FANTASY FOOTBALL FORECASTS

an evolutionary algorithm

Candidate: Christopher Dedominici
Professor: Edgar Ernesto Sanchez Sanchez
Collaboration: startup Teamies

ABSTRACT

The aim of the thesis is to improve an already existing application used to forecast the best formation that a player (fantasy manager) can line up in the game of the fantasy football.
The work has been done in collaboration with the startup Teamies.

Teamies already had a fully working platform but the process on how they used to collect and manage the data was not so efficient and performing. So, they wanted to create the same system but in a more reliable way.
Moreover, they also wanted to increase the precision of their forecasts finding the best parameters to use in their algorithm in order to better combine and weigh players and teams statistics used to generate these forecasts.

The main tasks I had to do during my thesis were:

- To redesign the database structure and its implementation
- To create several web scrapers to collect all the information needed to calculate the forecasts from different websites
- To create the scripts to insert/update the downloaded information inside the database
- To rewrite the original forecast algorithm in C++
- To evaluate the best parameters to use to improve the forecast results using an evolutionary algorithm

The choice of using an evolutionary algorithm has been dictated from the fact that Teamies already had a proprietary algorithm to evaluate players performances and they just wanted to optimize specific parameters. So the need was to have a search

technique to find exact or approximate solutions to optimize the search problems and the most suitable solution in this case was to use this type of approach.

- The main languages and environments used to develop the application are:
- Database: MySQL relational database
- Scrapers: JavaScript scripts executed on the run-time environment Node.js with the npm package “Puppeteer” provided by Google
- Database scripts: JavaScript scripts executed on the run-time environment Node.js with the npm package “mysql”
- Forecasts algorithm: C++
- Evolutionary algorithm: MicroGP evolutionary algorithm, C++, bash scripts

FANTASY FOOTBALL FORECASTS

an evolutionary algorithm

1 INTRODUCTION

1.1 QUICK OVERVIEW 5

1.2 THE AIM OF THE THESIS 5

2 BACKGROUND THEORY

2.1 THE GAME OF FANTASY FOOTBALL 7

2.2 STATISTICS CONSIDERED 9

2.3 APPLICATION GENERAL STRUCTURE 10

3 PROPOSED APPROACH

3.1 QUICK OVERVIEW 11

3.2 THE APPLICATION IN DETAILS 14

3.2.1 Job Scheduler: Cron 14

3.2.2 Web automotion: the scrapers 15

➢ Intro 15

➢ Puppeteer overview 16

➢ Puppeteer components 17

3.2.3 The database scripts 22

3.2.4 The database 27

3.2.5 The C++ software 30

➢ Indexes 30

➢ Main classes overview 33

3.2.6 MicroGp 35

➢ Intro 35

➢ How it works 35

➢ General population settings 37

➢ Evaluation options 38

➢ Stopping conditions 39

➢ Fitness function 40

3.3 THE DATASET 41

3.4 GENERAL TESTS CONSIDERATIONS 44

3.4.1 Data training and testing 44

3.4.2 Model fitting problems 45

3.5 MICROGP PARAMETERS SETTINGS 47

4 RESULTS ANALYSIS
4.1 INTRO 49
4.2 SINGLE DAYTIME TESTS 51
4.2.1 Intro 51
4.2.2 Results 52
4.2.3 Tests conclusions 58
4.3 ALL DAYTIMES TESTS 59
4.3.1 Intro 59
4.3.2 Results 60
4.3.3 Tests conclusions 67
4.4 ALL DAYTIMES TESTS WITH FILTERS 68
4.4.1 Intro 68
4.4.2 Probability larger than 70% 69
4.4.3 Probability between 45% and 70% 76
5 CONCLUSIONS
5.1 TESTS RESULTS 80
5.2 WHAT COULD BE IMPROVED 82
5.3 NEURAL NETWORK: ANOTHER APPROACH 83
6 BIBLIOGRAPHY

INTRODUCTION

1.1 QUICK OVERVIEW

I collaborated with the startup Teamies in order to improve their software which is able to try to predict the best possible formation that a player (fantasy manager) could use to win the game of the fantasy football. The player has several footballers on his team and for every match he has to select 15 of them to play. We wanted to boost the performance of the current suggestion system to further increase the chance of the players to choose the best formation according to our software advices.

1.2 THE AIM OF THE THESIS

The thesis has two main purposes:

- to redesign all the application structure:
    - the redesign and the implementation of the database
    - the development of scrapers to collect all the information
- to rewrite and improve the forecast algorithm

The startup Teamies in the past years has used a proprietary algorithm to try to predict the best formation that a player could line up. This algorithm uses several different players and teams statistics that are combined together using various parameters that were studied and chosen by who designed the algorithm. Furthermore, these parameters were chosen by a human being instead of a software

so they probably could have had several improvements therefore we wanted to increase the correctness of this process finding more reliable parameters. To achieve this we decided to use an evolutionary algorithm.

BACKGROUND THEORY

2.1 THE GAME OF FANTASY FOOTBALL

GENERAL DESCRIPTION

Fantasy football is a game in which all the players have to assemble a team of footballers from the same championship and score points according to their real performance on the field. In this case I considered the Italian championship.

TEAMS

In the Italian fantasy football usually players have to choose (buy) at the beginning of the championship 25 players. They have a fixed amount of “fantasy money” so they have to plan carefully all the purchases (or the bids if more than one player wants the same footballer) in order to maximize the combination cost-footballer performance.
A team is usually composed by 3 goalkeepers, 8 defenders, 8 midfielders and 6 strikers.

POINTS SCORING

Points are gained or deducted depending on the players' performance. Points systems vary between games: for example Fantagazzetta scores are different from the Gazzetta ones. But in general points are awarded for some or all of the following achievements:

- playing in a match or a part of it
- scored goals (or conceded goals if the player is a goalkeeper)
- assists
- the winning of the team

As well as the above, points can be deducted for some or all of the following:

- conceded goals (if players are goalkeepers or defenders)
- yellow or red cards
- missing a penalty
- self goals
- getting substituted
- the defeat of the team

2.2 STATISTICS CONSIDERED

The main statistics we took into account on how players or their teams have performed are:

Players

- match grade
- match grade with bonus and malus (e.g.: if a player scored a goal)
- scored goals
- conceded goals
- missed penalties
- scored penalties
- rejected penalties (if a player is a goalkeeper)
- yellow cards
- red cards
- assists
- shots on goal
- recovered balls
- market value
- role

Teams

- ranking
- points
- matches played
- number of won, tie and lost matches
- scored goals
- conceded goals
- bets on their performance

2.3 APPLICATION GENERAL STRUCTURE

The application can be divided in four main components:

- **The database structure**

  The application runs on a MySQL relational database. There are several tables created to contain and connect all the statistics previously explained. All the values are inserted or updated using several scripts which read the values from several files created by various scrapers.

- **The scrapers**

  To collect all the information needed from the websites there are several scrapers. All the scrapers executions are launched at specific times using Chron, a time-based job scheduler in Unix.

- **The forecast software**

  The software (written in C++) is used to create the players forecasts and to provide the fitness function values to the evolution algorithm so that it can evaluate the quality of a specific solution.

- **The evolutionary algorithm**

  An evolutionary algorithm (MicroGP) is used to iterate many times over the several forecasts in order to find the best parameters to obtain (hopefully) the optimal solution to the problem.

PROPOSED APPROACH

3.1 QUICK OVERVIEW

The application is composed by several independent softwares which run at different times. A quick overview of them is provided in the following pages.

CRON

Cron is the software which decides what software runs and when it runs.

It can launch three different types of applications:

- the scrapers
- the database scripts
- the C++ software

THE SCRAPERS

The scrapers are written in JavaScript and executed on the run-time environment Node.js, they will collect all the players and teams statistics from the web and store them in JSON files.

THE DATABASE

The database scripts are written in JavaScript and executed on the run-time environment Node.js, they read the players and teams statistics from the JSON files created by the scrapers and insert/update them in the SQL database.

THE C++ SOFTWARE

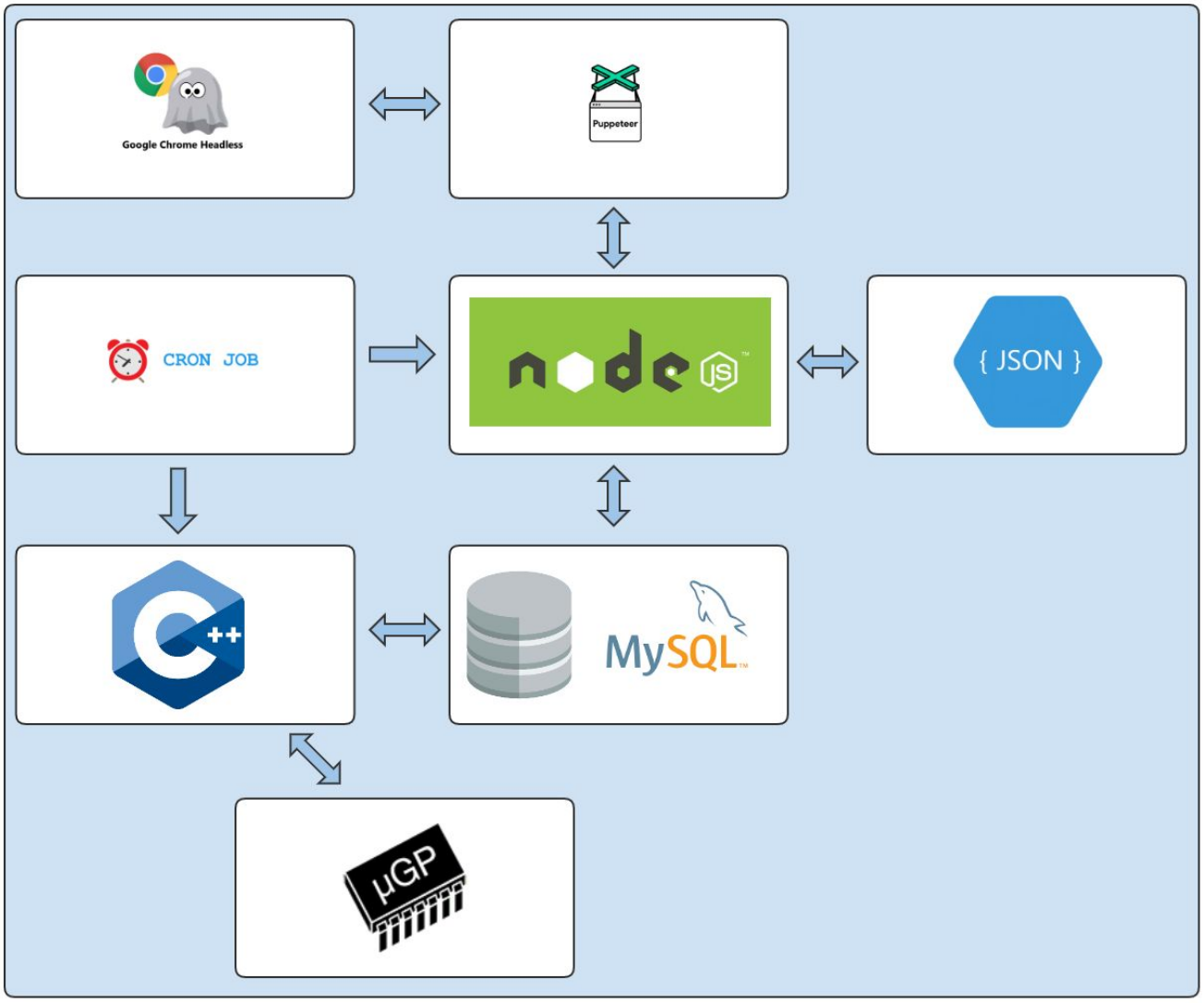
The C++ software performs 2 main tasks:

- to evaluate the best players for the next matches reading the information needed for the the forecast from the database
- to provide the quality of the solution (how far the forecasts were from the real footballers performances) to the evolution algorithm so that it can iterate to improve the forecasts.

MICROGP

MicroGP is used to run many evaluations to find the best parameters to get the best forecast about the players. It generates several random parameters in a JSON file and then using a C++ function it checks the quality of the solution.

Application general structure:



3.2 THE APPLICATION IN DETAILS

3.2.1 JOB SCHEDULER: CRON

“The software utility Cron is a time-based job scheduler in Unix. It is used to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals. It typically automates system maintenance or administration though its general-purpose nature makes it useful for things like downloading files from the Internet and downloading email at regular intervals.”[1]

As said before Crone is used to launch three different type of applications:

- the scrapers
- the database scripts
- the C++ software

In this application the scrapers and the database scripts are run every day in order to check every minimum change like player’s accidents, formation changes, etc.

3.2.2 WEB AUTOMATION: THE SCRAPERS

INTRO

To collect all the information needed various websites had to be scraped.
To achieve this goal several web-scrapers written in JavaScript with the npm packet Puppeteer provided by Google were coded. Puppeteer is a product for browser automation: when installed, it downloads a version of Chromium, which it then drives using puppeteer-core. All the information collected from the web pages are stored in different JSON files that will be used by several scripts to insert/update the information inside the database.

