Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Machine Learning and Feature Selection for Videogame Beta Testing

Laureando

Lorenzo Fabris

Relatore

Prof.ssa Silvana Badaloni

Correlatore

Dott. Francesco Sambo

21 Aprile 2015

Anno Accademico 2014/2015

Acknowledgements

I would like to thank Prof.ssa Silvana Badaloni and Dott. Francesco Sambo for the opportunity to work on a topic that has always fascinated me.
I am also grateful to Gianpaolo Greco and Marco Oliviero from Synthetica Lab, who allowed me to be part of the development of a commercial video game.
I also thank my parents and my entire family for the unending encouragement, support and attention they gave me throughout my life, that contributed to what I am today.
Last but not least, I want to thank all my friends whom I shared with so many unforgettable experiences: the Dominatori di Grado, the True GDT del Fanfredì and the Quarto Canale di Ingegneria dell'Informazione 2008/09.

Contents

Abstract

This work presents the integration of Machine Learning techniques in the beta testing phase of Heartbot - Escape, a commercial video game, in order to help the game designer in the identification of the game parameters and player characteristics that have the most influence on metrics such as entertainment and difficulty. The game was analysed to define the actual input features and several classification outputs, data collection was designed and performed and data were then analysed. Random Forests and Support Vector Machines were used to predict the game sessions outputs and Recursive Feature Elimination was applied to identify the features of most importance for each output measure. Classifiers trained on selected features were then evaluated on a separate test set to obtain an unbiased quality assessment. Selected features are shown to be meaningful for each classification output.

Chapter 1

Introduction

The video game and entertainment software industry has now grown to be one of the biggest productive segments of the technological industry, with grosses rivalling those of the film industry[1]. The producer ecosystem[2] is fragmented with at one end of the spectrum large studios, with hundreds of developers and publishing budgets in the millions of dollars, and at the other the independent developers, that have much lesser resources and mostly distribute their products single-handedly on the web[3].
The abundance of cheaper video games determines a sharp competition for visibility on the self-publishing channels, in the hope of either becoming a viral phenomenon or being chosen by a dedicated niche of players.
Machine learning can help the video game designer by identifying and analysing the most important game play elements of a product in order to influence different metrics of players' entertainment [1], in relation to different demographic segments. The formalization of a classification problem that links the gameplay numeric settings to a players' appreciation questionnaire, followed by a feature selection phase, can guide the game designer in the fine tuning and finalization of the video game.
The data to be used in the process can be obtained during the beta testing phase of the product, when the game mechanics are already established, but tuning can still take place. Beta testing takes place before the actual release and is primarily targeted to finding bugs and retrieving an evaluation of the product by actual users. Thanks to the self-publishing of most independent titles, an increasing number of games are released while unfinished[4], allowing for an extensive collection of playing experiences before being finalized.
This work will present and discuss the application of these concepts to the beta test of the video game Heartbot - Escape by Synthetica Lab[5], and in-

[1] http://www.gartner.com/newsroom/id/2614915
[2] http://www.gamasutra.com/
[3] http://store.steampowered.com/
[4] http://www.gamasutra.com/view/feature/197190/when_players_buy_your_game_before_.php
[5] http://www.syntheticalab.com/

dependent game developed for the mobile market. Heartbot is a steampunk stealth game with puzzle elements that primarily targets mid-core gamers, a niche of more expert players that require a more refined and challenging experiences than those usually offered by common mobile games.
The objective of this work is to identify the set of game parameters and players characteristics that have the most influence on the evaluation of the game's entertainment and difficulty; the game designer also requested the evaluation of the control system and insights on the duration of the game sessions.
Chapter 2 will contain a brief overview of the existing publications on related arguments, discussing how they differ from this work.
Heartbot - Escape, will be presented in chapter 3, describing its development history and its gameplay elements. An explanation and discussion of the features selected for the main analysis will follow.
Chapter 4 will introduce the Random Forest and Support Vector Machines classifiers, followed by a brief discussion of feature selection techniques using the two classification methods.
Chapter 5 will present the formalization of the classification problems for the game and the experiment design, discussing the data collected and the strategies adopted in the analysis.
Chapter 6 contains the actual experiment results and their interpretation from the game designer's point of view.
Chapter 7 draws the conclusions and the directions of future work.

Chapter 2

State of the Art

Games are an important part of human culture, having a central role in the development of children into adults. As such, games have been analysed from many different points of views like sociology and psychology.
In recent times, the diffusion of video games and other entertainment software has tied this particular segment of the games spectrum to the advancement in computer science, across algorithm design, graphic techniques and innovations in artificial intelligence.
Several books have been published about the analysis and breakdown of classic games from a designer's point of view, with the intent of forming an effective lexicon to further understand the subject. The ultimate goal is to identify the crucial elements of a game that entertain the player and develop design strategies to maximise their desired effects.
Classic books are [2] regarding the analysis of classic games and [3], [4] for a detailed description of the game design process of modern computer games.
Another seminal article has been [5] on the more general topic of analysing the mechanics that make activities entertaining, culminating in the concepts of Challenge, Curiosity and Fantasy. The popular theory of Flow was proposed in [6] and applied more specifically to computer games in [7].
In [8], the recurring elements in video games of different genres are categorized and [9] discusses a compact descriptive model based on the triad Mechanics, Dynamics and Aesthetics.
All these publications have however treated game design as a strictly artistic process and, while successful in developing a formal language for research, do not attempt to quantify the ingredients of the resulting games' entertainment.
On the matter of finding mathematical relationships between the game elements and the player satisfaction, many works have experimented the possible applications of artificial intelligence and machine learning techniques under two main approaches, sometimes overlapping: online adaptation of the game difficulty and procedural generation of game levels.

In [10] and [11], Hunicke et al. delineate the requirements of Dynamic Difficuly Adjustment systems for video games under the assumption of being able to detect during a play session undesired mental states of the player, for example to prevent frustration or excessive relaxation.
In [12], Andrade et al. implement a system for dynamic difficulty adjustments of fighting games that selects moves of non playable character with the intent of neither being nor too difficult nor too easy to be beaten by the player, on the basis of the system's own learned experience. In [13] the authors compare the balancing performances of algorithmically adapted enemy behavious and of hand-tuned ones, finding that the former provided more balanced enemies.
Togelius et al. examine the procedural generation of tracks for a racing game, using a model of the player's driving style to evolve circuits that will result entertaining when tried by the real player [14], [15]. The player model adopts a rather simplistic set of assumptions and the results, while good from a mathematical perspective, have not been validated by an extensive human testing.
Yannakakis et al. explore the effects of adaptation of non playable characters behaviours in predator/prey games such as Pac Man[1] in relation to different iterations of a quantitative interest metric, defined on emergent gameplay features. In [16], [17] and [18] different enemy behaviours using neural networks are evolved against computer controlled players to maximise an interest index. In [19], [1] the approach is extended to an augmented-reality game and applied to human players testing.
Pedersen and Yannakakis [20] analyse the effects of a conjunction of level generation parameters and gameplay features in predicting different evaluations of a game session, such as fun, challenge and frustration, using Infinite Mario Bros[2] as a test bed. Neural networks are trained on the data, collected by human players, and feature selection is performed to identify the most important settings and gameplay characteristics.
This work expands the previous research on linking low level settings of the game parameters to interest and difficulty metrics retrieved from actual human players. The main difference from previous works is the focus on the ability of the game designer to tune the game in order to maximise entertainment in relation to the players' characteristics. The classifiers used will be the Random Forest and Support Vector Machines, with feature selection being applied to find the most relevant ones for each metric.

[1] http://en.wikipedia.org/wiki/Pac-Man
[2] https://github.com/cflewis/Infinite-Mario-Bros

Chapter 3

Heartbot - Escape

This section will present the videogame HeartBot - Escape, which will be the test subject of this work.
After a brief introduction to its genre and themes, the development process will be detailed and a breakdown of its gameplay elements will follow. Finally, the collection of features and the set of classification problems defined on those will be discussed.

3.1 Introduction

Heartbot - Escape is a videogame released in March 2015 on the Windows Phone Store, by Synthetica Lab, an Italian independent game developer.
The player is tasked with controlling the titular robot, which at the beginning of the game suffers from amnesia and finds itself capable of feeling emotions because of a human heart inside it.
By exploring the castle it finds itself in, overcoming puzzles and other hostile robots, it will find the truth behind its condition.
The game is a 3D stealth adventure with puzzle elements and adopts a steam-punk visual style; its demographic target is the mid-core segment, comprising players that require more aesthetically curated and challenging games, which is currently an underserved niche on the mobile market. It is a cross-platform application primarily developed for Windows Phone tablets and smartphones.
The app is monetized as free-to-play with the possibility of paying hard currency to unlock additional levels and aestethic goods.

3.2 Development

Heartbot has its roots in a prototype made during the 2013 Tallin Game-Founder hackathon, a contest in which small teams of developers have a short fixed time to develop a video game. The game won the first prize.

The initial theme was to experiment with the concept of an emotional character: such a character presents to the player controlling it another layer of mechanics to profit from (or difficulties to overcome) to beat the game.
The prototype featured a robot that could be scared by environment elements, such as falling objects; its reaction would disturb the control system, making the robot sway from the direction chosen by the player.
The decision to evolve towards a stealth game was heavily influenced by the initial fear mechanic, as it essentially determined a penalty for exposing the robot to danger.
The game has been developed in the Unity[1] engine, using the C# programming language[2]. Unity is an industry standard engine used mainly by independent game developers, thanks to its ease of use, large community, cross platform development and competitive licensing options.
At the beginning of 2014 the project was resumed to be expanded in a full game: in this phase the first draft of the game design document was written, detailing the aesthetic style and the main gameplay elements. Following the document, the core systems were implemented and the initial 3D assets were contracted externally. The work presented in this paper started in June 2014 using the then available features.
In September 2014 the team decided to reimplement from scratch many core systems, as they were found inadequate. In particular, the artificial intelligence governing the enemies non-playable-character was switched from using hard coded finite state machines to using scripted behaviour trees [21]. Because of the lack of versatile tools for this paradigm, many efforts have been expended to implement a full featured behaviour editor integrated in the Unity engine.
In late September 2014, a first public beta was held at the Artificial Intelligence Lab of the Department of Information Engineering of the University of Padua, using an Android build. Due to the heavy changes that were being operated on the main code base, however, the beta version of the game used levels and non playable characters from the previous implementation, rendering most of the data collected incompatible with future developments. The beta version however proved that the data collection systems were effective and its initial results were used to for a first tuning of the game parameters, most notably regarding the robot's movement system.// In the same month, the game was accepted in the Nokia Appcampus, a startup accelerator program targeted to young mobile development firms. The program offered an initial investment in the developers' team, requiring the accepted applications to be released as Windows Phone Store exclusives, allowing for them to be published on competing platforms after three months.
Development accelerated to meet the deadline for the Windows Phone Store

[1] http://unity3d.com/
[2] https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

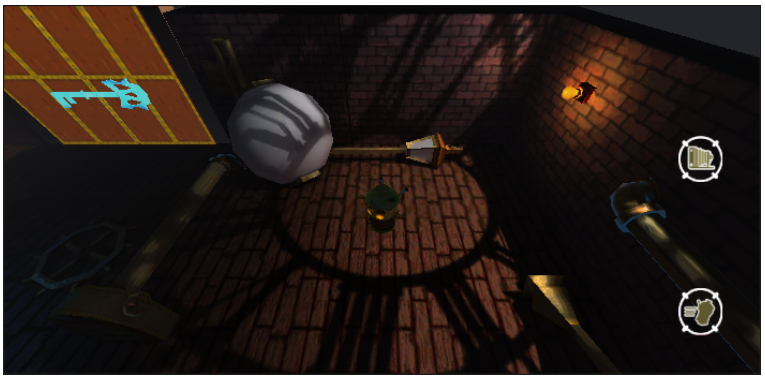

Figure 3.1: Heartbot and the user interface; the higher button is the camera switch and the one below is the dash button

approval, fixed for late December 2014; the author of this work actively participated during the last days to fix bugs and finalize some required features. In the first half of December the last beta testing phase was started, using an Android build of the updated implementation of the core systems; this version was then frozen and the beta testing continued until March 2015, with only minor bug fixes. The first data collection was done as a public event at the Artificial Intelligence Lab of the Department of Information Engineering of the University of Padua and subsequent collection batches were held as sparse events.
The final version of Heartbot - Escape was released publicly on the Windows Phone Store at the end of March 2015. The feedback of the beta testing data was used in tuning the in-game tutorials and adjusting the robot's control system, while subsequent results will be used for future releases.

3.3 Gameplay

This section will break down the main elements of the game, starting with the control system; then there will be a description of the game environment and the possible interactions with it.
The last is an overview of the enemies non-playable-characters.

3.3.1 Control System

The game requires the player to navigate a 3D environment without exposing the character to danger, thus stressing the importance of careful exploration and analysis of the surroundings.
The control system adopts a third person camera with the character's movement relative to it; movement is restricted to the x-z plane.

Using the touchscreen, the player can move the robot by tapping on any point of the screen and dragging his or her finger in the desired direction; the resulting effect imitates the well-known control sticks found on the console gamepads.
Movement is implemented with an animation based approach: the speed is implicitly defined by the translations and rotations that the artist applies in the 3D animation editor. The advantage of this method is that the resulting transitions between movement phases are smoother and programmatic control is still possible by scaling the individual animation speed; this solution, however, requires careful tuning, as a transition between animations with an excessive speed difference can be aesthetically unacceptable.
The player can also activate a short dash, by tapping a button on the bottom right part of the screen. The dash is then deactivated for a brief time. When Heartbot is dashing, collisions with static objects will trigger a noise that propagates up to a small distance; this is represented by a red ring and will alert enemies that are reached.
The camera follows the character and can not be explicitly controlled; it has a fixed roll, its yaw is only changed in some predetermined environment sections and it adjusts its height and pitch to evade obstacles on the line of sight towards Heartbot. The player can rotate the camera by 180 degrees by tapping a button in the bottom right of the screen.
When the robot feels scared, for example when spotting a close enemy or hearing an unexpected noise, it will panic: the control system becomes unresponsive and the robot moves away from the danger. This can be deadly to the robot, as the uncontrolled movement could place it in a difficult spot to escape from, once an enemy has begun chase.

