Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

Labelling Multivariate Time Series Representation of StarCraft II Replays Using Hidden Markov Models

Gerard, Gauthier

Award date:
2020

Awarding institution:
University of Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 03. Jul. 2025

UNIVERSITÉ
DE NAMUR

FACULTÉ
D'INFORMATIQUE

Labelling Multivariate Time Series Representation of StarCraft II Replays Using Hidden Markov Models

Gauthier Gérard

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Abstract

In recent years, the video game field has been used a lot for testing machine learning algorithms but none of them specifically targeted the classification of replays of video games. This is why this master thesis is oriented towards the automatic annotation of replays using HMMs.

Several formats of replay files exist, and the format that is used during this master thesis is one using multivariate times series in order to represent the game's state at any given moment of a replay.

The usability of HMMs for the replay annotation problem has been evaluated using three experiments and the results are that HMMs could be useful for some annotations but more experiments have to be conducted in order to have a final word.

Keywords: replay annotation – HMMs – semi supervised HMMs – time series

Résumé

Ces dernières années, le domaine du jeu vidéo a été beaucoup utilisé pour tester des algorithmes d'apprentissage automatique, mais la classification des replays de jeux vidéo n'a pas été abordée spécifiquement. C'est pourquoi ce mémoire est orienté vers l'annotation automatique des rediffusions de jeu vidéo en utilisant des HMMs

Il existe plusieurs formats de fichiers de replays et le format utilisé durant ce mémoire est un format utilisant les séries temporelles à plusieurs variables afin de représenter l'état du jeu tout moment d'un replay.

L'utilisabilité des HMMs dans le problème d'annotation de replays a été évaluée à l'aide de trois expériences et les résultats nous montrent que les HMMs pourraient être utiles pour certaines annotations simples cependant, d'autres expériences doivent être menées afin d'avoir un avis final sur le sujet.

Acknowledgments

First of all, I would like to express my gratitude to my promoter Benoît Frénay for accepting my proposed subject for this master thesis and for his useful comments and remarks during all the duration of the thesis. Furthermore, I would like to thank my co-promoter Adrien Bibal for his numerous advice and all the time he dedicated to helping me. Last but not least, I would like to thank my family and friends for their support even though they could not provide much help when writing had to be done.

Table of Contents

1. Introduction

This chapter will introduce the domain of e-sport, which is the domain for which a solution is proposed, as well as the big picture about replay annotation in video games and will.

1.1. Contextualization

E-sport is the term used to refer to electronic sport, such as video game competitions. An e-sport event may not be considered a physically-challenging event, but like the game of checkers it can be characterized as a mental sport. E-sport events really started in the late 90s when people gathered at LAN parties, places where people had to bring their own computer and monitors, in order to fight against each other in their favourite type of video games (Figure 1). LAN parties usually took place between video game amateurs, but with the growing number of people interested in e-sport, they slowly became huge e-sport tournaments where good players or good teams of players could win a lot of money.

Figure 1: example of LAN party taking place in the 90s. Chris Dickens. 20 April 2005 05:04. LAN Party How to – Part 1: Planning and Power, < https://www.smallnetbuilder.com/lanwan/lanwan-howto/24248-lanpartyhowtopt1>

Nowadays, the most popular types of competitive e-sport games are multiplayer online battle arenas (MOBA) such as League of Legends, battle royal games like Fortnite, first person shooters (FPS) like Counter Strike Global Offensive (CSGO) and real time strategy games (RTS) such as StarCraft II. According to The Loadout (https://www.theloadout.com/biggest-esports-games), a news network dedicated to the gaming community, grouping together the four most popular e-sport games of 2019 (League of Legends, Fortnite, Dota 2 and CSGO) and adding up their total number of viewers results in a staggering 9.4 million people watching e-sport tournaments (3.9 million for League of Legends, 2.3 million for Fortnite, 2 million for Dota 2 and 1.2 million for CSGO). The amount of money used as cash prizes exceeds a total of $176 million thus encouraging good video game players to pursue a career in the e-sport industry.

A professional e-sport player, or **pro player** for short, is someone who plays video games for a living. It is often believed that being a professional e-sport player is easy since, unlike other sports where you need equipment and good physical condition, e-sport only requires a computer with an internet connection. Nevertheless, anyone that has tried making a living from e-sport soon realizes that the competition is harsh and that only the few best are able to live off it.

The International is a good example of a famous e-sport tournament. It is an annual Dota 2 tournament (Figure 2), which in 2019 opposed 18 teams of 5 players and, at the end, each team won a prize according to its ranking. The last place (the 18th place) was awarded $85,756 and the first place won $15,607,638.

Figure 2: The International 2019 e-sport event, Monster Energy, 27 August 2019. <https://www.monsterenergy.com/news/the-international-2019-recap>

Thanks to the internet, more and more professional replays (video of a video game matches) are shared online. This and the increasing popularity of e-sport made jobs such as professional replay casters a reality. This expresses a need for in-depth replay analysis since the two main roles of casters are to entertain and explain what the replay players are doing, thinking and planning.

For now, the current (i) ways for replay spectators to know what is going on during any given time of a replay are either watching pro casters analyse and comment on a replay (Figure 4) or (ii) loading up a replay themselves and trying to interpret what the players' strategies are with some help provided by the game itself. The general help provided by the game itself is under the form of a tag on a timeline of the replay. Tags represent moments when a specific action took place. Such actions are low-level actions such as when a player died or when a player killed another player (Figure 3). The problem with watching a replay is that it is time consuming since you have to watch the entirety of the replay, and it can last from one minute up to several hours depending on the type of games and the players' level. Another annoying factor for players trying to improve their skill is the fact that replays are not often labelled or time-stamped by the casters or by the game. This means that for a player to find a specific moment of a replay, like when fights are taking place or when a player is using a specific strategy, the user needs to watch the entire replay and note where its perception of interesting things are happening (where he/she observes interesting things happening).

Figure 3: Replay of CSGO where skulls are showing during which round of the competitive match the player MeatAlive (user name) killed an enemy player. Skull with a hole in it shows when the player performed a headshot.

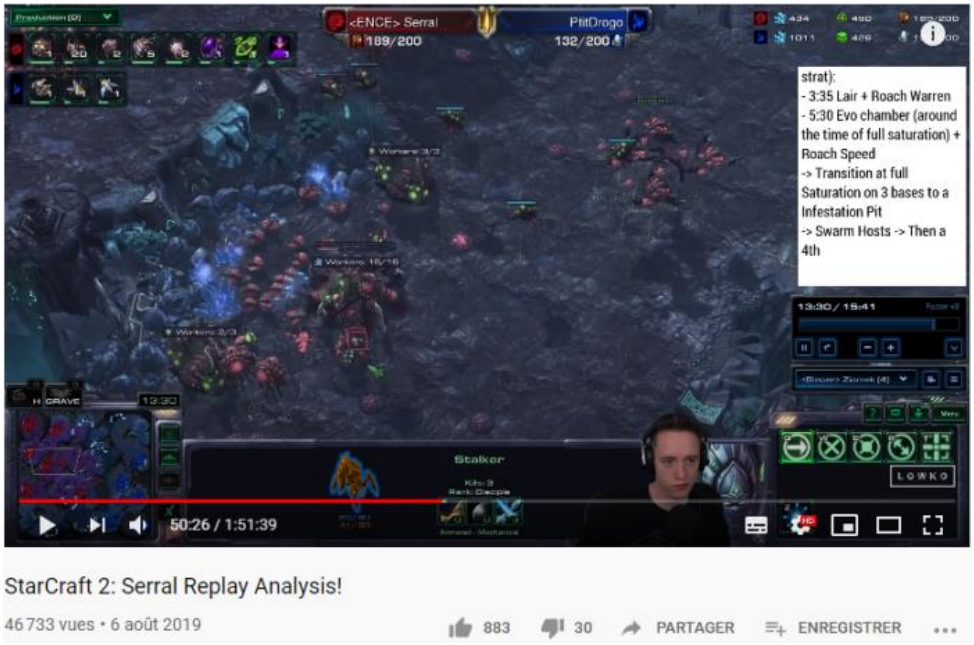


Figure 4: StarCraft II caster analysing a pro replay on YouTube.[1]

Exploring the idea of replay annotation, we would like to create an automatic replay annotation system that offers higher level labels to the user contrarily with what is done now (Figure 3.). In order to show the usability of such system, the RTS game **StarCraft II** is used.

1.2. Problem Statement

To our knowledge, video game **replay annotation** using machine learning is something that has not been done before. For now, one of the two ways to know what is going on inside a replay is to watch it in its entirety and have the skills necessary to know what is happening at any given moment. The other way consists of watching a caster commenting on a replay. In both cases, this is time consuming and this is why using a simple, yet well-known, machine learning algorithm should be useful to solve this problem that more and more players are experiencing.

This problem is composed of two distinct sub-problems. First, a replay needs to be able to express the state of the StarCraft II game at any given time. This means that if a raw replay is not sufficient to represent the game's state at any given time, our solution will need to process it in order to make it usable by our machine-learning model. The main reason why a replay would not be sufficient to represent the game state is that replays are usually made in such a way that they minimize the replay's size while allowing the game's engine to play the replay back. Second, we need to be able to train HMM on fully labelled and partially labelled replays since a user is not restricted to entire replay annotations.

1.3. Outline

The structure in chapters for this master thesis is the following. First, after this introduction (chapter 1.), there will be a chapter dedicated to the StarCraft II game's mechanic and replay file format (chapter 2.). Second, the state of the art about automatic replay annotation and hidden Markov models in general is presented (chapter 3.) followed by a next chapter specifically designed to explain the

[1] LowkoTV. "StarCraft 2: Serral Replay Analysis" Online video clip. Youtube, 6 August 2019. https://www.youtube.com/watch?v=M0W1_NzcSP8&t=3026s

training process of hidden Markov models (chapter 4.). After that, a chapter regroups and explains the main contributions of this master thesis (chapter 5.) and finally, experiments are conducted (chapter 6.) and their results are analyses and discussed (chapter 7.). This master thesis ends on the conclusion and future work chapter (chapter 8.).

2. StarCraft II

As mentioned at the end of the contextualization section, section 1.1, the video game StarCraft II will be our use case. This means that the models created for the test of the proposed method will be models trained on data issued from StarCraft II replays. This section is therefore specifically designed to explain what StarCraft II is.

StarCraft II is a real-time strategy (RTS) game that was developed by Blizzard Entertainment in 2010. It is a science fiction and military combat-oriented game that contains a single player campaign as well as a multiplayer arena aspect. In multiplayer, players all around the globe can compete against one another and in recent years, the increasing popularity of the game made competitive StarCraft II players able to earn a living playing it. Players dominating 2020 are players such as Dark, Maru (Figure 5.) and Serral (these are their in-game names).

Figure 5: Maru (Terran) vs Dark (Zerg) - Quarterfinals - 2019 WCS[2]

The goal of a StarCraft II multiplayer game is to beat its opponent. For this, a player has to develop his/her in-game economy as well as create military units in order to attack the enemy faction and ultimately destroy it. A game ends when the opponent surrenders or when all his/her production buildings are destroyed.

There are three main factions that the user can choose from to start a multiplayer game. The first one is called the "**Terran**", which is the human faction of the game. The second is the "**Zerg**" faction, an alien race with insect-like units. The last one is the "**Protoss**" faction, which represents a technologically advanced alien species. Each faction has its own units and its own strength and weakness. For example, the Zerg faction has relatively cheap units that are weak and thus is able to quickly overwhelm its opponent's main base without compromising its in-game economy. On the other hand, the Protoss faction has expensive units that are extremely strong: they have lots of health points (HP) and deal tons of damage. This usually results in a smaller amount of military units for the

[2] StarCraft Esports. « Maru vs Dark TvZ – Quarterfinals – 2019 WCS Global Finals – StarCraft II" Online video clip. Youtube, 2 November 2019. https://www.youtube.com/watch?v=T7fExr5SBc4

Protoss compared to its opponent. The Terran faction is somewhere in between those two previous factions because its units prices are usually affordable and the damage they can deal during a fight is reasonable. Nevertheless, it is important to note that all these three factions are balanced and this has been achieved thanks to multiple game's updates and nerfs (action of reducing the damage of a previously overpowered attack or increasing the price of previously inexpensive, game changing objects like unlocks or upgrades, etc.).

As mentioned earlier, the game contains an **economical aspect**. This aspect can be observed by the players by looking at their current resources count (minerals and vespene gas count), their number of town halls and their current number of workers extracting resources (Figure 6). There are two types of resources: minerals and vespene gas, which are only collectable using the faction's workers and its town hall. Workers have to deliver the collected resources to the town hall in order for the resources to be used by the respective player. One town hall and twelve workers are available at the beginning of any StarCraft II competitive game and this allows the players to start their economy and build new units and structures in order to win the match.

Concerning the structures, there are five main types of them. The first one is the **town hall** (or main building), which is used for the creation of workers and the gathering of resources since all collectable resources have to be brought back to the town hall. The second type of structure contains only one building and this building allows the vespene gas extraction from geysers. This special building has to be placed on unoccupied geysers and, once built, it can be used by workers to collect vespene gas. Compared to the mineral resources which only need a worker in order to be extracted, the vespene gas extraction requires a **refinery**, an assimilator or an extractor according to the player's faction. The next type of structure, the **supply depot**, directly affects the total number of units a player can have on the map, with the maximum unit cap being set at 200 living units at any given time. Then there are the **production** buildings, which are used by the Protoss and Terran factions to produce military units. For the Zerg faction, the hatchery (the Zerg faction's town hall) is the one building responsible for all the unit production. It acts like a beehive where the queen lays eggs in order to give birth to workers or military bees. The last type of building is the **upgrade** building type. These structures' only goal is to offer the player upgrades to the attack and to the defence of most of its military units or infrastructures.

Figure 6: example of resources gathering from a Protoss player.

A good StarCraft II player is a player that manages the macro and micro levels of the game while at the same time reading his/her opponent's tactic. This means that the player has to be able to implement a long-term strategy as well as micro managing fights while scouting the map to find out what the other player is planning to do in order to counter it and win the game.

This last aspect is related to **fog of war** (Figure 7.) and the technology tree of each faction. In order for a faction to produce a unit, the unit needs to be unlocked in the technology tree. Unlocks are done by building a structure lower in the tree. If a player sees some specific building of the opponent, he could potentially guess what military units will come to attack him in the near future. Guessing the opponent's tactic is even more difficult since the game's map is covered with a fog of war that only disappears around each unit or building of the player. This is why pro players actively send units to scout the enemy base. Not sending one of its workers (or overlord for the Zerg faction) scouting at the beginning of a game is usually a sign that the player is inexperienced.

Figure 7: Zerg sending an Overlord to scout the enemy base and seeing a military unit guarding the main base entry as well as a building being created.

An important aspect of StarCraft II is that it is a one hundred percent **deterministic** video game. No aspect of the game relies on randomness and this small detail affects how players play and how the game saves its replays. For the player, knowing that no random actions can happen means that he/she can more easily determine a current objective. For example, if the time taken for a unit to open fire on an enemy unit was random, let's say between zero and one second, a player might think twice before attacking a unit. It could even be game breaking, such as in the extreme example when a competitive CSGO match opposing two professional teams (Cloud9 and Fnatic)[3] was taking place on the Overpass map and a round was lost due to a random game event. This event was the passage of a train which cancelled a player's action.

Using StarCraft II replays and extracting meaningful information from them is not straightforward. This is because replays' main task is to be played again thanks to some game's engine. To optimize this task, replay's structure is minimalistic and usually only uses one sequence of the players' actions.

[3] 3kliksphilip. « Cloud 9 VS Fnatic : Did the train ruin it?" Online video clip. Youtube, 6 July 2015. https://www.youtube.com/watch?v=3wQib7egjMk

This is possible because for deterministic games, else more information about the replays need to be saved as well. This sequence allows the game's engine to easily play back the match but, without the proper program it is impossible to know the number of units killed by a player at any given time, or its army count and composition, etc.

```
00.00    PlayerSetupEvent
00.00    PlayerSetupEvent
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceOverlord upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceStalker upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceGhost upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceColossus upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceRoach upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceOracle upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceMule upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceViking upgrade completed
00.00    Player 1 - AGOElazer (Zerg) - RewardDanceInfestor upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceOverlord upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceStalker upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceGhost upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceColossus upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceRoach upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceOracle upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceMule upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceViking upgrade completed
00.00    Player 2 - Serral (Zerg) - RewardDanceInfestor upgrade completed
00.00               None - Unit born MineralField [1]
00.00               None - Unit born MineralField750 [40001]
00.00               None - Unit born MineralField [80001]
00.00               None - Unit born MineralField750 [C0001]
00.00               None - Unit born MineralField450 [100001]
00.00               None - Unit born MineralField [140001]
00.00               None - Unit born MineralField [180001]
```

Figure 8: first elements contained in a raw StarCraft II replay file. In this case, the initialization of player Serral and player AGOElazer as well as mineral fields initialisation.

```
00.20   BlyOnFire       CameraEvent at (54.67578125, 31.25390625)
00.20   AGOElazer       CameraEvent at (134.6015625, 115.55859375)
00.20   Indy            CameraEvent at (49.50390625, 31.453125)
00.20   Indy            CameraEvent at (49.3046875, 31.3671875)
00.20   BlyOnFire       CameraEvent at (54.67578125, 29.41015625)
00.20   BlyOnFire       CameraEvent at (54.67578125, 28.87109375)
00.20   Serral          Right Click; Target: MineralField750 [028C0001]; Location: (48.0, 25.5, 4910
4)
00.20   Indy            CameraEvent at (48.78125, 31.125)
00.20   AGOElazer       CameraEvent at (146.50390625, 91.140625)
00.20   Indy            CameraEvent at (48.6171875, 30.94921875)
00.20   Serral          Right Click; Target: MineralField750 [028C0001]; Location: (48.0, 25.5, 4910
4)
00.20   Indy            CameraEvent at (48.46875, 30.7109375)
00.20   BlyOnFire       SelectionEvent['Drone [3440002]', 'Drone [3940001]', 'Drone [3980001]', 'Dron
e [39C0001]', 'Drone [3A00001]', 'Drone [3A40001]', 'Drone [3A80001]', 'Drone [3AC0001]', 'Drone [3B0
0001]', 'Drone [3B40001]', 'Drone [3B80001]', 'Drone [3BC0001]', 'Drone [3C00001]']
00.20   AGOElazer       CameraEvent at (146.50390625, 91.16796875)
00.20   Indy            CameraEvent at (0.0, 0.0)
00.20   Serral          Right Click; Target: MineralField750 [028C0001]; Location: (48.0, 25.5, 4910
4)
00.20   AGOElazer       CameraEvent at (147.19921875, 91.31640625)
```

Figure 9: raw StarCraft II replay where camera events, right click events and a unit selection event are taking place.