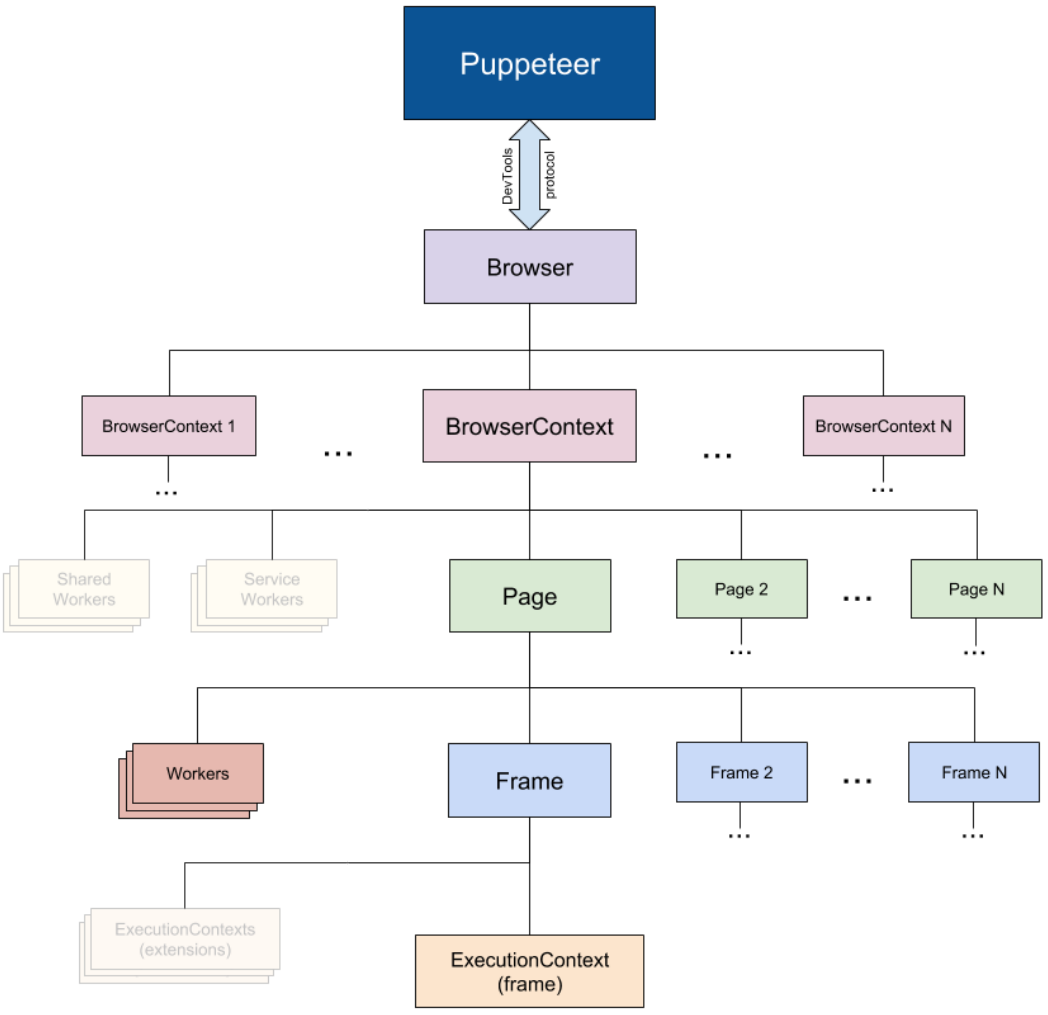
A JSON file is a file that stores simple data structures and objects in JavaScript Object Notation (JSON) format, which is a standard data interchange format. It is primarily used for transmitting data between a web application and a server. JSON files are lightweight, text-based, human-readable, and can be edited using a text editor.

PUPPETEER OVERVIEW

The API provided by the Puppeteer npm package are extremely powerful and easy to use. They give the possibility to manipulate the headless browser like a real human being so, for this reason, in the following pages the main features used in this application are reported to give a general idea about how they work.

The official Puppeteer documentation is used, you can find it at the following link: https://github.com/GoogleChrome/puppeteer/blob/master/docs/api.md.[2]

The Puppeteer API is hierarchical and mirrors the browser structure.



The main components of Puppeteer are:

- **Browser** instance can own multiple browser contexts
- **BrowserContext** instance defines a browsing session and can own multiple pages
- **Page** has at least one frame: main frame. There might be other frames created by iframe or frame tags
- **Frame** has at least one execution context - the default execution context - where the frame's JavaScript is executed
- **Worker** has a single execution context and facilitates interacting with **WebWorkers**

PUPPETEER COMPONENTS

class: Puppeteer

Puppeteer module provides a method to launch a Chromium instance. The following is a typical example of using Puppeteer to drive automation.

Example: how to launch Puppeteer.

```
const puppeteer = require('puppeteer');
puppeteer.launch().then(async browser => {
        const page = await browser.newPage();
         await page.goto('https://www.google.com');
         // other actions...
         await browser.close();
});
```

puppeteer.connect(options)

- options <Object>
    - browserWSEndpoint <string> a browser websocket endpoint to connect to.
    - ignoreHTTPSErrors <boolean> Whether to ignore HTTPS errors during navigation. Defaults to false.
    - defaultViewport <?Object> Sets a consistent viewport for each page. Defaults to an 800x600 viewport. null disables the default viewport.
        - width <number> page width in pixels.
        - height <number> page height in pixels.
        - deviceScaleFactor <number> Specify device scale factor (can be thought of as dpr). Defaults to 1.
        - isMobile <boolean> Whether the meta viewport tag is taken into account. Defaults to false.
        - hasTouch<boolean> Specifies if viewport supports touch events. Defaults to false.
        - isLandscape <boolean> Specifies if viewport is in landscape mode. Defaults to false.
        - slowMo <number> Slows down Puppeteer operations by the specified amount of milliseconds. Useful so that you can see what is going on.
- returns: <Promise<Browser>>

class: Browser

A Browser is created when Puppeteer connects to a Chromium instance, either through puppeteer.launch or puppeteer.connect.

Example: using a Browser to create a Page.

```
const puppeteer = require('puppeteer');
puppeteer.launch().then(async browser => {
        const page = await browser.newPage();
         await page.goto('https://example.com');
         await browser.close();
});
```

class: BrowserContext

BrowserContexts provide a way to operate multiple independent browser sessions. When a browser is launched, it has a single BrowserContext used by default. The method browser.newPage() creates a page in the default browser context.

If a page opens another page, e.g. with a window.open call, the popup will belong to the parent page's browser context.

Puppeteer allows creation of "incognito" browser contexts with browser.createIncognitoBrowserContext() method. "Incognito" browser contexts don't write any browsing data to disk.

Example: create a new incognito browser context.

```
const context = await browser.createIncognitoBrowserContext();
// Create a new page inside context.
const page = await context.newPage();
// ... do stuff with page ...
await page.goto('https://example.com');
```

```
// Dispose context once it's no longer needed.
await context.close();
```

class: Page

Page provides methods to interact with a single tab or extension background page in Chromium. One Browser instance might have multiple Page instances.

Example: creates a page, navigates it to a URL, and then saves a screenshot.

```
const puppeteer = require('puppeteer');
puppeteer.launch().then(async browser => {
        const page = await browser.newPage();
        await page.goto('https://example.com');
        await page.screenshot({path: 'screenshot.png'});
        await browser.close();
});
```

The Page class emits various events (described below) which can be handled using any of Node's native EventEmitter methods, such as on, once or removeListener.

Example: log a message for a single page load event.

```
page.once('load', () => console.log('Page loaded!'));
```

class: Frame

At every point of time, page exposes its current frame tree via the page.mainFrame() and frame.childFrames() methods.

class: Worker

The Worker class represents a WebWorker[1]. The events workercreated and workerdestroyed are emitted on the page object to signal the worker lifecycle.

Example: create and destroy workers.

```
page.on('workercreated', worker => console.log('Worker created: ' + worker.url()));
page.on('workerdestroyed', worker => console.log('Worker destroyed: ' +
worker.url()));
console.log('Current workers:');
for (const worker of page.workers())
        console.log('  ' + worker.url());
```

[1]WEB WORKER: A Web Workers makes it possible to run a script operation in a background thread separate from the main execution thread of a web application. The advantage of this is that laborious processing can be performed in a separate thread, allowing the main (usually the UI) thread to run without being blocked/slowed down.

3.2.3 THE DATABASE SCRIPTS

The scripts are written in JavaScript for the run-time environment Node.js and use the npm package “mysql” which makes it possible to insert or update information stored in the JSON file (created by the web scrapers) into the database.

Through this npm package many operations can be performed, the most important in this case are:

- establishing connections
- set connection options
- terminating connections
- performing queries
- error handling

The API provided through this npm package are very powerful and extremely easy to us so, for these reason, in the following pages the main features used in this application are reported in order to give a general idea on how they work. The official mysql documentation is used, you can find it at the following link: https://www.npmjs.com/package/mysql.[3]

Establishing connections.

```
let mysql      = require('mysql');
let connection = mysql.createConnection({
      host     : 'localhost',
      user     : 'me',
      password : 'secret',
      database : 'my_db'
});
connection.connect(function(err) {
      if (err) {
            console.error('error connecting: ' + err.stack);
            return;
      }
      console.log('connected as id ' + connection.threadId);
});
```

However, a connection can also be implicitly established by invoking a query:

```
let mysql      = require('mysql');
let connection = mysql.createConnection(...);
connection.query('SELECT 1', function (error, results, fields) {
      if (error) throw error;
      // connected!
});
```

Connection options

There are several connection options that can be set while creating a new connection, some of them are:

- host: The hostname of the database you are connecting to. (Default: localhost).
- port: The port number to connect to. (Default: 3306).
- localAddress: The source IP address to use for TCP connection. (Optional).
- user: The MySQL user to authenticate as.
- password: The password of that MySQL user.
- database: Name of the database to use for this connection (Optional).
- timezone: The timezone configured on the MySQL server. This is used to type cast server date/time values to JavaScript Date object and vice versa. This can be 'local', 'Z', or an offset in the form +HH:MM or -HH:MM. (Default: 'local').
- connectTimeout: The milliseconds before a timeout occurs during the initial connection to the MySQL server. (Default: 10000).
- ssl: object with ssl parameters or a string containing name of ssl profile.

Terminating connections

There are two ways to end a connection. Terminating a connection gracefully is done by calling the end() method:

```
connection.end(function(err) {
        // The connection is terminated now
});
```

This will make sure all previously enqueued queries are still before sending a COM_QUIT packet to the MySQL server. If a fatal error occurs before the COM_QUIT packet can be sent, an err argument will be provided to the callback, but the connection will be terminated regardless of that.

An alternative way to end the connection is to call the destroy() method. This will

cause an immediate termination of the underlying socket. Additionally destroy() guarantees that no more events or callbacks will be triggered for the connection.

```
connection.destroy();
```

Unlike end() the destroy() method does not take a callback argument.

Performing queries

The most basic way to perform a query is to call the .query() method on an object (like a Connection, Pool, or PoolNamespace instance).

The simplest form of .query() is .query(sqlString, callback), where a SQL string is the first argument and the second is a callback:

```
connection.query('SELECT * FROM `books` WHERE `author` = "David"',
function (error, results, fields) {
    // error will be an Error if one occurred during the query
    // results will contain the results of the query
    // fields will contain information about the returned results fields (if any)
});
```

The second form .query(sqlString, values, callback) comes when using placeholder values (see escaping query values):

```
connection.query('SELECT * FROM `books` WHERE `author` = ?', ['David'],
function (error, results, fields) {
    // error will be an Error if one occurred during the query
    // results will contain the results of the query
    // fields will contain information about the returned results fields (if any)
});
```

The scripts use the second form:

- the sql query is created with the information needed (name of the table, which information are wanted) and then stored inside a string passed as first argument.
- then an array is created and all the value that will replace the “?” character in the string are stored inside it; then it will be passed as the second argument.

Error handling

This module comes with a consistent approach to error handling in order to write solid applications.

Most errors created by this module are instances of the JavaScript Error object. Additionally they typically come with two extra properties:

- err.code: Either a MySQL server error (e.g. 'ER_ACCESS_DENIED_ERROR'), a Node.js error (e.g. 'ECONNREFUSED') or an internal error (e.g. 'PROTOCOL_CONNECTION_LOST').
- err.fatal: Boolean, indicating if this error is terminal to the connection object. If the error is not from a MySQL protocol operation, this properly will not be defined.
- err.sql: String, contains the full SQL of the failed query. This can be useful when using a higher level interface like an ORM that is generating the queries.
- err.sqlState: String, contains the five-character SQLSTATE value. Only populated from MySQL server error.
- err.sqlMessage: String, contains the message string that provides a textual description of the error. Only populated from MySQL server error.

3.2.4 THE DATABASE

The database used in support to the application is a MySQL relational database. Both to design and create the database MySQL Workbench has been used. It is a visual database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environment for the MySQL database system.