3.3.2 Environment

The game takes its first steps in the interiors of a floating castle, built by a scientist specialized in mechanical life forms. As the protagonist awakes, it is contacted by a mysterious radio transmission that will guide it through the castle's dangers lying; it appears that the transmission is from the scientist, who finds himself trapped by his own mechanical creations and needs to be rescued by his most recent masterpiece.
Throughout the levels environments the player will find different elements to interact with.
The most important are doors: if closed, they prevent access to an area to both Heartbot and its enemies, thus being crucial to both advance the exploration of the castle and evade dangers. Closed doors can usually be opened by collecting the appropriate key, which is then consumed when used. Other doors can both be opened and closed by activating levers.
Levers are another main element of the levels, as their activation can significantly alter the environment.

Other than the aforementioned switch of a door's state, some levers change the position of objects blocking narrow passages or preventing reach of items such as door keys. Some levels also feature chasms that can only be traversed by standing on moving platforms; these are moved by the activation of levers. Scattered through the rooms are also some large pipes that Hearbot can enter: other than connecting to secluded area, these pipes hide the robot from enemies chasing it.
Additionally, some areas host scripted events activated by the robot's proximity. These can trigger the appearance of tutorial screens in the user interface or activate the fall of items, whose noise may put the player in danger. Another type of item that can be picked up is the top hat; some areas host a small number of them and the scientist will ask the player to retrieve them to help him after his rescue.

3.3.3 Enemies

In its wandering through the castle, Heartbot will encounter a number of other hostile character that will try to capture it; the player is then required to restart at the beginning of the area he or she was captured in. Each of these enemies is a mechanical version of a different animal, with its own movement pattern and its unique way of contributing to the hunt of the protagonist.
Heartbot has no mean of harming its enemies and during its roaming the player will need to find cunning ways to evade them.

Fireflies Hunter

These robotic insects behave like flying patrolling units, roaming along a predefined path and searching for intruders. At the end of their elongated

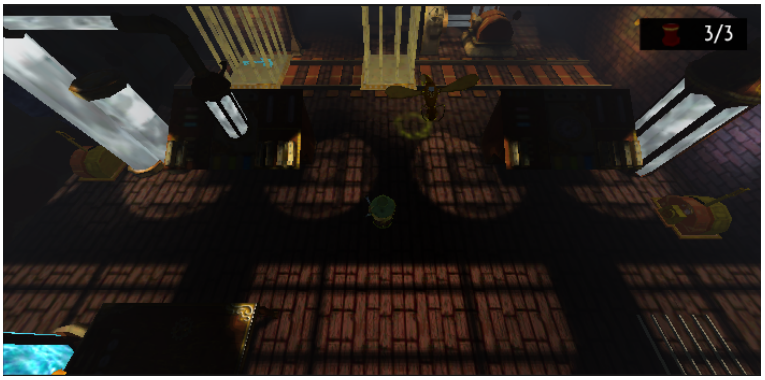

Figure 3.2: A fireflies hunter in its patrol is moving towards Heartbot

bodies, they host a beacon that projects a circle of light on the ground: the hunters can detect foreign objects that enter this vision cone, thus transitioning to a state of alarm.
When in this state, an hunter will stop its movement and call for the intervention of a spider, described below, to dispose of the detected intruder.
Fireflies hunters also react to sounds: when sensing an anomalous noise, they will steer from their normal patrolling to inspect the source of the disturbance. If no abnormality is found, they will return to their default duty after a short while. This zealous behaviour can however be used as a weapon against them, by luring them away from strategic passages and sneaking undetected.

Spider

The spiders are mechanical creatures that do not usually roam the castle's rooms, preferring to stay dormant, attached to the ceilings. When a fireflies hunter detects an intruder, however, the nearest spider awakes and descends on the floor to give chase.
Spiders are fearsome hunters and can detect their prey's presence through walls, keeping the pursuit up until either the capture is successful, or the chased manage to hide inside one of larger pipes in the castle.
As soon as a spider reaches its target, it spits a web that traps the unfortunate one and returns to the ceiling bringing the new trophy with itself.

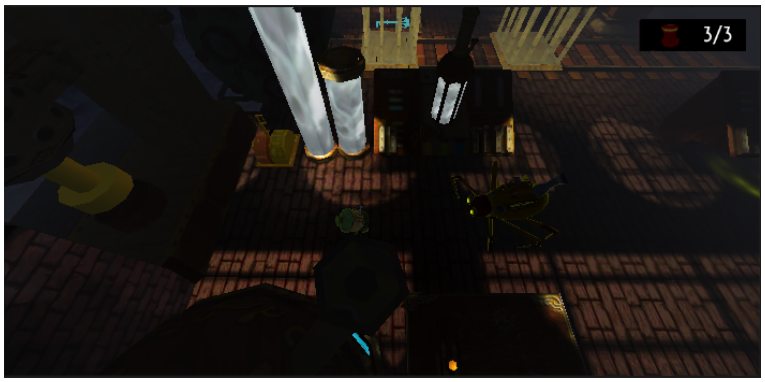

Figure 3.3: A spider chasing Heartbot

3.4 Feature Collection

The main target of this work is the identification of the characteristics of the game that have the most importance on the players' entertainment and the game's perceived difficulty. The features describing a playthrough have been broken down in three main groups:

- **Controllable:** these comprise the values that a game designer can modify to tune the characteristics of the gameplay mechanics, such as movement parameters of Heartbot and its enemies
- **Player:** these features represent a player as a set of demographic informations and previous experiences in video games
- **Emergent:** this category contains features that derive from the actual interaction of the players with the game and reflects their personal approach to the system; events such as the number of times a lever is activated or the collection of an item, together with temporal metrics, form the bulk of emergent features
- **Output:** a collection of the players' evaluation of the game, such as difficulty and interest

Level design is certainly a controllable element of the game, but has been omitted from the model as there is no compact way of representing its characteristics: Hearbot's environment and interactive elements are laid out by hand, following more of an artistic rather than mechanical approach.
Another approach to level design recently rising in popularity is the procedural generation of levels, adopting randomization and tuning parameters to pilot placement algorithms, offering a great variety of areas, enemies and even story lines. With this strategy it is straightforward to model a level using the parameters of the algorithm as features.
As for emergent features, this work fully implemented their collection but did not integrated them in the analysis. While previous works in the literature (see [20]) have shown the importance of such features in predicting the player experience, they still are for the most part a consequence of the level design. As this aspect of the game can not be modified by the program in an automatic manner, emergent features regarding interaction with enemies and the environment are of little value to the analysis discussed here.
Only the playing session duration has been used as an output feature, as will be explained later.

Controllable Features

For every controllable feature, a lower and upper bound were selected by the game designer in order to maintain the balance of the gameplay mechanics.

As it will be further clarified in the chapter about experiment design, the beta version of the game generates randomized settings of the controllable features to present each player with a different game experience.

Identifier	Lower bound	Upper bound
`bot_snd_rad`	8	10
`bot_mov_spd`	0.7	1.3
`bot_dsh_spd`	0.85	1.15

The first feature is Heartbot's sound radius and controls the maximal extension, in in-game units, of the noise generated by colliding with the environment during a dash; higher values will make Heartbot easier to be spotted if not careful. Heartbot's movement speed and dash speed are coefficients used for scaling the relevant animation speeds; obviously higher values will result in a faster robot.

Identifier	Lower bound	Upper bound
`fh_view_cone`	0.49	1
`fh_mov_spd`	0.85	1.15
`fh_mov_acc`	0.7	1

The angle at the apex of a fireflies hunter is controlled by the first parameter, which sets the value in radians. Higher values determine larger detection volumes and it is important to note that the graphical representation is updated accordingly; the game mechanic regarding detection must respect a contract of clarity with the player.
In a similar way to Heartbot, fireflies hunters have a movement speed coefficient; they however also have an acceleration coefficient that influences the transition between idle and moving state.

Identifier	Lower bound	Upper bound
`sp_mov_spd`	0.85	1.15
`sp_mov_acc`	0.2	0.5

Spiders only allow for the control of their movement parameters, by selecting a movement speed coefficient and an acceleration value, the latter used during the transition from idle to moving.

Player Features

The following is a list of multiple-choice questions that inquire about the relevant player's demographic and gaming experience informations.

Identifier	Question	Answers
`plr_gender`	What gender are you?	Male, Female
`plr_age`	How old are you?	10 or less, fom 11 to 15, from 16 to 22, from 23 to 28, 29 or older
`plr_stealth_exp`	What's your expertise in stealth games?	Newbie, Average, Expert
`plr_mobile_exp`	How often do you play with mobile games?	Never or Rarely, Usually, Often
`plr_port_con_exp`	How often do you play with portable console games?	Never or Rarely, Usually, Often
`plr_home_con_exp`	How often do you play with home console games?	Never or Rarely, Usually, Often
`plr_pc_exp`	How often do you play with pc games?	Never or Rarely, Usually, Often
`plr_hb_runs`	How many times have you played HeartBot?	None, 1, 2, 3 or more

Apart from obvious questions about gender and age, there is a question regarding the experience the player has with stealth video games, as Heartbot is targeted to players with a consistent background in such games. The four questions coming next ask about the gaming habits of the player, because in the current state of the industry there are sharp differences between the scope of games on the different platforms.
Based on personal experience from Synthetica Lab mobile games are designed for short play sessions, do not usually sport complex graphics and are tailored to a more casual, less dedicated players.
Console games are by contrast very focused on the graphic experience, their targets being players that can invest time in long play sessions and are willing to confront themselves with difficult challenges.
As for personal computers, they are usually a replacement of home consoles for older demographic segments and as such their users share the same characteristics. The main difference between computer games and console games are the genres, with consoles having very few titles in the strategy genre and computers having almost no fighting games.
It is expected that players having a solid background in console or computer gaming will find Heartbot being easier than those who only play mobile games, with those experienced in stealth games further skewed towards perceiving the game as being easy to beat.

The last feature is used to identify players that have already completed the Heartbot beta and can be used to check if the game is perceived differently after being beaten.

Output Features

At the end of the gaming session, if either the game was abandoned or beaten, the player is required to complete another multiple choice questionnaire evaluating their perception of the game.

Identifier	Question	Answers
`out_ctrl_reac`	Were the controls reactive enough?	Yes, No
`out_ctrl_acc`	Was the control scheme easy to use?	Yes, No
`out_diff`	Overall, was the game difficult?	Easy, Just right, Difficult
`out_int`	Overall, was the game interesting?	Not interesting, Scarcely interesting, Interesting, Very interesting

The first two questions are about the control system, which is a crucial component of the game and requires careful tuning to accommodate different players expectations. The player is then asked an evaluation of the overall difficulty level, with the possibility of a neutral answer, and an evaluation of the interest the game raise.
The term interest has been chosen because it represents a positive evaluation of the game as a whole, not being tied to a particular aspect such as aesthetics or fun, and it has already been adopted in previous works in literature [18].

Game Duration

While the duration of a game session is actually an emergent feature, it is however a crucial metric for mobile games, as they are usually played as time fillers during commute or other short downtimes. Being able to predict abnormally shorter or longer sessions can help the game designers in tuning the game parameters for given time windows.
Session duration was collected, measured as the time in seconds between the start of the game and the time of completion or leave; these values will then be transformed in an ordinal categorical variable identifying the quartile they fell in.
With $Q1$, $Q2$ and $Q3$ being the hinges of the quartiles distribution,

Identifier	Values
`time`	$\leq Q1$, $> Q1$ and $\leq Q2$, $> Q2$ and $\leq Q3$, $> Q3$

It is expected that this output feature will be correlated with the difficulty metric, with shorter sessions being linked with lower difficulty evaluations and longer ones being tied to harder game settings.

3.5 Content of the Beta Version

The version of Heartbot used for the data collection was modified to accommodate the questionnaires, the randomization of the controllable features and the enforcement of a rigid sequence of six playable levels. Results from a game session are stored as text files and sent to a server for future elaboration.
The playable levels are the first ones from the full game and comprise four tutorial areas and two levels with complete gameplay; a more detailed description is provided below.
In the first level, the player just takes the first steps as Heartbot and is presented with the doors and keys mechanics.
The second area introduces the safe pipes and a lever activating a moving platform.
At the start of the third level, Heartbot unlocks the ability to dash and the noise mechanics is explained by a falling object.
In order to beat the fourth level, the player has to elude the surveillance of a fireflies hunter guarding a narrow passage. The only solution is to use the newly acquired dash to generate a noise and lure the hunter away from the passage, rapidly sneaking toward the exit door.
The fifth area tasks the player with picking up three top hats scattered though the level and to reach a key blocked inside a cage; this requires eluding a fireflies hunter patrolling a large part of the level and activating in the correct order a series of levers.
The final level increases the difficulty by confronting the player with two fireflies hunters, each moving in one of the two main areas the level is divided in. The player will have to carefully plan their movement to collect a set of top hats and the keys needed to advance.

Chapter 4

Methods

4.1 Introduction

This section will present the problem of classification and two solution strategies: Random Forest and Support Vector Machines. Then It will briefly discuss the process of feature elimination and its application to the previous algorithms.

