Artem Vasenin

Predicting statistical results in competitive computer games

Computer Science Tripos – Part II

Clare College

May 18, 2017

Proforma

Name Artem Vasenin

College Clare College

Title Predicting statistical results in competitive computer games.

Examination Computer Science Tripos – Part II, June 2017

Word Count 9706[1]

Project Originator Artem Vasenin (av429)

Project Supervisor Yingzhen Li (yl494)

Original Project Aim

The aim of this project was to develop an algorithm which could predict whether certain events would happen in multiplayer computer games. In most modern multiplayer games, players are provided with a summary after each match describing their performance in that match and outlining key choices they have made. The project should be able to predict the values present in such summary. The mean squared error of such predictions should be less than half of variance of the variable.

Summary of Work Completed

The project was done in two stages. First, a prototype was created using ad-hoc use of existing ranking algorithms. The prototype was analysed and a list of problems was compiled. To solve these problems a graphical model, similar to a ranking algorithm, was created with conditional probabilities approximated using neural networks. The final system achieved the project aim on one variable and had good results on another three.

Special Difficulties

There was a mistake in the project proposal success criteria, which is explained in section 2.2.2.

Declaration of Originality

I, Artem Vasenin of Clare College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date: May 18, 2017

[1] Calculated using `pdftotext Dissertation.pdf -f 9 -l 42 - | wc -w`.

Acknowledgements

I would like to thank my supervisor, Yingzhen Li, for the weekly meetings throughout the project's duration and helping me understand the complex methods used in modern machine learning approaches.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Multiplayer computer games are becoming very popular, more than a 100 million people play League of Legends every month [4]. A game's success often rests on how enjoyable it is. Current method of improving player experience is to make sure that all players have equal chance of winning a match. For that purpose their skill has to be tracked and teams have to be arranged such that they are of equal strength.

In many popular games several roles have to be filled on each team for optimal performance. Current algorithms, such as TrueSkill [5], only track player's overall skill and do not consider what roles the player prefers and how good they are at each one. This often leads to teams being composed of players all wanting to play the same role, which either leads to team under-performing or some players not enjoying the game as much as they could. Moreover, I believe that the events that happen in a match are more important to many player's experience than the actual outcome.

To be able to match players better, a system has to be built that will take into account how the game is played and what strategies exist in it. Creating such a system using classical techniques would require deep knowledge of workings of the game. Unfortunately, I do not have such knowledge of most games and obtaining such knowledge although might be fun, will take too long and will be inefficient. Furthermore, most such games change their rules every few months to keep players interested, therefore performance of hard coded system would decrease as the time goes on. As a result I decided to use machine learning techniques which would help me create a system which can adapt to different games by itself and also update its judgement as game changes.

1.2 Related Work

This project draws heavily from various ranking algorithms that already exist, most of them are based on Elo system [3]. Before starting the project, I have looked into papers and articles about predicting events in sports and computer games. I have found out that nothing similar has been done before. Most closely related algorithms (such as [5] and [7]) were made to only predict the outcome of a match, not any related results.

Outside of academia, other algorithms, such as [2], were made which used machine learning to improve their performance, but again only focusing on the outcome of a match. The game I am considering, "Dota 2", was also studied a few times, such as [1],

but again only the outcome of the match was considered.

1.3 Project Aim

The project aim was to produce a system which could predict the outcome of a match between different players, given their past match results. As the input the system should take match history of the players being considered and team composition (which player belongs to which team). As the output the system should produce expected values for different variables in match summary. The mean squared error of produced predictions should be less than one half of variance for that variable.

Chapter 2

Preparation

2.1 Proposal Refinement

I had two drafts of my project proposal, this being my first major project I had little experience in preparing a plan. After receiving comments from my overseers regarding the first draft of the proposal, I have incorporated several changes in the project proposal, including:

- Add a time-line, deadlines and deliverables to the project plan, to be able to track the progress of the project accurately.
- Limited and listed all of the potential variables that will be considered, to keep the project goals concise.
- Add criteria to library selection to streamline the process.

Overall this allowed me to stay on target throughout the project and control "feature creep"[1].

2.2 Requirement Analysis

2.2.1 Type of variables to predict

Values included in the match summary can be roughly separated into two types: class based and regression based. For example in many games players have the ability to buy items. These items are usually represented as numbers in match summary, but nearby values usually do not have much in common. Therefore, if such variables have to be predicted, a classification method should be used is a good approach.

On the other hand something like players score is a value which increases progressively, therefore nearby values have similar significance. In such cases regression techniques would work better.

2.2.2 Requirements

The projects aimed to produce a system that would be able to predict match summary data from players' history.

[1] https://en.wikipedia.org/wiki/Feature_creep

Figure 2.1: Screenshot of a match summary in the game 'Dota 2'.

- The system must be able to work with data from different games, as long as it is formatted properly.
- The predictions should be based on skills of all players in the match, not just the player for which predictions are being made.
- The system should be able to maintain a belief state about a player skill through time and make time-sensitive predictions, not the same prediction for all matches.
- The system should be able to make predictions in under a minute after being trained, on my machine[2].
- The mean squared error of system's predictions should be less than one half of player's variance for that variable.

Regarding the last point: in the project proposal my success criteria contained a mistake and had 'standard deviation' instead of 'variance', which made the success criteria almost impossible to achieve. If that interpretation is used the systems deviation is squared, while original deviation is not. That means that achieving this requirement gets more difficult the larger the scale of the variable is. This also makes a big difference once the variables are normalised, since squaring values less than one decreases it and makes achieving this result easier. In my evaluation I will analyse the results as though the success criteria was stated as above.

2.3 Starting Point

At the beginning of the project I had:

- Basic knowledge of artificial intelligence from *Artificial Intelligence I* Part IB course.

[2]Specifications of machine's hardware are given in the project proposal

- Programming experience in Python (acquired by working on personal projects) and Java (acquired by completing courses in first two years of the tripos).

- Knowledge of probability rules and Bayes' theorem from A-Level maths and math courses in the first two years.

During the course of the project I had to gain following qualities:

- Understanding of Bayes' inference and its use in ranking.

- Understanding of various machine learning techniques.

- Familiarity with the chosen ML library (Tensorflow).

- Knowledge of LaTeX and related packages (such as Tikz for graphs[3]) to write-up this dissertation.

2.4 Theoretical Background

2.4.1 Probability background

There are a few probability rules, which are used in this model.

Sum rule Sum/Integral rule allows us to remove unneeded variables from our probability estimates:

$$P(A) = \int_B P(A, B)\ dB = \sum_B P(A, B)$$

Product/Chain rule Product rule allows us to calculate joint probabilities only using conditional probabilities and priors:

$$P(A, B) = P(A \mid B)P(B)$$

Bayes' rule I will be using Bayesian networks and inference in my system, which is based on Bayes' theorem:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

- $\boldsymbol{P(A)}$ is the marginal probability of observing A, called *prior* of A.

- $\boldsymbol{P(A \mid B)}$ is the probability of observing A given B, called *posterior* of A.

- $\boldsymbol{P(B \mid A)}$ is the conditional probability of B given A, called *likelihood* of A.

[3] All figures, except 2.1 and 2.3, have been generated using Tikz or pyplot python library.

2.4.2 Machine Learning

Machine learning allows computers to make predictions on data by learning from past examples and without being explicitly programmed. The aim of my project was to create an algorithm which could work with any game, therefore machine learning was the perfect set of techniques to apply.

Since I decided to use machine learning in my project I had to get some background in the area. Unfortunately, computer science course is in Lent term, which is too late, therefore I attended a similar course in Engineering department in Michaelmas term, before beginning the work on the theory.

After I have finished my research, I have decided to try the following machine learning techniques in my projeject:

- Naive bayes (NB)
- Linear Regression (LinReg)
- Logistic Regression (LogReg)
- Polynomial Regression (PolReg)
- Ridge Regression (RidReg)
- Neural Networks (NN)
- Support Vector Machines (SVM)
- Gaussian Processes (GP)
- Random Forests (RF)

2.4.3 Over and under fitting

When training an algorithm a decision has to be made when to stop the training, this is difficult decision and is usually made using some kind of heuristic. If the training is stopped too early it leads to under-fitting, if it is stopped too late, it leads to over-fitting, in both cases the algorithm performs below optimum.

Under-fitting An under-fit algorithm cannot model the training data well and is not suitable. It is often easy to detect using a good loss metric. The basic solution is to use more complex model or train longer.

Over-fitting An over-fit algorithm models the training data too well, the algorithm learns the noise in the training data. This leads to training loss decreasing, while validation loss remains the same or even starts increasing, an example can be seen in figure 2.2. Noise is random, therefore it will be different in the real/test data. By modelling the noise in the training data, the real world-performance is decreased. Over-fitting is a bigger problem, since it is much more difficult to detect, therefore different methods are usually used to prevent it. I outlined some of the methods I used in subsection 4.2.1.

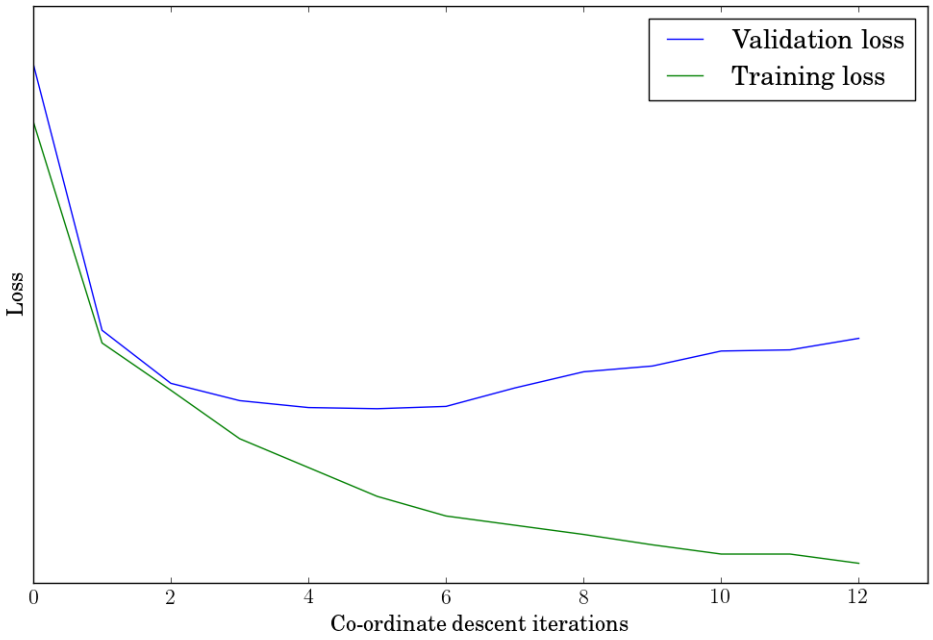


Figure 2.2: Overfitting example.

In this case the optimisation should be stopped at the fifth iteration, after which training loss keeps decreasing, but validation loss starts increasing.

