\log p(w|\mathcal{D}) = -\log p(w) - \sum_{n=1}^{N} \log p(y^n | x^n, w) = \lambda ||w||^2 - \sum_{n=1}^{N} \log \frac{1}{Z(x^n, w)} e^{-\langle w, \phi(x,y) \rangle}$$

$$= \lambda \left\| w \right\|^2 + \sum_{n=1}^{N} \langle w, \phi(x^n, y^n) \rangle + \sum_{n=1}^{N} (x^n, w).$$

We can calculate the gradient of $\mathcal{L}(w)$. The first two terms can be calculated easily

$$\nabla_w \lambda \left\| w \right\|^2 = 2\lambda w,$$

$$\nabla_w \sum_{n=1}^{N} \langle w, \phi(x^n, y^n) \rangle = \sum_{n=1}^{N} \phi(x^n, y^n).$$

To compute the third term we use chain rule,

$$\nabla_w \sum_{n=1}^{N} \log Z(x^n, w) = \sum_{n=1}^{N} \nabla_w \log Z(x^n, w) = \sum_{n=1}^{N} \nabla_w \log \sum_{y \in Y} e^{-E(x^n, y, w)}$$

$$= \sum_{n=1}^{N} \frac{1}{\sum_{y \in Y} e^{-E(x^n, y, w)}} \nabla_w \sum_{y \in Y} e^{-E(x^n, y, w)}$$

$$= \sum_{n=1}^{N} \frac{1}{\sum_{y \in Y} e^{-E(x^n, y, w)}} \sum_{y \in Y} e^{-E(x^n, y, w)} \nabla_w [-E(x^n, y, w)]$$

$$= \sum_{y \in Y} \frac{e^{-E(x^n, y, w)}}{\sum_{y \in Y} e^{-E(x^n, y, w)}} \nabla_w [-E(x^n, y, w)]$$

$$= \sum_{y \in Y} \frac{e^{-E(x^n, y, w)}}{\sum_{y \in Y} e^{-E(x^n, y, w)}} \nabla_w [-\langle w, \phi(x^n, y) \rangle].$$

$$\text{Since } \frac{e^{-E(x^n, y, w)}}{\sum_{y \in Y} e^{-E(x^n, y, w)}} = p(y|x^n, w) \text{ and } \nabla_w [-\langle w, \phi(x^n, y) \rangle] = \phi(x^n, y),$$

$$\nabla_w \sum_{n=1}^{N} \log Z(x^n, w) = -\sum_{y \in Y} p(y|x^n, w) \phi(x^n, y) = -\sum_{n=1}^{N} \mathbb{E}_{y \sim p(y|x^n)}(\phi(x^n, y)).$$

Finally, the gradient of $\mathcal{L}(w)$ looks like this,

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{n=1}^{N} \phi(x^n, y^n) - \sum_{n=1}^{N} \mathbb{E}_{y \sim p(y|x^n)}(\phi(x^n, y))$$

1.2 Hierarchy and Exclusion (HEX) Graphs

Before explaining the model itself, it is important to introduce Hierarchy and Exclusion (HEX) graphs [5]. Because this allows us to explain the concept of labels.

Let us denote the HEX graph as $G = (V, E_h, E_e)$. This G has nodes $(v_1, .., v_n)$ on it and these are elements of set V. Next, the G graph has two types of edges: directed $E_h \subseteq V \times V$ and undirected $E_e \subseteq V \times V$. Thus, the structure of the G is a combination of two graphs: one containing only hierarchy edges $G_h = (V, E_h)$ and another one with only exclusion edges $G_e = (V, E_e)$. G_h is directed acyclic graph (DAG). This means that it has no cycles in its structure [6].

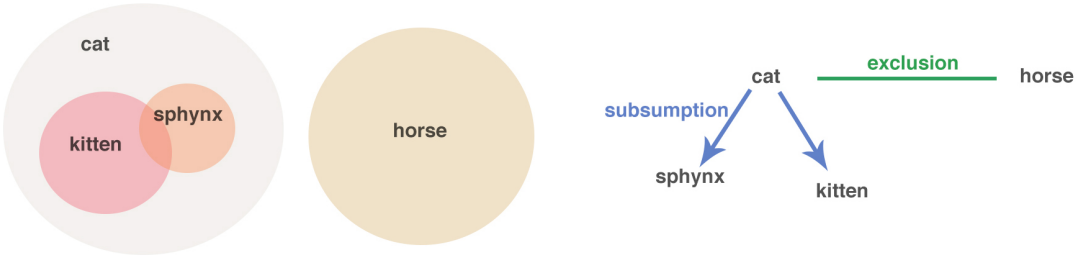

Figure 1. HEX graph example

For each unique label on the graph, there is a special node $v \in V$. An edge $(v_i, v_j) \in E_h$ is called as hierarchy edge. This means that v_i is a "parent" for v_j (e.g. cat subsumes kitten). Next, an edge