The database contains several tables in which all the statistics of the players and their teams are stored with their relative relations.
The tables are:

- **championship**
  contains all the championships available in the application (only the Italian one for the thesis purpose)
- **daytime**
  contains the numbers of all the daytimes played in a specific championship
- **real_team**
  contains the information of the team
- **teams_match**
  contains all the played matches with the relative most important information like teams ids, match results and the value of the bets
- **real_teams_statistics**
  contains all the statistics of the teams update daily. For example the ranking of the teams, their total points, the total number of their scored or conceded goals, the number of win/lose/tie matches

- **membership**

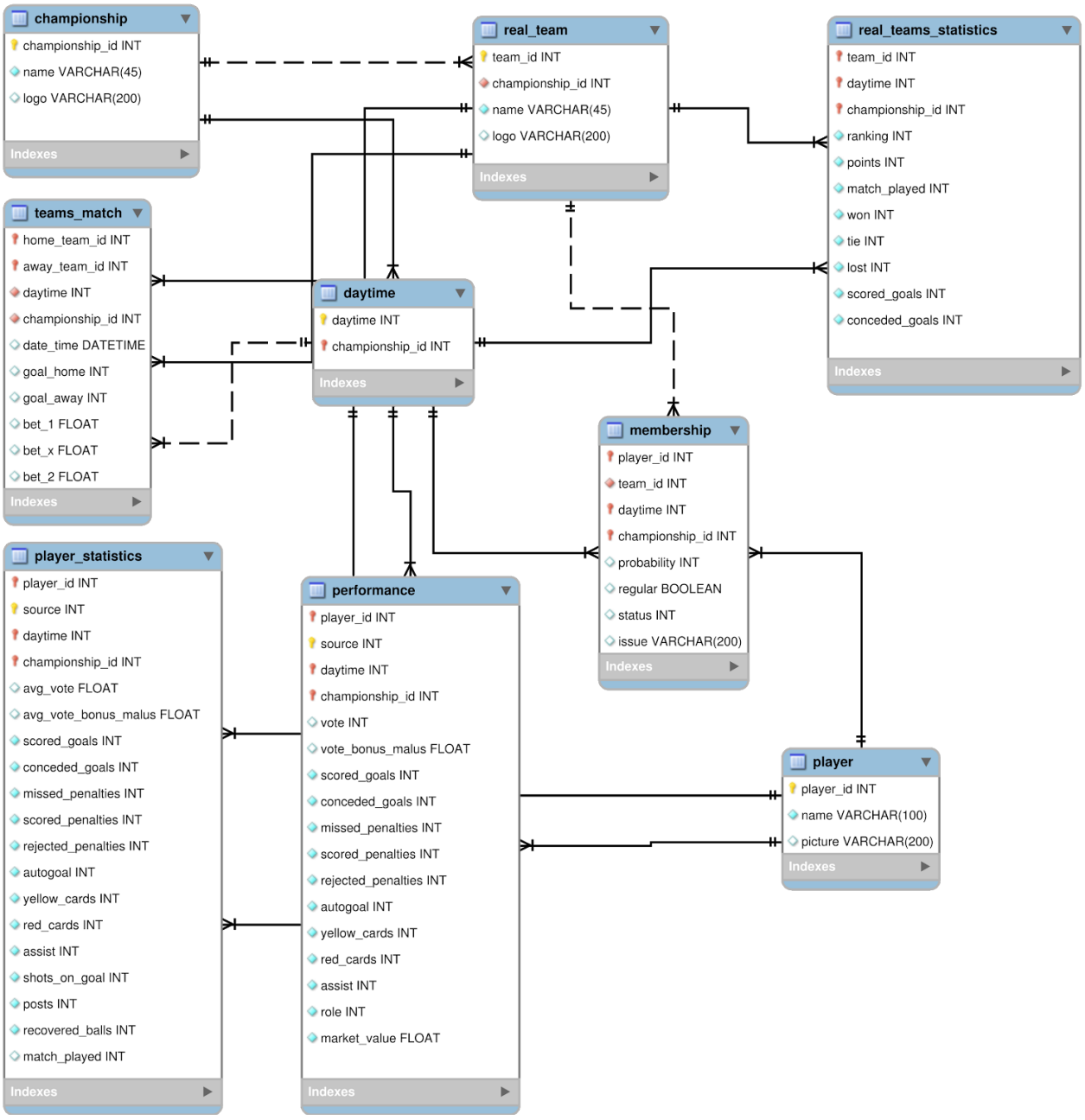
  contains the information relative to which player plays in which team in which daytime with his probability to actually enter in the field
- **player**

  contains all the information about the players
- **performance**

  contains the information relative to every single match for every player. For example his grade, his bonus-malus grade, how many goals he scored (or conceded if he is a goalkeeper), his market value, etc.
- **player_statistics**

  contains all the statistics of the players, update daily. For example his average grade and average bonus-malus grade during the year, the total number of his scored goals (or conceded if he is a goalkeeper), his total number of played matches, etc.

Database design:



3.2.5 THE C++ SOFTWARE

INTRO

The startup Teamies had several excel worksheets with all the statistics saved on and, always on the same worksheets, they used specific math formulas to evaluate players performance. Of course this approach was not very efficient and optimized so we decided to move the data elaboration process in a C++ program and the data storage into the database (see database section).

To create the forecasts there are many indexes that combined together will give the player evaluation.

The software strategy is carried out in the following order:

1. evaluate teams indexes
2. evaluate players indexes considering the current player statistics combined with his team indexes and his opponent team indexes
3. evaluate the prediction using the players indexes

INDEXES

Team indexes

- **[IS] = team index**

  Index evaluated considering the current team ranking

- **[IGSF] = scored and conceded goals index**

  Index evaluated considering the total scored goals and the total conceded goals of the team

- **[IA] = strike index**

  Index evaluated considering the total sum of scored goals of a team until the last played match

- **[ID] = defense index**

  Index evaluated considering the total sum of conceded goals of a team until the last played match

- **[ITOT] = total index**

  Index evaluated combining the **IS** and the **IGSF** indexes

- **[IAC] = strike home index**

  Index evaluated considering the total sum of scored goals of a team until the last played match when playing home

- **[IDC] = defense home index**

  Index evaluated considering the total sum of conceded goals of a team until the last played match when playing home

- **[IAF] = strike away index**

  Index evaluated considering the total sum of scored goals of a team until the last played match when playing away

- **[IDF] = defense away index**

  Index evaluated considering the total sum of conceded goal of a team until the last played match when playing away

- **[IF] performance index**

  Index evaluated considering how the team has performed in the previous matches
- **[IB] = bet index**

  Index evaluated considering the value of the bets placed on the team

Player indexes:

- **[IT] frequency index**

  Index evaluated considering how many matches a player has played
- **[IG] player index**

  Index evaluated combining the past players' grades, grades with bonus and malus and the **IT** index
- **[ICF] match away or match home index**

  Index evaluated considering if the team plays at home or away
- **[ISc] match index**

  Index evaluated considering the team's indexes **IAC** or **IAF** and **IDC** or **IDF** used in different ways accordingly if the player's team plays at home or away
- **[IFo] performance index**

  Index evaluated considering the current player's team **IF** index and the current player's opponent team's **IF** index
- **[BW_fix] best week fixed index**

  Index evaluated considering the current player's team indexes (**IG**, **IB**) and his indexes **ISc** and **IFo**
- **[BW] best week index**

  Index evaluated considering the player's probability to play and the player's index **BW_fix.** This index will be the forecast on how the player will perform.

MAIN CLASSES OVERVIEW

The main classes of the software are:

- **Team**

  Contains all the team information (and the methods to get/set them) like team id, team name and more importantly it contains the float array of all the values of its indexes.

- **Player**

  Contains all the player's information (and the methods to get/set them) like player id, player name, his team, his probability to play and more importantly it contains the float array of all the values of its indexes. Moreover it also contains the error about his prediction (the software compares its prediction with the actual player's performance to see how close they are).

- **Various Classes: performances indexes**

  All the players and teams indexes have their specific classes but all of them are inherited from the parent class "Index" which provides default standard methods like "compute()" (used to calculate the value of the index).

- **Comparator**

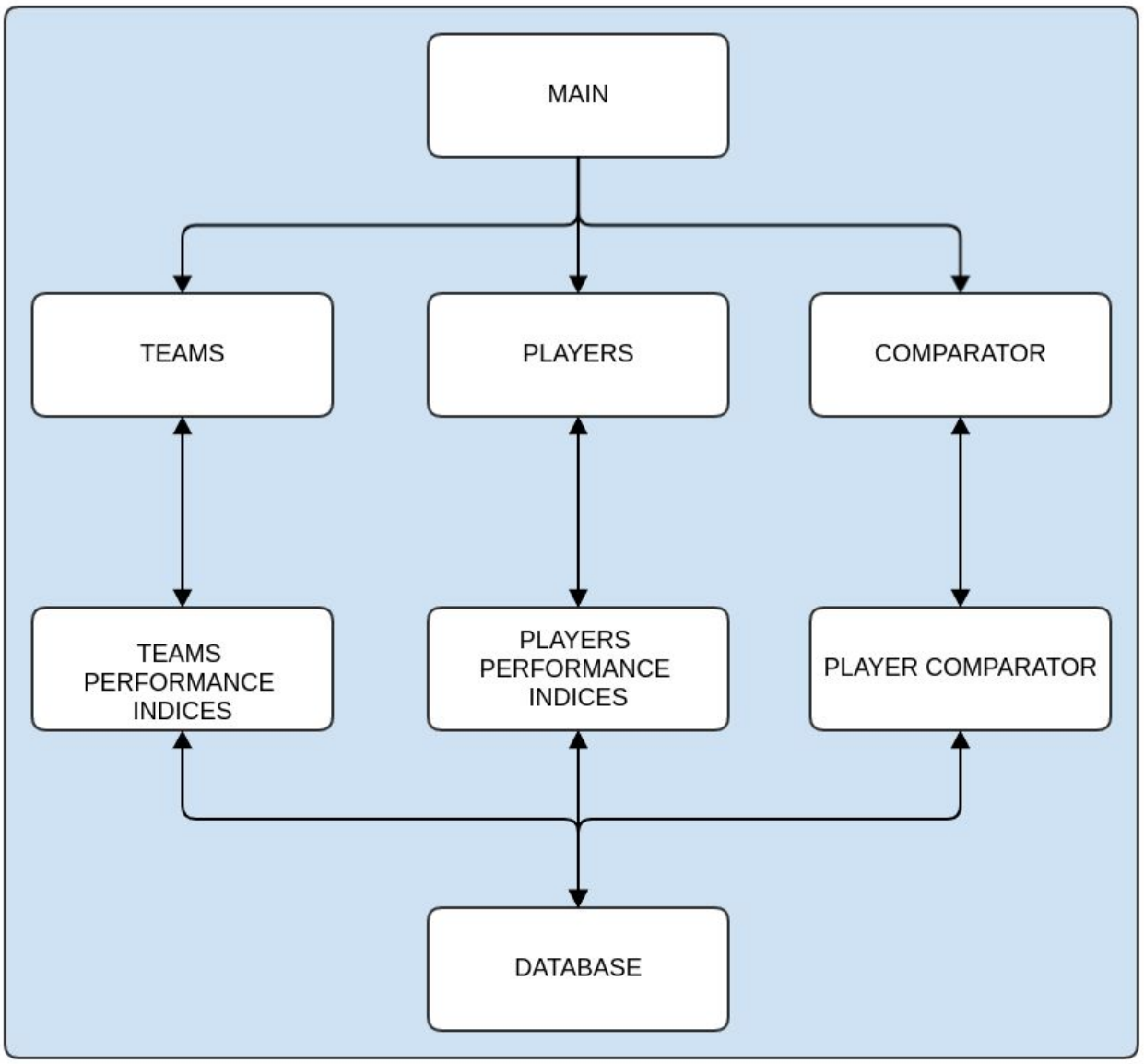
  The comparator is used to return the total forecast error of the software prediction (considering all the players errors) in order to understand how good the software predicts the best players.

  The single player error is retrieved from the class "PlayerComparator".

- **PlayerComparator**

  This class is used to calculate the error between the predicted performance of the player and his real performance.

C++ software general structure:



3.2.6 MICROGP

INTRO

MicroGP is an optimizer used to find the optimal solution of difficult problems. Given a task, it first generates a set of random solutions, then iteratively refines and try to improve them. Analysing the results of the fitness function it can explore the solution space searching for the best solutions and hopefully find the optimal one.

MicroGP is an evolutionary algorithm, it generates at every iteration a new population of individuals combining the results of the previous generations using specific strategies. Both the best and the worst solutions at every step are taken into account, of course the more efficient solutions will have a better chance to be chosen in order to reproduce.

HOW IT WORKS

The main steps in order to run the optimization are:

- **bash main script**

  A main bash script controls the execution of the MicroGP. As it will be discussed later, the forecasts are made dividing the footballers into their roles and probabilities to play. Therefore these parameters, along with others, have to be passed to the script so it will be able to launch the specific tests through MicroGP. In addition it will also automatically generates several CSV files containing the statistics results and a JSON file with the best parameters found.