2.5 Software Engineering Techniques

2.5.1 Workflow

I have decided to use an agile approach of working on this project, seen in figure 2.3, since I was not sure exactly how the algorithm should work at the start of the project. The project would be done in two phases:

1. Primary research would be done during the second half of Michaelmas term. A prototype would be developed during the Christmas break.

2. At the beginning of Lent term the prototype would be evaluated. In the first two weeks of February the theory would be improved using insights gained during the development of the prototype. In the second two weeks of February the prototype would be refactored to comply with changes in the theory. At the beginning of March the revised implementation would be evaluated.

This approach allowed me to incorporate the knowledge I gained during the evaluation of a prototype into the final version of the algorithm.

2.5.2 Version Control

Git was used for version control. Frequent commits were made after every self-contained change. This allowed me to roll-back to a previous version of the project when a mistake was made. It also allowed me to compare and evaluate different versions

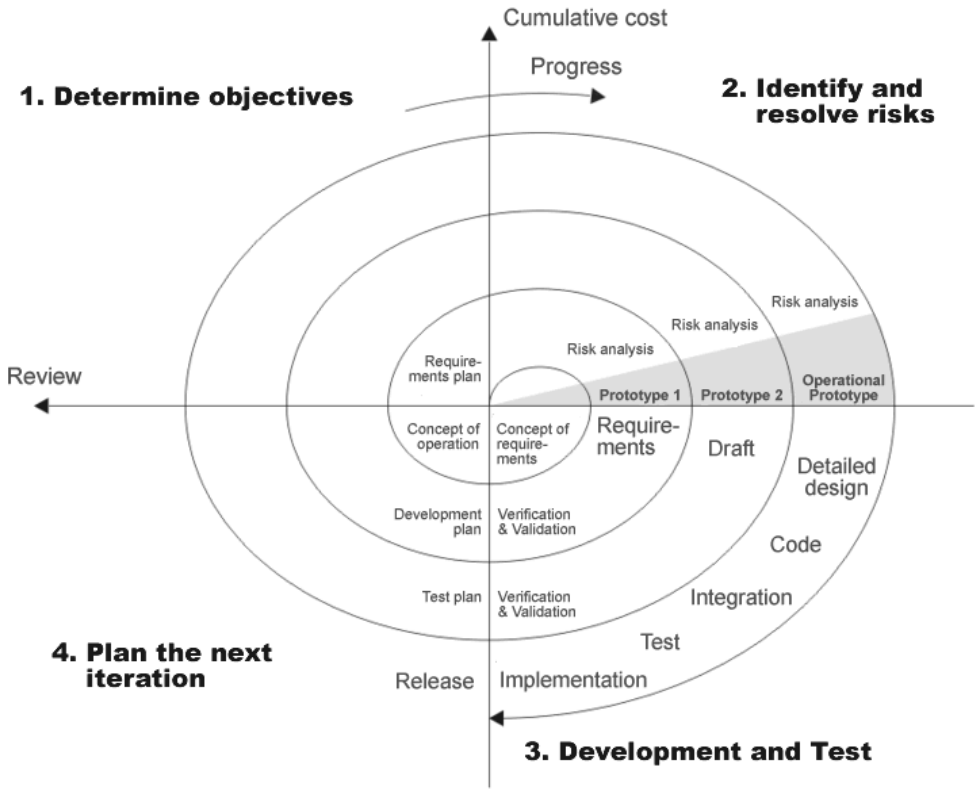


Figure 2.3: Agile approach using spiral model.
Image from https://en.wikipedia.org/wiki/Spiral_model.

of a specific component. The repository was hosted on GitHub[4] which allowed me to work on the project from multiple machines.

2.5.3 Backup

The code was continuously synchronised to Dropbox[5]. Weekly checkpoints were saved to an external hard drive.

2.6 Tools used

2.6.1 Programming Language and ML Libraries

Machine learning is one of the key elements of this project, therefore I needed a library that would implement key techniques used in ML for me, so that I don't have to write them myself. I have compared a number of libraries (such as Tensorflow[6] and Torch[7]). I was primarily interested in how much functionality they provide, their performance and how easy it would be to use them.

[4] https://github.com
[5] https://dropbox.com
[6] https://tensorflow.org
[7] http://torch.ch

I have compared their functionality by completing standard machine learning task, such as creating a neural network to classify images in the MNIST[8] dataset. Performance was compared by timing how long it would take to achieve 99% accuracy, in most cases it came down whether GPU acceleration was supported. Since I would have to learn how to use the chosen library in detail, I also paid attention to the amount of support material available. I took note of quality of documentation and also compared number of questions and answers on StackOverflow.

In the end I have arrived at the conclusion that they all provided required functionality and were similarly easy to use, but API for most of them were written in different languages. I did not want to learn a new language in addition to learning a new library, therefore I decided to use TensorFlow, API for which was written in Python, a language I was most comfortable writing code in. TensorFlow is a low-level library, in which each layer of neural network has to be constructed explicitly, I did not need that level of customisation, therefore I used Keras[9] to help me build the neural networks.

Additionally Python has a higher level machine learning library called scikit-learn[10]. It provides easy way to quickly implement many machine learning methods, this library was used to evaluate prototype evaluation. To a lesser extent Python was also chosen since it has a package manager[11], which makes it much easier to install additional packages.

2.6.2 IDE

Pycharm was chosen as the integrated development environment (IDE) for the project. Pycharm has all of the core features of an IDE, such as syntax highlighting, autocompletion, code execution, etc. The use of an IDE streamlined the process of development and allowed me to focus on improving the algorithm rather than worrying about language syntax or library functions signatures.

[8] http://yann.lecun.com/exdb/mnist
[9] https://keras.io
[10] http://scikit-learn.org
[11] https://pypi.python.org/pypi/pip

Chapter 3

Implementation

This chapter discusses design and implementation of the prediction system produced. The system was made in two stages: first a prototype was made, tested and analysed. Based on the result of the analysis a better system was designed and implemented. The main system consists of two models: prediction model and skill update model.

Prediction model is used to make predictions based on the estimated skills of the players and to calculate player performance based on match results. Skill update model is used to estimate the skills of the players at a particular time using past performances.

3.1 Installation of required libraries

Python has a package index (PyPI) which is used to distribute most available packages. The PyPI provides a tool called *pip* which can manage package/library installation, it can be installed on Ubuntu 16.04 using:

```
$ sudo apt install python3-pip
```

After pip is installed all required packages can be installed using:

```
$ sudo pip install <package name>
```

Pip will manage all of the required dependencies and put the package files in the correct directories.

3.2 Data Collection and Pre-processing

3.2.1 Collecting the data

To train and evaluate my algorithm I had to find a complex dataset which would have different types of variables for me to predict. I decided to use a game called "Dota 2" for several reasons:

- Summary of each professional match is publicly available and easily accessible through the provided API.
- "Dota 2" is one of the leading E-Sports with hundreds of professional matches each year, which provided me with plenty of data points.

- The game requires a lot of skill with very little amount of luck involved, which makes it very suitable to prediction.

- The game is team-based and requires a lot of co-operation, which makes it more difficult to predict using classical rating algorithms.

3.2.2 Pre-processing

Cleaning the data The API which provides the summaries sometimes returns incomplete data with some of the values, such as the outcome of the match, missing. I had to find and remove all data points which did not contain complete set of results. The amount of such summaries is less than one percent of total data, therefore its removal did not negatively affect training and evaluation.

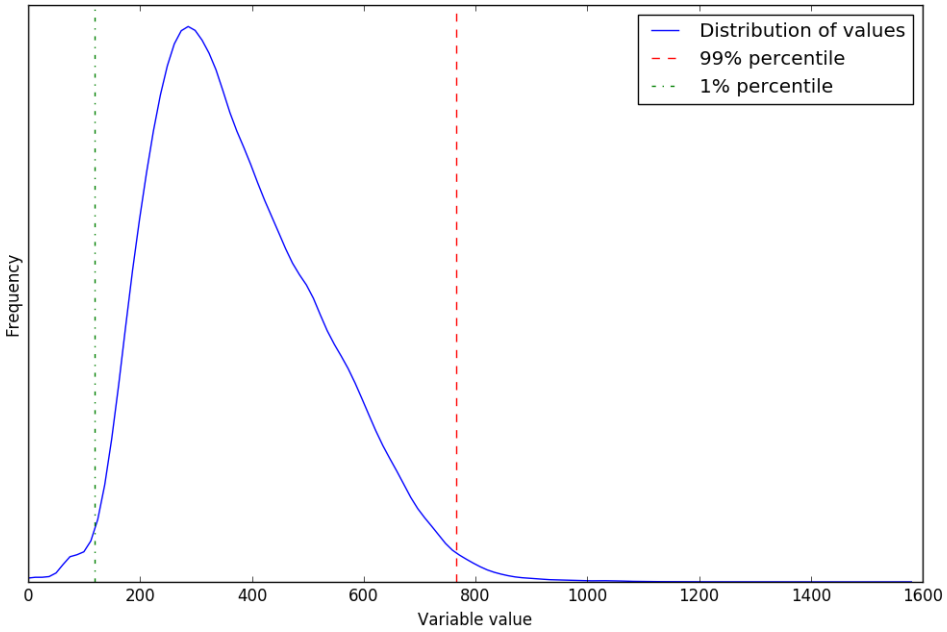


Figure 3.1: The distribution of values one of the variables.

Outliers Most variables contain some outliers which are extreme cases. In some cases the maximum value is almost twice as high as the 99% percentile, as can be seen in figure 3.1. To simplify the learning for the algorithm I decided to clip all the value for each variable between the 1st and 99th percentile.

Standardisation Some of the variables in the summary occur over much bigger scales than others. Most machine learning algorithms would not train properly and would prioritise predicting large variables more accurately than smaller ones. I would like to

consider all variables with equal priority, therefore I had to standardise the data. To standardise data a simple transformation should be applied to all data points:

$$x' = \frac{x - \overline{x}}{\sigma}$$

Where $\overline{x}$ is the mean of X and σ is its standard deviation

Normalisation During the analysis of data, outlined in subsection 3.4.1, I decided that the data is better modelled using a gamma distribution. Gamma distribution can only represent positive numbers, therefore I could not use standardisation for this dataset. Instead I used normalisation which also rescales the values, but keeps them all positive, it uses the following transformation:

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)}$$

3.3 First Prototype

Frequently the main problem in machine learning is the generation of correct features to represent the data. In my case I had summaries of more than twenty thousand matches. Most players had a few hundred matches in the dataset. Summary for each match is represented as a dictionary of key-value pairs (in JSON[1] format), overall there are more than 200 values per summary, an example of a summary can be found in Appendix A. This meant that using raw data for input would be impractical, therefore some kind of aggregation had to be created.