$(v_i, v_j) \in E_e$ is called as exclusion edge. This specifies the mutual exclusiveness between nodes in it (e.g. it is impossible to be cat and horse at the same time). Next, it is important to explain the assignments (states) on the edges. Each score receive whether 1 or 0 value ($v_i \in \{0, 1\}$). Every edge considers relation of two scores and have rules on the values received by these scores. For instance, $(v_i, v_j) \in E_h$ indicates that it is impossible to have $(v_i, v_j) = (0, 1)$ assignment (e.g. being a kitten, not being a cat). While, $(v_i, v_j) \in E_e$ specifies that it is not valid to assign $(v_i, v_j) = (1, 1)$ (e.g. horse and cat) for this edge. Thus, we can now deduce that for every assignment $(v_i, v_j) = (0, 1)$ to be legitimate this $(v_i, v_j) \notin E_h$. And, every assignment $(v_i, v_j) = (1, 1)$ is legitimate if this $(v_i, v_j) \notin E_e$. We call the set of all legitimate assignments or states on G graph as $S_G \subseteq \{0, 1\}^n$.
At that point, the above description of the HEX graph is not complete. It is still possible at the same time for two nodes to have a hierarchy and an exclusion edge between them. In this case, the subsumed node always has a 0 score. We can call this as a "dead" label. Therefore, it is necessary to add consistency condition: all nodes in the HEX graph are not allowed to be "dead", i.e. they must be "active". Graphically the consistency condition can be explained as: "all the parents of a node cannot have exclusion edges between each other and between the node itself and its parents". Next, we will introduce the probabilistic model for the classification. This model uses the principles of HEX graphs. Thus, we call it the HEX model.

1.3 HEX Model

$$\tilde{P}(y|x) = \prod_i e^{f_i(x;w)[y_i=1]} \prod_{(v_i,v_j)\in E_h} [(y_i, y_j) \neq (0, 1)] \prod_{(v_i,v_j)\in E_e} [(y_i, y_j) \neq (1, 1)]$$

$\tilde{P}$ is then normalized by partition function $Z(x) = \sum_{\hat{y}} \tilde{P}(\hat{y}|x)$. Thus the normalized probability is $P(y|x) = \tilde{P}(y|x)/Z(x)$. We consider the case when x is a vector and $f_i(x; w) = w_i^T x$.
There are several facts that can describe the model. First, the illegal assignments would give 0 for this probability. Second, the probability for the legal assignment is calculated as taking a sum of the labels with score 1, taking exponent for this, and finally, it is normalized by the partition function. Third, there is consistency between label probabilities and their relations, i.e. an ancestor would always have a higher probability than its descendant, and the sum of probabilities of two nodes

having an exclusion edge between them is not more than 1. Fourth, the model is open to cope with unknown categories. E.g. if we add another subcategory for the cat, the model could assign larger marginal for the cat, but would not assign a larger probability for a known subcategory.

In fact, there is a good feature of the HEX model. The ordinary existing models are combined in its structure. The case when all nodes have exclusion edge between each other, and hierarchy edges are not used, brings us to the softmax: $Pr(y_i = 1|x) = \frac{e^{f_i}}{1+\sum_j e^{f_j}}$. In addition, in the case when there are no edges on the graph, the HEX model have the following structure: $Pr(y|x) = \prod_i \frac{e^{f_i[y_i=1]}}{1+e^{f_i}}$. For each label we would have an independent logistic regressions.

Now, we want to prove that the HEX model is a special case of the CRF model. This would allow us to use gradient calculation for the CRF model on our model, in other words, perform the learning process for the HEX classification model.