There are three methods available to use replays in our machine learning algorithm. The first method is to use **raw replays** and to train a model on the list of players' actions (Figure 8.). This implies that spammed keys and noise will be present in the data and this need to be addressed. A spam key is a key that is repeatedly pushed in order for a pro player to warmup or keep the rhythm during a game because it is important for them to have a high rate of actions per minutes (APM). The second method is the more human-like method and is used by the latest iteration of the AlphaStar artificial

intelligence [1]. It consists of viewing the world through the lens of a camera and to interpret the replay like a human would do. This means using every **frame** of the replayed game as data. This can cause problems due to the extreme complexity of information extraction from raw-pixel data. The last of the three methods consists of modifying raw StarCraft II replays into **an appropriate format** (Figure 9.). This can be done using libraries specially developed to manipulate replays (sc2reader 0.7.0).

In resume, StarCraft II is a 100% deterministic game that is extremely popular, thus providing a quit abundant source of replay files and there exist three ways of using replay files as data for our algorithms. Having introduced the replay annotation problem and the video game used as a use case by this master thesis, the next section will review what has been done for automatic replay annotation.

3. State of the Art – Automatic Replay Annotation (Sequence Labelling)

Machine learning algorithms have been applied on video games since decades. It started with checkers in 1959 [2] and continued todays with real time strategy games (RTS) like StarCraft II with the famous AlphaStar AI [3] or on the multiplayer online battle arena (MOBA) Dota II with the AI of Elon Musk's start-up (OpenAI) [4]. In recent years, more and more video games allow replay analysis and data extraction. This in turns allows for easier machine learning algorithm tests to be performed.

Weirdly enough, the problem of automatic **replay annotation**, which is closely related to the sequence labelling problem, has not been tackled even though thousands of unlabelled replays are available and even though thousands of players wish for such solution. In order to have a big picture of what has been done to solve the problem of replay annotation, a first section will explain the type of replay representation chosen for this master thesis (section 3.1). A second section (section 3.2) introduces the popular machine learning algorithms used to provide a sequence of label when a sequence of observation is given. In our case, the sequence of observations will be a sequence of the game's state representation. Finally, since this master thesis aims to demonstrate the usefulness of HMMs in automatic replay annotation, a section will be dedicated to the explanation of what an HMM is and how it works (section 3.3).

3.1. Replay Annotation

As written at the end of chapter 2, there are three different methods available in order to use replays as data for our model. In the case of StarCraft II, using the first method is not ideal since StarCraft II replays only store the user's actions. This does not directly allow for a good game's state representation. The second method, the one using video frames of the game, is not suitable either for this master thesis due to the fact that information extraction from pixel-data requires humongous amounts of data and computer power. For these reasons, the method chosen for this master thesis is the third one. This method consists in transforming the raw sequence of the players' actions into a sequence of values that enable easy game state representation. Such values are the number of military units at any given time, the amount of enemy units killed at any given time, etc. To be more precise, replays will be transformed into multivariate time series of twelve variables and the time step will be one second between each observation of the time series (Figure 10).

	workers	army	Building	enemy_building_killed	enemy_army_killed	enemy_scv_killed	slope_workers	slope_army	slope_Building
0	12	1	1	0	0	0	0.483754	0.040313	0.040313
1	12	1	1	0	0	0	0.529697	0.044141	0.044141
2	12	1	1	0	0	0	0.600096	0.050008	0.050008
3	12	1	1	0	0	0	0.625930	0.052161	0.052161
4	12	1	1	0	0	0	0.562139	0.046845	0.046845
...	...	...	...	...	...	...	...	...	...
553	54	69	18	3	92	12	-0.000007	-0.346149	0.020569
554	54	68	18	3	101	12	-0.000005	-0.218217	0.013405
555	54	68	18	3	106	12	-0.000003	-0.095845	0.007161
556	54	66	18	3	108	12	-0.000002	-0.053671	0.004969
557	54	65	18	3	109	12	-0.000003	-0.122917	0.007527

558 rows × 12 columns

Figure 10: multivariate time series representation of a StarCraft II replay. Here 9 of the 12 dimensions are visible and each line corresponds to a second of the replay.

The time step (represented in the extreme left column) is 1 second and this as been chosen because the other options would have been to use replay information at the frame granularity. A replay being saved with 24 frames per second, and relatively few actions happening by frames, the choice of using 1 second was justified.

3.2. Sequence Labelling

Sequence labelling of time series is a problem that occurs in lots of different scientific fields. It consists of assigning a categorical label to each observation of a sequence. The classical example for sequence labelling is part of speech tagging (**POS tagging**) and represents the task of associating a word to a part of speech. The most popular frameworks for sequence labelling are hidden Markov models [5], maximum-entropy Markov models (MEMMs), conditional random fields (CRFs) [6] and recurrent neural networks (RNNs). Of course, lots of variants of these models exist as well as combinations of them. For example, an RNN can be coupled with an HMM thus creating a hybrid solution for the sequence labelling problem [7]. Another use of RNNs that performs well is when RNNs are coupled with an LSTM structure. LSTM stands for Long Short-Term Memory and as explained in [8], if an RNN is coupled with it, this allows the model to not only look at the previous context, but to also look further back in the sequence. This can have a meaningful impact if previous observations help at labelling the current one.

These four popular frameworks have some fundamental differences that need to be pointed out. First, **HMMs** make three strong assumptions about the modelled process. The first assumption is that the process they are modelling is a Markov process. This means that the probability of an event only depends on the previous state of the HMM and the current observation. The second assumption is that state transition probabilities are independent of the time at which they occur. The last assumption is that the emission probability of the current observation is independent of the previous observation given the current state of the HMM. HMMs are generative models, well known for representing sequential data, and this type of model has successfully been applied in fields such as POS tagging [9], text segmentation and information extraction. Nevertheless, it is important to notice that the training of the HMM is generative even though sequence labelling is a discriminative task.

Second, **MEMMs** are the discriminative alternatives to HMMs. MEMMs are models used to represent the probability of transitioning to a state if the model has a given observation o and the previous state s'. In a MEMM, the HMM transition and observation functions are replaced by a single function $P(s|s', o)$ that provides the probability of the current state s given the previous state s' and the current observation o [10]. The bigger difference between a MEMM and an HMM is that MEMM uses state-observation transition functions rather than the separate transition and observation functions of the HMM. In other words, MEMM combines the advantages of HMMs and maximum-entropy models that allow state transition to depend on non-independent features of the sequence under analysis.

Third, **CRFs** represent the decision boundary between the different classes it has to learn, thus including CRFs in the discriminative type of classifier. CRFs are a supervised method for structured prediction similar to HMMs while relaxing the independence assumption of the observations and the Markov assumption. A CRF gives a sequence of labels for a given sequence of observation and to do so it takes also advantage of the information form the entire sequence to estimate the probability of the entire sequence of observation.

The last commonly used framework for sequence labelling is the **RNN**. Compared to the previous models (HMMs & CRFs), RNNs do not need prior knowledge about the data. For example, RNNs do not need loads of task specific knowledge while HMMs do. HMMs need them in order to correctly

design their hidden states. Furthermore, RNNs do not make any assumptions, such as the Markov assumption, about the data. RNNs can be trained discriminatively [11] and their internal state provides a powerful and general mechanism for modelling time series. In addition, they tend to be robust to temporal and spatial noise. But still, the most effective use of RNNs for sequence labelling is when RNNs are coupled with HMMs in the so-called hybrid approach [7].

Having a global view of the literature to label time series and since the proposed solution of this master thesis uses HMMs, the last part of this section explains HMMs in more details.

3.3. Hidden Markov Models (HMMs)

The decision of using HMMs for this master thesis is driven by the fact that HMMs are known to correctly model processes that can be divided into phases and which phases changes in a more or less consistent fashion. Furthermore, the choice of a relatively simple, yet well-known algorithm will be the first step in verifying if replay annotation is doable with simple models.

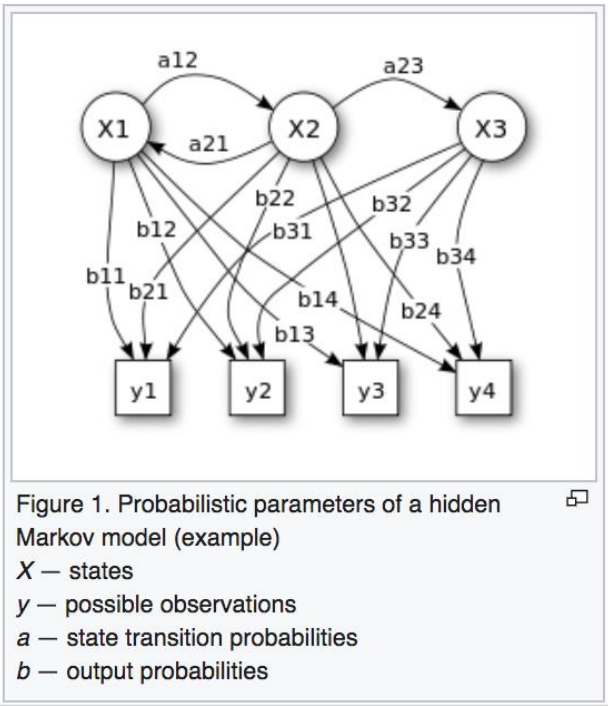


Figure 11: example of HMM with 3 hidden states. Its transitions (a12, a21 and a23) and emission probabilities (b11, ... , b34) are displayed. Source: Tdunning. 01:20, 20 January 2012. "Hidden Markov Model with Output", Wikipedia. Accessed 23 May 2020, <https://en.wikipedia.org/wiki/Hidden_Markov_model>

HMMs can be represented by a triplet $\lambda = (\pi, A, B)$ where $\pi = \{\pi_i\}$ is the prior probability distribution of the hidden states and $\sum_i^n \pi_i = 1$, A is the transition probability matrix such that, each a_{ij} represents the probability to move from state i to state j and the $\sum_{j=1}^{n} a_{ij} = 1$ and B represents the emission probabilities allowing the HMM to know the probability of observing observation o_i in the different states of the HMM. In our use case (StarCraft II), an observation o_i is a set of twelve continuous dimensions representing a second of a replay.

An HMM is [5] a statistical model that represents a system which is assumed to be a Markov process with unobservable states, also known as, hidden states. The assumption that the process modelled by the HMM is Markovian means that for any given observation, its hidden state only depends on the previous hidden state value.

$$P(q_i|q_1 \dots q_{i-1}) = P(q_i|q_{i-1})$$

As stated in [5], HMMs are useful in real world applications. To prove their usefulness, HMMs have been used to solve three key problems of interest appearing in the real world.

3.3.1. The Evaluation Problem

The first problem is called the evaluation problem and is described as follows. Given a sequence of observations $O = o_1, o_2, \dots, o_t$ and an HMM $\lambda = (\pi, \mathrm{A}, \mathrm{B})$, what is $P(O|\lambda)$. This problem is solved by the **Forward algorithm**.

The explanation of the Forward algorithm is taken and adapted from [12]. An HMM with N hidden states and a sequence of observation of size T, there are N^T possible sequence of hidden states of length T. This means that for large values of N and T, the number of different paths increases exponentially. To solve, the first problem needs an efficient algorithm. This algorithm is the forward algorithm and has a complexity of $O(N^2T)$ instead of an exponential complexity.

The forwards algorithm works by iteratively computing the likelihood probability of the sequence of states according to a given sequence of observation. This means that the algorithm is computing a matrix (NxT) where each cells $\alpha_t(q_i)$ of the matrix represents the probability of being in state q_i after observing the first t number of observations according to some HMM λ. To compute this matrix, the algorithm starts by computing $\alpha_1(q_i)$ the probability of being in the state q_i given the first observation. Those values are computed as $\alpha_1(q_i) = \pi_i b_i(o_1)$ where π_i is the probability to start the sequence in state q_i and $b_i(o_1)$ is the probability of observing the first observation in state $b_i(o_1)$. This gives the first column of the matrix and allows for the algorithm to continue.

The second step of the algorithm is a recursion step where it recursively compute the next column of the matrix by looking at the values of the previous column (this is why the first step is required). The formula for the recursion step is the following, $\alpha_t(q_i) = \sum_{k=1}^{N} \alpha_{t-1}(q_k) a_{ki} b_i(o_t)$, $1 \leq \mathrm{i} \leq \mathrm{N}$, $1 < \mathrm{t} \leq \mathrm{T}$. Where a_{ki} represents the probability of transitioning from state q_k to state q_i. After this step, the entire (NxT) matrix will be filled and thus the termination part will happen. The final action of the algorithm is to return the probability $P(O|\lambda)$ and does so by adding all the cells of the last column of the matrix together $P(O|\lambda) = \sum_{k=1}^{N} \alpha_T(q_k)$.

3.3.2. The Decoding Problem

The second problem is referred to as the decoding problem and is described as follows. Given an HMM $\lambda = (\pi, \mathrm{A}, \mathrm{B})$ and a sequence of observations $O = o_1, o_2, \dots, o_t$, what is the most likely hidden state sequence that can generate the sequence O of observations? The decoding problem is solved by the **Viterbi Algorithm**.

The Viterbi algorithm is a recursive optimal solution to the second problem [13]. The Viterbi algorithm follows the same principles as the Forward algorithm. It also computes a (NxT) matrix by looking at the sequence of observations from left to right [12]. But now, each cell $v_t(q_i)$ of the matrix represents the probability that the HMM is in state q_i after observing the first t observations and taking the path with the higher probability.

The initialization step of this algorithm is the same as the Forward algorithm but the second and last steps are different. The second step, which computes the values of $v_t(q_i)$, $1 < \mathrm{t} \leq \mathrm{T}$, uses $v_t(q_i) = \max v_{t-1}(q_k) a_{ki} b_i(o_t)$, $1 \leq \mathrm{k} \leq \mathrm{N}$. Compared to the first algorithm presented, the Viterbi

algorithm termination step returns a list of the best hidden state sequence that generates the given sequence of observations O. To do so, the algorithm keeps track of the previous best hidden state for each observation. The returned sequence of hidden states is obtained by going backwards from the last best hidden state q_T and always looking at the previous best hidden state saved by the algorithm.

The Learning Problem

The last problem is the learning problem and is described as follow. Given and an HMM $\lambda = (\pi, \mathrm{A}, \mathrm{B})$ and a sequence of observations $O = o_1, o_2, \dots, o_t$ or a sequence of states $S = s_1, s_2, \dots, s_t$, how to modify $(\pi, \mathrm{A}, \mathrm{B})$ so that it maximizes the probability of observing sequence O or S. This last key problem can be solved using the **Baum-Welch algorithm**.

The Baum-Welch method [5] is an expectation-maximization (EM) algorithm. This algorithm is used to update the parameters of a given HMM λ in order to increase its probability of generating a sequence of observation O. The HMM's parameters the algorithm is allowed to update are the transition matrix A, the emission probability matrix B and the initial probability of each state q_i. The algorithm will give a locally optimal solution and does so by iteratively estimating the best triplet $(\pi, \mathrm{A}, \mathrm{B})$ that can generate the observed sequence or sequences. Each of the algorithm iterations are guaranteed to increase the probability of observing the data.

Now that the basics of sequence labelling and HMMs have been explained, the next chapter is dedicated to the different types of training for HMMs in the sequence labelling task.

4. HMM Training

The data used during the training phase of any model can either be labelled or unlabelled. For the classification problem, this means that some data have their associated label while others do not have that information. In the more precise case of sequence labelling, where a sequence of label has to be attributed to a sequence of observation (each observation has to be classified), this means that some training sequences of observations do have their respective sequence of labels. Sequences of observations are denoted as $S = \{s_1, \dots, s_n\}$ where n is the number of sequences. Each $s_i, 1 \leq i \leq n$ contains a list of observations such as $s_i = \{o_1, \dots, o_{t_i}\}$ where t_i is the length of sequence s_i.

The next sections will describe the three main types of training processes used to learn the HMM's parameters. The first section will explain the supervised training of HMMs, the second will explore the opposite type of training called the unsupervised training and the last section will introduce the semi-supervised training of HMMs.

Supervised Training

In supervised training, the model has only access to a set of labelled sequences $S = \{s_1, \dots, s_n\}$ and their respective sequences of label $Y^{(l)} = \{{y_1}^{(l)}, \dots, {y_n}^{(l)}\}$ where $y_i = \{l_1, \dots, l_{t_i}\}$ and where l_j is the class to which observation o_j of sequence s_i belongs to. The goal is to compute the transition probability matrix A, the emission probabilities B and the initial probabilities for each state q_i. Computing those three parameters will define an HMM $\lambda = (\pi, \mathrm{A}, \mathrm{B})$ fitted on the given labelled data.

Considering an HMM λ of N hidden states (one for each type of possible label) characterized by a triplet $(\pi, \mathrm{A}, \mathrm{B})$. Considering a sequence of observation $s = \{o_1, \dots, o_t\}$ where t is the length of the sequence and where o_i is a vector of m values. Considering $l = \{l_1, \dots, l_t\}$ the sequence of labels corresponding to the s sequence. Training the HMM λ with the labelled sequence s starts by computing the N values of π, the initial probabilities of each states. π_i can easily be obtained from labelled sequences by dividing the number of times state q_i started a sequence s by the total number of sequences in the training set. The transition probability matrix A can also easily be computed using only the sequences of labels (labels representing hidden states) by counting how many times the hidden state q_i moved to state q_j and dividing this value by the number of hidden state transitions initiated by the selected q_i state. Doing this iteratively for each states gives the transition probability matrix A of dimension (NxN).