4.2 Classification Problems

Classification is the problem of identifying to which of a set of categories or classes a new observation belongs, on the basis of a training set of data in which the category membership of each element is known.
Observations are in the form of numeric vectors $\mathbf{x}_1, \dots \mathbf{x}_n$, with a label y_i identifying to which output category the i-th observation belongs. The output labels of a binary problem are often subject to $y \in \{-1, +1\}$.
The elements of each observation are called features or explanatory variables and can take values from continuous intervals or discrete sets (either categorical or ordinal); let m be the number of such features for the problem at hand.
The objective of a classification problem is determining a function $\mathcal{D}(\mathbf{x})$ that outputs the same label that an expert supervisor would assign to $\mathbf{x}$.
A special case is the one of a linear separator: $\mathcal{D}(\mathbf{x})$ is a linear function of the features, usually a hyperplane in the feature space. A problem that admits a linear separator that never commits errors is said to be linearly separable.

4.3 Decision Trees

A Decision Tree [22] is a method of classification where a tree is built with the data from a training set data and new observations are classified by a

traversal of the tree.
The tree nodes are characterized as follow:

- internal nodes contain some kind of split function that, based on the features of an observation, directs the traversal to one of the node's children sub-trees
- leaf nodes contain the label of the output class, ending the traversal

The shape and size of the tree is determined by the split functions of the internal nodes; a desirable property of a decision tree is to be of minimum size, both for computational performance and for the Occam's Razor.
The problem of building such optimal tree is, however, NP-complete (see [23]): the majority of the tree learning algorithms are thus greedy procedures that construct the tree by recursively choosing a split amongst those that maximize a performance index. Such algorithms are often followed by a procedure to prune the resulting tree, both to reduce size and to prevent overfitting.

The selection of the split function is a crucial component of the algorithm and is usually based on a statistical score of the split informativeness.
Elementary split functions handle a single feature and can have different forms depending on its type, generally resulting in one or more hyperplanes orthogonal to its axis in the feature space; let A_a be the attribute at hand

- if the feature is discrete with z outcomes, valid split functions are either
  - $A_a =?$ that operates a selection with two outcomes, hence the node will have two children
  - $A_a \in G_i$ where $G = \{G_1, \ldots, G_g\}$, $2 \leq g \leq z$ is a partition of the output values; the node will have g children and can describe non-linear relations between the attribute and the output. The partition for the test is usually determined by greedy strategies
- if the feature is numerical, $A_a \leq \theta$ gives again a node with two children

Starting from the root and the entire training set S, the algorithm iteratively checks if the node is trivially a leaf or needs to become an internal node; if the cases in S belong to more than one output class, many different split functions are generated and evaluated to select the one that maximises some kind of performance score.
Such performance index are usually based on statistical properties of S and the candidate split function.
Most pruning strategies involve the removal of sub trees that, by statistical estimates, have misclassification rates higher than their siblings.

The main advantages of Decision Trees over other classifiers are that they can handle data with both continuous and discrete features without the need to transform them and that the resulting classifier is easily understandable by a human.
The main disadvantage is that much care must be taken in the pruning step to maintain the accuracy in classification.

4.4 Classification and Regression Trees

Proposed by Breiman in [24], CARTs can be used for either classification and regression, depending if the output value is discrete or numeric. This brief overview will be only about the classification mechanisms.
The algorithm is a variant of the general greedy procedure that only generates binary split functions, using as performance index the information gain of the split function computed with the Gini Impurity Index (see [25]).
Let S be the set of data to be split at the current node and $RF(C_i, S)$ the relative frequency of cases in S that have label C_i, $i = 1, \ldots, n$. To estimate the intrinsic information in S, the Gini Impurity index is used to estimate the probability that a random element of S is misclassified by only using the label distribution of the cases in the set:

$$I_{Gini}(S) = \sum_{i=1}^{|S|} RF(C_i, S)(1 - RF(C_i, S)) = 1 - \sum_{i=1}^{|S|} RF(C_i, S)^2$$

Let B be a candidate split function that splits S in subsets $S_1, \ldots, S_t$; the information gain of such split is then:

$$G(B, S) = I_{Gini}(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I_{Gini}(S_i)$$

This formulation, however, is biased toward split functions that generate more partitions and should be adjusted by including an estimate of the information potential of the split partitions:

$$P(S, B) = -\sum_{i=1}^{t} \frac{|S_i|}{|S|} \log(\frac{|S_i|}{|S|})$$

then the function to be maximised is $G(S, B)/P(S, B)$.

The pruning technique is the *minimal cost complexity pruning*, based on the assumption that the bias of a tree's resubstitution error linearly increases with he number of leaf nodes.
Let $R(T)$ be the resubstitution error rate of a subtree T, α a parameter of the estimate and $L(T)$ the number of T's leaves; then the score assigned to T is

$$R_\alpha = R(T) + \alpha L(T)$$

For any α, there exist only one T_α^* minimizing R_α and, although α can assume an infinity of values, the subtree candidates are a finite set; let t be the cardinality of this set.
Let $T_1 \succ T_2 \succ \cdots \succ T_t$ be the sequence of minimal subtrees obtained varying α from 0 to ∞.
An important property of this sequence is that the trees are nested: T_{i+1} is contained in T_i and an efficient weakest-link algorithm can thus be constructed to generate the sequence.
This pruning step is used inside a 10-fold cross validation scheme to obtain more sophisticated error estimates and determine smaller resulting subtrees, at the cost of a substantial increase in computation time.
Also, the algorithm limits the chosen subtree to be the smallest subtree whose estimated error rate differs for at most a single (estimated) standard error from the estimated error rate of the optimal subtree.

4.5 Random Forest

Random Forest was originally an umbrella term introduced by Breiman in [26] to describe an ensemble classifier that builds a set of weak tree classifiers by introducing some randomness in their construction process; their outputs are then combined according to a voting strategy to determine the classification output.
Breiman's paper formalizes the concept and analyses different techniques for the construction of a Random Forest; most notably, he sketches the proofs for important properties of Random Forest accuracy and ability to prevent overfitting. The term has since become the name of a classifier adopting CARTs as classifiers in conjunction specific randomization techniques.
The k-th classifier h_k in the forest has an associated random vector Θ_k, independent of other classifiers' vectors, but with the same distribution; Θ_k will be used to generate the classifier according to some strategy of inputting randomness, resulting in a function $h_k = h(\mathbf{X}, \Theta_k)$, with $\mathbf{X}$ the training set. The shape and distribution of Θ_k depends on the specific strategies chosen to build the Random Forest.
Let K be the number of trees and $I(\cdot)$ bet the indicator function; the define

$$mg(\mathbf{X}, Y) = \frac{1}{K}\sum_{k=1}^{K} I(h_k(\mathbf{X}) = Y) - \max_{j \neq Y}\left[\frac{1}{K}\sum_{k=1}^{K} I(h_k(\mathbf{X}) = j)\right]$$

as the margin function, measuring the extent to which the average number of votes for the combination $\mathbf{X}, Y$ for the right class exceeds the average vote for any other class.
The generalization error of the classifier is then

$$\mathbf{PE}^* = P_{\mathbf{X},Y}(mg(\mathbf{X}, Y) < 0)$$

It can be shown that, for the Law of Large Numbers, as K grows then $\mathbf{PE}^*$ converges to

$$P_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X},\Theta)=Y) - \max_{j\neq Y}[P_\Theta(h(\mathbf{X},\Theta)=j)] < 0)$$

This means that a Random Forest does not overfit, but produces a limiting value for the generalization error.
Now define the margin of a random forest as

$$mr(\mathbf{X},Y) = P_\Theta(h(\mathbf{X},\Theta)=Y) - \max_{j\neq Y} P_\Theta(h(\mathbf{X},\Theta)=j)$$

and the strength as

$$s = E_{\mathbf{X},Y}[mr(\mathbf{X},Y)]$$

Assuming $s \geq 0$, Chebychev's inequality gives

$$\mathbf{PE}^* \leq \frac{var_{\mathbf{X},Y}(mr(\mathbf{X},Y))}{s^2}$$

By exploiting the fact that the random vectors Θ are independent and identically distributed, a more revealing bound on $\mathbf{PE}^*$ can be obtained.
Defining

$$\hat{j}(\mathbf{X},Y) = \arg\max_{j\neq Y} P_\Theta(h(\mathbf{X},Y)=j)$$

$$rmg(\Theta,\mathbf{X},Y) = I(h(\mathbf{X},Y) - I(h(\mathbf{X},\Theta)=\hat{j}(\mathbf{X},Y))$$

Given Θ and Θ', let $\bar{\rho}$ be the mean value of the correlation between $rmg(\Theta,\mathbf{X},Y)$ and $rmg(\Theta',\mathbf{X},Y)$: then it can be shown that

$$\mathbf{PE}^* \leq \bar{\rho}\frac{1-s^2}{s^2}$$

This result, although loose, reveal the dependence of $\mathbf{PE}^*$ from the strength of the classifiers and the correlation between them. Such relation is then used by Breiman to guide the choice of randomization strategies in defining the final classifier.

The most popular version of Random Forest uses two different strategies to produce unrelated trees: bagging and random feature selection.
Bagging consists in the creation, for the k-eme tree, of a bootstrap T_k with replacement from the training set: the subset of observations selected by the bootstrap is called in-bag set, the remaining constitute the out-of-bag set. This partition determine that the k-eme tree, built on the in-bag-set, bears no bias towards the items in its out-of-bag set; testing the classification error on the out-of-bag set gives a fair estimate of the "true" error rate of the classifier.
Random feature selection acts during the construction of the trees by limiting

the choice of the split function of an internal node to those that use only a random subset of size *mtry* (parameter to the Random Forest) of features. The individual classifiers are unpruned CARTs built according to the above criteria.

4.6 Support Vector Machines

SVMs (Support Vector Machines) are supervised learning models that can be used for both classification and regression analysis. The basic formulation builds a non-probabilistic binary linear classifier, but extensions are available for multi-class classifiers and continuous regression; also, non-linear extensions are provided.
First introduced in [27], SVMs are now a popular solution for classification problems: the algorithm works by determining a maximum-margin linear classifier in the feature space, exploiting a convenient formulation of the problem.
In their basic version, SVMs produce a linear separator that maximizes its margin from the observations (interpreted as points in the feature space). The problem is formalized as a quadratic optimization:

$$\begin{cases} \min\limits_{\mathbf{w},b} \dfrac{\|\mathbf{w}\|}{2} \\ \qquad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 \quad \forall i = 1,\ldots,n \end{cases} \tag{4.1}$$

While solvable, this problem formulation can be translated in its dual form by using a vector of Lagrangean multipliers $\boldsymbol{\alpha}$ to make the constraints appear inside the objective function:

$$\begin{cases} \min\limits_{\mathbf{w},b} \max\limits_{\boldsymbol{\alpha}} \dfrac{\|\mathbf{w}\|}{2} - \sum\limits_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1\right] = \min\limits_{\mathbf{w},b} \max\limits_{\boldsymbol{\alpha}} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) \\ \qquad \alpha_i \geq 0 \qquad\qquad \forall i = 1,\ldots,n \end{cases} \tag{4.2}$$

By applying the property of the first minimization over $\mathbf{w}$ and b it is possible to relate bounds on $\boldsymbol{\alpha}$ to said elements:

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \Leftrightarrow \mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = 0 \Leftrightarrow 0 = \sum_{i=1}^{n} y_i \alpha_i$$

and the problem becomes:

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \sum\limits_{i=1}^{n} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \alpha_i y_i \alpha_j y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \quad \alpha_i \geq 0 \qquad\qquad \forall i = 1, \ldots, n \\ \quad \sum\limits_{i=1}^{n} y_i \alpha_i = 0 \end{cases} \tag{4.3}$$

of the parameters of the hyperplane, $\mathbf{w}$ need not be explicitly calculated, as the decision function can use the equivalent

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} y_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x})$$

and

$$b = y_k - \mathbf{w} \cdot \mathbf{x}_k \text{ for any } \alpha_k \neq 0$$

The strong point of this problem formulation is that the training data appear only in dot products between themselves; this fact will be exploited by the non-linear extension. Also, the $\boldsymbol{\alpha}$ multipliers will be different from 0 only for the observations that lie exactly on the margin of the hyperplane (called Support Vectors), thus reducing the size of the computation needed for building the hyperplane parameters.
To account for mislabeled training cases, the primal formulation can be modified to allow a vector of tolerances $\boldsymbol{\xi}$ in the bounds and penalizing such tolerances in the objective function by a controllable parameter C:

$$\begin{cases} \min\limits_{\mathbf{w},b} \frac{\|\mathbf{w}\|}{2} + C \sum\limits_{i=1}^{n} \xi_i \\ \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n \\ \quad \xi_i \geq 0 \qquad\qquad\qquad\quad \forall i = 1, \ldots, n \end{cases} \tag{4.4}$$

the dual formulation is then modified as follows:

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \sum\limits_{i=1}^{n} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \alpha_i y_i \alpha_j y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \quad 0 \leq \alpha_i \leq C \qquad\qquad \forall i = 1, \ldots, n \\ \quad \sum\limits_{i=1}^{n} y_i \alpha_i = 0 \end{cases} \tag{4.5}$$

$$b = y_k (1 - \xi_k) - \mathbf{w} \cdot \mathbf{x}_k \text{ for } k = \arg\max_k \alpha_k$$

The $\boldsymbol{\xi}$ tolerances allow for observation points to be inside the margin of the separating hyperplane, thus giving them a $\alpha > 0$ even if not Support Vectors.