1.3.1 HEX is CRF

To show that the HEX model is a special case of the CRF model, it is necessary to define $E(\mathbf{x}, \mathbf{y}, \mathbf{w})$ and $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$.

$$E(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) \rangle, \quad \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \boldsymbol{\phi}_1(\mathbf{x}, \mathbf{y}) \\ \boldsymbol{\phi}_2(\mathbf{x}, \mathbf{y}) \\ \vdots \\ \boldsymbol{\phi}_c(\mathbf{x}, \mathbf{y}) \end{bmatrix}$$

In the HEX model that we proposed,

$$E(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \begin{cases} -\sum\limits_{i:y_i=1} f_i(\mathbf{x}, \mathbf{w}) & \text{y}\sim \text{ HEX} \\ +\infty & \text{y}\nsim \text{ HEX} \end{cases}$$

As a result,

$$E(\mathbf{x}, \mathbf{y}, \mathbf{w}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}, \mathbf{y}) \rangle = -\sum_{i:y_i=1} f_i(\mathbf{x}, \mathbf{w})$$

For $\mathbf{y}$ satisfying the HEX,

$$\sum_{i:y_i=1} \mathbf{w}_i^T \boldsymbol{\phi}_i(\mathbf{x}, \mathbf{y}) = -\sum_{i:y_i=1} \mathbf{w}_i^T \mathbf{x}.$$

Thus, $\boldsymbol{\phi}_i$ function for the model will look like this,

$$\boldsymbol{\phi_i}(\mathbf{x}, \mathbf{y}) = \begin{cases} -\mathbf{x} & y_i = 1 \text{ and } y \sim HEX \\ \mathbf{0} & y_i = 0 \text{ and } y \sim HEX \\ \text{undefined} & \text{otherwise} \end{cases}$$

Since the HEX model is the CRF model. We use formula

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{n=1}^{N} \phi(x^n, y^n) - \sum_{n=1}^{N} \mathbb{E}_{y \sim p(y|x^n)}(\phi(x^n, y))$$

for calculating the gradient of negative log-likelihood of HEX probability classification model. We need to know how to calculate $\mathbb{E}_{y \sim p(y|x^n)}(\phi(x^n, y))$.

Let's consider the case when we have probability distribution over $\mathcal{Y} = \{0, 1\}^n$:

$$p(\mathbf{y}) \sim \tilde{p}(\mathbf{y}) = \prod_i e^{f_i[y_i=1]} \prod_{(v_i,v_j) \in E_h} [(y_i, y_j) \neq (0, 1)] \prod_{(v_i,v_j) \in E_e} [(y_i, y_j) \neq (1, 1)]$$

As can be seen, this is the special case of the CRF model. By taking negative logarithm of this, we get the formula for the energy:

$$E(\mathbf{y}, \mathbf{f}) = \begin{cases} -\sum\limits_{i:y_i=1} f_i & \text{y}\sim \text{ HEX} \\ +\infty & \text{y}\nsim \text{ HEX} \end{cases}$$

Since $E(\mathbf{y}, \mathbf{f}) = \langle \mathbf{f}, \boldsymbol{\phi}(\mathbf{y}) \rangle$, we have $\langle \mathbf{f}, \boldsymbol{\phi}(\mathbf{y}) \rangle = -\sum_{i:y_i=1} f_i$ if $\mathbf{y}$ satisfies the HEX. Thus, we can understand that i^{th} entry of $\boldsymbol{\phi}(\mathbf{y})$:

$$[\boldsymbol{\phi}(\mathbf{y})]_i = \begin{cases} \text{-1} & y_i = 1 \\ 0 & y_i = 0 \end{cases}$$

As a result, we deduce that,

$$\boldsymbol{\phi}(\mathbf{y}) = \begin{cases} -\mathbf{y} & \mathbf{y} \sim \text{HEX} \\ \text{undefined} & \mathbf{y} \nsim \text{HEX} \end{cases}$$

We rewrite our initial formula as:

$$p(\mathbf{y}|\mathbf{f}) = \frac{e^{-\langle \mathbf{f}, \boldsymbol{\phi}(\mathbf{y}) \rangle} \text{HEX}(\mathbf{y})}{\sum_{y \sim \text{HEX}} e^{-\langle \mathbf{f}, \boldsymbol{\phi}(\mathbf{y}) \rangle}}$$

Let's define $L = -\log p(\mathbf{y}|\mathbf{f})$. Therefore, if $\mathbf{y}$ satisfies HEX:

$$L = -\langle \mathbf{f}, -\mathbf{y} \rangle + \log \sum_{y \sim \text{HEX}} e^{-\langle \mathbf{f}, -\mathbf{y} \rangle}$$

Next, we consider:

$$\frac{\partial L}{\partial f_i} = y_i + \frac{\sum_{y \sim \text{HEX}} y_i e^{-\langle \mathbf{f}, -\mathbf{y} \rangle}}{\sum_{y \sim \text{HEX}} e^{-\langle \mathbf{f}, -\mathbf{y} \rangle}} = y_i + \sum_{y \sim \text{HEX}} y_i \, p(\mathbf{y}) = y_i + \mathbb{E}(y_i).$$