Finally, the emission probabilities matrix of HMM λ is computed according to the type of observations in the sequence. There are two types of observations: finite observations and continuous observations. Finite observations correspond to symbols belonging to a finite set of symbols. Thus, for this type of observation, computing the emission probability of a given symbol α by a hidden state q_i is done by counting how many times this symbol appeared in hidden state q_i divided by the total number of symbols observed by q_i. For continuous observation, things are different. This is because in order to have non-zero probabilities of observing a real number in an infinite set of real numbers, the model needs to approximate its emission probability by using a probability density function (pdf). If an observation o_i of a sequence s is a single continuous value, this can be achieved using the probability density function of a Normal distribution of mean μ and standard deviation σ. The value obtained by doing the probability density function will approximate well enough the emission probability by giving a probability of observing values in a range of real numbers instead of giving the probability of observing a specific real number (which is zero). To return to our use case that uses a vector of twelve continuous values instead of a single one, the emission probability can be computed using

multivariate Gaussian distributions and their respective probability density function. Multivariate Gaussian distributions (Figure 12) are essentially a set of Gaussian (or Normal) distributions and the pdf value is obtained using the different means μ_i and standard deviation σ_i of the different Gaussian distributions. In our use case that uses twelve continuous values per observation, this means that the HMM needs to store twelve means μ_i and twelve standard deviations σ_i.

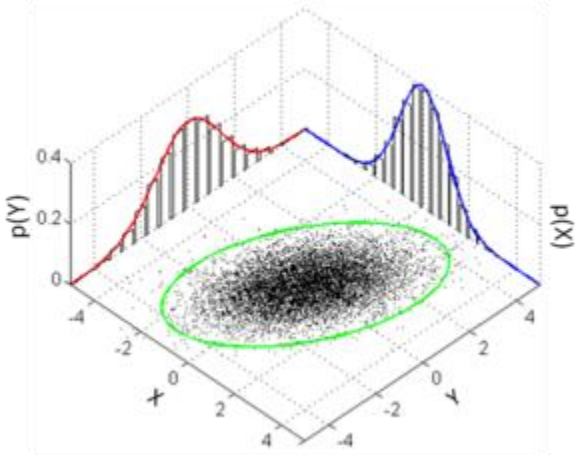


Figure 12: example of multivariate Gaussian distribution (in this case of 2 random variables). Source: Bscan. 15:28, 22 March 2013. "File:MultivariateNormal.png", Wikipedia. Accessed 24 May 2020, <https://en.wikipedia.org/wiki/Multivariate_normal_distribution>

Now, the triplet $(\pi, \mathrm{A}, \mathrm{B})$ is fitted and the HMM was fitted on the labelled sequences s.

4.1. Unsupervised Training

In unsupervised training, the model does not have any label to work with. This means that an artificial sequence of labels needs to be determined for each unlabelled sequence of observation. This can be achieved thanks to different clustering methods before training takes place or during the training.

Associating a label before the training process can be achieved using a classic clustering algorithm such as the **k-means** algorithm. This classic algorithm for unsupervised learning was invented by Lloyd in 1957 [14] and later, other versions of it such as the adaptive k-means of Macqueen [28] were developed. After the creation of artificial sequences of labels for the training data, the training process is the same as in supervised training since now the model does have a set of labelled sequences $S = \{s_1, \dots, s_n\}$ and their respective sequences of label $Y^{(l)} = \{{y_1}^{(l)}, \dots, {y_n}^{(l)}\}$.

Another way of dealing with unsupervised learning is to first initialize the model and then train on the unlabelled data. As illustrated in [15] and [16] where the HMM is first initialized and where the HMM's parameters are iteratively updated using the Baum-Welch algorithm to best fit the unlabelled data. [16] Proposes a clustering methodology for sequence data using HMM. Even if our sequence labelling case is different from the clustering problem, the training process will remain the same.

4.2. Semi-Supervised Training

Using unlabelled data in order to increase the model's performance and robustness is possible as demonstrated in [17] and applying such a method for HMMs has also been done [18]. This is due to the ever increasing enthusiasm in developing semi-supervised algorithms since most of the real world problems do not have an infinite quantity of training labels to train models on. This section will start by presenting what is intended by semi-supervised learning and will finish by listing and explaining

the three different strategies that exist in order to utilize unlabelled data in the semi-supervised training process.

As specified in [19], semi-supervised training consists in the training of an HMM λ with labelled data $L = \{o_i^{(l)}, y_i^{(l)}\}_{i=1}^{N_l}$ and unlabelled data $U = \{o_i^{(u)}\}_{i=1}^{N_u}$ where N_l is the number of labelled data, where N_u is the number of unlabelled data and $y_i^{(l)}$ the label of the i^{th} data. In order to train the HMM λ on unlabelled data, a label has to be given to each unlabelled data.

This can be done using **graph-based algorithm** such as the label propagation algorithms [20] (Figure 13.) to infer the label of unlabelled data by connecting similar observations together and propagating the label information through it from labelled to unlabelled node [21]. However, label propagation quality is hugely affected by the type of graph used by the algorithm.

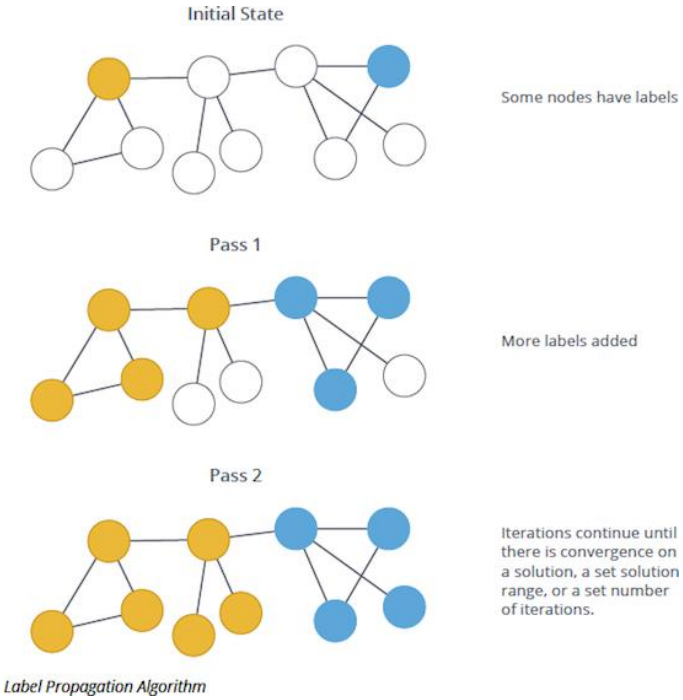


Figure 13: Example of how the Label Propagation Algorithm works. Authors: Mark Needham & Amy E. Hodler, Neo4j. 4 March 2019. <https://neo4j.com/blog/graph-algorithms-neo4j-label-propagation/>

Another method used to train a model in a semi-supervised way is called the **self-training method**. This method consists in using the few labelled data to train a first iteration of the model, followed by using this first iteration of the model to give a label to each of the unlabelled data. Finally, the model trains itself again with its newly labelled data and does so until no more unlabelled data are available. This way for a model to train itself has been popularized in fields such as natural language processing [22] and image recognition [23]. A recurrent problem with self-training models is that if they wrongly label unlabelled data during the first iteration, it will reinforce itself during the next iteration and thus learn something incorrect and lose in efficiency.

To conclude this section, three semi-supervised algorithms to train HMMs will be explained [19]. These three algorithms belong to the self-training method of semi-supervised learning since they do not use any other classifier to annotate the unlabelled data.

The first algorithm is the SSHC-1 algorithm and works as follows:

Algorithm: SSHC-1

Input: A set of N_l labeled sequences $O^{(l)} = \{o_1^{(l)}, ..., o_{N_l}^{(l)}\}$ with labels $Y^{(l)} = \{y_1^{(l)}, ...y_{N_l}^{(l)}\}$, N_u unlabeled sequences $O^{(u)} = \{o_1^{(u)}, ..., o_{N_u}^{(u)}\}$, HMM structure $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_K\}$.

Output: Trained model parameters $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_K\}$ and a partition of the unlabeled sequences given by the identity vector $Y^{(u)} = \{y_1^{(u)}, ...y_{N_u}^{(u)}\}$, $y_i^{(u)} \in \{1, ..., K\}$.

Steps:

1. Supervised training: for each class j, let $O_j^{(l)} = \{o_i^{(l)} | y_i^{(l)} = j\}$, an HMM model is trained as

$$\lambda_j = \max_{\lambda} \sum_{o \in O_j^{(l)}} \log p(o|\lambda) ;$$

2. Data assignment: for each unlabeled sequence $o_i^{(u)}$, set $y_i^{(u)} = \arg\max_j \log p(o_i^{(u)}|\lambda_j)$;

3. Model estimation: let $O_j^{(u)} = \{o_i^{(u)} | y_i^{(u)} = j\}$ and $O_j = O_j^{(l)} \cup O_j^{(u)}$, the parameters of model λ_j is re-estimated as $\lambda_j = \max_{\lambda} \sum_{o \in O_j} \log p(o|\lambda)$;

4. Stop if $Y^{(u)}$ does not change, otherwise go back to Step 2.

Figure 14: SSHC-1 algorithm. (Semi-supervised Sequence Classification with HMMs, Shi Zhong, 2004)

The HMM is using the SSHC-1 algorithm defined in [19] to fit the HMM on the unlabelled multivariate time series data. The SSHC-1 algorithm works by taking as input a set of N_l labelled sequences $O^{(l)} = \{{o_1}^{(l)}, \dots, {o_{N_l}}^{(l)}\}$ with labels $Y^{(l)} = \{{y_1}^{(l)}, \dots, {y_{N_l}}^{(l)}\}$, N_u unlabelled sequences $O^{(u)} = \{{o_1}^{(u)}, \dots, {o_{N_u}}^{(u)}\}$, and an HMM λ structure. The algorithm is composed of four steps.

The first step of this algorithm is an initializing step (already done in our case) and consists of using the labelled sequences $O^{(l)}$ in order to have the transition matrix A and the emission probabilities B of the HMM.

After this first step, each unlabelled sequence ${o_i}^{(u)}$ will have a label given by ${y_i}^{(u)} = argmax \log P({o_i}^{(u)} | \lambda)$, where λ is the initialized HMM. In our case, the Viterbi algorithm is used in order to give a label to all the unlabelled observations.

The third step consists of the model estimation step. During this step, the parameters of the λ HMM will be re-estimated in order to fit the newly labelled data. This is done by updating the mean and covariance of the Gaussian distributions of each hidden state using the MLE [24] estimates on the observations. Then, the transition probability matrix A is updated using the new sequence of hidden states given by the Viterbi algorithm. The final step simply consists of checking if labels given by the

previous λ HMM are the same as the newly updated HMM. If this is the case, the SSHC-1 algorithm stops, otherwise it goes back to step 2.

The second algorithm uses the first and last steps of the first algorithm (SSHC-1) but modifies the second and third steps. The second algorithm frees the labelled sequences after the supervised learning step and consequently a labelled sequence given in the training set may be assigned a completely difference sequence of labels in the later semi-supervised iterative steps.

Algorithm: SSHC-2 (Step 2&3)
Steps:

2. Data assignment: for every (**labeled and unlabeled**) sequence o_i, set $y_i = \arg\max_j \log p(o_i|\lambda_j)$;
3. Model estimation: let $O_j = \{o_i^{(l)}|y_i^{(l)} = j\} \cup \{o_i^{(u)}|y_i^{(u)} = j\}$, the parameters of model λ_j is re-estimated as $\lambda_j = \max_\lambda \sum_{o \in O_j} \log p(o|\lambda)$;

Figure 15: SSHC-2 algorithm. (Semi-supervised Sequence Classification with HMMs, Shi Zhong, 2004)

The last of the three semi-supervise HMM-based classification algorithms (SSHC-3) also uses the first and fourth step of SSHC-1 but like the SSHC-2 algorithm, it also modifies the second and third steps. Compare to the first two algorithms, SSHC-3 only uses once the labelled sequences, during the supervised step. Then it only performs the model's parameters update using the trained HMM from the supervised step and the unlabelled sequences. As verified in [19], this model underperforms since it uses less data compare to the other that still uses all labelled and unlabelled data during the semi-supervised step.

Algorithm: SSHC-3 (Step 2&3)
Steps:

2. Data assignment: for each unlabeled sequence $o_i^{(u)}$, set $y_i^{(u)} = \arg\max_j \log p(o_i^{(u)}|\lambda_j)$;
3. Model estimation: let $O_j^{(u)} = \{o_i^{(u)}|y_i^{(u)} = j\}$, **using only unlabeled sequences**, the parameters of model λ_j is re-estimated as $\lambda_j = \max_\lambda \sum_{o \in O_j^{(u)}} \log p(o|\lambda)$;

Figure 16: SSHC-3 algorithm. (Semi-supervised Sequence Classification with HMMs, Shi Zhong, 2004)

This concludes this chapter on HMM types of training and leads this master thesis to the next chapter.

5. Contribution

The main contribution of this master thesis consists in the training of a semi-supervised HMM on multivariate time series representation of replays. This chapter starts by explaining the multivariate time series representation of replays, followed by a section explaining how realistic user annotation was obtained and ends with a section describing the HMM training process.

5.1. Multivariate Time Series Representation of Replays

The first task was the transformation of raw StarCraft II replays (Figure 8, 9.) into multivariate time series of twelve dimensions (Figure 10). This is needed because raw replays are not directly usable to represent the game's state at any moment of the replay. Furthermore, this simplification of the game's state into twelve dimensions will allow the removal of noise from StarCraft II replays, such as the spam of a key by the user, and it will allow more efficient training/testing of our model by limiting the number of dimensions taken into account.

A **multivariate time series** is a sequence of multiple time-dependent variables. In our case, twelve variables need to be known for each second of a replay and those values will represent the state of the StarCraft II replay at any given second. Each replay will be represented by a multivariate time series and will constitute the set of observations used by the HMM. This dataset of n replays can be represented as $D = \{r^1, \dots, r^n\}$ where $r^i = \{o^1, \dots, o^{t_i}\} \forall\ 1 \leq i \leq n$ and where t_i is the number of seconds in the i^{th} replay of the dataset D. Each observation $o^j = \{v_1, \dots, v_{12}\}$ where v_1 represents the first dimension of the j^{th}second of a replay and where v_{12}represents the 12^{th} dimension for the j^{th} second of the replay. It is important to note that all of the replays in the dataset do not have the same duration. The length of a replay (in seconds) varies from 100 seconds up to 2000 seconds.

The second task dealt with the different possible types of user annotation. In order to show that the proposed solution is viable for replay annotation, we had to train HMMs on different types of realistic user annotation.

5.2. Realistic User Annotations

Types of user annotation can vary from entirely labelled to entirely unlabelled time series, passing by partially labelled time series. While entirely labelled sequences do not cause any trouble to the HMM training, sparse annotation causes lots of troubles. This is because HMM needs a continuous sequence of labels in order to learn its transition matrix. This section will explain the different approaches followed by this work in order to deal with the sparse annotation problem.

First of all, a partially labelled multivariate time series is a multivariate time series without labels for all of its components. This means that for a sequence of observation ${o_i}^{(l)}$, its sequence of labels ${y_i}^{(l)}$ is such that there is at least one position where the label is not a user state and thus is not defined (unlabelled states are represented by the '0' sign). An extreme case would be where no labels are given by the user and, to solve this issue, our solution would have to resort to clustering techniques in order to generate some labels.

HMM needing a sequence of observations and a sequence of hidden states (labels) to be able to adjust its parameters, we decided to try two different methods in order to give a label to all unlabelled observations.

The first method consists of labelling all unlabelled observations as being part of the so called "unlabelled" state. This "unlabelled" state will be represented by the value '0'. This guarantees that the

model does not modify or manipulate the user's representation of the states but it also implies that a single state can characterize all the unlabelled portions of a multivariate time series. This new question raised will be tested and answered in section 6, the experiment section.

The second method uses multiple "unlabelled" states in order to represent all the unlabelled portions of multivariate time series. The different "unlabelled" states are represented by values ranging from '1000' to '1010'. The way used to compute the exact number of "unlabelled" states used by the model will be defined in the next section. This increase in the number of states should allow for a more precise fitting of the unlabelled data and this is why we will be looking at this in section 6, the experiment section.

5.2.1. Add an "unlabelled" State

In this section, the method used to fill the user's annotations is a method using a single "unlabelled" state. This means that all unlabelled observations will be tagged with the label '0'.

Sparse annotations of data can happen for two main reasons. The first one being that a user may have forgotten to annotate a portion of a replay or he may potentially skip parts of a replay due to a lack of time. In this case, an "unlabelled" state should not exist since the user is interested in full annotation of the data. This means that inserting a new "unlabelled" state could cause precision problems since the model will learn a specific hidden state for it. This will result in a sparse annotation of the data by our HMM.

Secondly, a user may voluntarily skip portions of a replay that are not interesting for him. This means that the model's label for some replays could be sparse in order to only label the portions of the replay the user is interested in. This is the opposite of the above example where a user would not need an "unlabelled" state.

In the case of partially labelled replays with the use of a single "unlabelled" state, the training steps are the same as for the fully-labelled training except that a new hidden state has to be added. User's partially labelled replays are used to initialized the HMM's transition probability matrix as well as the emission probabilities and then, fitting on the unlabelled data occurs.

Add Multiple "Unlabelled" States

Another method to complete the user partial annotation is to use multiple "unlabelled" states. These "unlabelled" states have an id ranging from '1000' to '1010' and if the model labels an observation as being one of those states, it will simply not tag it at all. The user will only see his/her own states (the states he/she defined during his/her replay annotation).

To find out the appropriate number of "unlabelled" states for some user's partially annotated replays, we are trying to find the best **Gaussian mixture model** that represents the unlabelled data. In order to achieve this task, the information-theoretic criterion (BIC) is used. To find the best suited model, a model selection step is necessary. The model selection is composed of two dimensions, the first one being the number of components composing the Gaussian mixture and the second one being the covariance type used by the mixtures (Figure 17). This has been implemented using the algorithm given by the Scikit-learn library and modifying it to suit our data.