3.3.1 Naive approach

At first I tried using means and standard deviations of players results as features. While this approach made predictions which were better than just predicting variable mean, they were not good enough. A variety of machine learning techniques were tried, including: Neural Networks (NN), Ridge Regression (RR) and Naive Bayes (NB), Random Forests (RF). I have tried NN and RF in both classification and regression configuration, some of the results are shown on figure 3.2. All of the methods had similar performance, which indicated that features generated were not sufficient. To improve the features I first tried putting a limit on how many matches were used to calculate player's mean and average, while this improved performance a bit, it remained quite low. Next thing to try was rating players based on their performance in each variable.

3.3.2 Rating of players

The easiest algorithm I found for rating players was TrueSkill, since it offered rating games with multiple players and there was a library[2] for python which provided the functionality. TrueSkill only uses the final ranking of the individual at the end of the match, rather than absolute value, therefore some informations is lost. This also meant that I had to do further processing on the data to convert raw values into rankings. This

[1] http://json.org
[2] httpz://trueskill.org

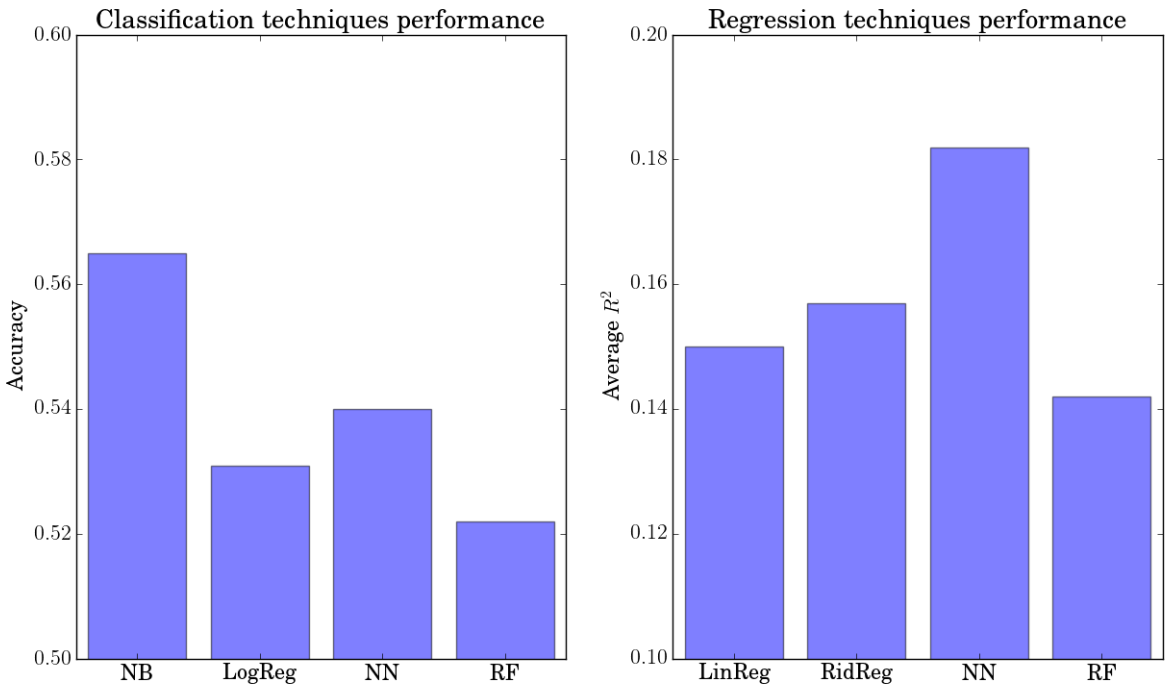


Figure 3.2: Results of a naive approach.

Naive Bayes (NB) had very good results even using naive features, which I could not achieve even using the final system.

was mostly straightforward, one thing to note is that some rankings (such as death count) had to be reversed, since players try to keep those results low rather than make them high.

Such ad-hoc use of TrueSkill meant that one of the assumptions behind the algorithm was not met: the algorithm assumes that all players are competing against each other, but in reality they are separated in two teams. This meant that resulting ratings were not very accurate. Player skills were then used as features for machine learning techniques. This approach generated much better results than simple mean and variance, with best variable achieving R^2 value of 0.43. Although this is much better than previous results, it was below the required value of 0.5.

3.3.3 Analysis of different machine learning techniques

As said above most of the techniques provided similar results. In the end I decided to use neural networks as my main technique for the following reasons:

- Neural networks can do both regression and classification, thereby I will only have to focus on properly learning and understanding one technique.
- One neural network can estimate values for multiple results, thereby reducing the complexity of the system and increasing efficiency.
- Chosen library (Tensorflow) has better support for neural networks than other techniques.

3.3.4 Analysis of the prototype

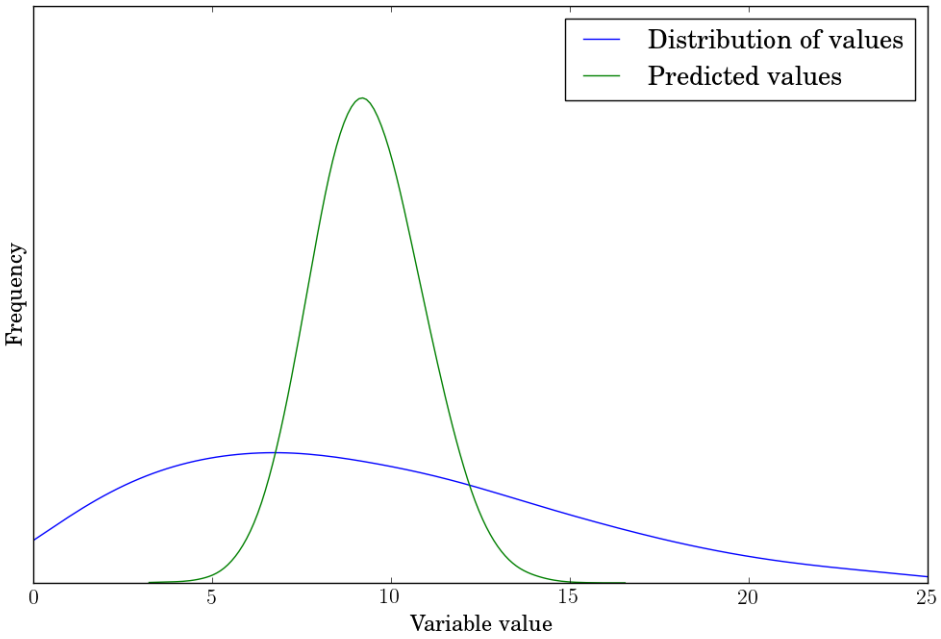


Figure 3.3: Distribution of predicted values vs. actual variable distribution

During analysis of predictions produced by the prototype, I observed that the range of predictions was much smaller than the actual range for most variables. An example of this can be seen on figure 3.3, where predictions roughly range from 5 to 15, while actual variable range is roughly 0 to 30. In addition, most of the machine learning techniques had similar predictive performance. These two facts combined led me to believe that the problem was with quality of features.

3.3.5 Required refactoring and redesign

To create better features I decided to modify the structure of the TrueSkill graphical model to remove the wrong assumptions. This required adding dependency links between player ranks and their teams. Additionally, I decided to use machine learning techniques throughout to calculate player ranks, rather than using hard-coded rules, since I do not know which variables are important.

I also decided to use a Bayesian approach because it reduces the risk of overfitting at the expense of increasing the risk of underfitting. Since underfitting can be solved much easier, by increasing the size of the model for example, it made the training process much easier.

3.4 Final System

3.4.1 Choosing distributions for different variables

When predicting the values for each regression-based variable, rather than predicting the probability for each value independently, it is better to use a distribution to predict

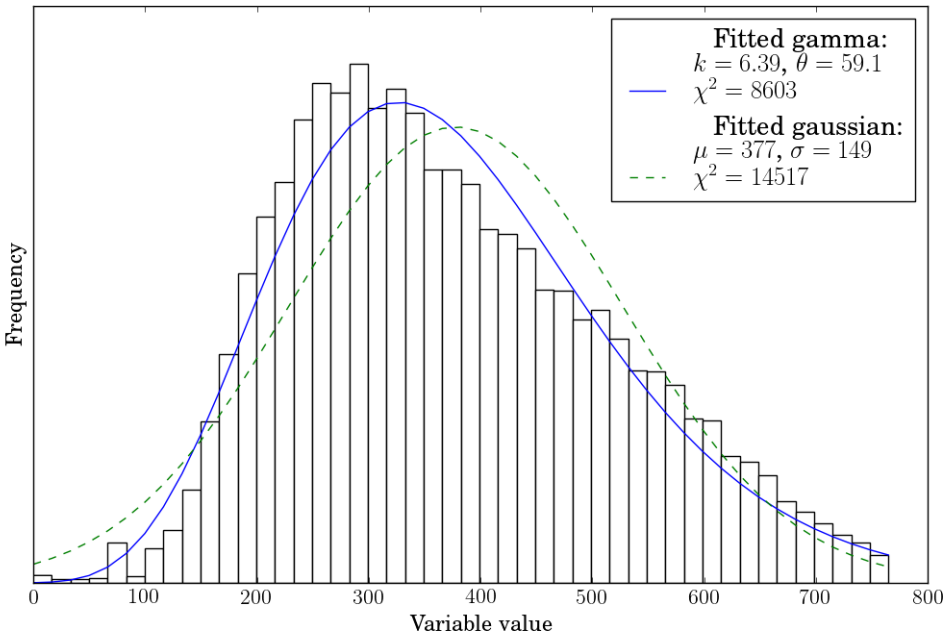


Figure 3.4: Fitted gaussian and gamma distributions on one of the variables. Gamma distribution has a lower χ^2 value, therefore it is a better fit.

the probabilities for all values. To decide which distribution fits each variable best, I used *method of moments* to fit each potential distribution to all observed values of the variable. Then I used *goodness of fit* to decide which of the potential distributions is best.

Method of Moments To fit a distribution its parameters have to be estimated. Parameters for most distributions can be easily expressed in terms of mean and variance of the data. For example for Gamma distribution the equations are:

$$k = \mu^2/\sigma^2 \qquad \theta = \sigma^2/\mu$$

Goodness of fit To estimate how well the distribution fits the data use chi-squared test can be used, which measures the sum of differences between observed and expected outcome frequencies:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where O_i is the observed frequency and E_i is the expected frequency.

3.4.2 Player skills

Every game is different and trying to pick specific set of skills for each one, would not be practical. Therefore, I decided to make an assumption that players skill in a game can be described by n real numbers. Such n will be different for each game and therefore is a hyper parameter in my model. Since the system can never be sure on the exact

skill values for each player, skills of each player are represented as vector of n random variables, which follow a Gaussian distribution.