Further,

$$\mathbb{E}(y_i) = 0 \, p(y_i = 0) + 1 \, p(y_i = 1) = p(y_i = 1).$$

Thus,

$$\frac{\partial L}{\partial f_i} = y_i + p(y_i = 1).$$

Next, let us define $J(w) = \sum_{i=1}^{n} L_i(w)$. Now, we want to compute ∇J that is $\frac{\partial J(w)}{\partial w_j}$.

$$\frac{\partial J(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{i=1}^{n} L_i(w) = \sum_{i=1}^{n} \frac{\partial L_i(w)}{\partial w_j}$$

By the chain rule,

$$\frac{\partial L_i(w)}{\partial w_j} = \sum_{k=1}^{c} \frac{\partial L_i(w)}{\partial f_k} \frac{\partial f_k}{\partial w_j} = \sum_{k=1}^{c} \frac{\partial(-\log p(y^i|x^i, w))}{\partial f_k} \frac{\partial f_k(x^i, w)}{\partial w_j}$$

$$= \sum_{k=1}^{c} [y_k^i + p(y_k = 1|x^i, w)] \frac{\partial f_k(x^i, w)}{\partial w_j}$$

Thus, we need to be able to calculate the $\frac{\partial f_k}{\partial w_j}$ for the function that we would have in our task. Therefore, the form of f is not crucial. This could be linear, deep neural network etc. Next, it is important to know how to calculate the marginals. For this purpose we introduce the exact inference algorithm in the next section.

1.3.2 Exact Inference Algorithm

Algorithm 1: Exact Inference

Input: Graph $G = (V, E_h, E_e)$
Input: Scores $f \in \mathcal{R}^{|V|}$
Output: Marginals, Pr$(v_i = 1)$
1. $G^* \leftarrow$ Sparsify(G)
2. $\overline{G} \leftarrow$ Densify(G)
3. $T \leftarrow$ BuildJunctionTree(G^*)
4. For each clique $c \in T$, $S_c \leftarrow$ ListStateSpace$(\overline{G}[c])$
5. Perform (two passes) message passing on T using only states S_c for each clique c.

Let us briefly discuss the 5 steps of Algorithm 1. First, sparsification means removing all redundant edges from the original graph G. The concept of redundant edges is explained in the next section. The second step is to densify the G graph. This step is opposite to sparsification, we add as much hierarchy and exclusion edges as possible, without violating the HEX principles. The third step is the junction tree algorithm applied to sparsified G that is G^*. After this step we get

cliques (c), and fourth step performs ListStateSpace for every clique. The final step is needed for implementation of message passing for these cliques.

Chapter 2

Main

The objective of the research is to introduce algorithms for Step 1 and Step 2 of the exact inference algorithm (Algorithm 1). There are publicly available MATLAB codes for the exact inference algorithm [8]. However, sparsification and densification parts are missing, the author used the same G graph for its sparsified G^* and densified $\overline{G}$ versions. Before we proceed to the algorithms let us explain the sparsification and densification processes in detail.

2.1 Sparsification and Densification

Let us start by stating that some HEX graphs can be equivalent. The graphs are considered to be equivalent if their state spaces are equal. In other words, G and G' are equivalent, if S_G is equal to $S_{G'}$. There are two cases that can give rise to equivalent graphs. First, when we have transitivity of the hierarchy relation. For example, we have a hierarchy edge from "mammal" to "cat" and from "cat" to "kitten". As a result, "mammal" should also have a hierarchy edge directed to "kitten". However, this edge is redundant. Another case is when children obtain exclusion edges due to their parents. For example, "cat" and "horse" have an exclusion edge between them. As a result, all children of "cat" should also have exclusion edges with "horse". For instance, there should be an exclusion edge between "sphynx" and "horse", but it would be redundant. To sum up, redundant edges are those that a graph can obtain or get rid of, but its state space would stay the same.

Suppose we have a $G = (V, E_h, E_e)$ graph. If there is a directed edge $e \in V \times V$ such that $G' = (V, E_h \setminus \{e\}, E_e)$ and $G'' = (V, E_h \cup \{e\}, E_e)$ are equivalent to G, then this edge e is redundant. And if there is an undirected edge $e \in V \times V$ such that $G' = (V, E_h, E_e \setminus \{e\})$ and $G'' = (V, E_h, E_e \cup \{e\})$ are equivalent to G, then this edge e is redundant.