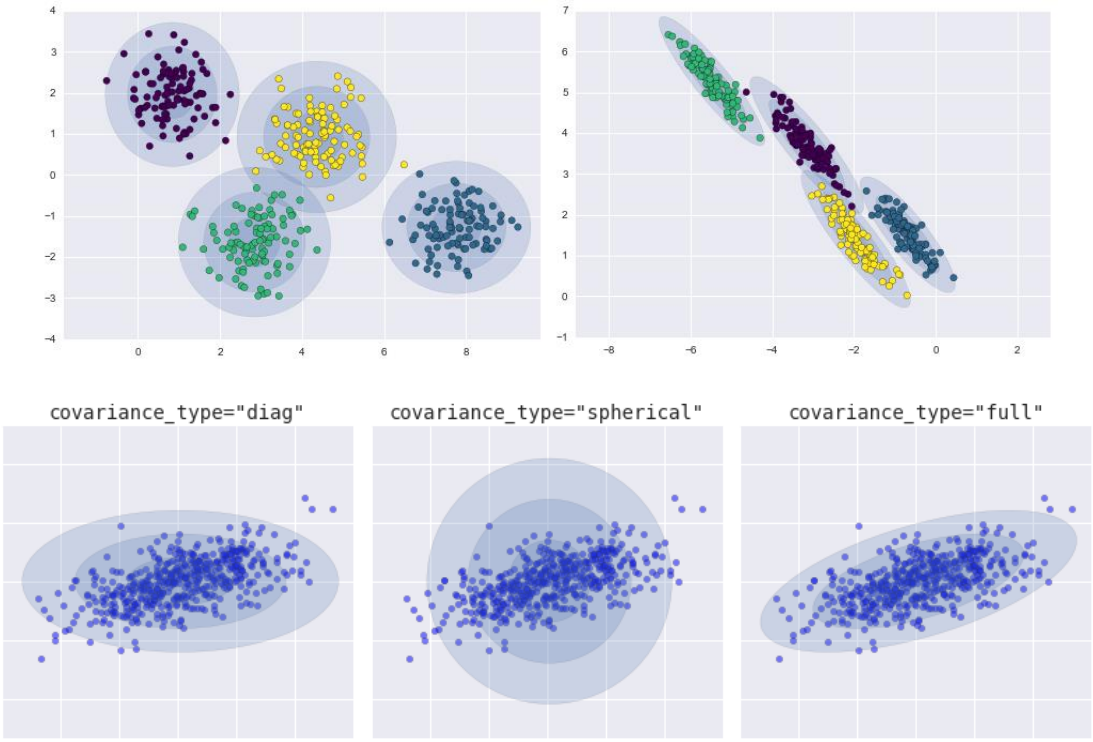


Figure 17: Example of clustering using different Gaussian mixture models (GMMs). Author Jake VanderPlas. Python Data Science Handbook, 2016. <https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html>

5.2.1.1. Model Selection Using Bayesian Information Criterion (BIC)

Choosing the best model amongst a finite set of model is a common problem lot of statisticians are facing [25]. It is necessary to use a metric that takes into account the number of parameters used by the model. This will allow the selection of the model that best fit the data without overfitting it. In our case, an extreme case of overfitting over the training set would be if the model uses a mixture for every unlabelled data point.

BIC is a parametric method and represent the likelihood criterion penalized by the model complexity in terms of number of parameters. The BIC value is computed for each of the different models following the equation [26].

$$BIC = -2 * L + \log(n) * k,$$

Where L is the maximized value of the likelihood function of the Gaussian mixture model, n is the number of observations and k is the number of parameters estimated by the model (here k=2, the number of component in the model and the covariance type of the model's components).

When all the BIC values have been computed, the best value (the lowest one) is selected and the corresponding model is chosen to label unlabelled data point (Figure 18.). The data points are given a label corresponding to the component of the Gaussian mixture they are belonging to. This means that a conversion step needs to be undertaken in order to change the label proper to the Gaussian mixture model to the corresponding label for the HMM. Labels that are obtained by the Gaussian mixture model are updated by adding 1000 to all of the labels.

$$l_i = l'_i + 1000,$$

Where l_i' is a label form the Gaussian mixture model and l_i a label used by the HMM.

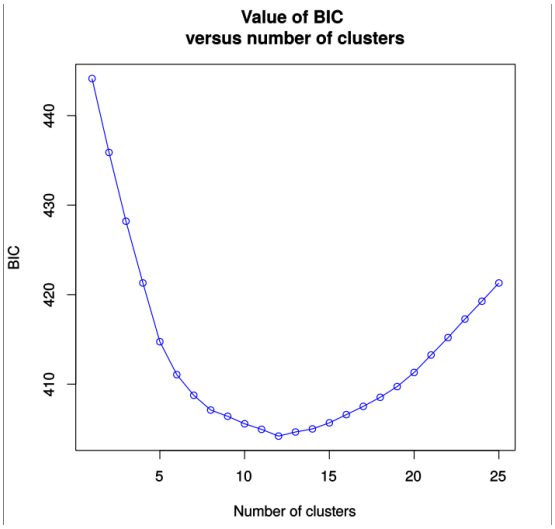


Figure 18: Graph showing the evolution of the BIC value with the number of clusters. Pierce, Magretha & Dzama, Kennedy & Hefer, Charles & Muchadeyi, Farai. (2015). Genomic population structure and prevalence of copy number variations in South African Nguni cattle. BMC Genomics.

For example, if the user partially annotated a replay of length 10 as {1,1,1,1, **0**, **0**, **0**, **0**, 2,2} (where 0s are unlabelled seconds of the time series), a Gaussian mixture model will be selected to best fit the observations where label is equal to 0. Then supposing the resulting model has 2 components, a proposed annotation of the unlabelled observations could be {1,1,2,2}. Those labels would then be updated according to the rule defined in the previous paragraph and the resulting final annotation for the replay would be {1,1,1,1, **1001**, **1001**, **1002**, **1002**, 2,2}. Which implies that the HMM now has an hidden state corresponding to label 1, 2, 1001 and 1002.

5.2.1.2. Fitting of the Gaussian Mixture with EM Algorithm

The Gaussian mixture model of Scikit-learn implements and uses the **expectation-maximization** (EM) algorithm for the fitting of the model. This algorithm is composed of two major steps [27]. The first step is the expectation step (E-step) which estimates the missing variables in the dataset. And the second step is the maximization step (M-step) that maximizes the parameters of the model in the presence of the data (Figure 19).

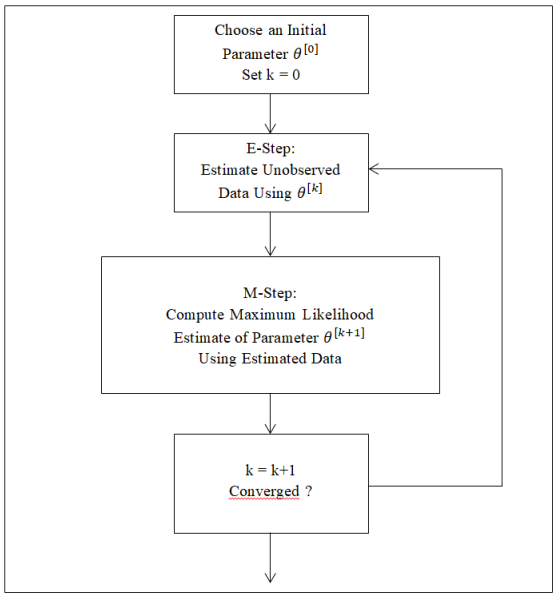


Figure 19: An overview of the EM algorithm. After initialization, the E-step and the M-step are alternated until the parameter estimate has converged (no more change in the estimate). (Moon, T. K. (1996)) [27]

Finally, all the unlabelled observations have a label and the training steps are now the same as in fully labelled replays but with the difference that "unlabelled" states have been added to the user's states and HMM's hidden states.

5.3. HMMs training

The third task was the actual training of an HMM in a semi-supervised way. To achieve this task, the SSHC-1 algorithm described in [19] was chosen and implemented using the Python programming language. This algorithm uses the labelled data (data provided by the user) to constrain the HMM. It is clear that the accuracy of the HMM will depend on the quality of the user's labelled multivariate time series and the amount of user's multivariate time series. This is why an important work on managing the different types of user's annotation was done. The **SSHC-1 algorithm** was chosen because compare to SSHC-2/3, it retains the user 'labels and only attributes new labels to the unlabelled replay, thus not modifying the user's input.

If all of the observations contained in all of the multivariate time series of the user have a label, the HMM training can be done without intermediary steps. This is because if all observations have a label, there is no need to insert an 'unlabelled' hidden state to the model (to represent unlabelled observations of the training replays). If it is not the case, the user has to choose between using one or multiple "unlabelled" state(s).

The training of HMM is also influenced by the presence or the lack of unlabelled replays. Since our solution's plan is to resort to semi-supervised training, the user can decide to add unlabelled replays or not. Now having a sequence of labels for every replay and a list of unlabelled replays (or an empty list), the training of the HMM in a semi-supervised way can occur using the SSHC-1 algorithm.

The next chapter focuses on conducting some experiments in order to verify the usefulness of HMMs for the sequence labelling problem.

6. Experiments

This chapter concerns all the processes needed in order to make valuable experiments for the proposed method. This starts from the data extraction from raw StarCraft II replays and ends in test of the method. The first section introduces the data extraction process from raw replays and the creation of the datasets from those extracted data.

6.1. Dataset Extraction

The dataset used during this research has been created from StarCraft II replays downloaded from https://lotv.spawningtool.com/. This website is specialized in StarCraft II (related) content and contains hundreds of pro replays.

197 replays are used to generate our dataset and all of them are replays of 1 vs 1 pro match. This means that we can get two points of views for each of the replay, the point of view of player 1 and the one of player 2, thus giving us a total of 394 entries in our dataset. As mentioned in the introduction, replays are essentially an ordered list of the players' actions. This format of replay makes it convenient for the StarCraft II engine to render a replay but it makes it difficult to represent the state of the game at any given time. Our model will need to label each second of a replay as being either part of one of the user's defined states or the model's unlabelled state. This means that a pre-processing step needs to be applied on the replays before using them as training material. The pre-processing step has to convert the 197 StarCraft II replays into 394 multivariate time series with as dimensions things such as the number of military unit, the number of workers, the number of buildings, etc.

6.1.1. Dataset Creation

The dataset creation process is composed of two steps. First, StarCraft II replays are converted into a multivariate time series of 6 dimensions. Second, the first derivative of each of the 6 dimensions will be computed and added to the time series. This will allow our model to bypass the Markov hypothesis (hypothesis that the modelled process is Markovian) as well as to more easily make links between the values for the dimensions of a replay at time t and the values at time t+1.

Game's information is extracted from a replay using the **sc2reader** Python library, which is open source and MIT licensed. This library allows the extraction of various information contained in StarCraft II replays and is combined with our own python code (Figure 20.) to extract the 6 dimensions we need. Having those 6 dimensions, we will be able to make the conversion of the replay file into a time series of 6 dimensions. The 6 dimensions desired are:

- The worker count : "workers"
- The army count : "army"
- The building count : "building"
- The enemy worker killed count : "enemy_scv_killed"
- The enemy army killed count : "enemy_army_killed"
- The enemy building destroyed count : "enemy_building_killed"

The time series will have values for each of the 6 dimensions for every second of any StarCraft II replay.

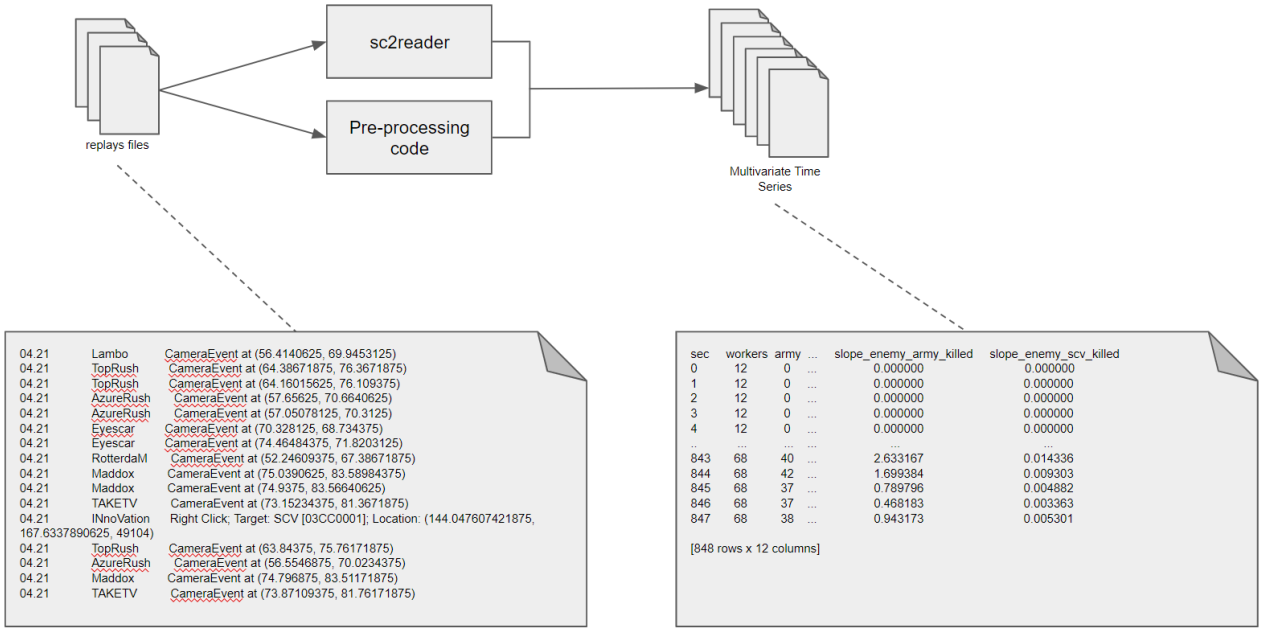


Figure 20: Action of transforming raw replay files into multivariate times. This is achieved by using the sc2reader python library and our own python code to read and manipulate replays in order to count the number of unit killed at any given second of replays, etc.

Since replays do not key track of the value of those 6 dimensions, we had to keep track of those values in our "Pre-processing Code". This was done by initializing the value to their default start according to the player's faction (Table 1.).

	workers	army	Building	enemy_Building_killed	enemy_army_killed	enemy_scv_killed
Terran	12	0	1	0	0	0
Protoss	12	0	1	0	0	0
Zerg	12	1	1	0	0	0

Table 1: initial values of the 6 dimensions according the player's faction

After initializing the dimensions, the time series is filled with values for each of the dimensions for each second of the game and for each of the two players. In this study, we are only focusing on 1vs1 game and so we start computing the first 3 dimensions ('workers', 'army' and 'Building') for the player 1 and then for the player 2 since those values can easily be computed.

Having done this step, we can compute the number of enemy workers, army and building units killed for each of the two players by looking at the values of 'workers', 'army' and 'Building' of the other player. This is done by looking at the value of a dimension at the time t and looking if at time t+1 this value increased or decreased. This has been done in order to better estimate the number of enemy units killed since the classic StarCraft II replay format does not allow an easy way to figure out these values.

Finally, after completing the time series for the 6 dimensions for every second of the replay, we can compute the slope for the 6 dimensions for a final number of 12 dimensions for each second of a StarCraft II replay.

6.1.2. Dataset Completion with the First Order Derivative

To compute and have a precise first derivative, we first have to smooth the values for the six dimensions of our time series. This is necessary because the number of any type of units easily makes "stair" diagrams as you can see in blue in the graphs below (Figure 21.).

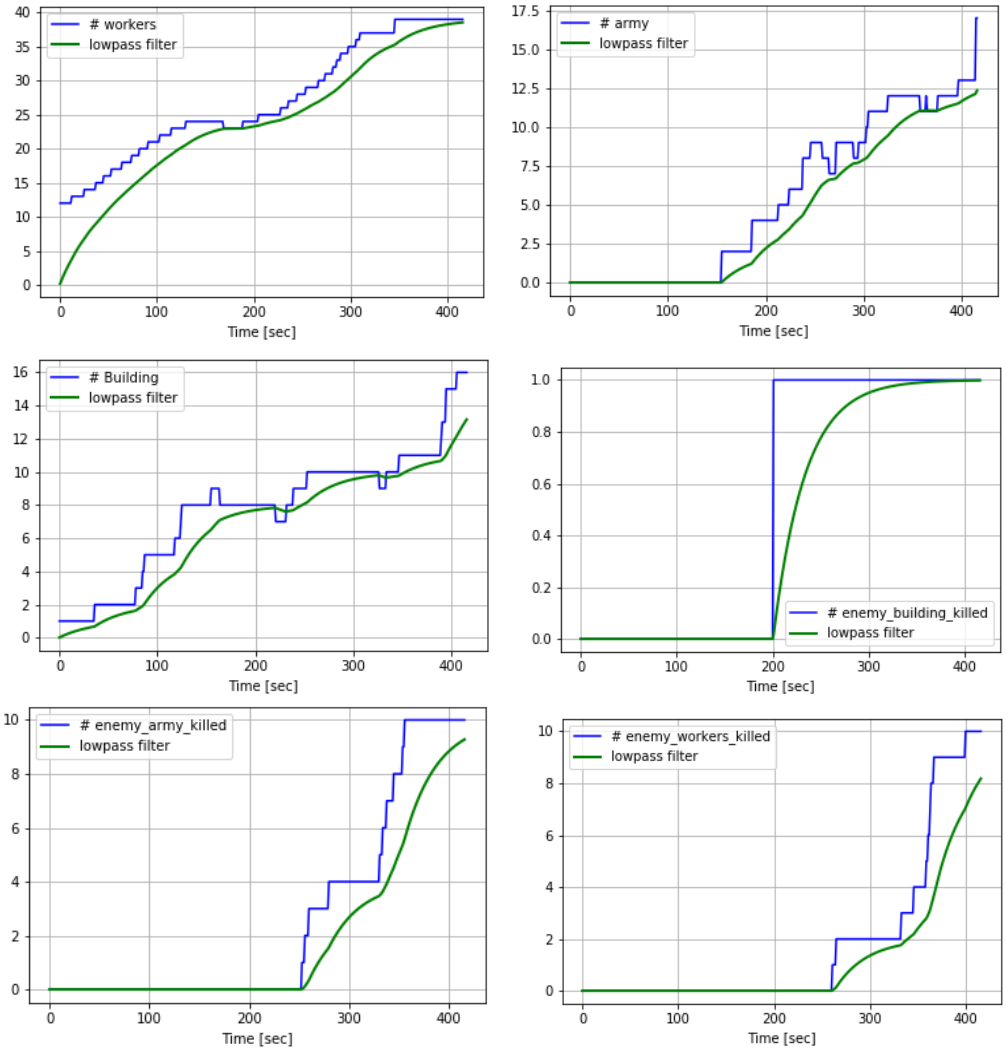


Figure 21: graphs of the 6 dimensions over time (before & after smoothing). The blue line represents the raw data extracted from a replay file. The green line represents the smoothed version of the raw data and is obtained using a lowpass filter.

