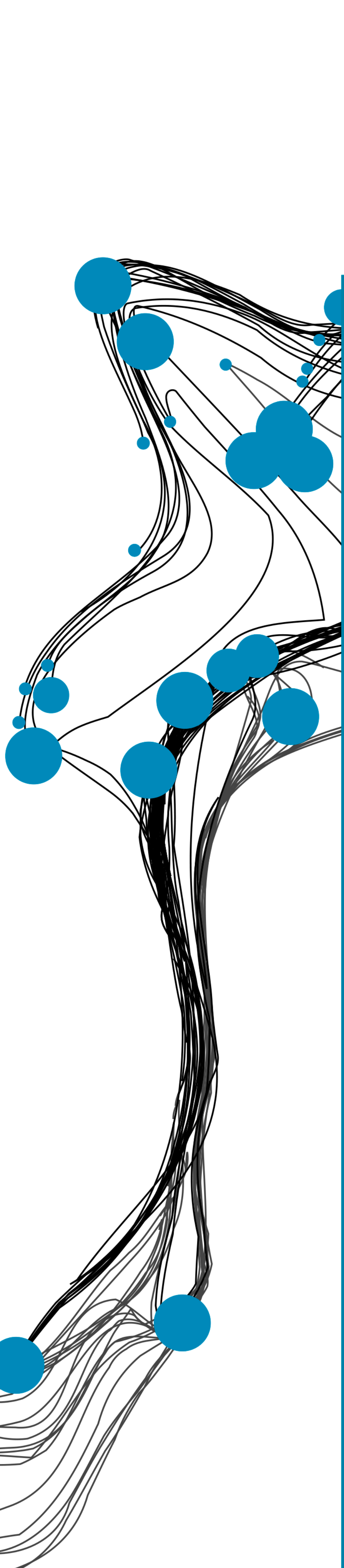


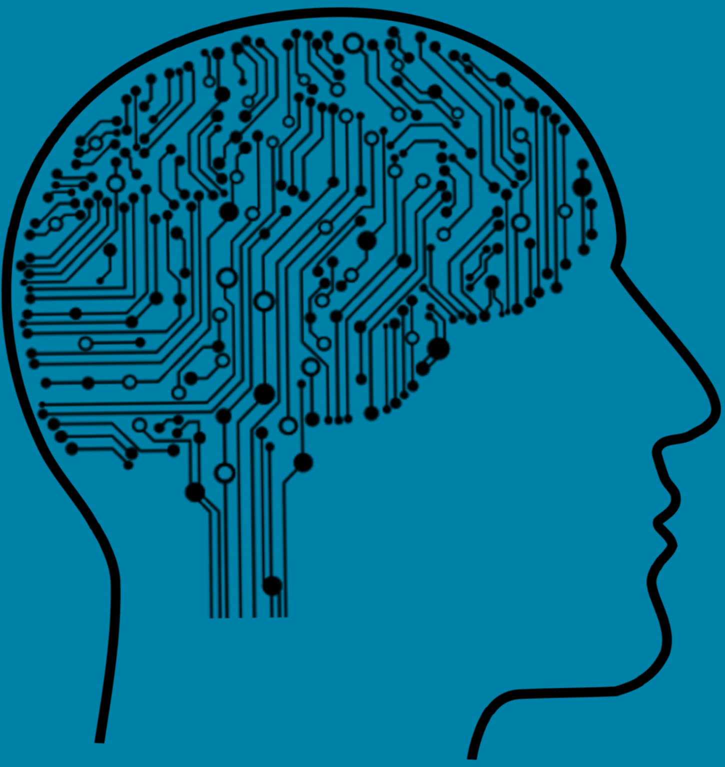
Moldel

Predicting the ’Mol’ in ’Wie is de Mol?’ using machine learning and mathematical modelling.

Author:	H.M.R. Dorenbos
Supervisor & Examiner:	prof.dr. A.J. Schmidt-Hieber
Senior-Examiner:	dr. M. Poel
Co-Supervisor:	dr. A.F.F. Derumigny

University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science

Monday 12th April, 2021



ABSTRACT

'Wie is de Mol?' is a Dutch game show annually broadcast by the AVROTROS on NPO1 since 1999 with players having to fulfil exercises in order to earn money. A *mol* among them unknown to the spectators and other players however has to prevent them from earning money. The goal for these players and spectators is to discover the *mol*. Although no spectator has shown to be able to consistently find the real *mol*. This thesis presents the Moldel, an algorithm able to predict the *mol*. To do so the Moldel aggregates the predictions of four separate models using stacking. These models are named the Exam Drop layer, the Exam Pass layer, the Wikipedia layer and the Appearance layer. All of them uses either mathematical modelling or machine learning to predict the *mol*. Different models are discussed as possible implementations for these layers, which includes Bayesian Models, Probabilistic Programs, Logistic Regressions, Cosine Similarity, Nearest Neighbor, Gaussian Naïve Bayes and Kernel Density Estimation. For each of these implementations their strengths and weaknesses are compared with respect to themes such as overfitting, generalization and complexity. Moreover the power of some feature preparation techniques used is shown, such as Feature Discretization, Feature Clustering, Principal Component Analysis, Natural Language Processing, Logarithmic Transformation and Linear Discriminant Analysis. Besides statistical tests are used to backup the results of the Moldel, which includes the Mann-Whitney U Test, Student Paired T-Test, Wilcoxon Signed Rank Test, Pearson Correlation Test and Kendall Correlation Test. Last of all, the performance of the Moldel is evaluated using metrices such as (Mol) Log Loss, Concordant-Discordant Ratio, Mean Mol Likelihood and Mean Mol Rank which reveals that the Moldel performs better than uniform guessing in the past. Nevertheless this breakthrough is not only beneficial for the 'Wie is de Mol?'-community, but also for the Data Science community in general. Dealing with shortage of data is a returning issue for many projects and the Moldel provides solutions how to deal with shortage of data, which are data augmentation, using multiple models combined with stacking and feature reduction techniques. Moreover the Moldel also illustrates how to evaluate purely probabilistic predictions rather than deterministic predictions. This is useful as well, because most well-known evaluation methods are only meant for deterministic predictions, e.g. confusion matrices, recall, precision and accuracy. Furthermore two less commonly used models are introduced, i.e. the Split Classifier and a Kernel Density Estimation classifier. Both of these are models for 1-dimensional feature classification. The Split Classifier is a model that does not require a lot of data and can provide decent and stable predictions. On the other hand the Kernel Density Estimation classifier is a very general model able to understand almost any (probabilistic) pattern as long as enough data is provided. Secondly two feature processing techniques are emphasized, i.e. feature encoding using clustering and a forward information gain selection procedure to determine the number of bins per feature when using discretization. Feature encoding using clustering is a powerful type of encoding which does not depend on the label, thus having a low risk of overfitting. Similarly the forward information gain selection procedure to determine the number of bins does neither depend on the labels and is also a useful method to determine how many bins should be assigned to each feature. Last of all, a natural language processing technique is introduced named Subword Extraction. Subword Extraction is a method to recognize similarities between different words by using a dictionary. This method has a low false negative rate, but unfortunately has a high false positive rate.

CONTENTS

Abstract **1**

Glossary **4**

1 Introduction **6**
- 1.1 Popularity . . . 6
- 1.2 Related Work . . . 7
- 1.3 Game Mechanics . . . 8
- 1.4 Goal . . . 9

2 Exam Layer **12**
- 2.1 Introduction . . . 12
- 2.2 Previous Approach . . . 14
- 2.3 Current Approach - Drop Layer . . . 18
- 2.4 Current Approach - Pass Layer . . . 27

3 Wikipedia Layer **30**
- 3.1 Introduction . . . 30
- 3.2 Previous Approach . . . 31
- 3.3 Current Approach . . . 33

4 Appearance Layer **41**
- 4.1 Introduction . . . 41
- 4.2 Previous Approaches . . . 43
- 4.3 Current Approach . . . 47

5 Aggregation **53**
- 5.1 Introduction . . . 53
- 5.2 Approaches . . . 54
- 5.3 Current Approach . . . 57
- 5.4 Social Media Exclusions . . . 62

6 Results **64**
- 6.1 Approach . . . 64
- 6.2 Evaluation . . . 67

7 Discussion **76**
- 7.1 Significance Testing . . . 77
- 7.2 Reflection . . . 80

8 Conclusion **82**

Acknowledgements **83**

References **86**

A Exception Handling **87**
A.1 Exam Drop Layer 87
A.2 Wikipedia Layer 88
A.3 Aggregation 89

B Additional Bins for Exam Drop Layer **90**

C Wikipedia Layer - Old Results **91**

D Dependency of Appearance Values **92**

E Raw Results **93**

F Final Prediction of Season 22 **96**

Glossary

21 The additional season of 'Wie is de Mol?' broadcast in the period from 5 September 2020 until 24 Oktober 2020 is defined to be the 21th season of 'Wie is de Mol?'. This season is also named the Renaissance season.

alive A player is defined as *alive* in a given episode if that player did not drop out before that episode and did not return back during that episode.

executie The *executie* is a phase in the game that happens after the *test*. The player excluding the *mol* that has the least number of correct answers on the *test* will drop out during this phase. In case of a tie the player that took the longest time for the *test* will drop out.

executies Plural form of *executie*.

joker A *joker* is an item that can be used during the *test*. A player using a *joker* will have a wrong answer marked as correct per *joker* used in the *test*.

jokers Plural form of *joker*.

mol One of the players has secretly been assigned the role *mol* at the start of the game. The goal of the *mol* is to reduce the money that the players earn without being noticed.

normalization Normalization is the process in which a vector of probabilities is *normalized*.

normalize Normalizing a group of likelihoods $\mathcal{P} = (\rho_1, \rho_2, \ldots, \rho_n)$ means that every likelihood ρ_i in $\mathcal{P}$ is divided by the sum of all likelihoods, i.e.

$$\rho_i^{\text{new}} \leftarrow \frac{\rho_i^{\text{old}}}{\sum_{i=1}^{n} \rho_i^{\text{old}}}$$

which ensures that all probabilities sum up to 1, i.e. $\sum_{i=1}^{n} \rho_i^{\text{new}} = 1$.

normalized See the definition of *normalize*.

penningmeester The *penningmeester* is the player that is responsible for keeping all earned money.

potential mol A *potential mol* is someone that could theoretically be the *mol*. Every player that has not dropped out yet, has not yet seen a red screen or has not yet been revealed during the game show not being the *mol* is a *potential mol*. Note that *potential mol* players are different from *alive* players. A player could have seen a red screen without dropping out, could have dropped out and returned back to the game or could have been revealed by the cast to not be the *mol*. In those circumstances a player is *alive*, but no *potential mol* anymore.

test During the *test* players have to answer questions about the behavior and identity of the *mol*.

tests Plural form of *test*.

voluntarily See the definition of *voluntary*.

voluntary A dropout is defined as *voluntary* if the reason of that dropout was not related to having the worst *test* score. A non-*voluntary* dropout is therefore a dropout which is related to having the worst *test* score. An example of a *voluntary* dropout is the dropout of Manuel in season 10, because he felt sick.

vrijstelling A *vrijstelling* is an item that can be used during the *test*. A player using a *vrijstelling* will not take part in the *executie* and therefore will automatically go on to the next episode. Among the players not using a *vrijstelling* the dropout is selected.

vrijstellingen Plural form of *vrijstelling*.

zwarte vrijstelling The *zwarte vrijstelling* is an item that can be used during the *test*. When used it will cancel out the effects of all *jokers* and *vrijstellingen* used during the *test*. This item has been introduced since season 14.

zwarte vrijstellingen Plural form of *zwarte vrijstelling*.

1 INTRODUCTION

1.1 Popularity

'Wie is de Mol?' is a television show in the Netherlands produced by IDTV that has been broadcast (almost) annually since 1999 by the AVROTROS on NPO1 [1]. In 1999, when 'Wie is de Mol' started, it was still a quite unknown game show, but over the years the number of spectators increased rapidly. In 2013 'Wie is de Mol' received an award named the 'Gouden Televizier-Ring' for being the best Dutch television programme [2]. And currently with a record of around four million spectators [3], 'Wie is de Mol?' is considered to be one of the most popular game shows in the Netherlands. The game show 'Wie is de Mol?' is about discovering the identity of the *mol*, which is one of the participants that secretly tries to sabotage the assignments which the players have to fulfil in order to earn money. The player that discovers the identity of the *mol* wins all this earned money at the end of the game show.

The interesting aspect of the game show is that the audience can also play along from home, because the cast of 'Wie is de Mol?' hides the identity of the *mol* for the audience as well. Hence spectators can also look for clues to discover the *mol*. This has become a large business in itself by:

- Youutube channels that discuss their suspicions and clues about the *mol*, e.g. 'de therMOLmeter', 'Gido Verheijen - Wie is de Mol?', 'Felix - Wie is de Mol 2020', 'Siem', 'WIDM TV' and 'Wouter'.[1]
- 'De Wie is de Mol? podcast' which is a radio programme that looks back to the last episode and sometimes has an interview with the latest dropout of 'Wie is de Mol?'.[2]
- The television programme 'MolTalk' which is broadcast after every episode of 'Wie is de Mol?'. In this television programme all suspicions and clues of the last episode are discussed.
- Forums related to 'Wie is de Mol' where users can discuss their suspicions and clues, e.g. 'de Mol Fansite forum' and the 'Reality Net - Wie is de Mol? forum'.[3]

Furthermore a lot of news channels and television/radio programmes cover suspicions during the period that 'Wie is de Mol?' is broadcast. Although there are lots of channels discussing suspicions about the *mol*, none of them has shown to be able to consistently find the real *mol*. A reason for this is that none of these channels has an objective or systematic approach to

[1]These Youtube channels can be accessed by: `https://www.youtube.com/channel/UC-uL4PfhVutX62ucvmABB6A` (de therMOLmeter), `https://www.youtube.com/channel/UCp7JFGkv9oIkMhyCg7byWZg` (Gido Verheijen - Wie is de Mol?), `https://www.youtube.com/channel/UCogf9qjOOzVT89RKdRSJbtQ` (Felix - Wie is de Mol 2020), `https://www.youtube.com/channel/UC8pE9riHyfmzqg9Mfj9mdkg` (Siem), `https://www.youtube.com/channel/UCaHf2Hyygl5KxrYooH5Xx9w` (WIDM TV), `https://www.youtube.com/channel/UCbX2TUQGsV4F2jAcxWsQJ5w` (Wouter)

[2]This podcast can be accessed by: `https://www.nporadio2.nl/podcasts/de-wie-is-de-mol-podcast`

[3]These fora can be accessed by: `https://www.widm.nl/forum.html` and `https://www.wieisdemol.com/forum`

discover the *mol*, despite that they discuss verifiable facts. However none of them has seriously investigated the validity of their clues. Moreover it is impossible for these channels to prove that all clues are covered by their channel. On the contrary, it usually happens that a channel has tunnel vision, i.e. the channel takes it for granted that a player is the *mol* and only searches for evidence that supports it. With all of these issues, it is hard to reflect on what went wrong with your predictions and to improve your predictions for the next season. This motivates the use of machine learning and mathematical modelling to determine the *mol* in an objective[4] and systematic manner.

1.2 Related Work

Using machine learning, mathematical modelling or big data techniques to do predictions for game shows is not something entirely new. For example the game show ”Let’s Make a Deal” is one of the most famous examples, also known as the ”Monty Hall Problem”, where mathematicians were able to predict the outcome using a simple mathematical model [4]. At the end of this show the host, Monty Hall, lets the participant choose between three closed doors. Behind one of these doors is a car and behind the others is a goat. After the participant selected a door, Monty Hall opens another door with a goat behind it and asks the participant whether he wants to switch to the other closed door. Contrary to popular belief, there is a probability of 2/3 of the car behind the other closed door [4]. Hence this is an old game show in which predictions by mathematical modelling has shown to be effective. But also for more modern television shows, like Survivor, it is possible to do predictions [5]. This research tried multiple machine learning techniques, i.e. Logistic Regression and Naïve Bayes Classifier to predict the winner. The features used by these models were events that happened during the game or characteristics about the player itself, which were combined with proper feature extraction techniques such as Linear Discriminant Analysis and Principal Component Analysis [5]. Hence machine learning has also recently been used to make predictions for game shows. Last of all based on social media analysis, which is a big data technique, scientists have been able to predict the winner of American Idols [6]. In American Idols the audience decide the winner by voting, thus a model was used that counts the number of tags and mentions on Twitter related to any of the contestants to estimate the popularity of contestants in different regions [6].

So these models apply either machine learning, mathematical modelling or big data techniques to do predictions for a game show. Though all of these game shows are American game shows, none of them are ’Dutch’ game shows. And to the best of our knowledge no scientific article has yet been written regarding predictions for a typical ’Dutch’ game show. This thesis is therefore the first scientific article about making predictions for a typical ’Dutch’ gameshow, which in this case is the gameshow ’Wie is de Mol?’. However note that having no scientific articles about ’Wie is de Mol?’ does not imply that it has never been investigated or analysed. There have been serious attempts to discover the *mol* in an objective and/or systematic manner, which includes:

- The social media analysis of Jaap van Zessen, which checks the online and social media activity of players during the recording period of ’Wie is de Mol?’ [7]. Van Zessen argues that players with a high activity on social media during the recording period drop out early and therefore could not be the *mol*. The results of his analysis are quite accurate and therefore these results are used in adjusting the final predictions of the Moldel as discussed in Section 5.4.
- The face recognition analysis of Mattijn van Hoek on ’Wie is de Mol’ [8]. For this project a face recognition library of Adam Geitgey was used [9] to detect the appearance of players

[4]Though the building of the Moldel is subjective, because of the subjective design choices made.

during episodes. During the first four episodes of season 18 up to 20 the *mol* appeared less than other players according to his analysis. For other seasons no analysis has been done so far. Also Van Hoek did not build a machine learning model on top of his analysis. Therefore for this project permission is requested and received by Van Hoek to continue investigations in the appearance of players. In Chapter 4 this investigation is discussed, which includes the questions whether the *mol* indeed appears less and how this can be turned into a prediction model.
– The 'Gelijkekansentheorie' (in English: equal probability hypothesis), analyzing assignments where players are split up in different groups [10]. According to this hypothesis the *mol* is evenly distributed over the groups, i.e. every player ends up the same number of times with the *mol* in his/her group. Unfortunately this hypothesis was not valid for newer seasons [10], so this hypothesis is not investigated any further in this project.
– The WIDM-hints algorithm that analyses clues posted on social media and clues received from visitors of their website.[5] Visitors of their website can vote for these clues and based on these votings the algorithm is able to predict the *mol*. Unfortunately the algorithm is closed source. Moreover circular reasoning could be a major problem here, e.g. if your algorithm suspects a player to be the *mol* then the visitors tend to vote for the clues related to that player.

1.3 Game Mechanics

'Wie is de Mol' is usually recorded in the months May & June [11] and is broadcast weekly in the months January, February & March, where every episode takes about one hour. The television show starts with ten players including one *mol*.[6] They have to fulfil assignments in order to earn any money. The *mol* on the other hand, whose identity is unknown to the other players, tries to secretly sabotage these assignments and to reduce the earned money. Every episode normally consists of three assignments, which can vary from a laser quest where players have to follow a trail without getting shot to assignments where players have to transfer a message to one another. Nevertheless a central aspect of these assignments is to form groups, which all have to function properly in order to fulfil the assignment. At the end of the assignments the players earn (part of) the money (or even lose money) based on how they have performed. Also players can receive certain items, e.g. *jokers*, *vrijstellingen* and *zwarte vrijstellingen*, during the assignments that can help them to pass the *executie*. These items can be kept by the player that received them, however the money is kept by the *penningmeester*. The *penningmeester*, one of the players (approved by the majority of all players), is responsible for keeping the money.

At the end of an episode there is a *test*. For the *test*, players normally have to answer 20 multiple choice questions about the *mol* which are the same for every player. An example of a question is to which group the *mol* belonged during a given assignment. Furthermore the players are allowed to use their items, e.g. *jokers*, *vrijstellingen* and *zwarte vrijstellingen*, during the *test*. But once an item is used, a player cannot re-use it again. In addition there are rules about the usage of items sometimes, e.g. that a *vrijstelling* should be used directly in the episode in which it was earned. But if there are no rules then the player is allowed to use their items during any *test*, except for the final *test*. When all players have filled in the *test*, an *executie* happens. During the *executie* the player with the least correct answers excluding the *mol* drops out.[7] This happens by showing the players a screen. If that screen turns green then the player is safe (or the *mol*). But if the screen turns red then the player drops out. This is usually how the *test* and

[5]This website can be accessed by: `http://widm-hints.nl/`
[6]Except for season 3 which started with 11 players.
[7]In case of a tie the player among them that took the most time drops out.

executie works. However there are some special events that can happen during the *executie*, for example:

- Only part of the players see their screen. If there is no red screen among them then all players pass on.
- They have used a group *vrijstelling*, which means that all players pass on to the next episode.
- Two players with the least number of correct answers drop out instead of one player.

But normally only a single player drops out during the *executie*, after which the episode ends. If the *penningmeester* was the dropout then his/her role and money is transferred to one of the remaining players. Moreover sometimes the dropout is allowed to give his/her remaining items (*jokers*, *vrijstellingen* and *zwarte vrijstelling*) to the players that pass on to the next episode.

The next episode follows the same procedure, which is three assignments followed by a *test* and *executie*, but with one player (and *potential mol*) less.[8] This continues until the finals in which usually 3 players (including the *mol*) are left.[9] These players do the final *test*, which consists of 40 questions. The player, except for the *mol*, that has the most correct answers wins all earned money in the showdown episode, which is the episode right after the finals. In the showdown episode the winner and the *mol* are revealed. Likewise in the showdown episode flashbacks happen to the sabotage actions of the *mol* and all clues are revealed that referred to the *mol*.

1.4 Goal

With these sabotage actions and clues revealed it often seems to be obvious to predict the *mol*. However these clues and sabotage actions are not revealed before the showdown episode. Therefore it is a challenge to predict the *mol* before the showdown episode. The algorithm discussed in this thesis, which is named the Moldel, faces this challenge. To predict the *mol* before the showdown episode, the Moldel uses multiple separate layers that all analyse a particular aspect of the game with machine learning and/or mathematical modelling. These layers are:

- The Exam Drop layer, which analyses answers given on the *test* by dropouts and whether players referred to by these answers are the *mol*.
- The Exam Pass Layer, which analyses the relationship between *joker* and *vrijstelling* usage and being the *mol*.
- The Wikipedia Layer, which analyses the influence of famousness and job on being the *mol*.
- The Appearance Layer, which analyses the relationship between appearance during episodes and being the *mol*.

The output of all these layers results in a likelihood distribution, i.e. $\{(p_1, \rho_1), \ldots, (p_n, \rho_n)\}$, where p_i is the player and ρ_i is the likelihood of being the *mol*. Likelihoods are equal to probabilities in the sense that a player p_i with likelihood ρ_i is expected to be relatively ρ_i times the *mol* in similar scenarios[10] and that:

- The likelihood ρ_i of an arbitrary player p_i is a value between 0 and 1, i.e. $0 \leq \rho_i \leq 1$.

[8] Except for some cases where the dropout is allowed to return back to the game.

[9] Except for season 7 & 20 which both had 4 players in the finals.

[10] Which are scenarios whose modeling representations by the Moldel are equal.

– All likelihoods of the players sum up to 1, i.e. $\sum_{i=1}^{n} \rho_i = 1$.

The next step of the Moldel is to aggregate these likelihood distributions into a single likelihood distribution, followed by excluding all non-*potential mol* players as *mol* and excluding players as *mol* based on Social Media data. A general overview of the Moldel is shown in Figure 1.1.

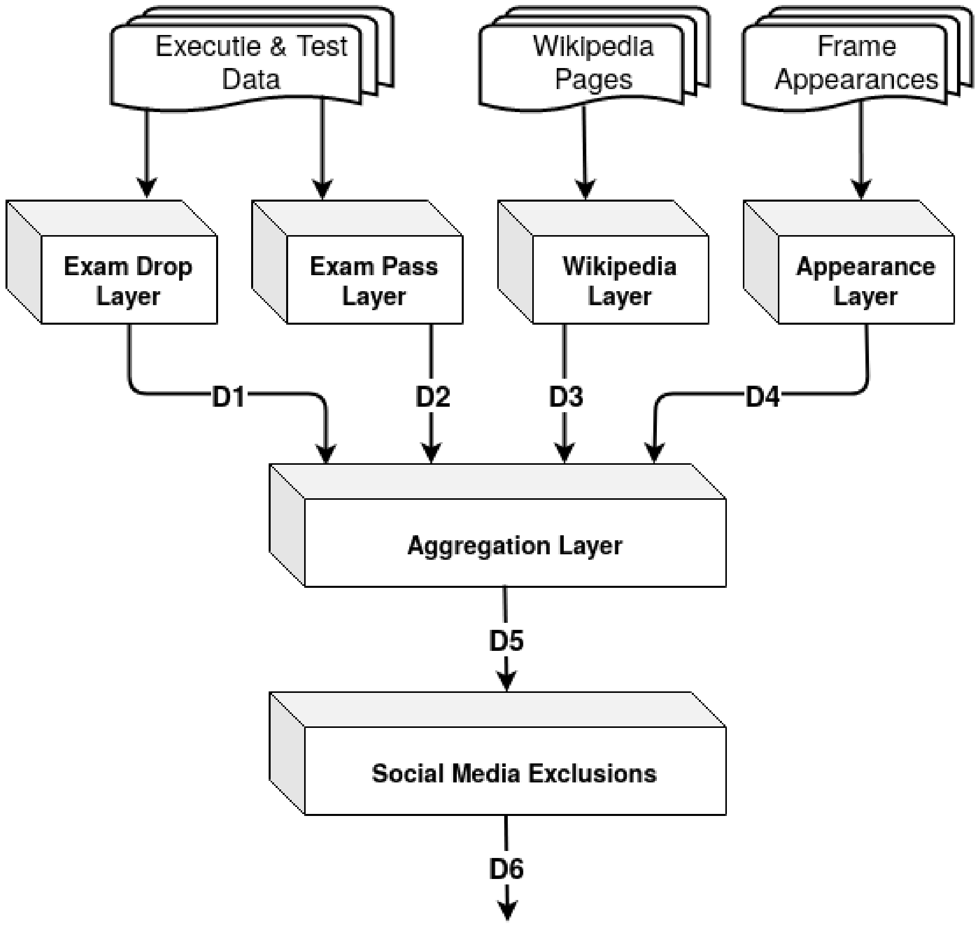


Figure 1.1: Overview of the Moldel

In this overview each D_i represents an output likelihood distribution, where D_6 is the final prediction of the Moldel. The goal with respect to this final prediction D_6 is to:

Goal Statement. *Predict the actual* mol *of the game show 'Wie is de Mol?' with a likelihood higher than 0.5 right after the finals in the season that is broadcast in 2021. Moreover the final predictions of seasons 9 up to 21 should be significantly better than random guessing, i.e. the hypothesis that these predictions are as good as uniform random guessing or worse should be rejected with a p-value smaller than or equal to 0.05.*

In this goal statement uniform random guessing is defined as the uniform distribution $\{(p_1, \rho_1), \ldots, (p_n, \rho_n)\}$ with:

$$\rho_i = \begin{cases} \dfrac{1}{|P_+|} & \text{if } p_i \in P_+ \\ 0 & \text{if } p_i \notin P_+ \end{cases} \tag{1.1}$$

where P_+ is the set of all *potential mol* players and $|P_+|$ is the size of this set. In the following chapters it is described how the Moldel has evolved over time, how the Moldel is currently implemented and why certain implementation decisions are/were made. Finally the Moldel is evaluated and statistically tested upon which we reflect. More specifically:

- In Chapter 2 the previous implementation and current implementation of the Exam Drop Layer and the Exam Pass Layer are discussed.
- In Chapter 3 the previous implementation and current implementation of the Wikipedia Layer are discussed.
- In Chapter 4 the previous implementation and current implementation of the Appearance Layer are discussed.
- In Chapter 5 the different type of approaches to aggregate these layers are described and compared. Finally it is discussed which of these approaches is selected and how it is implemented.
- In Chapter 6 it is discussed how the predictions of the Moldel are evaluated. Likewise some of the predictions of the Moldel are shown in this chapter.
- In Chapter 7 the Moldel is statistically tested. Furthermore the results, these tests and other aspects of the Moldel are reflected upon.

For more information and details about the current state of the Moldel, you can access the project by the following URL:

`https://github.com/Multifacio/Moldel`

2 EXAM LAYER

2.1 Introduction

The research on predicting the *mol* started with the Exam Layer dating back to 16 February 2019. Which was 3 days after episode 7 of the 19th season of 'Wie is de Mol?'. During this episode there were 2 dropouts, who were my first and second most suspected *mol* Rick-Paul and Jamie.[1] The outcome was unexpected, not only because both my suspected *mol* players dropped out, but also because they both used *jokers*. In addition 3 out of the 6 players did not use any *jokers* at all, which made the outcome even more unexpected since usually a player not using any *jokers* drops out. So this episode was closely analysed at that time on what the players answered during the *test* and how many *jokers* they used, which revealed the following:

Player	Answered On[2]	Used Jokers
Jamie	Rick-Paul	2
Merel	Jamie	0
Niels	Jamie	2
Rick-Paul	Jamie	1
Sarah	Merel	0
Sinan	Rick-Paul, Jamie, Merel	0

Table 2.1: Analysis of the *test* in season 19, episode 7

Based on this analysis the reasoning started which of the players left could be the *mol* (Merel, Niels, Sarah or Sinan). For each of these players it was argued whether the test outcome could be explained if that player was the *mol* and these were the arguments:

Niels If Niels is the *mol* then all answers were wrong. This means that the players using *jokers* were more likely to pass the *executie*. Which did not happen, since Jamie and Rick-Paul both dropped out whereas Merel, Sarah and Sinan passed the *executie*. Thus this scenario is very unlikely.

Sarah In case Sarah is the *mol* then the same reasoning used for Niels can be applied here. However it explains why Sarah passed the *executie*, since she is the *mol*. But it remains an unlikely scenario, because Merel or Sinan, who both used less *jokers* than Jamie and Rick-Paul, did not drop out.

Sinan Sinan's scenario of being the *mol* is similar to Sarah's scenario of being the *mol*. It explains why Sinan passed the *executie*, but it still does not explain why Merel or Sarah did not drop out. Hence this scenario is unlikely.

Merel For Merel the situation is totally different. If one assumes that Merel is the *mol* then it is more reasonable why Sarah and Sinan passed the *executie*, because they both had

[1]This episode is available at: `https://www.npostart.nl/wie-is-de-mol-aflevering-7-2019/16-02-2019/AT_2111648`

[2]These players were covered by the answer revealed of that player.

a correct answer. And it explains why Merel did not drop out, since she always had her answer wrong regardless whether Niels, Sarah, Sinan or Merel was the *mol*. Therefore this scenario is the most likely scenario.

Thus according to the reasoning, Merel was expected to be the *mol*, which was also confirmed during the showdown episode.

So was this just a lucky guess, or a valid approach to determine the *mol*? One could argue that in almost all *tests* at most one single answer is revealed per player. And each *test*, except for the final *test*, consists of 20 multiple choice questions. Hence there are 19 questions left for every player which can turn around the entire outcome. However this is highly unlikely to happen if it is assumed that the dropout performs similar as other players on the remaining questions.[2] Hence it explains why in most cases the shown answer of the dropout is incorrect. If the shown answer was correct then the dropout was at least one or two correct answers ahead on all other players.[3] But since he/she dropped out, all other players would have caught up with this advantage which is very unlikely.[2] Thus the scenario that the shown answer is incorrect of the dropout is most plausible, especially during earlier episodes. With these insights the first "Wie is de Mol" prediction layer was created. A layer that makes predictions by looking at what the dropouts answered, how many *jokers* & *vrijstellingen* players used and what the players that passed on answered. And by looking at these answers on the multiple choice questions a shared structure can be found, e.g. some of these multiple choice questions are:

1. Is the *mol* a male or a female? Answers: Male, Female.
2. Is the *mol* the current *penningmeester*? Answers: Yes, No.
3. In which room did the *mol* slept last night? Answers: 105D, 203A, 255G, 307F
4. How many *jokers* did the *mol* collect during the last exercise? Answers: 0, 1, 3.
5. Who is the *mol*? Answers: Jamie, Merel, Niels, Rick-Paul, Sarah, Sinan.

Each of these questions is a set of answers $Q' = \{A_1, A_2, \ldots, A_n\}$, where each answer A_i is a set of players. More concretely, each question is a partitioning over the set of players *alive*.

Definition 1. *$Q' = \{A_1, A_2, \ldots, A_n\}$ is a partition of set S if and only if:*

$$S = \bigcup_{i=1}^{n} A_i \qquad \textit{and} \qquad \forall_{i \neq j}\ A_i \cap A_j = \emptyset$$

Which means that every player *alive* is included in exactly one answer. For example a player slept either in room 105D, 203A, 255G or 307F and a player either collected 0, 1 or 3 jokers during the last exercise. This also implies that exactly one answer is correct and that the other answers are wrong. Based on this structure and the findings discussed in this section, a first prediction model was created. This model uses a Bayesian approach. Computing the probability that a question is answered correctly given that someone drops out is generally hard to do. However computing the probability that someone drops out given that an answer is correct is much easier. Bayes theorem tells how to express this former probability using the latter, see Section 2.2. After this more accurate and stable prediction models were created, known as the current Exam Drop Layer and the current Exam Pass Layer. Both implementations uses machine learning techniques to determine the likelihood that someone is the *mol*. These models focuses more on feature gathering, selection and extraction (see Sections 2.3 and 2.4).

[2]Which is illustrated at end of Section 2.2 by Table 2.2.

[3]By assuming that his/her answer was correct, one immediately also assumes that answers shown of some other players are incorrect. For example if the dropout answers that the *mol* is a woman and another player answers that the *mol* was part of a team of only men then either one of the answers is incorrect. So if his/her answer was correct, then he is immediately two correct answers ahead of that other player.

2.2 Previous Approach

The goal of the Exam Layer is to compute the probability that someone is the *mol* given all *executie* results known so far. Therefore the first implementation of the Exam Layer tries to determine for every *potential mol* player p

$$\mathcal{P}(p = \text{Mol} \mid D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n))$$

where $E_1, \ldots, E_n$ are the *executies* and $D_1, \ldots, D_n$ are the sets of dropouts corresponding to these *executies*.[4] This probability is unfortunately hard to compute directly, but it is easier to compute the probability that an *executie* result happened given previous *executie* results and given player p' is the *mol*, i.e.

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

because with the assumption that someone is the *mol* one knows which answers during the *test* were correct and which answers were wrong. And with this knowledge one can argue how likely every player would drop out. Thus it is preferred to use terms of

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

rather than terms of

$$\mathcal{P}(p = \text{Mol} \mid D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n))$$

Therefore the latter term needs to be expressed in the former terms, which is possible with Bayes theorem:

$$\mathcal{P}(p = \text{Mol} \mid D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n)) = \\ \frac{\mathcal{P}(D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n) \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol})}{\sum_{p'} \mathcal{P}(D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n) \mid p' = \text{Mol}) \cdot \mathcal{P}(p' = \text{Mol})}$$

where $\sum_{p'}$ is the sum over all *potential mol* players. Furthermore the probability that any *potential mol* player p' is the *mol* given no information is $\mathcal{P}(p' = \text{Mol}) = \frac{1}{\#\text{players}}$ where #players is the number of *potential mol* players. Moreover by applying the chain rule we obtain:

$$\mathcal{P}(D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n) \mid p = \text{Mol}) = \\ \prod_{i=1}^{n} \mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

Thus the latter term is now fully expressed in the former terms, where the former term

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

can be estimated by analysing the *test* corresponding to that *executie* E_i. To estimate this term, we should remind ourselves that every *test* usually consists of 20 questions[5] and that the cast reveals (assumed randomly) some of the answers given by the players on some of these questions. Based on an arbitrary player p^+, these questions can be categorized in 3 groups:

Own Questions These are the questions of which the question-answer structure is known and of which the selected answer of player p^+ was revealed (and probably of other players as well). Thus these questions leak information about the suspicions of player p^+.

[4]An exception to this rule is described in the Exception Handling appendix at A.1.1.

[5]Except for the final *test* which consists of 40 questions.

Other Questions These are the questions of which the question-answer structure is known, but of which the selected answer of player p^+ was not revealed. So the answer partitioning of this question is known. And maybe it is known what other players have answered on this question as well. However it is unknown what player p^+ has answered on this question.

Unseen Questions These are the questions of which the question-answer structure is unknown, but do exists since a *test* consists of 20 questions.

When estimating this former term it is assumed that for the 'Other Questions' every player p^+ picks another player *alive* p^* uniformly random per question on which he fills in his answer (which might cover other players as well). And for the 'Unseen Questions' we assume that these questions have 1 separate answer per player[6], where every player picks an answer uniformly random as well. Thus the probability of a correct guess for 'Unseen Questions' is $\frac{1}{|P|-1}$, where $|P|$ is the number of players *alive* during that *test*. The reasons for the assumptions were:

- For simplification purposes, the initial idea of the Moldel was to build a simple and understandable algorithm to predict the *mol*.
- To estimate an upper bound on $\mathcal{P}(E_i \mid E_{i-1}, \ldots, E_1,\ p' = \text{Mol})$, because by assuming that the dropout performs similar on the remaining questions as other players an upper bound is obtained. In reality the dropout would perform worse on the remaining questions.
- To prevent this upper bound estimation from being too rough. The assumption that players guess uniformly random on 'Other Questions' and 'Unseen Questions' combined with the structure of 'Unseen Questions' makes players perform worse on these questions than in reality. So passing the *executie* mostly depends on the 'Own Questions'.

Note though that an estimated upper bound is different from an actual upper bound. For later *executie* results the estimation of the term

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

by this model is often not an actual upper bound on this term, which is why these modelling assumptions are sometimes inaccurate. Nevertheless with these assumptions we randomly sample the number of correct answers per player and *test* given that someone is the *mol*. The pseudocode for a single random sample procedure is: (parameter definitions are on the next page)

Algorithm 1 Random Sampling of Correct Answers

```
function sampleCorrectAnswers($p^+$, $p'$, $P$, $Q_{own}$, $Q_{other}$, $|Q_{unseen}|$, $|Q|$, jokers)
 1: score = 0                                          ▷ The number of correct answers
 2: for question, answer in $Q_{own}$ do          ▷ Loop over questions with corresponding answer
 3:     if $p' \in$ answer then score += 1
 4: end for
 5: for question in $Q_{other}$ do
 6:     Pick $p \in P/\{p^+\}$ uniform random
 7:     Pick answer $\in$ question, s.t. $p \in$ answer
 8:     if $p' \in$ answer then score += 1
 9: end for
10: for $i = 1, \ldots, |Q_{unseen}|$ do
11:     if rand() < $\frac{1}{|P|-1}$ then score += 1                ▷ Note $0.0 \leq$ rand() $\leq 1.0$
12: end for
13: return min(score + jokers, $|Q|$)                    ▷ Add the jokers to the score
```

[6]Similar to question 20 of every *test*, which is the question "Who is the Mole?" that has a separate answer for every player.