There are some patterns that can be used to find redundant edges in a consistent HEX graph. Suppose we have a hierarchy edge (v_i, v_j). It would be redundant if and only if we can find another route from v_i to v_j. For the exclusion edge between two nodes: an edge would be redundant if and only if the parents of the nodes already have an exclusion edge between them or the exclusion edge between the parent of the first node and the second one.

At that point, we understood that it is possible to remove or add redundant edges into a HEX graph without changing its state space. The process of removing redundant edges is called sparsification. By removing these edges one by one we can finally reach the minimally sparse graph. The reverse process to the sparsification, when we add redundant edges, is called densification. By adding redundant edges we can reach the maximally dense graph.

2.2 Algorithms

In this section, we want to propose algorithms for the sparsification and densification of the HEX graph. We are going to consider the sparsification and densification of each G_e and G_h separately. First, we start with the algorithm for the densification of the G_h. The main thing that we do here is to add transitivity of the hierarchy relation. For example, if we have directed edge from a to b, and from b to c. Then, we need to add a directed edge from a to c, if we do not have one. By going all over these cases, our G_h would become maximally dense. The densification of the G_h would help us to sparsify the G_e.

Next, we want to sparsify the G_e. The main algorithm here is that if there is an exclusion edge between a and b, then we are going to eliminate any other exclusion edges between a and b's children and vice versa (between b and a's children). In addition, we eliminate any exclusion edges between their children. To find the descendants of the node, we use the densified version of G_h.

The maximally dense G_h shows every child of nodes. So that we can immediately eliminate exclusion edges.

Next, let us explain the algorithm for the G_h sparsification. Suppose we have a directed edge from a to b and from b to c. Then, if there is a hierarchy edge from a to c, we eliminate it. During the densification process of G_h, we always get the same maximally dense version of the original graph. However, the sparsification of G_h strongly depends on the order. Depending on the order of the elimination, we might have different maximally sparse graphs. For better understanding let us consider a simple example (see Figure 2). Suppose our initial G_h graph has hierarchy edges from a to b, from b to c and from c to d. Additionally it has hierarchy edges from a to d and from b to d. Here, we have two different orders of G_h sparsification, i.e. elimination of redundant directed edges. First, using the mentioned sparsification algorithm on the initial graph G_h, we start by eliminating the directed edge from b to d, since there are hierarchy edges from b to c and from c to d. Then, according to the algorithm, we do not have redundant edges on this graph. Thus, this is the minimally sparse graph that we achieved using this order.

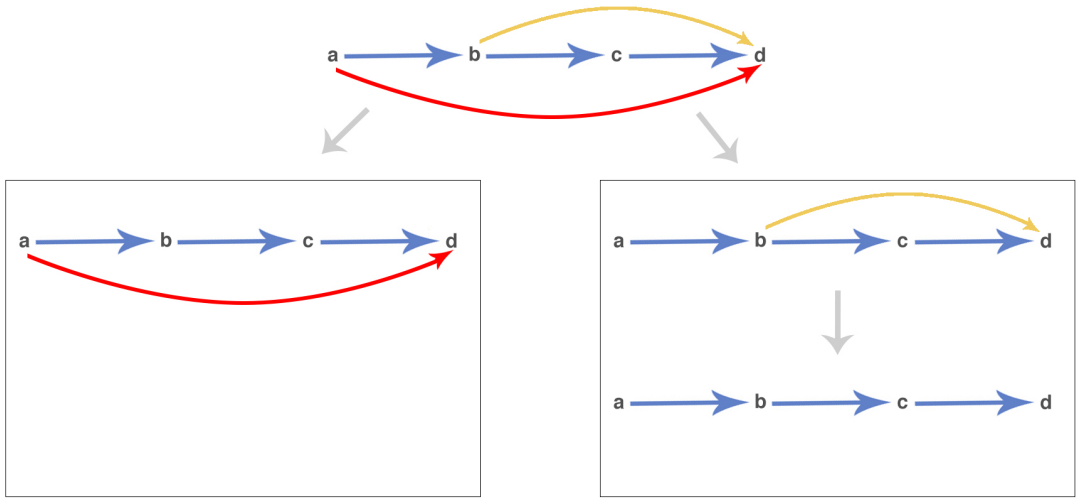

Figure 2. Example 1 of sparsification process

Now, let us try to sparsify the initial graph (from Figure 2) G_h using the different order. First, we eliminate the hierarchy edge from a to d, since there are hierarchy edges from a to b and from b to d. Next, we can make another elimination that is directed edge from b to d, since there are