In order to smooth the values of the six current dimensions, a **low pass filter** is applied on them. The choice of this method was discussed during the development of this master thesis and was further motivated by [28]. This filter was applied in order to remove as much high-frequency noise as possible and in order to compute a more precise first derivative. Several procedures exist in order to find the best low pass filter adapted to the data [28], such as the nearly equal ripple approximation (NER) or the nearly equal ripple derivative filter (NERD), but in our case, the low pass filter was manually tweaked until satisfactory results occurred (Figure 21.).

After this smoothing step, we use the **linregress** statistical function to calculate a linear least-squares regression for our dimensions and then get the slope of the regression line. **Linregress** uses a segment of 15 seconds in orders to compute the first order derivate of any dimension.

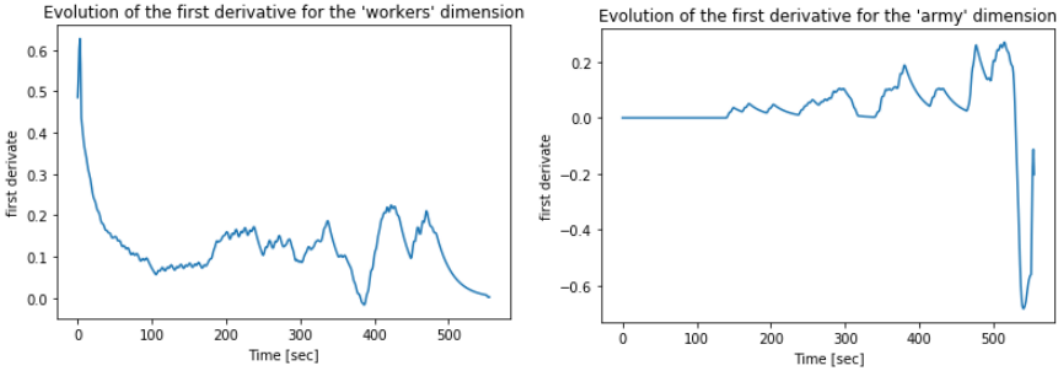


Figure 22: Graphs showing the evolution of the first derivate (slope) for the 'workers' dimension (left) and 'army' dimension (right).

6.2. Evaluation Protocol

This section aims at explaining the protocol used to evaluate the quality of our model. First of all, the evaluations of the models are done on testing replays and this means that none of those replays have been used during the training process. The testing set of replays is composed of 247 replays and models evaluation is done 30 times on 20 replays each time. This allows us to have the mean score of the model as well as the 95% confidence interval and the variance.

The specificity of our evaluation protocol is that we first constituted 30 testing sets of 20 random replays before computing the score of the different models on those 30 testing sets. This guarantees an equal and correct evaluation of the models thus allowing the comparison between the models' score.

The models are evaluated according to three different metrics. Those values are the f1-score, the recall and the precision of the model. Those values are able to be computed only if we can have the correct sequence of labels for a given replay. This is why we decided to use labelling rules to get the correct answer. This also means that the states trying to be learned by the models can be described using simple to complex rules and thus the comparison of the models label and the correct labels is possible.

Finally, it is important to clarify that the model's evaluation will only happen for parts of the replays where a label is expected. As schematized in (Figure 23.), the top parts illustrate the evaluation of a replay annotation where a label is expected for every second of the replay (labels are 's0', 's1' and 's2'). A red cross symbolizes where the model was evaluated and where the annotation was wrong or missing. The green square represents correct annotation by the model. An empty space (not a red cross or a green square) represents a section (a second or multiple seconds) of the replay that is not taken into account by the evaluation process. Sections like this are section where no labels are expected (Figure 24.) and are represented by "/" in the correct labels timeline.

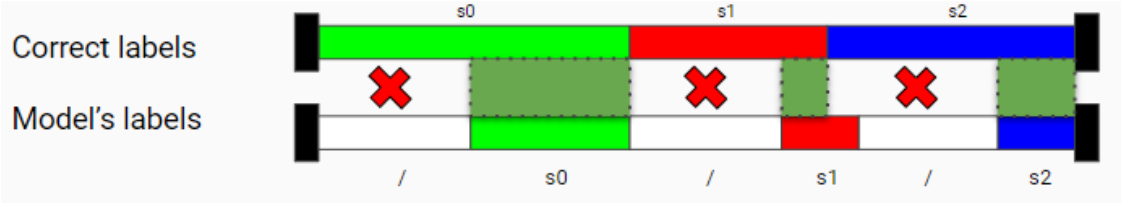


Figure 23: Example of evaluation showing the part of the replay used in the evaluation process (the green regions between the correct label and the Model's labels)

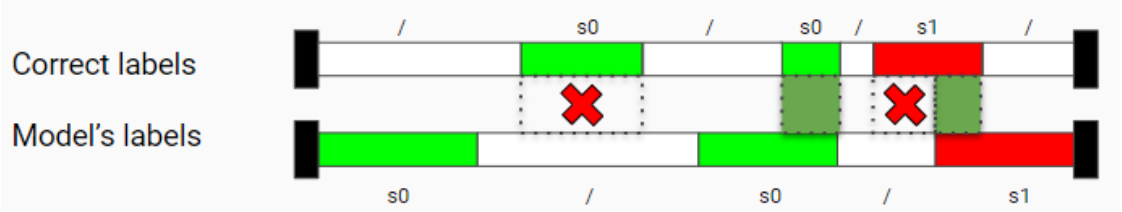


Figure 24: Example of evaluation showing the part of the replay used in the evaluation process (the green regions between the correct label and the Model's labels)

6.2.1. Recall, Precision and F1-Score

This section explains the three evaluation method used for the HMMs evaluation. Those 3 methods are the recall computation method, the precision computation method and the f1-score computation method [29]. But first, we need to define what 'true positive', 'false positive', 'true negative' and 'false negative' means and to do this a simple example with two classes is chosen. The two possible classes are the 'positive' class and the 'negative' class. In this context,

- 'True positive' = correctly identified label as being 'positive'
- 'False positive' = incorrectly identified label as being 'positive'
- 'False negative' = incorrectly identified label as being 'negative'

In our case, there can have more than two classes but this does not change things since a multiclass problem can be represented as a set of multiple two-class problem.

The first metric is

$$Recall = \frac{true\ positive}{(true\ positive + false\ negative)}$$

The recall is the ability of the model to find all the positive samples and the best score recall can achieve is 1 and the lowest being 0.

Next, the **precision** computation method is.

$$\text{Precision} = \frac{true\ positive}{(true\ positive + false\ positive)}$$

The precision is a bit different from the recall since precision expresses the ability of the model not to label as positive a sample that is negative.

Finally, we will explain the **f1-score**.

$$f1 = 2 * \frac{(precision * recall)}{(precision + recall)}$$

The f1-score value is computed from the precision and the recall value by taking their harmonic mean. The f1-score is a value that seeks a balance between the precision and the recall thus providing a more accurate measurement for our model. The best f1-score value is 1 and implies that the recall and the precision have 1 as value as well.

6.2.2. Labelling Rules

As mention in section 5.2, labelling rules are used in our evaluation method. This section aims at explaining what are those methods and how are they used in the evaluation process.

Our labelling rules are used to allow the computation of the recall and the precision score which in turns allows the computation of the f1-score. The reason our labelling rules allows the computation of those metrics is that our rules gives the correct expected label for any given second of any given StarCraft II replays. This means that we can now compare the labels output of the different models with the correct expected value and so compute the number of 'true positive', 'false positive' and 'false negative'.

Labelling rules are used during the experiments section of the thesis.

6.3. Experiments Conducted

In order to evaluate the usefulness of HMMs as sequence labelling tools for automatic replay annotation, three experiments have been conducted. Each experiment is focused on a specific parameter of the model in order to see its effect on the HMM's efficiency. As stated in section 6.2.1, the f1-score, the precision and the recall values will be utilized in order to follow the model's efficiency according to the different parameters used in the experiments.

The three experiments are also done on two distinct cases, one simple case and another more challenging case. This is done in order to evaluate the HMM's efficiency on different scenarios with different learning complexity. The complexity of a state to be learn is linked to how often this specific state appears during a replay and how many dimensions (from the twelve dimensions used to represent the game's state) are required in order to create rules that finds all of the occurrence of this state.

The first case is called the **"simple case"** and is composed of 3 states. Those three states are characterized using one dimension, the number of workers of the player, and all seconds of a replay should be labelled as one of those three states (Figure 25.). The labelling rule used for this case is the following. Given a replay $r = \{o_1, \dots, o_t\}$, where o_i is the i^{th} second of the replay represented by a vector of twelve dimensions (workers, army, Building, etc.). The label expected for o_i is given by

$$\forall\, o_i in\, r, rule(o_i) = \begin{cases} 1\ if\ (the\ number\ of\ workers\ alive\ during\ o_i) \leq 40 \\ 2\ if\ 40 < \ (the\ number\ of\ workers\ alive\ during\ o_i) \leq 70 \\ 3\ if\ (the\ number\ of\ workers\ alive\ during\ o_i > 70 \end{cases}$$

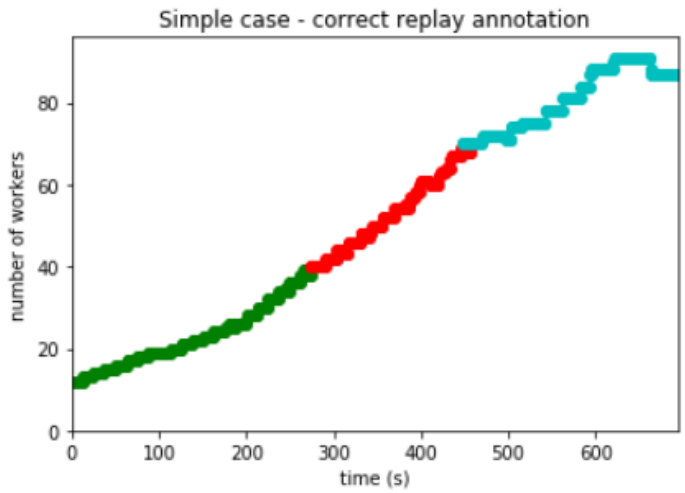


Figure 25: Example of correct replay annotation for a simple case. The annotation is symbolized by the colour code (green=state 1, red=state 2 and cyan=state3).

The second case is referred as the **"complex case"** and is composed of 2 hidden states. One state is used for the military fights and the other state is used for the attacks against workers. This case is more complex due to the rarity of such labels in replays and because in order to create a labelling rule for this case, at least 4 dimensions are needed and the rule needs to look at several observation of the sequence instead of only one. Those four dimensions are the army and worker count, the number of enemy military unit killed and the number of enemy workers killed. The correct sequence of labels for this complex case is also obtained using a rule and this rule tags as '1' the fights between military units if at least 6 military units died in a row and if less than 5 seconds separates each death (in green) and tags as '2' if an attack on workers happens and if at least 3 workers died with less than 5 seconds between each worker death (in red). In a case where a military fight happens at the same time as a fight against workers, the second event takes precedence and thus labels the second as being '2'. (Figure 26. shows an example of correctly labelled replay for the complex case.).

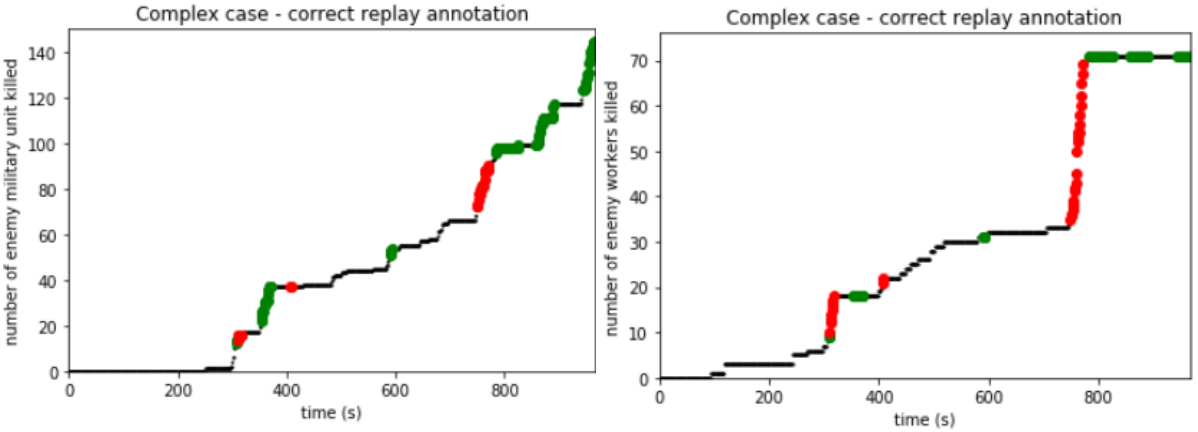


Figure 26: Example of correct replay annotation for a complex case. The annotation is symbolized by the colour code (green=state 1, red=state 2). Two different views of the same replay.

Each models used during the three experiments were models trained using the SSHC-1 algorithm which trains HMMs on labelled and unlabelled sequences of observations. In the case where no unlabelled data were used, the training of the model was done using supervised learning.

6.3.1. Experiment 1 - The Effect of Unlabelled Replays

This experiment has been performed to compare HMMs trained on different quantities of unlabelled replays. This experiment has been performed on the simple and on the challenging case. For each case, the experiment was repeated three times. One with the number of labelled replays set to 1, a second time with 15 labelled replays and a third times where there were 50 labelled replays.

During the training of the HMMs, the labelled replays of the simple case were all entirely labelled and for the complex case, all the military fights and attacks on the workers have been labelled. Nevertheless, the complex case has most of its labels consisting of the "unlabelled" state since the labelling rule doesn't label observation not belonging to fights or workers attacks.

6.3.1.1. Simple Case

The f1-score, precision and recall graphs for the different values of labelled replays are stationary and do not seem to fluctuate with the number of unlabelled replays.

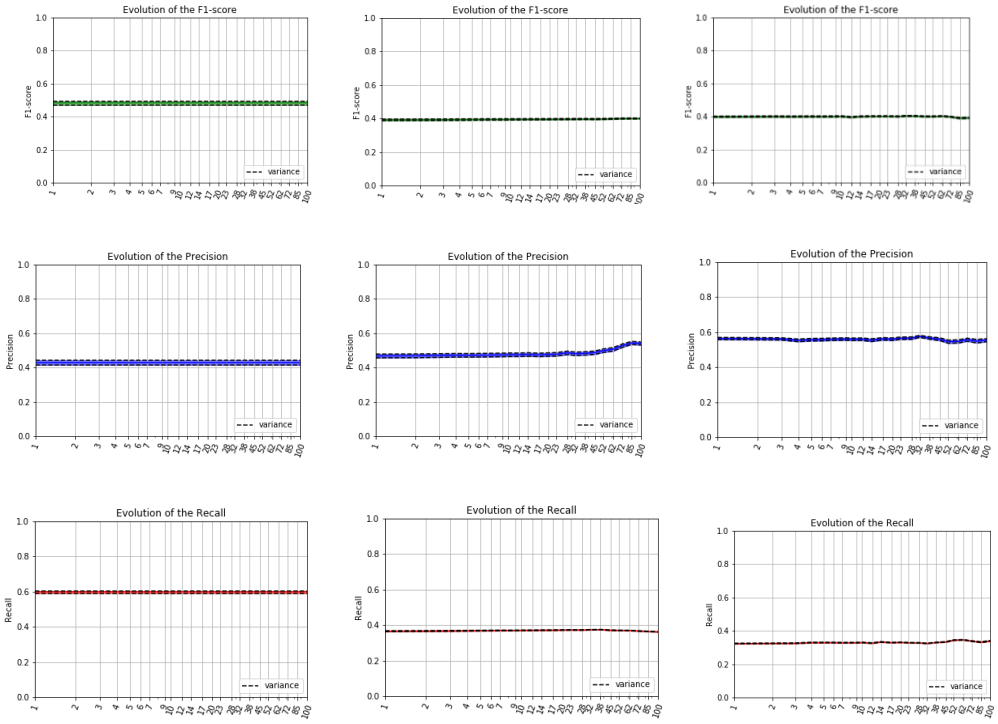


Figure 27: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 1 (left), 15 (middle) and 50 (right) labelled replays and a varying number of unlabelled replays.

The figures in (Figure 27.) show a constant value for their respective score but also show that the variance decreases with the number of labelled replays used during the training of the model. Furthermore, HMMs seem to correctly learn the different states since the means of the workers is between the ranges fixed by the labelling rule (see section 6.3 for more information about the labelling rule, results in Table 2.).

State	Mean of workers
'1'	26.59
'2'	58.03
'3'	75.94

Table 2: Mean of workers for state '1', '2' and '3'. This are the values for the HMM trained with 50 labelled replays and 72 unlabelled replays.