And the parameters for this procedure are:

- p^+ is the player for which the number of correct answers is randomly sampled.
- p' is the player that is assumed to be the *mol*.
- P is the set of all players that were *alive* during that *test* (and $|P|$ is the size of this set).
- Q_{own} is a set of questions-answer pairs of player p^+ which were revealed.
- Q_{other} is a set of questions which were revealed, but of which the answer of player p^+ was not revealed.
- $|Q_{\text{unseen}}|$ is the number of unseen questions.
- $|Q|$ is the number of questions in total.
- jokers is the number of *jokers* used by player p^+, which is equal to ∞ if p^+ used a *vrijstelling*.

If one executes this procedure for all players $p^+ \in P \setminus \{p'\}$ and for the *test* corresponding to *executie* E_i then one can sample the dropout(s) D' for *executie* E_i, because the player(s) with the least correct answers drop(s) out. So by doing this many times

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

can be estimated as[7]

$$\frac{\text{\#samples s.t. } D' = D_i \text{ with } p' = \text{Mol}}{\text{\#samples}}$$

where #samples should be large enough.[8] And by doing this estimation for every *executie* E_i and every possible *mol* p', we can finally estimate

$$\mathcal{P}(p = \text{Mol} \mid D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n))$$

for every possible *mol* p, which gives us the *mol* likelihood distribution over all players.

This first implementation of the Exam Layer is therefore a mathematical model rather than a machine learning model. It has the advantage that one does not need train data or has to learn something in order to predict the *mol*, because the model already understands the mechanics. Thus there are not any issues related to the shortage of training data and misunderstanding the pattern in the data. On the other hand, this model has a lot of disadvantages, which includes:

- Inaccurate assumptions are made. The assumption that the players pick a random answer for 'Other Questions' and 'Unseen Questions' during the first episodes might be accurate, however in later episodes (especially in the semi-finals and finals) this assumption is violated. Also it is highly unlikely that all 'Unseen Questions' have a separate answer per player. It is more common that there are also easy questions among them. Furthermore the assumption that players have an equal guessing probability seems inaccurate as well. The dropout often has a lower probability of making a correct guess. Last of all, the model indirectly assumes that the cast reveals random answers of players and does not select particular answers of players, which we are unsure about. So this implementation is not the most accurate one.

[7] An exception to this rule is described in the Exception Handling appendix at A.1.2.

[8] The sample size #samples used in the original Moldel was 10000.

– Inconsistent predictions over episodes. It is quite common for this model to have totally different predictions for sequential episodes. Which is understandable, because only a few answers per episode are revealed and we assumed independence between answers given by the same player in different *tests*, because for 'Unseen Questions' and 'Unknown Questions' the answer is randomly selected. Thus all answers in previous *tests* are ignored, which carries valuable information, since the answers of a player are quite similar for sequential episodes.

– Issues with random sampling. To approximate $\mathcal{P}(E_i \mid p' = \text{Mol})$ accurately one needs a lot of samples, which makes the algorithm very slow, because $\mathcal{P}(E_i \mid p' = \text{Mol})$ is estimated per episode and per player. To illustrate this point, for predicting the *mol* after episode 3 with 8 players still left in the game and #samples $= 10000$, you sample the number of correct answers $3 \cdot 8 \cdot 10000 \cdot 7 = 1,680,000$ times. A solution to speed up the sampling is to reduce #samples, but this results in more inaccurate and more unstable predictions.

Nevertheless this model illustrates the aspects of the *test* and *executie* quite well. For example with the theory of this model one can estimate the likelihood of dropping out for a non-*mol* player if his/her answer was correct (assuming all 20 questions are 'Unseen Questions'):

#P \ #W	0	1	2	3	4	5	6	7	8
1	0.4373	0.2612	-	-	-	-	-	-	-
2	0.2456	0.1647	0.1157	-	-	-	-	-	-
3	0.1538	0.1065	0.0762	0.0562	-	-	-	-	-
4	0.1018	0.0709	0.0507	0.0371	0.0278	-	-	-	-
5	0.0692	0.0478	0.0338	0.0244	0.0179	0.0133	-	-	-
6	0.0475	0.0323	0.0223	0.0157	0.0112	0.0081	0.0059	-	-
7	0.0324	0.0215	0.0144	0.0098	0.0068	0.0047	0.0033	0.0023	-
8	0.0217	0.0139	0.0090	0.0059	0.0039	0.0025	0.0017	0.0011	0.0008

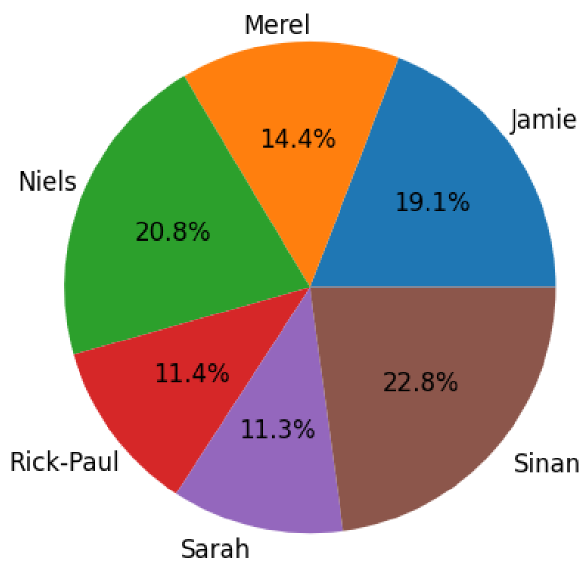
Table 2.2: Likelihood of dropping out if the answer was correct

where #P is the number of non-*mol* other players and #W is the number of non-*mol* other players that had at least one wrong answer. What becomes clear from this table is that the chance of dropping out is very rare when the revealed answer is correct, especially when #P is still large. Thus the answer of the dropout is often wrong according to this model.

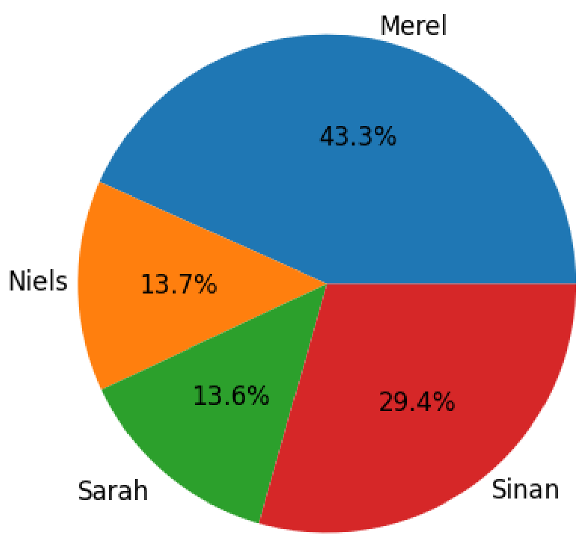
Furthermore this model can be surprisingly accurate for early *executie* results. For instance this model was able to predict the *mol* correctly after episode 7 of the 19th season of 'Wie is de Mol?' (see Section 2.1). In this episode the dropouts during the *executie* were $D_6 = \{\text{Jamie}, \text{Rick-Paul}\}$ and we had:

$$\mathcal{P}(D_6 = \text{Dropout}(E_6) \mid D_5 = \text{Dropout}(E_5), \ldots, D_1 = \text{Dropout}(E_1),\ \text{Merel} = \text{Mol}) \approx 7.90 \cdot 10^{-2}$$
$$\mathcal{P}(D_6 = \text{Dropout}(E_6) \mid D_5 = \text{Dropout}(E_5), \ldots, D_1 = \text{Dropout}(E_1),\ \text{Niels} = \text{Mol}) \approx 1.79 \cdot 10^{-2}$$
$$\mathcal{P}(D_6 = \text{Dropout}(E_6) \mid D_5 = \text{Dropout}(E_5), \ldots, D_1 = \text{Dropout}(E_1),\ \text{Sarah} = \text{Mol}) \approx 3.12 \cdot 10^{-2}$$
$$\mathcal{P}(D_6 = \text{Dropout}(E_6) \mid D_5 = \text{Dropout}(E_5), \ldots, D_1 = \text{Dropout}(E_1),\ \text{Sinan} = \text{Mol}) \approx 2.83 \cdot 10^{-2}$$

where there are only 6 *executie* results, because in episode 6 there was no *executie*. Moreover, as becomes clear by these results, is that the scenario of Jamie and Rick-Paul dropping out is most plausible if Merel was the *mol* (which was indeed the case). So after episode 7, there was a major altering in the result, see Figure 2.1. After episode 6 Merel was not the most suspected *mol* by the Moldel with a likelihood of 0.144. However with the reasoning of *test* and *executie*



(a) Prediction after episode 6



(b) Prediction after episode 7

Figure 2.1: Predictions of Previous Exam Layer

in episode 7 (as explained in Section 2.1), Merel was the most suspected *mol* by the Moldel after episode 7 with a likelihood of 0.433. Thus the old version of the Exam Layer was able to predict the actual *mol* for this season. Nevertheless for earlier episodes and other seasons, this approach was often inaccurate. Therefore there was a need for a more stable and accurate approach for the Exam Layer, which is discussed in the next sections.

2.3 Current Approach - Drop Layer

The new approach for the Exam Layer, also known as the current approach of the Exam Layer, is split up in two separate layers: the Exam Drop Layer and the Exam Pass Layer. The Exam Drop Layer discussed in this section investigates the answers of the dropouts and tries to exclude players based on the given answers. On the other hand the Exam Pass Layer, discussed in the next section, predicts the *mol* based on *joker* and *vrijstelling* usage of the *potential mol* players. These are the only aspects that are analysed based on the *test* and *executie* data. So the current implementation of the Exam Layer does not analyse the answers of players that pass the *test*, which were analysed by the previous implementation of the Exam Layer. Although we should notice that players that pass the *executie* might drop out later during the season and therefore are used by the Exam Drop Layer. A second aspect of the game that is also not analysed are *executies* where only part of the players see their screen and nobody drops out, which were analysed in the previous approach (see exceptions A.1.1. and A.1.2.). These situations unfortunately have not happened often enough to proper analyse them and are therefore fully ignored by the Exam Pass Layer and mostly ignored by the Exam Drop Layer. However the Exam Drop Layer takes into account the answers given during these type of episodes for players that drop out later, but does not assume who would have dropped out during these episodes. So there are some aspects of the *tests* and *executies* which are not analysed anymore, but were analysed by the previous implementation. However there are also new aspects that are analysed by this new implementation, which includes answers given during previous episodes by the dropouts.

2.3.1 Encoding

The Exam Drop Layer looks at all combinations of player-question-answer-player quartets, where:[9]

- The first player P_1 in this quartet corresponds to the player that has dropped out non-*voluntarily*.
- The question $Q' = \{A_1, \ldots, A_n\}$, which is a set of answers, in this quartet corresponds to a question that has been answered by this dropout. These questions are not limited to only questions of the *executie* where the player dropped out, but include questions of previous episodes as well.
- The answer A_k, which is a set of players, in this quartet corresponds to the answer that is selected by player P_1 for question Q'.
- The second player P_2 in this quartet corresponds to any player that was *alive* during this episode, which is either included in A_k or not included in A_k. The purpose of this second player P_2 is to determine the likelihood of being the *mol* for this player.

The example below shows all quartets for a fictive example season:

Example 1. *Suppose there is a season that started with 4 players named Diederik, Jochem, Sanne and Thomas of whom Diederik dropped out during the first episode, Jochem dropped out during the second episode (finals), Sanne was the winner and Thomas was the* mol*. During the first episode the following questions are revealed:*

$$Q'_1 = \{\{\textit{Diederik}, \textit{Jochem}\}, \{\textit{Sanne}, \textit{Thomas}\}\} \quad Q'_2 = \{\{\textit{Sanne}\}, \{\textit{Diederik}, \textit{Jochem}, \textit{Thomas}\}\}$$
$$Q'_3 = \{\{\textit{Diederik}, \textit{Sanne}\}, \{\textit{Jochem}\}, \{\textit{Thomas}\}\}$$

where Diederik answered on $Q'_1 : \{\textit{Diederik}, \textit{Jochem}\}$, Thomas answered on $Q'_2 : \{\textit{Sanne}\}$ and Jochem answered on $Q'_3 : \{\textit{Diederik}, \textit{Sanne}\}$. Furthermore in the second episode the following questions are revealed:

$$Q'_4 = \{\{\textit{Jochem}\}, \{\textit{Sanne}, \textit{Thomas}\}\} \quad Q'_5 = \{\{\textit{Jochem}\}, \{\textit{Sanne}\}, \{\textit{Thomas}\}\}$$

where Sanne answered on $Q'_5 : \{\textit{Thomas}\}$, Thomas answered on $Q'_5 : \{\textit{Sanne}\}$ and Jochem answered on $Q'_4 : \{\textit{Sanne, Thomas}\}$ and $Q'_5 : \{\textit{Sanne}\}$. Then all quartet combinations (14 in total) for this season are:

$$\begin{array}{cc}
(\text{Diederik}, Q'_1, \{\text{Diederik}, \text{Jochem}\}, \text{Diederik}) & (\text{Diederik}, Q'_1, \{\text{Diederik}, \text{Jochem}\}, \text{Jochem}) \\
(\text{Diederik}, Q'_1, \{\text{Diederik}, \text{Jochem}\}, \text{Sanne}) & (\text{Diederik}, Q'_1, \{\text{Diederik}, \text{Jochem}\}, \text{Thomas}) \\
(\text{Jochem}, Q'_3, \{\text{Diederik}, \text{Sanne}\}, \text{Diederik}) & (\text{Jochem}, Q'_3, \{\text{Diederik}, \text{Sanne}\}, \text{Jochem}) \\
(\text{Jochem}, Q'_3, \{\text{Diederik}, \text{Sanne}\}, \text{Sanne}) & (\text{Jochem}, Q'_3, \{\text{Diederik}, \text{Sanne}\}, \text{Thomas}) \\
(\text{Jochem}, Q'_4, \{\text{Sanne}, \text{Thomas}\}, \text{Jochem}) & (\text{Jochem}, Q'_4, \{\text{Sanne}, \text{Thomas}\}, \text{Sanne}) \\
(\text{Jochem}, Q'_4, \{\text{Sanne}, \text{Thomas}\}, \text{Thomas}) & (\text{Jochem}, Q'_5, \{\text{Sanne}\}, \text{Jochem}) \\
(\text{Jochem}, Q'_5, \{\text{Sanne}\}, \text{Sanne}) & (\text{Jochem}, Q'_5, \{\text{Sanne}\}, \text{Thomas})
\end{array}$$

Note that in a normal non-fictive season, there are about 10 players which includes 8 dropouts and 9 episodes with a *test*. Hence a normal season has much more quartet combinations than the example shown above. The Exam Drop Layer converts these quartets (P_1, Q', A_k, P_2) into a matrix of feature values F, where the rows in this matrix correspond to the different quartets and the columns correspond to the features values.

[9]An exception to this rule is described in the Exception Handling appendix at A.1.3.

The features used to encode every quartet $Q = (P_1, Q', A_k, P_2)$ are:

Exam Episode Number The episode number in which the *test* happened with question Q'. The corresponding *executie* is defined as the Exam Executie. This feature is included, because it is expected that answers given during later *tests* are more likely correct than answers given during earlier *tests*.

Drop Episode Number The episode number of the *executie* in which P_1 dropped out. The corresponding *executie* is defined as the Drop Executie.[10] This feature is included, because it is expected that earlier dropouts are more likely to suspect the wrong *mol* than later dropouts.

Fail Executie A binary value which is 1 if the Exam Executie is equal to the Drop Executie and 0 otherwise. In other words this feature is 1 if P_1 dropped out (or his/her red screen was shown) during this *executie* and 0 otherwise. This feature is included, because it is expected that someone dropping out has more wrong answers during the *test* which caused him to dropout than in earlier *tests*.

Real Executie A binary value which is 1 if someone/multiple dropped out during the Exam Executie (including P_1), because he/she/they had the worst *test* score. Otherwise the feature value is 0 if nobody dropped out during the Exam Executie (even if some screens were revealed) or if someone dropped out *voluntarily*. This feature is included, because it is expected that if nobody dropped out then you could have dropped out, which means that your answers are more likely to be wrong.

Exam Players The number of non-*mol* players that were *alive* before the Exam Executie.[11] This feature is included, because it is expected that with less players *alive* you are more likely to suspect the actual *mol*.

Drop Players The number of non-*mol* players that were *alive* (including P_1) before the Drop Executie.[12] This feature is included, because it is expected that if you made it far in the game, then you are more likely to suspect the actual *mol*.

Entropy The entropy is defined as $-\sum_{i=1}^{n} \frac{|A_i|}{|P|} \ln\left(\frac{|A_i|}{|P|}\right)$ where $|P|$ is the number of players *alive* before the Exam Executie and $|A_i|$ is the number of players included in answer A_i. This feature is included, because it is expected that questions with a higher entropy are more likely to be answered wrong than questions with a lower entropy.

Answer Players The number of players included in answer A_k, excluding player P_1, which is $|A_k \setminus \{P_1\}|$. This feature is included, because it is expected that if you selected an answer with more players then your answer is more likely to be correct.

Answer Probability The probability of selecting answer A_k if player P_1 would have selected a random player as *mol* and filled in question Q on that player. This is defined as $\frac{|A_k \setminus \{P_1\}|}{|P| - 1}$. This feature is included by similar reason as the previous feature.

Same Pickers For this we define the suspicions for every player p and every *executie* e as $S_{p,e}$ which is the union of all answers given by player p in the test corresponding to *executie* e excluding player p itself. If no answer was revealed of player p or $S_{p,e} = \emptyset$ then $S_{p,e}$ is

[10]An exception to this rule is described in the Exception Handling appendix at A.1.4.

[11]The number of non-*mol* player *alive* is equal to the number of players *alive* minus 1.

defined as all players P that were *alive* before *executie* e excluding player p. The Same Pickers feature for the Exam Executie E is now defined as:[12]

$$\frac{\#\{p : p \in P \setminus (D \cup \{P_1\}) \ \wedge \ S_{p,E} \cap (A_k \setminus \{P_1\}) \neq \emptyset\}}{|P \setminus (D \cup \{P_1\})|}$$

where D is the set of non-*voluntary* dropouts during this *executie* (if nobody dropped out non-*voluntarily* then this set is empty). In simple words this feature represents the relative number of other players that pass the Exam Executie and share a common suspected player with your given answer A_k. This feature is included, because it is expected that if you suspect a player not suspected by other players and you drop out then that player is less likely to be the *mol*.

Same Pick Probability The Same Pick Probability feature is closely related to the Same Pickers feature, but also takes into account how many suspected players are shared. It is defined as:

$$\text{mean}\left(\text{SPP}(A_k \setminus \{P_1\},\ p) : p \in P \setminus (D \cup \{P_1\})\right)$$

where the function SPP is defined as:[13]

$$\text{SPP}(A, p) = \text{mean}\left(\frac{|A \cap (a \setminus \{p\})|}{|a \setminus \{p\}|} : a \in \text{AnswerGivenBy}(p, E)\right)$$

and AnswerGivenBy(p, e) is defined as the function that returns all answers given by player p in the test corresponding to *executie* e. Similarly to Same Pickers feature E is defined as the Exam Executie, P is defined as the players *alive* before this *executie* and D is defined as the non-*voluntary* dropouts in this *executie*. Moreover in case no answer was given by player p in the test corresponding to the Exam Executie E, SPP(A, p) is defined as:

$$\text{SPP}(A, p) = \frac{|A|}{|P| - 1}$$

This feature is included by similar reason as the previous feature.

Drop Test Jokers More This feature is equal to the relative number of other players p that participated in the Drop Executie which were not part of the non-*voluntary* dropouts and used more *jokers* than player P_1 (which dropped out). Using a *vrijstelling* is considered as using ∞ number of *jokers* in this feature definition and in the next feature definitions. Mathematically spoken, if we define function $J(p, e)$ as the joker usage of player p in the test corresponding to *executie* e, then this feature is defined as:

$$\frac{\#\{p : p \in P \setminus (D \cup \{P_1\}) \wedge J(p, E) > J(P_1, E)\}}{|P \setminus (D \cup \{P_1\})|}$$

where E is the Drop Executie, P are the players *alive* before this *executie* and D are the non-*voluntary* dropouts of this *executie*. This feature is included, because it is expected that it is more likely you suspected the actual *mol* if you dropped out because other players used more *jokers* and *vrijstellingen* than you did.

Drop Test Jokers Less This feature is equal to the relative number of other players p that participated in the Drop Executie which were not part of the non-*voluntary* dropouts and used less *jokers* than player P_1 (which dropped out). Mathematically spoken, this feature is defined as:

$$\frac{\#\{p : p \in P \setminus (D \cup \{P_1\}) \wedge J(p, E) < J(P_1, E)\}}{|P \setminus (D \cup \{P_1\})|}$$

[12] $|P \setminus (D \cup \{P_1\})| \geq 1$, because the *mol* always passes the *executie* together with at least another player. Thus $|P \setminus D| \geq 2$, which implies that $|P \setminus (D \cup \{p\})| \geq 1$ for any player p including P_1.

[13] An exception to this rule is described in the Exception Handling appendix at A.1.5.

where E is the Drop Executie, P are the players *alive* before this *executie* and D are the non-*voluntary* dropouts of this *executie*. This feature is included, because it is expected that it is more likely you suspected the wrong *mol* if you drop out during an *executie* when using more *jokers* than other players.

Exam Test Jokers More This feature is equal to the relative number of other players p that participated in the Exam Executie which were not part of the non-*voluntary* dropouts and used more *jokers* than player P_1. Mathematically spoken, this feature is defined as:[14]

$$\frac{\#\{p : p \in P \setminus (D \cup \{P_1\}) \wedge J(p, E) > J(P_1, E)\}}{|P \setminus (D \cup \{P_1\})|}$$

where E is the Exam Executie, P are the players *alive* before this *executie* and D are the non-*voluntary* dropouts of this *executie*. This feature is included, because it is expected that it is more likely you suspected the actual *mol* if you passed an *executie* where other players used more *jokers* and *vrijstellingen* than you did.

Exam Test Jokers Less This feature is equal to the relative number of other players p that participated in the Exam Executie which were not part of the non-*voluntary* dropouts and used less *jokers* than player P_1. Mathematically spoken, this feature is defined as:[15]

$$\frac{\#\{p : p \in P \setminus (D \cup \{P_1\}) \wedge J(p, E) < J(P_1, E)\}}{|P \setminus (D \cup \{P_1\})|}$$

where E is the Exam Executie, P are the players *alive* before this *executie* and D are the non-*voluntary* dropouts of this *executie*. This feature is included, because it is expected that it is less likely you suspected the actual *mol* if you passed an *executie* when using more *jokers* and *vrijstellingen* than other players.

Drop More Jokers A binary value which is 1 if any of the non-*voluntary* dropouts in the Exam Executie used more *jokers* than player P_1. It is 0 if there are no non-*voluntary* dropouts or if none of these dropouts used more *jokers* than player P_1. This feature is included, because it is expected that it is more likely you suspected the actual *mol* if you passed an *executie* where the dropout used more *jokers* than you did.

Drop Less Jokers A binary value which is 1 if any of the non-*voluntary* dropouts in the Exam Executie used less *jokers* than player P_1. It is 0 if there are no non-*voluntary* dropouts or if none of these dropouts used less *jokers* than player P_1. This feature is included, because it is expected that it is less likely you suspected the actual *mol* if you passed an *executie* where you used more *jokers* and *vrijstellingen* than the dropout.

2.3.2 Discretization

With these 17 features, the most important aspects of the *executie* and *test* data are covered. Also note that for the computation of these features you do not have to know the *mol*, so you can compute these features for a season for which we want to do a prediction. But there is still an important aspect that is missing in these features which is whether $P_2 \in A_k$ or not in every quartet $Q = (P_1, Q', A_k, P_2)$. Moreover to allow the Exam Drop Layer to learn more complex relationships than a simple linear relationship, the feature matrix F is discretized using a K-bins discretization with one-hot encoding [12, sections 6.3.4 & 6.3.5.1]. This technique partitions all features independently into K one-dimensional intervals (bins) and encodes every feature value as a sequence of binary values which represents whether that value is part of a given interval. And since a value can only be part of exactly one interval, exactly one of the binary values is

[14]For this it is assumed that $\infty < \infty$, $\infty = \infty$ and $\infty > \infty$ are all false statements.

equal to 1 and the others are 0's. This is why it is named one-hot encoding. An example of K-bins discretization is presented in the following example:

Example 2. *Suppose we have the data points:*

$$X_1 = (9, 4) \quad X_2 = (-5.5, -3.5) \quad X_3 = (10, 0)$$

with the following selected intervals:

$$I_{1,1} = (-\infty, -5) \quad I_{1,2} = [-5, 10) \quad I_{1,3} = [10, \infty) \quad I_{2,1} = (-\infty, -3) \quad I_{2,2} = [-3, 4) \quad I_{2,3} = [4, \infty)$$

where the I_1 intervals correspond to the first feature and the I_2 intervals correspond to the second feature. Then these data points are transformed with K-bins discretization into:

$$X_1 \to (0, 1, 0, 0, 0, 1) \quad X_2 \to (1, 0, 0, 1, 0, 0) \quad X_3 \to (0, 0, 1, 0, 1, 0)$$

Other options to deal with more complex non-linear relationships are polynomial encoding and spline encoding [13, sections 2.4.3-2.4.5], which are more suitable in case the features are continuous. However the Exam Drop Layer has a lot of discrete features and for continuous features the same value often occurs multiple times. Hence K-bins discretization is preferred over polynomial encoding and spline encoding [13, sections 2.4.3-2.4.5]. Furthermore the features could have also been discretized multi-dimensionally with clustering techniques. Though with one-dimensional discretization all separable functions $f(x_1, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$ can already be approximated, which are a ton of functions. Thus multi-dimensional clustering is not required for feature discretization.

Definition 2. $f(x_1, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$ *is a separable function if it can be written as:*

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{n} f_i(x_i)$$

with $f_i(x_i) : \mathbb{R} \to \mathbb{R}$ an arbitrary one-dimensional function.

Example 3. $f(x_1, \ldots, x_n) = \ln(\prod_{i=1}^{n-1} x_i) + x_n$ *is a separable functions, because:*

$$f(x_1, \ldots, x_n) = \ln\left(\prod_{i=1}^{n-1} x_i\right) + x_n = \sum_{i=1}^{n-1} \ln(x_i) + x_n = \sum_{i=1}^{n} f_i(x_i)$$

However the bin sizes and the number of bins have to be chosen properly. To select the sizes of the bins for the Exam Drop Layer, a K-means clustering approach is used [14, section 9.1] for each feature independently. K-means clustering assigns to each group of feature values F_i^T a vector of centers $\mu_1, \ldots, \mu_k$ such that:

$$\sum_{x \in F_i^T} (x - \mu(x))^2$$

is minimized, where $\mu(x)$ is the closest center (of $\mu_1, \ldots, \mu_k$) to value x. The k intervals $I_1, \ldots, I_k$ used for K-bins discretization are now defined as the regions that are closest to respectively the centers $\mu_1, \ldots, \mu_k$ from all these centers, i.e.

$$I_j = \{x \in \mathbb{R} : \mu(x) = \mu_j\}$$

To select the number of bins for the Exam Drop Layer, we use a procedure in which different number of bins are allowed per feature. This procedure starts with 2 bins for each feature and it iteratively expands the number of bins for the feature i with the most information gain with respect to the clustering, i.e.[15]

$$\arg\max_i E(F_i^T, b_i + 1) - E(F_i^T, b_i)$$

where F_i^T is the group of values for the ith feature and b_i is the current number of bins for feature i. Moreover the entropy function $E(F_i^T, b)$ is defined as:

$$E(F_i^T, b) = -\sum_{j=1}^{b} p(F_i^T, j) \cdot \ln(p(F_i^T, j)) \quad \text{with} \quad p(F_i^T, j) = \frac{\#\{x \in F_i^T : \mu(x) = \hat{\mu}_j\}}{|F_i^T|}$$

where $\hat{\mu}_j$ are optimally placed means by the K-means clustering procedure. This process of selecting and expanding a bin is repeated 40 times, after which we obtain the number of bins used by K-bins discretization for all features.[16] Doing this for season 5 up to *21* as training data, results in the number of bins for each feature as shown by Table 2.3.

Feature Name	#Bins	Feature Name	#Bins	Feature Name	#Bins
Exam Episode Number	5	Drop Episode Number	7	Fail Executie	2
Real Executie	2	Exam Players	7	Drop Players	6
Entropy	7	Answer Players	5	Answer Probability	6
Same Pickers	8	Same Pick Probability	6	Drop Ep. Jokers More	3
Drop Ep. Jokers Less	2	Exam Ep. Jokers More	2	Exam Ep. Jokers Less	2
Drop More Jokers	2	Drop Less Jokers	2		

Table 2.3: Number of bins per feature

So with these settings, the matrix F can finally be discretized into a binary matrix F^+. And to include the $P_2 \in A_k$ component into the features, we transform every row F^+ that corresponds with quartet Q into:

$$(F_{\text{new}})_Q^+ = [(F_{\text{old}})_Q^+, (F_{\text{old}})_Q^+ \cdot \mathbb{1}_{P_2 \in A_k}, \mathbb{1}_{P_2 \in A_k}]$$

where $\mathbb{1}_{P_2 \in A_k}$ is the indicator function which is equal to 1 if $P_2 \in A_k$ and 0 otherwise. After this last transformation the final raw feature encoding is obtained for every quartet Q. The corresponding labels for these quartets Q is whether P_2 is the *mol* or not.

2.3.3 Feature Reduction

In total there are 2910 of these quartets when using all data from season 5 up to *21* and each of these quartets Q is encoded by 149 features. A lot of these features are unfortunately not correlated to the likelihood of being the *mol* for player P_2 and are correlated to each other which increases the risk of overfitting. Hence the features should be reduced to a group of informative features which are not correlated to each other. To achieve this two methods are used: Mann-Whitney U Selection [15] and Principal Component Analysis [14, section 12.1]. We start by using a Mann-Whitney U Selection, which filters out all features that are uninformative. This method loops over all feature groups $(F^+)_i^T$ and separates it in two groups M_i and N_i. M_i are all feature values of $(F^+)_i^T$ with P_2 being the *mol* and N_i are all feature values of $(F^+)_i^T$ with P_2 not being the *mol*. For every feature i a Mann-Whitney U test is applied which assumes the values in each group (M_i and N_i) are independently and identically distributed. This test

[15] In the argmax we ignore all features for which the number of bins b_i is already equal to the number of distinct values for feature i.

[16] We achieve a good accuracy with 40 additional bins, see appendix B.

checks if M_i is more biased towards lower or higher values than N_i. The null hypothesis and alternative hypothesis for this test are respectively:

$$H_0 : \forall_{z\in\mathbb{R}}\, F_{M_i}(z) = F_{N_i}(z) \quad H_1 : \forall_{z\in\mathbb{R}}\, F_{M_i}(z) < F_{N_i}(z) \bigvee \forall_{z\in\mathbb{R}}\, F_{M_i}(z) > F_{N_i}(z)$$

with $F_{M_i}(z)$ the cumulative distribution function behind M_i and $F_{N_i}(z)$ the cumulative distribution function behind N_i. After applying the Mann-Whitney U Test, the corresponding p-value is checked. Every feature i is removed if the corresponding p-value is larger than 0.01, which means there is no significant difference between the values of group M_i compared to group N_i.

After using the Mann-Whitney U Selection for seasons 5 up to *21*, we end up with 21 features in total. Unfortunately a lot of these features are correlated with each other and could therefore cause issues with overfitting. Thus to prevent this from happening a second method is used called Principal Component Analysis (PCA) which is a well-known method that combines correlated features into single components. This is done by doing an eigenvalue decomposition on the covariance matrix of F^+ (after features have been excluded by the Mann-Whitney U selection). The eigenvectors with the largest eigenvalues (which explains the most of the variance) are now taken as Principal Components on which the feature encoding of every quartet is projected upon. The number of Principal Components that we use is the smallest number such that at least 95% of all variance is explained. Using this method reduces the number of features from 21 down to 12. Hence we finally end up with 12 informative uncorrelated features, which is better than having 149 mostly correlated and uninformative features.

2.3.4 Logistic Regression

By applying both feature selection methods, the training data has been transformed. The same procedure can of course also be used to obtain the predict data in proper format. The only difference is that we do not retrieve all quartets corresponding to that season, but only all quartets that are known up to the latest episode of that season. To predict the corresponding labels whether P_2 is the *mol* for the predict data, a Logistic Regression model [14, section 4.3.2] is trained on the train data, which is a probabilistic binary classifier model.[17] The likelihood of P_2 being the *mol* for quartet $Q = (P_1, Q', A_k, P_2)$ in this model is estimated as $\sigma(w^T x)$ where:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

and where x is the feature encoding of quartet Q, plus a bias term 1 and w are the weights corresponding to the features that get trained.[18] The likelihood of not being the *mol* in a Logistic Regression model is estimated as $1 - \sigma(w^T x)$. And because $0 < \sigma(z) < 1$ for all $z \in \mathbb{R}$ the likelihood is always well defined (see Section 1.4) for being the *mol* and for not being the *mol*. To train the Logistic Regression model a maximum likelihood estimation is applied on w, i.e.

$$\arg\max_{w} \prod_{x,y,m\in T} \sigma(w^T x)^{my} \cdot (1 - \sigma(w^T x))^{m(1-y)} \tag{2.1}$$

where T is the group of all train triplets in which x is the feature encoding (plus a bias term) for every quartet Q, y is the corresponding label (whether P_2 in quartet Q is the *mol*) and m is the train multiplier for the corresponding train triplet.[19] This train multiplier for quartet Q is defined

[17] Note that a Logistic Regression models can also be applied on classification problems with more than 2 classes. However this is not discussed here, since there are only 2 classes.

[18] So x and w are vectors of size 13, because there are 12 features plus an additional bias term.

[19] In Section 4.3.2 of 'Pattern Recognition and Machine Learning' [14] is described how w is computed without train multipliers. To determine w in this specific formula one follows the same approach, however there is an additional m term in the gradient.

as $\frac{1}{\text{\#answers}(Q)}$ where #answers(Q) is the number of questions revealed of player P_1 in the *test* in which question Q' was revealed.[21] A multiplier term is used in the training process, because answers given by the same player in the same *test* are expected to be similar, thus they do not provide much additional information.

For the implementation of the Logistic Regression, a Scikit-learn library is used [16]. This library uses a Limited Memory Broyden–Fletcher–Goldfarb–Shanno algorithm (LBFGS) [17] to iteratively update w based on the gradient of the log loss associated with Equation 2.1. This procedure stops after 100 iterations of updating w or if the largest gradient coefficient is smaller than 10^{-4} during an iteration. After the training process we predict the *mol* likelihood of P_2 for every prediction quartet Q and aggregate them using the following steps:

1. If players included in the answer A_k gets a higher *mol* likelihood than players not included in the answer A_k, then the *mol* likelihood prediction is changed to $1/|P|$ for all players *alive* during that episode. The reasoning behind this is that in the latest seasons it is more common that the *mol* is not suspected by late dropouts, whereas in earlier seasons almost all late dropouts suspected the *mol*.
2. All quartets Q corresponding to the same *test* and same player P_1 are grouped together. For every group the geometric mean (see Section 5.2.3) of the *mol* likelihoods of P_2 is taken.
3. The geometric mean of all groups are naïvely aggregated (see Section 5.2.2), i.e. all geometric means for player P_2 are multiplied together.
4. When all likelihoods have been multiplied, all non-*potential mol* players have their likelihood set to zero, after which the likelihoods are *normalized*.

An example of this aggregation procedure as follow-up of example 1 is given below.

Example 4. *Suppose the predicted* mol *likelihoods for the quartets in example 1 are:*

P_1	Q'	A_k	P_2	Likelihood
Diederik	Q'_1	$\{\text{Diederik}, \text{Jochem}\}$	Diederik	0.15
Diederik	Q'_1	$\{\text{Diederik}, \text{Jochem}\}$	Jochem	0.15
Diederik	Q'_1	$\{\text{Diederik}, \text{Jochem}\}$	Sanne	0.35
Diederik	Q'_1	$\{\text{Diederik}, \text{Jochem}\}$	Thomas	0.35
Jochem	Q'_3	$\{\text{Diederik}, \text{Sanne}\}$	Diederik	0.30
Jochem	Q'_3	$\{\text{Diederik}, \text{Sanne}\}$	Jochem	0.20
Jochem	Q'_3	$\{\text{Diederik}, \text{Sanne}\}$	Sanne	0.30
Jochem	Q'_3	$\{\text{Diederik}, \text{Sanne}\}$	Thomas	0.20
Jochem	Q'_4	$\{\text{Sanne}, \text{Thomas}\}$	Jochem	0.40
Jochem	Q'_4	$\{\text{Sanne}, \text{Thomas}\}$	Sanne	0.30
Jochem	Q'_4	$\{\text{Sanne}, \text{Thomas}\}$	Thomas	0.30
Jochem	Q'_5	$\{\text{Sanne}\}$	Jochem	0.40
Jochem	Q'_5	$\{\text{Sanne}\}$	Sanne	0.20
Jochem	Q'_5	$\{\text{Sanne}\}$	Thomas	0.40

Table 2.4: Predicted Likelihoods

*Then the non-*normalized mol *likelihood of Sanne by the Exam Drop Layer is:*

$$0.35 \cdot 0.25 \cdot \sqrt{0.30 \cdot 0.20} \approx 2.14 \cdot 10^{-2}$$

*And the non-*normalized mol *likelihood of Thomas by the Exam Drop Layer is:*

$$0.35 \cdot 0.25 \cdot \sqrt{0.30 \cdot 0.40} \approx 3.03 \cdot 10^{-2}$$

After normalization the mol *likelihood of Sanne is* 0.414 *and of Thomas* 0.586

2.4 Current Approach - Pass Layer

Thus with the technique described in previous sections, the *mol* can be predicted based on the answers of the dropouts. However when it comes to *test* and *executie* results there is more to analyse. One of these things is the relationship between the usage of *jokers* & *vrijstellingen* and being the *mol*. This relationship looks odd at first sight, because the *mol* is not more/less likely to receive *jokers* and *vrijstellingen* compared to other players. Nevertheless if someone used many *jokers* and *vrijstellingen* then it is more explainable why that player reached the finals compared to a player that did not use any of them. So the player that did not use any *jokers* and *vrijstellingen* is more likely to be the *mol* than the player that did use many of them. Another "Wie is de Mol?"-analyst [18] had a similar feeling and recognized the relationship between this problem and the Monty Hall problem [4], which inspired me to the following modelling approach for the Exam Pass Layer. The rules of this layer are:

1. The likelihood of player p being the *mol* without any information $\mathcal{P}(p = \text{Mol})$ is equal to $1/|P_+|$ for every *potential mol* p, where $|P_+|$ is the number of *potential mol* players.
2. All *executie* results with a non-*voluntary* dropout given p is the *mol* are computed using the chain rule, i.e. if $D_1, \dots, D_n$ are the set of dropouts in different *executies* then
$$\mathcal{P}(D_1 = \text{Dropout}(E_1), \dots, D_n = \text{Dropout}(E_n) \mid p = \text{Mol}) =$$
$$\prod_{i=1}^{n} \mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \dots, D_1 = \text{Dropout}(E_1),\ p = \text{Mol})$$
3. The likelihood of a set of players D dropping out which includes the *mol* in any *executie* E is equal to 0, i.e.
$$\mathcal{P}(D = \text{Dropout}(E) \mid \text{Mol} \in D) = 0$$
4. The likelihood of a set of players D dropping out, where at least 1 of the players in set D used a *vrijstelling*, is equal to 0 for any *executie* E, i.e.
$$\mathcal{P}(D = \text{Dropout}(E) \mid D\ \cap\ \text{Vrijstellingen}(E) \neq \emptyset) = 0$$
where Vrijstellingen(E) is the set of all players that used a *vrijstelling* in the *test* corresponding to *executie* E.