hierarchy edges from b to c and from c to d. Finally, we have no redundant edges on our graph. However, this graph is more sparse than the graph we achieved in the first case. Therefore, we can see that the order of sparsification might bring us to a different version of sparse graphs. This example brings us to the first principle for the elimination of hierarchy edges. Suppose we have two edges (u, v) and (u', v), where $u < u'$ (u is an ancestor of u'). These are both must be eliminated according to the algorithm. The rule we apply is that in this case we first eliminate edge that has starting node earlier than the other edge. In other words, the longest edge is eliminated first, since both edges have the same ending node. Thus, we first eliminate (u, v) and then (u', v).

Let us consider the second example of the sparsification process (see Figure 3). Here, as well as in the previous example, we can see that the order of edge elimination is important. And, as well as in the first principle, it is better to eliminate the longest edge. However, we cannot use the first principle here, since we do not have the same ending node. In this case, we have the same starting point for both directed edges. Therefore, the second principle is that in this case we first eliminate the edge that has an ending point later than the other edge. So, having two edges (u, v) and (u, v'), where $v < v'$, we first eliminate the (u, v').

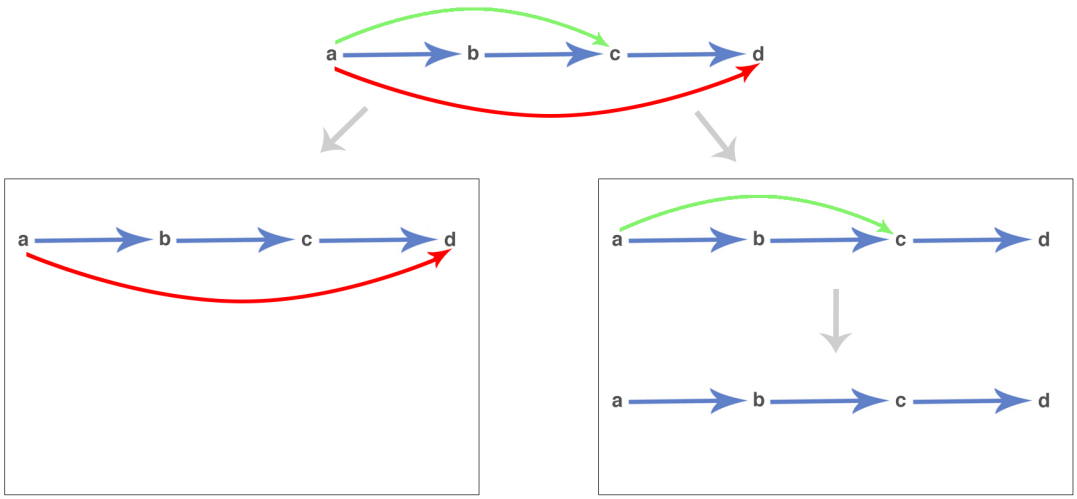

Figure 3. Example 2 of sparsification process

Finally, the last algorithm is the densification of G_e. During this process, we add exclusion edges between a node, which already share exclusion edges with some nodes, and children of these

nodes. Also, having an exclusion edge between two nodes, we add exclusion edges between each child of one node and each child of the other one. For getting the information of descendants of a node we use a densified version of G_h. For example, there is an exclusion edge between a and d. Also, there are hierarchy edges from a to b and to c, and from d to e. In this case, according to the mentioned algorithm, we are going to add exclusion edges between a and $d`s$ child that is e. Then, we add exclusion edges between all descendants of a (b and c) and d. Also, we add exclusion edges between each descendant of a and each descendant of d.

Algorithm 2: Sparsification of G

1. $\overline{G}_h \leftarrow$ Densify(G_h)
2. $G_e^* \leftarrow$ Sparsify(G_e) using $\overline{G}_h$
3. $G_h^* \leftarrow$ Sparsify(G_h)
4. $G^* \leftarrow G_e^* \bigcup G_h^*$

Algorithm 3: Densification of G

1. $\overline{G}_h \leftarrow$ Densify(G_h)
2. $\overline{G}_e \leftarrow$ Densify(G_e) using $\overline{G}_h$
3. $\overline{G} \leftarrow \overline{G}_h \bigcup \overline{G}_e$

2.3 Topological Sort

There is a crucial part that is needed to be done before proceeding to use the mentioned algorithms. Above, we assumed that a graph is sorted. But almost all the time the graph is unsorted. Therefore, it is important to perform the topological sorting for the graph before the densification and sparsification processes. The topological sort or topological ordering can be explained as a process of linearly ordering the nodes of a directed graph. This is performed in such a way that for each directed edge from a to b, a must appear before b in ordering [4].