3.4.3 Neural networks

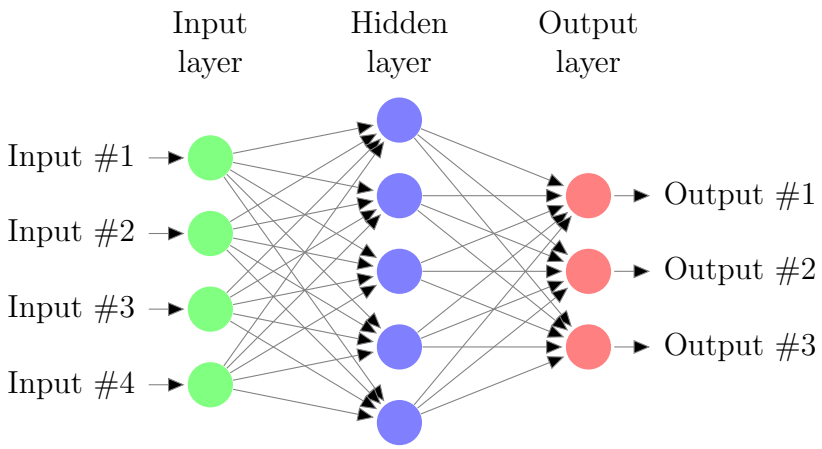


Figure 3.5: An example of fully connected feedforward neural network with one hidden layer.

I decided to use neural networks in my system, therefore I had to learn basic theory behind them and how to construct them. Neural networks are usually made up of three types of layers: input, hidden and output. The size of the input and output layer are determined by the data being processed, deciding on the size of hidden layers is more difficult. If the hidden layers are too small under-fitting can occur, similarly if the hidden layers are too big or there too many of them over-fitting can occur.

Perceptron Each layer of a NN is composed of a number of perceptrons. Perceptron is a simple linear classifier which is can be described by the following equation:

$$y = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

Where y is the output, $\sigma()$ is the activation function, $\boldsymbol{w}$ a vector of weights, $\boldsymbol{x}$ is a vector of inputs and b is the bias.

These can be combined to form a layer of a neural network, where all perceptrons accept the same input. The equation for a layer can be written as:

$$\boldsymbol{y} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

where $\boldsymbol{W}$ is a matrix of weights.

Types of activation function There are several types of activation function that can be used in neural networks, I considered the following:

Identity	$\sigma(x) = x$
Rectified Linear Unit (ReLU)	$\sigma(x) = \max(x, 0)$
Softplus	$\sigma(x) = \ln(1 + e^x)$

TanH $\sigma(x) = \tanh(x)$

Most of them map the value of x onto a specific range, which is useful if the output must also fall in a certain range. Normally the activation function is based on which layer the perceptron is in. Activation functions such as ReLU are also useful in hidden layers to produce a non-linear transformation.

Type of NN There are many types of neural networks, I decided to use the simplest one: *fully connected*, *feedforward* neural network.

Fully connected Input for a perceptron in layer n is composed of output of each perceptron in layer $n - 1$.

Feedforward The data only flows one way in the network, there are no loops.

An example of a structure of such neural network can be seen in figure 3.5.

3.4.4 System overview

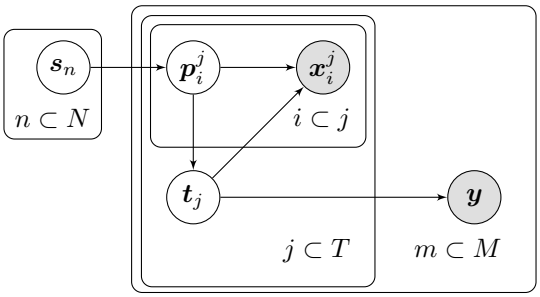


Figure 3.6: Summed up view of the prediction model.

There are N players tracked by the system, participating in M matches. In each match there are T teams. $\boldsymbol{x}_i^j$ are the results of player i in team j and $\boldsymbol{y}$ are the overall match results. All variables are vectors.

The model assumes that there are two types of results for each match:

- Player specific results, such as the amount of gold earned or the number of enemies killed.
- Overall match results, such as which team won or how long the match took.

The naive approach is to have only one neural network which takes player skill estimates as inputs and outputs predictions of results. There are several problems with such system:

- As the size of input grows, so does the size of the neural network required to accurately process them. This leads to massive neural network which takes a long time to process data and therefore quite inefficient.
- As the size of the neural network grows, the number of examples required to train it also increases. Although I had quite a large data-set, most successful approaches use millions or even tens of millions of data-points. Therefore, I judged that my dataset was not large enough to properly train such a big network.

Therefore I made a few assumptions to create a graphical model which resolves those problems. The assumptions are:

- Skills of all players playing on the same team can be combined into a smaller set of skills.
- Player specific results only depend on the skills of relevant player and combined skills of each team taking part in the match.
- Overall match results only depend on combined team skills.

The summary of resulting system is seen in figure 3.6.

There are two models in the system: predicting results from inferred player skills and inferring player skills from their match history. An example of a prediction model made for a game in which four players play in two teams can be seen in figure 3.7 and a factor graph expansion can be found in Appendix C. The skill update model is the same for all games and can be seen in figure 3.8.

Training both the variable prediction model and skill update model at the same time would greatly increase the data required for processing. It will also raise some difficult problem such as *vanishing gradient*. Unfortunately I did not have enough time to solve these problems properly. Instead, I decided to use the technique called *coordinate descent*, to optimise the models alternatively.

Coordinate descent When there are multiple parameters to optimise, instead of optimising them at the same time they can be optimise alternatively. Each parameter is optimised until it reached a stable value, then the next parameter is optimised. Once each parameter is optimised, loop back and start again from the first one. This cycle is repeated until all parameters reach sufficiently stable values across multiple cycles.

3.4.5 Prediction model

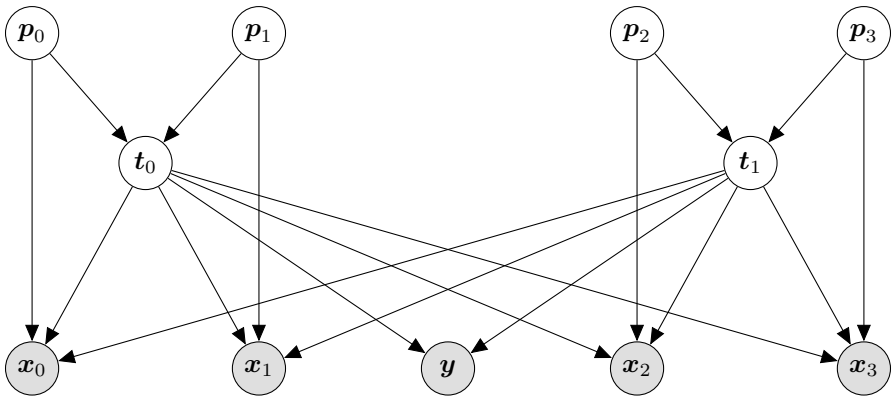


Figure 3.7: An example of an inference model for four players in two teams. Grey circles represent observed values and white circles represent latent variables.

There are four variables in this model: $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t}, \boldsymbol{p}$. To make predictions we would like to calculate $P(\boldsymbol{x}, \boldsymbol{p})$ and $P(\boldsymbol{y}, \boldsymbol{p})$. As was said above, calculating these probabilities directly is expensive, by expressing them using the terms in the model, redundant links can be

removed greatly simplifying the computation. First the conditionals probabilities are expressed according to model structure:

$$P(\boldsymbol{x}_i, \boldsymbol{t}, \boldsymbol{p}) \quad \text{and} \quad P(\boldsymbol{y}, \boldsymbol{t}, \boldsymbol{p})$$

Using product rule the joint probabilities are expanded into several conditional probabilities:

$$P(\boldsymbol{x}_i, \boldsymbol{t}, \boldsymbol{p}) = P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}) P(\boldsymbol{t} \mid \boldsymbol{p}) P(\boldsymbol{p})$$

Using the assumption that player results only depend on the skills of that player and team skills, most of the players can be integrated out, leaving:

$$P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}) = P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}_i)$$

Doing the same to other parts leaves:

$$P(\boldsymbol{x}_i, \boldsymbol{t}, \boldsymbol{p}) = P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}_i) P(\boldsymbol{t} \mid \boldsymbol{p}) P(\boldsymbol{p}) \qquad P(\boldsymbol{y}, \boldsymbol{t}, \boldsymbol{p}) = P(\boldsymbol{y} \mid \boldsymbol{t}) P(\boldsymbol{t} \mid \boldsymbol{p}) P(\boldsymbol{p}) \tag{3.1}$$

Which is now expressed in terms of conditional probabilities we can calculate and priors which we know.

The skills of each player represent the prior on their performances. Each player starts with their skill represented by n standard Gaussians. After each match, the parameters of those Gaussians are adjusted, such that for the next match the priors are different.

Neural Networks Conditional probabilities such as $P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}_i)$ are specific to every game and are most likely very complex. In such cases I use machine learning techniques to approximate there probabilities, since this is a probability of a vector given other vectors, neural networks are the perfect candidate to use. All of the variables in the model are defined using distributions, therefore neural networks must output the parameters of required distribution. For example, for gamma distribution:

$$P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}_i) = \text{gamma}(\boldsymbol{x}_i; \boldsymbol{k}_{\boldsymbol{\gamma}}(\boldsymbol{t}, \boldsymbol{p}_i), \boldsymbol{\theta}_{\boldsymbol{\gamma}}(\boldsymbol{t}, \boldsymbol{p}_i))$$

Where parameters $\boldsymbol{k}_{\boldsymbol{\gamma}}$ and $\boldsymbol{\theta}_{\boldsymbol{\gamma}}$ are outputs of a neural network with inputs $\boldsymbol{p}$ and $\boldsymbol{t}$ parametrised by $\boldsymbol{\gamma}$.

3.4.6 Inferring player performance

Player performance is a latent variable, therefore there are no values available to train a neural network. To infer the performance of each player from the match results, the conditional probability $P(\boldsymbol{p} \mid \boldsymbol{t}, \boldsymbol{x})$ has to be calculated. Using Bayes' theorem this can be transformed to use conditional probabilities already known:

$$P(\boldsymbol{p} \mid \boldsymbol{t}, \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid \boldsymbol{t}, \boldsymbol{p}) P(\boldsymbol{t} \mid \boldsymbol{p}) P(\boldsymbol{p})}{P(\boldsymbol{t}, \boldsymbol{x})}$$

Unfortunately, because the conditional probabilities are calculated using neural networks, calculating the joint probability $P(\boldsymbol{t}, \boldsymbol{x})$ is almost impossible. Normal approach would be to use sum rule on the whole probability $P(\boldsymbol{p}, \boldsymbol{t}, \boldsymbol{x})$ and integrate $\boldsymbol{p}$ out:

$$P(\boldsymbol{t}, \boldsymbol{x}) = \int P(\boldsymbol{p}, \boldsymbol{t}, \boldsymbol{x}) \; d\boldsymbol{p} \tag{3.2}$$

That would require to do a sum over all possible weights in all of the neural networks, which is intractable.