The formulae expressed above refer to an SVM for linearly separable data, at most with some noisy observations; SVMs can be adapted to handle non-linearly separable data by mapping them to a space with a higher number of dimensions and searching for a linear separator in said new space.
Let $\varphi(\mathbf{x}_i)$ be the mapping, in a higher dimensional space, of $\mathbf{x}_i$ and let $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$; by applying the formulae from the linear case to the mappings of the data it results:

$$\begin{cases} \max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \alpha_i \geq 0 \qquad \forall i = 1, \ldots, n \\ \sum_{i=1}^{n} y_i \alpha_i = 0 \end{cases} \tag{4.6}$$

and

$$\mathbf{w} \cdot \varphi(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x})) = \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x})$$

This use of the classifier is known as "kernel trick" and K is called a kernel function: to be of practical use, a proper kernel function must satisfy Mercer's condition: for any $g(\mathbf{x})$ such that $\int g^2(\mathbf{x}) d\mathbf{x}$ is finite,

$$\int K(\mathbf{x}, \mathbf{z}) g(\mathbf{x}) g(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0$$

The condition guarantees the existence of a solution for the dual problem in the SVM algorithm and makes the kernel function be continuous, symmetric and have a positive definite Gram matrix.
Standard non linear kernel functions are:

- **Gaussian radial basis function:** $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$ for $\gamma > 0$
- **Polynomial:** $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + c)^d$
- **Sigmoid:** $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma(\mathbf{x}_i \cdot \mathbf{x}_j) + c)$

Advantages of SVMs over other classifiers are the small number of tuning parameters (dependent on the kernel function adopted) and the computational efficiency in solving the underlying quadratic optimization problem; the resulting classifier, however, does not lend itself to an easy interpretation. Another disadvantage is that SVMs may underperform when the training data have much different domain ranges, requiring a standardization preprocessing step.

4.7 Feature Selection

Feature selection is the process of choosing the most relevant subset of dimensions of the feature space to build a more compact classifier. Its usefulness stems from the reduction of training times and the possibility of removal of redundant or irrelevant features from the original problem formulation, to reduce the risk of overfitting the training data.
Different feature selection methods have been studied in literature, such as in [28], which describes three main strategies: wrappers, filters and embedded methods.
In wrapper methods, after an enumeration of all possible feature subsets, each one is used to build a different classifier. This is then evaluated through some performance score, such as classification error on a hold out set. These methods are computationally expensive and result are intractable when the feature set are too large or the construction and testing of new classifiers take up excessive resources.
Filter methods involve a scoring of single features based on statistical properties of the training set, such as correlation of features with the output labels or class distribution. Usually these methods are much faster to apply than the wrappers, but since they are not tied to a specific predictive model, they may result in lower performances. Filters have been used as a pre processing step to reduce the feature space prior to the application of wrapper methods.
Embedded methods is an umbrella term for methods that perform feature selection as a part of the model construction.These algorithms act iteratively by building a classifier and retrieving from its internal structure a ranking of the features; selection takes place and based on the ranking, a new classifier is built until a performance threshold is reached. The computational expenditure is thus in a middle ground between wrappers and filters.
When a classification problem has many features whose effects are partially overlapping, filter methods that examine correlation of single features with the output label can not make use of their mutual importance. As a result, the ranking produced by these methods are more suited to univariate classification and may not perform well for feature selection for a multivariate classifier.
For these problems, it is then crucial to find subsets of features whose combined effect retain the most descriptive power. A common embedded approach, when a feature ranking is easily available, is to recursively build smaller subsets $F_0 \succ F_1 \succ \cdots \succ F_r$ where F_0 is the full feature set and F_{i+1} is obtained by removing the bottom ranked features from F_i. Feature removal can take place both by removing a single feature and removing an entire subset; the method is called Recursive Feature Elimination and was proposed for SVMs in [29].
The assumption of this method is that a multivariate classifier produces rankings that promote entire subsets of features that "work well" together,

while penalizing features that are both weak alone and in subsets.

4.7.1 Feature Ranking with Random Forest

The bagging mechanism of Random Forests can be exploited to retrieve unbiased estimates of single trees performance indices from the evaluation of the out-of-bag observations.
To observe the effect of a given feature on the classification output, Breiman proposes to randomly permutate that feature values in the training set and to perform classification of this new data; the results are then compared to the original.
For feature ranking, two criteria are used: mean increase in classification error, called by Breiman Variable Importance, and total decrease of nodes' Gini Impurity Index. In [30] feature selection with Random Forest is discussed and compared to other classifiers on real and fictitious data, proposing Variable Importance as the better criteria for feature selection mechanisms to be built upon.

4.7.2 Feature Ranking with SVMs

In the work about Recursive Feature Elimination, Guyon et al. propose the use of a particular feature ranking derived from the internal parameters of a Support Vector Machine.
For SVM the cost function being maximized is

$$J = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_i \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Let H be the matrix such that $H_{hk} = y_h y_k K(\mathbf{x}_h, \mathbf{x}_k)$; the cost function can then be rewritten as minimization of

$$J = \frac{1}{2} \boldsymbol{\alpha}^T H \boldsymbol{\alpha} - \sum_{i=1}^{n} \alpha_i$$

To compute the effect of the removal of feature i, introduce

$$DJ(-i) = J - J(-i) = \frac{1}{2} \boldsymbol{\alpha}^T H \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T H(-i) \boldsymbol{\alpha}$$

where $H(-i)$ is the matrix H computed on observations whose feature i is removed.
When $DJ(-i)$ has a greater value, it means that the removal of feature i determines an estimate of the cost function which is a better solution for the minimization problem. The best candidate for removal is thus the feature i

that minimizes $DJ(-i)$. The recomputation of H is an expensive operation for large datasets, but because the $\alpha_j > 0$ only for support vectors, only a possibly much smaller subset of the data has to be taken into effect.
In the case of SMV with a linear kernel, the coefficient for feature i in the cost function can be computed explicitly with $w_i = \sum_{j=1}^{n} \alpha_j y_j (\mathbf{x}_j)_i$ where $(\mathbf{x}_j)_i$ is the value of feature i for the observation $\mathbf{x}_j$.
Then $J(-i)$ and $DJ(-i)$ can be computed in closed form with

$$J(-i) = \sum_{h=1}^{n} \alpha_h - \frac{1}{2} \sum_{h=1}^{n} \sum_{j=1}^{n} \alpha_h y_h \alpha_j y_j \left[\mathbf{x}_h \cdot \mathbf{x}_j - (\mathbf{x}_h)_i (\mathbf{x}_j)_i \right]$$

$$J(-i) = J - \frac{1}{2} \sum_{h=1}^{n} \sum_{j=1}^{n} \alpha_h y_h \alpha_j y_j (\mathbf{x}_h)_i (\mathbf{x}_j)_i$$

$$J(-i) = J - \frac{1}{2} \sum_{h=1}^{n} \alpha_h y_h (\mathbf{x}_h)_i \sum_{j=1}^{n} \alpha_j y_j (\mathbf{x}_j)_i = J - \frac{1}{2} w_i^2$$

$$DJ(-i) = J - J(-i) = \frac{1}{2} w_i^2$$

Linear kernel SVMs thus allow for a computationally cheap retrieval of the feature ranking.

Chapter 5

Experiment Design

5.1 Introduction

This chapter will present the collected data and formulate the decision problems that will be the object of the analysis. Then it will describe the experiment design and the actual analysis, divided in a preprocessing phase and the application of the different methods.

5.2 Collected Data

The data set consists of 49 game sessions, collected from December 11th 2014 to March 10th 2015 at different public and private events, primarily involving players in the author's age group. The number of observations might not be high enough to obtain good classification results, but it provides sufficient data to evaluate the application of the methodologies proposed.
At the start of each session, the game forces the player to answer a questionnaire with the questions detailed in the third chapter; for each controllable feature, it generates a value from a uniform distribution between its lower and upper bound.

5.2.1 Players Characteristics Breakdown

The following table represents the breakdown of the observations in the respective classes; a detailed description follows.

Identifier	Classes Breakdown	Classes Identifier
`plr_gender`	40, 9	Male, Female
`plr_age`	11, 35, 3	from 16 to 22, from 23 to 28, 29 or older
`plr_stealth_exp`	23, 19, 7	Newbie, Average, Expert
`plr_mobile_exp`	21, 21, 7	Never or Rarely, Usually, Often
`plr_port_con_exp`	27, 14, 8	Never or Rarely, Usually, Often
`plr_home_con_exp`	27, 16, 6	Never or Rarely, Usually, Often
`plr_pc_exp`	16, 11, 22	Never or Rarely, Usually, Often
`plr_hb_runs`	43, 3, 1, 2	None, 1, 2, 3 or more

Of the 49 sessions, 40 were from male players and only 9 from females; as for players' age, 11 were "from 16 to 22", 35 were "from 23 to 28" and only 3 were "29 or older", thus lacking in the representation of younger age groups. In regard to the experience level with stealth games, 23 players defined themselves as "Newbie", 19 as having "Average" experience and only 7 as being "Expert".
The question about the expertise with mobile gaming sees 21 players playing "Never or Rarely", 21 playing "Usually" and only 7 playing "Often". Similar results are collected for experience with portable consoles (27, 14, 8) and home consoles (27, 16, 6). Conversely, experience with personal computers is generally higher, with a breakdown of 16, 11 and 22 in the three classes. These values identify a player population that is more oriented to playing on the computer rather than on other devices; this however proves useful in validating Heartbot's ability to entice mid core gamers.
As for having already played the game, 43 were playing Heartbot for the first time, 3 had already tried the game once, 1 had played 2 times and 2 players had completed the game three or more times.

5.2.2 Output Formalization and Breakdown

It was decided to formulate the actual classification problems in a binary form, as all the output taken in consideration were ordinal; the classifiers would then act as separators between "low" and "high" values for each output category, with split values between adjacent classes.
The following table describes the breakdown of the output labels in the re-

sulting decision problems and is followed by a more detailed description.

Identifier	Classes Breakdown	Classes Identifier
`out_ctrl_reac`	25, 24	Yes, No
`out_ctrl_acc`	34, 15	Yes, No
`out_diff_1`	15, 34	Easy, Just Right or Difficult
`out_int_2`	10, 39	Not Interesting or Scarcely Interesting, Interesting or Very Interesting
`out_int_3`	39, 10	Not Interesting or Scarcely Interesting or Interesting, Very Interesting
`time_1`	13, 36	$\leq Q_1, > Q_1$
`time_2`	25, 24	$\leq Q_2, > Q_2$
`time_3`	37, 12	$\leq Q_3, > Q_3$

Regarding the distribution of the original values, 25 players evaluated the controls as reactive enough, while 24 considered them unresponsive; 34 considered the control system as being accessible, while 15 considered it not easy to use.
In the question about the game's difficulty, 15 players evaluated the game as being "Easy", 32 as being "Just Right" difficult and only 2 found the game "Difficult". Since so few players considered the game as being "Difficult", differentiating between that output class and the others would be of little or no importance and thus only one problem is derived.
As for the interest evaluation, 4 players found the game to be "Not Interesting", 6 graded it "Scarcely Interesting", 29 thought it was "Interesting" and 10 answered "Very Interesting". Similarly to the discussion above, the classes "Not Interesting" and "Scarcely Interesting" were combined and two binary problems were defined, respectively differentiating between sessions "Interesting" or more and between "Very Interesting" sessions.
Concerning the duration of the game sessions, they were divided on the basis the quartile they fell in; the shortest game session lasted 350s and the longest 1237s, with the first quartile being Q_1 =598s, the median Q_2 =722s and the third quartile Q_3 =931s. These values prove that the game can be played in short sessions, which is a desired property for mobile games.
Only problems `out_ctrl_reac` and `time_2` feature an almost even distribution of the output labels, while the others are all imbalanced; this can result

in the training of classifiers that always return the majority label. The evaluation of such classifiers can not be simply reduced to the computation of their accuracy, but must take into account the distribution of classes in assessing the quality of the prediction. One such metric for binary classification is the Matthews Correlation Coefficient.

5.3 Performance Measures

The most immediate measure that can be used to describe the quality of a classifier is the classification accuracy, but when used for problems with imbalanced classes it might lose descriptive power. In such problems, a majority classifier that always output the class with the most individuals achieves accuracy as high as the percentage of observations in the majority class, while being obviously an undesired result.

5.3.1 Matthews Correlation Coefficient

The Matthews Correlation Coefficient is a numerical measure of the quality of a binary classification, computed from the confusion matrix. It is regarded as a more descriptive metric than the simple accuracy of the prediction, since it measures the correlation between predicted and real labels and maintains its usefulness in the case of imbalanced classes.
Given a confusion matrix

		Real Labels	
		+1	-1
Predicted Labels	+1	TP (True Positives)	FP (False Positives)
	-1	FN (False Negatives)	TN (True Negatives)

Matthews Correlation Coefficient is computed as

$$MCC = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$

The coefficient takes values in $\{-1, 1\}$ with $MCC = 1$ indicating a perfect prediction, $MCC = 0$ pointing out a prediction no better than a random one and $MCC = -1$ showing total disagreement between predicted and real labels.
A majority classification would result in a row composed of zeroes in the confusion matrix, thus giving a Matthews Correlation Coefficient of 0.

5.4 Data Analysis

This section will present the approach taken for the application of Feature Selection to the decision problems defined in the first section. At first, the collected data has to be organized in a train set and a test set in order to obtain unbiased results from the following analysis. Then, the actual feature selection takes place and the results are evaluated on the training set, using stratified cross validation, to determine the best feature set for each problem. Finally, a classifier is trained with the optimal features and evaluated on the test set.

