Master Thesis

Topic:

Unsupervised learning in decision making

Author:	Domagoj Fizulic
	Felix Gutmann
Student number:	125604
	125584
Program:	M.S. Data Science
E-Mail:	domagoj.fizulic@barcelonagse.eu
	felix.gutmann@barcelonagse.eu

I Table of Contents

II List of Figures

III List of Tables

IV List of Listings

V List of mathematical symbols

Symbol	Meaning
a_t	Action at time t
$Q(a)_t$	Value function at time t
ϵ	Probability of exploration in epsilon greedy
α	Learning rate
τ	Softmax parameter

VI List of abbreviations

Abbreviations	Description

1 Introduction and conceptual approach

2 Simmulation

2.1 Data generation

Challange: Categorical data with time series attributes. Apply transformation choice probabilities () Define similarity meassures on categories

2.2 Reinforcement Learning backround

Reinforcement Learning (RIL) is a branch of *Machine Learning* try model how an artificial agents interact with its environment and learns from the process over time.
In particular an agents is confronted with the task of choosing sequently from a set of choices. In comparison to *supervised learning*, where an agent is learning based on set of examples an agent in RIL The objective is to let the agent learn within a certain the optimal action. Since it doesn't have any apriori information about the system it has to explore new possible action and so has to deviate from the optimal action. Furthermore, it has to keep track of value of each action he did so far. So the main task of the agent is to balance exploration and explotation. There are to basic approaches to model this trade-off; An *"Epsilon-Greedy"* selection method and Soft *"Softmax"* selection method.
Considering first epsilon greedy action selection mehtod: Let $Q_t(a)$ be the value function of action. In general select the next action as a as $a_{t+1} = \arg\max Q_t(a)$. However, to model exploration there is a probability to deviate from that greedy strategy. Choose next action Let p_e be the probability of exploration. $p = \epsilon$

$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{K_\alpha}}{K_\alpha}$

$Q_{k+1} = Q_k + \alpha \left[R_k - Q_k\right]$

There are two basic

a. "Epsilon Greedy"

Soft Max Selection

$P(a_t|X) = \frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_i^K e^{\frac{Q_t(i)}{\tau}}}$

2.3 Unsupervised Learning

2.4 Simmulation results

3 List of Literature

Appendix

Metrics and Similarities

This part of the appendix is formally definining metrics and similarities and dissimilarities (proximity) used in this paper. The concepts are somehow related. First we describe the general concepts followed by a description of the applied tools.

First we introduce the concept of *distance*. Let $\boldsymbol{X}$ be a dataset and $\boldsymbol{x_i}, \boldsymbol{x_j}$ two datapoints, such that $\boldsymbol{x_i}, \boldsymbol{x_j} \in \boldsymbol{X}$. A distance function assign for pairs a points a real number as distance. $d : \boldsymbol{X} \times \boldsymbol{X} \mapsto \mathbb{R}_0^+$. Formally there for axioms, which have to be staisfied:

1. $d(\boldsymbol{x_i}, \boldsymbol{x_j}) \geq 0$
2. $d(\boldsymbol{x_i}, \boldsymbol{x_i}) \geq 0$
3. $d(\boldsymbol{x_i}, \boldsymbol{x_j}) = d(\boldsymbol{x_j}, \boldsymbol{x_i})$
4. $d(\boldsymbol{x_i}, \boldsymbol{x_j}) \leq d(\boldsymbol{x_i}, \boldsymbol{x_j}) + d(\boldsymbol{x_j}, \boldsymbol{x_i})$

Besides distance some algorithms operate on a *similarity* matrix. Formally a similarity is a function $S : \boldsymbol{X} \times \boldsymbol{X} \mapsto [0, 1]$. Also for similarity we can define the following properties:

1. $0 \leq S(\boldsymbol{x_i}, \boldsymbol{x_j}) \leq 1, \text{for } i \neq j$
2. $S(\boldsymbol{x_i}, \boldsymbol{x_j}) = 1$
3. $S(\boldsymbol{x_i}, \boldsymbol{x_j}) = S(\boldsymbol{x_j}, \boldsymbol{x_i})$

Once we have computed distance for two data points we can use this information to transform it to a similarity:

$$S(\boldsymbol{x_i}, \boldsymbol{x_j}) = \frac{1}{1 + d(\boldsymbol{x_i}, \boldsymbol{x_j})} \tag{1}$$

Principal Components and Multidimensional Scaling

Spectral Clustering

For the spectral clustering algorithm we formally introduce some graph notation. If not stated otherwise the following derivation follows [SOURCE]. In the following we consider a weighted and simple undirected graph.

$$G = \{V, E\} \tag{2}$$

$$V = \{v_1, \ldots, v_n\} \tag{3}$$

$$E = \{e_1, \ldots, e_n\} \tag{4}$$

Furthermore let the graph has a weighted and symetric ($|V| \times |V|$) adjacency matrix, such that:

$$\boldsymbol{W} = \begin{cases} w_{i,j} & \text{,if } v_i v_j \in E \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The *degree* of a node is defined as the sum of edge weights of connected nodes. Formmally we denote the degree of node i as:

$$d_i := \sum_{j=1}^{n} w_{ij} = \sum_{i=1}^{n} w_{ij} \tag{6}$$

Using the last expression we define matrix $\boldsymbol{D}$ as the diagonal matrix of the degress

$$\boldsymbol{D} := diag(\boldsymbol{d}) \tag{7}$$

The algorithm works on the *Laplacian* matrix defined by:

$$\boldsymbol{L} := \boldsymbol{D} - \boldsymbol{W} \tag{8}$$

Former versions of the algorithm are applied on the graph laplcian. However, there were proposed newer versions using the so called *normalized laplacian*. Since also the python version is using this package we will focus on this version of the algorithhm. Following that the normalized graph laplacian is defined as:

$$\boldsymbol{L}_{norm} := \boldsymbol{D}^{1/2} \boldsymbol{L} \boldsymbol{D}^{1/2} = \boldsymbol{I} - \boldsymbol{D}^{1/2} \boldsymbol{W} \boldsymbol{D}^{1/2} \tag{9}$$

Specification	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	**CL Size**	**SD**	**Decision**	$\boldsymbol{\alpha}$	$\boldsymbol{\tau}$	**N**	**ALG**	**TRNS**	**MI**	**NMI**	**AMI**	**CS**	**HS**	**VMS**
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-		

Table 1: Overview Outcome