In case D does not include the *mol* and does not include any player that used a *vrijstelling*, we estimate his/her likelihood of dropping out for every $p \in D$ using a Logistic Regression (see Section 2.3.4) and multiply them together[22], i.e.

$$\mathcal{P}(D = \text{Dropout}(E) \mid \text{Mol} \notin D) = \alpha \cdot \prod_{p \in D} \sigma(w^T x_p)$$

where α is a shared constant that ensures the likelihoods of all possible combinations (of same length) of dropouts sum up to 1, w are the trained weights and x_p is the feature encoding for player p, which is defined as:

[21] So this includes question Q' as well, meaning that #answers$(Q) \geq 1$.

[22] Only episode results are taken into account with non-*voluntary* dropouts, thus D is never an empty set.

Less Jokers The number of other players than p, which used less jokers than p.

Equal Jokers The number of other players than p, which used a same number of jokers as p.

More Jokers The number of other players than p, which used more jokers than p.

Bias A bias term, which is always equal to 1.

and these features are used directly. Thus we do not discretize these features, filter features out with Mann-Whitney U Selection or use PCA. Furthermore we also do not use any train multipliers.[23] All of these methods are not needed, because there are less features and the model is simpler than the Exam Drop Layer.[24] So based on this feature encoding in combination with a Logistic Regression model or the predefined rules we can determine

$$\mathcal{P}(D = \text{Dropout}(E) \mid p = \text{Mol})$$

for any possible *mol* p. And these likelihoods can be used in combination with Naïve Bayes model to predict the likelihood of being the *mol* as:

$$\mathcal{P}(p = \text{Mol} \mid D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n)) =$$
$$\frac{\mathcal{P}(D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n) \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol})}{\sum_{p'} \mathcal{P}(D_1 = \text{Dropout}(E_1), \ldots, D_n = \text{Dropout}(E_n) \mid p' = \text{Mol}) \cdot \mathcal{P}(p' = \text{Mol})} =$$
$$\frac{\mathcal{P}(p = \text{Mol}) \cdot \prod_{i=1}^{n} \mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p = \text{Mol})}{\sum_{p'} \mathcal{P}(p' = \text{Mol}) \cdot \prod_{i=1}^{n} \mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p = \text{Mol})}$$

where $\sum_{p'}$ is the sum over all player p' that could possibly be the *mol*, $E_1, \ldots E_n$ are all *executies* so far with respectively the non-*voluntary* non-empty set of dropouts $D_1, \ldots, D_n$. An example of this is given below:

Example 5. *Suppose we have a season with 6 players Jeroen, Nikkie, Patrick, Peggy, Ron and Tygo of which Jeroen is the* mol. *And we have seen 2* executies *in this season:*

1. *In the first* executie E_1 *we had 2 dropouts Patrick and Ron. Moreover Nikkie and Tygo used a* vrijstelling *in the* test *corresponding to* executie E_1*.*
2. *In the second* executie E_2 *Peggy dropped out. Furthermore Tygo used a* joker *in the* test *corresponding to* executie E_2*.*

Hence the only potential mol *players are Jeroen, Nikkie and Tygo. Moreover in the first* executie E_1 *the possible combinations of dropouts could have been: Jeroen & Patrick, Jeroen & Peggy, Jeroen & Ron, Patrick & Peggy, Patrick & Ron, Peggy & Ron. And in the second* executie E_2 *the possible dropouts could have been: Jeroen, Patrick, Peggy, Tygo. Since no* jokers *have been used in the* test *corresponding to* executie E_1*, all of the mentioned combinations have the same likelihood of dropping out, excluding the ones which includes the* mol*. Thus:*

$$\mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Jeroen} = \textit{Mol}) = \frac{1}{3}$$

because if we exclude all dropout combinations with Jeroen included in it, we have 3 combinations left. On the other hand we have:

$$\mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Nikkie} = \textit{Mol}) =$$
$$\mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Tygo} = \textit{Mol}) = \frac{1}{6}$$

[23]Which is similar as using a train multiplier of 1 for all data.

[24]But unfortunately the Exam Pass Layer has also a lot weaker prediction than the Exam Drop Layer.

since all 6 combinations are still possible. For the second executie E_2 *we get the following feature encoding:*

$$x_{\textit{Jeroen}} = x_{\textit{Nikkie}} = x_{\textit{Peggy}} = (0, 2, 1, 1)^T \qquad x_{\textit{Tygo}} = (3, 0, 0, 1)^T$$

and if we assume that $w^T \approx (-0.6045, -0.2297, -0.0346, -0.3252)$ *after training the Logistic Regression gives the following estimated dropout probabilities:*

$$\sigma(w^T x_{\textit{Jeroen}}) = \sigma(w^T x_{\textit{Nikkie}}) = \sigma(w^T x_{\textit{Peggy}}) \approx 0.3059 \qquad \sigma(w^T x_{\textit{Tygo}}) \approx 0.1054$$

Hence the estimated probability of Peggy dropping out in the second executie E_2 *is the following for these* potential mol *players:*

$$\begin{gathered}\mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Jeroen} = \textit{Mol}) = \\ \mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Nikkie} = \textit{Mol}) \approx \\ \frac{0.3059}{0.3059 + 0.3059 + 0.1054} \approx 0.4265\end{gathered}$$

$$\begin{gathered}\mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Tygo} = \textit{Mol}) \approx \\ \frac{0.3059}{0.3059 + 0.3059 + 0.3059} \approx 0.3333\end{gathered}$$

So finally the mol *likelihood for all players can be computed. In case of Jeroen, this likelihood is computed as:*

$$\begin{gathered}\mathcal{P}(\textit{Jeroen} = \textit{Mol} \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1), \{\textit{Peggy}\} = \textit{Dropout}(E_2)) = \\ \frac{\begin{gathered}\mathcal{P}(\textit{Jeroen} = \textit{Mol}) \cdot \mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Jeroen} = \textit{Mol}) \cdot \\ \mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Jeroen} = \textit{Mol})\end{gathered}}{\begin{gathered}\mathcal{P}(\textit{Jeroen} = \textit{Mol}) \cdot \mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Jeroen} = \textit{Mol}) \cdot \\ \mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Jeroen} = \textit{Mol}) + \\ \mathcal{P}(\textit{Nikkie} = \textit{Mol}) \cdot \mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Nikkie} = \textit{Mol}) \cdot \\ \mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Nikkie} = \textit{Mol}) + \\ \mathcal{P}(\textit{Tygo} = \textit{Mol}) \cdot \mathcal{P}(\{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1) \mid \textit{Tygo} = \textit{Mol}) \cdot \\ \mathcal{P}(\{\textit{Peggy}\} = \textit{Dropout}(E_2) \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_1),\ \textit{Tygo} = \textit{Mol})\end{gathered}} \approx \\ \frac{\frac{1}{3} \cdot \frac{1}{3} \cdot 0.4265}{\frac{1}{3} \cdot \frac{1}{3} \cdot 0.4265 + \frac{1}{3} \cdot \frac{1}{6} \cdot 0.4265 + \frac{1}{3} \cdot \frac{1}{6} \cdot 0.3333} \approx 0.5289\end{gathered}$$

And for Nikkie and Tygo the likelihoods are similarly computed as:

$$\begin{gathered}\mathcal{P}(\textit{Nikkie} = \textit{Mol} \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_2), \{\textit{Peggy}\} = \textit{Dropout}(E_3)) \approx \\ \frac{\frac{1}{3} \cdot \frac{1}{6} \cdot 0.4265}{\frac{1}{3} \cdot \frac{1}{3} \cdot 0.4265 + \frac{1}{3} \cdot \frac{1}{6} \cdot 0.4265 + \frac{1}{3} \cdot \frac{1}{6} \cdot 0.3333} \approx 0.2644\end{gathered}$$

$$\begin{gathered}\mathcal{P}(\textit{Tygo} = \textit{Mol} \mid \{\textit{Patrick}, \textit{Ron}\} = \textit{Dropout}(E_2), \{\textit{Peggy}\} = \textit{Dropout}(E_3)) \approx \\ \frac{\frac{1}{3} \cdot \frac{1}{6} \cdot 0.3333}{\frac{1}{3} \cdot \frac{1}{3} \cdot 0.4265 + \frac{1}{3} \cdot \frac{1}{6} \cdot 0.4265 + \frac{1}{3} \cdot \frac{1}{6} \cdot 0.3333} \approx 0.2067\end{gathered}$$

3 WIKIPEDIA LAYER

3.1 Introduction

The Exam Layer is an approach with better predictions as time progresses, which implies that during the first episodes it is a less powerful prediction model. So there is more demand for a layer that already can make predictions during the early episodes. Of course it is hard to predict the *mol* if all players are still in the game, but it should definitely be possible to already exclude some players as *mol* during early episodes. Because when looking at the personality of the *mol* there seemed to be a bias. It is remarkable that the *mol* is often not very famous compared to other players. Also the *mol* was very often an actor or a television/radio show host, whereas a lot of sportsmen, singers, writers and comedians also participated in ’Wie is de Mol?’. Thus with these findings a new layer was created.

The reasoning behind this layer was obvious, but unfortunately it was not so easy to create this layer. How do you measure being ’famous’ and when does someone belong to a certain job group? Moreover there was a need for a source from which background information could be retrieved about the players. A source which already exists for a long time, so that early seasons can also be used as training data. Furthermore this source should have information about all players, not just a few of them. To the best of our knowledge Wikipedia is the only sources that satisfies these requirements. Plus Wikipedia has the advantage that one can go back in time to extract the Wikipedia page before the first episode of that season is broadcast. Although a clear disadvantage of Wikipedia is that everyone has access to it and can add false information as well. Moreover not all celebrities had a detailed Wikipedia page by 2009. So it is not possible to (fairly) evaluate the Wikipedia layer for seasons that were broadcast before 2009, i.e. 5, 6, 7 and 8.[1] Despite these disadvantages there were no other consistent and reliable sources that satisfied all these requirements. Thus Wikipedia was used as source to gather data for this layer. And therefore this new layer was named the Wikipedia Layer.

The Wikipedia Layer was the first built pre-layer. A pre-layer is a layer that can already be used before the first episode has been broadcast. This has the advantage that it can already do predictions for the first episodes. Therefore pre-layers play an important role in closing the gap between earlier and later predictions. However pre-layers often do not have a major contribution in the predictions for later episodes, which also holds for the Wikipedia layer. In the next section the previous approach of the Wikipedia layer is discussed. This approach manually links words to certain jobs and counts the number of words for that job for each player providing a vector of job scores for every player, which can be compared to one another using cosine similarity (a commonly used method in Data Science for vector comparison[19]). Similar players in the training data as a given player are then be used to predict whether that player is the *mol*. Unfortunately this approach was inaccurate, so in Section 3.3 an approach is discussed that uses improved natural language processing techniques, better feature encoding and dimensionality reduction techniques to exclude players as *mol* with an outlying score.

[1]However it is possible to use a later version of these Wikipedia pages as training data.

3.2 Previous Approach

The Wikipedia Layer is a challenging layer, because it has to deal with natural language processing. Words on Wikipedia pages need to be understood in order to determine the likelihood of being the *mol* for every player, which is generally hard for an algorithm to do. Therefore the problem is simplified by only linking words to a fixed group of jobs. These selected jobs, which commonly appeared among 'Wie is de Mol?' players, were: Comedian, DJ, Journalist, Musician, Radio Host, Radio Producer, Singer, Stage Actor, Stage Producer, Songwriter, Television Actor, Television Host, Television Producer, Voice Actor and Writer.[2] And to link the words to the fixed group of jobs, the following steps are applied first:

1. Replace all special characters in the Wikipedia page by spaces, which includes tabs and new lines as well.
2. Convert all capital letters in the Wikipedia page to lower case.
3. Tokenize the Wikipedia Page by splitting it on spaces, e.g. "this is an example sentence" becomes "this", "is", "an", "example", "sentence".

After these steps, a manual mapping is used to link the words to the jobs. This mapping was created by myself, by manually going through the most frequently occurring words and linking them to jobs. It is allowed in this mapping that words link to no job at all, e.g. words as 'de' and 'het'. Likewise it is allowed to link a word to multiple jobs, e.g. 'televisie' is linked to television actor, television host and television producer. With this mapping between tokenized words and jobs, we can count the number of words that are linked to an arbitrary job. Finally these counts are used as features to encode every player, i.e. a player i is encoded as a vector of counts $x^i = (c^i_1, c^i_2, \ldots, c^i_{14})$.[3] To predict the likelihood y of being the *mol* for a given player encoded as $x = (c_1, c_2, \ldots, c_{14})$, a weighted cosine similarity sum over all training data is used, i.e.[4]

$$y = \frac{\sum_i \text{cossim}(x, x^i)^2 \cdot y^i}{\sum_i \text{cossim}(x, x^i)^2} \quad \text{with} \quad \text{cossim}(x, x^i) = \frac{\sum_j x_j \cdot x^i_j}{\sqrt{\sum_j (x_j)^2} \cdot \sqrt{\sum_j (x^i_j)^2}}$$

where y^i is 1 if the ith train player was the *mol* and 0 if that player was not the *mol*. And the cosine similarity score of x' and x^+ is a value between 1 and -1[5] for which it holds that:

- The closer it is to 1 the more similar the points are, i.e. both points are aligned in the same direction.
- The closer it is to 0 the more the points are unrelated, i.e. both points are aligned in orthogonal direction.
- The closer it is to -1 the more the points are opposites of each other, i.e. both points are aligned in opposite direction.

However negative values do not occur, since all feature are word counts which are positive values. Thus cossim(x, x^i) returns a score between 0 and 1, where a larger value means a higher similarity. Hence similar points as x get a larger weight in the sum and therefore have more influence on the outcome of y. Moreover because a weighted sum is taken the outcome

[2]Also a lot of athletes have participated in 'Wie is de Mol?', but it is hard to classify someone as athlete. Words related to athletes are very diverse, thus athlete was not included as job. Moreover Movie Actors and Television Actors are considered to be the same. Likewise Movie Producers are considered to be the same as Television Producers.

[3]If someone appears in multiple seasons then it is treated as a different player in both seasons.

[4]An exception to this rule is described in the Exception Handling appendix at A.2.1.

[5]Assuming that x' and x do not only consist of zeros.

of y is always a value between 0 and 1, which means that the predictions can be interpreted as likelihood of being the *mol*. But of course the likelihoods of every player needs to be *normalized* to obtain the final predictions.

This first implementation of the Wikipedia Layer was a good starting point, but unfortunately these predictions were inaccurate, see appendix C. The likelihood of non-*mol* was often way higher than the likelihood of *mol* players. A cause for these inaccurate predictions could be that there is no relationship between job and being the *mol*. However there are also some issues with this implementation which could have caused these inaccurate predictions:

- The natural language processing techniques used are quite simple. More advanced natural language processing techniques could be used to reduce similar words to the same form or recognize the same form in similar words, which is needed because it is impossible to classify all possible words manually. For example a plural Dutch word as ”films” should be reduced to the word ”film” or should be recognized to be similar as the word ”film”. By deciding not to do so, any form of any word has to be manually classified, which is nearly impossible.
- The encoding of the features as absolute counts of words related to jobs might not be appropriate. There are several issues with this encoding, e.g.
  - It does not take into account the length of their Wikipedia pages. If you have more words in your Wikipedia page you are also expected to have higher job counts. Moreover there could also be a relationship between the number of words in your Wikipedia page and being the *mol*, which is currently ignored.
  - Words related to more jobs have the same importance as words related to fewer jobs which is questionable. One could argue that words related to more jobs should be weighted less, because they are less informative.
  - A normal scale is currently used for the job counts which is questionable as well. For example is someone with a ’journalist’ count of 80 twice as much ’journalist’ as someone with a ’journalist’ count of 40?
  - Vectors of only zeros are ignored in the current approach and players encoded as a vector of only zeros have a likelihood of zero of being the *mol*. For seasons up to the last season this hypothesis has indeed hold, but it is doubtful whether it holds for future seasons as well.
- There are way too many features in this model, which is inappropriate when using cosine similarity. All jobs are treated as equally important, whereas some jobs occur more frequently than other jobs. Thus players might be similar based on the most occurring jobs, but still have a low cosine similarity score, since they are not similar based on less occurring jobs. Also this implementation does not take into account that a lot of jobs are related to each other, e.g. radio host is similar to DJ and stage actor is similar to television actor.
- There are unfortunately no theoretical grounds for this model. The closest model this implementation is related to is a nearest neighbor model [20, section 2.3.2], however the cosine similarity score is not really a distance metric. Thus it is arguable whether the outcome could be interpreted as a likelihood. Also there is no clear choice which power should be used to strengthen or weaken the cosine similarity score. Currently a power of 2 is used, i.e. cossim$(x, x^i)^2$, but a power of 1, 3 or 0.5 could be used as well.

3.3 Current Approach

So there are a lot of issues that have to be filtered out, before concluding whether there is actually a relationship between a Wikipedia page and the likelihood of being the *mol*. Each of these issues is solved in the next subsections. It starts by solving the natural language processing issues, after which the problem with feature encoding is fixed and finally the feature selection is improved together with the prediction model for the Wikipedia Layer.

3.3.1 Natural Language Processing

The natural language processing in the previous approach started with replacing special characters by spaces, converting all letters to lower case and finally tokenizing the Wikipedia pages based on spaces. We stick to this approach, beside that by doing so one discards some valuable information. For example Loretta Schrijver[6] is suddenly classified as 'Writer', because her last name Schrijver occurs a lot in her Wikipedia page, which is the Dutch word for writer. And by discarding capital letters, it becomes harder to distinguish names from words. Nevertheless this is a single case where capital letters provide crucial information, so this error is tolerated, because converting every letter to lowercase simplifies the manual classification process a lot. Moreover we should keep in mind that there does not exists any natural language processing technique which works for all cases. Thus the pros and cons have to be weighted for each technique to decide which technique is the best for our case. This also becomes especially clear in the following case, i.e. stemming versus subword extraction. As explained in previous section it is impossible to manually classify any form of any word. One technique that could be used to overcome this problem is stemming [21], which reduces similar words to the same smaller form that is called the 'stem'. For example the (Dutch) Snowball Stemmer, a commonly used stemming algorithm, maps the following Wikipedia words to these stem forms:

Full Word	Stem Form	Full Word	Stem Form	Full Word	Stem Form
program	program	programma	programma	televisieprogramma	televisieprogramma
presentator	presentator	presenteer	presenter	presenteren	presenter
presentatrice	presentatric	presenteert	presenteert	presenteerde	presenteerd
speel	spel	speelde	speeld	spelen	spel
spelers	speler	speler	speler	gespeeld	gespeeld
film	film	films	film	speelfilm	speelfilm
single	singl	singles	singles	hitsingle	hitsingl
zang	zang	zanger	zanger	zangeres	zangeres

Table 3.1: Word to 'stem' examples

Based on these examples, we notice that stemming is able to reduce some words to a similar form. For example the Dutch words 'presenteren' and 'presenteer' are mapped to the same form 'presenter'. Likewise 'films' and 'film' are mapped to the same form 'film'. However in most cases it does not recognize the relationship between similar words. For example the Dutch words 'presentator', 'presenteert', 'presenteerde', 'presentatrice' are also related to the word 'presenteer', but are not mapped to a similar form. So it becomes clear from this given example that stemming has a high false negative rate, i.e. a lot of similar words are not recognized as similar words. Consequently these examples show that there are no non-similar words which are reduced to the same form. Hence the false positive rate of stemming is very low, which is a clear advantage of stemming. In the next technique that is discussed, subword extraction, an

[6]Loretta Schrijver is a player that participated in season 10 of 'Wie is de Mol?'. Her last name translated to English is Writer.

opposite trend can be seen, i.e. this method has a higher false positive rate, but a lower false negative rate.

Subword extraction is a method that looks at all substrings w' of a word w which are words in itself recognized by the Dutch dictionary and it is defined that w is a subword of w even if it is not recognized by the Dutch dictionary. Subword extraction could therefore be more powerful than stemming, because it knows how to deal with composite words and recognizes more conjugations of the same word than stemming. However it causes a lot of issues, since small words w' could accidentally be included in a larger word w without being related to that larger word w. To prevent this as much as possible only subwords w' of at least length 4 are considered and if the length of word w is smaller than 4 then w is the only considered subword of w. Examples of subword extraction for Dutch words is shown below:

Full Word	**All Sub Words**
televisieprogramma	televisieprogramma, televisieprogram, visieprogramma, visieprogram, programma, televisie, program, visie, gram
presentator	presentator, present, pres
presenteert	presenteert, presenteer, presente, present, teert, eert, pres, teer
presenteerde	presenteerde, presenteer, presente, present, teerde, eerde, pres, teer
gespeeld	gespeeld, gespeel, speel, gesp, peel
speler	speler, spel
speelfilms	speelfilms, speelfilm, films, speel, film, peel
hitsingles	hitsingles, hitsingle, singles, single, hits
zangeres	zangeres, zanger, zang

Table 3.2: Subword extraction examples

What becomes clear from this example is that subword extraction is able to recognize more words as similar. For example the subword 'presenteer' is recognized inside the words 'presenteert' and 'presenteerde'. Likewise the word 'film' is recognized inside the word 'speelfilms'. These are examples of cases where stemming failed to recognize the similarity, but where subword extraction recognizes the similarity. However subword extraction unfortunately also recognizes similarities between different words. For example the Dutch word 'eerde' has nothing to do with the word 'presenteerde'.[7] And even if a smaller word is not accidentally included in a larger word, it does not mean that there is immediately a relationship between both words. For example one could argue that the Dutch child television program 'Fabeltjeskrant' has less to do with the Dutch word 'krant'.[8] Hence subword extraction has a much higher false positive rate than stemming, but a lower false negative rate on the other hand. Thus neither of these techniques outperforms the other, i.e. there are always cases where one of the techniques performs better. Therefore it has to be decided whether a low false negative rate is preferred or a low false positive rate. In this case a lower false negative rate is preferred, because Wikipedia pages consists of many words. And as long as most of these words are correctly mapped then we still have reliable results. Whereas with a higher false negative rate, the similarity between most words is not recognized. Hence the results depends only on a few recognizable words in the Wikipedia page, which gives far less reliable results. Thus subword extraction is the preferred choice, which extract a set of subwords W out of every word/token w. The following rules are then applied to determine as which job j_i word w should be classified:

1. Define $W(j)$ as all words (manually selected) that correspond to job j and define J as the jobs that are still in the race, which is initially defined as all jobs $J = \{j_1, \ldots, j_n\}$.

[7]'eerde' is the Dutch word for honouring and 'presenteerde' is the Dutch word for presenting.
[8]'krant' is the Dutch word for newspaper.

2. If no word is shared between any job $W(j)$ and W, i.e. $\forall_{j \in J}\, W(j) \cap W = \emptyset$, then we stop immediately, define $J = \emptyset$ and conclude that w does not belong to any job j. Otherwise we continue with the next step.

3. Remove all jobs j from J which share less words $W(j)$ with the set of subwords W as any other given job $j' \in J$. Formally spoken J is redefined as:

$$J_{\text{new}} = \{j \in J_{\text{old}} : m = |W \cap W(j)|\} \quad \text{with} \quad m = \max_{j' \in J} |W \cap W(j')|$$

   where $|S|$ is defined as the number of words in set S. This rule is applied, because a word is more related to a certain job if it shares more subwords.

4. Remove all jobs j from J of which the longest shared word w' between $W(j)$ and W is smaller than the longest shared word w^* between $W(j')$ and W for any other given job $j' \in J$. Formally spoken J is redefined as:

$$J_{\text{new}} = \{j \in J_{\text{old}} : l = L(W \cap W(j))\} \quad \text{with} \quad l = \max_{j' \in J} L(W \cap W(j')) \quad \text{and} \quad L(S) = \max_{w' \in S} |w'|$$

   where $|w'|$ is defined as the length of word w'. This rule is applied, because a larger subword carries more information and thus indicates better to which job this word corresponds.

5. The initial word w from which the subwords W are extracted is now classified to belong to any job j in the set of jobs left J.

An example of these classification rules is given below:

Example 6. *Suppose there are four jobs: 'Journalist', 'Radio Host', 'Television Host', 'Voice Actor'. With the following manual mapping:*

Job	Coresponding Words
Journalist	journaal, journalist, krant, nieuws
Radio Host	nieuws, produce, programma, radio
Television Host	nieuws, programma, televisie
Voice Actor	acteur, fabeltjeskrant, stem

Table 3.3: Subword extraction examples

If these rules are applied for the word 'fabeltjeskrantjes' with extracted subwords W_1:

'fabeltjeskrantjes', 'fabeltjeskrantje', 'fabeltjeskrant', 'fabeltjes', 'fabeltje', 'krantjes', 'krantje', 'fabelt', 'fabel', 'krant', 'abel', 'belt'

then after step 3 the possible jobs left are 'Journalist' and 'Voice Actor', since they both share 1 word with W_1 (respectively 'krant' and 'fabeltjeskrant') which is the highest number of shared words. However after step 4. 'Journalist' is removed as option, because 'fabeltjeskrant' is a longer word than 'krant'. So 'fabeltjeskrantjes' is classified to belong only to 'Voice Actor'. Moreover if the rules are applied for the word 'nieuwsprogramma' with extracted subwords W_2:

'nieuwsprogramma', 'nieuwsprogram', 'programma', 'program', 'nieuws', 'nieuw', 'gram'

then after step 3. the possible jobs left are 'Radio Host' and 'Television Host', because they both share two words with w_2 (which are 'nieuws' and 'programma') which is the highest number of shared words. Note that 'Journalist' is not included, since only one word is shared with W_2 (which is 'nieuws'). After step 4 none of these jobs is removed, because 'programma' is the longest word for both jobs which has the same length as itself. So 'nieuwsprogramma' is classified to belong to both 'Radio Host' and 'Television Host' as jobs.

With these classification rules, we can keep track of the frequency of jobs for every player p using a job counter $C_p(j)$. The job counter is initially defined as $C_p(j) = 0$ for all jobs j. And this counter is updated by looping over all words w in the Wikipedia page by increasing the job counter $C_p(j)$ by $\frac{1}{|J|}$ for all jobs $j \in J$ that belong to word w. This is a major difference with the previous approach where the job counter $C_p(j)$ was increased by 1 for all jobs $j \in J$ instead, because it weighs words that correspond to more jobs less. For example 'nieuwsprogramma' increases the job counter for 'Television Host' and 'Radio Host' both by 1/2, whereas 'fabeltjeskrantjes' increases the job counter of only 'Voice Actor' by 1. Furthermore note that this rule ensures that the sum of job counter increments is always equal to 1 for every word w that corresponds to at least one job, which is a nice property of this weighting.

3.3.2 Feature Encoding

Also regarding the feature encoding there are some major changes, i.e. the job counts are not directly used as features. Instead relative features are used with respect to the total number of words and with respect to the occurrence of this job among other players of the same season. To make this happen we first define the following terms:

- S_p is the set of players that participated in the same season as player p.
- P is the set of all players (which participated in any season).
- T_p is the total number of recognized words in the Wikipedia page of player p, which is equal to the sum of all job counts of player p, i.e. $T_p = \sum_i C_p(j_i)$.
- $C'_p(j_i)$ is the feature representation for the job count of player p and job j_i, i.e. $C'_p(j_i)$ is the feature representation of $C_p(j_i)$. Initially $C'_p(j_i)$ is defined to be equal to $C_p(j_i)$.
- T'_p is the feature representation for T_p.

Secondly, the job features are changed into relative features with respect to the total number of words, since players with more words in their Wikipedia page are expected to have more words that are linked to jobs as well. More specifically if a player has m times more words in his/her Wikipedia page then he/she is expected to have m times more words linked to every job. So the job feature representation $C'_p(j_i)$ is updated with the following rule for every player p and job j_i:[9]

$$C'_p(j_i) \leftarrow \frac{C'_p(j_i)}{T_p}$$

which ensures that a player with m times more words in his/her Wikipedia page and m times more words linked to a job j_i have the same score. Thirdly, the job features are adjusted such that they are relative with regard to the occurrence of the job among other players of the same season, because if a job occurs more in a particular season then it is less special if a given player p belongs to that job. Thus $C'_p(j_i)$ is simultaneously transformed for every player p and every job j_i into:[10]

$$C'_p(j_i) \leftarrow \frac{C'_p(j_i)}{\sum_{p^+ \in S_p} C'_{p^+}(j_i)}$$

Furthermore, because differences are expected to be better recognizable at a multiplicative scale rather than an additive scale, a logarithmic transformation is applied, i.e.[11]

$$C'_p(j_i) \leftarrow \ln(C'_p(j_i))$$

[9]An exception to this rule is described in the Exception Handling appendix at A.2.2.

[10]We assume that $\sum_{p^+ \in S_p} C'_{p^+}(j_i) > 0$. Although it does happen sometimes that $C'_{p^+}(J_i) = 0$ for a given player p^+ and job j_i, it is very unlikely that this is the case for all 10 players in a given season for any given job.

[11]An exception to this rule is described in the Exception Handling appendix at A.2.3.

For example a relative value of 0.6 is expected to be more similar to 0.9 than to 0.3, which is solved by applying a logarithmic transformation. Moreover the logarithmic transformation is also justified, because the frequency distribution how often job counter values occur is very skewed, e.g. Figure 3.1 contains the job counter frequencies for the job 'Singer' of seasons 5 up to *21*.

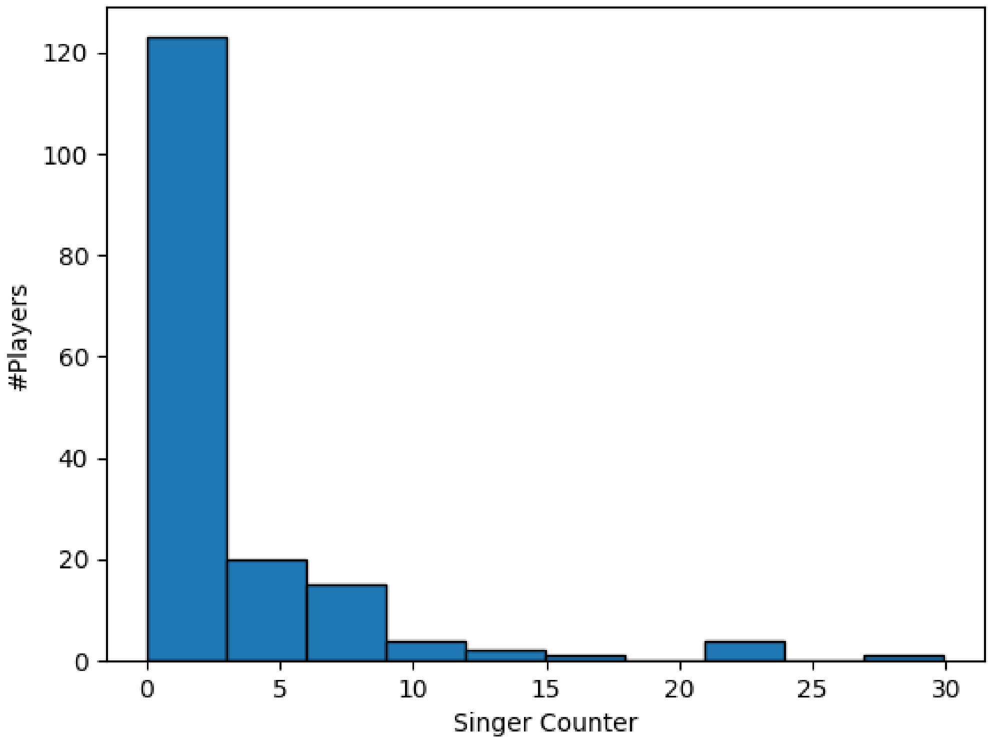


Figure 3.1: Histogram of the 'Singer' job counter (seasons 5 up to *21*)

After this logarithmic transformation, we have:

$$C'_p(j_i) = \ln\left(\frac{C_p(j_i)/T_p}{\sum_{p^+ \in S_p} C_{p^+}(j_i)/T_{p^+}}\right)$$

The last transformation is a discretization which is applied, because a player either belongs to a job or not. For this discretization it is assumed that the job features of each job follows a separate Gaussian Mixture distribution [14, section 9.2] with two clusters, i.e.

$$C'_p(j_i) \sim C_1 + C_2 \quad \text{with} \quad C_1 = \pi_{i,1} \cdot \mathcal{N}(\mu_{i,1}, \sigma_{i,1}) \quad \text{and} \quad C_2 = \pi_{i,2} \cdot \mathcal{N}(\mu_{i,2}, \sigma_{i,2})$$

where the parameters $\pi_{i,1}, \pi_{i,2}, \mu_{i,1}, \sigma_{i,1}, \mu_{i,2}, \sigma_{i,2}$ for each job j_i can be estimated using the EM-algorithm [14, section 9.2.2]. A Gaussian Mixture model is more appropriate here than a K-means clustering approach [14, section 9.1], because K-means clustering cannot handle different variances in groups which is expected to be the case here, since player that do not belong to a job j_i have more similar $C'_p(j_i)$ scores than player that do belong to the job j_i. Thus with the Gaussian Mixture model we update $C'_p(j_i) \leftarrow 0$ for each player p and job j_i if

$$\rho(C_1 = C'_p(j_i)) > \rho(C_2 = C'_p(j_i))$$

and else we update $C'_p(J_i) \leftarrow 1$, where ρ is the probability density function. In other words if $C'_p(j_i)$ belongs to the first cluster, $C'_p(j_i)$ is changed to 0 and if it belongs to the second cluster, its value is changed to 1.

For the total number of words, similarly it is made relative with respect to the total number of words of other players in the same season for every player p by:[12]

$$T'_p \leftarrow \frac{T'_p}{\sum_{p^+ \in S_p} T'_{p^+}}$$

since in earlier seasons Wikipedia pages were less detailed. So having more words in your Wikipedia page in earlier seasons was more special than having more words in your Wikipedia page in later seasons. Furthermore a logarithmic transformation is applied on the total number of word features for every player p by:[13]

$$T'_p \leftarrow \ln(T'_p)$$

because once again we expect differences to be better recognizable at a multiplicative scale rather than an additive scale. Moreover the frequency distribution of word counts is also skewed, which again justifies the logarithmic transformation, see Figure 3.2.

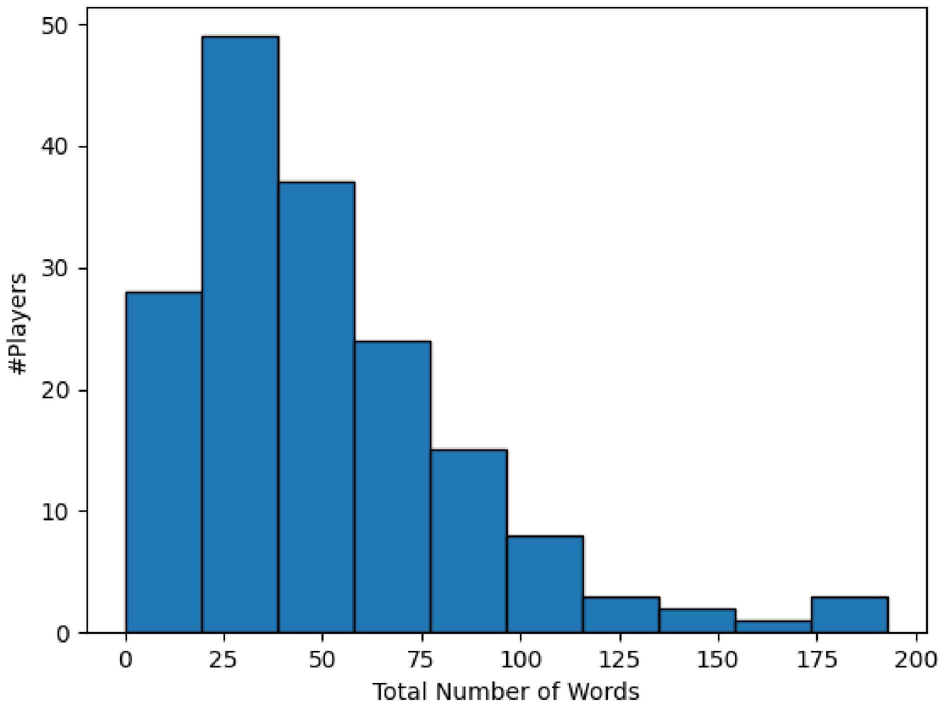


Figure 3.2: Histogram of the Total Number of Words (seasons 5 up to *21*)

The last transformation that is applied on the total number of words feature is a polynomial transformation up to the second degree, i.e.

$$T'_p \leftarrow (T'_p, {T'_p}^2)$$

to ensure that the Wikipedia Layer recognizes patterns where the standard deviation of the total number of words for *mol* players is smaller than for non-*mol* players. And after all these changes we end up with:

$$T'_p = (\ln\left(\frac{T_p}{\sum_{p^+ \in S_p} T_{p^+}}\right), \ln\left(\frac{T_p}{\sum_{p^+ \in S_p} T_{p^+}}\right)^2)$$

[12]We assume that $\sum_{p^+ \in S_p} T'_{p^+} > 0$, because it rarely occurs that $T'_{p^+} = 0$ for some player p^+. Let alone that this is the case for all players in a given season.

[13]An exception to this rule is described in the Exception Handling appendix at A.2.4.

3.3.3 Feature Reduction

The approach does not address yet the issue of co-correlation between many of the job features $C'_p(j_i)$ and it also does not take into consideration how often jobs occur. For example if all or none of the players in the training data belongs to a certain job then that job does not provide any information. Therefore to deal with these issues, Principal Component Analysis [14, section 12.1] is applied on all the job features $C'_p(j_i)$ which extracts the 5 largest components out of all 15 job features.[14] The benefit of applying Principal Component Analysis is that it reduces the number of features unsupervised, which is the reason why this technique is not prone to overfitting. On the other hand if Principal Component Analysis was not applied, then there are 15 job features and 2 word count features left for a data set of 170 data points (with season 5 up to *21* as training data) of which 17 are classified as *mol*. And applying a machine learning model on a data set with an equal number of features as data points belonging to the smallest class is guaranteed to overfit. To illustrate this, almost any system of n linear equations with n variables has a unique solution, even if the coefficients of the linear equations are randomly sampled. Hence finding a solution to this system of linear equations does not imply a relationship between any of the variables and the target outcome. Thus applying Principal Component Analysis is crucial in the prevention of overfitting.