Variational Methods Using variational inference, the conditional probability $P(\boldsymbol{p} \mid \boldsymbol{t}, \boldsymbol{x})$ can be approximated with a tractable distribution $q_{\boldsymbol{\theta}}(\boldsymbol{p} \mid \boldsymbol{t}, \boldsymbol{x})$ with free parameters $\boldsymbol{\theta}$ that are close in KL-divergence to the exact posterior. Therefore the KL-divergence has to be optimised, which is equivalent to maximising the evidence lower bound of the objective function (ELBO):

$$\mathbb{E}_{q(\boldsymbol{p}|\boldsymbol{t},\boldsymbol{x})}\left[\frac{P(\boldsymbol{p},\boldsymbol{t},\boldsymbol{x})}{q_{\boldsymbol{\theta}}(\boldsymbol{p} \mid \boldsymbol{t},\boldsymbol{x})}\right]$$

If $q_{\boldsymbol{\theta}}(\boldsymbol{p} \mid \boldsymbol{t}, \boldsymbol{x})$ is constructed such that it can be sampled from using Monte Carlo, then the integral from equation 3.2 can be approximated by a sum:

$$\int P(\boldsymbol{p},\boldsymbol{t},\boldsymbol{x})\ d\boldsymbol{p} \approx \frac{1}{k}\sum_{k=1}^{K}\frac{P(\boldsymbol{p}^k,\boldsymbol{t},\boldsymbol{x})}{q_{\boldsymbol{\theta}}(\boldsymbol{p}^k \mid \boldsymbol{t},\boldsymbol{x})} \qquad k = 1,\ldots,K$$

where $\boldsymbol{p}^k$ is the k-th sample.

Team skills are also latent variables which have to be estimated from match results. Another q-function has to be trained to estimate $P(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{y})$.

3.4.7 Skill update model

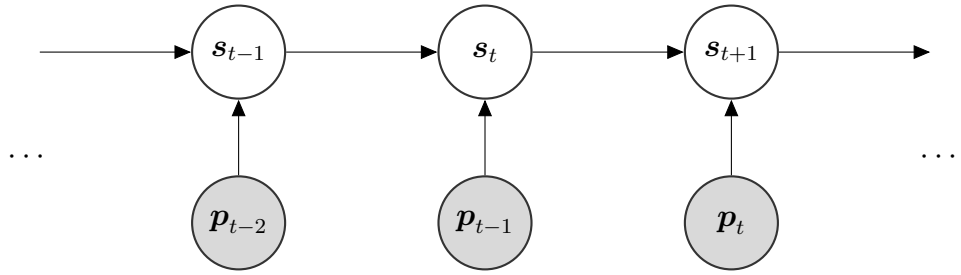


Figure 3.8: Skill update model

After every match the system updates its belief about the player's skills using the previous belief and player's performance in the match. This forms a chain of beliefs about player's skill at each particular moment in time. After each match, for each player the model updates the player skill using:

$$P(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{p}_{t-1}) = \mathcal{N}(\boldsymbol{s}_t; \boldsymbol{\mu}_{\boldsymbol{\lambda}}(\boldsymbol{s}_{t-1}, \boldsymbol{p}_{t-1}), \boldsymbol{\sigma}_{\boldsymbol{\lambda}}(\boldsymbol{s}_{t-1}, \boldsymbol{p}_{t-1}))$$

Where $\boldsymbol{\mu}_{\boldsymbol{\lambda}}$ and $\boldsymbol{\sigma}_{\boldsymbol{\lambda}}$ are outputs of the neural network[3] with inputs $\boldsymbol{s}_{t-1}$ and $\boldsymbol{p}_{t-1}$, which is parametrised by $\boldsymbol{\lambda}$.

3.4.8 Training neural networks

To train the system the likelihood of the conditional probabilities has to be maximised and KL-divergence of all approximated functions has to be minimised. To train a neural network a loss function is required.

[3]Code for this neural network is in Appendix B.

Loss function for conditional probabilities All of the outputs of conditional probabilities are probability distribution. Therefore, for continuous variables, probability density function (pdf) of their distribution can be used as the loss function. Similarly, for discrete variables, probability mass function (pmf), can be used as the loss funciton. The higher the pdf/pmf, the better the estimate, but loss function is minimised, therefore we need to negate the pdf/pmf to define loss. For example a loss function for a Gaussian distribution would be:

$$\mathcal{L}(x, \mu, \sigma) = -\frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.3}$$

where x is the desired value and μ & σ are estimated parameters.

Optimising loss function The loss function is defined across the whole system, therefore we need to multiply the pdfs of all the distributions. Multiplication is problematic due to two problems:

- It is computationally expensive
- Multiplying small probabilities can lead to underflow, which will mean that the final probability might be equal to zero.

The log of the loss function is taken, it turns products into sums thereby reducing required computation and reduces the range of exponent preventing overflow and underflow:

$$\mathcal{L} = \log\left(\prod_i p_i(x)\right) = \sum_i \log(p_i(x))$$

Since all probabilities are positive log of them is always defined, log is also a monotonically increasing function which means that decreasing log also decreases the function.

Taking log of the samples produced by the variational methods also reduces the variance of samples, which greatly decreases the number of samples required for approximation. In my case I found that variance of samples is small enough that only one sample was required during training.

Computing the pdf itself also poses a problem, since potentially very small and very large values are multiplied together the pdf itself can underflow or overflow. To prevent those problems, the log in the loss function can be pushed further in, e.g. Gaussian from equation 3.3 becomes:

$$\mathcal{L}(x, \mu, \sigma) = \frac{\log(2) + \log(\pi)}{2} + \log(\sigma) + \frac{(x-\mu)^2}{2\sigma^2}$$

The optimising function can only change the parameters produced by the neural networks, therefore constants in the loss function do not matter and can be removed, leaving:

$$\mathcal{L}(x, \mu, \sigma) = \log(\sigma) + \frac{(x-\mu)^2}{2\sigma^2} \tag{3.4}$$

Minimising KL-divergence Recent paper on variational inference optimisation [6] mentions three methods of minimising KL-divergence:

Full MC form $\mathcal{L}(\theta) = \mathbb{E}_q[\ln P(\boldsymbol{x}, \boldsymbol{p}) - \ln q_\theta(\boldsymbol{p} \mid \boldsymbol{x})]$

Entropy form $\mathcal{L}(\theta) = \mathbb{E}_q[\ln P(\boldsymbol{x} \mid \boldsymbol{p})] + \mathbb{H}[q_\theta]$

KL form $\mathcal{L}(\theta) = \mathbb{E}_q[\ln P(\boldsymbol{x} \mid \boldsymbol{p})] - \mathrm{KL}(q_\theta \parallel P(\boldsymbol{p}))$

Through trial and error, I found that *entropy* form worked best on my dataset. Therefore final equation to be optimised for prediction model was:

$$\mathcal{L} = \mathbb{E}_{q\sim\boldsymbol{z}}\left[\sum_i (\ln P(\boldsymbol{x}_i \mid \boldsymbol{t}, \boldsymbol{p}_i)) + \ln P(\boldsymbol{y} \mid \boldsymbol{t}) + \ln P(\boldsymbol{t} \mid \boldsymbol{p}) + \ln P(\boldsymbol{p})\right] + \mathbb{H}[q_\theta(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{y})] + \sum_i (\mathbb{H}[q_\psi(\boldsymbol{p}_i \mid \boldsymbol{x}_i, \boldsymbol{t})])$$

This way, all neural networks are optimised at the same time. The loss equation for skill update model can be constructed similarly.

Restricting the range of the output Some of the parameters have to fall within certain range, but the neural output produced by a node in a neural network is usually not bound to any range. This can be solved by using an appropriate activation function on the final layer of the neural network. *ReLU* is an obvious choice for enforcing positive values, but it is problematic in that the optimisation function may not properly adjust the weights of the NN. Instead, I decided to use *SoftPlus* activation function.

Although, sometimes different outputs of the same neural network must have different bounds placed on them. Since the activation function is usually defined on the whole layer, it is frequently better to keep activation function of the last layer as *linear* and apply appropriate activation function on each element individually.

3.4.9 Prediction

To predict the values for a particular match-up the players' skills corresponding to players in the match-up are taken and feed them into the top of the model. Since the system tracks the skill of the players through time, the skill of players at the appropriate time has to be found.

3.4.10 Restricting the time frame of training data

Many popular multiplayer computer games change their rules a few times a year to keep the game fresh and players interested as well as to balance the game. In addition, the strategies players employ evolve through time and different skills become more or less important in the game. If the system is trained on the whole history of matches, it is optimised to predict results across the whole time-frame, but most of the time only predicting future events makes sense. Therefore, to improve the accuracy of the system only the most recent matches are used for training, in which players employ current strategies. Although, there is a trade-off: the smaller the time-frame the fewer matches are available for analysis.

3.5 Summary

In this chapter the structure and design of the main system was introduced and explained. To design the main system, a graphical model was constructed, then conditional

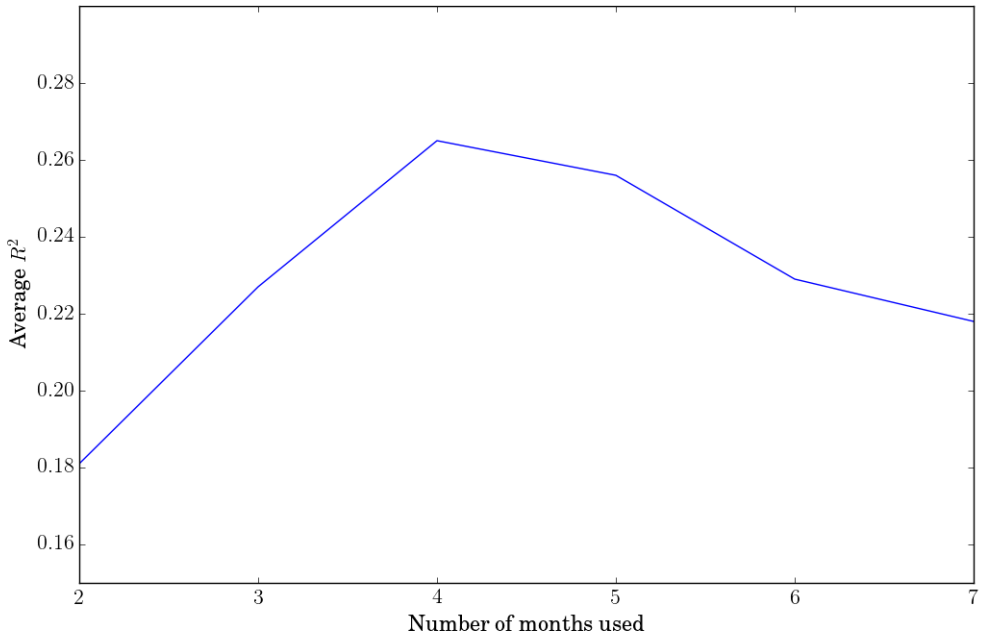


Figure 3.9: Average R^2 with respect to time range of data considered. Here it can seen that best choice would be around four months. This correlates with the fact that the rules of the game change roughly seasonally.

probabilities were estimated using neural networks. Using neural networks for conditional probabilities, meant that Bayes' theorem could not be used to make a model for performance estimate. To solve that problem variational methods were used to approximate the reverse conditional probability using sampling.