- **MicroGP**

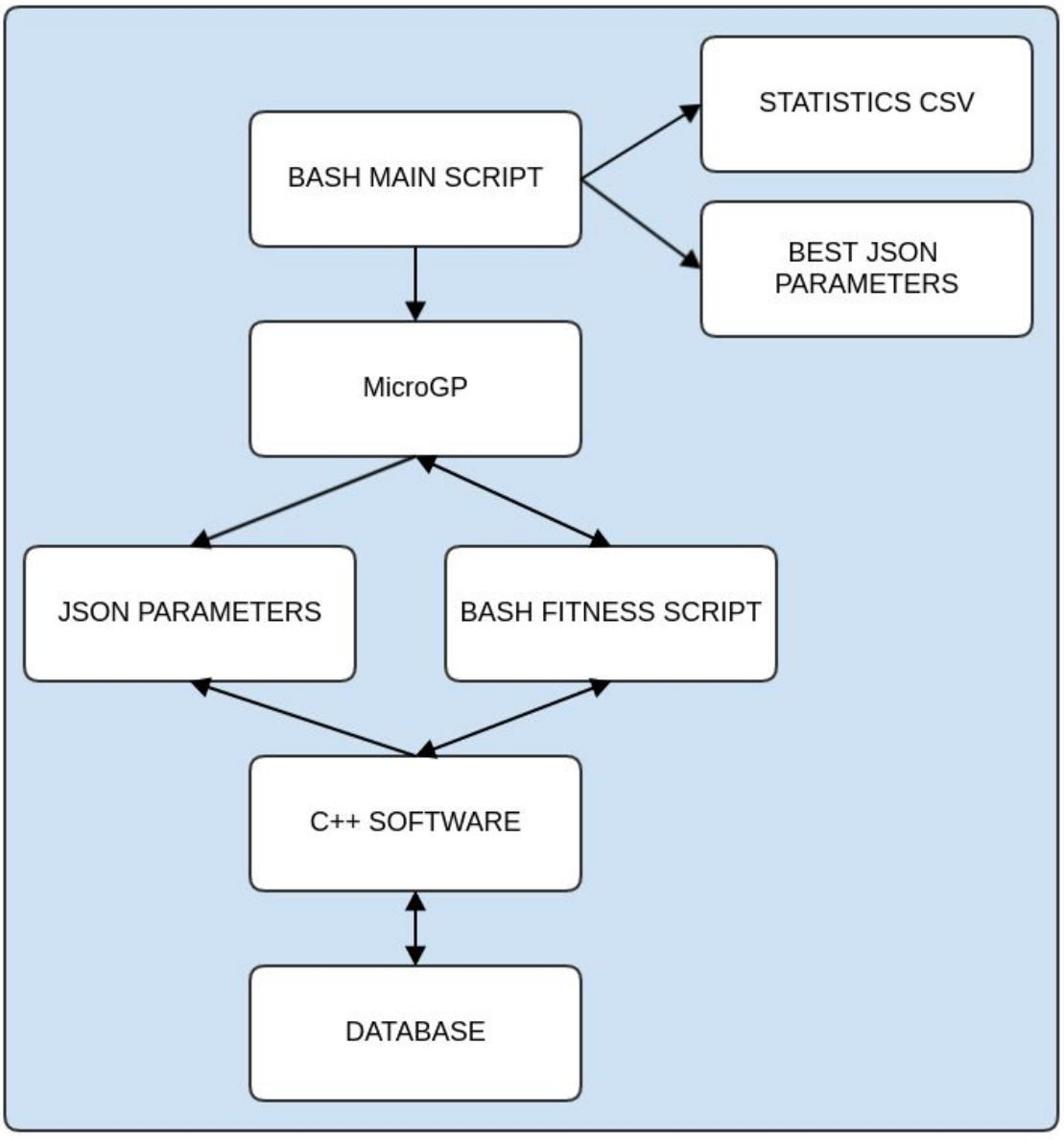
  As explained in the previous point MicroGP is launched by a bash script. MicroGP will generate a JSON file with the parameters and after that to evaluate the quality of this solution it will run a fitness function (see next point).

The iterative process, until a stop condition is reached, is to:

1. generate the JSON parameters files
2. run the fitness function to evaluate the solution
3. go to point 1 until a stop condition is reached

- **bash fitness evaluation script**

  This script is used by MicroGP to evaluate the quality of the current solution. It launches the C++ software to receive the total players' errors values (standard deviation and average error).



In the following pages the main configuration settings used in this application are reported using the official MicroGP documentation at the following link: http://ugp3.sourceforge.net.[4]

GENERAL POPULATION OPTIONS

- **mu:** is the size of the population. At the beginning and at the end of each generation, there will be exactly mu individuals.
- **nu**: is the size of the initial randomly-created population. After the first generation, the number of individuals will be reduced (or incremented) taking into account **mu**.
- **lambda**: is the number of genetic operators that will be applied at each step. Note that each genetic operator may create more than one individual, so the offspring size at each generation is usually bigger than lambda.
- **sigma**: regulates the strength of the genetic operators. As a rule of thumb, if sigma is high (> 0.5), the offspring of an individual will be more different from the original individual than when sigma is low (< 0.5). In our case sigma can be self-adapted to be higher in the beginning of the evolutionary process (when exploration of the search space is needed), and lower at the end (when exploitation is better). See inertia, below.
- **inertia**: is used to tune the self-adapting mechanism. The idea of self-adaptation is that the algorithm itself modifies its behavior as the evolution goes on, trying to always have the best performance given the current situation. Inertia, like real-world inertia, represents the resistance of the system to the self-adapting push towards new values. Inertia is a number between 0 and 1. The bigger it is, the slower the changes are. As a rule of thumb, inertia should always be around 0.9.

EVALUATION OPTIONS

- **fitnessParameters**: is the number of fitness parameter that MicroGP expects for this problem. Fitness value can be an array of floating point numbers.In our case they will be evaluated hierarchically, with the first one more important than the second.
- **cloneScalingFactor**: is a way to manage "clones", individuals that are exact (genotypical) copies of individuals already evaluated. Basically, the fitness of each clone will be multiplied by cloneScalingFactor (which is usually lower than 1) times the number of clones already existing in the population. So, if cloneScalingFactor=0.5, the original individual will retain its complete fitness, its first clone will have that fitness multiplied by 0.5, the second clone will have (0.5*0.5) = 0.25 times the original fitness, the third clone will have (0.5*0.5*0.5) = 0.125 times the original fitness, and so on.

STOPPING CONDITIONS

- **maximumFitness**: indicates that MicroGP will stop when it will find an individual having a fitness value equal to or higher than maximumFitness.
- **maximumSteadyStateGenerations**: is a stagnation condition. If the best fitness value in the population does not improve for the value specified with this option, MicroGP terminates.
- **maximumGenerations**: simply means that the execution of MicroGP will be terminated after a set number of generations.
- **maximumEvaluations**: implies that the evolution will be stopped at the end of the generation where a certain number of total calls to the evaluator have been reached. This is not equivalent to maximumGeneration, because MicroGP has not a fixed offspring size produced at each generation (see above for more details). Useful if you need to compare performance against other approaches.
- **maximumTime**: specify the maximum time (wall clock) MicroGP can be allowed to run. If the maximumTime tag appears, it must be specified in hours, minutes and seconds: none is mandatory, but at least some time must be specified. E.g., <maximumTime minutes="5" seconds="30"/>. When MicroGP is stopped, it stores the elapsed time. If it is restarted, the elapsed time is displayed and used for calculating the limit that will include all previous runs.

FITNESS FUNCTION

The error between the players predictions and the real players' performances is evaluated for every daytime accordingly to the following formulas.

Variable needed

The following variables are used to calculate every single player error.

- GRADER = real match player grade
- GRADEbmR = real match player grade with bonus and malus
- GRADEP = previous match player's grade
- GRADEbmP = previous match player's grade with bonus and malus
- PP = player's prediction value

Error calculation

For every player the idea is to calculate the difference between the real result and the predicted one.

- geometric average of GRADER and GRADEbmR (GAGGR)
  GAGGR = (GRADER ✕ GRADEbmR)^½
- geometric average of GRADEP and GRADEbmP (GAGG)
  GAGG = (GRADEP ✕ GRADEbmP) ^ ½
- expected match value (EMV)
  EMV = GAGG ✕ PP
- prevision error (PE) = GAGGR - EMV

For every daytime and for each role (goalkeeper, defender, midfielder and striker) will be calculated the total players' average error and the standard deviation. Obviously the more these two values will be close to zero the better the prediction will be. The evolution algorithm will receive these two results as parameters to improve the quality of the solution.

3.3 THE DATASET

The dataset used in this case is composed by all the teams and all the players in the Italian football championship. For every match the number of total playing teams is 20 while the total number of playing footballers for every single team is 11 plus an average of 3 possible substitutes. So usually we have between 220 and 280 total players to analyze per daytime (range calculation: 11*20 - 14*20).
After this consideration we can quickly estimate the total number of players and teams for all the matches in the championship: there are 38 total matches during the whole year so we have a total amount of 760 statistics for the teams and around 8360-10640 statistics for the players (the range varies depending on the substitutions). Unfortunately in this case the statistics in the database for the championship 2017-2018 (previous championship, not the current one), started from the 6° daytime so our dataset is restricted to only 33 matches (6-38).
So the real dataset has a total of 660 teams statistics and between 7260 and 9240 players statistics.
The dataset is then evaluated splitting the players by role which are goalkeepers, defenders, midfielders and strikers. In this case the number of players per match became quite different depending on the role. The reason is, obviously, that in a team there are, for example, more defenders than goalkeepers playing in a match.
To give a quick and rough example on how the size of the dataset changes accordingly to the role we can imagine an example scenario using a quite classic formation (4-4-2) with and without a possible player substitution for every role.

Possible scenario using the classic 4-4-2 formation:

1 goalkeeper per team

- full championship (38 matches):
    - no substitution:

      1 ✕ 20 teams ✕ 38 matches = 760
    - supposing 1 substitution

      2 ✕ 20 teams ✕ 38 matches = 1520
- our available dataset (33 matches):
    - no substitution:

      1 ✕ 20 teams ✕ 33 matches = 660
    - supposing 1 substitution

      2 ✕ 20 teams ✕ 33 matches = 1320

4 defenders per team

- full championship:
    - no substitution:

      4 ✕ 20 teams ✕ 38 matches = 3040
    - supposing 1 substitution

      5 ✕ 20 teams ✕ 38 matches = 3800
- our available dataset (33 matches):
    - no substitution:

      4 ✕ 20 teams ✕ 33 matches = 2640
    - supposing 1 substitution

      5 ✕ 20 teams ✕ 33 matches = 3300

4 midfielders per team:

- full championship:
    - no substitution:

      4 ✕ 20 teams ✕ 38 matches = 3040
    - supposing 1 substitution

      5 ✕ 20 teams ✕ 38 matches = 3800
- our available dataset (33 matches):
    - no substitution:

      4 ✕ 20 teams ✕ 33 matches = 2640
    - supposing 1 substitution

      5 ✕ 20 teams ✕ 33 matches = 3300

2 strikers per team

- full championship:
    - no substitution:

      2 ✕ 20 teams ✕ 38 matches = 1520
    - supposing 1 substitution

      3 ✕ 20 teams ✕ 38 matches = 2280
- our available dataset (33 matches):
    - no substitution:

      2 ✕ 20 teams ✕ 33 matches = 1320
    - supposing 1 substitution

      3 ✕ 20 teams ✕ 33 matches = 1980

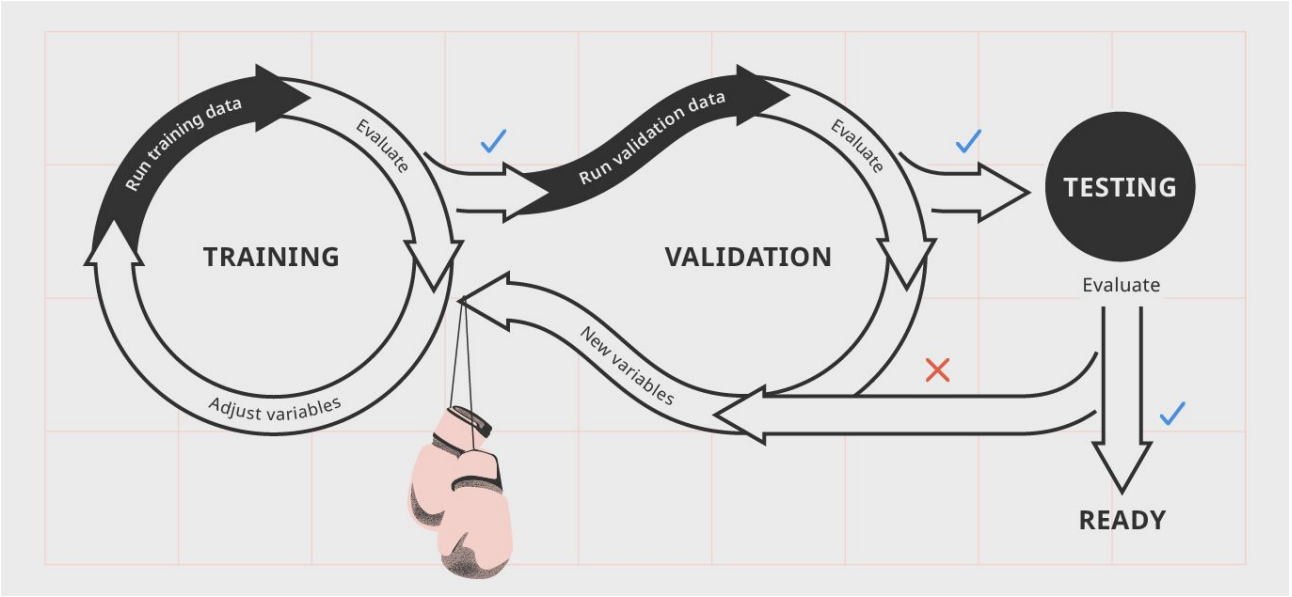
3.4 GENERAL TEST CONSIDERATIONS

3.4.1 DATA TRAINING AND DATA TESTING

The dataset has been split into two main parts, the training set (75%) and the testing set (25%). The training data includes both input and output and is the information used to "train" MicroGP to understand, evaluating the quality of the solutions, which are the finest parameters to obtain the best solution.