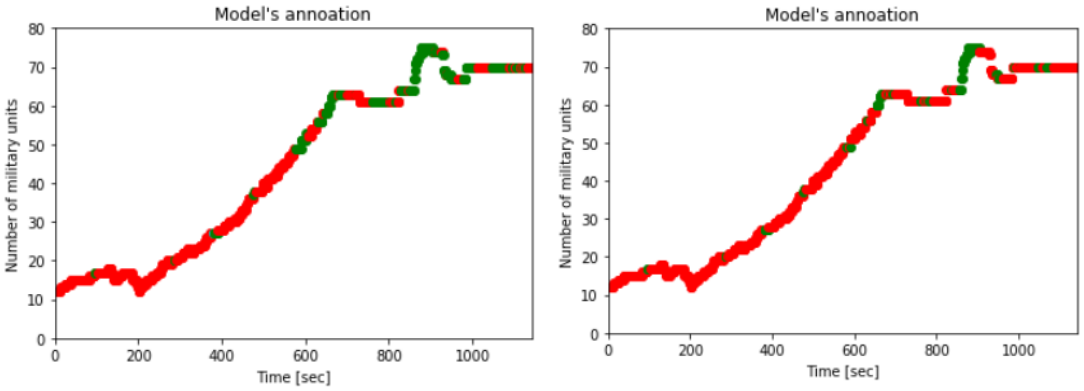


Figure 28: example of replays annotation using an HMM trained with 15 labelled replays and 0 unlabelled replays (left) and 100 unlabelled replays (right).

6.3.1.2. Complex Case – One "Unlabelled" State

For the complex case, graphs stopped being constant and take a flatten bell shape when the number of labelled replay used for training is 15. When the number of labelled replays is 50, the bell completely flattened but the curve seems to increase when the number of unlabelled replays passes 45 unlabelled replays.

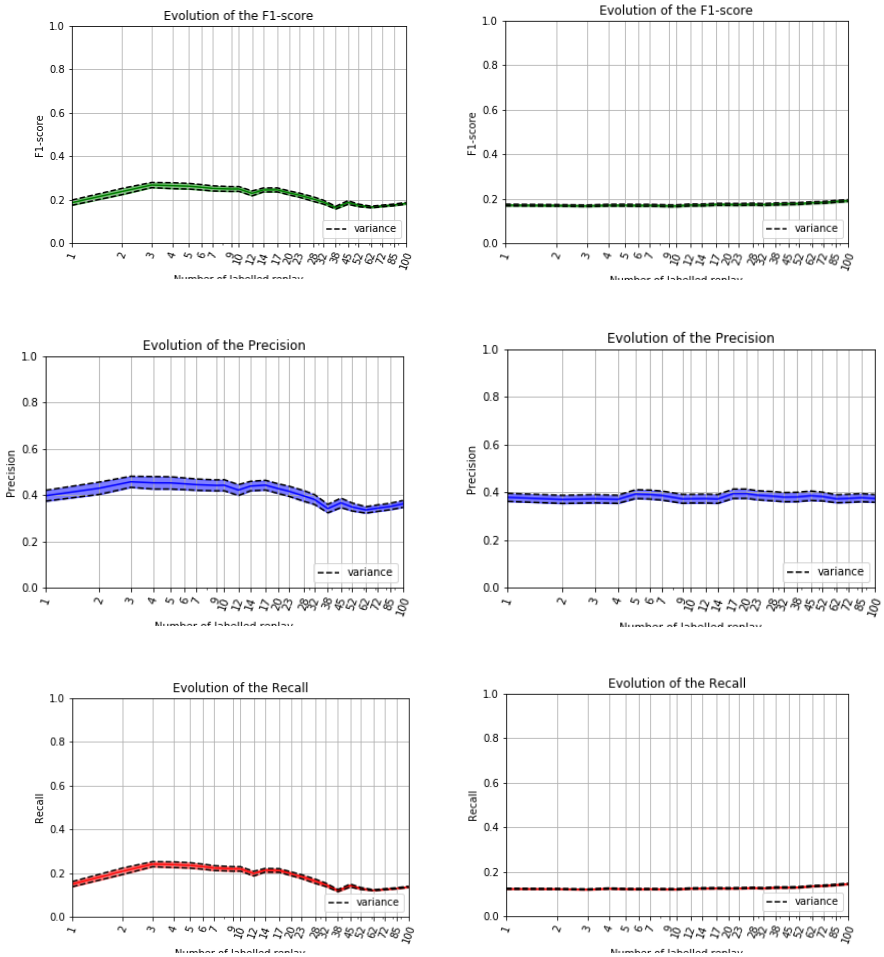
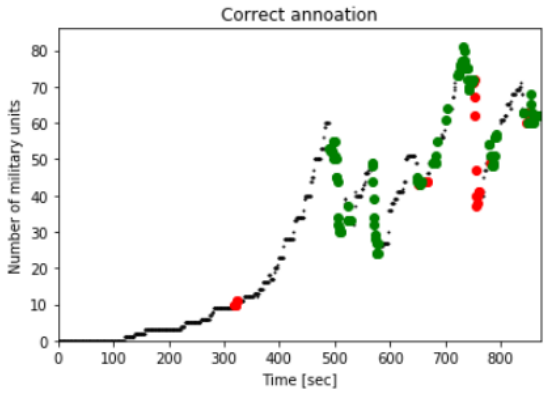
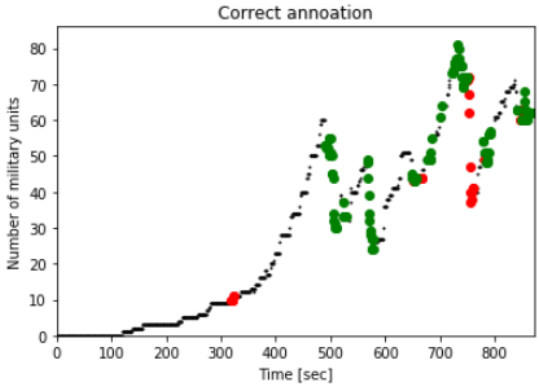
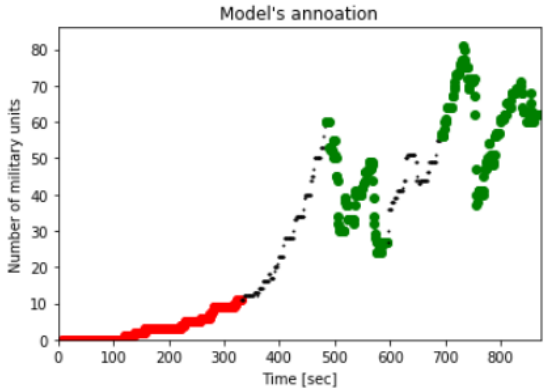


Figure 29: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 15 labelled replays (left), 50 labelled replays (right) and a varying number of unlabelled replays.

Here are two example of replay annotation using an HMM that maximizes the f1-score. The first annotation is given on a random replay using an HMM trained with 15 labelled replay and 3 unlabelled replays (Figure 29.) and the second example of annotation is given using an HMM trained on 50 labelled replays and 100 unlabelled replays (Figure 29.). Their respective correct annotations (using the labelling rules) are given as well. Each colour represents a hidden state and the black colour represents unlabelled seconds of a replay.







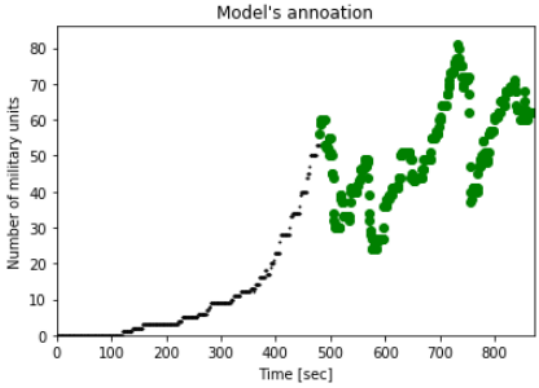


Figure 30: Correct and model's annotation for a replay.

Figure 31: Correct and model's annotation for a replay.

The model's annotations show that the HMM has difficulty in learning the transitions. In figure 31, there is only one transition from one hidden state to another while the correct annotation should have a dozen transitions. Nevertheless, the annotations of the first model (Figure 30.) are quite similar to the correct annotation while still having trouble in learning fast changing states. The first model also correctly identified the main uninteresting parts of the replay (from 350 sec to 480 and from 600 to 650)

6.3.1.3. Complex Case – Multiple "Unlabelled" States

When multiple hidden states are used to represent the unlabelled seconds of a replay, the overall model's efficiency stays the same, compare to the HMM using a single hidden state to represent unlabelled seconds, and converges at the same values (0.2 for the f1-score, 0.4 for the precision and 0.18 for the recall). Using multiple "unlabelled' hidden states completely remove the bell-shape of the graphs and as observed by the initial metrics' value, passing from 15 labelled replays to 50 does not seem to help the learning process of the HMM. Nevertheless, Adding 100 unlabelled replays seemed to slightly improve the model (passing the f1-score value from 0.1 to 0.2, the precision from 0.3 to 0.4 and passing the recall from 0.1 to 0.15).

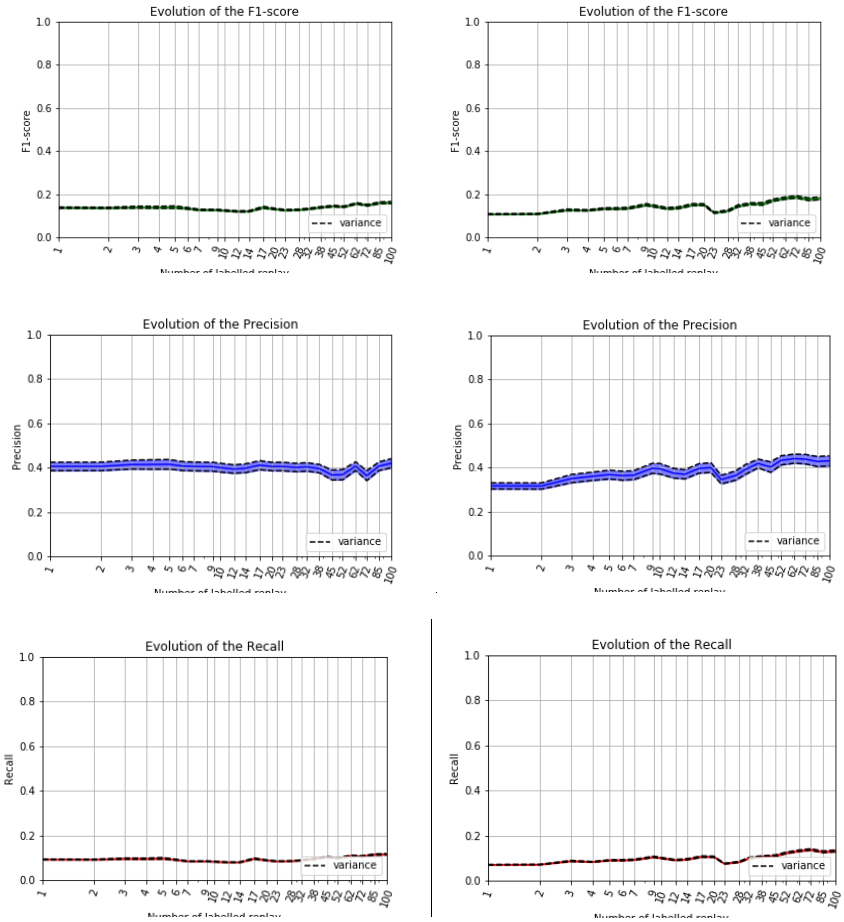


Figure 32: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 15 labelled replays (left), 50 labelled replays (right) and a varying number of unlabelled replays.

Here are two examples of annotations using an HMM trained on 50 labelled replays. The first annotation is produced by an HMM trained with a single unlabelled replay (Figure 33.) and the second annotation used an HMM trained with 100 unlabelled replays (Figure 34.). Like in section 6.3.1.2, the correct annotations are given as well to allow the comparison between the models 'annotations.

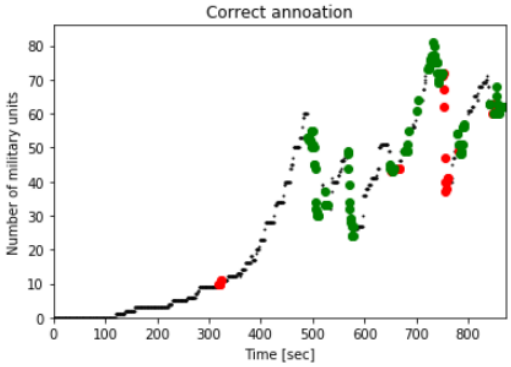


Figure 33: Correct and model's annotation for a replay.

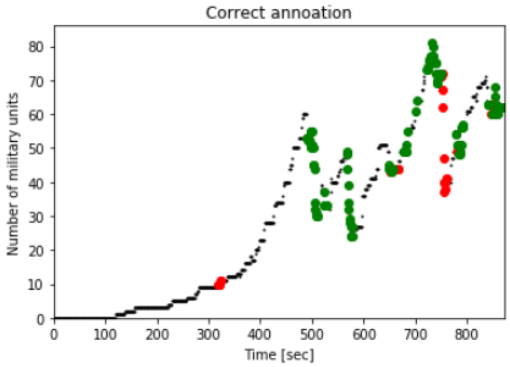
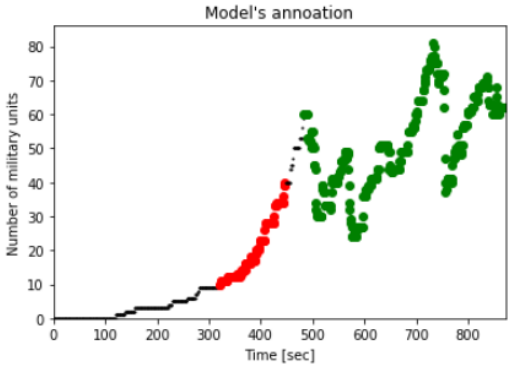
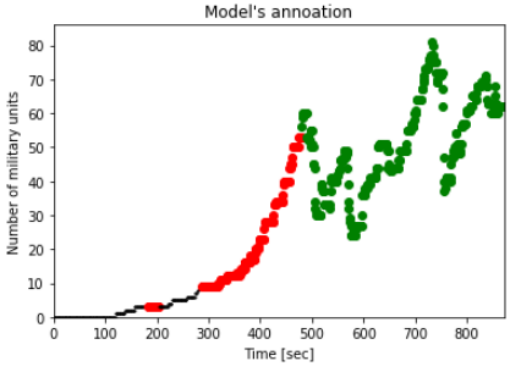


Figure 34: Correct and model's annotation for a replay.





In both annotations, the HMM does not seem to correctly learn annotations with rapidly changing hidden states. This is shown by the lack of unlabelled states (black colour) after the 500 seconds mark.

6.3.2. Experiment 2 - The Effect of Labelled Replays

This experiment has been performed to compare HMMs trained on different quantities of labelled replays. It is the opposite of the previous experiment and it fixes the number of unlabelled replays while varying the number of labelled replays used during the HMM training phase. HMMs have been trained on three quantities of fixed unlabelled replays. The first quantity is 0, the second one is 50 and the last one is 100.

6.3.2.1. Simple Case

In this case, for the three values of unlabelled replays used, the f1-score and precision score tends to increases when the number of labelled replays increases and seems to get constant when HMMs are using four or five labelled replays. For the recall metric, it seems to slightly decrease when labelled replays are added in the training process while still converging at the end. The variance of the three scores also hugely decreases when more labelled replays are used.

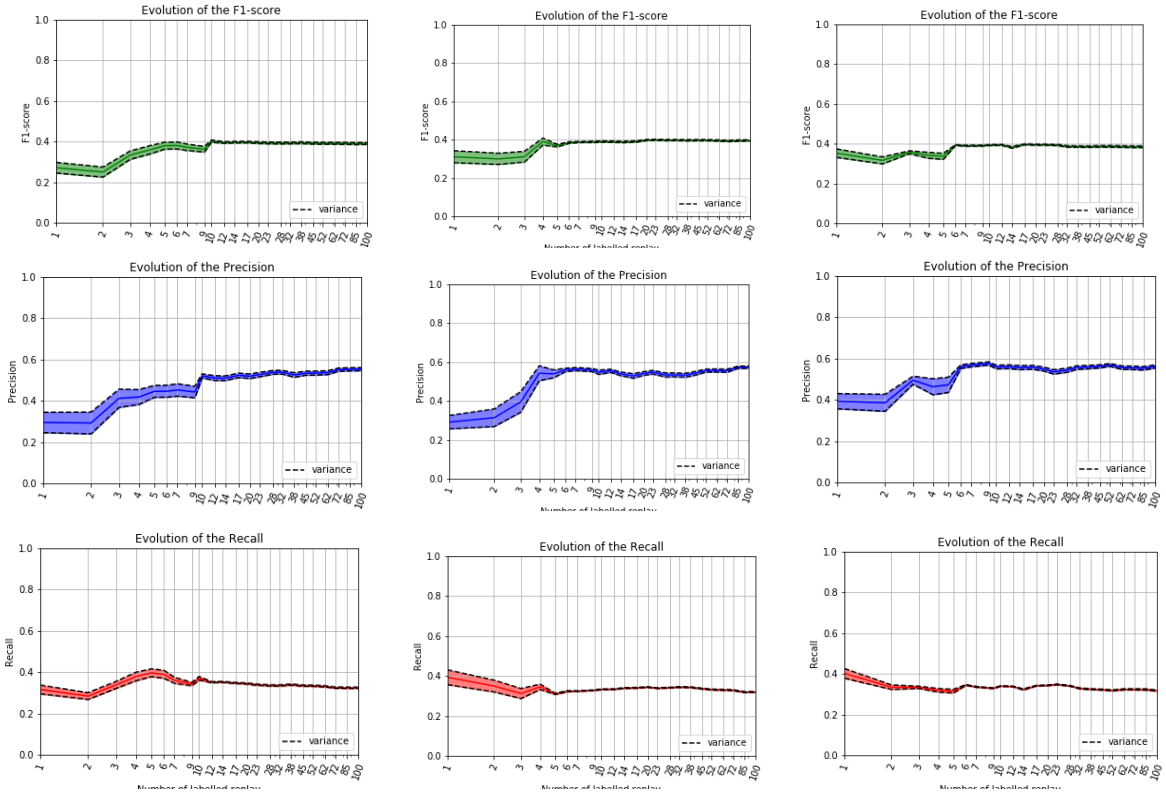


Figure 35: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 0 (left), 50 (middle) and 100 (right) unlabelled replays and a varying number of labelled replays.