The result of applying Principal Component Analysis is 5 reduced job features. These 5 reduced job features are combined with the 2 word count features, which results in 7 features in total. Based on these 7 features the *mol* and non-*mol* data points are separated for which Fisher's Linear Discriminant [14, section 4.1.4] is used. This technique separates both classes supervised using a linear projection. More specifically, every feature vector x with this technique is mapped to a one dimensional feature vector $y = w^T x$, where w is fixed weight vector used to separate both classes as much as possible. To do so, Fisher's Linear Discriminant [14, section 4.1.4] determines the mean vector for both classes:

$$\mu_1 = \frac{1}{|\text{non-Mol}|} \sum_{x \in \text{non-Mol}} x \qquad \mu_2 = \frac{1}{|\text{Mol}|} \sum_{x \in \text{Mol}} x$$

with non-Mol the set of all non-*mol* data points and Mol the set of all *mol* data points. Moreover with regard to these means μ_1 and μ_2, we select w such that:

$$w = \max_{w} w \cdot (\mu_2 - \mu_1) \quad \text{s.t.} \quad |w| = 1 \quad \text{with} \quad |w| = \sum_{i=1}^{7} w_i^2$$

since this w is the maximal separating unit vector. And according to section 4.1.4 of the book 'Pattern Recognition and Machine Learning' [14, section 4.1.4] this w is proportional to:

$$w \propto S^{-1}(\mu_2 - \mu_1) \quad \text{with} \quad S = \sum_{x \in \text{non-Mol}} (x - \mu_1)(x - \mu_1)^T + \sum_{x \in \text{Mol}} (x - \mu_2)(x - \mu_2)^T$$

Fisher's Linear Discriminant helps us therefore to separate both classes. However being cautious when applying this technique is advised, because Fisher's Linear Discriminant is a supervised technique that has the risk to overfit. Especially, because the least common class has 17 data points and 7 features are used to linear separate them. Although the risk of overfitting with Fisher's Linear Discriminant is drastically reduced by applying Principal Component Analysis first. If Principal Component Analysis was not applied, then Fisher's Linear Discriminant had to linear separate both classes based on 17 features which was guaranteed to overfit.

[14]We achieve the best accuracy with around 5 principal components.

3.3.4 Classification Model

After applying Fisher's Linear Discriminant 10 y_p values are obtained (assuming that 10 players participated in the given season). For these y_p values it holds by orientation of the Fisher's Linear Discriminant that if y_p is larger it is more likely to correspond to a *mol* player and if y_p is smaller it is more likely to correspond to a non-*mol* player. To classify these y_p values a simple approach is used which determines a z-score z_p (also known as standard score) for every y_p value, which is defined as:

$$z_p = \frac{y_p - \mu}{\sigma} \qquad \text{with} \qquad \sigma = \sqrt{\frac{\sum_{i=1}^{10}(y_i - \mu)^2}{9}} \qquad \text{and} \qquad \mu = \frac{1}{10}\sum_{i=1}^{10} y_i$$

And based on these z-scores z_p it is determined whether a player is less likely to be the *mol* or not using a threshold t. If a player has a z-score below this threshold t then he/she is considered less likely to be the *mol* and above this threshold t the player is considered more likely to be the *mol*. The selected threshold here is -0.524, because my estimation is that around three players in every season are less likely to be the *mol* based on their personality and popularity. And a z-score of -0.524 corresponds to:

$$\mathcal{P}(\mathcal{N}(0,1) \leq -0.524) \approx \frac{3}{10}$$

which ensures that around three players in each season (with 10 players) are less likely to be the *mol*. The advantage by using z-scores is that the prediction of players dependent on each other. For example if halve of all players have a low y value then less players have a z-score below this threshold. This is also in line with our expectation, i.e. if there are more low y values it is an indication that the Wikipedia layer does not perform well on this particular season. For example if all players have a low y value, we are certain that the *mol* also has a low y value. So less players should be assigned a lower likelihood for this season.

Thus this technique divides all players in two groups, i.e. the less likely *mol* group and the more likely *mol* group. For both groups a separate likelihood is given. For the less likely *mol* group a likelihood of 0.079 of being the *mol* is given, since on average 7.9% of all players with a negative z-score (< 0) are the *mol*. And all players that belong to the more likely *mol* group get a likelihood of 0.117 of being the *mol*, because on average 11.7% of all players with a non-negative z-score (≥ 0) are the *mol*. After assigning the probabilities to all players, the non-*potential mol* players are assigned a likelihood of zero and the probabilities are *normalized*. Note that a different threshold is used to determine the likelihoods (i.e. 0) than the threshold to separate less likely players from more likely players (i.e. -0.524), because only few players had a z-score below -0.524. So using -0.524 as threshold to determine the likelihoods gives inaccurate estimations.

4 APPEARANCE LAYER

4.1 Introduction

Other players in the game show always seem to be more suspicious than the actual *mol* himself. When non-*mol* players fail an exercise, it is shown to the audience. Whereas if the *mol* fails an exercise, it is often not shown to the audience. It seems that the cast of ”Wie is de Mol?” tries to manipulate us by showing only selected parts of episodes. Another ”Wie is de Mol?”-analyst [8] noticed this as well and assumed that the *mol* appeared less during episodes than average. Hence he built a face recognition analyser for ”Wie is de Mol” that counts for every player the frames in which that player appears [8].[1] Using this tool he analysed the first four episodes of season 18, 19 & 20 and concluded that the *mol* appeared less. His hypothesis might be true, however you cannot draw this conclusion based on just three seasons. Thus after asking for permission to use his code, which was given, I started analysing more seasons.

The first ten seasons were not suitable for face recognition analysis, since the production and the video quality of those seasons was different than for season 11 and onwards. Moreover in season 11 and 12 the *mol* does not appear less than other players. Therefore only seasons starting from season 13 were analysed. For these seasons a relative appearance feature was computed per player and episode, which can be computed by the following formula:

$$A_{p,e} = \ln\left(\frac{FC_{p,e}}{\sum_{p'} FC_{p',e}} \cdot \#\text{players}_e + 10^{-4}\right) \tag{4.1}$$

where $FC_{p,e}$ is the frame count, i.e. the number of frames of episode e in which player p is detected by the face recognition tool.[2] Besides $\#\text{players}_e$ is the number of players that were *alive* in that episode. It is necessary to multiply $\#\text{players}_e$ with the relative frame count

$$\frac{FC_{p,e}}{\sum_{p'} FC_{p',e}}$$

because with less players you have a higher relative frame count per player. Thus without this multiplication, you cannot compare the appearances in different episodes with one another. Moreover a logarithm is applied, because we are interested in the differences of

$$\frac{FC_{p,e}}{\sum_{p'} FC_{p',e}} \cdot \#\text{players}_e$$

at multiplicative scale rather than additive scale. For example a score of 0.6 is more similar to 1.0 than to 0.2. Besides a logarithmic transformation is also justified, since the frequency distribution of raw appearance values (before applying logarithmic transformation) is skewed. The frequency distribution of raw appearance values for the first five episodes of seasons 13 up to *21* is shown by Figure 4.1.

[1]This tool was built using the image face recognition tool of Adam Geitgey [9].

[2]Due to performance issues it takes too much time to analyse every frame. Therefore only every frame that is a multiple of 10 was analysed for video formats with a framerate of 15 and 25 frames per second and for video formats with a framerate of 30 frames per second only every frame that is a multiple of 15 was analysed.

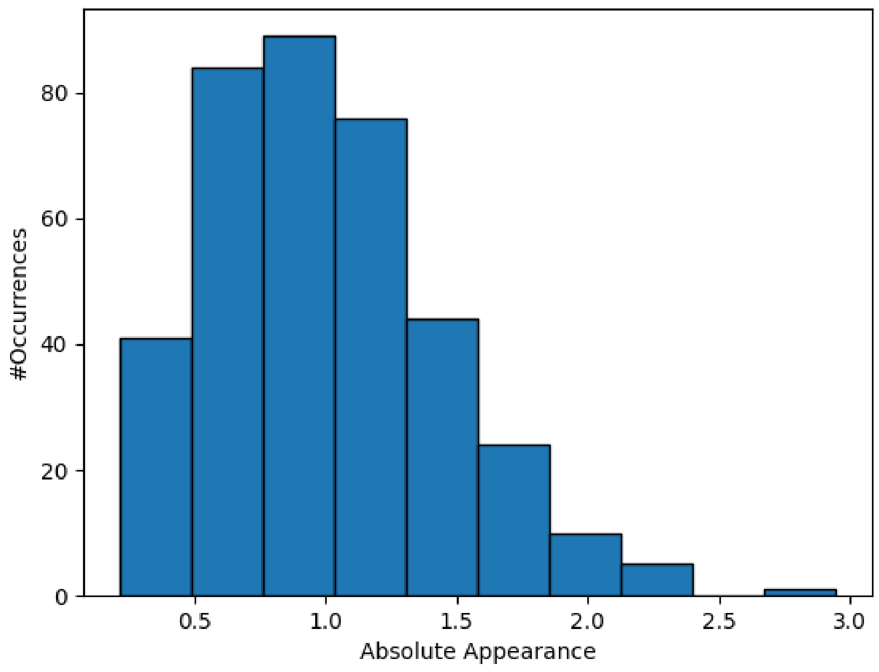


Figure 4.1: Raw Appearance values in the first 5 episodes of seasons 13 up to *21*

Furthermore a small number 10^{-4} is added before applying the logarithm, since the frame count $FC_{p,e}$ could be equal to 0 of which the logarithm is undefined, which is a drawback of logarithmic transformation. However an important benefit of the logarithm function is that it is strictly increasing, i.e. $x > y$ implies ln$(x) >$ ln(y). Thus if a player appears more in episodes then it also has a higher A value. So if the hypothesis of van Hoek [8] holds then the appearance value A for the *mol* is more biased towards lower values than the A value for non-*mol* players. A plot with the fully transformed appearance values A for all players of season 13 up to *21* for the first five episodes are shown by Figure 4.2.

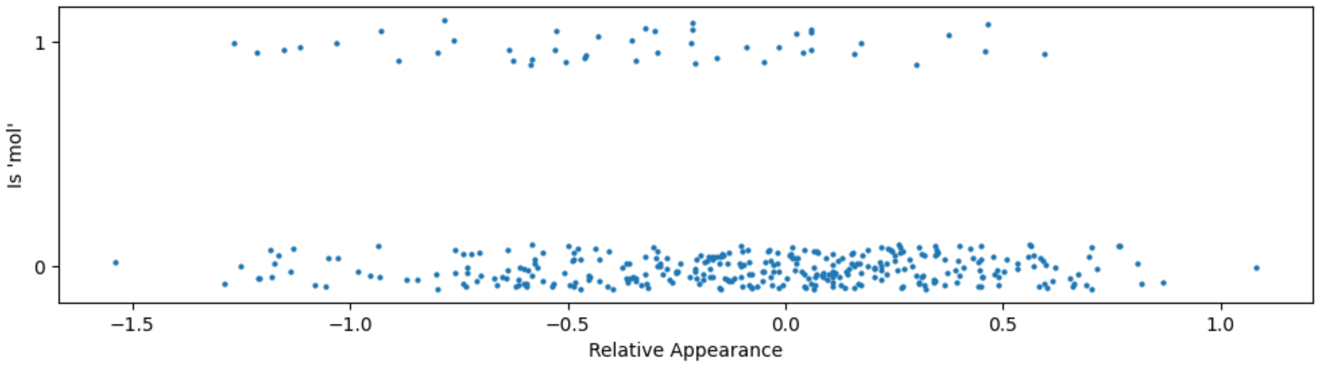


Figure 4.2: Appearance values in the first 5 episodes of seasons 13 up to *21*

The relative appearance values A are grouped based on whether that player was the *mol* or not. By looking at this scatter plot you might believe that the relative appearance value of non-*mol* and *mol* players follow the same probability distribution. On the other hand for non-*mol* players values between 0.0 and 0.5 occur more frequently than for *mol* players. But we are not sure about the probability distributions behind these appearance values. Hence it seems to be difficult to show that the appearance values of the *mol* is more biased towards lower values. Fortunately there is a statistical test for this, named the Mann-Whitney U Test (see Section 2.3.3). With this test it can be shown whether the appearance values of the *mol* is more biased towards lower values. For this test it is assumed that the *mol* appearance values X are independently and identically distributed. Likewise the non-*mol* appearance values Y are assumed to be independently and identically distributed.[3] The null hypothesis and alternative hypothesis for this test are respectively:

$$H_0 : \forall_{z \in \mathbb{R}}\ F_X(z) = F_Y(z) \quad H_1 : \forall_{z \in \mathbb{R}}\ F_X(z) < F_Y(z)$$

where $F_X(z)$ is the cumulative distribution function behind X and $F_Y(z)$ is the cumulative distribution function behind Y. The sample size for X was 45 and for Y it was 329, which results in a U value of 4992. And this U value corresponds to a Z-score of ≈ -3.543, resulting in a p-value of $\approx 1.977 \cdot 10^{-4}$. Hence the hypothesis of van Hoek holds, meaning that the *mol* appears significantly less than non-*mol* players. But how could one exploit this idea into a layer that predicts the *mol*? In Section 4.2 different attempts of the past are discussed to predict the *mol*. These approaches includes:

- Nearest Neighbor classifier, which is a non-parametric model that classifies appearance values based on similar appearance values.
- Gaussian naïve Bayes classifier, which is a parametric model that assumes appearance values follow a separate Gaussian distribution per class (*mol* and non-*mol*).
- A Split classifier, which classifies appearance values based on the interval it belongs to.

Furthermore in Section 4.3 the current approach, Kernel Density Estimation, is discussed. Also a data augmentation technique (how to increase the data size) and an outlier cutoff technique are discussed in this section.

4.2 Previous Approaches

4.2.1 Nearest Neighbor

The first implementation of the Appearance Layer was a Nearest Neighbor regressor model. The logic behind Nearest Neighbor is that a new unknown case should be classified as similar known cases in the train data set. For example in a K-Nearest Neighbor regressor model with train data points $(x^1, x^2, \ldots, x^n)$, where $x^i = (x^i_1, x^i_2, \ldots, x^i_m)$, and with corresponding train labels $(y^1, y^2, \ldots, y^n)$ the label y of an unknown point x is determined by:

$$y = \frac{1}{k} \sum_{x^i \in N_k(x)} y^i$$

where $N_k(x)$ are the closest k points to x [20, section 2.3.2] and y^i is in our case 1 if the corresponding player is the *mol* and 0 otherwise. To determine the closest points, one can use an arbitrary distance metric, e.g. Euclidean, Manhattan or Chebyshev distance. For this implementation a Manhattan distance was used, i.e.

$$d(x', x^+) = \sum_{i=1}^{m} |x'_i - x^+_i|$$

Moreover instead of the K-Nearest Neighbor regressor model, a Weighted Nearest Neighbor regressor model was applied to determine label y for point x, which takes a weighted sum over all labels based on the distance to x:

$$y = \frac{\sum_{i=1}^{n} w(x, x^i) \cdot y^i}{\sum_{i=1}^{n} w(x, x^i)} \quad \text{with} \quad w(x, x^i) = \frac{1}{1 + d(x, x^i)^2}$$

[3]The appearance values are unfortunately not fully independent, see appendix D.

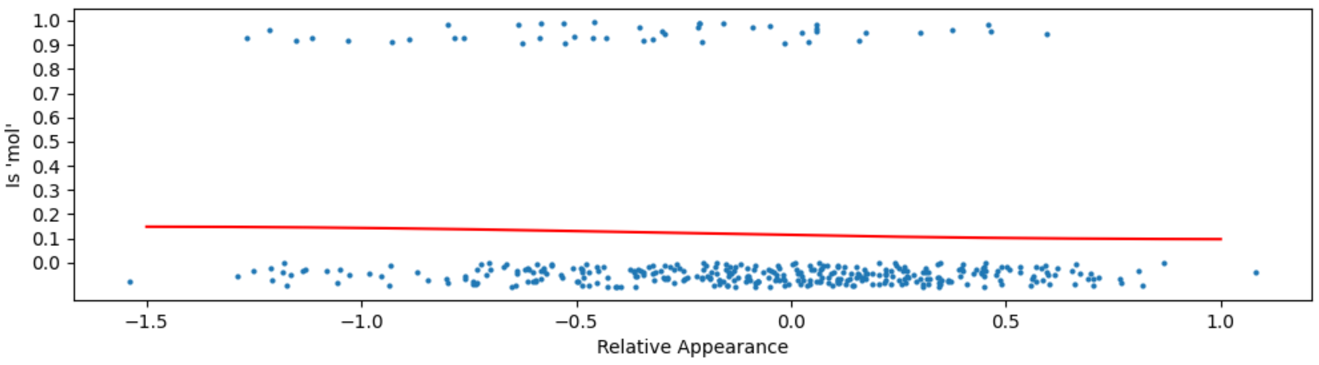


Figure 4.3: Weighted Nearest Neighbor Prediction Model

The outcome labels y for every point x are then *normalized* over all players, which makes them valid *mol* likelihoods as defined in Section 1.4. But how should the data point x be encoded? There are two common options for this:

1. Group all appearance values per player, where A_p are all appearance values for player p in a given season. In this case x is encoded as:

$$x = (Q(A_p, 0),\ Q(A_p, 1/4),\ Q(A_p, 2/4),\ Q(A_p, 3/4),\ Q(A_p, 1))$$

   where $Q(A_p, i)$ is the i relative quantile of set A_p.[4] Note that the quantile function is taken instead of sorting all appearance values, because then the dropouts from the first four episodes could also be included in the training data. Moreover the cases where the quantile i lies between 2 values $A_{(j)}$ and $A_{(k)}$, a linear interpolation between those values is taken. Choosing this option results in 81 non-*mol* training points and 9 *mol* training points with five features each if only data from season 13 up to *21* is used.

2. Treat all appearance values independently, which means that every appearance value for a single episode and a single player is a separated point x. All outcome likelihoods for the sample player are then naïvely aggregated (see Section 5.2.2). Choosing this option results in 329 non-*mol* training points and 45 *mol* training points with one feature each if only the data from season 13 up to *21* is used.

The second option is preferred over the first option, because with the second option the training data size is multiplied by 5 whereas the number of features is divided by 5. And since there is a shortage of training data, a larger data set and fewer features are preferred. And with a lower ratio

$$\frac{\text{\#features}}{\text{\#training data}}$$

the risk of overfitting decreases as well. Therefore the second option is selected as encoding for the nearest neighbor approach and further approaches.[5] But this encoding unfortunately did not have good results for the Nearest Neighbor approach (see Figure 4.3 and Figure 4.4), since Nearest Neighbor does not perform well if there is no clear distinction between classes. Nearest Neighbor either overfits on the train data, or underfits the pattern. In Figure 4.3 the prediction is shown for this Weighted Nearest Neighbor approach, where appearance values are treated independently. This prediction underfits, i.e. the likelihood (red line) is not much different for different appearance values. On the other hand in Figure 4.4 the prediction is shown of a 25-Nearest Neighbour model, which overfits a lot on the training data.

[4]By definition of the quantile function, $Q(A_p, 0)$ is the minimum of all appearance values and $Q(A_p, 1)$ is the maximum of all appearance values.

[5]However the second option is not fully justified, because there is some dependency between appearance values of the same player in different episodes, see Appendix D.

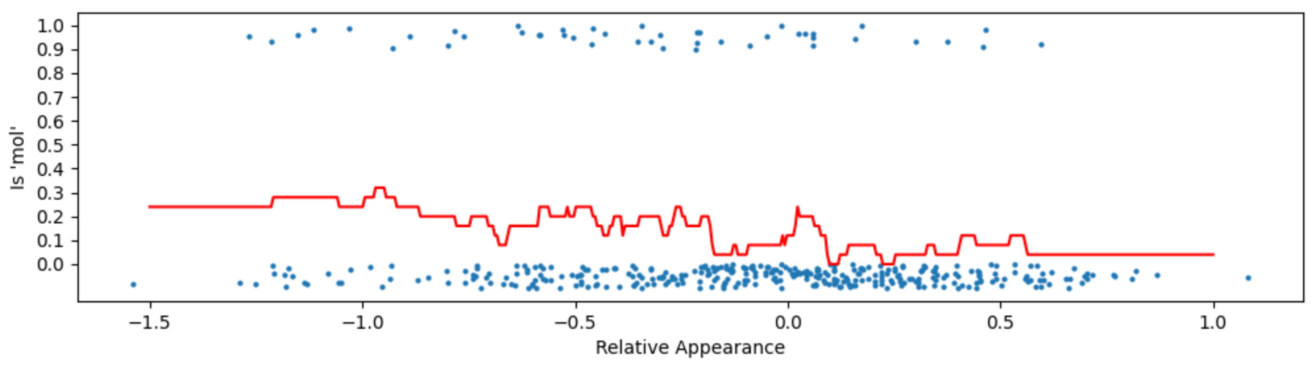


Figure 4.4: 25-Nearest Neighbor Prediction Model

4.2.2 Gaussian naïve Bayes

Nearest Neighbor was not the best implementation for the Appearance Layer, but it was a good starting point to illustrate that the model has to find the right balance between overfitting and underfitting. So the second implementation idea was a Gaussian naïve Bayes classifier [22, section 3.5]. In a Gaussian naïve Bayes classifier one determines the likelihood that player p is the *mol* given the appearance values by the following formula:

$$\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n) = \frac{\rho(A_1, \ldots, A_n \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol})}{\rho(A_1, \ldots, A_n \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol}) + \rho(A_1, \ldots, A_n \mid p \neq \text{Mol}) \cdot \mathcal{P}(p \neq \text{Mol})}$$

where $\mathcal{P}(p = \text{Mol})$ is determined by the relative amount of train data classified as *mol* and $\mathcal{P}(p \neq \text{Mol})$ is determined by the relative amount of train data classified as non-*mol*. For example for 374 training points of which 45 are classified as *mol*, we have $\mathcal{P}(p = \text{Mol}) = \frac{45}{374}$ and $\mathcal{P}(p \neq \text{Mol}) = \frac{329}{374}$. Moreover in a Gaussian naïve Bayes classifier it is assumed that the probability density functions $\rho(A_1, \ldots, A_n \mid p = \text{Mol})$ and $\rho(A_1, \ldots, A_n \mid p \neq \text{Mol})$ can be decomposed as:

$$\rho(A_1, \ldots, A_n \mid p = \text{Mol}) = \prod_{i=1}^{n} \rho(A_i \mid p = \text{Mol}) \quad \text{and} \quad \rho(A_1, \ldots, A_n \mid p \neq \text{Mol}) = \prod_{i=1}^{n} \rho(A_i \mid p \neq \text{Mol})$$

with $A_i \mid p = \text{Mol} \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $A_i \mid p \neq \text{Mol} \sim \mathcal{N}(\mu_2, \sigma_2^2)$ (normally distributed), where μ_1, μ_2, σ_1 and σ_2 are the parameters that are estimated using the training data. Hence after training, $\rho(A_i \mid p = \text{Mol})$ and $\rho(A_i \mid p \neq \text{Mol})$ can be determined for every appearance value A_i, which finally gives us $\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n)$ for every player p. But the likelihoods of all *potential mol* players might again not sum up to 1, i.e.

$$\sum_{p'} \mathcal{P}(p' = \text{Mol} \mid A_{p',1}, \ldots, A_{p',n}) \neq 1$$

because the Gaussian naïve Bayes classifier only ensures per player p that:

$$\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n) + \mathcal{P}(p \neq \text{Mol} \mid A_1, \ldots, A_n) = 1$$

Thus all likelihoods are *normalized* over all players. In Figure 4.5 the performance of this model is shown using the data of seasons 13 up to *21*. The red line is the density $\rho(A_i \mid p = \text{Mol})$, the green line is the density $\rho(A_i \mid p \neq \text{Mol})$ and the blue line is the posterior based on only a single appearance value $\mathcal{P}(p = \text{Mol} \mid A_i)$. Unfortunately this approach was inaccurate, since the *mol* appearance values are too evenly distributed to follow a Gaussian distribution. Neither do the *mol* appearance values seem to follow a Gaussian distribution, because the appearance values

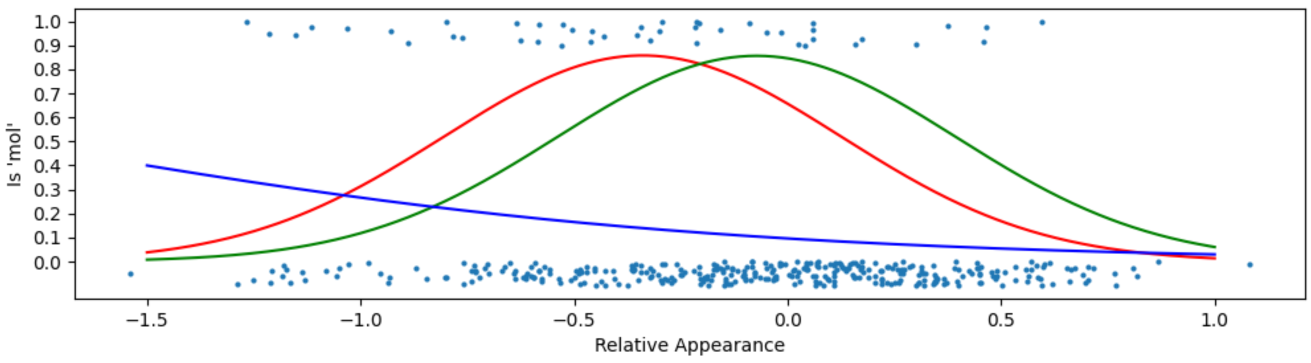


Figure 4.5: Gaussian Naïve Bayes Prediction Model

are not symmetrically distributed, e.g. lower outliers are more deviated from the mean than higher outliers. Therefore the posterior (blue line) is also inaccurate, because the underlying assumptions are invalid. This can also be seen in Figure 4.5, where there are 10 non-*mol* appearance values with *mol* likelihoods ≥ 0.3 compared to 3 *mol* appearance values with a *mol* likelihood ≥ 0.3. A *mol* likelihood of $3/13 \approx 0.23$ was more appropriate for these appearance values.

4.2.3 Split Classifier

Thus none approach has been successful so far. Therefore a new approach was used which again treats all appearance values independently. This classifier splits the appearance values into a lower and higher part, where both intervals get a separate likelihood. To determine whether an appearance value belongs to the lower or higher part, we compute the median m over all appearance values. Appearance values that are smaller than or equal to the median m belong to the lower part and the values larger than the median m belong to the higher part. By splitting at the median, both intervals contain an equal number of values, which ensures a good estimation of the *mol* likelihood for both intervals. These likelihoods $\mathcal{P}_{\text{lower}}$ (for the lower appearance values) and $\mathcal{P}_{\text{higher}}$ (for the higher appearance values) can be estimated by using the following formulae on the train data:

$$\mathcal{P}_{\text{lower}} = \frac{\sum_{x_i \leq m} y_i}{\#\{x_i \leq m\}} \qquad \text{and} \qquad \mathcal{P}_{\text{higher}} = \frac{\sum_{x_i > m} y_i}{\#\{x_i > m\}}$$

where y_i is 1 if the corresponding appearance value belongs to the *mol* and 0 if the corresponding appearance value did not belong to the *mol*. In other words these likelihoods $\mathcal{P}_{\text{lower}}$ and $\mathcal{P}_{\text{higher}}$ are equal to the relative number of *mol* cases in those intervals. Using these likelihoods, the likelihood that someone is the *mol* for new unseen cases can be computed as:

$$\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n) = \prod_{i=1}^{n} \mathcal{P}(p = \text{Mol} \mid A_i)$$

with

$$\mathcal{P}(p = \text{Mol} \mid A_i) = \begin{cases} \mathcal{P}_{\text{lower}} & \text{if } A_i \leq m \\ \mathcal{P}_{\text{higher}} & \text{if } A_i > m \end{cases}$$

after which the likelihoods are *normalized* over all *potential mol* players. Training this Split Classifier for the data of season 13 up to *21* results in the prediction shown in Figure 4.6, which was the first satisfactory prediction for the Appearance Layer. However it is an issue that appearance values slightly smaller than -0.05 are classified totally different than appearance values

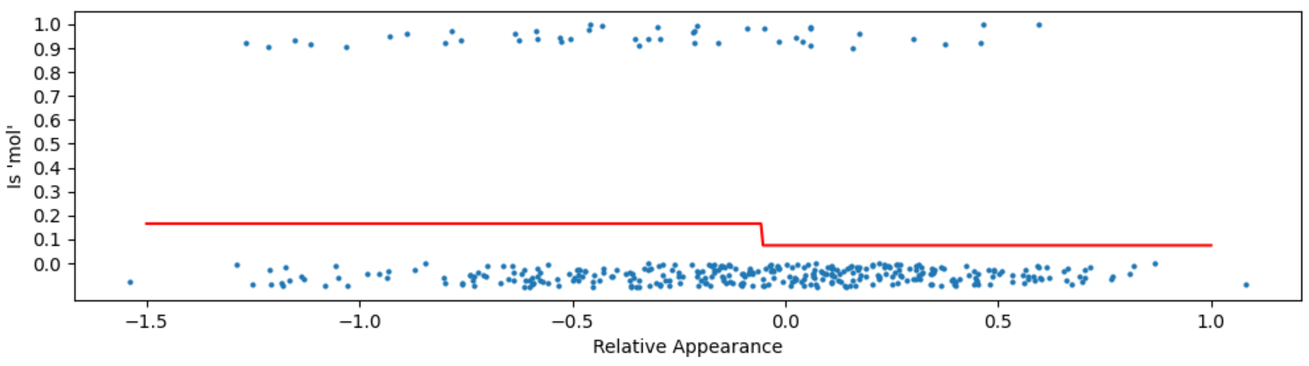


Figure 4.6: Split Classifier Prediction Model

slightly larger than -0.05 and there is no justification why the split should happen at -0.05. A solution to level out these differences is to split the appearance values into more intervals. Though three splits would not be an appropriate choice, because values in the middle are grouped together which is where we expect to be a distinction between *mol* and non-*mol* cases. And four or more splits worsens the *mol* likelihood estimations for those intervals, because with four or more splits there are intervals with five *mol* appearance values or less which is too few for an accurate estimation. Thus the Split Classifier has some major drawbacks. Nevertheless it is more stable and accurate than the Gaussian naïve Bayes classifier.

4.3 Current Approach

With only one feature and 374 train appearance values (of which 45 are *mol* appearance values), it should be possible to find a better prediction model than the models used before. However it was difficult to determine such prediction model. Therefore the problem was simplified to discovering an approach which estimates the probability density function of the appearance values for non-*mol* players and *mol* players. If it is possible to estimate the probability density function for both cases then a naïve Bayes classifier can use it to determine the likelihood of being the *mol*. Fortunately this method exists and it is named Kernel Density Estimation, which can estimate the probability density function of a random variable given a random sample of that random variable. However Kernel Density Estimation is a non-parametric method and therefore requires larger sample sizes than parametric methods, like a normal distributed density estimation, would require to obtain accurate estimations. Nevertheless Kernel Density Estimation performs better for more cases than parametric models would do.

4.3.1 Kernel Density Estimation

Kernel Density Estimation estimates the probability density function $f(x)$ for an independent and identically distributed sample $X = (X_1, X_2, \dots, X_n)$ as [23, section 2.4]:

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

where $K(x)$ is a kernel function and h is the bandwidth parameter. For selection of these kernel function $K(x)$ and the bandwidth parameter h you have plenty of options, which can make the difference between an accurate estimation and inaccurate estimation. Though there are some important conditions which have to be satisfied. The bandwidth parameter h should be a positive value and desirably the kernel function $K(x)$ should satisfy the following conditions:[6]

[6]Only the first condition is really required. The other conditions are desirable for a Kernel function $K(x)$.

1. $K(x)$ should be integrable and $\int_{-\infty}^{\infty} K(x)\partial x = 1$. [23, section 2.4]
2. $K(x)$ should be symmetric, i.e. $\forall_{x\in\mathbb{R}}\ K(x) = K(-x)$. [23, section 2.4]
3. $K(x)$ should be non-negative, i.e. $\forall_{x\in\mathbb{R}}\ K(x) \geq 0$. [23, section 2.4]
4. $K(x)$ should be decreasing as $|x|$ increases, i.e. $\forall_{x,y\in\mathbb{R}}\ |x| \leq |y| \implies K(x) \geq K(y)$.

Commonly selected kernel functions are the Gaussian, Rectangular, Triangular and Epanechnikov kernel functions [23, page 43]. For the Appearance Layer we stick to the Gaussian kernel, which is defined as:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

The advantages of the Gaussian kernel over other kernel functions is that it is a well-studied kernel, it is a smooth kernel and has a non-zero density for all values x. The latter reason is especially important, because kernels with zero density for some values x could cause troubles with zero division in the naïve Bayes computation, which becomes clear later. To determine the bandwidth h for the kernel density estimation, the following formula is used:

$$h = 1.06 \cdot \text{min}\left(\sigma, \frac{\text{IQR}}{1.34}\right) \cdot n^{-\frac{1}{5}}$$

which is a general rule of thumb when choosing h.[7] Remark that there are also different bandwidth estimation rules, but since IQR in this formula is the interquartile range, i.e. the difference between the third quartile and the first quartile of X. Furthermore σ in this formula is the estimated standard deviation of X, which is defined as:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \mu)^2} \qquad \text{with} \qquad \mu = \frac{1}{n}\sum_{i=1}^{n} X_i$$

With this method to select the bandwidth h and the selected kernel function $K(x)$, the probability density function of the appearance values can be estimated. And this is done for the *mol* cases $\rho(x \mid p = \text{Mol})$ and the non-*mol* cases $\rho(x \mid p \neq \text{Mol})$. Using these densities we are now finally able to compute the likelihood of being the *mol* with a naïve Bayes approach (we assume again independence between the appearance values). According to the naïve Bayes approach the *mol* likelihood can be computed as (similarly as in Section 4.2.2):

$$\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n) = \frac{\rho(A_1, \ldots, A_n \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol})}{\rho(A_1, \ldots, A_n \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol}) + \rho(A_1, \ldots, A_n \mid p \neq \text{Mol}) \cdot \mathcal{P}(p \neq \text{Mol})}$$

where $\mathcal{P}(p = \text{Mol}) = \frac{1}{\text{\#players left}}$ and $\mathcal{P}(p \neq \text{Mol}) = \frac{\text{\#players left}-1}{\text{\#players left}}$. Moreover because of the assumed independence it holds that:

$$\rho(A_1, \ldots, A_n \mid p = \text{Mol}) = \prod_{i=1}^{n} \rho(A_i \mid p = \text{Mol}) \quad \text{and} \quad \rho(A_1, \ldots, A_n \mid p \neq \text{Mol}) = \prod_{i=1}^{n} \rho(A_i \mid p \neq \text{Mol})$$

which makes it possible to compute the likelihood of being the *mol* given the kernel density estimation and the appearance values. Note however that the *mol* likelihoods of all players might not sum up to 1, so they have to be *normalized* first before being interpreted as *mol* likelihoods.

[7] This formula is mentioned in the book 'Density Estimation for Statistics and Data Analysis' [23, section 3.4.2], which is a combination of rule (3.28) and (3.30).

4.3.2 Data Augmentation

One problem with Kernel Density Estimation is that the accuracy of the estimation highly depends on the amount of training data. There are plenty of non-*mol* training data, i.e. 329 cases in total when the appearance values of the first five episodes are used for season 13 up to *21*. So increasing the number of non-*mol* cases does not have a major influence on the estimation of $\rho(x \mid p \neq \text{Mol})$. However with only 45 appearance values for *mol* cases, we can definitely get a better estimation of $\rho(x \mid p = \text{Mol})$ by acquiring more appearance values. A technique that could be used to increase the number of appearance values is to include the appearance values of more episodes in the training data and to use appearance values of more episodes to make predictions. There is however a risk by doing so. In later episodes there are fewer players *alive* which implies that the variance of the relative frame count $\frac{FC_{p,e}}{\sum_{p'} FC_{p',e}}$ in Formula 4.1 is larger. So using an additional episode increases the number of outliers. Moreover because the variance is larger in later episodes, these appearance values are less representative compared to the first episode. For example when including the sixth episode of every season in the data set, the sixth episode of season 17 & 18 is included as well with only five players *alive*. These appearance values are not comparable to the first episode with ten players *alive*. Thus the sixth episode is not added to the data set.

Another method to increase the data size is to cut every episode in 4 parts. These cuts are selected by joining all frame numbers for every player, in which the given player appears, together as a group of frame numbers $F = (f_1, f_2, \dots, f_m)$.[8] As cuts the first quartile, the median and the third quartile of F are chosen, which splits the group of all frames in 4 subgroups with an (approximately) equal number of frames. Now we enable/disable parts and compute the appearance value (see Formula 4.1) over only the enabled parts and this is done for all combinations where at least 2 parts are enabled, which are 11 combinations in total (see Figure 4.7). Therefore this method multiplies the number of appearance values by 11. Hence after using this method there are 495 *mol* appearance values and 3619 non-*mol* appearance values, which should give accurate estimations for the probability density functions $\rho(x \mid p \neq \text{Mol})$ and $\rho(x \mid p = \text{Mol})$. The only disadvantage of this technique is that it increases the variance of the appearance values. Although this is mostly prevented by only allowing combinations where at least 2 cuts are enabled. Furthermore with this method you get a lot of similar appearance values, because a player that appears a lot in one part of the episode appears more in another part of the episode as well. Moreover appearance values in combinations where 3 parts are enabled are very similar to the appearance values where all parts are enabled. However the benefit of this technique, i.e. multiplying the data size by 11, outweighs these disadvantages and therefore this technique is used.

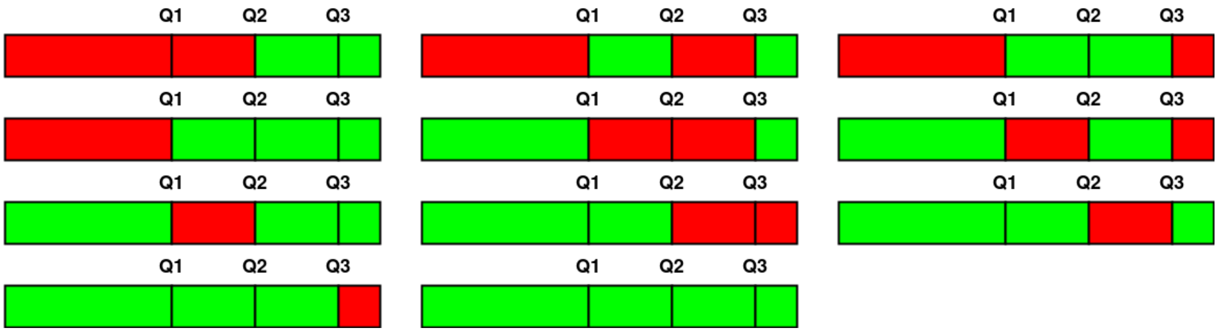


Figure 4.7: All Combinations (green means enabled, red means disabled)

[8]Note that F is not a set, which means that a frame number f can occur twice or more in F if multiple players appeared in that same frame f.