The topological sorting can be performed if and only if there are no cycles in the graph [11]. In

other words, we must have a directed acyclic graph (DAG). According to the properties of the HEX graph G, G_h is DAG. Therefore, we implement the topological ordering on it. As a result, the G_e would be topologically sorted too, since both G and G_h have identical vertices. Finally, we would obtain the topologically sorted HEX graph G.

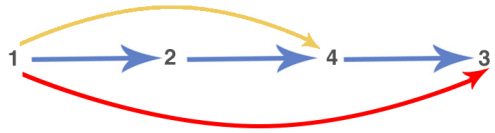

Figure 4. Example of unsorted DAG

In Figure 4 we can see the example of an unsorted DAG. If we do not apply the topological sort algorithm before sparsification and densification steps, we would go in the order [1,2,3,4]. This order gives an improper indication of the longest and shortest paths from one node to another. For instance, the directed edge from 1 to 4 would be considered longer than from 1 to 3. As a result, according to the second principle of the sparsification process, the edge from 1 to 4 would be eliminated before the edge from 1 to 3. This would disable the elimination of the edge from 1 to 3 after that. However, by applying the topological sort algorithm on this graph we would get the right order that is [1,2,4,3]. Using this order all of the principles would work properly.

2.4 Betting

Betting can be defined as placing money or other valuable things at risk to earn more. The money is wagered on an event that has an undetermined outcome. One of the popular and traditional types of betting is the sports betting that is predicting the results of different sports events. If you correctly predict an outcome you win, if not then you lose [12]. In this thesis we considered the betting on football (soccer) matches. The simplest bets that can be made in football are those predicting the outcome of the match. These can be: home win (1), away win (2), draw (X). There might be bets combining previous ones: home team win or draw (1X), away team win or draw (2X), home team win, or away team win (12). In addition, one can bet on the correct score, handicap, total goals,

etc. [9]

Now, we want to explain how the betting system works. Usually, bets are placed through a bookmaker. The bookmaker is the one that estimates probabilities of the events and then offers odds based on them. There are several odds formats: decimal, American, and fractional. Although they might look different, they all mean the same. Thus, for the explanation, we use only decimal (European) odds format. The amount of the win you get depends on how much you bet (stake) and the odd assigned by a bookmaker [2].

	Home Team	Away Team
Probability	0.8	0.2
Odds	1.25	5.00

Table 1. Example of Football Betting

To understand it better let us consider a simple example. Suppose we have two teams (home team and away team) that will play a football match. The bookmaker estimated the probabilities for the wins of the home team and the away team. These are 0.8 and 0.2 respectively (see Table 1). The bookmaker thinks that the home team is more likely to win. Decimal odds are calculated by taking reciprocals of probabilities. Thus, odds for the home team win and the away team win are 1.25 and 5.00. If we put \$100 for the home team win, we would get \$100(1.25) = \$125 back in case of win [2]. In the case of home team loss, we would lose our \$100.

In reality, it is a bit different than described above. Because bookmakers want to ensure that they would earn money in any of the possible cases. Let us consider some football matches, which are not played yet. For example, semi-final matches of Belarus Cup, Shakhter Soligors vs Dinamo Brest and BATE Borisov vs Slavia Mozyr on April 29. According to the gambling company named Betway, their bookmakers propose the odds for this match that are presented in Table 2 [1].

	Home	Draw	Away
Shakhter Soligors - Dinamo Brest	2.15	3.20	3.10
BATE Borisov - Slavia Mozyr	1.30	4.75	8.00

Table 2. Semi-final Matches of Belarus Cup

We can calculate the probabilities for each outcome using the odds given by bookmakers. For the first match the probabilities for home win, draw and away win are $\frac{1}{2.15}$, $\frac{1}{3.20}$ and $\frac{1}{3.10}$. Based on our previous example by adding them we should get 1.00. However, $\frac{1}{2.15}$+ $\frac{1}{3.20}$ +$\frac{1}{3.10}$ ≈ 1.10. This additional 0.10 or 10% is known as bookmaker's margin [3].

2.4.1 Betting Models

We wanted to modify the HEX model for the betting. Then, we used this model for testing the exact inference algorithm with the sparsification and the densification steps that we introduced before. Before we proceed to the explanation of the model, let us explain simpler models for a better understanding of the concept.