Adding more and more labelled replays, if the user takes the time to give lots of correct annotations, the metrics of the HMMs (f1-score, precision and recall) all converge towards the same values even though models uses different number of unlabelled replays for training (0, 50 or 100). This would suggest that using unlabelled replays for a simple case (a case equivalent to the first labelling rule) is useless.

6.3.2.2. Complex Case – One "Unlabelled" State

In the complex case with 50 unlabelled replays and with one "unlabelled" state, an interesting pattern appears. F1-score, precision and recall graphs all show a rapid increase in their values when the number of labelled replays increases, directly followed by a steady descent after their peak values have been achieved.

The same happens when the fixed number of unlabelled replays is equal to one hundred and in this case, another interesting thing happens. It seems that using 100 unlabelled replays (instead of 50) allows the HMM to maximize its f1-score value before an HMM using 50 unlabelled replays.

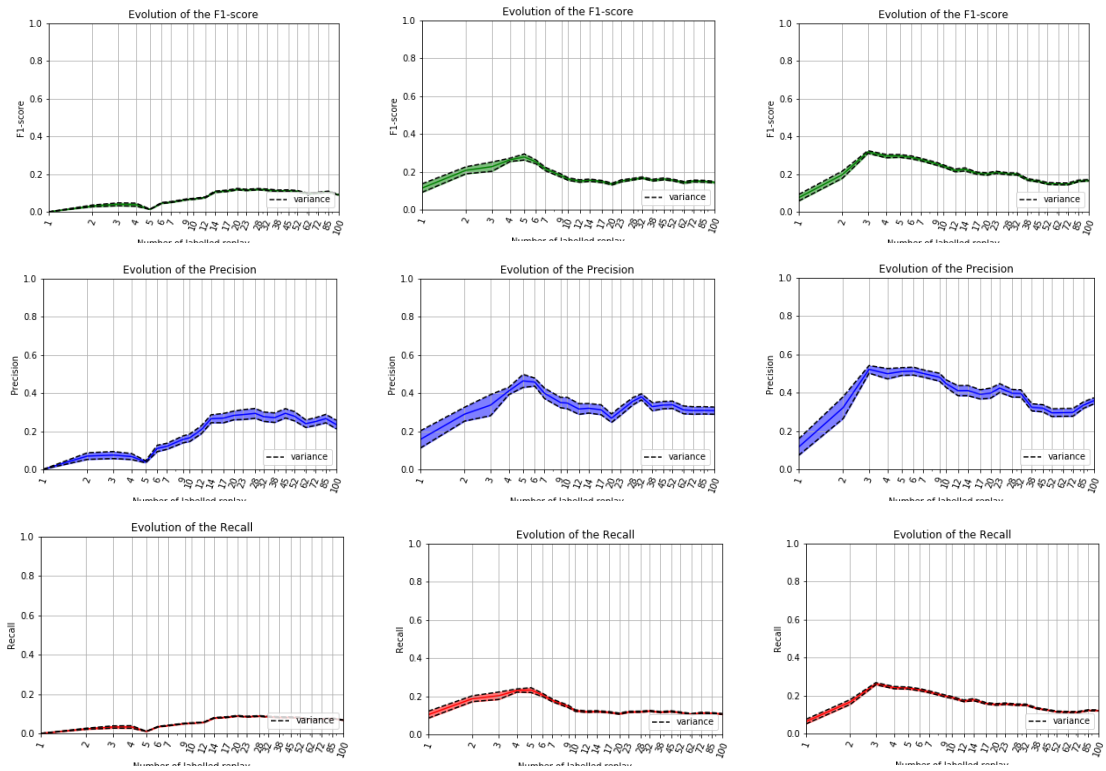


Figure 36: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 0 (left), 50 (middle) and 100 (right) unlabelled replays and a varying number of labelled replays.

Two example of replay annotations will be displayed to show annotation differences between an HMM that stopped when the f1-score reached its maximum (HMM using 100 unlabelled replays and 3 labelled replays) (Figure 37. and Figure 38.) and an HMM that used 100 labelled replays (Figure 39. and Figure 40.) This result is quite surprising since adding labelled replays should help the model (as observed in the simple case, section 6.3.2.1).

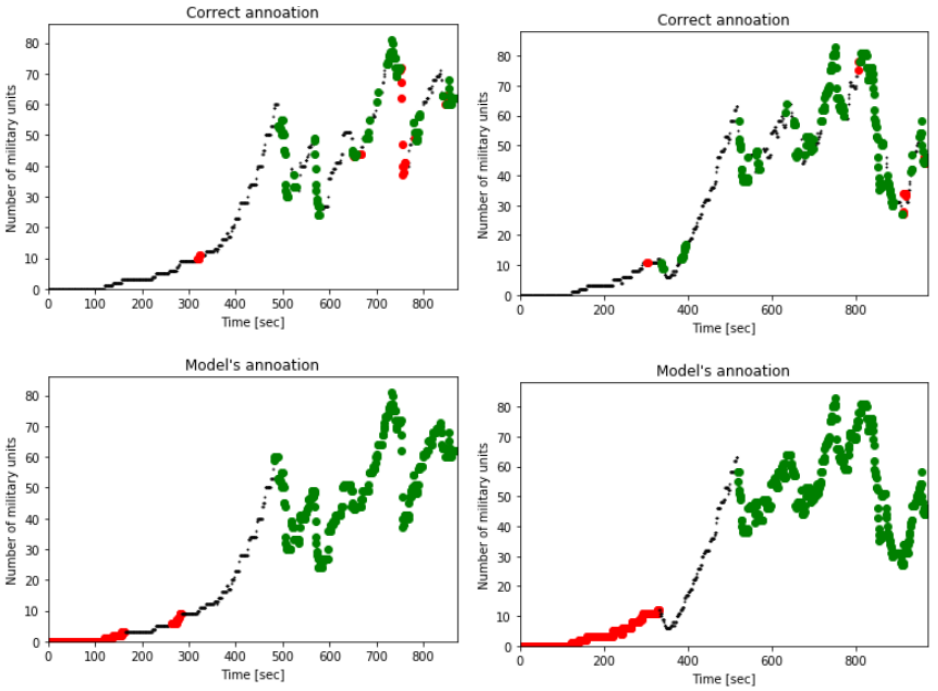


Figure 37: Correct and model's annotation for a replay.

Figure 38: Correct and model's annotation for a replay.

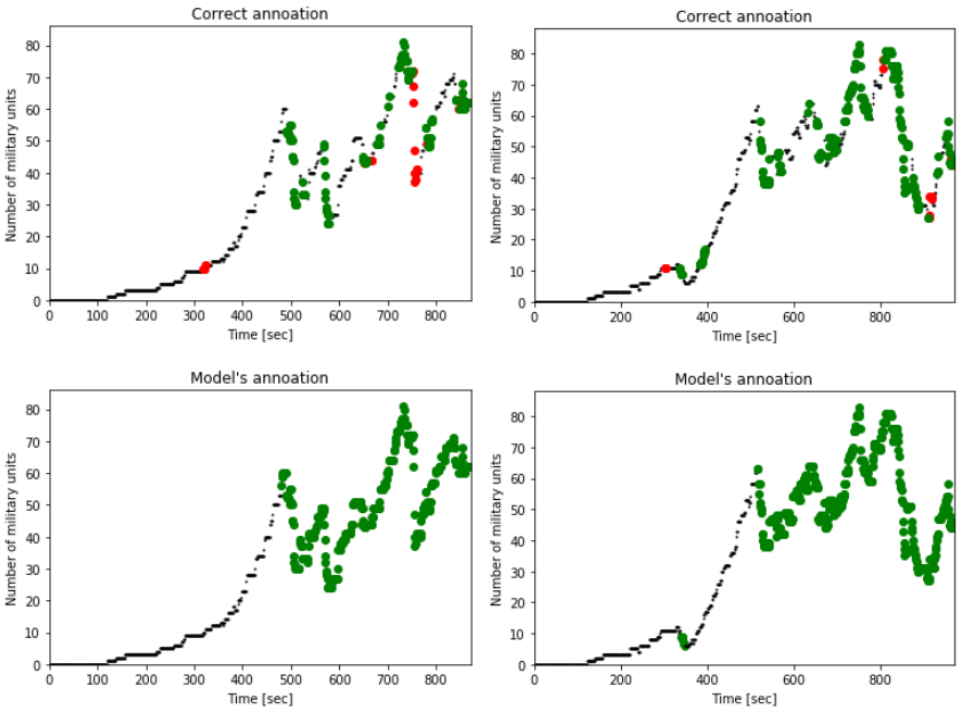


Figure 39: Correct and model's annotation for a replay.

Figure 40: Correct and model's annotation for a replay.

In both replays, the annotation of the HMM trained with 100 labelled replays is lacking the state '2' (red colour) while the HMM trained with only 3 labelled replays always used the state '2' label. Hidden state '2' representing attacks on workers, which is rare comparatively to army on army fights, this could mean that hidden state '2' was drowned by the unlabelled hidden state (black colour) and the hidden state '1' (green colour). In both HMMs, the transition matrix is similar, thus meaning that the learning of the emission probability is the main cause for this decrease in efficiency.

6.3.2.3. Complex Case – Multiple "Unlabelled" States

Results in figure 41 have a similar pattern to figure 36. Adding labelled replays seems to increase the scores of all the three metrics but passing a certain value of labelled data, the scores suddenly drops and seems to converge. Contrarily to the case with a single "unlabelled" state, here the graphs seem to display a more saw-tooth like shape.

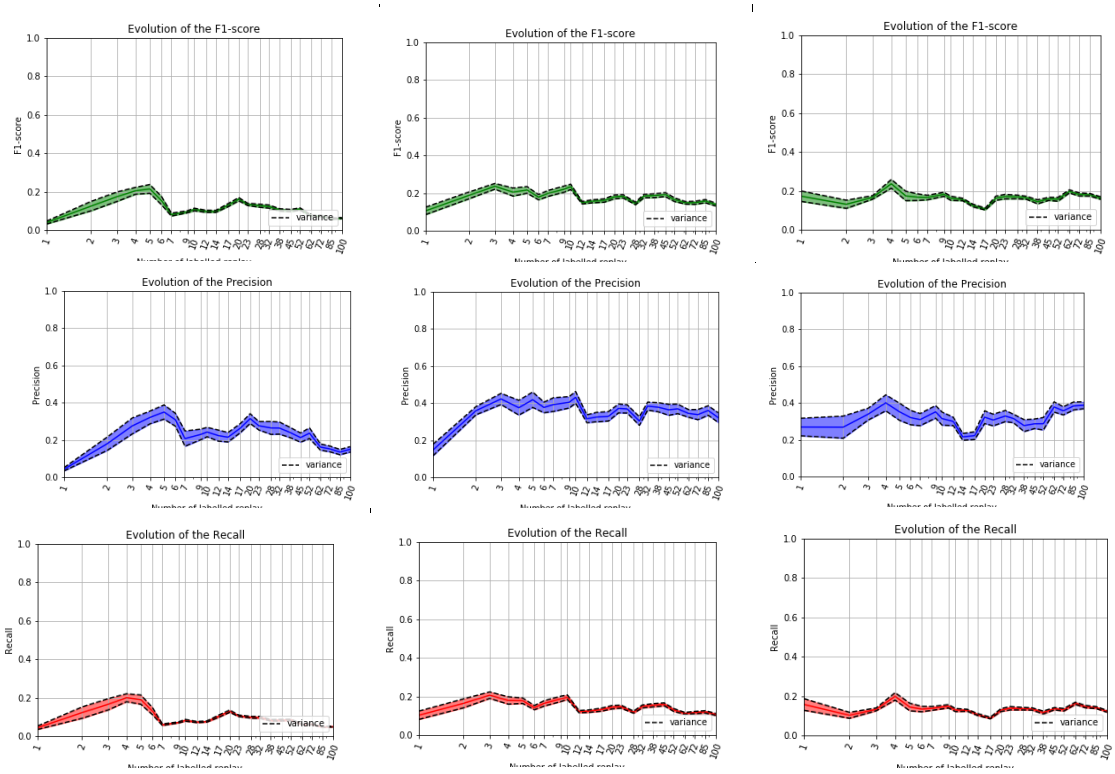


Figure 41: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 0 (left), 50 (middle) and 100 (right) unlabelled replays and a varying number of labelled replays.

6.3.3. Experiment 3 - The Effect of Partially Labelled Replays

This last experiment was made to determine the impact of label sparsity in the HMM's efficiency. To do this experiment, HMMs have been trained on replays with less and less of the available labels. The first batches of models were trained on fully labelled replays and the following batches of models so a constant decrease of 10% in their available labels.

The removal of labels in order to keep x percent of them was done by following this simple method. First, the appropriate labelling rule was used to obtain the correct sequence of labels for each replay. Second, for each replay, compute the number of user labels (labels different from '0' or '1000', '1001', etc.) and calculate how much labels need to be removed in order to keep x percent of the user annotation. Finally, remove the obtained number of user annotation in a random fashion. This last point may not be realistic since a human player that partially annotates replays usually annotates them by portions (Figure 42) instead of by random selected seconds. Nevertheless, this method will allow for an evaluation of HMMs without taking the user's way of annotating as parameter. This aspect could be tested in a whole new experiment where HMMs would be trained with different type of sparse annotations.

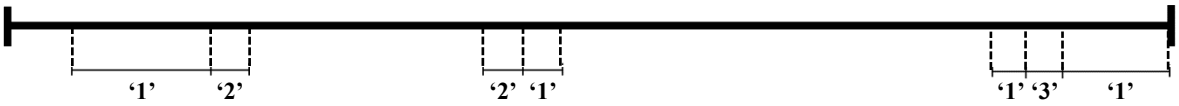


Figure 42: realistic sparse annotation of a replay. The user annotation is constituted of three states ('1', '2' and '3').

For this experiment, HMMs were trained with fixed number of labelled and unlabelled replays.

6.3.3.1. Simple Case

For the simple case, all HMMs were trained on 15 labelled replays but two parameters varied. First, three types of HMMs were trained. One type is without unlabelled replays, another with 50 unlabelled replays and a last type using 150 unlabelled replays. Second, the percentage of labels to remove varied from 0 to 90 percent.

The results of this experiment show that removing labels decreases the HMM's efficiency. But this decrease in efficiency is relatively small. When 90 % of labels are removed, there is only a decrease of 0.5 in the f1-score (from 0.4 to 0.3). This was not expected and means that for a simple case (equivalent to the first labelling rule) fully annotating replays is not really necessary. This also allows for a time gain since users could potentially annotate a small portion of the replay instead of its entirety.

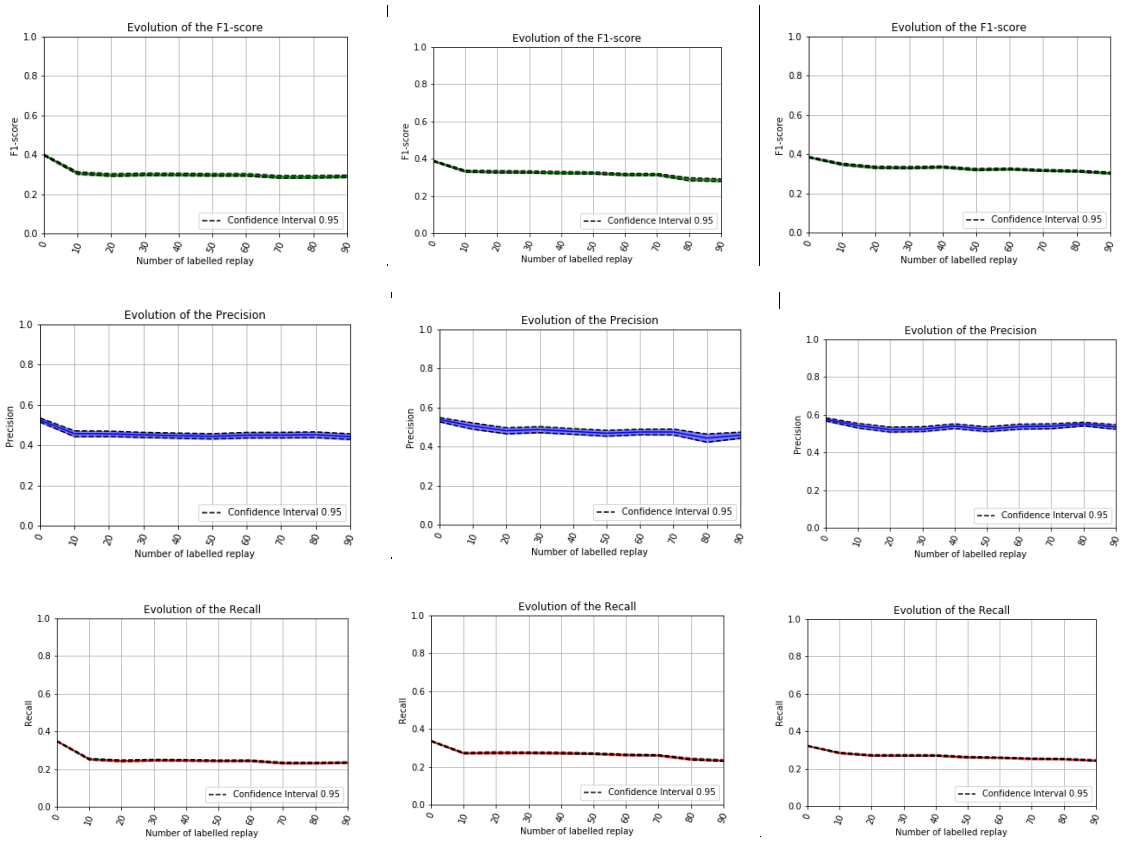


Figure 43: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 15 labelled replay and 0 unlabelled replays (left), 50 unlabelled replays (middle) and 150 unlabelled replays (right) when the percentage of label to remove is varying.