4.3.3 Outlier Cutoff

So after using the data augmentation method and training the kernel density classifier, we should have a proper classifier. Unfortunately this is not the case. When both is done, the appearance values (for season 13 up to *21*) as shown in Figure 4.8 are obtained. This corresponds to the prediction shown in Figure 4.9 that is based on a single appearance value (where it is assumed that there are 10 *potential mol* players).

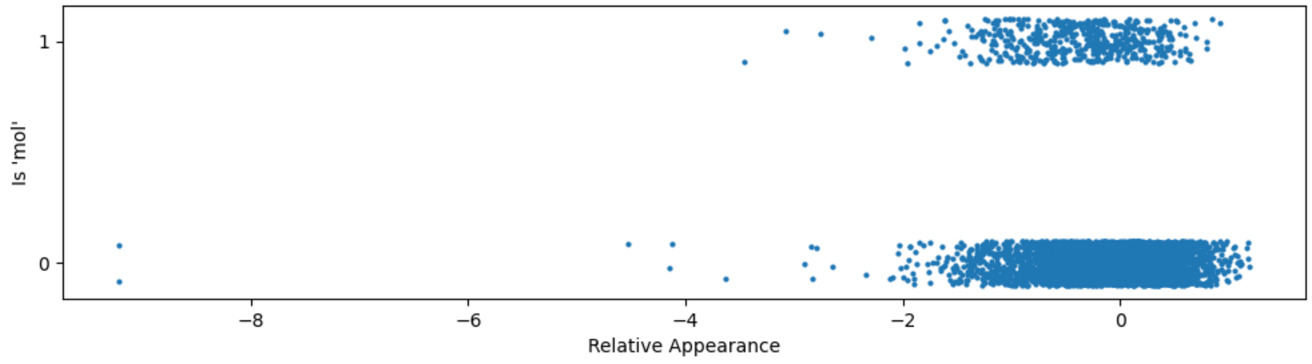


Figure 4.8: Only Augmented Appearance Values

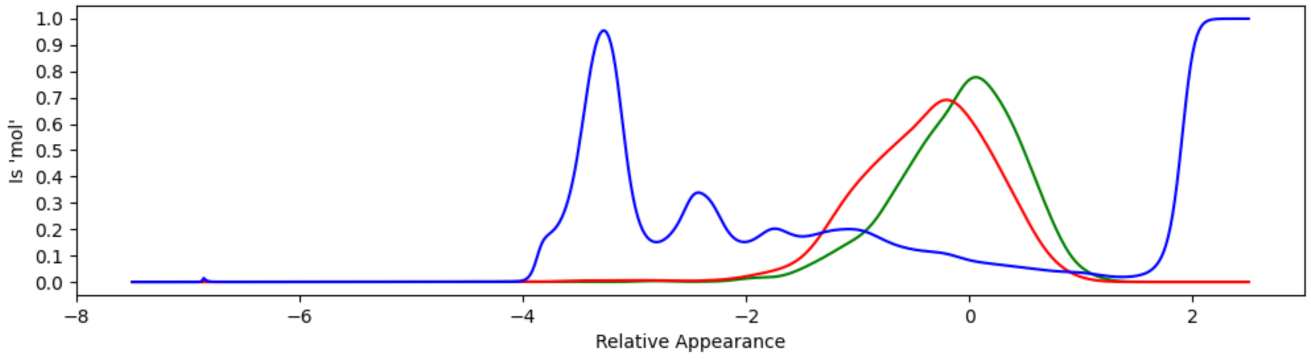


Figure 4.9: Only Augmented Kernel Density Prediction Model

The red line in Figure 4.9 is an estimated probability density of all the *mol* appearance values, i.e. $\rho(A_i \mid p = \text{Mol})$. The green line is an estimated probability density of all the non-*mol* appearance values $\rho(A_i \mid p \neq \text{Mol})$. And the blue line is the posterior based on a single appearance value $\mathcal{P}(p = \text{Mol} \mid A_i)$. What becomes clear from this plot is that the prediction is decent for appearance values in the range of -2 up to 1. But outside this range, the prediction is extreme. For example a player with an appearance value of -6 can never be the *mol*, whereas a player with an appearance value of -3.3 is always the *mol*. There are 2 likely causes for these extreme predictions:

- The probability densities $\rho(A_i \mid p = \text{Mol})$ and $\rho(A_i \mid p \neq \text{Mol})$ for an appearance value A_i outside the range $(-2, 1)$ is quite low. So in the naïve Bayes computation of the *mol* likelihood $\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n)$ a small value is divided by a small value which results in unexpected outcomes.
- There are outlier appearance values shown in Figure 4.8 that have a negative impact on the Kernel Density Estimation and the bandwidth selection.

To deal with the first issue, a lower bound LB and an upper bound UB are determined on the appearance values for which predictions are made. These bounds are defined such that:

$$\mathcal{P}(A \leq \text{LB}) = 0.005 \qquad \mathcal{P}(A \leq \text{UB}) = 0.995$$

where $\mathcal{P}(A \leq x)$ is the probability that an appearance value A is smaller than x which can be estimated by:

$$\begin{aligned}
\mathcal{P}(A \leq x) &= \mathcal{P}(A \leq x \mid p = \text{Mol}) \cdot \mathcal{P}(p = \text{Mol}) + \mathcal{P}(A \leq x \mid p \neq \text{Mol}) \cdot \mathcal{P}(p \neq \text{Mol}) \\
&= \int_{-\infty}^{x} \rho(A \mid p = \text{Mol}) \partial A \cdot \mathcal{P}(p = \text{Mol}) + \int_{-\infty}^{x} \rho(A \mid p \neq \text{Mol}) \partial A \cdot \mathcal{P}(p \neq \text{Mol}) \\
&= \int_{-\infty}^{x} \frac{1}{n_1 h} \sum_{i=1}^{n_1} K\left(\frac{A - X_{1,i}}{h}\right) \partial A \cdot \mathcal{P}(p = \text{Mol}) + \int_{-\infty}^{x} \frac{1}{n_2 h} \sum_{i=1}^{n_2} K\left(\frac{A - X_{2,i}}{h}\right) \partial A \cdot \mathcal{P}(p \neq \text{Mol}) \\
&= \frac{\mathcal{P}(p = \text{Mol})}{n_1 h} \sum_{i=1}^{n_1} \int_{-\infty}^{x} K\left(\frac{A - X_{1,i}}{h}\right) \partial A + \frac{\mathcal{P}(p \neq \text{Mol})}{n_2 h} \sum_{i=1}^{n_2} \int_{-\infty}^{x} K\left(\frac{A - X_{2,i}}{h}\right) \partial A \\
&= \frac{\mathcal{P}(p = \text{Mol})}{n_1 h} \sum_{i=1}^{n_1} \Phi\left(\frac{x - X_{1,i}}{h}\right) + \frac{\mathcal{P}(p \neq \text{Mol})}{n_2 h} \sum_{i=1}^{n_2} \Phi\left(\frac{x - X_{2,i}}{h}\right)
\end{aligned}$$

In this formula $\Phi(x)$ is the cumulative distribution function of a standard normal distribution. Moreover $(X_{1,1}, X_{1,2}, \ldots, X_{1,n_1})$ are all *mol* appearance values used as training data and $(X_{2,1}, X_{2,2}, \ldots, X_{2,n_2})$ are all non-*mol* appearance values used as training data. Thus $\mathcal{P}(A \leq x)$ is fully estimatable, which can be used in combination with the bisection method [24, section 4.3] to determine the lower bound LB and the upper bound UB on the appearance values. If an appearance value A occurs during the prediction phase that is lower than LB, the posterior is computed as if A would be equal to LB. More specifically:

$$P(p = \text{Mol} \mid A \leq \text{LB}) = P(p = \text{Mol} \mid A = \text{LB})$$

Similar if A is larger than UB, the posterior is computed as if A would be equal to UB, i.e.

$$P(p = \text{Mol} \mid A \geq \text{UB}) = P(p = \text{Mol} \mid A = \text{UB})$$

To deal with the second issue, 0.5% of the lowest appearance values are removed and 0.5% of the highest appearance values are removed from the train data, which removes all outliers. Remark that outliers are not false measurements, e.g. during episode 2 of season 17 some players were kidnapped and therefore did not appear during a large part of the episode. Thus these players got a rightful low appearance value for this episode. However they are removed, since they have a negative impact on the Kernel Density Estimation. Secondly this technique also removes some inliers, but there is plenty of data after using the data augmentation techniques, so removing some inliers does not worsen the Kernel Density Estimation a lot. In Figure 4.10, the prediction is shown after using these techniques (where it is assumed that there are 10 *potential mol* players).

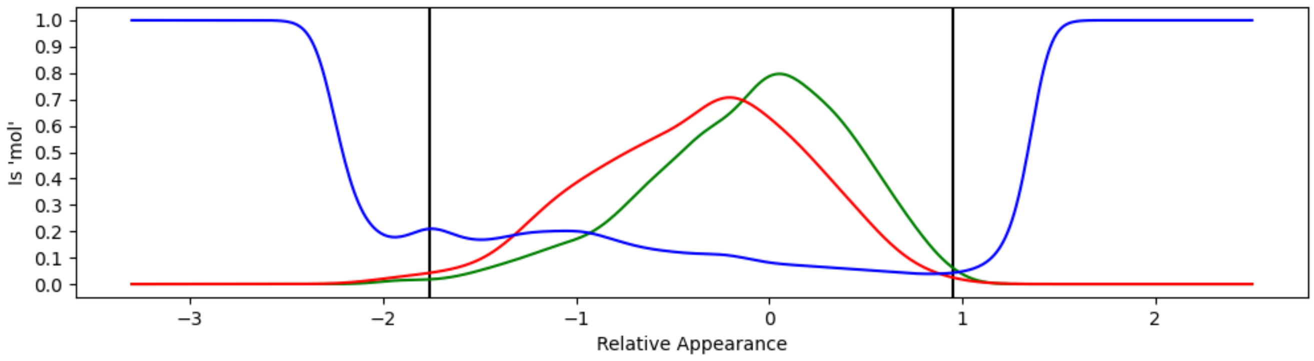


Figure 4.10: Kernel Density Prediction Model

This prediction is quite similar to the prediction in Figure 4.9 for the range $(-2, 1)$. Although in this prediction appearance values get cut off, i.e. the black vertical line on the left in Figure 4.10 represents the lower bound LB and the black vertical line on the right represents the upper bound UB. These bounds ensure that all predictions happen in a safe range. So with all these cut-off techniques and data augmentation techniques combined, the likelihood of being the *mol* can be computed, i.e.

$$\mathcal{P}(p = \text{Mol} \mid A_1, \ldots, A_n)$$

All of these likelihoods of course have to be *normalized*, because they might not sum up to 1 for all players. Moreover after *normalization*, all predictions are limited to the range:

$$\left(\frac{0.2}{\text{\#players left}}, \frac{1}{\text{\#players left}}\right)$$

This means that likelihoods lower than $\frac{0.2}{\text{\#players left}}$ are set to $\frac{0.2}{\text{\#players left}}$ and values higher than $\frac{1}{\text{\#players left}}$ are set to $\frac{1}{\text{\#players left}}$, except for non-*potential mol* players whose likelihood is set to zero. After this process the likelihoods are *normalized* again. The likelihoods of the appearance layer get limited to this range, because the prediction of the appearance layer went horribly wrong for season 16. Moreover in season 11 and 12 there was also no clear pattern in appearance values. And there are only 8 seasons of training data which is still not a lot. More specifically, the reason for this range is that the Appearance Layer should only be used exclude players as *mol*, thus the predictions have an upperbound of $\frac{1}{\text{\#players left}}$. Moreover the predictions have lowerbound of $\frac{0.2}{\text{\#players left}}$, because the Appearance Layer did not provide good results for seasons 11, 12 and 16. If we weight seasons 11 and 12 by 0.5 (since earlier seasons are less representative than later seasons) then relatively

$$\frac{0.5 + 0.5 + 1}{0.5 + 0.5 + 9} = 0.2$$

of the seasons had an inaccurate prediction by the Appearance Layer.

5 AGGREGATION

5.1 Introduction

The strength of the Moldel is of course that it uses multiple prediction models rather than a single one, which should improve the prediction strength. This is similar to a crime investigation, e.g. if you have only a single witness then it is hard to determine the perpetrator of this crime. The witness might be lying or could have memorized things wrongly. Whereas with multiple witnesses it is very unlikely that they all come up with the same lie or have all memorized the same thing wrongly. Thus with multiple witnesses it should be easier to determine the perpetrator. Similarly for the Moldel it sometimes happens that a layer predicts the wrong *mol*. It happens for example than an early dropouts suspected the actual *mol*, the *mol* characteristic did not match with those of previous *mol* players or the *mol* appeared a lot. However in contrast to the witnesses example, predictions of the layers also shows a certainty level. This adds an additional dimension to the problem, which can be used to improve the final prediction.

All the layers that are aggregated include the Exam Drop Layer, the Exam Pass Layer, the Wikipedia Layer and the Appearance Layer. An issue with aggregating these layers is that these layers cannot make predictions for all seasons. For example the Appearance Layer can only make predictions starting from season 13, whereas the Exam Pass Layer can already make predictions starting from season 5. To overcome this problem, the prediction of any layer for an undefined season is defined as the uniform distribution (see Equation 1.1). With this fix it is ensured that in all seasons every player p_i has 4 predictions, even if some layers are undefined for that season. These predictions for player p_i are: $\rho_{i,1}$ of the Exam Drop Layer, $\rho_{i,2}$ of the Exam Pass Layer, $\rho_{i,3}$ of the Wikipedia Layer and $\rho_{i,4}$ of the Appearance Layer. In the next sections the following details are discussed regarding aggregation of these predictions:

- In Section 5.2 all different approaches are discussed that could be used to aggregate the layers. It starts with simple techniques, which includes arithmetic mean, naïve aggregation and geometric mean. After which a more advanced ensemble learning technique is discussed, called stacking. The advantages and disadvantages of each of these methods are discussed as well.
- In Section 5.3 it is discussed which approach has been selected for the Moldel and how it has been implemented.
- In Section 5.4 it is discussed how the predictions can be adjusted based on social media analysis. Furthermore the reasoning behind the social media analysis is discussed.

5.2 Approaches

5.2.1 Arithmetic Mean

A frequently used technique to aggregate multiple predictions is to use the arithmetic mean of the predictions, i.e. for player p_i the *mol* likelihood ρ_i is computed as:

$$\rho_i = \frac{1}{4}\sum_{j=1}^{4} \rho_{i,j}$$

after which the *mol* likelihoods $\rho_1, \dots, \rho_n$ are *normalized* for respective players $p_1, \dots, p_n$. The advantage of this aggregation technique is that it is easy to implement and no additional training data is needed. Though this method does not provide realistic *mol* likelihoods, which is illustrated by the following example:

Example 7. *Suppose there are 4* potential mol *players Art, Karin, Patrick and Soundos. Moreover the Exam Drop Layer gives Art a* mol *likelihood of 0.01, whereas Karin, Patrick and Soundos all get a* mol *likelihood of 0.33 by the Exam Drop Layer. Furthermore the Exam Pass Layer, the Wikipedia Layer and the Appearance Layer are all undecided, i.e. they give all players a likelihood of 0.25. After using the arithmetic mean as aggregation technique, the following* mol *likelihoods are obtained:*

	Art	Karin	Patrick	Soundos
Mol likelihood	0.19	0.27	0.27	0.27

Table 5.1: Predictions after Arithmetic Mean

What is troublesome about the predictions after arithmetic mean is that Art has a decent likelihood of being the *mol* whereas the Exam Drop Layer almost excludes Art as *mol*. And these kind of predictions are not uncommon for the Moldel, where a player is often given only a very low *mol* likelihood by a single layer. In the next aggregation technique lower likelihoods have more influence on the outcome.

5.2.2 Naïve Aggregation

Likelihoods can also be aggregated using naïve aggregation. Naïve aggregation multiplies all likelihoods together, i.e. for player p_i the aggregated *mol* likelihood ρ_i is computed as:

$$\rho_i = \prod_{j=1}^{4} \rho_{i,j}$$

after which the *mol* likelihoods $\rho_1, \dots, \rho_n$ are *normalized* for respective players $p_1, \dots, p_n$. The benefits of naïve aggregation are similar as arithmetic mean in the sense that naïve aggregation also does not require any training data and is easy to implement as well. But in addition lower likelihoods have more influence on the outcome, e.g. the *mol* likelihoods for Example 7 would have been:

	Art	Karin	Patrick	Soundos
Mol likelihood	0.01	0.33	0.33	0.33

Table 5.2: Predictions after Naïve Aggregation

if naïve aggregation was used as technique. And these *mol* likelihoods are more realistic than the likelihoods given by the arithmetic mean. Moreover this technique is mathematically justifiable as well if:

1. The predictions are fully accurate, i.e. the prediction of the layer j for player p_i is equal to the probability of being the *mol* given the data provided to that layer j:

$$\rho_{i,j} = \mathcal{P}(p_i = \text{Mol} \mid \tbinom{\text{Layer j}}{\text{Data}})$$

2. The predictions of the layers are conditionally independent given the identity of the *mol*, i.e.

$$\mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}} \mid p_i = \text{Mol}) = \prod_{j=1}^{4} \mathcal{P}(\tbinom{\text{Layer j}}{\text{Data}} \mid p_i = \text{Mol})$$

since aggregation is used to estimate $\mathcal{P}(p_i = \text{Mol} \mid \tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}})$ which can be rewritten with both assumptions as:

$$\begin{aligned}
\mathcal{P}(p_i = \text{Mol} \mid \tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}}) &= \frac{\mathcal{P}(p_i = \text{Mol})}{\mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}})} \cdot \mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}} \mid p_i = \text{Mol}) \\
&= \frac{\mathcal{P}(p_i = \text{Mol})}{\mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}})} \cdot \prod_{j=1}^{4} \mathcal{P}(\tbinom{\text{Layer j}}{\text{Data}} \mid p_i = \text{Mol}) \\
&= \frac{\mathcal{P}(p_i = \text{Mol})}{\mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}})} \cdot \prod_{j=1}^{4} \frac{\mathcal{P}(\tbinom{\text{Layer j}}{\text{Data}}) \cdot \mathcal{P}(p_i = \text{Mol} \mid \tbinom{\text{Layer j}}{\text{Data}})}{\mathcal{P}(p_i = \text{Mol})} \\
&= \frac{\mathcal{P}(p_i = \text{Mol})}{\mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}})} \cdot \prod_{j=1}^{4} \frac{\mathcal{P}(\tbinom{\text{Layer j}}{\text{Data}}) \cdot \rho_{i,j}}{\mathcal{P}(p_i = \text{Mol})} \\
&= \frac{\prod_{j=1}^{4} \mathcal{P}(\tbinom{\text{Layer j}}{\text{Data}})}{\mathcal{P}(\tbinom{\text{Layer 1}}{\text{Data}}, \ldots, \tbinom{\text{Layer 4}}{\text{Data}}) \cdot \mathcal{P}(p_i = \text{Mol})^3} \cdot \prod_{j=1}^{4} \rho_{i,j} \\
&= \alpha \cdot \prod_{j=1}^{4} \rho_{i,j}
\end{aligned}$$

where α is a constant used to *normalize* the probabilities over all players. Although it is doubtful whether both conditions are satisfied. A lot of simplifications have been made in all layers and there is not enough training data to estimate the *mol* likelihood in every case, which causes

$$\rho_{i,j} \neq \mathcal{P}(p_i = \text{Mol} \mid \tbinom{\text{Layer j}}{\text{Data}})$$

Furthermore conditional independence between data of different layers is also not fully justifiable. For example the data for the Exam Drop Layer and the Exam Pass Layer are both extracted from the same source. So both conditions are unlikely to hold, in which case this aggregation technique could be very inaccurate. To illustrate this, if the Exam Drop Layer in Example 7 is aggregated with itself (which is a clear violation of the conditionally independence principle) then the prediction would be as shown by Table 5.3.

	Art	Karin	Patrick	Soundos
Mol likelihood	0.0001	0.3333	0.3333	0.3333

Table 5.3: Predictions after Naïve Aggregation of 2 Exam Drop Layers

5.2.3 Geometric Mean

An aggregation technique that is also considerable is geometric mean. Geometric mean aggregates the *mol* likelihoods by:

$$\rho_i = \sqrt[4]{\prod_{j=1}^{4} \rho_{i,j}}$$

after which the *mol* likelihoods $\rho_1, \ldots, \rho_n$ are normalized for respective players $p_1, \ldots, p_n$. The characteristic of the geometric mean is that low likelihoods have more influence on the prediction than for the arithmetic mean, but less than naïve aggregation. In Example 7 the prediction using geometric mean would have been:

	Art	Karin	Patrick	Soundos
Mol likelihood	0.1133	0.2956	0.2956	0.2956

Table 5.4: Predictions after Geometric Mean

The benefits of geometric mean are similar as arithmetic mean and naïve aggregation, i.e. geometric mean is easy to implement and no additional training data is needed. Though all these aggregation techniques so far weight layers equally, whereas some layers provide more reliable predictions. An unequal weighting could thus provide better results.

5.2.4 Stacking

Stacking [25, section 4.4.1] is a general framework rather than a specific method to aggregate multiple predictions into one prediction that allows unequal weighting. In stacking all the k individual classifiers, also named the first-level learners, are trained on a data set D:

$$D = \{(x_1, y_1), \ldots, (x_n, y_n)$$

where x_i is an input vector and y_i is the corresponding label. These first-level learners are then used on a different data set:

$$D' = \{(x'_1, y'_1), \ldots (x'_m, y'_m)\}$$

which means that $y'_1, \ldots, y'_m$ in this data set D' must be predicted by the first-level learners based on $x'_1, \ldots, x'_m$[1] resulting in a prediction matrix:

$$\begin{bmatrix} y^*_{1,1} & y^*_{1,2} & \cdots & y^*_{1,k-1} & y^*_{1,k} \\ y^*_{2,1} & y^*_{2,2} & \cdots & y^*_{2,k-1} & y^*_{2,k} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ y^*_{m-1,1} & y^*_{m-1,2} & \cdots & y^*_{m-1,k-1} & y^*_{m-1,k} \\ y^*_{m,1} & y^*_{m,2} & \cdots & y^*_{m,k-1} & y^*_{m,k} \end{bmatrix}$$

where $y^*_{i,j}$ is an estimate of y'_i by the jth classifier. And with this prediction matrix and corresponding actual labels a new data set is constructed:

$$D^+ = \{(x^+_1, y'_1), \ldots, (x^+_m, y'_m)\}$$

The input vector x^+_i in this data set D^+ is defined as the ith row in the prediction matrix, i.e.

$$x^+_i = (y^*_{i,1}, y^*_{i,2}, \ldots, y^*_{i,k-1}, y^*_{i,k})$$

[1]Note that it is important in this step that $y'_1, \ldots, y'_m$ are not leaked to the first-level learners.

After this step a new classifier M, called the second-level learner, is trained on data set D^+ which finalizes the training process of the stacking framework. To make predictions using the stacking framework, you make a prediction with all k individual classifiers for a new vector x^- resulting in predictions $(y_1^-, y_2^-, \ldots, y_m^-)$ and you make a prediction with M for $(y_1^-, y_2^-, \ldots, y_m^-)$ resulting in the final prediction label y^-. The benefit of stacking is that it is applicable for the Moldel as well. Though in the framework the same input $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_n$ is used by all first-level learners, whereas in the Moldel different input is used by different layers. Nevertheless in stacking it is allowed to use different input for every individual classifier. x_i and x'_i can be vectors of input vectors instead, i.e.

$$x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k}) \qquad x'_i = (x'_{i,1}, x'_{i,2}, \ldots, x'_{i,k})$$

This is an advantage over other ensemble learning methods, such as bagging, boosting and Bayesian model averaging that require the same input to be used by all first-level learners. Likewise stacking has a benefit over arithmetic mean, naïve aggregation and geometric mean, since it allows unequal weighting of the predictions and can therefore produce more accurate predictions. However stacking also has some drawbacks. There is often less training data available for the layers if stacking is used, because a part of the data is required for training the second-level learner. To minimize this issue one could use cross validation [26, section 11.2.4], where the data set D is split up in partitions $D = D_1 \cup D_2 \cup \cdots \cup D_t$ such that every partition D_i holds a part of the data. The second-level learner is then trained on every data set D_i, where the layers are trained every time on the data set $D \setminus D_i$. However this solution has as disadvantage that the total training time compared to arithmetic mean, naïve aggregation and geometric mean is multiplied by t, where t is the number of partitions. So this solution is much slower than arithmetic mean, naïve aggregation and geometric mean. The last drawback of stacking is that responsibilities become vague. Suppose for example that the aggregated predictions are inaccurate. Is this caused by an inaccurate second-level learner, is this caused by inaccurate first-level learners or both?

5.3 Current Approach

5.3.1 Second-Level Learner

Despite the disadvantages of stacking, it was used, since it is expected to provide more accurate predictions than the other aggregation methods. As second-level learner for stacking a Logistic Regression is taken for which a similar implementation (including stop criterion) as in Section 2.3.4 is used excluding training weights. Thus the *mol* likelihood ρ_i for player p_i is determined as:

$$\rho_i = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is equal to $w^T x_i$, w are the trained weights for the logistic regression and x_i is the input encoding of player p_i. The selected input encoding for x_i is:

$$x_i = (\sigma^{-1}(\rho_{i,1}), \sigma^{-1}(\rho_{i,2}), \sigma^{-1}(\rho_{i,3}), \sigma^{-1}(\rho_{i,4}), \sigma^{-1}\left(\frac{1}{|P_+|}\right), 1)^T$$

with $\sigma^{-1}(z)$ being the logit function which is the inverse of the logistic function $\sigma(z)$ and is defined as:[2]

$$\sigma^{-1}(z) = \log\left(\frac{z}{1-z}\right)$$

Moreover $\rho_{i,1}, \rho_{i,2}, \rho_{i,3}, \rho_{i,4}$ are the predictions of respectively the Exam Drop Layer, the Exam Pass Layer, the Wikipedia Layer and the Appearance Layer and $|P_+|$ is the number of *potential*

[2]An exception to this rule is described in the Exception Handling appendix at A.3.1.

mol players. The benefit by applying the logit function on the features is that it allows the logistic regression to take some weighted average of the predictions in a linear space and the benefit by adding the uniform prediction $\frac{1}{|P_+|}$ as feature is that the stacking algorithm can decide to temper the prediction if it is uncertain about the *mol*. Secondly to temper the predictions an L2 regularization is applied, which penalizes large training weights. In a normal non-regularized logistic regression, w is computed by minimizing the log loss (regarding Equation 2.1):

$$\arg\min_w \; -\sum_{x,y \in T} y \cdot \ln(\sigma(w^T x)) + (1-y)\ln(1-\sigma(w^T x))$$

In an L2 regularized version the following loss is minimized instead:

$$\arg\min_w \; -\sum_{x,y \in T} \big(y \cdot \ln(\sigma(w^T x)) + (1-y)\ln(1-\sigma(w^T x))\big) + \sum_{i=1}^{k} (w_i)^2$$

which prefers smaller training weights over larger training weights. The advantage of a L2 regularized version is that there is a smaller risk of overfitting and the prediction outcomes will be more moderate. This is preferred, because only a few seasons so far have been evaluated by the Moldel.

5.3.2 Training Procedure

Prediction in the Moldel are identified by a pair (s, e) where s is the season number for which the prediction is made and e is the episode number after which the prediction is made. A prediction identified by (s, e) is able to use all data from season s up to episode e to determine the *mol* likelihood of every player. It is allowed for an identifier to have $e = 0$ which means that the prediction is made before the first episode has even been broadcast.[3] If the Moldel is evaluated on an identifier (s', e') in the set of all identifiers:

$$I = \{(5,0), (5,1), \ldots, (11,10), \ldots, (21,6), (21,7)\}$$

for which there is data, then the available identifiers for training the second-level logistic regression are:

$$I' = \{(s,e) \in I \mid s \neq s'\}$$

This set of available identifiers for stacking is narrowed further down to I^+, by looking at how many *potential mol* players there are in season s' after episode e' which is defined as $|P_{s',e'}|$. The possible values of $|P_{s',e'}|$ are:

– $|P_{s',e'}| \geq 9$, in which case I^+ is defined as all identifiers (s, e) in I' where there are at least 9 *potential mol* players in season s after episode e, i.e.

$$I^+ = \{(s,e) \in I' \mid \; |P_{s,e}| \geq 9\}$$

– $|P_{s',e'}| \in \{7, 8\}$, in which case I^+ is defined as all identifiers (s, e) in I' where there are either 7 or 8 *potential mol* players in season s after episode e, i.e.

$$I^+ = \{(s,e) \in I' \mid \; |P_{s,e}| \in \{7,8\}\}$$

– $|P_{s',e'}| \in \{5, 6\}$, in which case I^+ is defined as all identifiers (s, e) in I' where there are either 5 or 6 *potential mol* players in season s after episode e, i.e.

$$I^+ = \{(s,e) \in I' \mid \; |P_{s,e}| \in \{5,6\}\}$$

[3]With $e = 0$ the prediction is solely based on the Wikipedia layer.

– $|P_{s',e'}| \leq 4$, in which case I^+ is defined as all identifiers (s,e) in I' where there are at most 4 *potential mol* players in season s after episode e, i.e.

$$I^+ = \{(s,e) \in I' \mid |P_{s,e}| \leq 4\}$$

The pairs (s,e) in data set I^+ are then used to train the logistic regression. An advantage of this categorization technique is that the stacking algorithm can temper earlier predictions if there are still many *potential mol* players, whereas it can strengthens predictions if there are only few *potential mol* players. To train this logistic regression with data set I^+, cross validation is applied (as mentioned at the end of Section 5.2.4). This means that we loop over all identifiers $(s^*,e^*) \in I^+$ (a), *train the layers* on the remaining seasons S^* (b) with:

$$S^* = \{s \mid \exists_e \ (s,e) \in I \land s \neq s^*\}$$

and *make a prediction* by all trained layers for identifier (s^*,e^*) (c) which results in the prediction vector for every *potential mol* players $p^* \in P_{s^*,e^*}$

$$\rho_{(s^*,e^*,p^*)} = (\rho_{(s^*,e^*,p^*,1)},\ \rho_{(s^*,e^*,p^*,2)},\ \rho_{(s^*,e^*,p^*,3)},\ \rho_{(s^*,e^*,p^*,4)})$$

where $\rho_{(s^*,e^*,p^*,1)}$ is the prediction of the Exam Drop Layer for player p^* in season s^* after episode e^* trained on the seasons S^*. Similarly $\rho_{(s^*,e^*,p^*,2)}$ is this prediction by the Exam Pass Layer, $\rho_{(s^*,e^*,p^*,3)}$ is this prediction by the Wikipedia Layer and $\rho_{(s^*,e^*,p^*,4)}$ is this prediction by the Appearance Layer. These prediction vectors are then transformed to (d):

$$\rho'_{(s^*,e^*,p^*)} = (\sigma^{-1}(\rho_{(s^*,e^*,p^*,1)}), \sigma^{-1}(\rho_{(s^*,e^*,p^*,2)}), \sigma^{-1}(\rho_{(s^*,e^*,p^*,3)}), \sigma^{-1}(\rho_{(s^*,e^*,p^*,4)}), \sigma^{-1}\left(\frac{1}{|P_{s^*,e^*}|}\right), 1)$$

which are all used to train the second-level logistic regression. The pseudocode for the training procedure is as follows:

Algorithm 2 Stacking Training Procedure

```
function trainStacker($I^+$)        ▷ $I^+$ are the available identifiers for training the Stacker.
 1: $D = \{\}$        ▷ Start with an empty data set.
 2: for $(s^*,e^*) \in I^+$ do        ▷ Execute step (a).
 3:     $L$ = array of all untrained layers
 4:     $S^* = \{s \mid \exists_e \ (s,e) \in I \land s \neq s^*\}$        ▷ $S^*$ are available seasons for training the layers.
 5:     for $l \in L$ do
 6:         $l$.train($S^*$)        ▷ Train layer $l$ on seasons $S^*$, which is step (b).
 7:     end for
 8:     for $p^* \in P_{s^*,e^*}$ do
 9:         $\rho' = [\,]$        ▷ The input encoding of this player.
10:         for $l \in L$ do
11:                 ▷ Make a prediction by layer $l$ for player $p^*$ in season $s^*$ after episode $e^*$,
12:             $r = l$.predict($s^*, e^*, p^*$)        ▷ which is step (c).
13:             $\rho'$.append($\sigma^{-1}(r)$)        ▷ Execute the first part of step (d).
14:         end for
15:         $\rho'$.append($\sigma^{-1}\left(\frac{1}{|P_{s^*,e^*}|}\right), 1$)        ▷ Execute the second part of step (d).
16:         $D$.append($\rho' \to$ isMol($p^*$))
17:     end for
18: end for
19: $M$ = untrained logistic regression
20: $M$.train($D$)        ▷ Train the stacker on data set $D$.
21: return $M$
```

5.3.3 Weakened Training

The Wikipedia Layer unfortunately does not have the benefit of having a variety of data compared to the other layers. Hence the likelihood outcomes of the Wikipedia Layer are regulated, which means that the outcome *mol* likelihoods are either 0.079 or 0.117 (see Sections 3.3.4). The issue with applying stacking on the Wikipedia Layer is that the outcome likelihoods are either 0.079 or 0.117 of being the *mol* (see Section 3.3.4). These likelihoods are determined based on a z-score threshold of 0, whereas the Wikipedia Layer uses a threshold of -0.524 to divide the players in less likely *mol* players and more likely *mol* players. If stacking is directly applied on the Wikipedia Layer then the likelihood for the less likely *mol* group is adjusted to the relative number of *mol* players (compared to all players) having a z-score smaller than -0.524. And for the more likely *mol* group it is adjusted similarly to the relative number of *mol* players (compared to all players) having a z-score larger or equal than -0.524. Thus the solution is to use Weakened Wikipedia Layer in the training phase that classifies all players with a z-score smaller than 0 as less likely to be the *mol* and with a z-score larger or equal than 0 as more likely to be the *mol*.

5.3.4 Prediction Procedure

After the training procedure, the prediction procedure of the stacking framework is executed. This happens by training all layers on the identifiers S' (e), with

$$S' = \{s \mid \exists_e \ (s, e) \in I'\}$$

The next step is using these trained layers to make a prediction for identifier (s', e') (f), which results in the prediction vector for every *potential mol* players $p' \in P_{s',e'}$:

$$\rho_{(s',e',p')} = \left(\rho_{(s',e',p',1)},\ \rho_{(s',e',p',2)},\ \rho_{(s',e',p',3)},\ \rho_{(s',e',p',4)}\right)$$

These predictions vectors are then transformed similarly as in the training phase to: (g)

$$\rho'_{(s',e',p')} = (\sigma^{-1}(\rho_{(s',e',p',1)}), \sigma^{-1}(\rho_{(s',e',p',2)}), \sigma^{-1}(\rho_{(s',e',p',3)}), \sigma^{-1}(\rho_{(s',e',p',4)}), \sigma^{-1}\left(\frac{1}{|P_{s',e'}|}\right), 1)$$

which is used as input for the second-level logistic regression M to compute the aggregated *mol* likelihood for player p' after episode e' in season s' (h). The pseudocode for the prediction procedure is shown by Algorithm 3.

Algorithm 3 Stacking Prediction Procedure

```
function predict($S', M, s', e', p'$)
1:  ▷ $S'$ are the available seasons for training the layers; $M$ is the trained stacker; $s'$, $e'$ and
    $p'$ are respectively the season, the episode and player for which the prediction is made.
2:  $L$ = array of all untrained layers
3:  for $l \in L$ do
4:      $l$.train($S'$)                    ▷ Train layer $l$ on the seasons $S'$, which is step (e).
5:  end for
6:  $\rho' = [\,]$                                        ▷ The input encoding of player $p'$
7:  for $l \in L$ do
8:                   ▷ Make a prediction by layer $l$ for player $p'$ in season $s'$ after episode $e'$,
9:      $r = l$.predict($s', e', p'$)                                  ▷ which is step (f).
10:     $\rho'$.append($\sigma^{-1}(r)$)                    ▷ Execute the first part of step (g).
11: end for
12: return $M$
```

5.3.5 Precomputed Stacking

But there is a major issue with the training phase of this stacking procedure. Training these layers on season 5 up to 20 for a computer with an Intel Core i7-4710MQ processor and 7.7GB memory takes:

Layers	Exam Drop Layer	Exam Pass Layer	Wikipedia Layer	Appearance Layer
Running Time	6.437s	0.014s	9.610s	1.574s

Table 5.5: Running Time of Layers

Thus running code lines 5, 6, 7 in Algorithm 2 once takes approximately 17.635 seconds, which is feasible if it only needs to be executed once. However the number of identifiers in I^*, if season 5 up to 20 are used as training seasons, are 160. So running this stacking procedure takes around 48 minutes, which is executed every time a prediction is made by the Moldel for a single identifier. This running time is too large, especially since often new approaches and small changes are applied to the Moldel, which needs to be quickly evaluated on multiple identifiers to check if these approaches/changes are beneficial. Hence

- Step (b), i.e. *train the layers* on the remaining seasons S^*.
- Step (c), i.e. *make a prediction* by all trained layers for identifier (s^*, e^*) and for all *potential mol* player $p^* \in P_{s^*,e^*}$.

are replaced by using a precomputed prediction instead, which uses the seasons of S^- for training the layers, where S^- is defined as:

$$S^- = \{s \mid \exists_e \, (s, e) \in I \land s \neq s^*\}$$

S^- here does unfortunately include the season s' which is also used to evaluate the Moldel on. Thus the evaluation of results in Chapter 6 seem better than they actually are which should generally be avoided in any machine learning framework. However there are two reasons why S^- includes season s', because:

1. Some layers of the Moldel already have a shortage of training data. For example if the Moldel is evaluated on season *21* then the Appearance layer trained for the stacking procedure has only 7 seasons available for training, which could cause inaccuracies in the predictions of the Appearance layer and thus results in the stacker to have low trust in the Appearance layer. Using the evaluation season (in this case season *21*) as additional training data for training the Appearance layer in the stacking procedure strengthens the predictions and thus results in the stacker to have higher trust in the Appearance Layer.

2. Even if the predictions are precomputed, these predictions need to be recomputed when the implementation of the Moldel is changed. Suppose that the evaluation season s' is removed from S^- and cross validation is used to evaluate the Moldel, then all precomputed predictions are configured for 16 different sets of training seasons, where one season is removed from the range $\{5, 6, \ldots, 20, 21\}$ in addition to the season that is used to train the stacker. The total number of precomputed predictions in this scenario that needs to be stored are 2416, which takes around 12 hours to recompute. So by doing this we do not only require a lot of storage space, but also a lot of computing time, which is infeasible for a small scale project.