The testing data includes only the input data and not the expected output so it is possible to estimate how good our model properties are. Usually, as a rule of thumb, only the 10-25% of the total data is used as data testing. Of course it is important not to have any overlapping between the training data and the testing one otherwise the accuracy results will be wrong.

The reason is that any algorithm that is tested on the same data used to train it will perform in an optimum way because it already knew the correct output for that information.



[image and information] [5]

3.4.2 MODEL FITTING PROBLEMS

In statistics a fit refers to how well you approximate a target function.
Our goal is to find parameters for our predictions that “fit” to provide the best results; to estimate how god our model is we have to compare the results with the optimal ones. In doing so we have to be aware that two main situations exist where the solution could have problems: overfitting and underfitting.

Overfitting

Overfitting refers to a model that generalizes the training data too well. That happens when the model learns all the details and the noises in the training dataset, we could say that it becomes too good to approximate our training data. The problem in doing this is that this approximation fits perfectly during the training but it cannot be extended to new datasets, so it is not possible to generalize.

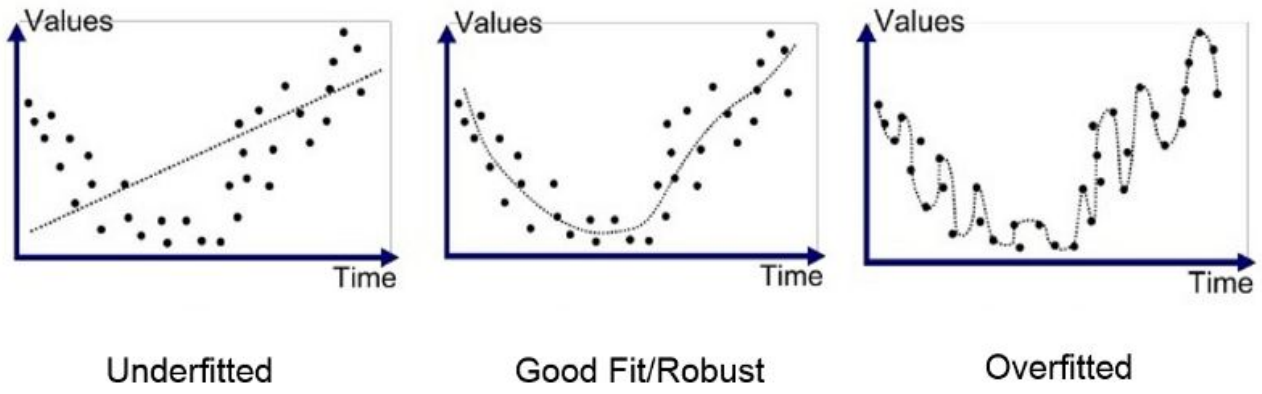
Underfitting

Underfitting refers to a model that can neither model the training dataset nor generalize the new data. It will have poor performance both in training and in testing. Usually underfitting is easy to detect compared to overfitting because the forecasts have very bad results.

Good fit

Ideally the concept is to have a model between the underfitting and the overfitting, a model able to perfectly generalize the fit function. The goal is very simple to explain but quite complex to reach. To better understand when the results are getting close to a good fit it is possible, for example, to look at the performance of our model on both the training and testing data while we are running it.
At the beginning the training and testing errors will be quite high (we are in an underfitting situation) while as soon as both errors start to decrease we can

understand that we are moving towards a good fit. But at a certain point if we continue to reduce our training errors we can see that our test errors start to increase. This is the clear sign of an overfitting model, in fact our algorithm is learning all the irrelevant details and noises in the training set.



[image and information] [6]

3.5 MICROGP PARAMETERS SETTINGS

In the the application description, under the section relative on MicroGP, the parameters that we were able to set in order to achieve our goal were introduced. Here it is possible to read which values we decide to attribute to them. The parameters that are not considered were left in the default settings.

Parameters configuration:

- **fitnessParameters value = 2**

  This is the number of dimensions of the fitness, in our case it is equal to 2. We decided to improve both the error average and the standard deviation of all the players.
- **maximumFitness value = 1000 1000**

  We wanted to reduce the errors as close as possible to zero. To evaluate how close we were to our goal, considering that MicroGP cannot receive negative values, our approach was to set a maximum value (something so big that the error could never be bigger) and then subtract from it our prediction error. The more the result was close to the “maximumFitness” the better the fitness.
- **maximumSteadyStateGenerations value = 5**

  If the best fitness value does not change for maximumSteadyStateGenerations generations, the evolution stops. In our case, after some experiments, we experienced that after 5 cycles of equivalent generations very rarely MicroGP was able to improve; for this reason we set the value to 5.
- **mu value = 250**

  The maximum size of the population was set to 250 considering that we had to evaluate around 40 different parameters.

- **nu value = 700**

  Also the initial size of the population was set considering the number of parameters that we had to evaluate (around 40 parameters). Moreover we wanted to have a quite large number of children at the beginning of the simulation in order to guarantee a big variance among the individuals.
- **lambda value = 150**

  In order to guarantee a strong variance of the the numbers of genetic operators applied at every step of the evolution we decided to set this parameters to 150.
- **inertia value = 0.9 and sigma value = 0.9**

  Considering that we wanted to have a high variance in the population but only at the beginning and then slowly reduce this variance in order to converge on the best individuals we decided to set both the inertia and sigma at 0,9.

RESULTS ANALYSIS

4.1 INTRO

We decided to run several tests to well understand the right strategy to use in order to reach a solution able to improve the original parameters of the algorithm.
As anticipated before the analysis have been carried out splitting the dataset in four different parts accordingly to the players' roles so these divisions will be implied in the next pages. Of course the whole dataset has been splitted also in two additional parts: a training one and a test one.

The main approaches used are:

- to run tests splitting the dataset for every single championship daytime (20 matches per day).
  E.g: run tests only on championship daytime 6, then only on the daytime 7, etc.
- to run tests on all the championship daytimes together, from daytime 6 to daytime 38, with of course the exceptions for the test datasets.
- to combine the previous tests with more "restrictive categories" like splitting the datasets accordingly to the players' probability to play.
  E.g: test in one group only striker footballers with more than 70% of probability to play and test in another group striker footballers but with the probability to play between 45% and 70%.

For all the tests carried out the errors considered in the fitness function were the average error for all the players and the standard deviation for all the players too (see section about C++ software for more detailed info).

For space reason when explaining the results and and when showing the relative charts not all the roles or daytimes are reported. The reason is that, usually, the experiments have all the same trends independently from the role so it would be useless to repeat the same analysis on different groups. Of course when a role will act strangely in comparison with the others a more profound discussion would be present.

4.2 SINGLE DAYTIME TESTS

4.2.1 INTRO

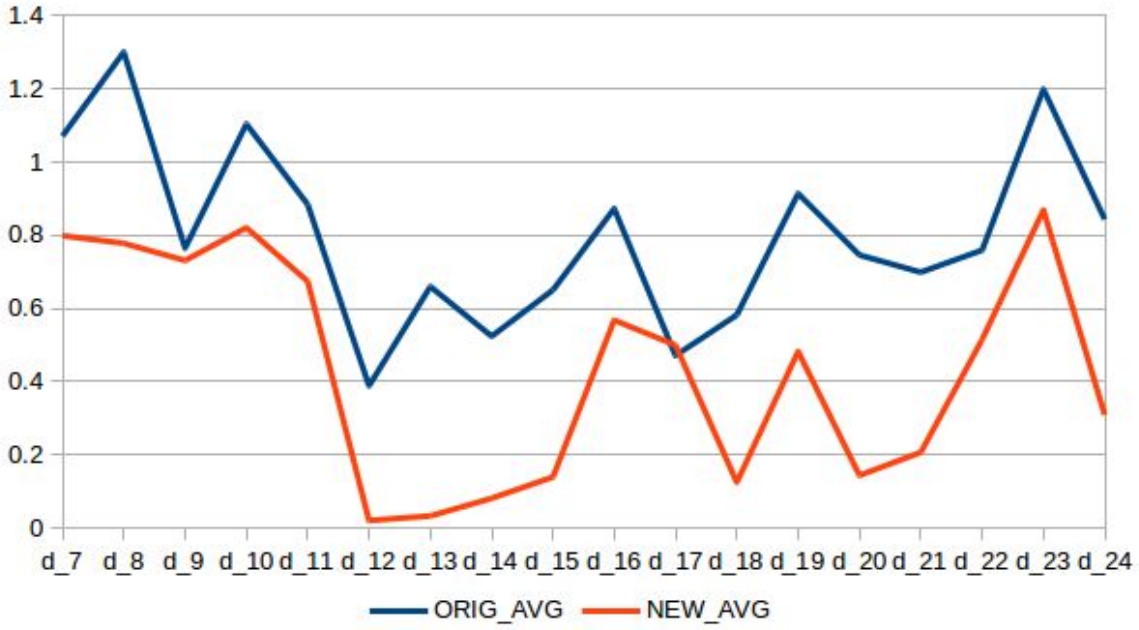
In performing single daytime tests we wanted to analyze how the model would have performed on all the matches in one daytime championship at a time for all the daytimes. Before starting I was aware that this approach could have ended up in an overfitting model in fact the parameters found, even if extremely good, could not be used on different championship daytimes or on the same championship daytime in the following years. The reason is that the model would be incredibly optimum only for the current data and not for others because it would have learned too many details and, moreover, every daytime has various differences from the others; in other words so precise parameters could not be applied in all these possible different scenarios. So the goal I wanted to achieve running these tests was not to find values to use for every daytime but to have a general idea on how the parameters vary in different daytimes and in different roles, to see if some of them are always in a specific range or change between the beginning of the championship and the end of it.

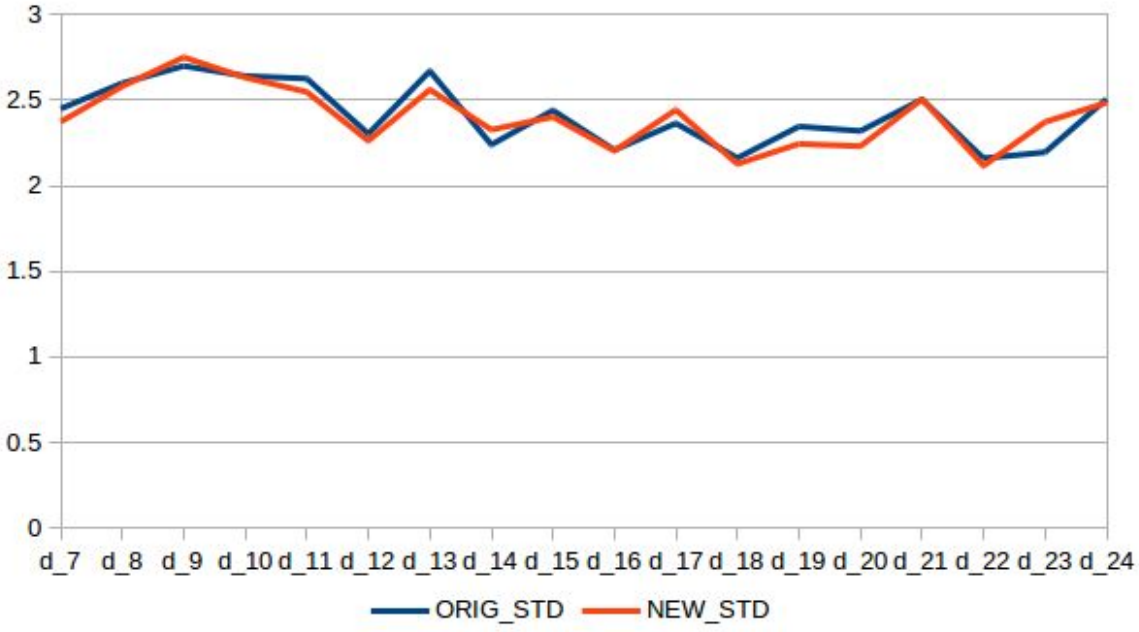
4.2.2 RESULTS

Mean error and standard deviation

After running some tests on a portion of the daytimes I could observe a significant general improvement for the error average and rather good improvements in the standard deviation error.