Chapter 4

Evaluation

This chapter discusses how the final system was optimised and evaluated. It introduces *train/validation/test* split and how it is used to evaluate machine learning models. In this chapter I also discuss how validation results can be used to optimise the model using *hyper-parameter search*.

4.1 Preprocessing

Once neural networks have been trained, matches have to be processed in chronological order to update the player skills. To make evaluation quicker a dictionary can be created which contains match id as keys and skills of the players taking part in that match. Player skills have to be calculated before processing the match and updating them, to accurately reflect real world scenario.

4.2 Train/Validation/Test split

One of the most important rules when evaluating machine learning systems is that the algorithm must be tested on previously unseen data. Therefore, right at the start the data can be split into two parts: *train/validation* and *test*. *Train/validation* set will be used to train the algorithm and adjust hyper-parameters. *Test* set will be used for evaluation of the algorithm.

4.2.1 Preventing Over-fitting

There are several methods to prevent over-fitting:

Using simple models If the model has the correct complexity, it will only be able to learn the data and not the noise. Another benefit of this approach is that training and prediction times are shorter. Using this method the model can be trained until loss stops decreasing. Unfortunately, if the model is too simple the model can suffer from under-fitting.

Model regularization Another approach is to penalise complexity in the model. For example the size of weights in neural networks can be limited to a certain value. Unfortunately, this can also lead to under-fitting.

Early stopping Over-fitting can also be prevented by emulating testing scenario during training. For that purpose the train/validation part is split into train and validation. The algorithm is trained on train part and after every iteration it is tested on validation part. When the validation error stops decreasing the training is stopped.

I decided to use validation since it does not suffer from potential of under-fitting and also has added benefit of allowing selection of hyper-parameters. The best hyper-parameter combination can be found by first training the system using each combination and validating it, and then comparing the results of each validation.

4.2.2 Cross-validation

If only one part of training data is used for validation, it might not properly represent the performance of the algorithm. To combat that, cross-validation can be used: train/validation cycle is run multiple times, each time a new set is used for validation. To ensure complete coverage of the train-set *k-fold cross-validation* can be used:

1. The train/validation set is divided into k subsets.
2. A subset is picked that has not been validated on, the rest are used for training and it is used for validation.
3. Step two is repeated until all k subsets have been validated on.
4. The results are averaged.

4.2.3 Split ratio

When splitting the dataset into train/validation and test parts it is important to keep in mind the total size of the dataset. Making the size of the test part too small can lead to statistically insignificant results and also increase the chance of sampling error. On the other hand, making the size of training data too small can lead to under-fitting for the model. I decided to use standard 80:20 train/test split using the Pareto principle.

4.3 Hyper-parameter selection

The designed model includes three hyper-parameters which have to be adjusted for each game[1].

1. Size of the vector representing player skills.
2. Size of the vector representing team skills.
3. Size and structure of each neural network.

They are ranked in order of importance, adjusting the first should one give the most accuracy increase, but also increase required resources the most.

As can be seen in figure 4.1 the accuracy of the system is mostly dependent on the size of vectors used to represent player skills. Increasing the size of team skills does not

[1]There are extra hyper-parameters such as optimisation rate, which are not specific to any game. All of the hyper-parameters which I considered are listed in appendix D.

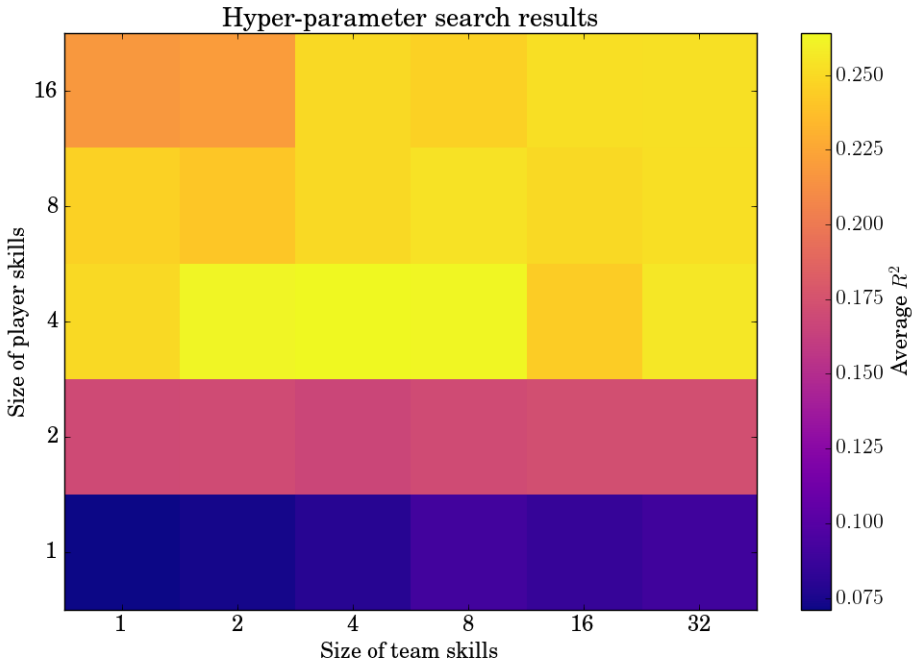


Figure 4.1: Hyper-parameter search

As can be seen, increasing the size of player skills has much bigger effect than changing size of team skills. **N.B.** Poor performance with small player skills is not necessarily because the system cannot predict well, but rather because it frequently cannot model the data well, which leads to model only learning the mean.

have such a large effect. I have found that the system performs best on my dataset when the size of team skills is between half and twice the size of player skills.

Another thing to keep in mind is that if the size of player skills is too small, the system may fail to learn the patterns present in the data, which will lead to it predicting the mean. With my dataset the system failed about two thirds of the time when the size of player skills was equal to one. This failure rate reduced to about a third when the size was increased to two and was negligible with size of three and greater.

Learning rate selection I have used Adam optimiser to train the neural networks. I have found that using a learning rate less than 5e-4 may lead to system failing to learn the data, learning rates greater than that all performed similarly well.

The only important observation is that since I am using stochastic gradient descent, the system tends to oscillate around the optimum. The size of such oscillations is determined by the learning rate, higher learning rate leads to bigger oscillations. Therefore selecting a smaller learning rate usually produces slightly better results, but takes longer to train, although the benefit is not very big. This can be seen on figure 4.2, which plots five run average of each learning rate.

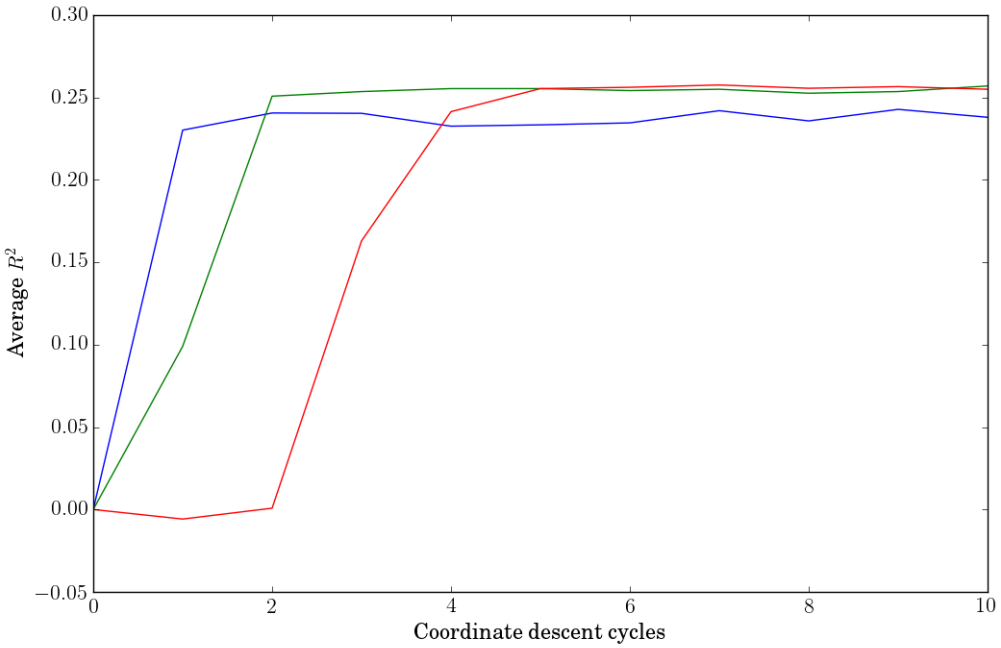


Figure 4.2: Learning rate comparison

4.4 Performance Evaluation

The system is designed such there is as little coupling as possible between the two models. This allowed me to work on each model independently and also made evaluation simpler by allowing me to evaluate both models separately from each other.

4.4.1 Prediction model evaluation

Prediction model is responsible for two functions in the system:

- Estimating player performance in a particular match
- Predicting player results given player skills.

To evaluate how well different configurations works, pairs of skills and results are taken and split into train/validation parts. Then the different configurations are trained and validated on the respective parts. After the results for all configurations are obtained, the best configuration is chosen and used to estimate player performances in each match. The configuration consists of settings for the five different neural networks from equation 3.1, namely $P(\boldsymbol{y} \mid \boldsymbol{t})$, $P(\boldsymbol{x}_i \mid \boldsymbol{tp}_i)$, $P(\boldsymbol{t} \mid \boldsymbol{p})$, $q(\boldsymbol{p}_i \mid \boldsymbol{t}, \boldsymbol{x}_i)$, $q(\boldsymbol{t} \mid \boldsymbol{x}, \boldsymbol{y})$.

Using hyper-parameter search I discovered that the model functions quite well even without any hidden layers, adding a single hidden layer improves performance a bit. Increasing the number of hidden layer to two or more gave negligible performance increase and therefore was not used. Therefore all five neural networks use a simple one hidden layer configuration.

4.4.2 Skill update model evaluation

The accuracy of prediction model depends directly on how accurate the skill estimations of the skill update models are, therefore it is important that skill estimation is as accurate as possible. To optimise the skill update model, sets of player skills, performances and next performances are taken and then split into train/validation parts. Skill update model uses only one neural network, $P(\boldsymbol{p}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{p}_t)$, therefore configuration consists only of the structure of this neural network. Similarly to prediction model, different configurations are trained and validated and the best is chosen. The chosen neural network is then used to update the skills of each player at time t.

Similarly to neural network in prediction model, neural network without hidden layers worked quite well. Although in this model, sufficient improvements have been seen in using up to threeZ hidden layers.