Nevertheless this design choice does not imply any changes to the prediction procedure, i.e. Algorithm 3 is untouched. Thus season s' is not used for training the layers in the prediction procedure. Neither does this design choice imply that the stacker itself is trained with predictions from the evaluation season. Code line 2 in Algorithm 2 remains unchanged, which means that the stacker M is only trained on predictions $(s^*, e^*) \in I^+$. The new pseudocode for the training procedure, when using precomputed predictions, is as follows:

Algorithm 4 Stacking Precomputed Training Procedure

```
function trainStacker($I^+$)                    ▷ $I^+$ are the available identifiers for training the Stacker.
 1: $D = \{\}$                                   ▷ Start with an empty data set.
 2: for $(s^*, e^*) \in I^+$ do                  ▷ Execute step (a).
 3:     for $p^* \in P_{s^*,e^*}$ do
 4:         $\rho' = [\,]$                       ▷ The input encoding of this player.
 5:         for $l \in L$ do
 6:                            ▷ Get precomputed prediction by layer $l$ for player $p^*$ in season $s^*$ after
 7:             $r = l$.precomputed($s^*, e^*, p^*$)   ▷ episode $e^*$, which uses $I^-$ as training data.
 8:             $\rho'$.append($\sigma^{-1}(r)$)       ▷ Execute the first part of step (d).
 9:         end for
10:         $\rho'$.append($\sigma^{-1}\left(\frac{1}{|P_{s^*,e^*}|}\right), 1$)   ▷ Execute the second part of step (d).
11:         $D$.append($\rho' \rightarrow$ isMol($p^*$))
12:     end for
13: end for
14: $M$ = untrained logistic regression
15: $M$.train($D$)                               ▷ Train the stacker on data set $D$.
16: return $M$
```

5.4 Social Media Exclusions

After applying this stacking procedure, we end up with a *mol* likelihood ρ_i for every player p_i. But this ρ_i is not the final *mol* likelihood of player p_i. Before obtaining the final *mol* likelihood, players are being excluded as *mol* with the Social Media Layer. The Social Media Layer is a layer based on the idea of Jaap van Zessen, which uses big data techniques to exclude players as *mol*. This idea dates back to 2014 in which year van Zessen analysed the social media accounts of 'Wie is de Mol?' players for season 14 using Buzzcapture [27]. Based on their activity on Social Media it was found out that season 14 was recorded between 13th of May and 6th of June in 2013, since most of the players were inactive on social media during that period. There was one exception to this case, which was Maurice, who was active on social media starting from the 20th of May. So according to van Zessen, Maurice could not have been the *mol*, since Maurice was active on social media from the 20th of May, which was during the recording period. Thus Maurice should have dropped out during the game show, which turned out to be true, and the *mol* never drops out. This simple reasoning was also applicable for the follow-up seasons of 'Wie is de Mol?', though it was harder to exclude players as *mol*, since the cast of 'Wie is de Mol?' is aware of this method. Hence early dropouts are mostly inactive on social media until the recording period is over or let others post things on social media for them. Nevertheless this idea is used by the Moldel, where players are excluded manually as being the *mol* if there is clear evidence that they were not present during the entire recording period of 'Wie is de Mol?'. What is clear evidence is a bit ambiguous here, but just posts or uploading pictures on social media is not considered as clear evidence, because these pictures could have been taken earlier and these posts could have been posted by someone else. Whether evidence is considered to be clear evidence depends much on insight rather than rules. Examples of clear evidence that player p_i can be excluded as *mol* are:

– Leaked pictures of the 'Wie is de Mol?' recording, where player p_i is missing.

- Another television or radio shows that was recorded during the recording period of ’Wie is de Mol?’ in which player p_i participated.
- Evidence that player p_i was in the Netherlands during the recording period, since ’Wie is de Mol?’ is always recorded in foreign countries.

If such evidence is discovered then the *mol* likelihood ρ_i of player p_i is set to 0. After which all *mol* likelihoods are *normalized* resulting in the final prediction.

6 RESULTS

How does the Moldel perform in predicting the *mol*? In Section 6.1 of this chapter is explained which approach is used to evaluate the Moldel and in Section 6.2 the results of this approach are presented.

6.1 Approach

The Moldel has a total of 17 seasons with data, which are seasons 5 up to *21*. Though unfortunately, seasons 5 up to 8 have too few data to evaluate the Moldel on. Therefore there is a total of 13 seasons available for evaluating the Moldel. For these 13 seasons it can be decided whether they should be used as training or validation data, for which there are three general validation schemes:

Classical Validation The classical validation scheme has a fixed set of training data and a fixed set of validation data [26, section 11.2.5]. For example the even seasons from 9 to *21* can be used as training data and the odd seasons from this range can be used as validation data. The benefit of this scheme is that training data is never mixed with validation data and thus there is a low risk of overfitting. However a major drawback of this scheme is that there is a shortage of either training data or validation data. This is especially problematic for the Appearance Layer which makes predictions and can only use training data starting from season 13.

Strict Validation The strict validation scheme validates the performance of the Moldel on every season in the range of 9 up to *21* using the seasons before as training data. The benefit of this scheme is that the performance of the Moldel is realistically evaluated as would have happened at that time. Moreover 'Wie is de Mol?' episodes sometimes refer back to the *mol* of previous seasons, so with this scheme you prevent information being leaked about the validation season by the training seasons. Though data gathered about 'Wie is de Mol?' by the Moldel for seasons does not contain references back to previous seasons. Moreover with this scheme there is again a shortage of training data, except for the last seasons.

Cross Validation The cross validation scheme validates every season in the range of 9 up to *21* using the other seasons and the season from 5 up to 8 as training data [26, section 11.2.4], including seasons that happen later. For example to evaluate season 15, seasons 5 up to 14 and season 16 up to *21* are used as training data. The benefit of this scheme is that there is no shortage of training data and validation seasons. But a drawback of this scheme is that cross validation is the most time consuming of all validation schemes.

Based on these benefits and drawbacks, cross validation is chosen as validation scheme to evaluate the Moldel, since small training samples are the biggest issue of the Moldel. For evaluation of the Moldel multiple configurations are evaluated on seasons 9 up to *21*, because we are not only interested in the performance of the entire Moldel, but also in the performance of individual layers. All these evaluated configurations are shown in Table 6.1, including the lay-

Configurations	Exam Drop Layer	Exam Pass Layer	Wikipedia Layer	Appearance Layer	Aggregation	Social Media Exclusions
Full Moldel	x	x	x	x	x	x
Plus Moldel	x	x	x	x	x	
Min Moldel	x	x	x		x	
Exam Drop	x					
Exam Pass		x				
Wikipedia			x			
Appearance				x		
Uniform						

Table 6.1: All evaluation configurations

ers enabled in these configurations. Note that no layer is enabled in the Uniform configuration, which assigns an uniform probability to all *potential mol* players (see Equation 1.1). This configuration is included to represent how well uniform guessing performs and is used to show that the other configurations perform better than uniform guessing.[1]

When evaluating these configurations, predictions are obtained which are encoded similar as in Section 5.3. These predictions are obtained for a set of validation pairs $(s, e) \in I$, where s is the season for which the prediction is made and e is the episode after which the prediction is made. And a prediction for a pair (s, e) results in the *mol* likelihoods $\rho_{s,e,p}$ for all $p \in P_s$, where P_s is the set of all players of season s (including non-*potential mol* players). The *mol* likelihoods as output of these predictions are evaluated using evaluation metrices on groups, where the predictions are grouped on:

– The number of *potential mol* players in the range from 10 down to 2, where only the latest episode e of every season is selected with that number of *potential mol* players.[2]

– Episode numbers in the range from 0 to 9, where episode number 0 represents the prediction before the first episode has even been broadcast. For seasons that did not have exactly 9 episodes before the showdown episode, e.g. season 11 and *21*, linear interpolation is used, where episode numbers e in the range from 0 to 9 for season s are rescaled with function $E(s, e)$ to:

$$E(s, e) = \text{round}\left(\frac{e \cdot |E_s|}{9}\right)$$

The round function in this function definition rounds the value to the closest integer and $|E_s|$ is the maximum episode of season s. For example the episode numbers for season *21* with 7 episodes, i.e. $|E_{21}| = 7$, are mapped to:

$$\{0 \to 0,\ 1 \to 1,\ 2 \to 2,\ 3 \to 2,\ 4 \to 3,\ 5 \to 4,\ 6 \to 5,\ 7 \to 5,\ 8 \to 6,\ 9 \to 7\}$$

For choosing the appropriate evaluation metrices, it should be remarked that they have to handle the stochastic nature of the Moldel, i.e. the output of the Moldel are not labels, but probabilities. So confusion matrices, accuracy, precision and recall are not applicable for the Moldel, since

[1]More about this can be found in Chapter 7 in which it is indeed shown that the Moldel performs significantly better than uniform guessing.

[2]In the finals of season 9 there was 1 *potential mol* player, this result is not included in the grouping.

these evaluation metrices assume that a label is given as output. The following evaluation metrices can handle likelihoods rather than labels as output and are therefore used to evaluate the different configurations:

Log Loss The Log Loss score [28] is the average log likelihood by the Moldel over all actual labels, which is computed as:

$$-\frac{1}{n}\sum_{(s,e)\in I}\sum_{p\in P_s}\mathbb{1}(p=\text{Mol}_s)\cdot\text{ln}(\rho_{s,e,p})+\mathbb{1}(p\neq\text{Mol}_s)\cdot\text{ln}(1-\rho_{s,e,p})$$

where Mol_s is the *mol* of season s, $\mathbb{1}$ is the identity function which is equal to 1 if the condition holds else it is equal to 0 and n is defined as:

$$n=\sum_{(s,e)\in I}\sum_{p\in P_s}1$$

A lower Log Loss score implies that the model performs better in classifying both *mol* players as non-*mol* players.

Mol Log Loss The Mol Log Loss is the average log likelihood by the Moldel over all *mol* players, which is computed as:

$$-\frac{1}{|I|}\sum_{(s,e)\in I}\text{ln}(\rho_{s,e,\text{Mol}_s})$$

where $|I|$ is the number of identifiers on which the configurations are evaluated. The difference between Log Loss and Mol Log Loss is that the average in the Mol Log Loss is only taken over all *mol* players, whereas in the Log Loss the average is taken over all players. A similarity between both scores is that a lower Mol Log Loss score also implies that the model performs better, but only on *mol* players.

Concordant-Discordant Ratio To compute the Concordant-Discordant Ratio [28], we first define the following two sets:

$$M=\{(s,e,p)\mid(s,e)\in I\wedge p=\text{Mol}_s\}\qquad N=\{(s,e,p)\mid(s,e)\in I\wedge p\in P_s\setminus\{\text{Mol}_s\}\}$$

where M is the set of all identifiers (s,e) combined with a *mol* player from that season s and N is the set of all identifiers (s,e) combined with a non-*mol* player from that season s. The number of concordant pairs C and the number of discordant pairs D are now defined as:

$$C=\sum_{\substack{(s,e,p),\\(s',e',p')\\\in M\times N}}\mathbb{1}(\hat{\rho}_{s,e,p}>\hat{\rho}_{s',e',p'})\qquad D=\sum_{\substack{(s,e,p),\\(s',e',p')\\\in M\times N}}\mathbb{1}(\hat{\rho}_{s,e,p}<\hat{\rho}_{s',e',p'})$$

where $\hat{\rho}_{s,e,p}=|P_{s,e}|\cdot\rho_{s,e,p}$ which is done to make likelihoods comparable and $M\times N$ is the Cartesian product between set M and N. The Concordant-Discordant Ratio is defined as:

$$\frac{C}{C+D}$$

which is defined as $1/2$ if $C+D=0$. A higher Concordant-Discordant Ratio means that the model is more consistent, i.e. *mol* players get a higher likelihood than non-*mol* players.

Mean Mol Likelihood The Mean Mol Likelihood score is the average over all *mol* likelihoods, which is computed as:

$$\frac{1}{|I|}\sum_{(s,e)\in I}\rho_{s,e,\text{Mol}_s}$$

The Mean Mol Likelihood is quite similar to Mol Log Loss, however the Mean Mol Likelihood is an arithmetic mean and the Mol Log Loss is equivalent to the geometric mean over the *mol* likelihoods. A higher Mean Mol Likelihood score implies a better model.

Mean Mol Rank To compute the Mean Mol Rank, $r_{s,e,p}$ is defined as the rank of the players likelihood for prediction of season s after episode e. If player p has the highest *mol* likelihood of all players then $r_{s,e,p} = 1$, whereas if player p has the lowest *mol* likelihood of all players then $r_{s,e,p} = |P_s|$. In case of ties the rank $r_{s,e,p}$ is defined as the average of the ranks. An example of ranks is defined below:

Example 8. *The respective ranks for the following* mol *likelihoods are:*

Likelihoods	0.1	0.3	0.0	0.0	0.1	0.0	0.1	0.2	0.0	0.2
Ranks	5	1	8.5	8.5	5	8.5	5	2.5	8.5	2.5

Table 6.2: Rank example for *mol* likelihoods

The Mean Mol Rank is computed as:

$$\frac{1}{|I|} \sum_{(s,e) \in I} r_{s,e,\text{Mol}_s}$$

A lower Mean Mol Rank score means that *mol* players get a higher likelihood than non-*mol* players. The Mean Mol Rank is therefore similar to Concordant-Discordant Ratio, though in Concordant-Discordant Ratio different predictions are compared which does not happen in the Mean Mol Rank.

6.2 Evaluation

In this section the outcome of the evaluation metrices, as defined in Section 6.1, are presented in visual format (the raw format can be found in Appendix E). The visual presentation of the evaluation metrices shown by Figures 6.1 up to 6.10, which shows all scores of the evaluation metrices as bar charts sorted per group in increasing/decreasing order. With Figures 6.1 up to 6.10 the different configurations can be compared and their progress can be analysed as the time passes. [3] Last of all there are three figures included at the end of this section, which show the best predictions, worst predictions and final predictions of the Full Moldel configuration for seasons 9 up to *21* as pie charts, where the percentages represent the *mol* likelihoods. All *potential mol* players are displayed in these plots even if their likelihoods are close to zero or zero (due to Social Media Exclusion), whereas non-*potential mol* are hidden in these plots. For more predictions of any configurations for season 9 up to *21* you can visit:[4]

`https://github.com/Multifacio/Moldel/tree/master/results/Thesis%20Results/Latest%20Thesis%20Results%20(19-03-2021)`

[3] Note that a worse score does not imply that a configuration performs worse than another configuration. Configurations are evaluated on a different range of seasons.

[4] Which also includes predictions where not all layers of that configuration could be used.

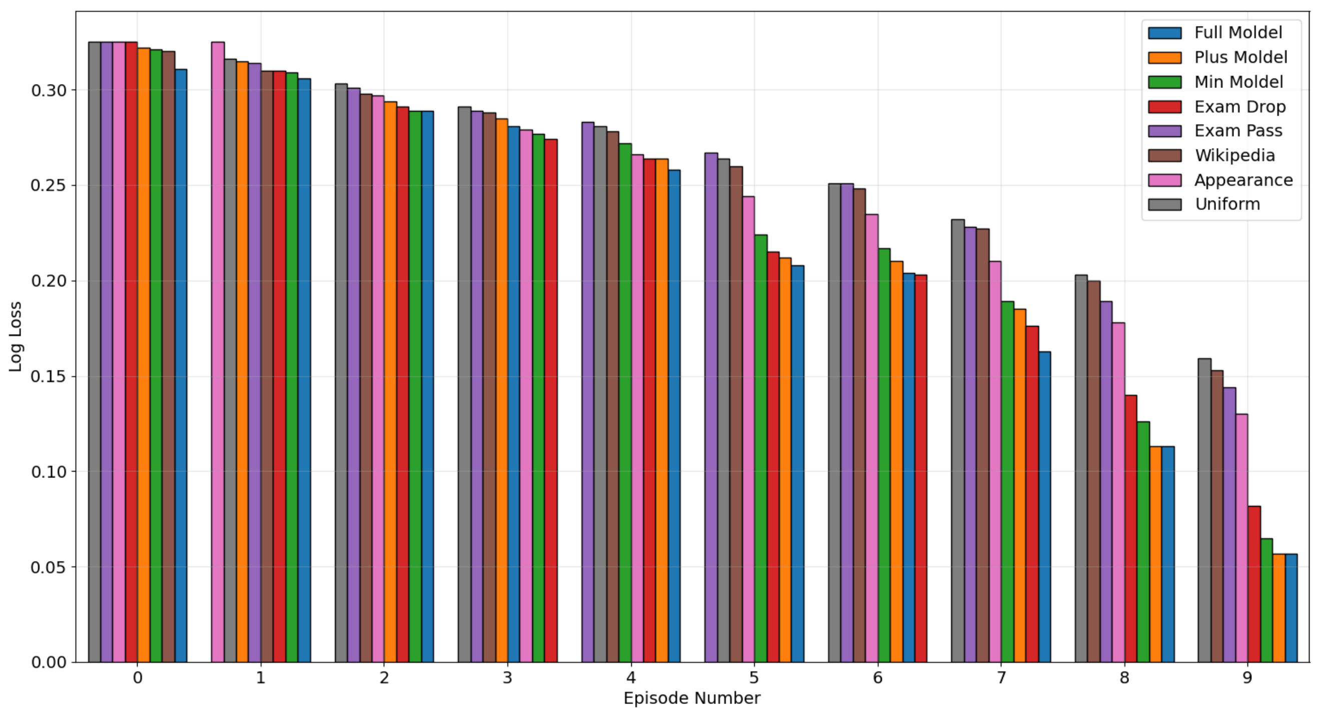


Figure 6.1: Log Loss for configurations grouped by episode number

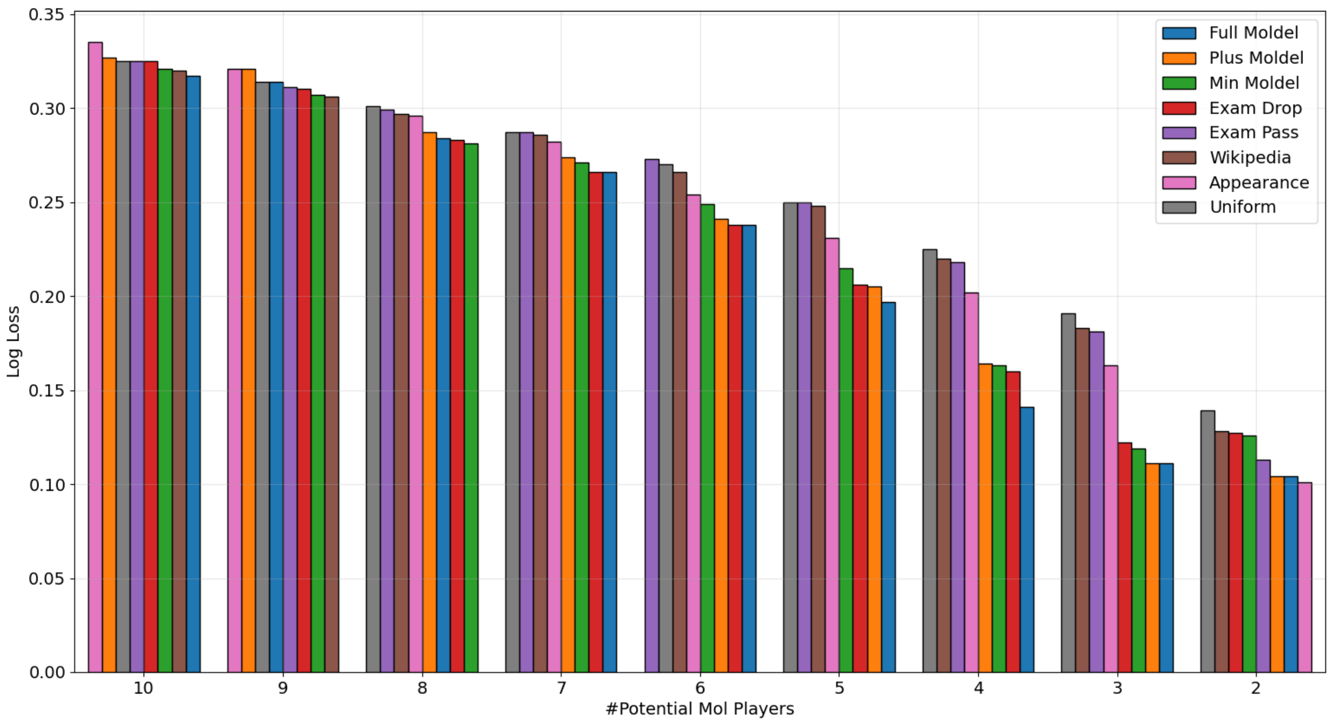


Figure 6.2: Log Loss for configurations grouped by number of *potential mol* players

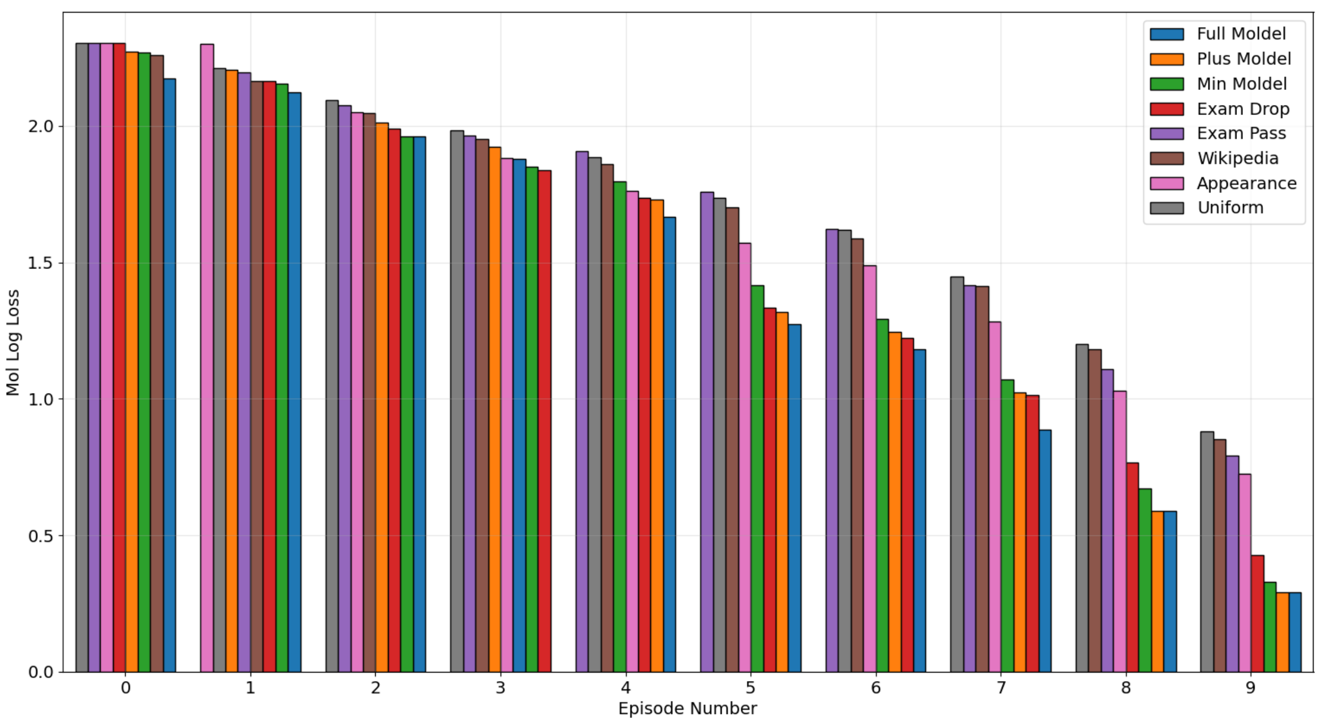


Figure 6.3: Mol Log Loss for configurations grouped by episode number

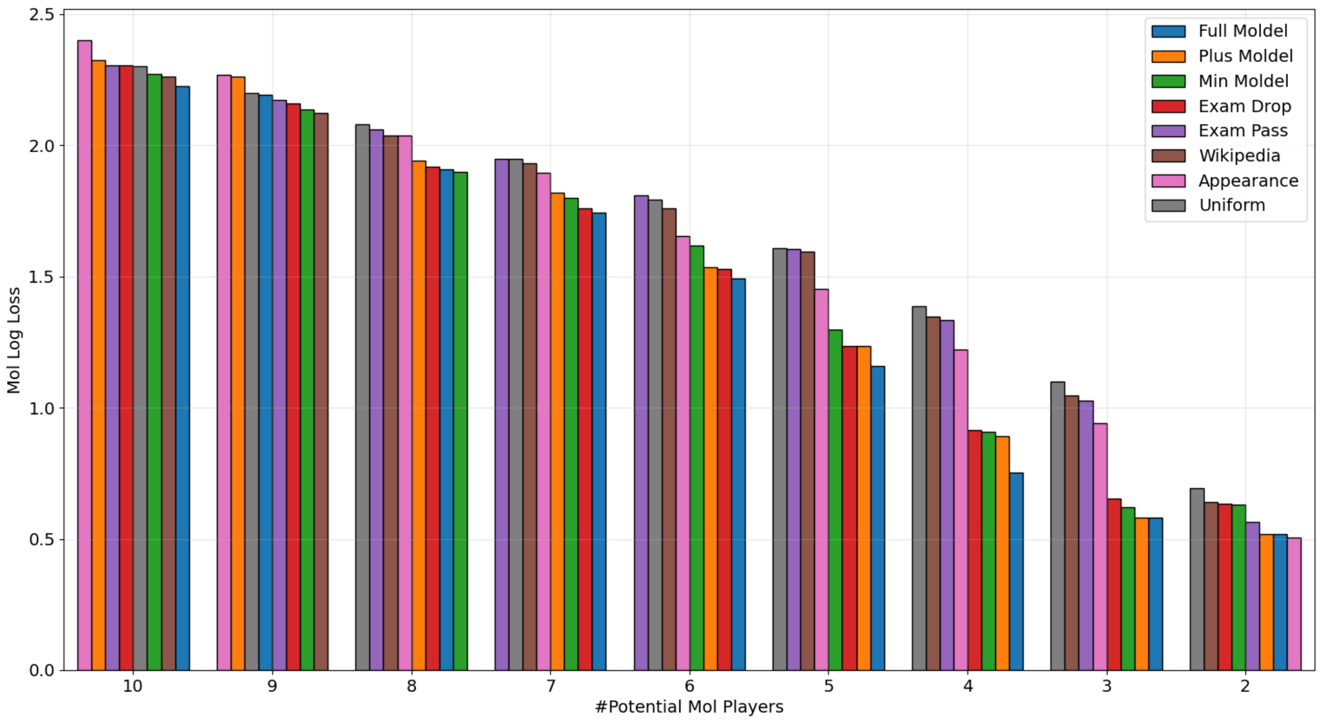


Figure 6.4: Mol Log Loss for configurations grouped by number of *potential mol* players

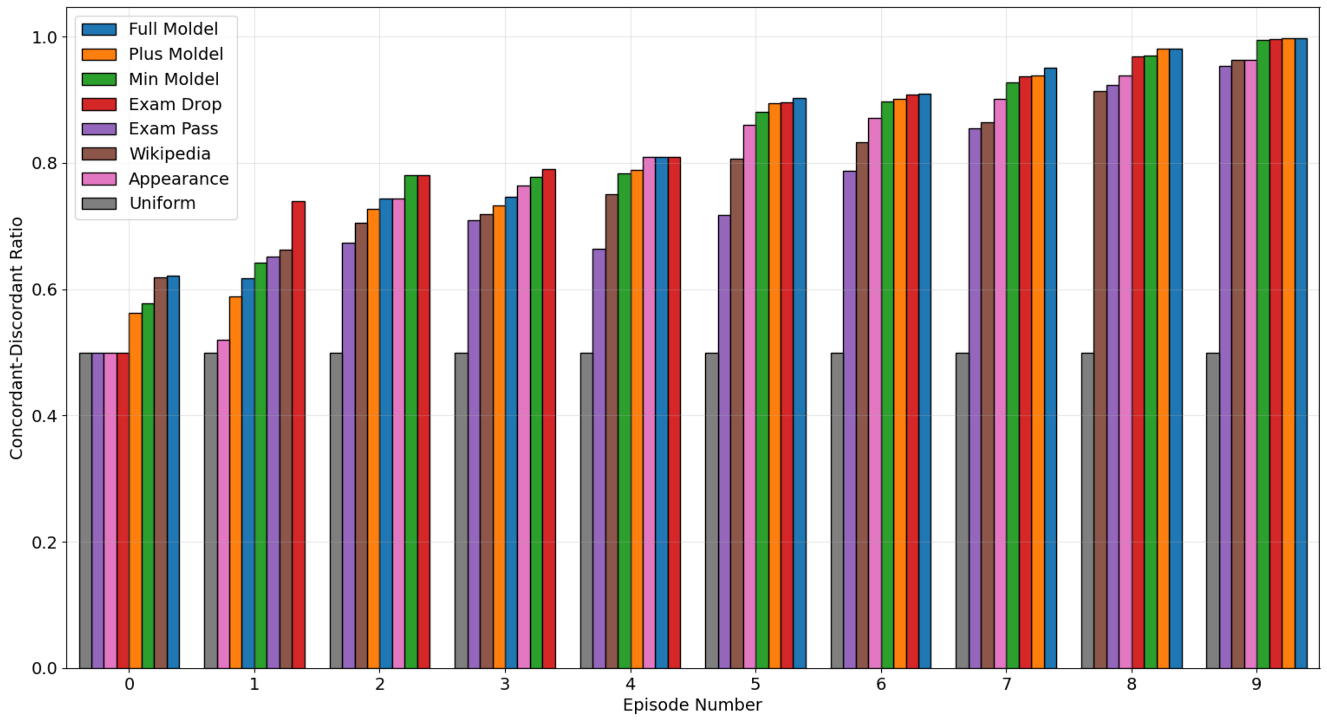


Figure 6.5: Concordant-Discordant Ratio for configurations grouped by episode number

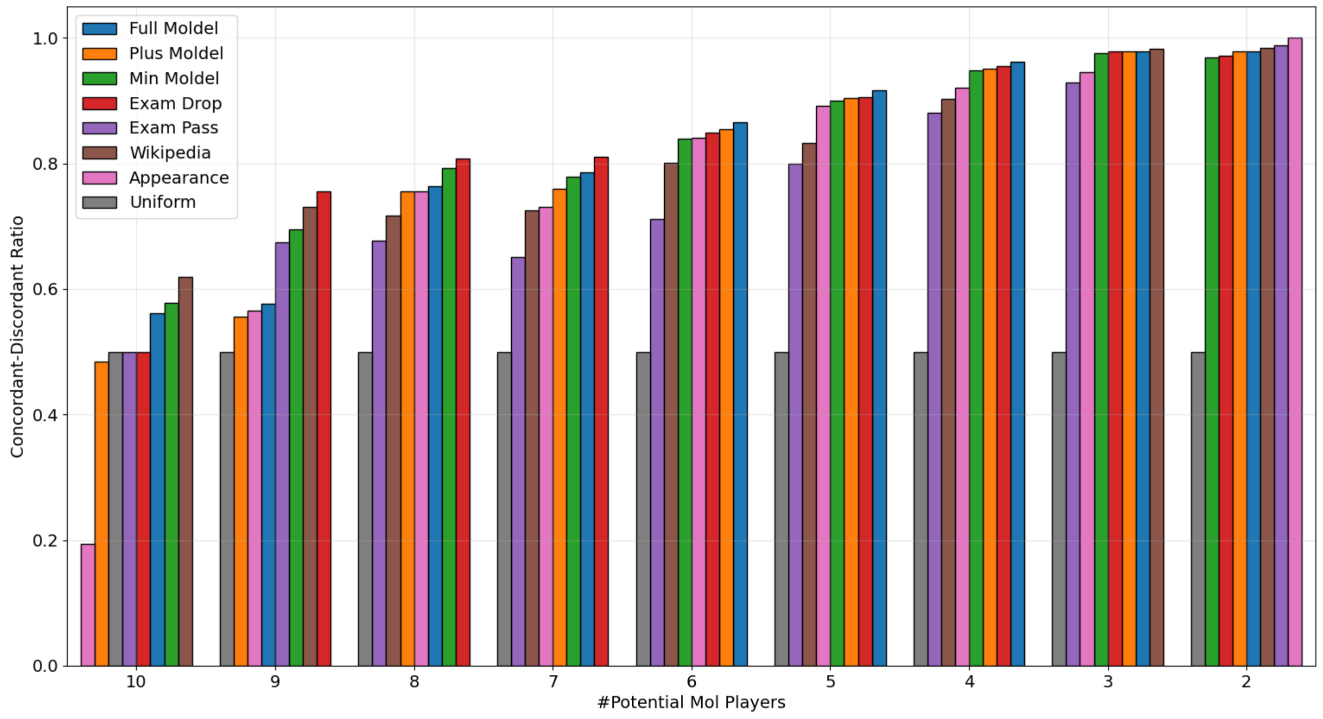


Figure 6.6: Concordant-Discordant Ratio for configurations grouped by number of *potential mol* players

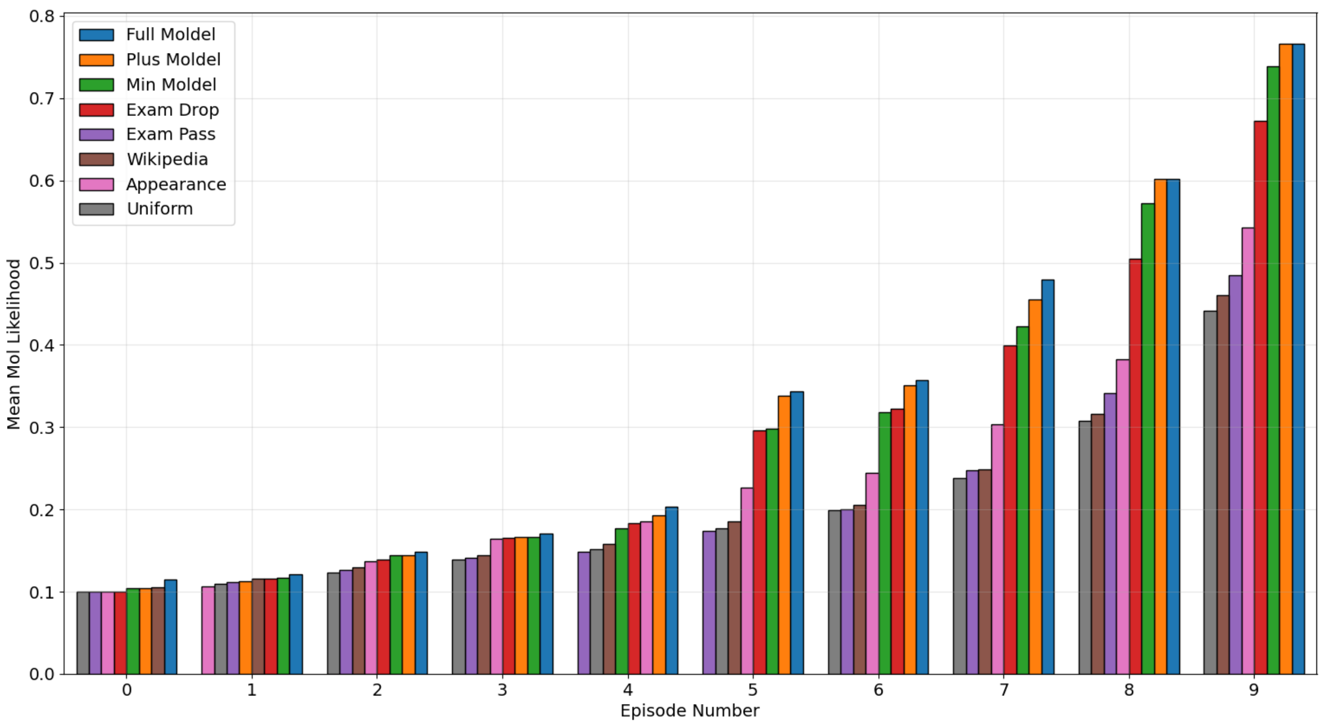


Figure 6.7: Mean Mol Likelihood for configurations grouped by episode number

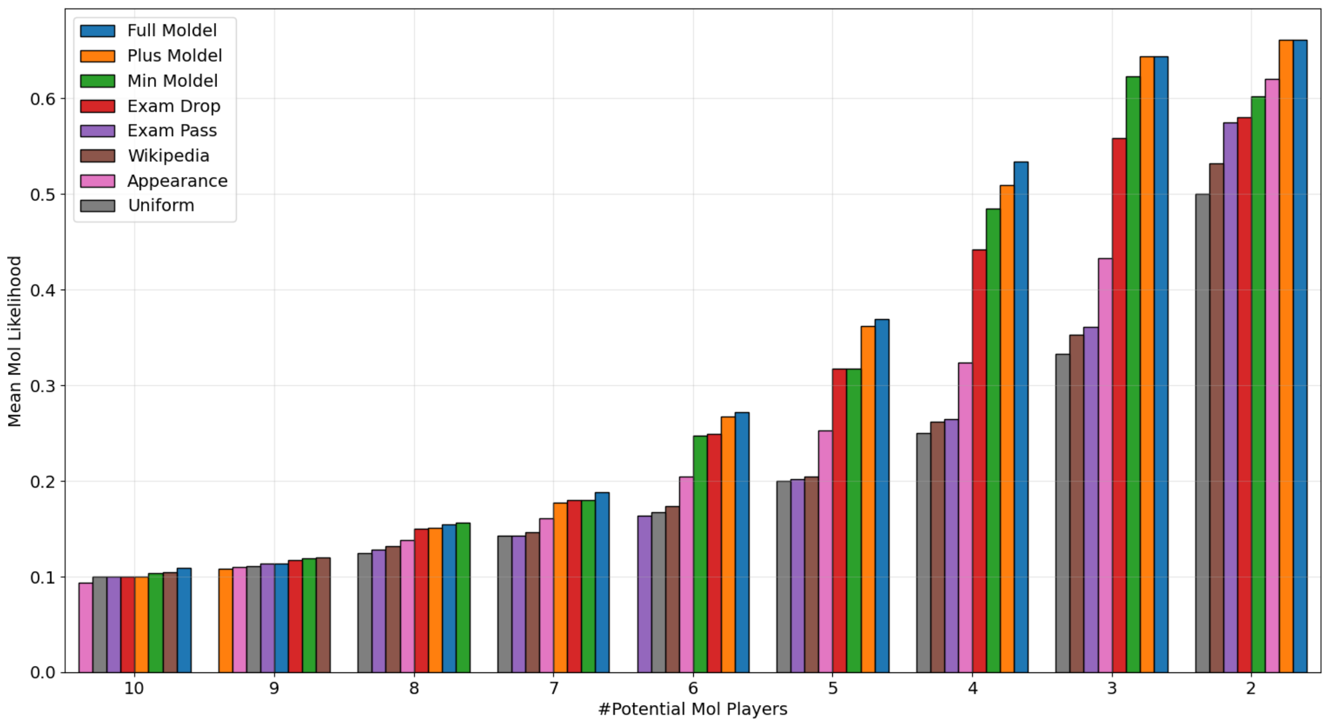


Figure 6.8: Mean Mol Likelihood for configurations grouped by number of *potential mol* players