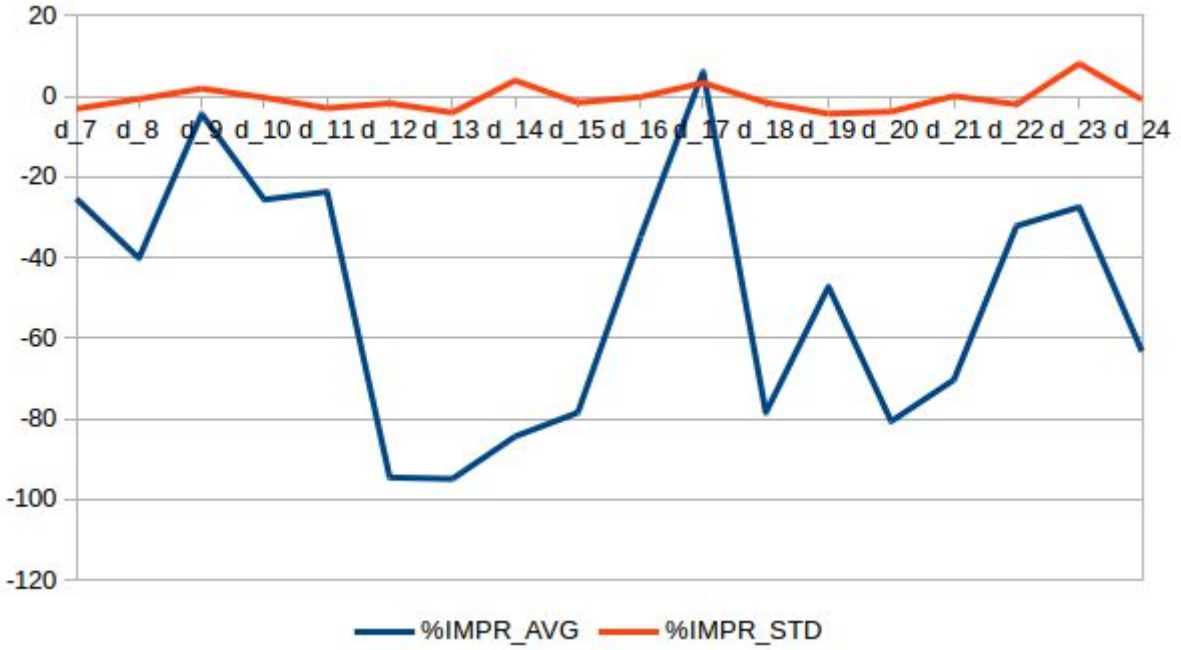
Comparison between the original errors and the new ones:





Percentage improvements: a negative value describes how much the original error has been decreased (positive result) while a positive value describes an increasing compared to the original value (negative result).

Percentage improvements:



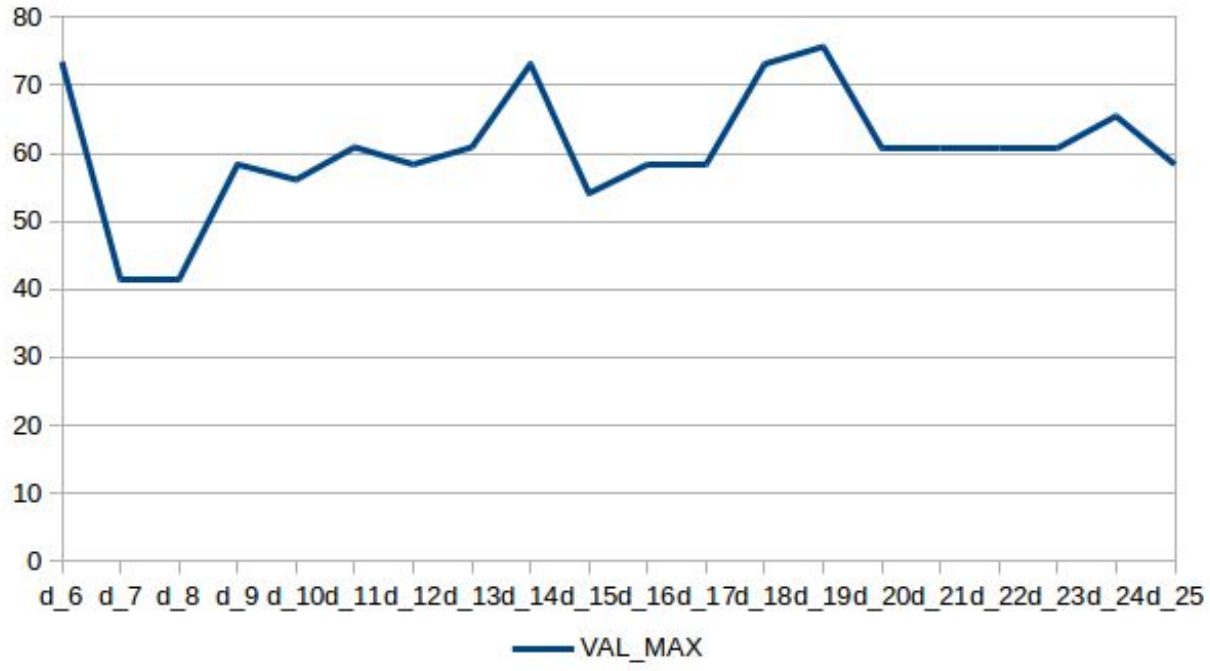
Parameter observations

Quite a lot of parameters range in specific intervals, although sometimes values deviate for a large percentage. But besides these “outsiders” it is possible to see that many daytime values range close to specific intervals. You can see an example of this trend in the following pages where for the defender and the striker roles various interesting charts are reported.

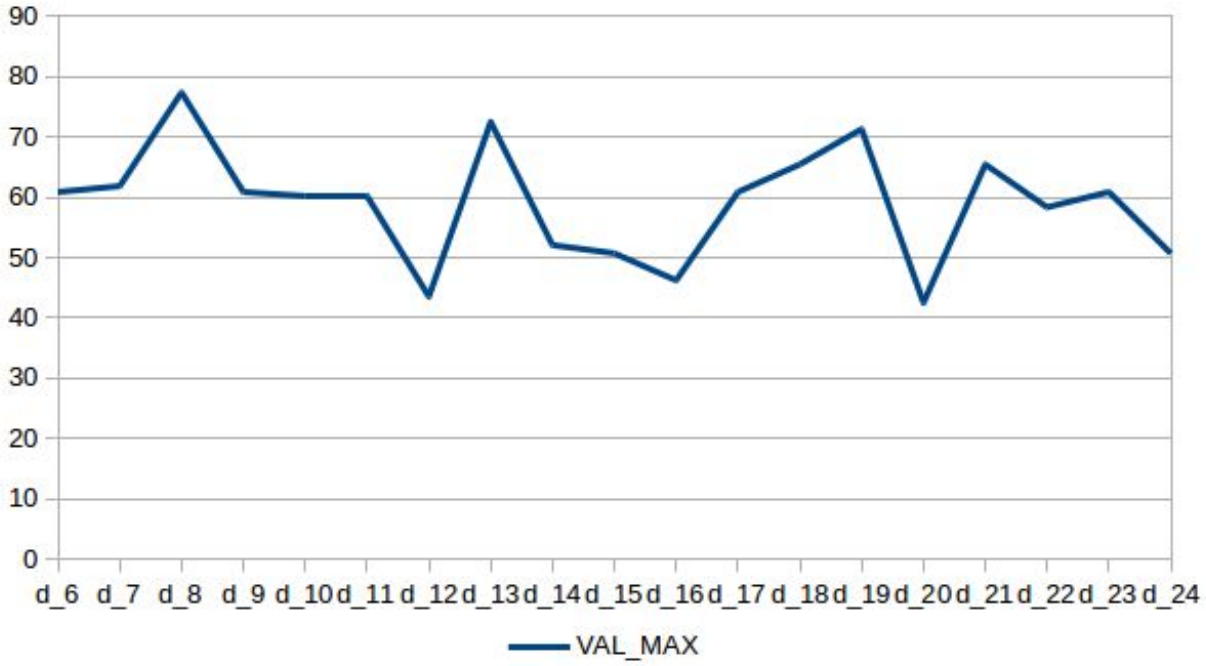
VAL_MAX

It is a parameters which is used to weigh the team statistics according to the current team ranking compared to all its opponents. As you can see for both the defenders and the strikers its value ranges around 50 and 80 although it is possible to see that there are outsider values like the daytimes near the value 40.

Defenders chart



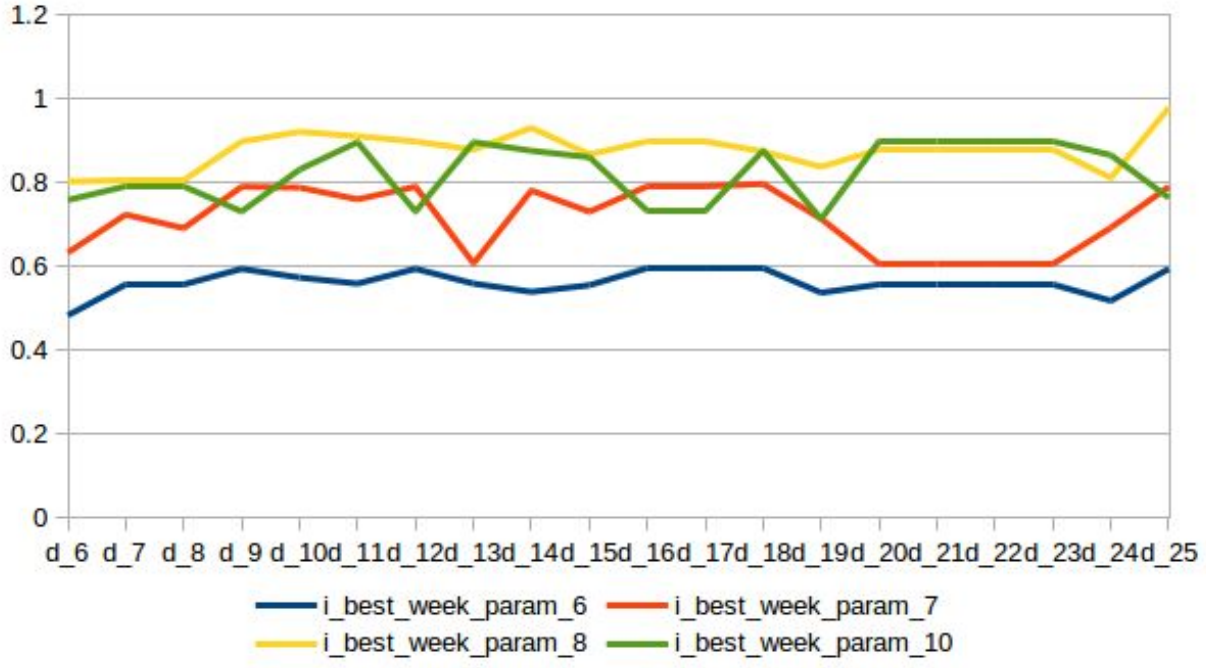
Striker chart:



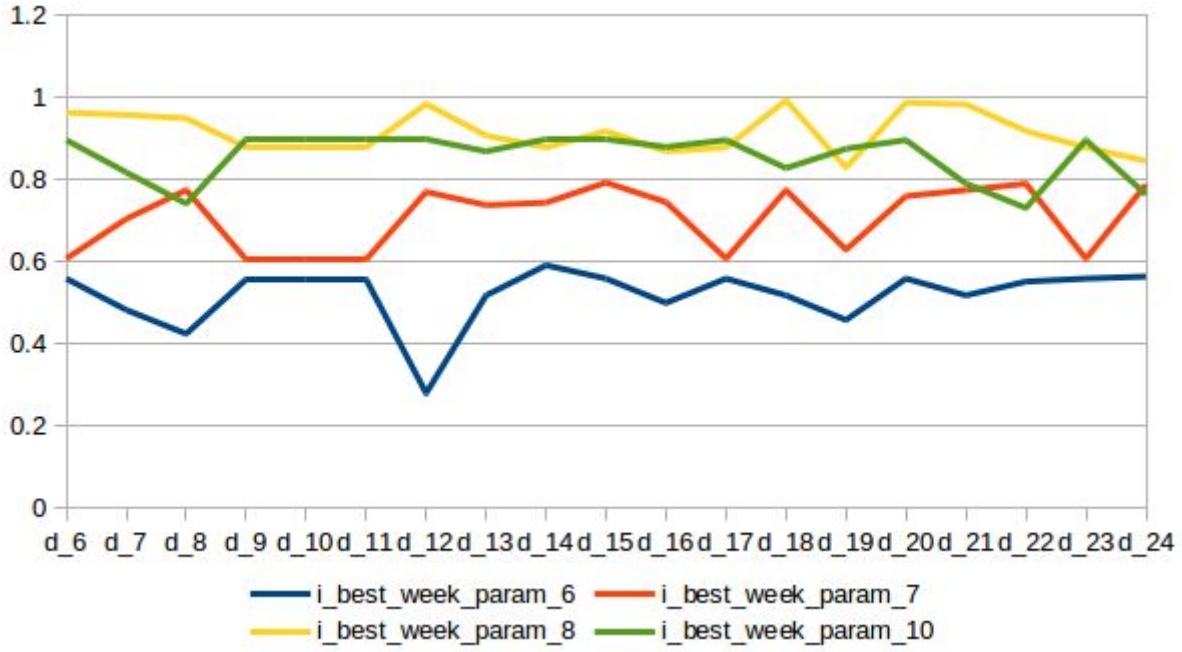
I_best_week_params

These parameters weigh how the footballer's probability to play will influence the forecast. Also in this case we can see that the parameters always range around specific values.