Suppose $\mathcal{X} = \mathbb{R}^c$, $\mathcal{Y} = \{0, 1\}^c$, where c is number of classes.

The Trivial Model. First, we introduce the simplest model. In this model the probability of the event is $\mathbb{P}(y_i = 1) = \frac{1+\epsilon}{x_i}$ where ϵ is a margin of organizer.

$$\mathbb{P}(y_i = 1) \propto e^{-\log(x_i)}$$

Parameters space is

$$\Theta = \{\mathbf{f} | \mathbf{f} \in \mathbb{R}^{29}\}$$

Less Trivial Model. Next, we introduced the HEX condition into the model.

$$p(\mathbf{y}|\mathbf{x}, \mathbf{f}) = \text{HEX}(\mathbf{y}) \prod_{i=1}^{29} e^{f_i \log(x_i)}$$

More Advanced Model. Here, we made a more advanced betting model. We introduced the concept of dependencies among the bets.

$$p(\mathbf{y}|\mathbf{x},\mathbf{f}) = \text{HEX}(\mathbf{y}) \prod_{i=1}^{29} e^{\sum_{j \in \text{Depend}_i} w_{ij} \log(x_j)}$$

where Depend_i is (informally), a set of bets, that influences an outcome of ith bet.
In code:

$$DP[i,j] = 1 \Leftrightarrow j \in \text{Depend}_i$$

$$\text{Depend}_i = \{i\}, \text{if } i \text{ is score}$$

2.5 Experiments

The experiments were made using the 6 datasets. These datasets have a collection of data on the same 459 football matches and the bets on them. They differ by that they have data on a different number of bets. First, we conducted experiments to calculate the marginals using the exact inference algorithm (see Algorithm 1) without the sparsification and densification steps. Then, we introduced these steps and conducted the same experiments. All of the experiments were made using the MATLAB software. The results for the execution time of all of the experiments are presented in the table below.

	Unmodified Algorithm (sec)	Modified Algorithm (sec)
Data [c = 20]	25.2406	132.4753
Data [c = 29]	586.1138	402.0480
Data [c = 38]	3.0693×10^3	3.1868×10^3
Data [c = 42]	656.8957	259.2053
Data [c = 45]	4.4815×10^3	884.9222
Data [c = 49]	$> 4.6800 \times 10^4$	4.1916×10^3

Table 3. Performances of the experiments without and with densification and sparsification steps

2.6 Conclusion

In this thesis we studied the probabilistic classification model based on the Hierarchy and Exclusion graphs. Since the HEX model was a special case of the Condition Random Fields model, we could use the framework of the CRF model to train it. Then, we presented the algorithms for the sparsification and densification steps for the exact inference algorithm. Next, we introduced the betting model that is a modification of the HEX model. We calculated marginals for this model using the Exact Inference algorithm without sparsification and densification steps. After that, we performed the same experiments using these steps. And the execution time for all these experiments were estimated. By comparing the execution time for unmodified and modified algorithms, we can see that for data with more features the modified algorithm performed better. Thus, as the size grows, the advantages of sparsification and densification steps become more evident.

Bibliography

[1] *Betway: Sports*, 2020 (accessed April 20, 2020). `https://sports.betway.com/en/sports`.

[2] D Beaudoin, R Insley, and TB Swartz. Studying the bankroll in sports gambling. *Proceedings of the Sixth Australian Conference on Mathematics and Computers in Sport*, page 69, 2002.

[3] Stephen R Clarke et al. Betting on excel to enliven the teaching of probability. Asian Technology Conference in Mathematics, 2002.

[4] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[5] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *European conference on computer vision*, pages 48–64. Springer, 2014.

[6] Jonathan L Gross and Jay Yellen. *Handbook of graph theory*. CRC press, 2003.

[7] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–317. IEEE, 2007.

[8] Ronghang Hu. Hex graph. `https://github.com/ronghanghu/hex_graph`, 2014.

[9] Niko Marttinen. *Creating a Profitable Betting Strategy for Football by Using Statistical Modelling*. PhD thesis, Citeseer, 2002.

[10] Sebastian Nowozin, Christoph H Lampert, et al. Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[11] Sriram Pemmaraju and Steven Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica®*. Cambridge university press, 2003.

[12] Marc N Potenza, David A Fiellin, George R Heninger, Bruce J Rounsaville, and Carolyn M Mazure. Gambling. *Journal of general internal medicine*, 17(9):721–732, 2002.