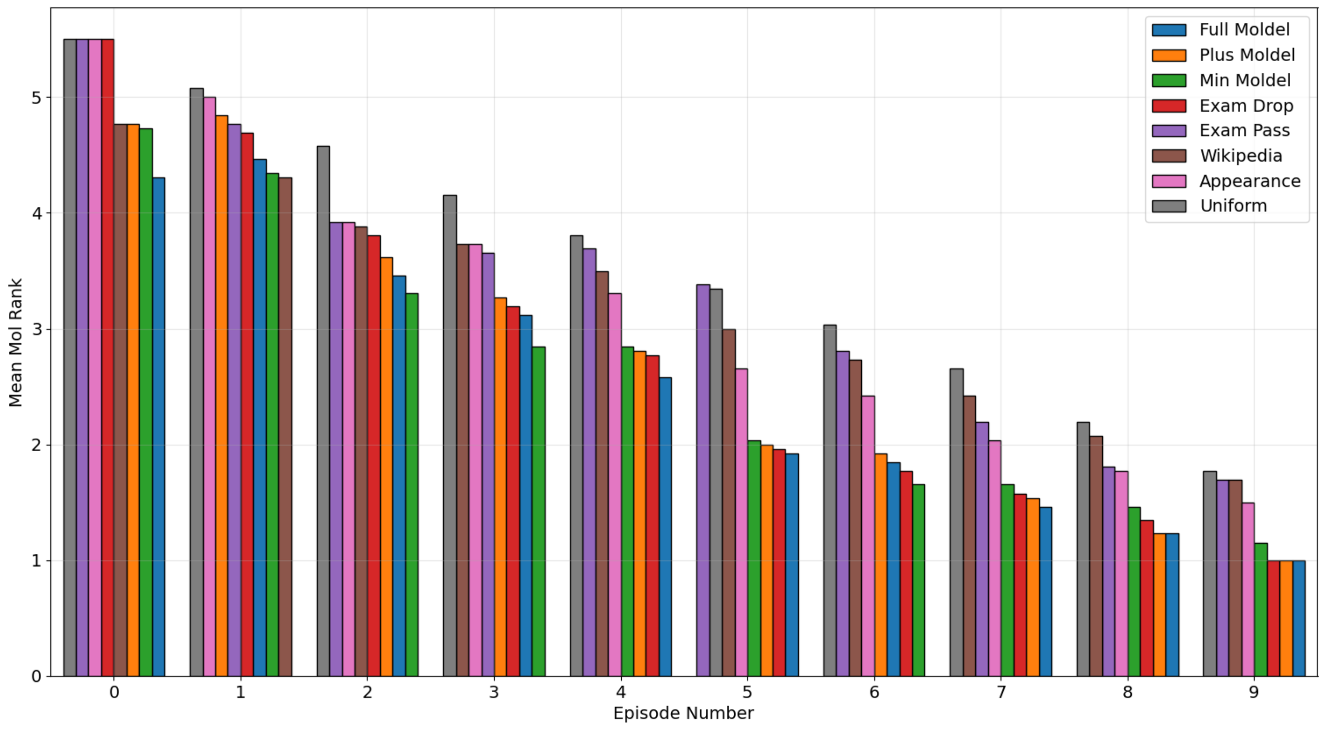


Figure 6.9: Mean Mol Rank for configurations grouped by episode number

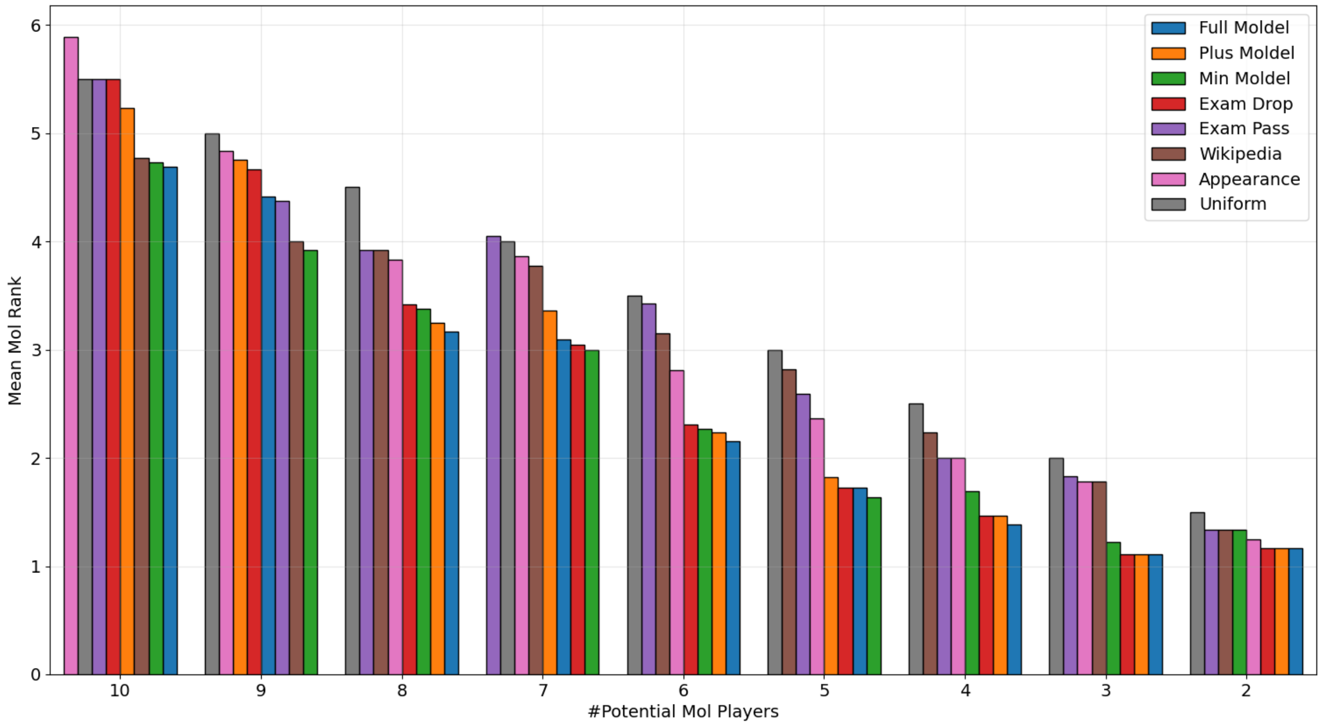


Figure 6.10: Mean Mol Rank for configurations grouped by number of *potential mol* players

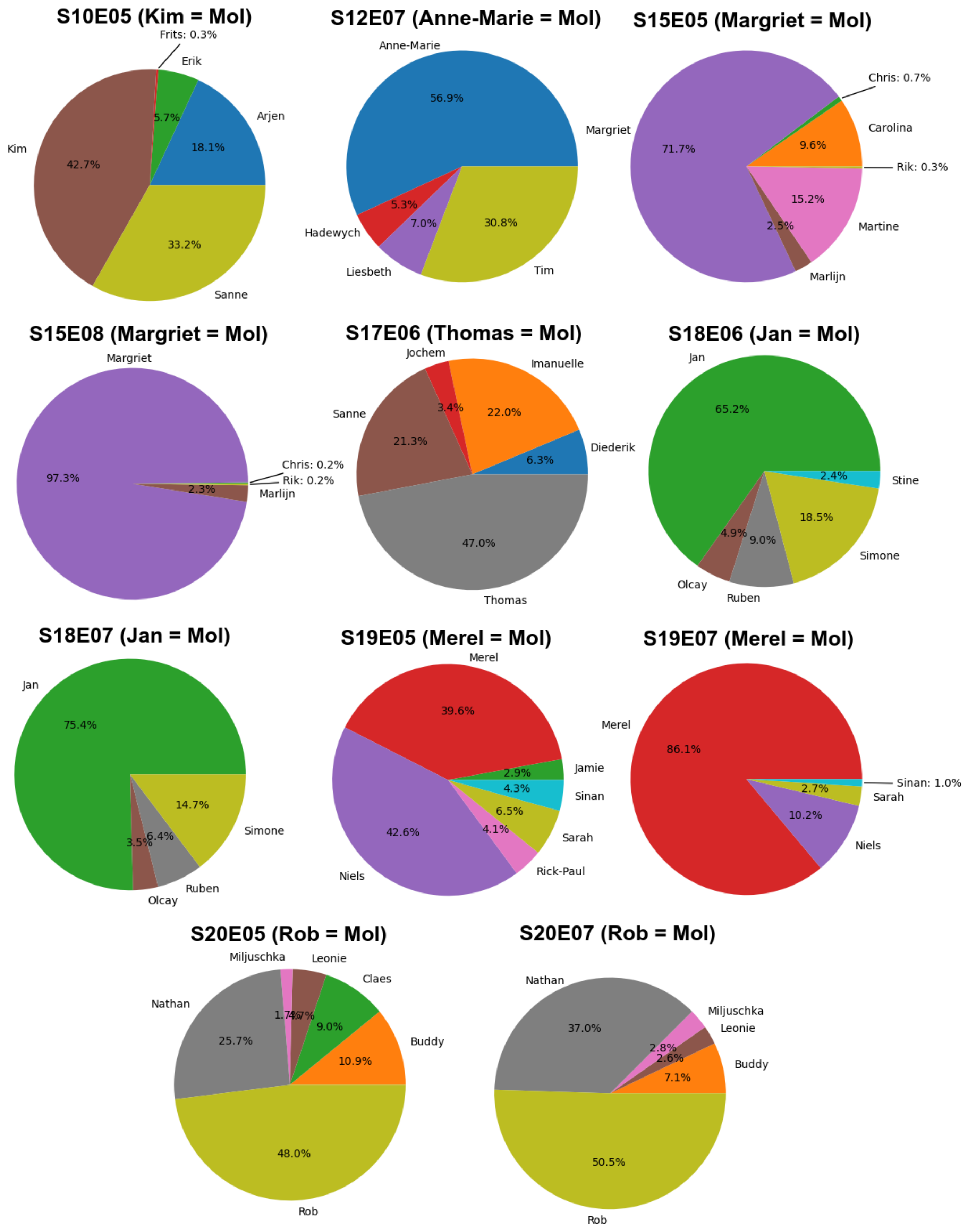


Figure 6.11: Best predictions of Full Moldel (excluding final predictions)

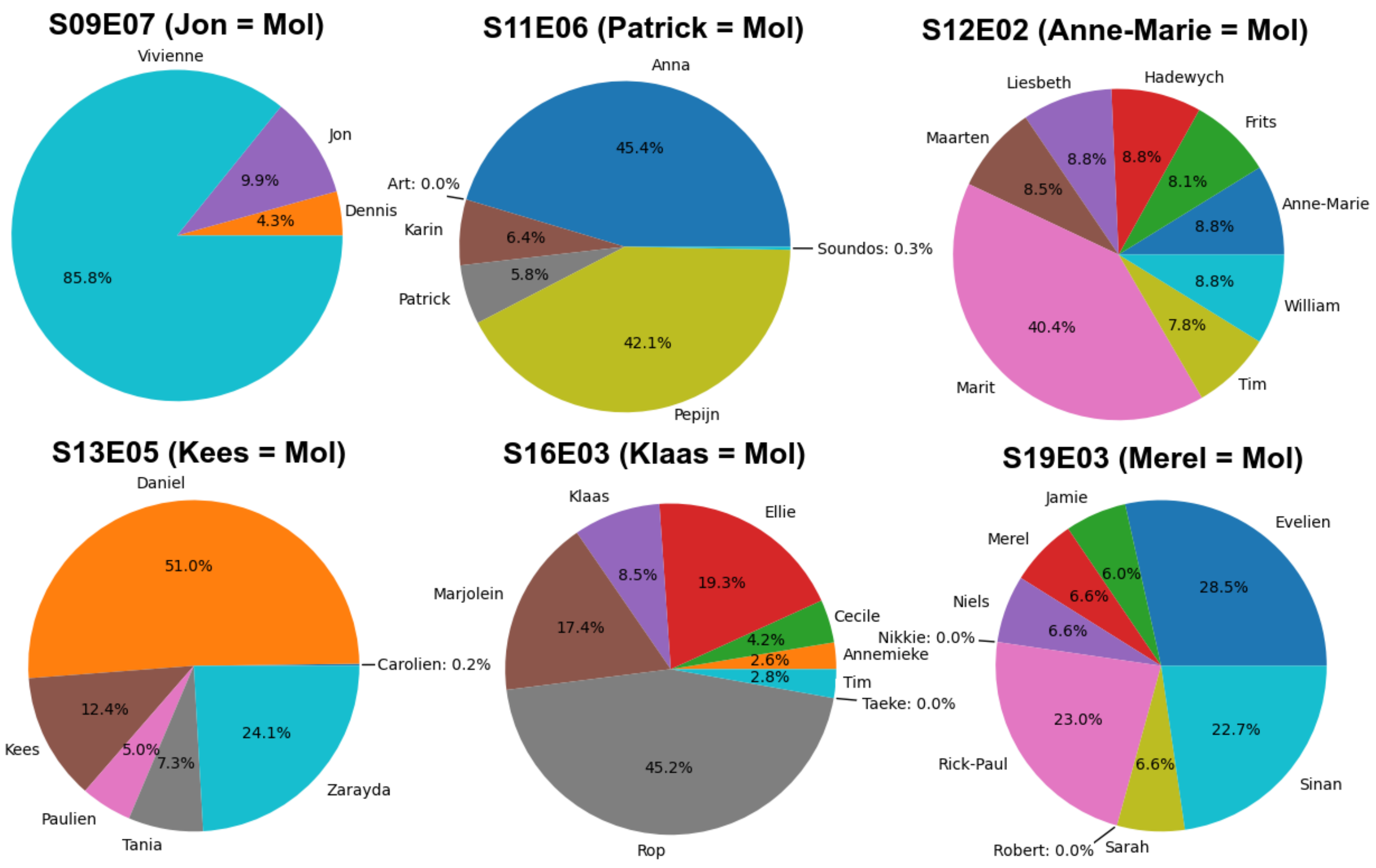


Figure 6.12: Worst predictions of Full Moldel

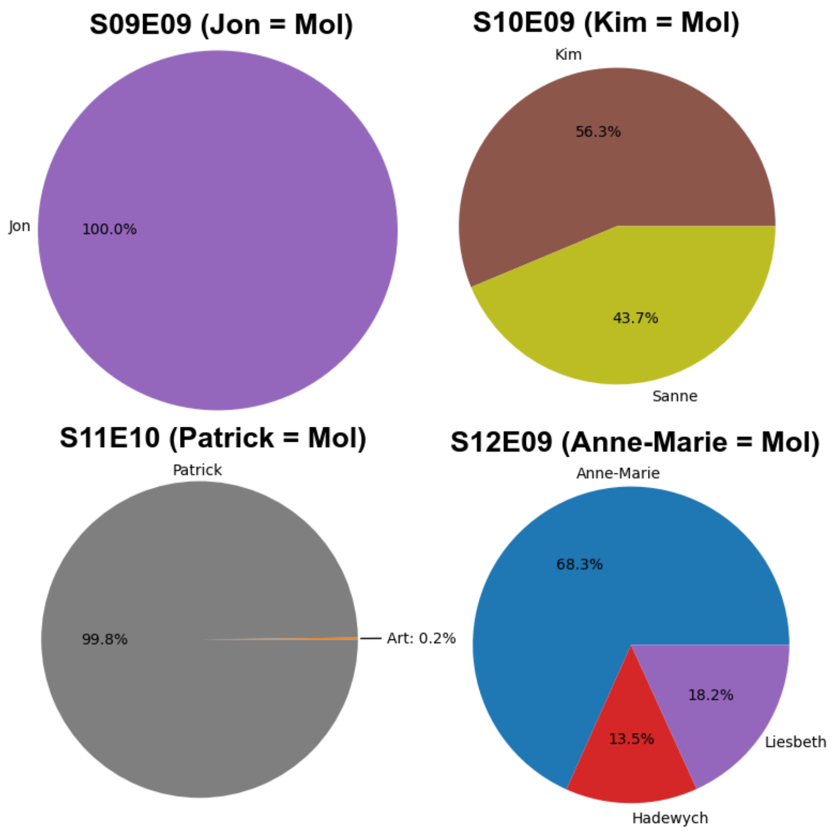


Figure 6.13: Final predictions of Full Moldel for seasons 9 up to 12

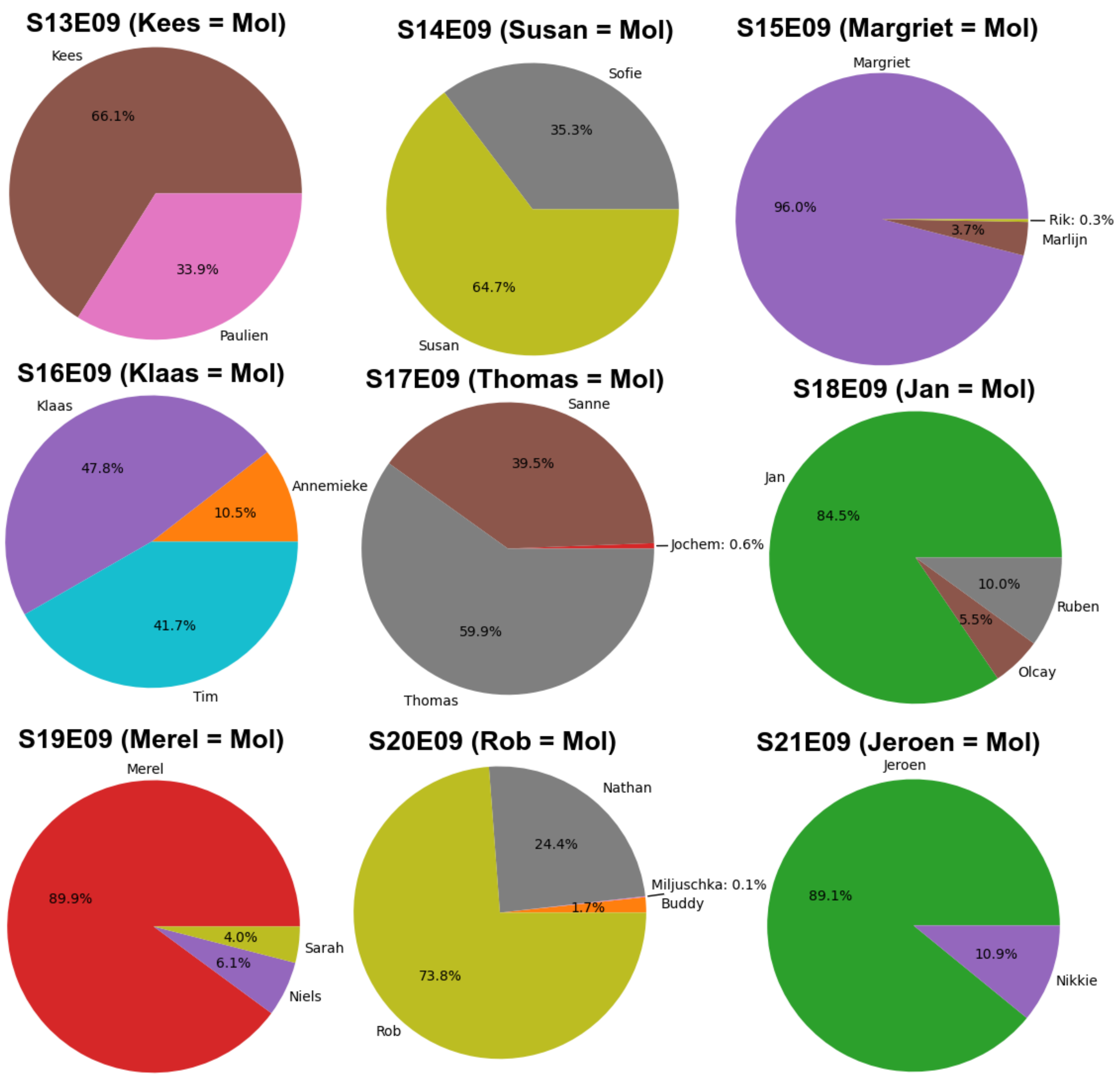


Figure 6.14: Final predictions of Full Moldel for seasons 13 up to 21

7 DISCUSSION

The results presented in Chapter 6 are promising. The scores of the Full Moldel configuration and other non-Uniform configurations shown by Figures 6.1 up to 6.10 are better than the Uniform configuration. Additionally the Full Moldel, Plus Moldel, Min Moldel and Exam Drop configurations also show consistency, i.e. the scores of those configurations diverges from the Uniform configuration as more episodes have been seen or as the number of *potential mol* decreases. Remarkable as well is that the Full Moldel, Plus Moldel and Exam Drop configurations have a Mean Mol Rank of 1.0 for the finals, see the column for episode 9 in Figure 6.9. This implies that the Moldel has always given the actual *mol* the highest likelihood in the finals, which is also confirmed by Figures 6.13 & 6.14. Likewise these configurations and the Min Moldel configuration gave the *mol* on average a likelihood higher than 0.5 after episode 8 or with at most 3 *potential mol* players, see Figures 6.7 & 6.8.

Furthermore some characteristics of the individual layer configurations, i.e. Exam Drop, Exam Pass, Wikipedia and Appearance, are also recognizable in the results. For example the Wikipedia configuration has a slight advantage over the Uniform configuration before the first episode has even been broadcast, but has a lesser advantage over the Uniform configuration compared to the Exam Drop & Appearance configurations for later episodes. This is expected, because the Wikipedia Layer is a pre-layer which has all its information available before the first episode that does not grow when new episodes are broadcast. On the other hand the information by the Exam Drop and the Appearance layer grows as more episodes are broadcast hence these configurations become much more accurate than the Wikipedia configuration. It is therefore not surprising that the Exam Drop configuration has the best scores compared to the other individual layer configurations, since the Exam Drop layer uses all episodes as data until the finals, whereas the Appearance layer only uses episodes up to 5 as data. Nevertheless the second best configuration is the Appearance configuration, which is followed by the Wikipedia configuration. The Exam Pass configuration is the worst performing individual configuration, which is also understandable since based on its logic one is not much less likelier to be the *mol* when using a *vrijstelling* or *jokers* in the early episodes. Hence the Exam Pass configuration is expected to start diverging from the Uniform configuration in later episodes. This is confirmed by Figures 6.1 up to 6.10, which show that the score of the Exam Pass configuration starts diverging from the Uniform configuration after episode 7 or with less than 4 *potential mol* players.

Thus both the individual configurations, i.e. Exam Drop, Exam Pass, Wikipedia and Appearance, and the advanced configurations, i.e. Full Moldel, Plus Moldel and Min Moldel, perform quite well. Nevertheless these configurations are nowhere near perfect. This is also illustrated by Figure 6.12. The Moldel can be completely wrong with some of its predictions and might have no idea about the *mol* in early episodes, which is emphasized by Figures 6.1 & 6.3 that show similar (Mol) Log Loss scores for both the Full Moldel and the Uniform configuration. Furthermore the Mean Mol Likelihood of the Full Moldel configuration before episode 5 is indeed higher than the Mean Mol Likelihood of the Uniform configuration, however it is also at most 0.2 meaning that the actual *mol* has a 4 times larger likelihood of not being the *mol*. Likewise the Mean Mol Rank after episode 6 for the Full Moldel configuration is around 2 implying that there is on average at least one non-*mol* player with a higher *mol* likelihood than the actual *mol*. So the Moldel also has some drawbacks and therefore it is important to statistical test whether the Moldel and its layers indeed perform significantly better than uniform guessing.

7.1 Significance Testing

To show that the Moldel and its layers perform significantly better than uniform guessing, the Plus Moldel and the individual configurations are statistically tested with regard to the Uniform configuration. For statistical testing, both configurations are evaluated on season 9 up to *21* where the likelihoods given to the actual *mol* of both configurations are pairwise compared which are grouped on episode numbers and number of *potential mol* players, similar as in Chapter 6. For every group the likelihood samples X and Y of both configurations are then compared using the following tests:

- Paired Student T-Test [29], which is a parametric test that assumes the differences between both configurations $X - Y$ is independently and normally distributed, i.e.

  $$X - Y \sim \mathcal{N}(\mu, \sigma)$$

  The null hypothesis and alternative hypothesis for this test are respectively:

  $$H_0 : \mu = 0 \qquad\qquad H_1 : \mu > 0$$

  where the null hypothesis represents that both configurations have the same performance and the alternative hypothesis represents that the first configuration performs significantly better than the second configuration.

- Log Paired Student T-Test, which is a Paired Student T-Test applied on the logarithm of the likelihoods in both samples X and Y.

- Wilcoxon Signed Rank Test [30], which is a non-parametric test that assumes X and Y are independently and identically distributed. This test assigns a rank to every likelihood with respect to all likelihoods in X and Y, where the lowest likelihood is assigned a rank of 1 and the highest likelihood is assigned a rank of $|X| + |Y|$. In case of ties the average of the tied ranks is assigned to the tied likelihoods. The goal of the Wilcoxon Signed Rank Test is to check if the mean μ_X of the ranks of X is significantly higher than the mean μ_Y of the ranks of Y. The null hypothesis and alternative hypothesis for this test are respectively:

  $$H_0 : \mu_X = \mu_Y \qquad\qquad H_1 : \mu_X > \mu_Y$$

  where the null hypothesis again represents that both configurations have the same performance and the alternative hypothesis represents that the first configuration performs significantly better than the second configuration.

The difference between the Paired Student T-Tests and the Wilcoxon Signed Rank Test is that the former test requires the difference to be normally distributed, whether the latter test does not require the difference to be normally distributed. Moreover if the normality assumption is justified then the Paired Student T-Test is stronger than the Wilcoxon Signed Rank Test. Tables 7.1 up to 7.10 contain the results of these tests, i.e. the p-values of these tests. Results in the tables are green if they pass the test, i.e. have a p-value smaller than or equal to 0.05, and are red otherwise.

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Paired Student T	0.166	0.283	0.064	0.098	0.034	0.008	0.010	0.006	<1e-3	<1e-3
Log Paired Student T	0.265	0.459	0.209	0.339	0.111	0.033	0.048	0.053	<1e-3	<1e-3
Wilcoxon Sign. Rank	0.084	0.137	0.108	0.137	0.040	0.016	0.016	0.007	0.002	<1e-3
Sample Size	13	13	13	13	13	13	13	13	13	13

Table 7.1: Plus Moldel versus Uniform p-values grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Paired Student T	0.501	0.650	0.044	0.075	0.037	0.016	0.003	0.004	0.106
Log Paired Student T	0.629	0.765	0.091	0.192	0.104	0.070	0.021	0.027	0.238
Wilcoxon Sign. Rank	0.271	0.455	0.055	0.103	0.064	0.026	0.005	0.006	0.109
Sample Size	13	12	12	11	13	11	13	9	6

Table 7.2: Plus Moldel versus Uniform p-values grouped by number of *potential mol* players

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Paired Student T	0.500	0.120	0.016	0.026	0.031	0.004	0.005	0.004	0.004	<1e-3
Log Paired Student T	0.500	0.130	0.016	0.041	0.048	0.009	0.005	0.004	0.006	<1e-3
Wilcoxon Sign. Rank	0.500	0.007	0.005	0.040	0.029	0.004	0.003	0.004	0.009	<1e-3
Sample Size	13	13	13	13	13	13	13	13	13	13

Table 7.3: Exam Drop versus Uniform p-values grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Paired Student T	0.500	0.181	0.014	0.029	0.027	0.010	0.003	0.004	0.231
Log Paired Student T	0.500	0.195	0.011	0.044	0.051	0.009	0.002	0.008	0.390
Wilcoxon Sign. Rank	0.500	0.013	0.007	0.034	0.040	0.007	0.004	0.006	0.156
Sample Size	13	12	12	11	13	11	13	9	6

Table 7.4: Exam Drop versus Uniform p-values grouped by number of *potential mol* players

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Paired Student T	0.500	0.177	0.164	0.294	0.749	0.689	0.420	0.083	0.012	0.029
Log Paired Student T	0.500	0.192	0.169	0.303	0.795	0.773	0.511	0.123	0.015	0.032
Wilcoxon Sign. Rank	0.500	0.010	0.211	0.172	0.712	0.751	0.363	0.104	0.020	0.047
Sample Size	13	13	13	13	13	13	13	13	13	13

Table 7.5: Exam Pass versus Uniform p-values grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Paired Student T	0.500	0.093	0.186	0.463	0.706	0.378	0.049	0.064	0.049
Log Paired Student T	0.500	0.102	0.202	0.531	0.755	0.438	0.057	0.073	0.056
Wilcoxon Sign. Rank	0.500	0.013	0.291	0.688	0.751	0.282	0.073	0.082	0.078
Sample Size	13	12	12	11	13	11	13	9	6

Table 7.6: Exam Pass versus Uniform p-values grouped by number of *potential mol* players

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Paired Student T	0.097	0.100	0.096	0.174	0.162	0.117	0.159	0.106	0.195	0.149
Log Paired Student T	0.165	0.134	0.122	0.234	0.260	0.190	0.212	0.212	0.303	0.255
Wilcoxon Sign. Rank	0.084	0.084	0.084	0.073	0.108	0.122	0.080	0.066	0.403	0.416
Sample Size	13	13	13	13	13	13	13	13	13	13

Table 7.7: Wikipedia versus Uniform p-values grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Paired Student T	0.097	0.016	0.110	0.296	0.136	0.273	0.096	0.057	0.182
Log Paired Student T	0.165	0.030	0.174	0.385	0.202	0.356	0.142	0.054	0.225
Wilcoxon Sign. Rank	0.084	0.017	0.117	0.160	0.122	0.183	0.091	0.296	0.165
Sample Size	13	12	12	11	13	11	13	9	6

Table 7.8: Wikipedia versus Uniform p-values grouped by number of *potential mol* players

Episode Number	0	1	2	3	4	5	6	7	8	9
Paired Student T	0.500	0.718	0.074	0.051	0.042	0.027	0.038	0.055	0.045	0.032
Log Paired Student T	0.500	0.805	0.342	0.208	0.151	0.106	0.153	0.112	0.072	0.093
Wilcoxon Sign. Rank	0.500	0.271	0.020	0.050	0.043	0.037	0.037	0.066	0.050	0.050
Sample Size	13	13	13	13	13	13	13	13	13	13

Table 7.9: Appearance versus Uniform statistical test grouped by episode number

#Potential Mol	10	9	8	7	6	5	4	3	2
Paired Student T	0.846	0.537	0.131	0.156	0.041	0.039	0.041	0.090	0.068
Log Paired Student T	0.840	0.714	0.359	0.351	0.134	0.146	0.112	0.184	0.065
Wilcoxon Sign. Rank	0.083	0.190	0.042	0.186	0.050	0.037	0.050	0.156	0.055
Sample Size	13	12	12	11	13	11	13	9	6

Table 7.10: Appearance versus Uniform statistical test grouped by number of *potential mol* players

7.2 Reflection

Tables 7.1 up to 7.10 show that some tests pass and others fail. For example the Plus Moldel configuration has a significant better performance than the Uniform configuration starting after episode 5 (see Table 7.1). Likewise the Exam Drop configuration passes all tests starting after episode 2, except for the Log Paired Student test for episode 4 (see Table 7.3). Though this is a good sign, it does not guarantee that the Moldel and the Exam Drop layer provide accurate predictions for upcoming seasons. Passing a test only implies that the configuration performed significantly better than the Uniform configuration for the past, which does not imply that the configuration has any predictive power. First of all, all validation seasons have been used in designing the Moldel, i.e. if a design failed on the validation seasons then a different design was used. And by doing so you eventually find an implementation that passes the tests which does not necessarily perform well for future seasons. Though in the process of selecting a proper design, simple designs were preferred over complex designs, even if the complex design had a better performance. Secondly, some hyperparameters are determined based on the performance on the validation seasons, e.g. the number of bins for the Exam Drop Layer, the lower/higher likelihood for the Wikipedia layer and the lower cut-off of the Appearance Layer. This also partially explains why the performance of the configurations are good. Thirdly, for the implementation of aggregation an approach was chosen which indirectly uses data from the predict season as train data (see Section 5.3). This could result in more optimistic test results, however since the prediction season is indirectly used it probably does not influence the test results a lot. Last of all, the Moldel is open source and hence it is vulnerable to so called adversary attacks. An adversary attack is that the cast of 'Wie is de Mol?' might become aware of this research and edit the episodes such that the Moldel is fooled in believing that another player is the *mol* or that the Moldel has no clue at all. This is not unlikely to happen, because after the research of van Zessen was published that dropouts were active on Social Media, the cast of 'Wie is de Mol?' encouraged all its participants to stay inactive on Social Media until the recording period was over. So this research might face the same destiny as the research of van Zessen. A solution for this is to make this project closed source, but then it is hard to convince others that these predictions are reliable.[1] Thus adversary attacks is a weakness that we unfortunately have to face.

On the other hand there are also a lot of tests that fail. For example the Wikipedia configuration barely passes any tests (see Tables 7.7 & 7.8). Though this is a bad sign, it does not imply that the configuration cannot provide accurate predictions for upcoming seasons. It should be remarked that the sample sizes are quite small to apply tests on which makes it harder to show that configurations are significantly better than the Uniform configuration compared to large sample sizes. This also makes it understandable why more tests grouped by number of *potential mol* players fail than those grouped by episode number, because the tests grouped by number of *potential mol* players have smaller sample sizes. Moreover it explains why the Exam Drop configuration fails the tests for 2 *potential mol* players, but passes the tests for 3 and 4 *potential mol* players. Despite that the Exam Drop layer has more information when there are only 2 *potential mol* players, the sample size 6 is too small to conclude that the Exam Drop configuration performs significantly better than the Uniform configuration. Furthermore some of these tests fail with a low p-value, e.g. a p-value of 0.060 fails the test, whereas a p-value of 0.040 passes the test and these p-values are quite similar. Nevertheless there is some point where one has to draw the line, though we have to keep in mind that some cases which fail the test are close to passing the test. Also not passing the test does not imply that the configuration performs worse than the Uniform configuration. On the other hand a large part of the tests have a p-value lower than 0.500 implying that the configurations perform better than the Uniform configuration, but unfortunately not significantly. Nonetheless even if all configurations performed significant better than the Uniform configuration, it is no reason to stop improving the Moldel. Passing the test only means that the configuration is significantly better than the Uniform configuration, not that the predictions cannot become more accurate.

[1]Which is why the WIDM-algorithm discussed at the end of Section 1.2 is not deemed to be reliable.

8 CONCLUSION

The Moldel performs significantly better than uniform guessing with a p-value smaller than 0.05 (see column '9' of Table 7.1). Moreover the Moldel also predicted the actual *mol* of 2021 with a likelihood higher than 0.5 in the finals (see Appendix F). However the Moldel is far from perfect with its predictions and therefore needs more data and further work to improve.

A possibility for improvement is to simplify the design for the Wikipedia layer and to let the Exam Drop Layer also investigate *executies* where only part of the players see their screen without any red screen among them. This was done by the old implementation of the Exam Drop Layer, but not by the latest. The Exam Drop Layer could then loop over the possible dropouts, compute a corresponding prediction and take a weighted combination of these predictions. Similarly the Exam Drop Layer could also try to forecast the next dropout and take a weighted combination of the predictions with the corresponding forecasted dropout. But most importantly is to develop new layers to improve the accuracy of the Moldel. A single layer is never able to predict the *mol* alone in an early stage due to a lack of data and the nature of layers. Every layer excludes some players as *mol* and thus having more layers improves the accuracy of the entire Moldel. New layers could be:

- The Money Layer, which analyses the relationship between earning money and being the *mol*. The *mol* is expected to earn less money, since the job of the *mol* is to reduce the amount of earned money. Thus this layer might be able to exclude certain players as *mol*. However the difficulty with this layer is that often not a single player is responsible for earning an amount of money during an exercise. Instead a group of players is responsible for earning that amount money, where some of them had a larger contribution than others.
- The Age Layer, which is an idea for a pre-layer that analyses the relationship between age and being the *mol*. Although age is probably not a direct criteria on which the cast of 'Wie is de Mol?' chooses a *mol*. Nevertheless age is related to the energy and experience of players which might be used as criteria to select a proper *mol*. Some attempts have already been made to include the Age Layer in the Moldel. Unfortunately the relationship between age and being the *mol* was not significant. Therefore this attempt is not discussed in detail in this thesis.

Moreover this project is currently only dedicated to the discovery of the *mol*. Another challenge is to select an optimal strategy for the 'Wie is de Mol?'-app where players can submit points on players who they suspect to be the *mol*. In this app all points submit on the players that pass on to the next episode are doubled and all points submit on the dropout are lost. Deciding how to spread your points in the 'Wie is de Mol?'-app can be tricky if the Moldel is not yet certain about the *mol*. Game theory might be able to deal with this problem, however game theory is outside the scope of this project.

ACKNOWLEDGMENTS

For the realisation of this project I would like to thank:

- Dr. A.J. Schmidt-Hieber [31]: for reviewing and giving feedback on this thesis, providing ideas and advice on how to evaluate and statistically test the Moldel, for providing ideas and advice on improvements and implementation of the layers and for guidance of this project.
- Dr. A.F.F. Derumigny [32]: for reviewing and giving feedback on this thesis, providing ideas and advice on how to evaluate and statistically test the Moldel, for providing ideas and advice on improvements and implementation of the layers and for guidance of this project.
- Dr. M. Poel [33]: for reviewing and giving feedback on this thesis and for guidance of this project.
- Mattijn van Hoek [8]: for his original idea and permission to built the Appearance Layer based on his idea. Moreover I thank van Hoek for the video parse code which is used to analyse how often players appear during episodes.[1]
- Adam Geitgey [9]: for using his library to recognize faces of players by the Appearance Layer.
- Contributors of Scikit Learn [34]: for using their library to implement many of the feature extraction, feature reduction and machine learning algorithms used by the Moldel.
- Jaap van Zessen [7]: to use his Social Media idea and analysis to exclude certain players as *mol* based on evidence showing that these players were not fully present during the recording of 'Wie is de Mol?'.
- Dr. J. de Jong [35]: for ideas leading to the development of the Exam Pass Layer.
- Jesse van Elteren [36]: who helped me at the start of the project by giving some advices and ideas for the development of the Moldel.

[1]Though the statistical testing, in depth evaluation and machine learning model behind the Appearance Layer are my contribution.

REFERENCES

[1] AVROTROS, "Wie is de Mol?." Available at: https://wieisdemol.avrotros.nl/info/. Accessed at 10 September 2020.