Defenders chart



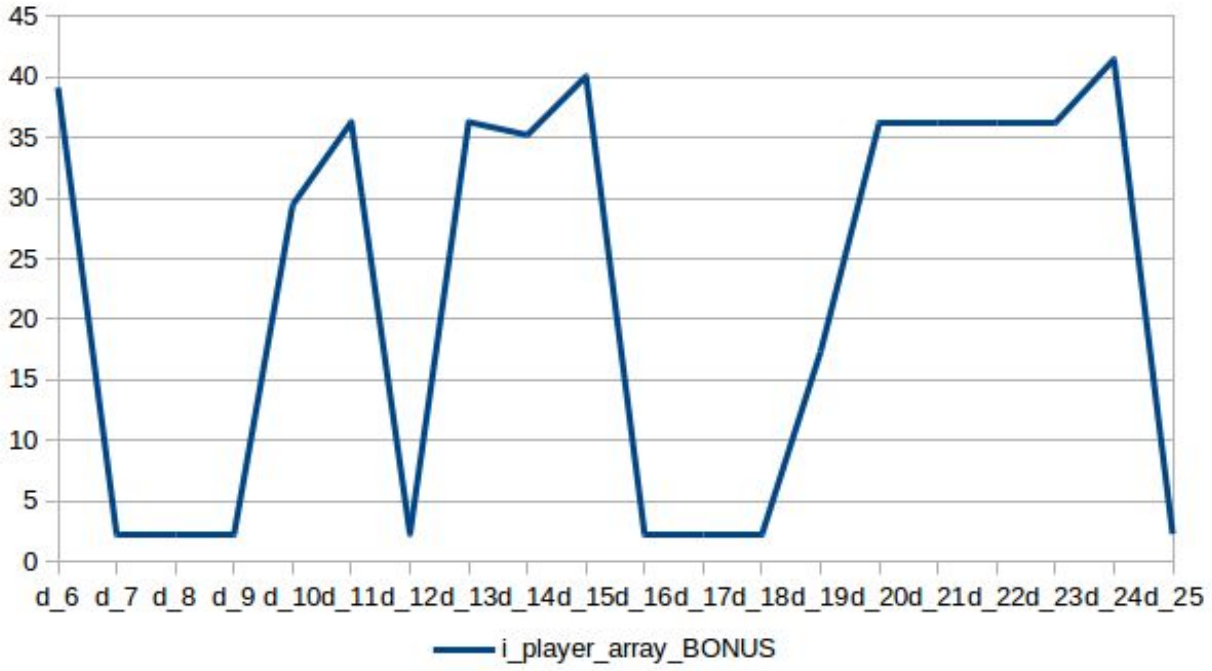
Strikers chart



Of course some parameters were very different between distinct daytimes and distinct roles, also with quite a lot of variations. You can see an example in the following charts, always plotted considering the defender's and striker's role.

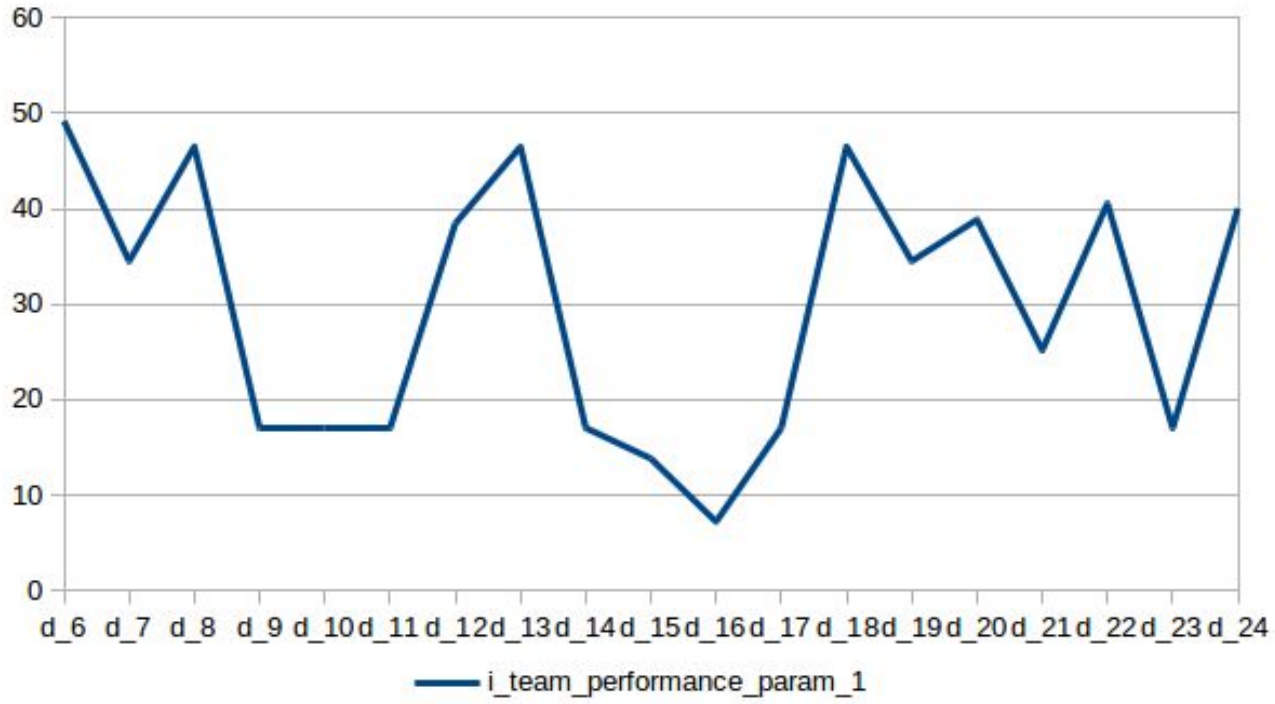
Defender: bonus_array

The bonus_array are the parameters used to weigh how the bonus and malus grades will impact on the forecast of the current player.



Striker: i_team_performance_param_1

i_team_performance_param_1 is the parameters that take into account how the current player team has performed in the previous daytimes.



4.2.3 TESTS CONCLUSIONS

After running these tests it is easy to observe that a good improvement has been make by our model (reduction of the error average and standard deviation). Moreover, the general trend is, for most of the parameters, to converge around specific values. This means that a possible optimization that can generalize all the championship daytimes is probably achievable. Moreover such generalization, being runned on multiple daytimes, will have a good probability not to end up in an overfitted model like in this test case.

4.3 ALL DAYTIME TESTS

4.3.1 INTRO

The idea of Teamies was to find the most efficient single set of parameters able to give good forecast results on all the championship daytimes. This request was dictated by the need of a “more simple model” to use during all the matches of the year, in order not to have to change too many parameter sets on different daytimes. To achieve this goal I run the next tests considering all the championship daytimes except for the 25% of the data used as testing set, hoping to find common parameters for our model.

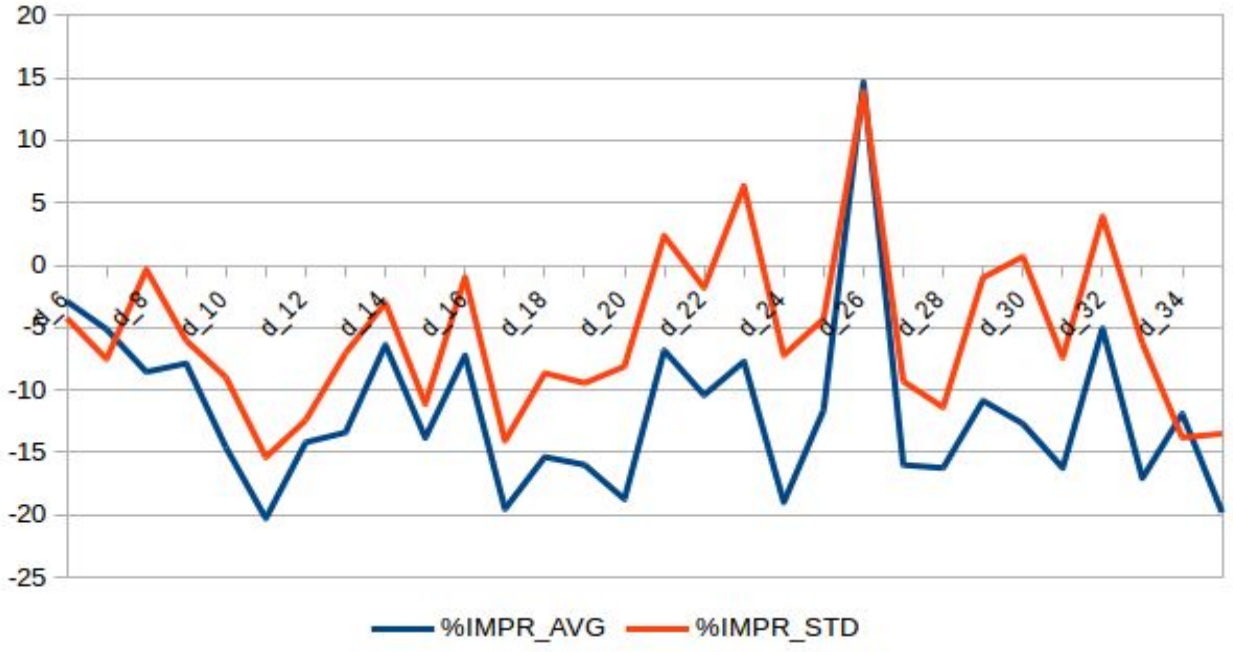
The set for the tests has been run considering the daytimes in the range from 6° to 35° with the exception of the daytimes 6, 11, 15, 20, 25, 30 and 35 used as a testing set.

4.3.2 RESULTS

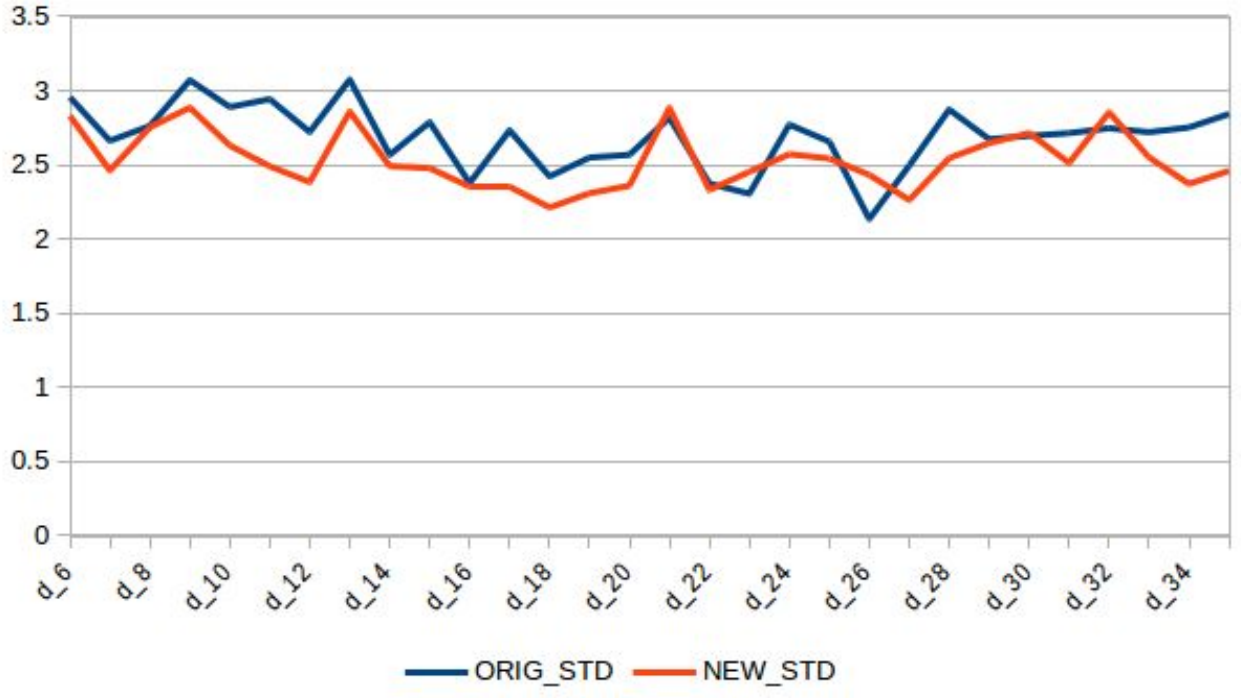
At first sight it is possible to observe that both the error average and the standard deviation improved quite well, with a better optimization on the error average.
In this analysis the results for the strikers and goalkeepers are reported.

Strikers

In the following graph it is possible to observe the percentage improvement for the error average and the standard deviation. The average is always reduced except in one case while the standard deviation has worsened in few occasions. A negative value means that the error was reduced by a certain percentage amount compare to the original forecast (positive result) while a positive value means it has been increased (negative result) .

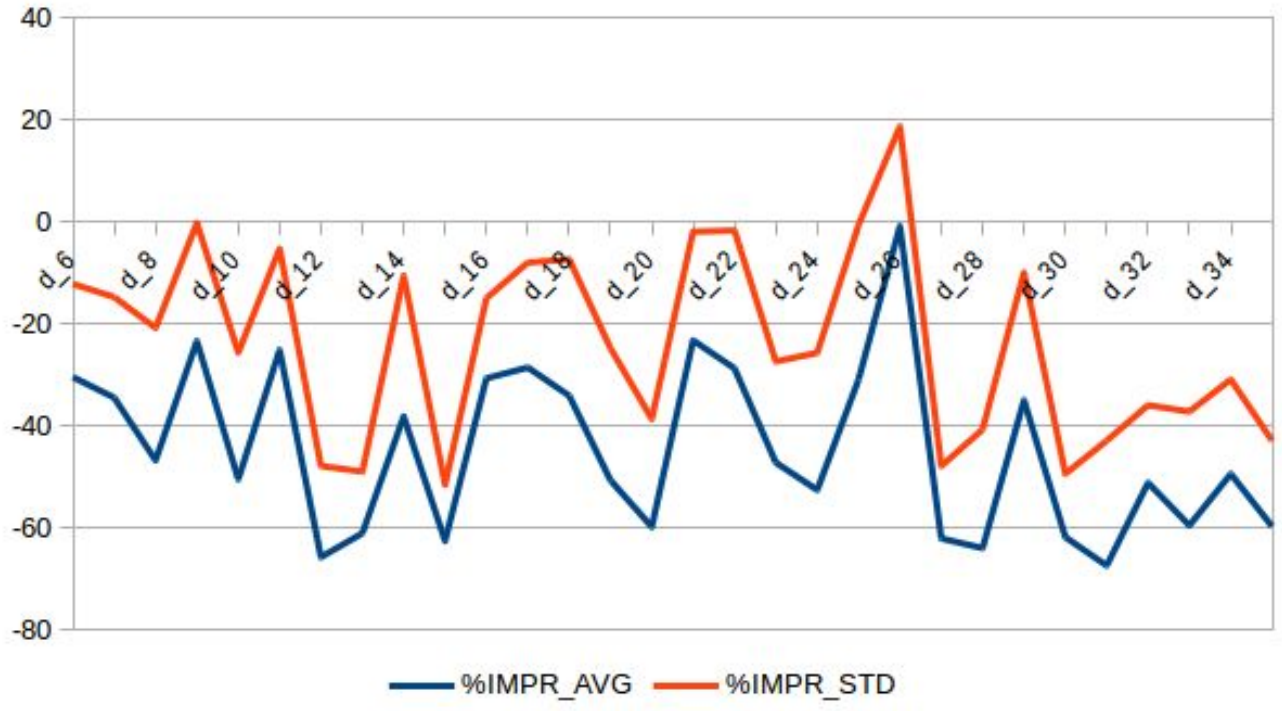


In the following two graphs it is possible to observe how the error average and the standard deviation changed in comparison to the original parameters values.