5.4.1 Data Preprocessing

In [31] Furlanello et al. discuss the effects of selection bias in the evaluation of feature elimination methods and delineate the need of a careful preprocessing of the data to remove such bias.
The crucial point is in splitting the available data in a train set and a test set: the feature selection process shall only take place on the train data, using them in any way, while the test data will only be used for a final evaluation of the selected feature sets.
In this work, the original data was first partitioned in four groups, by the player's gender and two age group (isolating the last age group by the others); as only 3 players were in the older subset, this division resulted mainly in a gender separation. Then a random sampling was operated to obtain a train set of roughly one fourth of total data, favoring a bigger train test and preserving the proportion of players in the groups; the remaining observations constitute the test set. The resulting train set counts 38 observations, while the test set size is 11.
For each problem a stratified folding of the observations in the train set was performed, to be used in the future steps of the analysis. Given c_{-1}, c_{+1} the count of respectively negative and positive labels in the train set, the number of folds was chosen as $\min(10, c_{-1}, c_{+1})$ and the observations were placed in each fold with a random sampling that preserved the starting proportions.
In addition to the stratified foldings, for each problem 100 stratified bootstraps with replacement were produced; again, the train set was split by the output labels values and every bootstrap consisted of a random sampling with replacement from each subset, thus preserving the labels distribution.

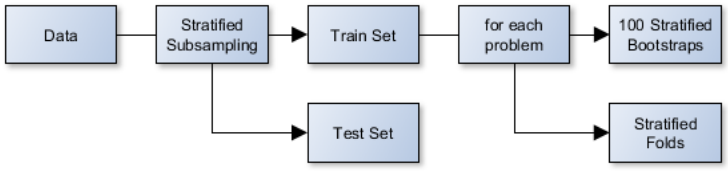

Figure 5.1: Data preprocessing

5.4.2 Evaluation of Recursive Feature Elimination

In the previous chapter two families of classifiers were presented: Random Forests and Support Vector Machines. The general Recursive Feature Elimination method was presented and for both classifiers a method of feature ranking was discussed.
Recursive Feature Elimination will be applied to Random Forests, SVM with Linear Kernel and SVM with Radial (Gaussian radial basis) Kernel. As Random Forests training depends on pseudo random number generation, the algorithm will be repeated 100 times on the same train test and the resulting features ranks will be averaged to obtain a final elimination order. Since SVMs have a deterministic training algorithm, the elimination order will be derived by a single run on the train set. Additionally, all three methods will be applied to the bootstraps generated during the data preprocessing phase and the resulting feature ranks averaged: it is expected that SVMs will benefit from this approach, while for Random Forest an increase in results noise might appear, since the classifier performs an internal bootstrap phase.
Thus, six different feature elimination set ups are analysed:

- Random Forest RFE repeated 100 times
- single SVM with Linear Kernel RFE
- single SVM with Radial Kernel RFE
- Random Forest RFE repeated on 100 bootstraps
- SVM with Linear Kernel RFE repeated on 100 bootstraps
- SVM with Radial Kernel RFE repeated on 100 bootstraps

Once a definitive feature ranking is obtained, the performance of their iterative removal is assessed with cross validation, using the folding created in the preprocessing phase; for the single run of SVM, this step is superfluous as results do not differ from the initial feature elimination.
The metrics collected are the prediction accuracy, accuracy standard deviation and the average Matthews Correlation Coefficient. The results are then analysed by inspection and for each method an optimal number of surviving

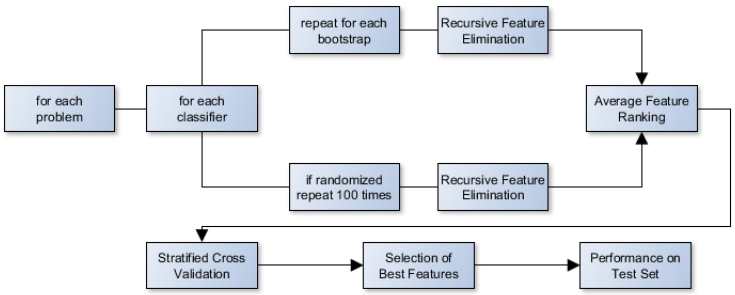

Figure 5.2: Recursive Feature Elimination evaluation

features is selected, by choosing the peak in Matthews Correlation Coefficient that requires the least number of features.
Finally, for each combination of problem and algorithm, a classifier using the optimal feature set is built on the train set and tested on the test set to obtain unbiased performance metrics.

5.5 Software Tools

Data collection was integrated in the game, using the C# language and the Unity game engine.
The analysis framework was implemented in the R programming language[1], using the packages `randomForest`[2] and `e1071`[3] respectively for Random Forests and Support Vector Machines.
With Random Forests, the number of trees was chosen as 1000 and the rest of parameters were left with the default values.
For SVMs, parameter tuning took place with an extensive grid search using the full train set, selecting for each kernel and problem pair the parameters that maximized the average MCC from stratified cross validation.

[1] http://www.r-project.org/
[2] http://cran.r-project.org/web/packages/randomForest/index.html
[3] http://cran.r-project.org/web/packages/e1071/index.html

Chapter 6

Experimental Results

6.1 Introduction

This chapter will present and discuss the results from the evaluation described in the previous chapter. For each one of the problems previously defined and for each classification method, an optimal feature set will be derived by inspection of the stratified cross validation on the train set. Then the different methods will be evaluated on the test set.
The plots will show the mean accuracy and its standard deviation in the training phase against the increasing number of surviving features. The mean MCC is shown as a dashed line, with its values scale to be fitted in the images.
For each problem and classification method, a table summarises the method's optimal feature set and its performance on the train and test set. A discussion of the results obtained follows.

6.2 Controls Reactivity

In the train set 20 sessions evaluated the controls as reactive, while 18 as not reactive; in the test set there were 5 evaluations as reactive and 6 as not reactive.

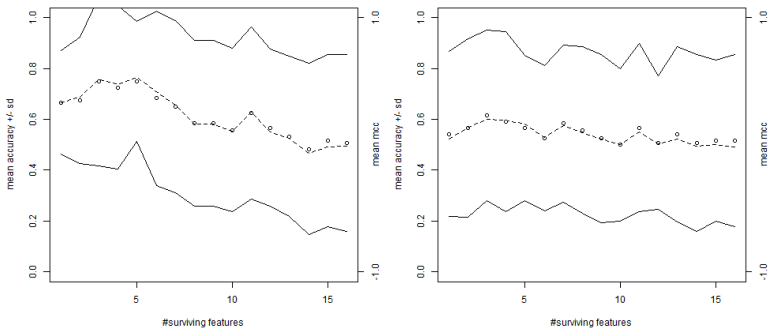

Figure 6.1: Train performance of Random Forest on `out_ctrl_reac`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

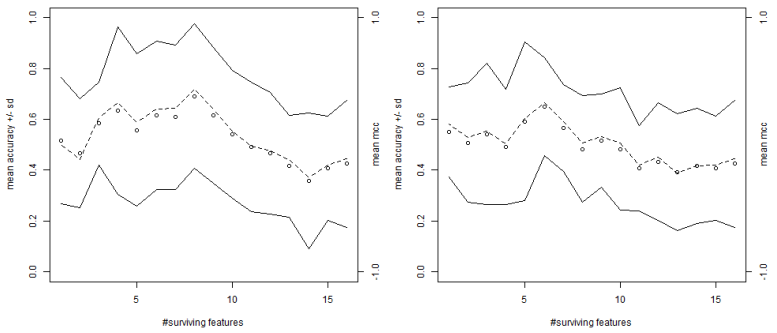

Figure 6.2: Train performance of SVM with linear kernel on `out_ctrl_reac`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

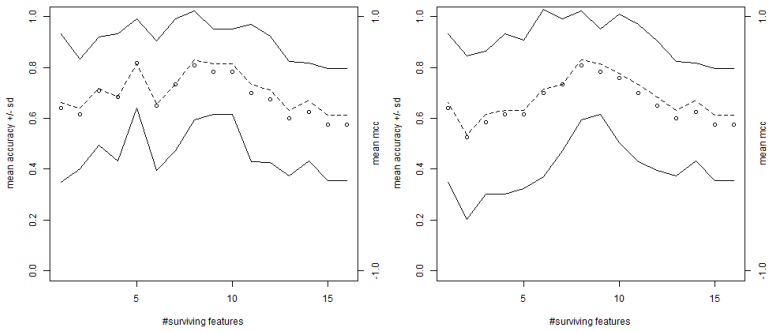

Figure 6.3: Train performance of SVM with radial kernel on `out_ctrl_reac`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.75	0.515	0.636	0.311	`plr_stealth_exp`, `bot_mov_spd`, `sp_mov_spd`
Random Forest on bootstraps	0.617	0.2	0.545	0.1	`sp_mov_spd`, `sp_mov_acc`, `bot_mov_spd`
Single linear SVM	0.633	0.331	0.545	0.149	`plr_stealth_exp`, `plr_pc_exp`, `bot_mov_spd`, `sp_mov_spd`
Linear SVM on bootstraps	0.65	0.331	0.545	0.149	`plr_pc_exp`, `bot_mov_spd`, `sp_mov_spd`, `plr_gender`, `plr_stealth_exp`, `plr_mobile_exp`
Single radial SVM	0.817	0.631	0.545	0.149	`sp_mov_spd`, `plr_gender`, `fh_mov_acc`, `bot_mov_spd`, `plr_stealth_exp`
Radial SVM on bootstraps	0.808	0.658	0.545	0.149	`sp_mov_spd`, `plr_stealth_exp`, `plr_gender`, `plr_mobile_exp`, `fh_mov_acc`, `plr_port_con_exp`, `bot_mov_spd`, `plr_pc_exp`

As the data are very balanced, MCC computed in the training phase closely follows the accuracy values.
In the training phase, SVMs with radial kernel performed much better than the other methods, closely followed by repeated Random Forest. The other

methods showed significantly lower performances, with Random Forest trained on bootstraps as the worst.
Results on the test set showed that in this problem only the Random Forest classifier maintained its prediction performance, although with slightly lower accuracy and MCC. SVMs with radial kernel performed on the test no better than SVMs with linear kernel.
The most important features in predicting the controls reactivity evaluation appear to be, in this order: the players' experience in stealth games, Heartbot's movement speed and the spiders' movement speed. The gender of the player is selected by the SVMs with radial kernel and while missing by the optimal feature set of repeated Random Foret, it is ranked fourth there.
Examining correlation between the output and these variables, no result was significant enough, as p-values were high; this suggests that the relationships between these values were non linear.

6.3 Controls Accessibility

The train set counts 27 sessions where the controls were evaluated as being easy to use and 11 contrary to that statement. The test set contained 7 evaluations as easy to use and 4 contrary.

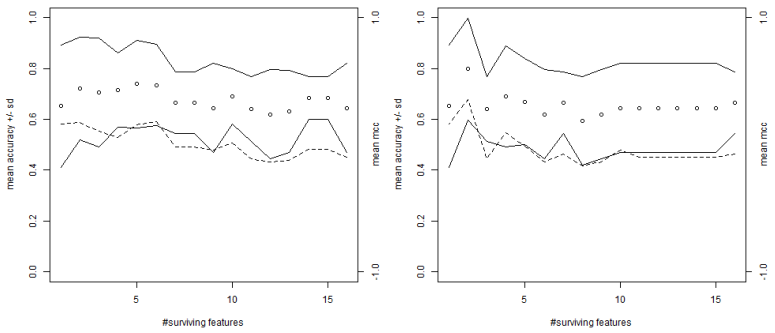

Figure 6.4: Train performance of Random Forest on `out_ctrl_acc`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

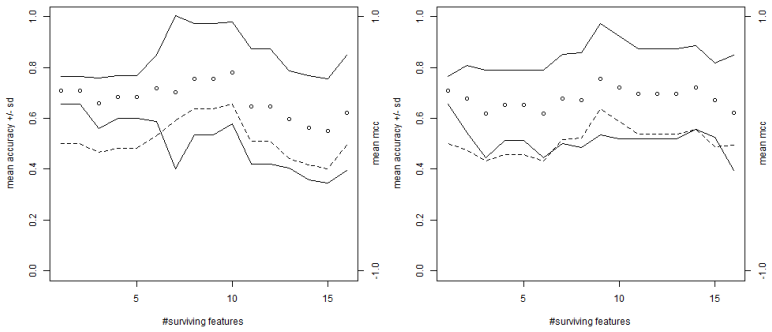

Figure 6.5: Train performance of SVM with linear kernel on `out_ctrl_acc`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

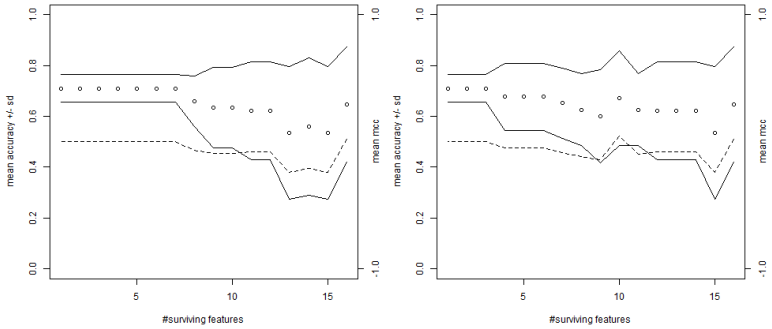

Figure 6.6: Train performance of SVM with radial kernel on `out_ctrl_acc`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.722	0.178	0.636	0.134	`bot_snd_rad`, `plr_stealth_exp`
Random Forest on bootstraps	0.798	0.359	0.545	0.069	`bot_snd_rad`, `sp_mov_spd`
Single linear SVM	0.755	0.278	0.545	-0.0386	`plr_pc_exp`, `plr_hb_runs`, `plr_mobile_exp`, `sp_mov_spd`, `fh_view_cone`, `plr_age`, `plr_port_con_exp`, `fh_mov_spd`
Linear SVM on bootstraps	0.755	0.278	0.545	-0.0386	`plr_pc_exp`, `plr_age`, `plr_mobile_exp`, `sp_mov_spd`, `sp_mov_acc`, `fh_mov_spd`, `plr_hb_runs`, `fh_view_cone`, `plr_port_con_exp`
Single radial SVM	0.71	0	0.636	0	`fh_mov_acc`
Radial SVM on bootstraps	0.672	0.0446	0.545	-0.0386	`fh_mov_spd`, `sp_mov_spd`, `plr_age`, `plr_pc_exp`, `fh_mov_acc`, `sp_mov_acc`, `plr_stealth_exp`, `plr_gender`, `plr_mobile_exp`, `plr_port_con_exp`

The train set shows a marked imbalance in the output classes and the MCC reveals many instances of majority classification.
It was unexpected to see SVMs with radial kernel giving such bad results; Random Forests quickly decrease performance as features are added, while SVMs with linear kernel show a peak in accuracy and MCC in the proximity of their eighth feature.
The evaluation on the test set selects Random Forest as the more generalizable method, as it is the only classifier that maintains MCC sign. Also, it is interesting to see the classifier trained on the bootstraps obtaining better test performance.
The features selected by the repeated Random Forest were the maximum

radius of expansion of a collision noise and the players' experience in the stealth games; the classifier trained on the bootstraps selected the spiders' movement speed as second feature.
The presence of the radius feature is surprising; however, as it regulates a mechanic that the players had to exploit to advance in the game, it appears as it contributed to their own evaluation in being able to use the control scheme. The spiders' movement speed, similarly, can be interpreted in having a punishing effect on players unable to effectively control Heartbot. As for the experience in stealth games, it is to be related to the pacing of the action: more expert players are able to predict the action rather than being forced to react to it, thus being more at ease with the controls.
The game designer expected to see a prevalence in selection of feature related to the players' gaming experience; the two best classifier in the training phase obtained moderately good results by selecting a lot of said features.
However, examination of the correlation of the features with the output did not reveal any statistically significant relationship, though the data were in agreement with general speculations from above.