[2] J. Willemsen, "Winnaar Gouden Televizier-Ring 2013: Wie is de Mol?." Available at: https://www.televizier.nl/televizier-ring/winnaar-gouden-televizier-ring-2013-wie-is-de-mol, Augustus 2020. Accessed at 10 September 2020.

[3] S. KijkOnderzoek, "Jaar top 100 inclusief sport." Available at: https://kijkonderzoek.nl/component/kijkcijfers/file,j1-0-1-p. Accessed at 10 September 2020.

[4] S. Selvin, M. Bloxham, A. I. Khuri, M. Moore, R. Coleman, G. R. Bryce, J. A. Hagans, T. C. Chalmers, E. A. Maxwell, and G. N. Smith, "Letters to the editor," *The American Statistician*, vol. 29, no. 1, pp. 67–71, 1975. Accessed September 25, 2020 at http://www.jstor.org/stable/2683689.

[5] D. N. Dobie, "Constructing a Predictive Model for the Winner of Survivor," May 2016. Available at: https://cornerstone.lib.mnsu.edu/etds/577/.

[6] F. Ciulla, D. Mocanu, A. Baronchelli, B. Gonçalves, N. Perra, and A. Vespignani, "Beating the news using social media: the case study of American Idol," *EPJ Data Science*, vol. 1, no. 1, p. 8, 2012. Available at: https://epjdatascience.springeropen.com/articles/10.1140/epjds8.

[7] J. van Zessen, "wie is de mol Archives - Jaap van Zessen." http://www.jaapvanzessen.nl/tag/wie-is-de-mol/. Accessed 14 September 2020.

[8] M. van Hoek, "Wie is de Mol gezichtsherkenning!." Available at: https://github.com/mattijn/widm, February 2020. Accessed at 10 September 2020.

[9] A. Geitgey, "Face Recognition." Available at: https://github.com/ageitgey/face_recognition, February 2020. Accessed at 10 September 2020.

[10] E. T. K. Sang, "Wie is De Mol?: Gelijkekansentheorie." Available at: https://ifarm.nl/demol/gkt.html, January 2009. Accessed at 14 September 2020.

[11] J. van Zessen, "Wie is de Mol?-deelnemers doen er alles aan om geheim te blijven, maar helaas," *Het Algemeen Dagblad*, January 2020. Accessed September 23, 2020 https://www.ad.nl/show/wie-is-de-mol-deelnemers-doen-er-alles-aan-om-geheim-te-blijven-maar-helaas~adef878e/.

[12] scikit-learn, "6.3. Preprocessing data." https://scikit-learn.org/stable/modules/preprocessing.html.

[13] F. E. Harrell, *Regression Modeling Strategies*. Springer, 2 ed., 2006.

[14] Christopher M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] H. B. Mann and D. R. Whitney, “On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other,” *Annals of Mathematical Statistics*, vol. 18, pp. 50–60, 03 1947. Available at: `https://doi.org/10.1214/aoms/1177730491`.

[16] scikit-learn, “sklearn.linear_model.LogisticRegression.” `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`.

[17] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, “A limited memory algorithm for bound constrained optimization,” *SIAM J. Sci. Comput.*, vol. 16, no. 5, pp. 1190–1208, 1995. Available at: `http://users.iems.northwestern.edu/~nocedal/lbfgsb.html`.

[18] Dr. J. De Jong, “Monty Hall Problem in Wie is de Mol?.” Dr. J. De Jong idea was mentioned by prof.dr. A.J. Schmidt-Hieber in a personal communication., September 2020.

[19] R. Alake, “Understanding Cosine Similarity And Its Application.” Available at: `https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a`, September 2020. Accessed at 4 November 2020.

[20] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics, Springer, 2009.

[21] “Snowball.” `https://snowballstem.org/`.

[22] K. P. Murphy, *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series, MIT Press, 2012.

[23] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. Springer, 1986.

[24] C. Vuik, F.J.Vermolen, M.B. van Gijzen, M.J. Vuik, *Numerical Methods for Ordinary Differential Equations*. Delft Academic Press, 2 ed., 2016.

[25] Z.H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Machine Learning & Pattern Recognition Series, CRC Press, 2012.

[26] Shalev-Shwartz, S. and Ben-David, S., *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[27] J. van Zessen, “Wie is de Mol deelnemer laten sporen achter op social media.” `http://www.jaapvanzessen.nl/social-media-analist-blogs/wie-de-mol-deelnemer-laten-sporen-achter-op-social-media/`, January 2014. Accessed 12 January 2021.

[28] T. Srivastava, “11 Important Model Evaluation Metrics for Machine Learning Everyone should know,” *Analytics Vidhya*, August 2019. Available at: `https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics`.

[29] StatsDirect, “Paired t Test.” Available at: `https://www.statsdirect.com/help/parametric_methods/paired_t.htm`. Accessed at 31 January 2021.

[30] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. Available at: `http://www.jstor.org/stable/3001968`.

[31] Dr. A.J. Schmidt-Hieber, “Johannes Schmidt-Hieber.” Available at: `https://wwwhome.ewi.utwente.nl/~schmidtaj/`. Accessed at 2 February 2021.

[32] University of Twente, “Welcome... dr. A.F.F. Derumigny (Alexis) Researcher & Director of the Data Science Lab.” Available at: `https://people.utwente.nl/a.f.f.derumigny`. Accessed at 2 February 2021.

[33] University of Twente, “Welcome... dr. M. Poel (Mannes) Assistant Professor.” Available at: `https://people.utwente.nl/m.poel`. Accessed at 2 February 2021.

[34] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[35] University of Twente, “Welcome... dr. J. de Jong (Jasper) Lecturer.” Available at: `https://people.utwente.nl/j.dejong-3`. Accessed at 2 February 2021.

[36] J. van Elteren, “Data Analyse.” Available at: `https://github.com/jvanelteren/wie_is_de_mol`, August 2020. Accessed at 10 September 2020.

[37] M. Hassan, “Machine Learning Hersenen Geest.” Available at: `https://pixabay.com/nl/illustrations/machine-learning-hersenen-geest-3161590/`. Accessed at 10 September 2020.

[38] University of Twente, “Templates and downloads.” Available at: `https://www.utwente.nl/en/organization/house-style/templates_and_downloads/`. Accessed at 10 September 2020.

[39] P. M. P. Peter Y. Chen, *Correlation: Parametric and Nonparametric Measures*. No. 139 in Quantitative Applications in the Social Sciences, Sara Miller McCune, Sage Publications, Inc., 2002.

A EXCEPTION HANDLING

A.1 Exam Drop Layer

A.1.1. The previous implementation of the Exam Layer also allows *executies* E_i with possible dropouts, which happens when only part of the players will see their screen. If there is no red screen among them, then the other players are considered as possible dropouts, because one of them would have seen a red screen and thus dropped out if all screens where revealed. For these cases Dropout(E_i) is defined as the set of all possible dropouts. Moreover

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

is estimated differently for *executies* with possible dropouts (see A.1.2.).

A.1.2. There are two important cases which should be taken into account. First of all, in case *executie* E_i had only possible dropouts, we estimate

$$\mathcal{P}(D_i = \text{Dropout}(E_i) \mid D_{i-1} = \text{Dropout}(E_{i-1}), \ldots, D_1 = \text{Dropout}(E_1),\ p' = \text{Mol})$$

as

$$\frac{\#\text{samples s.t. } D' \in D_i \text{ with } p' = \text{Mol}}{\#\text{samples}}$$

where D' is a sampled dropout for *executie* E_i. In this case D' is a player rather than a set of players, since the combination of multiple dropouts and only revealing some screens has never occurred. Note that the given estimation is equal to the relative number of samples that represents a possible scenario.

Secondly there could be multiple players that share the lowest score, which is called a tie. In the game show during a tie the player that took the most time for the *test* among the players with the lowest score drops out. What is done in case of a tie is simply select a player uniform random among the players with the lowest score, because it is almost never revealed during episodes how much time players took for their *test*. Thus it is assumed that every player is equally likely to be the slowest. To illustrate this:

Example 9. *Suppose there are 6 players Gijs, Isabelle, Jim, Lottie, Sander and Yvon of which Yvon is the* mol*. Furthermore suppose all these players participated in a* test *for which the following number of correct answers is sampled:*

Player:	Gijs	Isabelle	Jim	Lottie	Sander	Yvon
Correct Answers:	2	8	8	12	8	-

Table A.1: Sampled answers

Moreover suppose that there are 2 dropouts for the corresponding executie*. Then Gijs definitely drops out and among Isabelle, Jim and Sander the second dropout is uniform randomly selected.*

A.1.3. But every quartet $Q = (P_1, Q', A_k, P_2)$ is ignored if

(a) Player P_1 returned back to the game or stayed in the game after his/her red screen was revealed.

(b) And player P_1 has not had a non-*voluntary* dropout during any *executie* after *executie* E, or we are currently unaware when his next non-*voluntary* drop out will be.

because it is expected that player P_1 changes his strategy after he/she gets a second chance. Two examples where this situation happened include:

– The dropout of Paul in episode 3 of season 7, who returned back to the game in episode 4 and eventually became the winner of season 7. In this case any quartet Q with $P_1 =$ Paul and question Q' answered by Paul during/after episode 4 is not included in any train data nor used to make any prediction for season 7.

– The revealing of the red screen of Daphne in episode 1 of season 14, who finally dropped out in episode 4. In this case any quartet Q with $P_1 =$ Daphne and question Q' answered by Daphne, after her red screen was revealed, is used as training data. However these quartets Q are not used to make a prediction for season 14 until the dropout of Daphne during episode 4.

A.1.4. Note that there are seasons in which players dropped out twice, because they returned back to the game or they stayed in the game after their red screen was revealed (which is also considered as dropping out). Thus the Drop Episode Number and Drop Executie are not uniquely defined in some cases. For those exceptional cases the Drop Executie is defined as the earliest *executie* that has happened later than the Exam Executie or is equal to the Exam Executie. Likewise the Drop Episode Number is similarly defined as the episode number in which the Drop Executie happened. This design choice is made, because it is expected that player P_1 changes his strategy after he/she gets a second chance.

A.1.5. Remark that $|a \setminus \{p\}|$ in the SPP function might be equal to 0 for some answers a. It happened in the past (although it is very rare) that players filled in a question on only him/herself or that the answer did not cover any player *alive*. In those cases we get undefined values, because division by zero is undefined. When computing the mean, undefined values are simply ignored, e.g. mean$(2, \text{undefined}, 4) = 3$. In case all values are ignored, it is treated similarly as if no answers were given by player p, i.e.

$$\text{SPP}(A, p) = \frac{|A|}{|P| - 1}$$

Ignoring undefined values is a logical decision, because if only the player him/herself or nobody is covered by the answer then it is unknown who that player actually suspects.

A.2 Wikipedia Layer

A.2.1. Remark that cossim(x', x^+) is undefined if x' is a zero vector, i.e. $x' = (0, 0, \ldots, 0)$ or x^+ is a zero vector, i.e. $x^+ = (0, 0, \ldots, 0)$. There are two situations when this happens:

– The player for which a prediction is made has a zero vector as input $x = (0, 0, \ldots, 0)$. In that case his/her *mol* likelihood is equal to 0, i.e. $y = 0$, because it has never happened in the past that the *mol* had a zero vector as input $x = (0, 0, \ldots, 0)$.

– A player j in the training data has a zero vector as train input, i.e. $x^j = (0, 0, \ldots, 0)$ in which x^j is ignored in the sum. So theoretically the predicted likelihood y of a player with encoding x is defined as:

$$y = \frac{\sum_{i, x^i \neq \vec{0}} \text{cossim}(x, x^i)^2 \cdot y^i}{\sum_{i, x^i \neq \vec{0}} \text{cossim}(x, x^i)^2}$$

A.2.2. If $T_p = 0$ then division is undefined, in which case $C'_p(j_i)$ is updated as:

$$C'_p(j_i) \leftarrow 0$$

because if player p did belong to job j_i then it was expected that player p also had words in their Wikipedia page that are linked to job j_i. Which is not the case here, since the total number of words in his/her Wikipedia page is zero, i.e. $T_p = 0$. Therefore player p gets the lowest possible job score for j_i.

A.2.3. If $C'_p(j_i) = 0$ then the logarithm is undefined (this exception case always succeeds A.2.2.). For this case the job feature value $C'_p(j_i)$ is set to the minimum of all other translated & defined job feature values of the same job, i.e.

$$C'_p(j_i) \leftarrow \min\{\ln(C'_p(j_i)) : p \in P \text{ s.t. } C'_p(j_i) \neq 0\}$$

This rule ensures that players with words linked to a particular job j_i always get a larger $C'_p(j_i)$ value than players without words linked to job j_i.

A.2.4. If $T'_p = 0$ then the logarithm is undefined. For these cases T'_p is treated similarly as if that player p would have exactly 1 word in his Wikipedia page, thus T'_p in this case is updated as:

$$T'_p \leftarrow \ln\left(\frac{1}{\sum_{p^+ \in S_p} T'_{p^+}}\right)$$

A.3 Aggregation

A.3.1. Note that $\sigma^{-1}(0)$ and $\sigma^{-1}(1)$ are undefined, which happens if a layer excludes a certain player as *mol* or is fully convinced that a player is the *mol*. If one of the layer predictions $\rho_{i,j}$ in a training instance is 0 or 1 then that training instance is discarded. If one of the layer predictions $\rho_{i,j}$ for a player p_i, whose *mol* likelihood gets predicted, is 0 then that player gets assigned a final *mol* likelihood of respectively 0. Similarly if one of the layer predictions $\rho_{i,j}$ for player p_i is 1 then that player gets assigned a final *mol* likelihood of respectively 1. Layers of course are not fully reliable, but if a layer gives a player p_i a likelihood of 0 or 1 then it is very certain about that. And normally this situation only occurs if player p_i dropped out in which case a final *mol* likelihood of 0 is justifiable.

B ADDITIONAL BINS FOR EXAM DROP LAYER

The number of bins for the Exam Drop Layer is determined by trying all multiples of 2 in the range from 20 up to 80 as number of bins. For every of these configurations the Exam Drop Layer is evaluated on seasons 9 up to *21* using the other seasons as training data. This evaluation is performed by computing the Log Loss (see page 6.1) over all predictions for that season, where a prediction happens after an episode is broadcast and before the first episode is broadcast. The Log Loss scores for these configurations are shown in figure B.1.

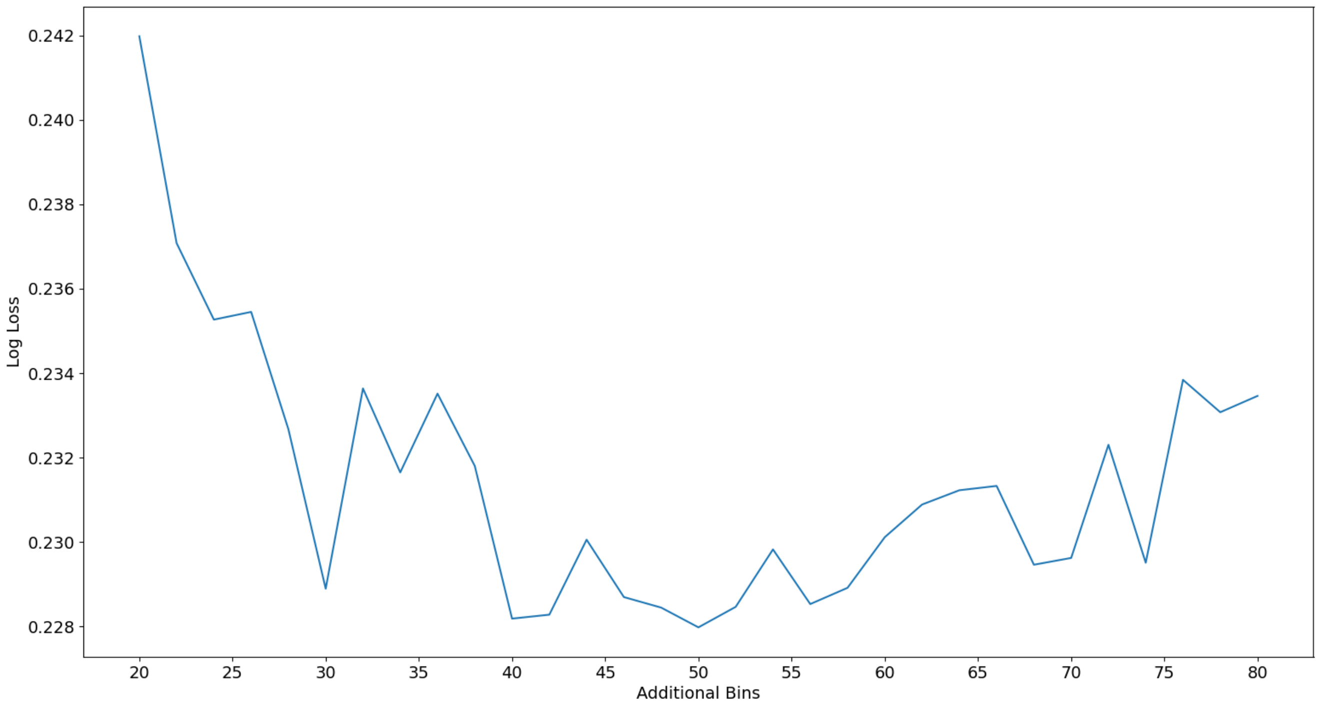
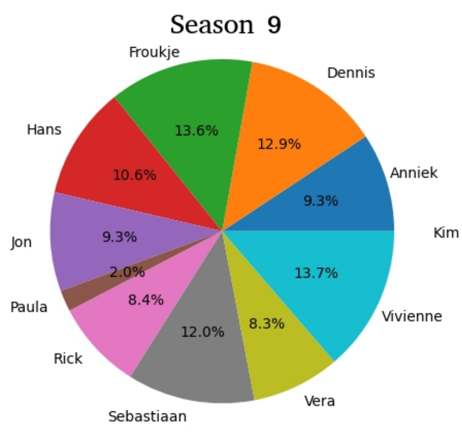
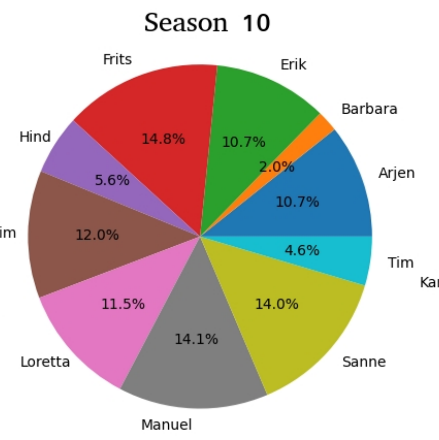


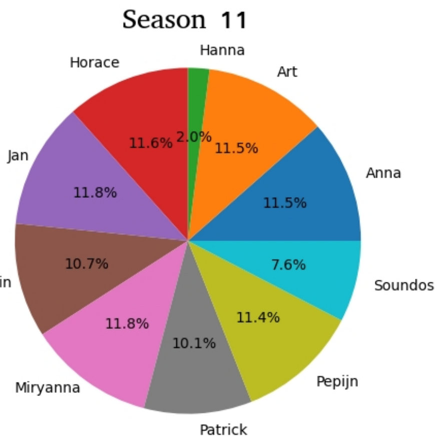
Figure B.1: Log Loss per Additional Bins

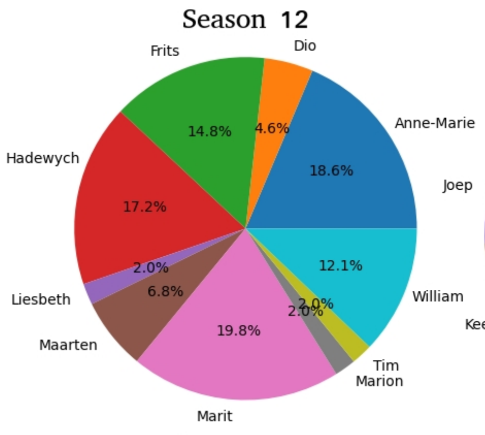
Thus with 50 additional bins the lowest Log Loss score of 0.22798 is obtained and with 40 additional bins the second lowest Log Loss score of 0.22819 is obtained. Having lesser bins is preferred, hence 40 additional bins are used for the Exam Drop Layer. Remark that by using this approach the results presented in Chapter 6 might give a more optimistic impression of the Moldel rather than what the Moldel would actually be. Though 40 additional bins is not a large amount.

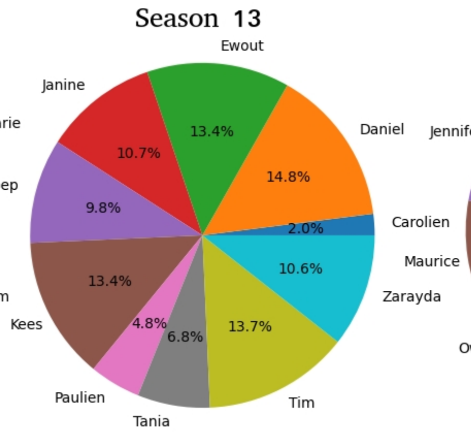
C WIKIPEDIA LAYER - OLD RESULTS

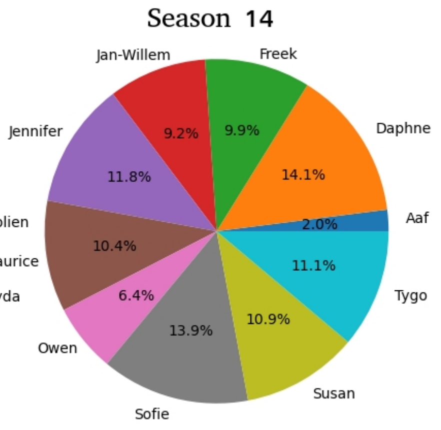


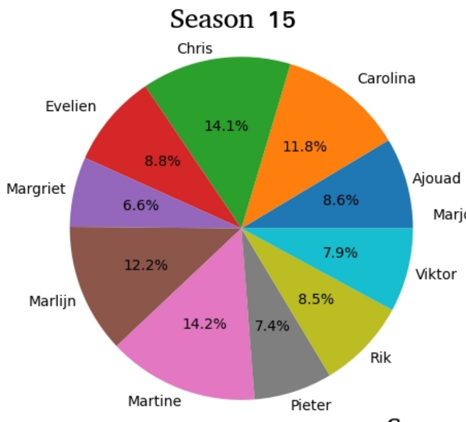


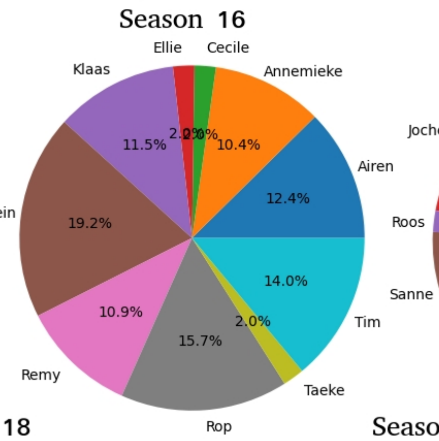


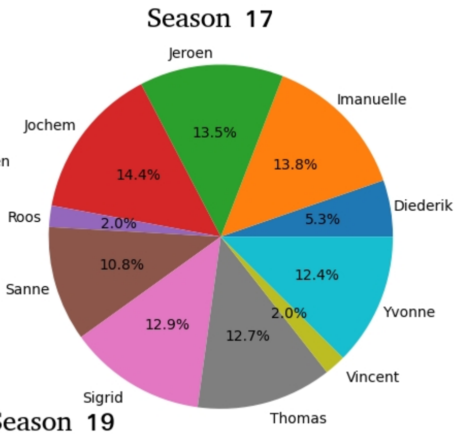


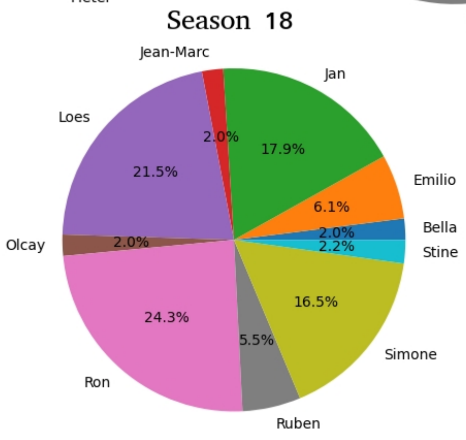


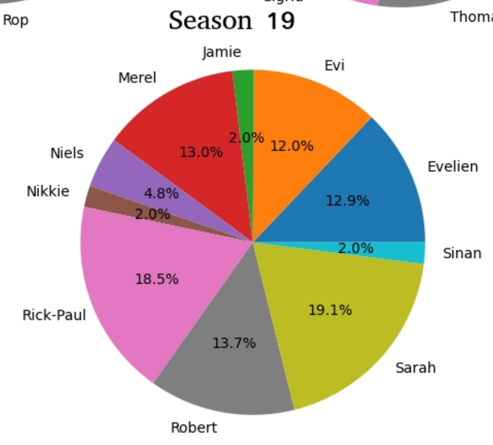












D DEPENDENCY OF APPEARANCE VALUES

The appearance values shown in figure 4.2 are not fully independent, because if $A_{p,e}$ is larger for some player p in episode e then $FC_{p,e}$ is also larger and so is $\sum_{p'} FC_{p',e}$. Hence $A_{p^+,e}$ for another player p^+ in the same episode e becomes smaller (assuming $FC_{p^+,e}$ does not change). So there is a dependence between $A_{p,e}$ and $A_{p^+,e}$, but this dependence is weak, because the increase/decrease of $FC_{p,e}$ has relatively a small effect on $\sum_{p'} FC_{p',e}$. Nevertheless two statistical test are applied to check how strong the relationship between $A_{p,e}$ and $A_{p^+,e}$ is, which are the Pearson's Correlation Test (test for linear relationships) and the Kendall's Correlation Test (test for monotonic relationships) [39, pages 9-10, 16-17, 82-84]. Both of these tests return a value between -1 and 1 where:

- The closer this value is to 1, the stronger these features are correlated in positive direction.
- The closer it is to 0, the weaker these features are correlated.
- The closer it is to -1, the stronger these features are correlated in negative direction.

Applying these tests on all paired combinations of appearance values between two distinct player p_1 and p_2 in the same episode results in:

- $r = -0.128$ for the Pearson Correlation test, which corresponds with a p-value of $3.0 \cdot 10^{-9}$.
- $\tau = -0.085$ for the Kendall Correlation test, which corresponds with a p-value of $3.7 \cdot 10^{-9}$.

So there is a relationship between appearance values of the same episode, but it is a weak relationship with correlation coefficients close to 0. Therefore this dependency is not a severe violation of the independence assumption for the Mann-Whitney U Test.

Moreover there is also a dependency between the appearance values of the same player for different episodes, i.e. between $A_{p,e}$ and $A_{p,e'}$. Players that appear less in episode e are also more likely to appear less in another episode e', regardless whether that player was the *mol* or not. Therefore Pearson's Correlation Test and Kendall's Correlation Test are applied on all paired combinations of appearance values of the same non-*mol* player p in different episodes (of the same season), which results in:

- $r = 0.502$ for the Pearson Correlation test, which corresponds with a p-value of $3.6 \cdot 10^{-75}$.
- $\tau = 0.360$ for the Kendall Correlation test, which corresponds with a p-value of $4.5 \cdot 10^{-75}$.

So this is more severe violation of the independence assumption for the Mann-Whitney U Test. Hence we group all appearance values of the same player together and take the minimum of these groups, i.e.

$$\hat{A}_p = \text{min}\{A_{p,e} : e \in E_p\}$$

which removes this dependency, however unfortunately it also decreases the sample size of *mol* cases X to 9 and the sample size of non-*mol* cases Y to 81. Applying Mann-Whitney U Test [15] on this minimum value $\hat{A}_p$ results in a U value of 180. And this U value corresponds to a Z-score of -2.475 resulting in a p-value of $\approx 6.667 \cdot 10^{-3}$. So the hypothesis of van Hoek also holds when this dependency is removed.

E RAW RESULTS

Tables E.1 up to E.10 show the raw evaluation scores rounded up to 3 decimals for season 9 up to *21* of the different configurations as presented in Chapter 6. If a score is **bold** then this score was the best score compared to other configurations (for which more than 3 decimals are taken into account).

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Full Moldel	**0.311**	**0.306**	0.289	0.281	**0.258**	**0.208**	0.204	**0.163**	**0.113**	**0.057**
Plus Moldel	0.322	0.315	0.294	0.285	0.264	0.212	0.210	0.185	**0.113**	**0.057**
Min Moldel	0.321	0.309	**0.289**	0.277	0.272	0.224	0.217	0.189	0.126	0.065
Exam Drop	0.325	0.310	0.291	**0.274**	0.264	0.215	**0.203**	0.176	0.140	0.082
Exam Pass	0.325	0.314	0.301	0.289	0.283	0.267	0.251	0.228	0.189	0.144
Wikipedia	0.320	0.310	0.298	0.288	0.278	0.260	0.248	0.227	0.200	0.153
Appearance	0.325	0.325	0.297	0.279	0.266	0.244	0.235	0.210	0.178	0.130
Uniform	0.325	0.316	0.303	0.291	0.281	0.264	0.251	0.232	0.203	0.159

Table E.1: Log Loss for configurations grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Full Moldel	**0.317**	0.314	0.284	0.266	**0.238**	**0.197**	**0.141**	**0.111**	0.104
Plus Moldel	0.327	0.321	0.287	0.274	0.241	0.205	0.164	**0.111**	0.104
Min Moldel	0.321	0.307	**0.281**	0.271	0.249	0.215	0.163	0.119	0.126
Exam Drop	0.325	0.310	0.283	**0.266**	0.238	0.206	0.160	0.122	0.127
Exam Pass	0.325	0.311	0.299	0.287	0.273	0.250	0.218	0.181	0.113
Wikipedia	0.320	**0.306**	0.297	0.286	0.266	0.248	0.220	0.183	0.128
Appearance	0.335	0.321	0.296	0.282	0.254	0.231	0.202	0.163	**0.101**
Uniform	0.325	0.314	0.301	0.287	0.270	0.250	0.225	0.191	0.139

Table E.2: Log Loss for configurations grouped by number of *potential mol* players

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Full Moldel	**2.175**	**2.124**	**1.963**	1.878	**1.666**	**1.274**	**1.181**	**0.886**	**0.590**	**0.292**
Plus Moldel	2.273	2.206	2.011	1.922	1.729	1.318	1.246	1.025	**0.590**	**0.292**
Min Moldel	2.270	2.154	1.963	1.852	1.796	1.416	1.293	1.071	0.672	0.328
Exam Drop	2.303	2.165	1.991	**1.837**	1.736	1.335	1.223	1.014	0.768	0.428
Exam Pass	2.303	2.197	2.077	1.966	1.908	1.758	1.621	1.416	1.109	0.793
Wikipedia	2.260	2.165	2.048	1.952	1.860	1.701	1.588	1.414	1.182	0.854
Appearance	2.303	2.302	2.049	1.883	1.761	1.572	1.488	1.282	1.029	0.726
Uniform	2.303	2.212	2.096	1.983	1.887	1.736	1.620	1.447	1.200	0.880

Table E.3: Mol Log Loss for configurations grouped by episode number

#Potential Mol	10	9	8	7	6	5	4	3	2
Full Moldel	**2.226**	2.192	1.908	**1.744**	**1.491**	**1.159**	**0.751**	**0.582**	0.518
Plus Moldel	2.324	2.262	1.940	1.819	1.535	1.235	0.890	**0.582**	0.518
Min Moldel	2.270	2.135	**1.898**	1.798	1.616	1.296	0.907	0.619	0.631
Exam Drop	2.303	2.160	1.919	1.761	1.527	1.236	0.915	0.653	0.634
Exam Pass	2.303	2.171	2.061	1.949	1.810	1.604	1.332	1.027	0.564
Wikipedia	2.260	**2.124**	2.037	1.932	1.759	1.593	1.346	1.046	0.640
Appearance	2.399	2.267	2.037	1.896	1.655	1.454	1.220	0.941	**0.505**
Uniform	2.302	2.197	2.079	1.946	1.792	1.609	1.386	1.099	0.693

Table E.4: Mol Log Loss for configurations grouped by number of *potential mol* players

Episode Number	0	1	2	3	4	5	6	7	8	9
Full Moldel	**0.622**	0.617	0.743	0.746	0.809	**0.903**	**0.909**	**0.951**	**0.981**	**0.997**
Plus Moldel	0.563	0.588	0.727	0.733	0.789	0.895	0.902	0.938	**0.981**	**0.997**
Min Moldel	0.578	0.642	0.780	0.778	0.784	0.881	0.897	0.928	0.970	0.995
Exam Drop	0.500	0.740	**0.781**	**0.790**	**0.810**	0.896	0.908	0.937	0.968	0.996
Exam Pass	0.500	0.652	0.674	0.709	0.664	0.717	0.788	0.855	0.923	0.953
Wikipedia	0.619	**0.663**	0.705	0.719	0.751	0.806	0.833	0.864	0.913	0.963
Appearance	0.500	0.520	0.744	0.764	0.809	0.860	0.871	0.902	0.939	0.963
Uniform	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500

Table E.5: Concordant-Discordant Ratio for configurations grouped by episode number

#Potential Mol	10	9	8	7	6	5	4	3	2
Full Moldel	0.562	0.577	0.763	0.786	**0.866**	**0.916**	**0.962**	0.979	0.978
Plus Moldel	0.485	0.556	0.755	0.759	0.855	0.904	0.951	0.979	0.978
Min Moldel	0.578	0.695	0.792	0.779	0.839	0.900	0.948	0.975	0.969
Exam Drop	0.500	**0.756**	**0.808**	**0.811**	0.849	0.905	0.955	0.978	0.972
Exam Pass	0.500	0.674	0.677	0.651	0.711	0.800	0.880	0.929	0.988
Wikipedia	**0.619**	0.730	0.717	0.725	0.801	0.833	0.902	**0.982**	0.984
Appearance	0.194	0.566	0.756	0.731	0.841	0.892	0.920	0.946	**1.000**
Uniform	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500

Table E.6: Concordant-Discordant Ratio for configurations grouped by number of *potential mol* players

Episode Number	0	1	2	3	4	5	6	7	8	9
Full Moldel	**0.115**	**0.121**	**0.149**	**0.171**	**0.203**	**0.344**	**0.357**	**0.480**	**0.602**	**0.766**
Plus Moldel	0.104	0.113	0.144	0.166	0.193	0.338	0.351	0.455	**0.602**	**0.766**
Min Moldel	0.104	0.117	0.144	0.167	0.177	0.298	0.318	0.423	0.572	0.739
Exam Drop	0.100	0.116	0.139	0.165	0.183	0.296	0.322	0.399	0.505	0.672
Exam Pass	0.100	0.112	0.126	0.141	0.149	0.174	0.200	0.248	0.342	0.485
Wikipedia	0.105	0.116	0.130	0.144	0.158	0.185	0.206	0.249	0.316	0.461
Appearance	0.100	0.106	0.137	0.164	0.186	0.227	0.245	0.304	0.383	0.543
Uniform	0.100	0.110	0.123	0.139	0.152	0.177	0.199	0.238	0.308	0.442

Table E.7: Mean Mol Likelihood for configurations grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Full Moldel	**0.109**	0.114	0.155	**0.188**	**0.272**	**0.369**	**0.534**	**0.644**	**0.661**
Plus Moldel	0.100	0.108	0.151	0.177	0.267	0.362	0.509	**0.644**	**0.661**
Min Moldel	0.104	0.119	**0.156**	0.180	0.247	0.317	0.485	0.623	0.602
Exam Drop	0.100	0.117	0.150	0.180	0.249	0.317	0.442	0.558	0.580
Exam Pass	0.100	0.114	0.128	0.143	0.164	0.202	0.265	0.361	0.575
Wikipedia	0.105	**0.120**	0.132	0.146	0.174	0.205	0.262	0.353	0.532
Appearance	0.094	0.110	0.138	0.161	0.205	0.253	0.324	0.433	0.620
Uniform	0.100	0.111	0.125	0.143	0.167	0.200	0.250	0.333	0.500

Table E.8: Mean Mol Likelihood for configurations grouped by number of *potential mol* players

Episode Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Full Moldel	**4.308**	4.462	3.462	3.115	**2.577**	**1.923**	1.846	**1.462**	**1.231**	**1.000**
Plus Moldel	4.769	4.846	3.615	3.269	2.808	2.000	1.923	1.538	**1.231**	**1.000**
Min Moldel	4.731	4.346	**3.308**	**2.846**	2.846	2.038	**1.654**	1.654	1.462	1.154
Exam Drop	5.500	4.692	3.808	3.192	2.769	1.962	1.769	1.577	1.346	**1.000**
Exam Pass	5.500	4.769	3.923	3.654	3.692	3.385	2.808	2.192	1.808	1.692
Wikipedia	4.769	**4.308**	3.885	3.731	3.500	3.000	2.731	2.423	2.077	1.692
Appearance	5.500	5.000	3.923	3.731	3.308	2.654	2.423	2.038	1.769	1.500
Uniform	5.500	5.077	4.577	4.154	3.808	3.346	3.038	2.654	2.192	1.769

Table E.9: Mean Mol Rank for configurations grouped by episode number

#Potential Mol	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
Full Moldel	**4.692**	4.417	**3.167**	3.091	**2.154**	1.727	**1.385**	**1.111**	**1.167**
Plus Moldel	5.231	4.750	3.250	3.364	2.231	1.818	1.462	**1.111**	**1.167**
Min Moldel	4.731	**3.917**	3.375	**3.000**	2.269	**1.636**	1.692	1.222	1.333
Exam Drop	5.500	4.667	3.417	3.045	2.308	1.727	1.462	**1.111**	**1.167**
Exam Pass	5.500	4.375	3.917	4.045	3.423	2.591	2.000	1.833	1.333
Wikipedia	4.769	4.000	3.917	3.773	3.154	2.818	2.231	1.778	1.333
Appearance	5.885	4.833	3.833	3.864	2.808	2.364	2.000	1.778	1.250
Uniform	5.500	5.000	4.500	4.000	3.500	3.000	2.500	2.000	1.500

Table E.10: Mean Mol Rank for configurations grouped by number of *potential mol* players

F FINAL PREDICTION OF SEASON 22

Season 22 is defined as the season that was broadcast in the period from 2 January 2021 up to 6 March 2021. The prediction after the finals is shown by Figure F.1. Renée turned out to be the *mol* of this season.

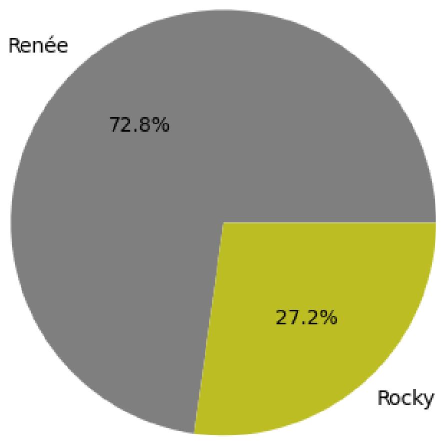


Figure F.1: Final predictions of Full Moldel for seasons 9 up to 12