4.4.3 Overall system evaluation

To evaluate the system overall co-ordinate descent is performed where prediction model and skill update model are optimised alternatively. At the start there are no estimates of player skills, therefore the model assumes that all skills are modelled as a standard Gaussian: $\mathcal{N}(0, 1)$. In most cases the optimisation is quite fast only taking two to five rounds of co-ordinate descent.

The co-ordinate descent follows a a simple procedure:

1. Optimise prediction model and update player performances.

2. Optimise skill update model and update player skills.

3. Validate the whole system.

4. Compare validation result to previous, if it is better save the state of the system and go back to step 1. If the result is worse, load back the state of the system from last validation and proceed to next step.

5. Test the system using the loaded configuration and report the result.

The system is tested and validated by running the prediction model using the player skill estimates from appropriate time. Since the system uses cross-validation, steps one, two and three are performed k times, once for each fold of k-fold cross-validation. The validation result in step four is the mean of cross-validation results.

4.5 Results

4.5.1 Requirements Satisfaction

There are five requirements for this project, all of which were satisfied.

As I said in in the subsection 2.2.2, the original accuracy requirement was not achieved. If the requirement is interpreted with mistake corrected then the requirement is achieved on one variable.

Predicting the values of different games Unfortunately, I did not have enough time test my system on another dataset. This was due to the fact that I could not find any suitable dataset available on the internet and collecting one would require a few weeks. Although, the system was constructed in such a way that it does not directly rely on any patterns in the current dataset. The structure of the system is similar to that of TrueSkill, and that system works on many hundreds of games every day. Therefore, I believe that this system should be able to predict similar summaries from variety of games as long as those games require sufficient level of skill.

Prediction based on skills of all players. The prediction model includes latent variables for team skills which are computed from player skills during prediction. As can be seen in figure 4.1, changing the size of these skills changes the performance of the system. Therefore I conclude that team skills must be utilised in some form during prediction. Since team skills are computed from skills of all players, this means that the prediction is affected by the skills of all players and therefore I consider this requirement satisfied.

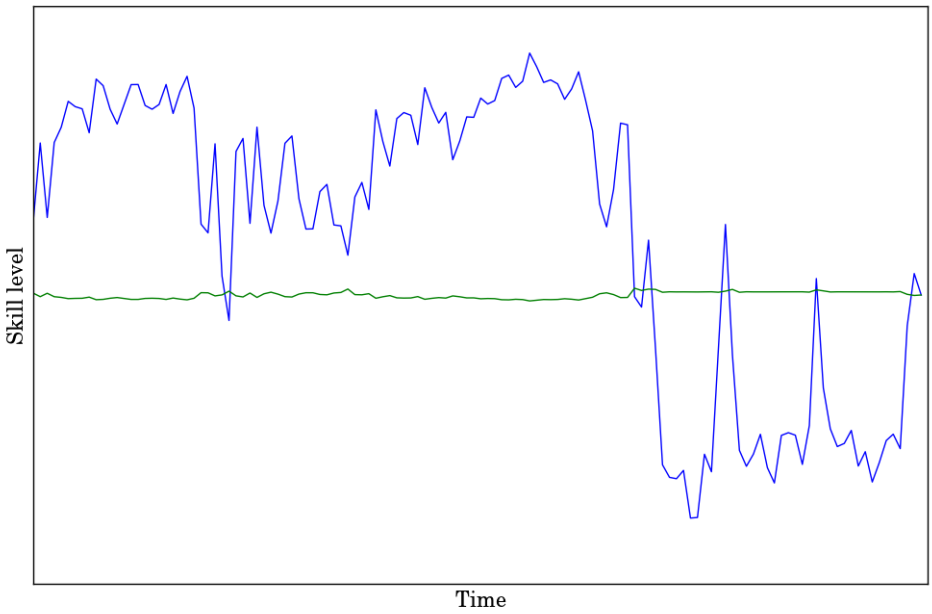


Figure 4.3: Change in player's skill through time

These skills are generated using variational inference, I can not tell what each of them represent, therefore they are unlabelled.

Time dependent player skill As can be seen on figure 4.3, the magnitude of player's skill varies through time. The model predicts different values for different input, therefore the values predicted by the model will also vary as the players skills varies. Consequently, I consider this requirement satisfied.

Time of execution Due to me using approximation by variational methods, the training of the system and prediction times are very small. Depending on the values of hyper-

parameters the training of the system takes between a few minutes and a few hours and prediction in most cases is always extremely quick taking less than a second to predict several hundred match results. Therefore, I consider this requirement satisfied.

Accuracy and precision of results As can be seen in figure 4.4, prediction of only one variable has achieved R^2 values of more than 0.5, therefore while this requirement was partly satisfied.

4.5.2 Quantitative Results

	GPM	**XPM**	**CS**	**DN**	**Kills**	**Deaths**	**Assists**	**Level**
original σ^2	0.056	0.044	0.054	0.037	0.047	0.050	0.057	0.048
error σ^2	0.027	0.025	0.033	0.025	0.038	0.048	0.055	0.037
R^2	**0.510**	0.441	0.397	0.308	0.179	0.047	0.033	0.223

Figure 4.4: Quantitative results.

As was expected, different variables were predicted with different levels of success. The results of predicting the eight main variables[2] can be seen in figure 4.4. Unfortunately, the accuracy requirement has only been achieved on one variable, but three more variables have shown quite good results as well.

4.5.3 Result validity

To check the validity of my results I will use *one-tailed*, *one-sample* T-test. I assume that R^2 values produced are normally distributed. To produce the sample, I run my train, validate and test ten times, each time splitting the data randomly. My null hypothesis will be that R^2 value is less than or equal 0.5 and my alternative hypothesis is that it is greater:

$$H_0 : R^2 \leq 0.5 \qquad H_1 : R^2 > 0.5$$

My significance level is 0.01 and my sample size is ten, with sample being:

$$0.516,\ 0.506,\ 0.503,\ 0.517,\ 0.502,\ 0.508,\ 0.519,\ 0.522,\ 0.499,\ 0.507$$

Which leads to sample mean being 0.5099 and sample standard deviation 0.0079, therefore test statistic is:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}} = \frac{0.5099 - 0.5}{0.0079/\sqrt{10}} = 3.96$$

3.96 is greater than 2.82 required, therefore my the test result is significant and null hypothesis has to be rejected, hence R^2 is greater than 0.5.

4.5.4 Baseline comparison to existing systems

There do not exist any systems which predict player results in the game, the closest thing that I can use to compare are the rating systems. TrueSkill is one of the most recent

[2]The explanation of what each variables signifies in game can be found at the end of project proposal in appendix E

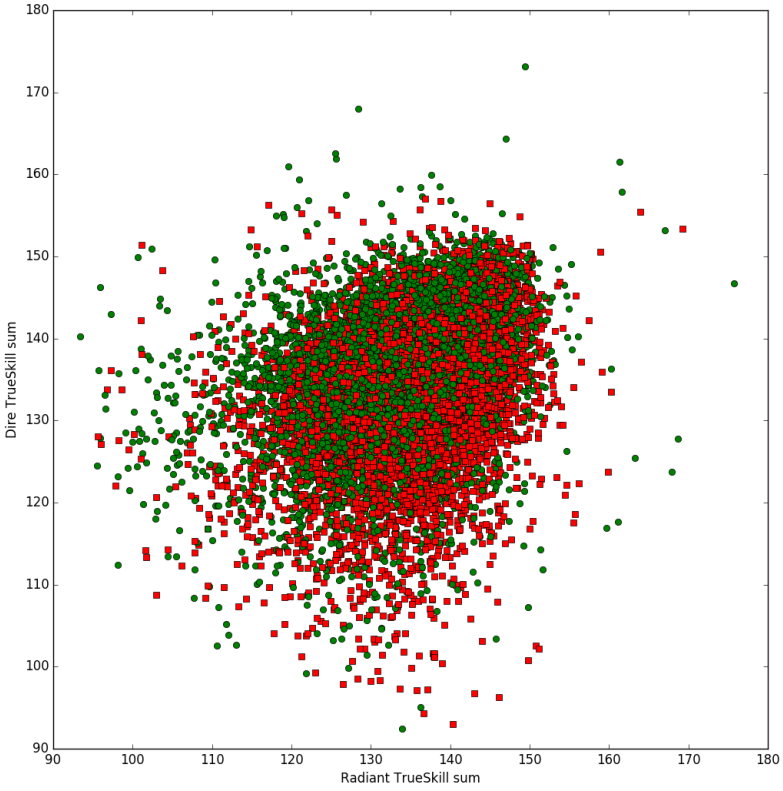


Figure 4.5: Distribution of skills produced by TrueSkill
Red points represent matches when Radiant (horizontal axis) team won, while green points represent matches when Dire (vertical axis) team won.

open such systems, I have already used parts of it in my prototype, therefore comparing my system to it required little effort.

TrueSkill library does not directly contain the functionality to predict the outcome of the game, only the probability of a draw. To make this comparison I made a simple assumption: if the A team's skill calculated by TrueSkill was greater than team B' skill, I assumed that TrueSkill expected team A to win more when playing against team B. Using this assumption the accuracy of TrueSkill on my dataset was about 0.507. The graph showing distribution of teams' skills and match outcomes can be seen on figure 4.5.

My system also does not output the winning team directly, but rather outputs a Bernoulli distribution. I made a similar assumption, if $p > 0.5$ I assumed team A would win and team B otherwise. Using this assumption, my system had an accuracy of 0.548, while this is not great, it is much better than TrueSkill accuracy.

4.6 Summary

In this chapter I described how the system was optimised and tested. The evaluation of the system showed that all five requirements have been satisfied and that primary success criteria has been achieved. The result was checked using a hypothesis test.

Chapter 5

Conclusions

5.1 Summary of Achievements

The project has seen reasonable level of success. All five requirements have been satisfied. The corrected success criteria was achieved on one variable and three other variables were predicted sufficiently well.

The project has shown that using simple aggregated player statistics is not sufficient to accurately predict player results, as the skill of most players changes with time. By creating a graphical model which can track the skills of the players through time prediction accuracy has been greatly increased. The graphical model operates on n dimensional vectors and manually determining correct relationships between each value in the vectors was considered impractical. Instead, neural networks were used to learn such relationships from available data.

The biggest challenge was to decide what each value in the vector of skills should represent. Since that would require detailed knowledge of the game, variational methods were used to let the model create the interpretation.

5.2 Future work and Improvement Ideas

During the work on this project I had many ideas about how different things can be done and what methods can be used. Unfortunately, I did not have enough time to try many of them, but I think they should definitely be tested in the future, these include:

Using Kalman filter for skills updates The system represents skills and performances as gaussians, Kalman filter is a perfect candidate for the task.

Using RNN for skill update The skill update model treats player performances as observations, therefore using recurrent neural networks with previous n matches as input might produce better results.

Using more results for each match The current system only used a very small amount of data available about the game, using more information about each match would definitely improve performance, it would be interesting to know how much.