6.3.3.2. Complex Case – One "Unlabelled" State

In a the complex case with one "unlabelled" state used to represent unlabelled seconds of a replay, the different do not seem to be affect by the percentage of labels. This means that for complex case (similar to the complex labelling rule), using sparse annotation do not seem to affect the HMM's efficiency. This could be explained by the fact that in the complex case, labels are already scarce. If labelled portions of a replay already represent less than 5% of the replay, passing from 5 % to 1% will not fundamentally change the HMMs.

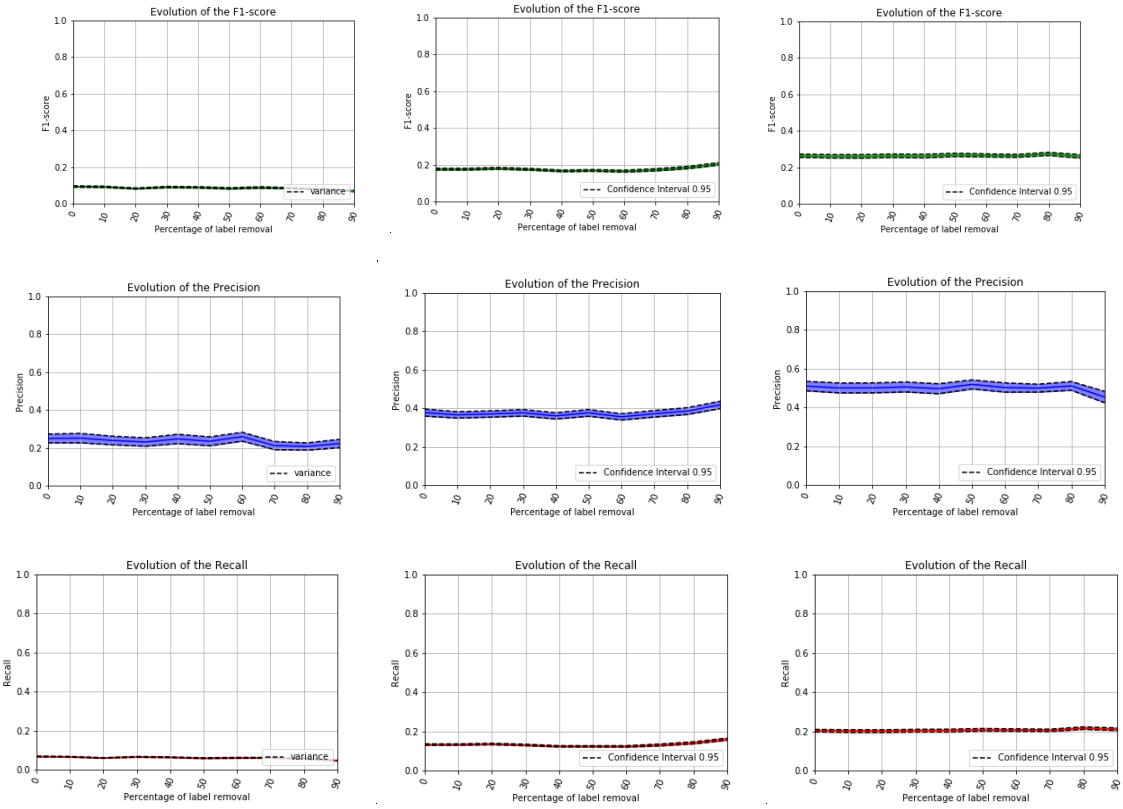


Figure 44: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 15 labelled replay and 0 unlabelled replays (left), 50 unlabelled replays (middle) and 150 unlabelled replays (right) when the percentage of label to remove is varying.

Nevertheless, the use of unlabelled replays increases the starting score of the 3 marks (f1-score, precision and the precision) as seen in the graphs horizontal graphs of the three scores. For the f1-score, the score of the starting HMMs using 15 labelled replays and 0 unlabelled replays while the score of the HMMs using the same number of labelled replays but with more unlabelled replays (passing from 0 to 150) is starting at 0.3.

6.3.3.3. Complex Case – Multiple "Unlabelled" States

The complex case with multiple unlabelled hidden states to represent the unlabelled portions of a replay exhibits the same behaviour as the previous experience. Nevertheless, all scores are slightly lower and this is due to the higher number of hidden states the HMMs have to learn.

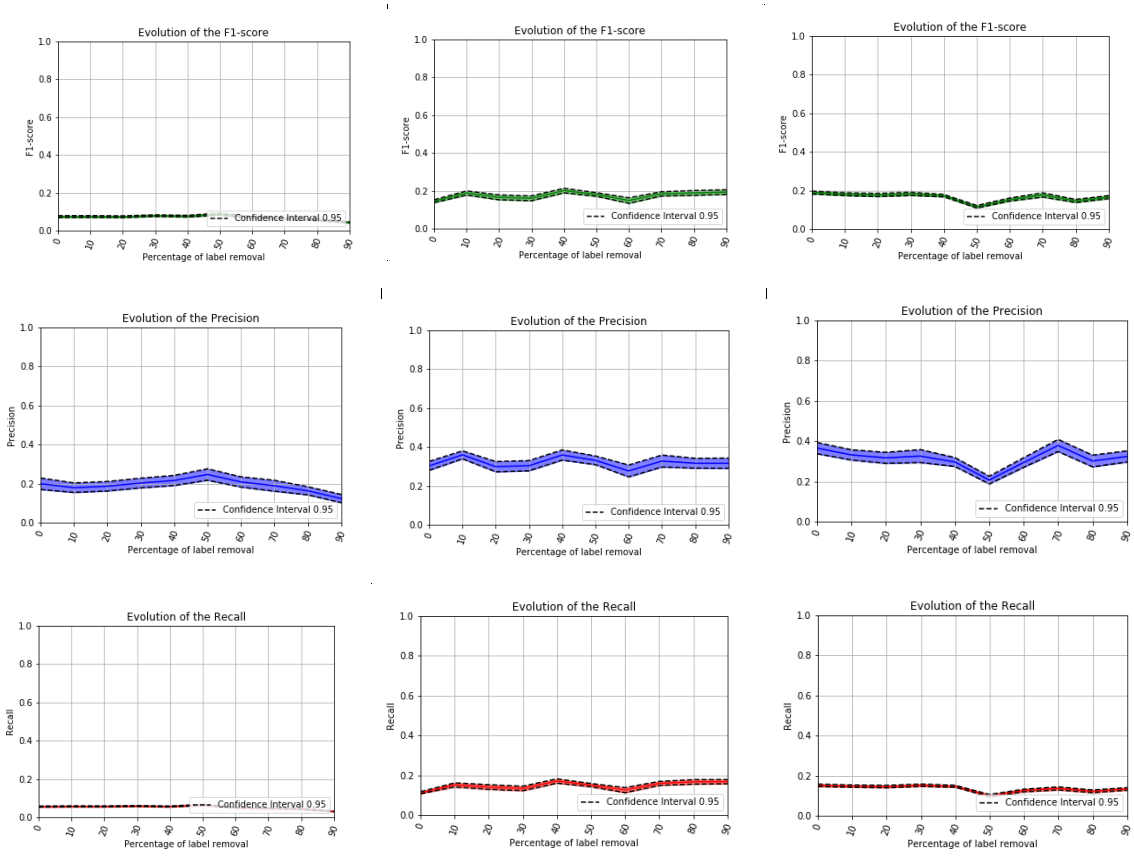


Figure 45: f1-score (top row), precision (middle row), recall (bottom row) evolution of HMMs trained with 15 labelled replay and 0 unlabelled replays (left), 50 unlabelled replays (middle) and 150 unlabelled replays (right) when the percentage of label to remove is varying.

7. Analysis and Discussion

Three experiments were developed to evaluate the efficiency of replay annotation using HMMs. The HMMs were using a set of hidden states to represent the different types of user labels, one or multiple hidden states to represent unlabelled portions of replays and a multivariate time series representation of replay files (section 5.1).

Before the experiments, it was believed that HMMs would easily be able to learn simple annotation cases but the experiments seem to contradict this hypothesis. Graphics of Sections 6.3.1.1, 6.3.2.1 and 6.3.3.1 all show a low recall score which results in a low f1-score value as well (the f1-score is computed from the recall and the precision scores). After analysis, this does not seem to be the result of an error in the code or to be a problem in the learning of the HMMs, since the expected values for the means are respected and correct. At least two hypotheses are available to explain this unexpected result. First, this could be caused by the replays used as training material. Replays vary a lot in terms of length, thus some labels (hidden states) are scarce. For example, the last label of the simple case only happens when the number of workers is greater than 70, which only happens in games lasting more than 5 minutes and where fights against workers are rare. However, labels '1' and '2' often appear in replays of small lengths (Figure 46). The second hypothesis is that the representation of replays as multivariate time series is not appropriate. This was part of our master thesis hypothesis (section 3.1) but can now be considered as a future work to be able to confirm (or not) the usefulness of HMM in the replay annotation case, where simple states have to be learned by the model.

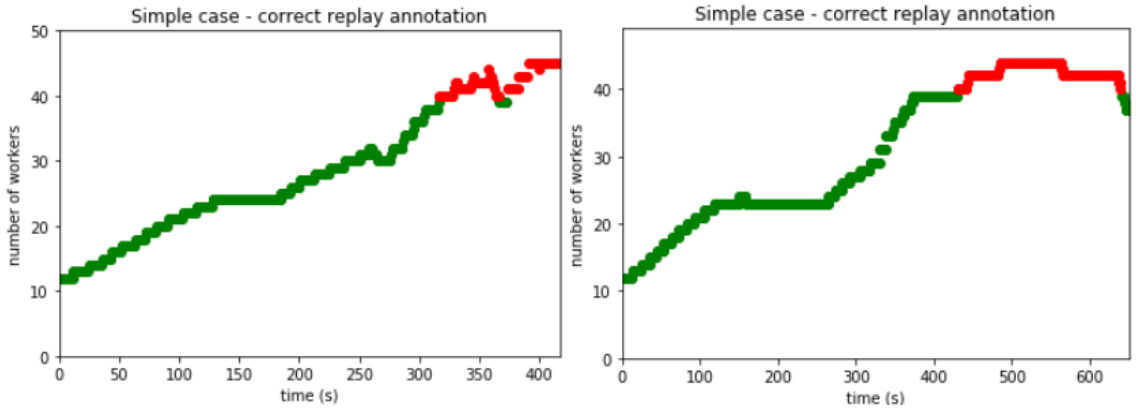


Figure 46: Replays of 400 and 650 seconds where the state '2' is not present (green = state '1', red = 'state 2').

Contrarily to the simple case, the complex case was not expected to be easily learned by HMMs and the results given by the experiment section are mixed results. Annotations given by trained HMMs in Figures 30 and 31 show the potential of HMMs but also show their flaws. For example, the HMMs detect the red and green labels (state '1' and '2') but do not properly learn the fast transitions that are happening in the complex labelling rule. The use of multiple hidden states in order to learn the fast transitions between states do not seem successful since all the metrics (f1-score, precision and recall) are slightly higher when HMMs are using a single unlabelled state (this can be observed by comparing metrics of figures 29 and 32 as well as by comparing metrics of figures 36 and 41 and comparing figure 44 with the figure 45). This could imply that using a single hidden state to represent unlabelled portions of replays is relevant and allows the model to better learn the labelling rule. It is also computationally interesting since this decreases the training time of the HMMs. Nevertheless, the overall efficiency of the HMMs for complex annotation is low with a recall and f1-score around the 0.2 mark.

Another analysis that can be done is that when an HMM needs to learn simple states (similar to the first labelling rule), using unlabelled data does not seem to help the model whatsoever. It is also unexpected for the best f1-score metric to be obtained by the models using a single labelled replay (Figure 26.). An explanation for this behaviour could be that since the testing procedure evaluated the score of 10 different HMMs trained on 1 labelled replay (and different values of unlabelled replays) out of 600 testing replays, the randomly selected 10 replays used for the training were extremely good replays (having all three states appearing). On the other hand, HMMs trained on 50 labelled replays have lots of replays only containing the '1' and '2' states (Figure 46.). This can happen since some replays last only a few minutes and since, in these short replays, the number of workers does not have the time to climb above the 70 mark. This also seems to happen in the complex case but contrarily to the simple case, the complex case shows that using few unlabelled replays can help the learning of the HMMs when labelled replays are few in number as well. This can be explained by the scarcity of labels and the rapid transitions between them, therefore the model does not seem to correctly learn such complex states.

Analysing the second experiment (section 6.3.2) confirms the general belief that using more labelled data usually yields better results. Nevertheless, sections 6.3.2.2 and 6.3.2.3 tell us that after a certain quantity of labelled replays, things start to degenerate and the different scores of the different evaluation metrics decrease. This was not expected but could be explained by the variations in quality of the training replays, some being more representative than others that are generally too short or do not have lots of labels. Nevertheless, the presence of representative replays which are not as good should not be a negative point since, in real life, the percentage of short or not-action-packed replays is similar to the training set.

Finally, analysing the effect of sparse replay annotations tells us that the sparsity of labels does not seem to greatly affect the overall HMMs' learning process. The only affecting parameters are the number of labelled and unlabelled replays used for the training and the complexity of the states.

8. Conclusion and Future Work

In conclusion, to be able to accept HMMs as a good replay labelling tool, more experiments need to be conducted and more types of learning algorithm and state representation have to be tried. To discover the true efficiency of the solution proposed in this work, HMMs should be compared to other types of models applied on the replay labelling task. The comparison between HMMs and other types of models was not part of the scope of this master thesis but could be an interesting subject for a future work.

In the future, trying to solve the replay annotation problem using Gaussian mixtures distributions instead of multivariate Gaussian distributions to represent the hidden states could allow the HMMs to learn complex annotation such as the complex case discussed in the experiment section.

Another promising path would be the introduction of Active Leaning in the HMM learning process. This would allow the user to more easily train a model to fit his/her labels and, with the different Active learning methods present in the literature good results are likely to emerge from this.

Trying to solve the replay annotation problem using another type of HMM, such as hierarchical HMMs, or another completely different model like RNNs could also be an interesting point to start from. Of course, applying such methods on a game other than StarCraft II can yield different results and thus is worth exploring.

9. Bibliography

[1] AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning, published online: 30 October 2019

[2] Samuel, A. L. (1959).Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of Research and Development, 3(3), 210–229

[3] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, et al. 2019. AlphaStar: Mastering the RealTime Strategy Game StarCraft II. https://deepmind.com/blog/ alphastar-mastering-real-time-strategy-game-starcraft-ii/. (2019).

[4] Jonathan Raiman, Susan Zhang, Filip Wolski, 13 Dec 2019. Long-Term Planning and Situational Awareness in OpenAI Five.

[5] L. Rabiner and B. Juang, "An introduction to hidden Markov models," in *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4-16, Jan 1986

[6] CRFs; Lafferty et al., "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". 28 June 2001

[7] Anders Krogh and Soreen Kamaric Riis, "Hidden Neurla Networks", Neural Computation 1999

[8] Graves, A., Jaitly, N., & Mohamed, A. (2013).Hybrid speech recognition with Deep Bidirectional LSTM.2013 IEEE Workshop on Automatic Speech Recognition and Understanding.

[9] Julian Kupiec, Robust part-of-speech tagging using a hidden Markov model, Computer Speech & Language, Volume 6, Issue 3, 1992, Pages 225-242

[10] McCallum, a., & Freitaf, D., & Pereira, F., (2000). Maximum Entropy Markov Models for Information Extraction and Segmentation

[11] Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks

[12] Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright c 2019. All rights reserved. Draft of October 2, 2019.

[13] Forney, G. D. (1973). The viterbi algorithm. Proceedings of the IEEE, 61(3), 268–278.

[14] Stuart Lloyd. Least squares quantization in PCM. In IEEE Transactions Information Theory, 28, pages 129–137, 1982.

[15] Singer, H., & Ostendorf, M. (n.d.). Maximum likelihood successive state splitting. 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings.

[16] Li, Cen & Biswas, Gautam. (1999). Clustering sequence data using hidden Markov model representation. Proceedings of SPIE - The International Society for Optical Engineering. 3695. 14-21.

[17] Text Classification from Labeled and Unlabeled Documents using EM, Kaml N., Andrew K.MC., Sebastian T., Tom M., 1999

[18] Inoue, Masashi & Ueda, Naonori. (2004). Exploitation of Unlabeled Sequences in Hidden Markov Models. Pattern Analysis and Machine Intelligence

[19] Semi-supervised Sequence Classification with HMMs, Shi Zhong, 2004

[20] Learning from Labeled and Unlabelled Data with Label Propagation, Xiaojin Z., Zoubin G., 2002

[21] Chen, X., & Wang, T. (2017). Combining Active Learning and Semi-Supervised Learning by Using Selective Label Spreading. 2017 IEEE International Conference on Data Mining Workshops (ICDMW)

[22] E. Riloff, J. Wiebe and T. WSLSon, “Learning subjective nouns using extraction pattern bootstrapping,” Proceedings of the Seventh Conference on Natural Language Learning, 2003.

[23] C. Rosenberg, M. Hebert and H. Schneiderman, “Semi-supervised selftraining of object detection models,” Seventh IEEE Workshop on Applications of Computer Vision, 2005.

[24] Brian G. Leroux, Maximum-likelihood estimation for hidden Markov models, Stochastic Processes and their Applications, Volume 40, Issue 1, 1992, Pages 127-143

[25] G. Schwarz, \Estimating the dimension of a model", The Annals of Statistics, vol. 6, pp 461-464, 1978

[26] G. Clasekens, “Statistical model choice”, KU Leuven, Faculty of economics and business, page 3

[27] Moon, T. K. (1996). The expectation-maximization algorithm. IEEE Signal Processing Magazine, 13(6), page 47–60

[28] J.F. Kaiser, W.A. Reed, Data smoothing using lowpass digital filters. Rev. Sci. lnstrum. 48, 1447–1457 (1977)

[29] N. Jardine & C.J van Rijsbergen - Information Storage and Retrieval, Volume 7, Issue 5, December 1971