6.4 Difficulty

As discussed in the previous chapter, only one decision problem was formulated on the difficulty level, because the class "Difficult" was heavily under represented.
In the train set, 11 players graded the game as "Easy", while 27 evaluated its difficulty level as "Just Right" or "Difficult". In the test set the output labels were respectively 4 and 7.

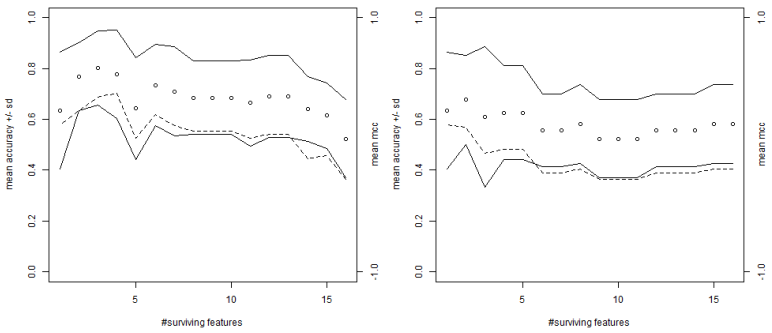

Figure 6.7: Train performance of Random Forest on `out_diff_1`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

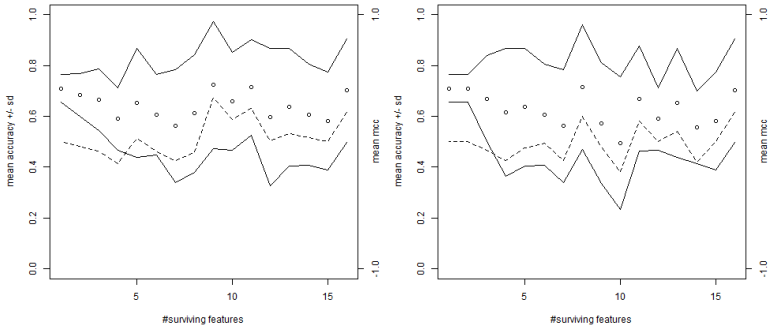

Figure 6.9: Train performance of SVM with radial kernel on `out_diff_1`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

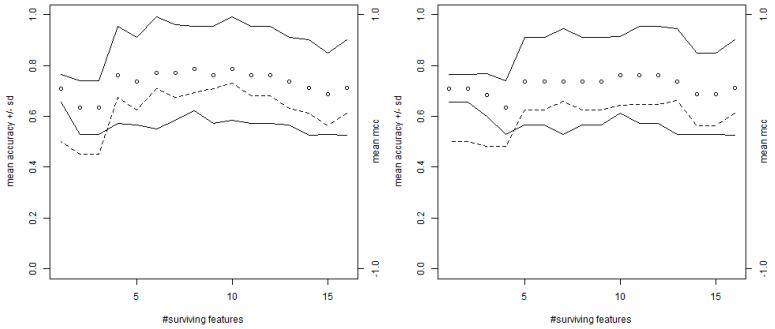

Figure 6.8: Train performance of SVM with linear kernel on `out_diff_1`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.777	0.408	0.727	0.386	`fh_mov_spd`, `plr_gender`, `plr_pc_exp`, `plr_age`
Random Forest on bootstraps	0.635	0.159	0.727	0.386	`fh_mov_spd`

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Single linear SVM	0.763	0.352	0.636	0	plr_age, plr_gender, fh_view_cone, fh_mov_spd
Linear SVM on bootstraps	0.738	0.319	0.636	0.134	plr_gender, fh_view_cone, bot_snd_rad, fh_mov_spd, plr_age, sp_mov_acc, plr_hb_runs
Single radial SVM	0.723	0.35	0.455	-0.179	plr_pc_exp, plr_mobile_exp, bot_mov_spd, bot_dsh_spd, plr_stealth_exp, sp_mov_acc, fh_mov_acc, plr_home_con_exp, plr_age
Radial SVM on bootstraps	0.715	0.2	0.636	0.214	plr_pc_exp, bot_mov_spd, plr_stealth_exp, sp_mov_acc, fh_mov_acc, bot_dsh_spd, plr_mobile_exp, fh_mov_spd

The distribution of output labels in the train and test set is almost specular to the previous problem.
Similar to the results before, Random Forests show a quick performance fall as features are added after the first few; SVMs with linear kernel default to a majority classifier when supplied with too few features and SVMs with radial kernel reach their peak performance near their eighth selected feature. Random Forests results as the best classifier on the train set in its repeated version; SVMs with linear kernel use a similar number of features and have comparable performance. SVMs with radial kernel require a larger number of feature and do not provide significant performance improvement.
On the test set, both Random Forests training methods obtain the same evaluation, beating the SVMs trained with bootstraps. SVMs applied once are slightly inferior to a majority classifier, as evidenced by the MCC.
Almost all classifiers with good performance selected the fireflies hunter

movement speed; the hunters are the main threat to be evaded and thus determine the game's challenge. Other features frequently selected are the players' gender and age; repeated Random Forest also selected the PC experience.
The game designers were surprised that the stealth experience was selected only by SVMs with radial kernel, which also required many more features than the other methods to behave properly. A similar consideration applies to the fireflies hunter view cone, that was expected to have the same importance as the enemies' movement speed; an enemy, however, is only as dangerous as the speed it can reach and its detection mechanism can be evaded easily independently from its size once the player masters the enemies' behaviour.
Examining correlation of the discussed variables, no statistically significant relationships were found, though the general speculations above were in agreement with the data. The fireflies hunters' view cone angle had similar correlation results to those of the fireflies hunters' movement speed.

6.5 Interest

As explained in the previous chapter, two binary problems were formulated regarding the interest evaluation: the first discerns between positive interest evaluations and negative ones; the second identifies the game sessions that were deemed as the most interesting.

6.5.1 Positive Interest Against Negative Interest

The train set contains 9 sessions where the game was evaluated as being "Not Interesting" or "Scarcely Interesting", while 29 players found it to be "Interesting" or "Very Interesting". As for the test set, the breakdown was respectively of 1 and 10 observations.
The test set is so heavily imbalanced that the evaluation of the MCC on the test will be mostly determined by the classifier's ability to predict the first class and as such results will be unpredictable.

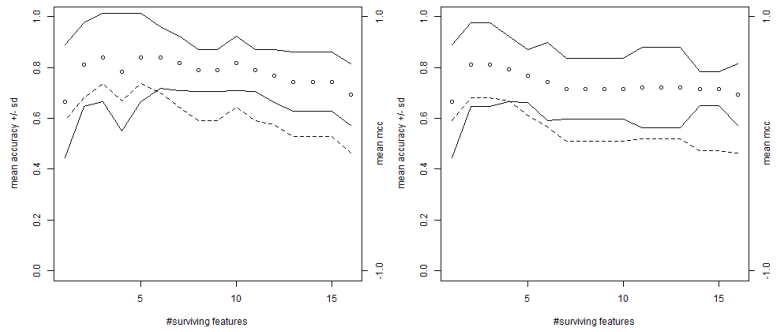

Figure 6.10: Train performance of Random Forest on `out_int_2`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

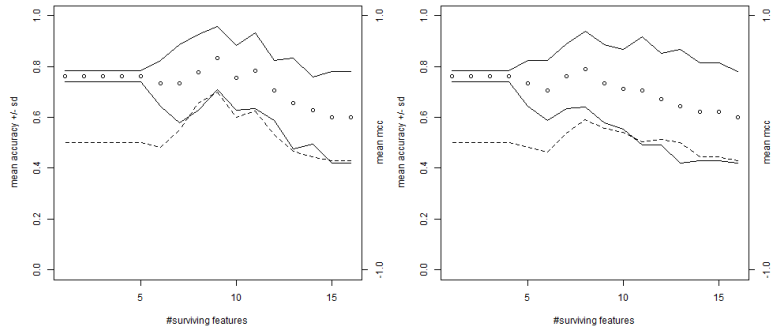

Figure 6.11: Train performance of SVM with linear kernel on `out_int_2`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

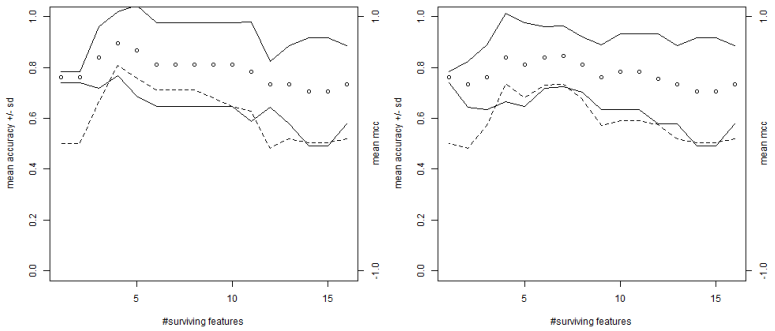

Figure 6.12: Train performance of SVM with radial kernel on `out_int_2`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.839	0.475	0.818	-0.1	`bot_mov_spd`, `plr_pc_exp`, `plr_home_con_exp`
Random Forest on bootstraps	0.811	0.364	0.727	-0.149	`bot_mov_spd`, `plr_pc_exp`
Single linear SVM	0.833	0.397	0.818	-0.1	`fh_mov_spd`, `plr_pc_exp`, `plr_home_con_exp`, `plr_mobile_exp`, `bot_snd_rad`, `plr_stealth_exp`, `plr_hb_runs`, `fh_view_cone`, `bot_dsh_spd`
Linear SVM on bootstraps	0.789	0.185	0.909	0	`plr_pc_exp`, `plr_home_con_exp`, `plr_mobile_exp`, `fh_mov_spd`, `plr_stealth_exp`, `sp_mov_spd`, `fh_mov_acc`, `bot_mov_spd`

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Single radial SVM	0.894	0.62	0.818	-0.1	`bot_mov_spd`, `plr_home_con_exp`, `fh_mov_acc`, `plr_pc_exp`
Radial SVM on bootstraps	0.839	0.472	0.818	-0.1	`bot_mov_spd`, `plr_pc_exp`, `fh_mov_acc`, `bot_snd_rad`

Again, Random Forests rapidly decrease their performance after the first few features and SVMs with linear kernel reach their peak at about eight features. SVMs with radial kernel instead seem to follow the performance of RandomForests.
On the train set, all classifiers have good accuracy and MCC values, with single SVM with radial kernel being the top achiever. Close contenders are the single SVM with linear kernel and the repeated Random Forest.
Results on the test set are instead disheartening: at best, a majority classifier is obtained and most of the classification methods do not overcome that threshold.
As for the most informative features, all the best methods selected a set containing at least Heartbot's movement speed and the players' PC and home console experience. The selection of the fireflies hunter acceleration is surprising, as it should be less important than features related to the difficulty evaluation.
Initial interpretation hints at the protagonist's movement speed being positively correlated with the interest level, but no other guess could be made with confidence. While the data supported the claim, no statistically significant relationship was found.

6.5.2 Highest Level of Interest

In the train sets 31 players evaluated the game as being one of "Not Interesting", "Scarcely Interesting" or "Interesting" and 7 found it "Very Interesting"; in the test set the breakdown was respectively 8 and 3.