Additionally, by using predictions produced by the current system a more advanced matchmaking algorithm can be developed that would take into account events happening during the match, not just the final outcome. Combined with analysis of what players truly want from the game, a better playing experience can be delivered to them.

5.3 Lessons Learned

The main lesson I learned, is that machine learning is not the all-powerful tool that can solve all problems and that most of the time there are still a lot of trade-offs to be made. Even when using neural networks features have to be selected or created, since frequently using all the data is impractical. Although, if these challenges are resolved machine learning can fairly easily deliver performance comparable to that of hand-crafted systems, while requiring less effort.

Bibliography

[1] Atish Agarwala and Michael Pearce. *Learning Dota 2 Team Compositions*. Tech. rep. tech. rep., Stanford University, 2014.

[2] DataInformed. *Predict the Winners of the Big Games with Machine Learning*. 2016. URL: http://data-informed.com/predict-winners-big-games-machine-learning (visited on 04/16/2017).

[3] Arpad E. Elo. *The rating of chessplayers, past and present*. New York: Arco Pub., 1978.

[4] Forbes. *Riot Games Reveals 'League of Legends' Has 100 Million Monthly Players*. 2016. URL: https://forbes.com/sites/insertcoin/2016/09/13/riot-games-reveals-league-of-legends-has-100-million-monthly-players (visited on 04/16/2017).

[5] Ralf Herbrich, Tom Minka, and Thore Graepel. "TrueSkill(TM): A Bayesian Skill Rating System". In: MIT Press, Jan. 2007, pp. 569–576. URL: https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system/.

[6] G. Roeder, Y. Wu, and D. Duvenaud. "Sticking the Landing: An Asymptotically Zero-Variance Gradient Estimator for Variational Inference". In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.09194 [stat.ML].

[7] Ruby C. Weng and Chih-Jen Lin. "A Bayesian Approximation Method for Online Ranking". In: *J. Mach. Learn. Res.* 12 (Feb. 2011), pp. 267–300. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.1953057.

Appendix A

Example of a Summary

Some parts of this summary are repetitive and take up a lot of space. In such parts, all but the first set were hidden, indicated by comments.


```
{
   "result":{
      "players":[
         {
            "account_id":87382579,
            "player_slot":0,
            "hero_id":60,
            "item_0":239,
            "item_1":92,
            "item_2":46,
            "item_3":108,
            "item_4":29,
            "item_5":6,
            "backpack_0":0,
            "backpack_1":0,
            "backpack_2":0,
            "kills":3,
            "deaths":7,
            "assists":17,
            "leaver_status":0,
            "last_hits":72,
            "denies":2,
            "gold_per_min":254,
            "xp_per_min":298,
            "level":14,
            "hero_damage":5682,
            "tower_damage":277,
            "hero_healing":170,
            "gold":73,
            "gold_spent":9220,
            "scaled_hero_damage":0,
            "scaled_tower_damage":0,
            "scaled_hero_healing":0,
            "ability_upgrades":[
               {
                  "ability":5275,
                  "time":886,
                  "level":1
               },
```

```
                "There is one set per level and players can advance up to level 25"
            ]
         },
         "There are nine more sets of player data"
      ],
      "radiant_win":false,
      "duration":2368,
      "pre_game_duration":90,
      "start_time":1471142574,
      "match_id":2569610900,
      "match_seq_num":2244488581,
      "tower_status_radiant":1540,
      "tower_status_dire":1956,
      "barracks_status_radiant":3,
      "barracks_status_dire":63,
      "cluster":113,
      "first_blood_time":89,
      "lobby_type":1,
      "human_players":10,
      "leagueid":4664,
      "positive_votes":38429,
      "negative_votes":2503,
      "game_mode":2,
      "flags":1,
      "engine":1,
      "radiant_score":27,
      "dire_score":30,
      "radiant_team_id":2512249,
      "radiant_name":"Digital Chaos",
      "radiant_logo":692780106202747975,
      "radiant_team_complete":1,
      "dire_team_id":1836806,
      "dire_name":"the wings gaming",
      "dire_logo":352770708597344369,
      "dire_team_complete":1,
      "radiant_captain":87382579,
      "dire_captain":111114687,
      "picks_bans":[
         {
            "is_pick":false,
            "hero_id":79,
            "team":1,
            "order":0
         },
         "There are about 20 sets in this list, detailing the drafting phase"
      ]
   }
}
```

Appendix B

Code for skill update model

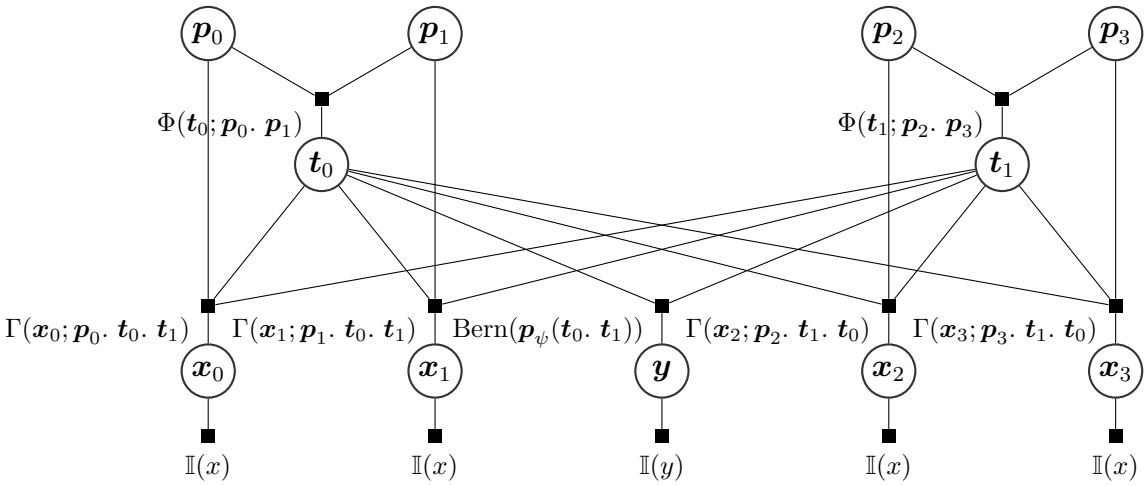
This as an example of how skill update can be implemented with TensorFlow and Keras. Implementation of prediction model is similar, but in addition neural networks have to be connected between each other.

```
import keras
import numpy as np
import tensorflow as tf
from keras.layers import Dense


def make_sql_nn(in_dim: int, out_dim: int):
    """
    Create a simple fully-connected, feedforward neural network with two hidden layers.
    """
    p = keras.models.Sequential()
    p.add(Dense(units=int((in_dim + out_dim) * 2 / 3), input_dim=in_dim, activation='relu', kernel_initializer='random_normal'))
    p.add(Dense(units=int((in_dim + out_dim) / 3), activation='relu', kernel_initializer='random_normal'))
    p.add(Dense(units=out_dim, activation='linear', kernel_initializer='random_normal'))
    return p


def log_normal(x, mu, sigma):
    """
    Calculate log of a gaussian pdf.
    """
    error = -(tf.pow((x - mu) / sigma, 2) / 2 + tf.log(tf.clip_by_value(sigma, 1e-8, np.infty)))
    return tf.reduce_sum(error, 1)


def make_mu_and_sigma(nn, tensor):
    """
    Apply neural network to the tensor and then separate the output into parameters for a gaussian distribution.
    """
    mu, log_sigma = tf.split(nn(tensor), num_or_size_splits=2, axis=1)
    sigma = tf.exp(tf.clip_by_value(log_sigma, -5, 5))
    return mu, sigma
```

```
33
34
35  PLAYER_DIM = 8
36
37  # TensorFlow input variables
38  player_pre_skill = tf.placeholder(tf.float32, shape=(None, PLAYER_DIM * 2))
39  player_performance = tf.placeholder(tf.float32, shape=(None, PLAYER_DIM))
40  player_next_performance = tf.placeholder(tf.float32, shape=(None, PLAYER_DIM))
41
42  # Create a neural network
43  nn = make_sql_nn(PLAYER_DIM + PLAYER_DIM * 2, PLAYER_DIM * 2)
44
45  # Apply neural network to input
46  nn_input = tf.concat([player_pre_skill, player_performance], axis=1)
47  player_post_skill, sigma = make_mu_and_sigma(nn, nn_input)
48  log_result = log_normal(player_next_performance, player_post_skill, sigma)
49
50  # Save updated skill to a variable
51  post_skill = tf.concat([player_post_skill, sigma], axis=1)
52
53  # Define loss to optimise
54  loss = -tf.reduce_mean(log_result)
```

Appendix C

An example of factor graph



$$\Theta(\boldsymbol{p}; \boldsymbol{s}) = \mathcal{N}(\boldsymbol{p}; \boldsymbol{\mu}_\theta(\boldsymbol{s}), \boldsymbol{\sigma}^2_\theta(\boldsymbol{s}))$$
$$\Phi(\boldsymbol{t}; \boldsymbol{p}_i, \boldsymbol{p}_j) = \mathcal{N}(\boldsymbol{t}; \boldsymbol{\mu}_\phi(\boldsymbol{p}_i, \boldsymbol{p}_j), \boldsymbol{\sigma}^2_\phi(\boldsymbol{p}_i, \boldsymbol{p}_j))$$
$$\Gamma(\boldsymbol{x}; \boldsymbol{p}, \boldsymbol{t}_i, \boldsymbol{t}_j) = \text{gamma}(\boldsymbol{x}; \boldsymbol{k}_\gamma(\boldsymbol{p}, \boldsymbol{t}_i, \boldsymbol{t}_j), \boldsymbol{\theta}_\gamma(\boldsymbol{p}, \boldsymbol{t}_i, \boldsymbol{t}_j))$$

This is the factor graph expansion for graphical model in figure 3.7.

Appendix D

Complete list of system hyper-parameters

D.1 Game-specific hyper-parameters

The following hyper-parameters have to be adjusted for every game:

Size of the vector representing player skills
This depends on complexity of the game and number of variables which the system has to predict. Games which require a wide variety of skills would require this vector to be bigger, it will also have to be bigger if number of required variables is large.

Size of the vector representing team skills
This primary depends on the size of the vector representing the player skills as it has to summarise the collective skills of the team. It also depends on number and variety of roles that can be fulfilled in the game, if all players do the same thing, this will be smaller, if different players try to complete different objectives or play different roles this would have to be bigger.

Period of time to consider during training
This hyper-parameter depends on how quickly the strategies used by players change and how frequently the rules of the game change. If the same strategies are used for long periods of time, then much older matches can be considered.

D.2 General hyper-parameters

The following hyper-parameters are related to the system in general:

1. Optimisation algorithm and parameters of chosen algorithm.
2. Stopping condition for optimisation algorithm.

These are standard machine learning parameters and adjusting them is a complicated task with many possible approaches.

Appendix E

Project Proposal