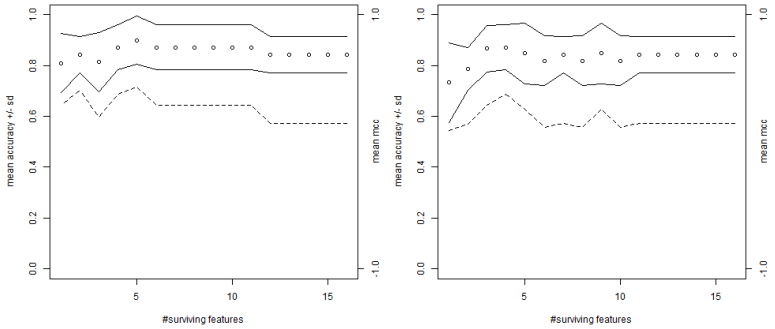

Figure 6.13: Train performance of Random Forest on `out_int_3`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

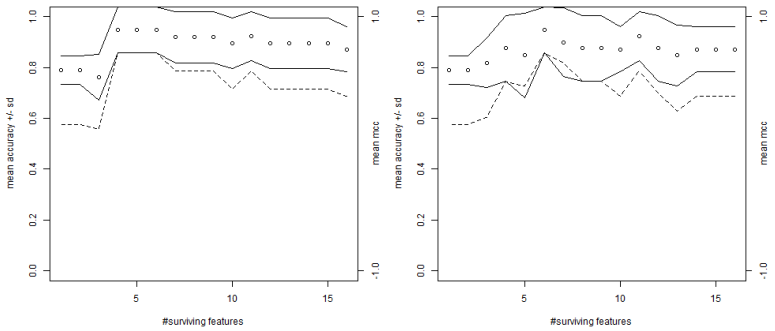

Figure 6.14: Train performance of SVM with linear kernel on `out_int_3`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

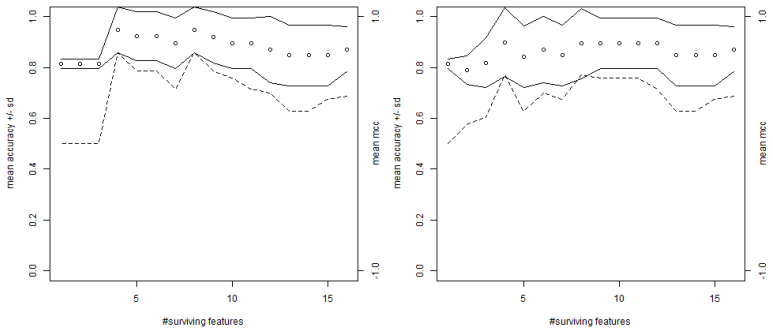

Figure 6.15: Train performance of SVM with radial kernel on `out_int_3`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.843	0.408	0.636	0.0833	`plr_gender`, `fh_mov_spd`
Random Forest on bootstraps	0.871	0.373	0.727	0.241	`fh_mov_spd`, `bot_snd_rad`, `plr_gender`, `fh_mov_acc`
Single linear SVM	0.948	0.714	0.818	0.516	`plr_gender`, `fh_mov_acc`, `fh_mov_spd`, `plr_stealth_exp`
Linear SVM on bootstraps	0.948	0.714	0.727	0.241	`plr_gender`, `bot_snd_rad`, `fh_mov_spd`, `plr_pc_exp`, `fh_mov_acc`, `plr_stealth_exp`

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Single radial SVM	0.948	0.714	0.818	0.516	`plr_stealth_exp`, `fh_mov_acc`, `fh_mov_spd`, `plr_gender`
Radial SVM on bootstraps	0.9	0.543	0.727	0.241	`fh_mov_spd`, `plr_gender`, `bot_snd_rad`, `plr_stealth_exp`

This is the problem that provided the best results, in term of the classifiers' performance on both train and test sets. Random Forests performs slightly worse than the other methods and SVMs with linear kernel trained with bootstraps require much more features than other classifiers.
Test results identify both SVMs with linear and radial kernel as having the highest performance in both accuracy and MCC.
The most important features for predicting the highest level of interest are: the players' gender, their experience with stealth games and the fireflies hunter acceleration and movement speed.
Somewhat surprisingly for the game designer, there is a significant positive correlation between maximum interest and female players (coefficient: 0.41 , p-value of Pearson's test: 0.003) and with the fireflies hunters' movement speed (coefficient: 0.37 , p-value of Pearson's test: 0.009).
For most of the female players, Heartbot - Escape was a novel and fun experience; as for the fireflies movement speed, it appeared throughout the beta test that slower enemies dictated a slower pace of the game, forcing the players to wait the enemies' moves for long periods of time.

6.6 Session Duration

In the previous chapter, three problems were formulated regarding the session duration: based on the quartiles (Q_1, Q_2, Q_3) of the distribution, the first problem decides if the duration will be less or greater than the first quartile Q_1, the second will decide against the median Q_2 and the third will use the third quartile Q_3.

6.6.1 Lower Quartile

The train set was comprised of 11 sessions that were shorter than the first quartile and 27 longer; in the test set, only 2 were shorter and 9 were longer. As in the case of the first problem about the interest, the imbalance in the

test set might determine unpredictability of the MCC values.

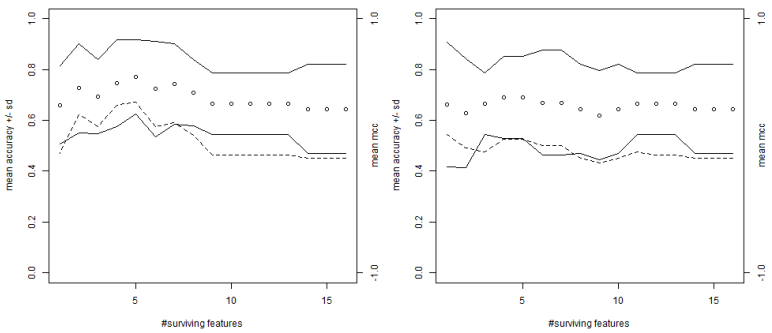

Figure 6.16: Train performance of Random Forest on `time_1`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

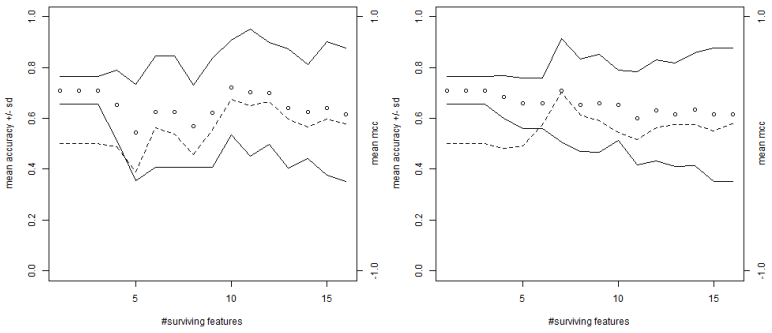

Figure 6.17: Train performance of SVM with linear kernel on `time_1`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

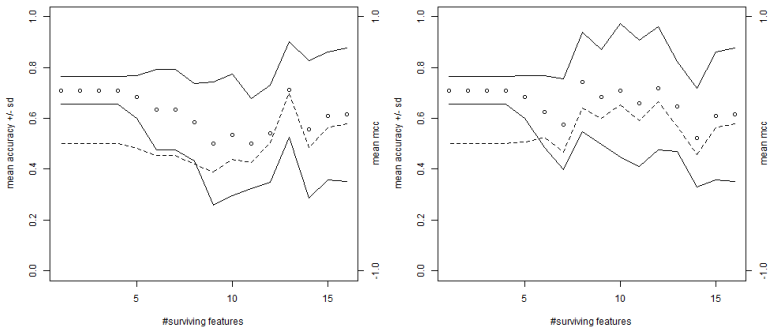

Figure 6.18: Train performance of SVM with radial kernel on `time_1`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.772	0.343	0.818	0.389	`plr_stealth_exp`, `sp_mov_acc`, `plr_port_con_exp`, `fh_mov_spd`, `plr_home_con_exp`
Random Forest on bootstraps	0.69	0.05	0.364	-0.43	`sp_mov_acc`, `fh_mov_spd`, `fh_mov_acc`, `bot_mov_spd`
Single linear SVM	0.722	0.352	0.545	-0.289	`plr_pc_exp`, `bot_mov_spd`, `fh_view_cone`, `plr_home_con_exp`, `bot_snd_rad`, `plr_hb_runs`, `fh_mov_spd`, `plr_stealth_exp`, `plr_mobile_exp`, `plr_port_con_exp`
Linear SVM on bootstraps	0.71	0.407	0.545	0.043	`plr_home_con_exp`, `plr_pc_exp`, `bot_mov_spd`, `plr_hb_runs`, `bot_snd_rad`, `fh_mov_acc`, `fh_view_cone`

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Single radial SVM	0.713	0.402	0.727	-0.149	plr_stealth_exp, plr_port_con_exp, bot_dsh_spd, plr_pc_exp, fh_mov_acc, fh_mov_spd, bot_snd_rad, plr_home_con_exp, bot_mov_spd, sp_mov_spd, plr_mobile_exp, fh_view_cone, plr_hb_runs
Radial SVM on bootstraps	0.743	0.283	0.727	0.241	plr_home_con_exp, plr_pc_exp, bot_snd_rad, bot_mov_spd, fh_mov_acc, plr_hb_runs, fh_mov_spd, sp_mov_spd

On the train set, repeated Random Forest achieves much better results than Random Forest trained on bootstraps; SVMs, on the contrary, appear to require much more features in their single version to achieve results comparable with the versions trained on the bootstraps.
As for the results on the test set, repeated Random Forest is the top achiever, consistently outperforming the other methods in both accuracy and MCC. SVM with radial kernel trained on bootstraps and single SVM with linear kernel follow, and SVM wih linear kernel trained on bootstraps is only barely better than a majority classifier. The other two methods show strong disagreement between prediction and true labels.
Regarding the optimal features, every method seemed to select a different subset. The top performer uses the players' stealth experience, the spiders' acceleration, the experience with portable consoles, the fireflies hunters' movement speed and the players' experience with home consoles. This last feature appears also at the first place in SVMs trained with bootstraps and, with lower rankings, in the subsets selected by single application of SVMs.
An interesting fact is that repeated Random Forest achieved the best results using mostly features related to the players' gaming experience, while the SVMs selected mostly game parameters. The relevant features also show

little overlap with those selected in the difficulty level problem, with only the fireflies hunters' speed appearing in both.
Examining correlation of the above features with the output, no statistically significant relationship were found, though as expected, players experienced with Heartbot were associated with shorter sessions.

6.6.2 Median

In the train set 18 sessions were shorter than the median and 20 were longer; in the test set, in those categories there were respectively 7 and 4 sessions.

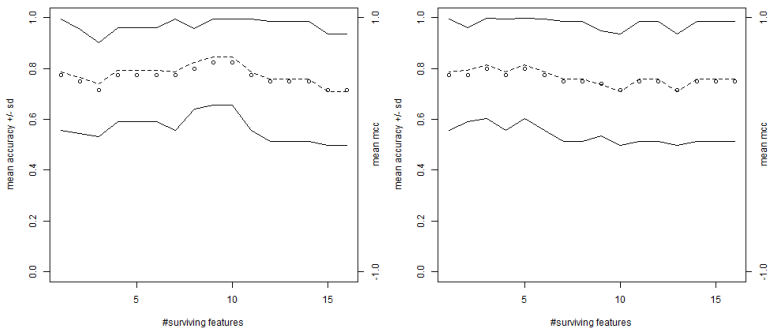

Figure 6.19: Train performance of Random Forest on `time_2`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

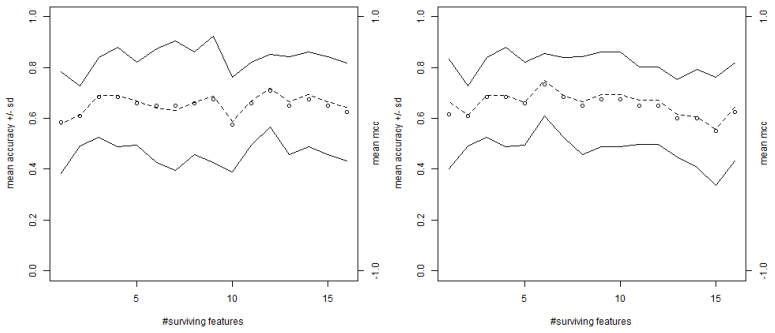

Figure 6.20: Train performance of SVM with linear kernel on `time_2`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

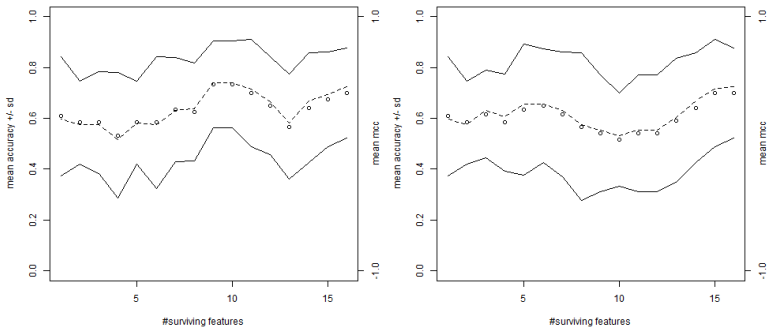

Figure 6.21: Train performance of SVM with radial kernel on `time_2`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.775	0.589	0.273	-0.449	`sp_mov_acc`, `fh_view_cone`, `plr_hb_runs`, `sp_mov_spd`
Random Forest on bootstraps	0.8	0.631	0.273	-0.449	`sp_mov_acc`, `fh_view_cone`, `sp_mov_spd`
Single linear SVM	0.683	0.381	0.364	-0.311	`sp_mov_acc`, `fh_view_cone`, `plr_hb_runs`
Linear SVM on bootstraps	0.733	0.496	0.545	0.069	`fh_view_cone`, `sp_mov_acc`, `plr_hb_runs`, `plr_age`, `sp_mov_spd`, `plr_gender`
Single radial SVM	0.733	0.481	0.727	0.571	`sp_mov_acc`, `fh_mov_spd`, `plr_home_con_exp`, `plr_mobile_exp`, `plr_age`, `plr_port_con_exp`, `fh_view_cone`, `bot_dsh_spd`, `plr_hb_runs`
Radial SVM on bootstraps	0.633	0.315	0.545	-0.0386	`sp_mov_acc`, `fh_mov_spd`, `fh_view_cone`, `plr_home_con_exp`, `plr_age`

As in the first problem, the train set is very balanced in the distribution of output labels and the MCC follows the classification accuracy values.
All classifiers obtain good results in the training phase, but only a few translate well on the test: both Random Forests classifiers show low accuracy and MCC scores, together with single SVM with linear kernel. The single SVM with radial kernel managed to obtain good performance with a large set of selected features, but the remaining classifiers were almost equivalent to a random choice.
This is surprising, as most of the top ranked features of every method are very similar: the spiders' movement acceleration shows strong importance, together with the fireflies hunters' view cone angle. The players' previous experience with Heartbot is also chosen frequently.
Examining their correlation with the output, the spiders' acceleration shows statistically significant negative correlation with the sessions duration (coefficient: -0.34 , p-value of Pearson's test: 0.017); the view cone angle and the experience with Heartbot show respectively positive and negative correlation, but the p-values are slightly above the statistical significance threshold (0.07 for view cone angle, 0.06 for the previous experience).
The results on the previous experience with Heartbot and the view cone angle were expected, but the strong negative correlation of the spiders' acceleration was surprising and will require further investigation.

6.6.3 Upper Quartile

The train set contains 28 sessions shorter than the third quartile and 10 that lasted longer; in the test set the count was of 9 and 2 respectively. The same discussion regarding the imbalance in the test set and the MCC evaluation applies here.

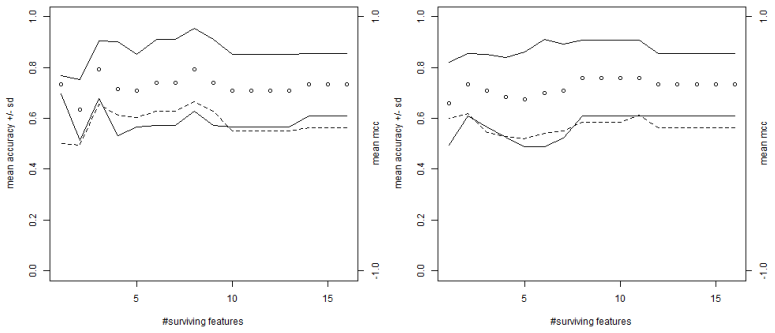

Figure 6.22: Train performance of Random Forest on `time_3`; on the left, RFE was repeated 100 times; on the right, RFE was repeated on 100 bootstraps of the train set

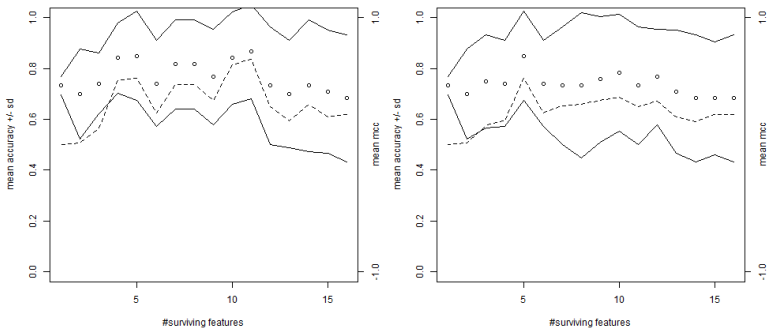

Figure 6.23: Train performance of SVM with linear kernel on `time_3`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

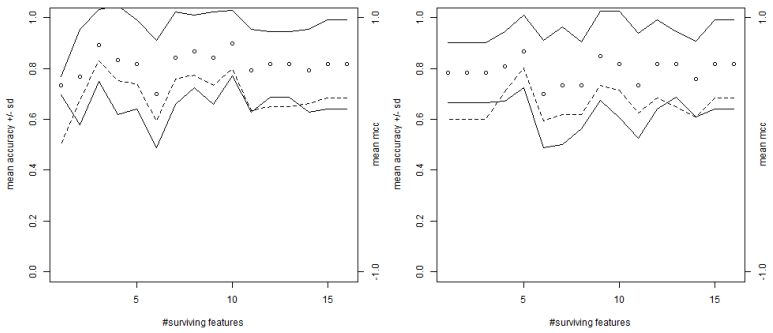

Figure 6.24: Train performance of SVM with radial kernel on `time_3`; on the left, RFE was applied once; on the right, RFE was repeated on 100 bootstraps of the train set

	Train		Test		
Classifier	Acc.	MCC	Acc.	MCC	Selected Features
Repeated Random Forest	0.792	0.315	0.818	0	`plr_port_con_exp`, `bot_dsh_spd`, `plr_pc_exp`
Random Forest on bootstraps	0.733	0.24	0.818	0	`bot_dsh_spd`, `sp_mov_acc`
Single linear SVM	0.842	0.508	0.727	0.241	`fh_view_cone`, `plr_port_con_exp`, `plr_pc_exp`, `bot_snd_rad`
Linear SVM on bootstraps	0.85	0.524	0.818	0.389	`plr_port_con_exp`, `fh_view_cone`, `plr_home_con_exp`, `plr_pc_exp`, `bot_snd_rad`
Single radial SVM	0.892	0.658	0.727	0.241	`plr_port_con_exp`, `plr_pc_exp`, `fh_view_cone`
Radial SVM on bootstraps	0.867	0.608	0.727	-0.149	`fh_view_cone`, `plr_port_con_exp`, `sp_mov_spd`, `plr_pc_exp`, `sp_mov_acc`

In the training phase, Random Forest show worse results than the other methods; the top performances were achieved by the SVMs with radial kernel and SVMs with linear kernel follow closely.
On the test set, SVMs with linear kernel managed to translate better their train results; Random Forest methods instead behave no better than a majority classifier.
Three features appear as the most relevant to the problem: the fireflies hunters' view cone angle and the players' experience with portable console and PC games; these features are selected by all the well performing classifiers.
The players' experience with the two gaming platforms correlates negatively, almost significantly, with the game sessions being longer (p-values of Pearson's test respectively 0.06 and 0.09); this is of course expected. In a similar way, fireflies hunters' view cone angle shows a positive correlation with the output that is statistically significant (coefficient: 0.28 , p-value of Pearson's test: 0.048).
This can be explained with the conjecture that inexperienced player do not

correctly interpret the enemies' movement patterns and try to move dangerously near them; even though the view cone angle wasn't a strong predictor for the difficulty, it makes the game unforgiving without incrementing the perceived difficulty.
Surprisingly, none of the classifiers included the players' stealth experience in the selected features, as the game designer was expecting; examining its correlation with longer sessions showed however an almost significant negative correlation (coefficient: -0.27 , p-value of Pearson's test: 0.06).

6.7 Discussion

Results from the analysis are heavily influenced by the small size of the data set and its imbalance; separation of the data set in train and test sets contributed to the harsh training conditions for the classifiers.
In almost all problems, classifiers showed difficulty in translating good performance in the training phase to good results on the test set; this might reveal that overfitting of the small train set took place.
The performance of the best classifiers on the respective problems, however, were good in the sense that they were able to significantly outperform a majority classifier, with MCC values on the test set mostly above 0 and sufficient accuracy. Feature selection also proved effective in removing unnecessary variables and increasing the classification strength.
No single method proved to be the best in every situation: repeated Random Forest showed strongly on the first, second, third and sixth problem; SVMs with radial kernel, applied a single time, were the top achiever on the seventh problem and together with SVM with linear kernel, on the fifth. SVMs with linear kernel, applied once, yield the best results for the last problem.
The comparison also showed that training performing feature selection with classifiers trained on bootstraps did not always increase the robustness of the results: in particular, Random Forest was most prone to decrease its performance, as its internal bagging mechanism is essentially a repetition of the bootstrapping process.
The different problems were shown to have varying intrinsic difficulty: discerning the highest level of interest was the problem in which the classifiers obtained the best results, while predicting the controls accessibility was quite difficult and classifiers behaved at most slightly better than a majority classifier on the test set.
The worst results were obtained in discerning a positive or negative evaluation of the interest: all the classifiers obtained good results on the train set, but on the test set, no one of them managed to behave better than a random guess.
The adoption of the MCC as the main metric of quality proved effective in selecting the best classifiers and feature subsets, as it revealed instances of

majority classification and gave hints that overfitting of the train data was taking place.
The subsets of selected features were, for the most part, expected by the game designer; in some cases, they exposed surprising relationships with the outcomes, such as showing that female players were those more interested in the game.

Chapter 7

Conclusions and Future Work

This work presented the integration of Machine Learning in the beta test of Heartbot - Escape, a commercial video game, adopting feature selection to identify the most important game parameters and player characteristics in predicting players' evaluation of the game.
Random Forest and Support Vector Machines, with two different training methods, were adopted as the underlying classifiers using stratified cross validation on a train set and the resulting optimal feature sets were evaluated on a separate test set.
Results of the different classifiers and training methods were discussed and compared.
As it was shown, variation of classifiers and training methods had very different performance on different problems and no silver bullet was found. Random Forests proved to be effective on half of the problems, but in some instances Support Vector Machines managed to achieve better performances. In many cases it was difficult to translate a method's good results in the training phase to comparable results on the test set; this might be due to the scarcity of data collected and their heavy imbalance in the distribution of the output classes.
Since the performances of the best classifiers for each problem were very different between different instances, it follows that some problems were intrinsically more difficult to describe with the model used.
The proposed approach, however, proved to be effective in selecting meaningful feature sets, as in the majority of the problems the classification performance overcome a trivial majority classifier and in some cases, surprising relationships between input and output were revealed.
The best results have been obtained in discerning game sessions evaluated as being "Very Interesting", which was one of the most important requirements from the game designer's point of view.
The main direction of future work is toward the retrieval of the optimal parameters values in order to obtain the desired output in the evaluation by the

players. Possible approaches include the inspection of the numerical model of the best classifiers and the use of Bayesian Networks after appropriate discretization of the data.
Additionally, it would be interesting to apply the methodology to more complex games, with a larger set of input features possibly including procedural generation of the environment.

Bibliography

[1] G. N. Yannakakis and J. Hallam, "Real-time game adaptation for optimizing player satisfaction.," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 1, no. 2, pp. 121–133, 2009.

[2] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals.* The MIT Press, 2003.

[3] C. Crawford, *The Art of Computer Game Design.* Berkeley, CA, USA: Osborne/McGraw-Hill, 1984.

[4] R. Rouse, *Game Design Theory and Practice.* Plano, TX, USA: Wordware Publishing Inc., 2nd ed., 2000.

[5] T. W. Malone, "Toward a theory of intrinsically motivating instruction," *Cognitive Science*, vol. 5, no. 4, pp. 333–369, 1981.

[6] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience.* New York, NY: Harper Perennial, March 1991.

[7] P. Sweetser and P. Wyeth, "Gameflow: A model for evaluating player enjoyment in games," *Comput. Entertain.*, vol. 3, pp. 3–3, July 2005.

[8] S. Björk, S. Lundgren, and J. Holopainen, "Game design patterns," in *in Level Up: Digital Games Research Conference 2003*, pp. 4–6, 2003.

[9] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A formal approach to game design and game research," in *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, pp. 1–5, July 2004.

[10] R. Hunicke and V. Chapman, "AI for Dynamic Difficulty Adjustment in Games," 2004.

[11] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, (New York, NY, USA), pp. 429–433, ACM, 2005.

[12] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Challenge-sensitive action selection: An application to game balancing," in *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '05, (Washington, DC, USA), pp. 194–200, IEEE Computer Society, 2005.

[13] G. Andrade, G. Ramalho, A. S. Gomes, and V. Corruble, "Dynamic game balancing: An evaluation of user satisfaction," *AIIDE*, pp. 3–8, 2006.

[14] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," 2006.

[15] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pp. 252–259, IEEE, 2007.

[16] G. N. Yannakakis and J. Hallam, "Interactive opponents generate interesting games," *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pp. 240–247, 2004.

[17] G. Yannakakis and J. Hallam, "A generic approach for obtaining higher entertainment in predator/prey computer games," *Journal of Game Development*, vol. 1, no. 3, pp. 23–50, 2005.

[18] G. N. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," *Appl. Artif. Intell.*, vol. 21, pp. 933–971, Nov. 2007.

[19] G. N. Yannakakis and J. Hallam, "Modeling and augmenting game entertainment through challenge and curiosity.," *International Journal on Artificial Intelligence Tools*, vol. 16, no. 6, pp. 981–999, 2007.

[20] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation.," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.

[21] R. Dromey, "From requirements to design: formalizing the key steps," in *Software Engineering and Formal Methods, 2003.Proceedings. First International Conference on*, pp. 2–11, Sept 2003.

[22] R. Kohavi and R. Quinlan, "Decision tree discovery," in *IN HANDBOOK OF DATA MINING AND KNOWLEDGE DISCOVERY*, pp. 267–276, University Press, 1999.

[23] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15 – 17, 1976.

[24] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series, Taylor & Francis, 1984.

[25] C. Gini, "Variabilitá e mutabilitá.," *Memorie di metodologia statistica*, 1912.

[26] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[27] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[28] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artif. Intell.*, vol. 97, pp. 273–324, Dec. 1997.

[29] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Mach. Learn.*, vol. 46, pp. 389–422, Mar. 2002.

[30] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, "Variable selection using random forests," *Pattern Recogn. Lett.*, vol. 31, pp. 2225–2236, Oct. 2010.

[31] C. Furlanello, M. Serafini, S. Merler, and G. Jurman, "Entropy-based gene ranking without selection bias for the predictive classification of microarray data.," *BMC Bioinformatics*, vol. 4, p. 54, 2003.