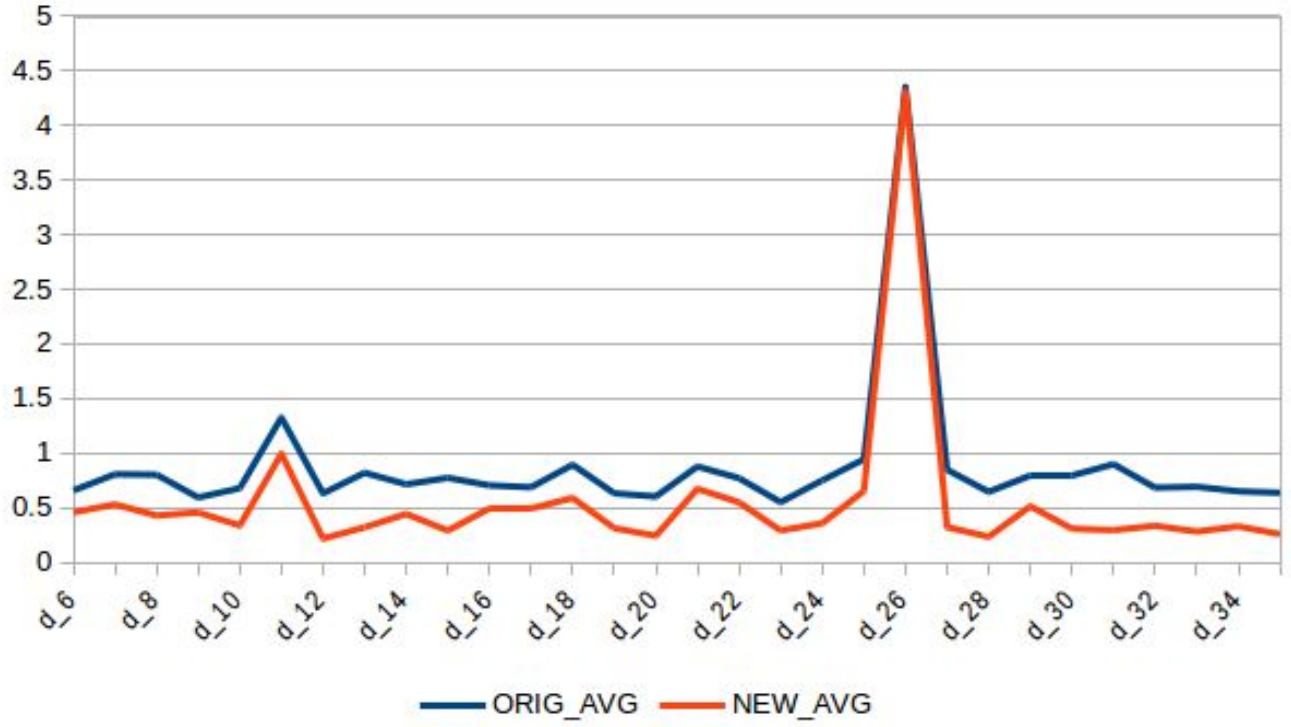


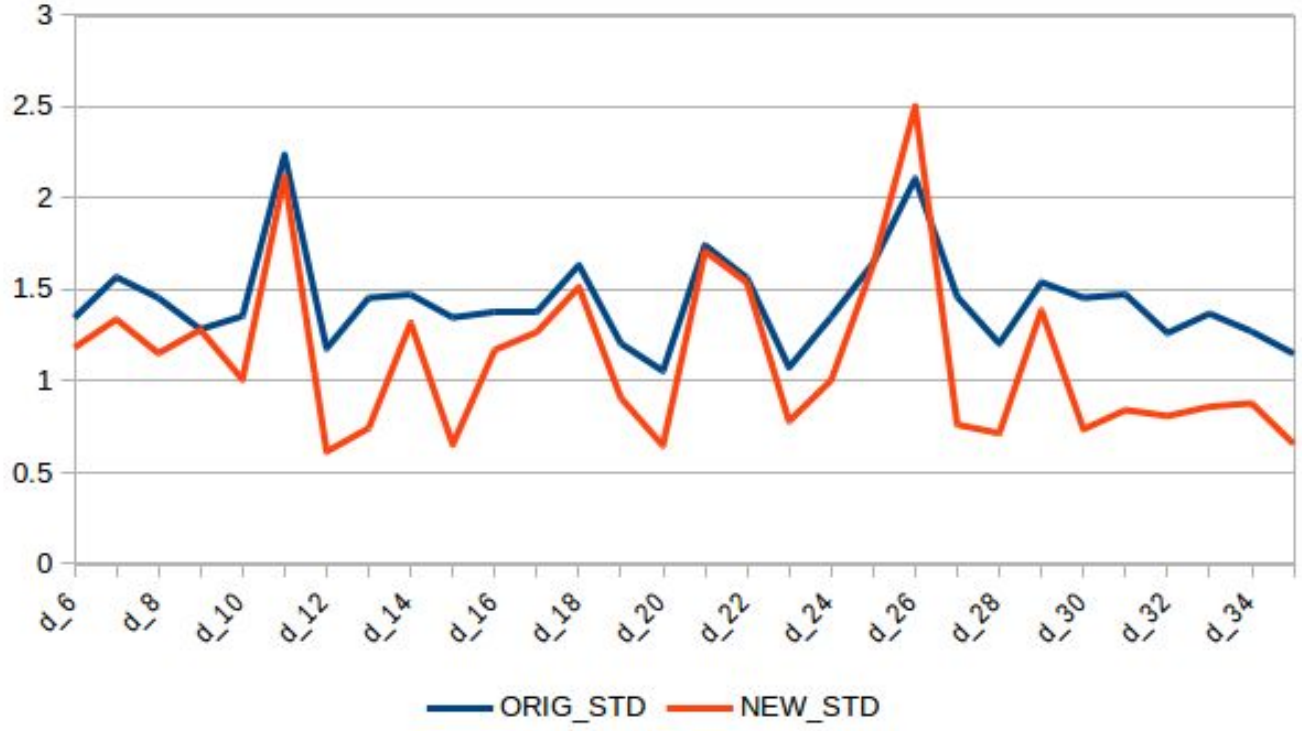
Goalkeepers

In the following graph it is possible to observe the percentage improvement for the error average and the standard deviation. It is interesting to observe that the average error is always reduced (more than the striker role) and also the standard deviation has good results with only a worsening value in one occasions. A negative value means that the error was reduce by a certain percentage amount compare to the original prevision (positive result) while a positive value means it has been increased (negative result) .



In the following two graphs it is possible to observe how the mean error and the standard deviation changed compared to the original parameters





Distinct player's improvements

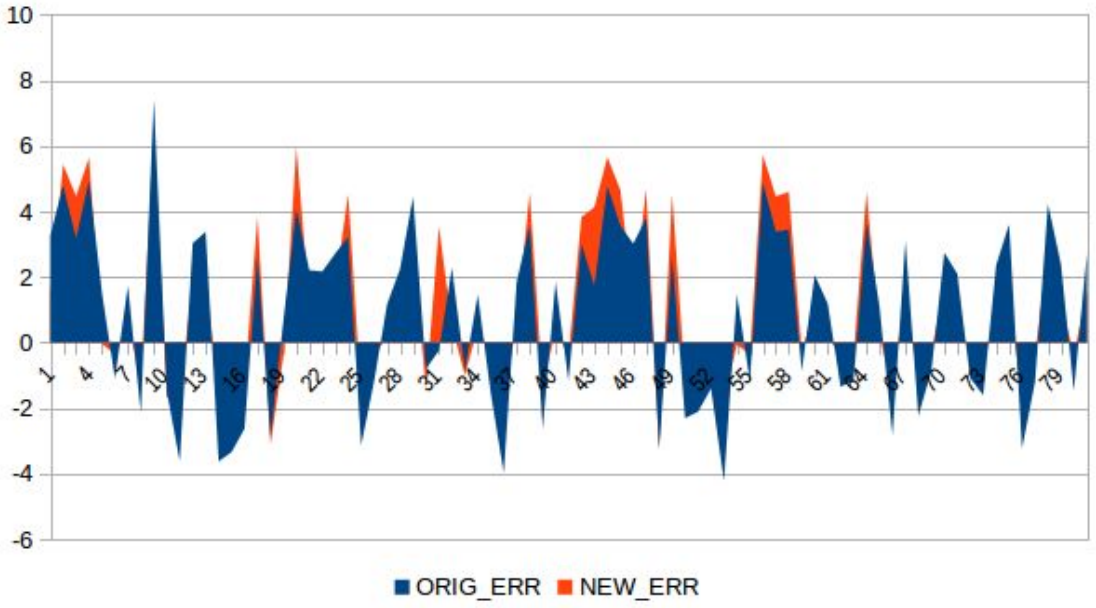
Considering the error average and the standard deviation it is easy to understand that a general improvement has been achieved but looking closely on how many players have a lower error compared to their evaluation with the original parameters it is possible to notice that they are not so many. Even in this case it is possible to see an improvement but it is still not enough. The reason is that in these tests we evaluated together all the players with probability to play from 0% to 100%. So the dataset presents values, when that probability to play is low, with a large variation due to the high chance not to play. Moreover, a lot of players with a 0% probability to play were considered in the optimization even if it was not necessary; actually it could even worsen the results. It is possible to see these statistics in the following charts for some of the daytimes used as data test (11° - 15° - 20°). The results are relative to the strikers role with the numbers of improved, worsened, and unchanged error values.

Daytime 11

Total number of improved players: 59/104. Percentage: 57%

Total number of worsened players: 22/104. Percentage: 21%

Total number of unchanged players: 23/104. Percentage: 22%

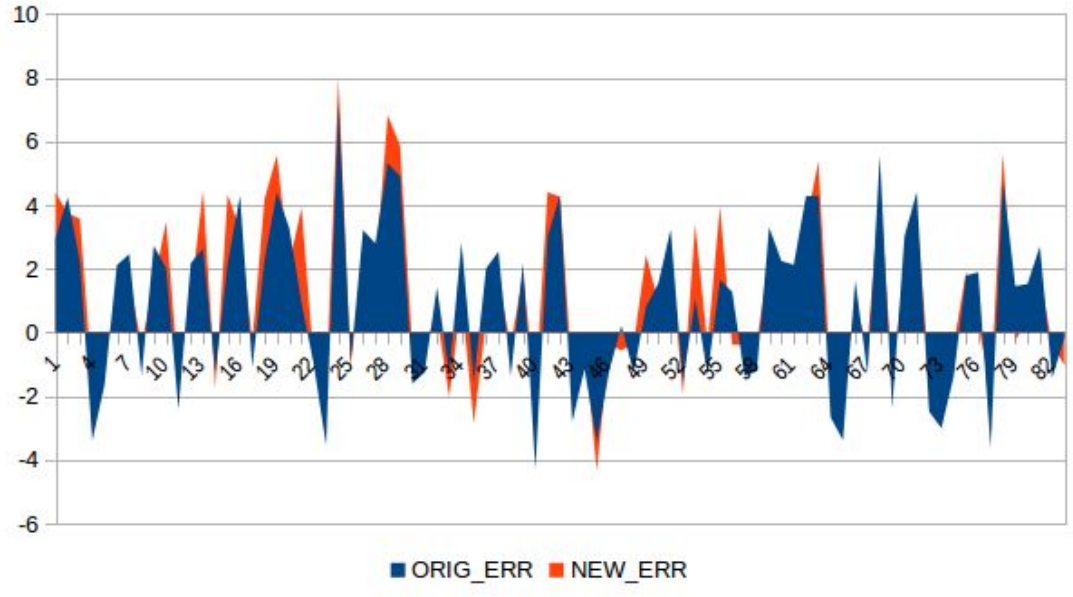


Daytime 15

Total number of improved players: 57/104. Percentage: 55%

Total number of worsened players: 26/104. Percentage: 25%

Total number of unchanged players: 21/104. Percentage: 20%

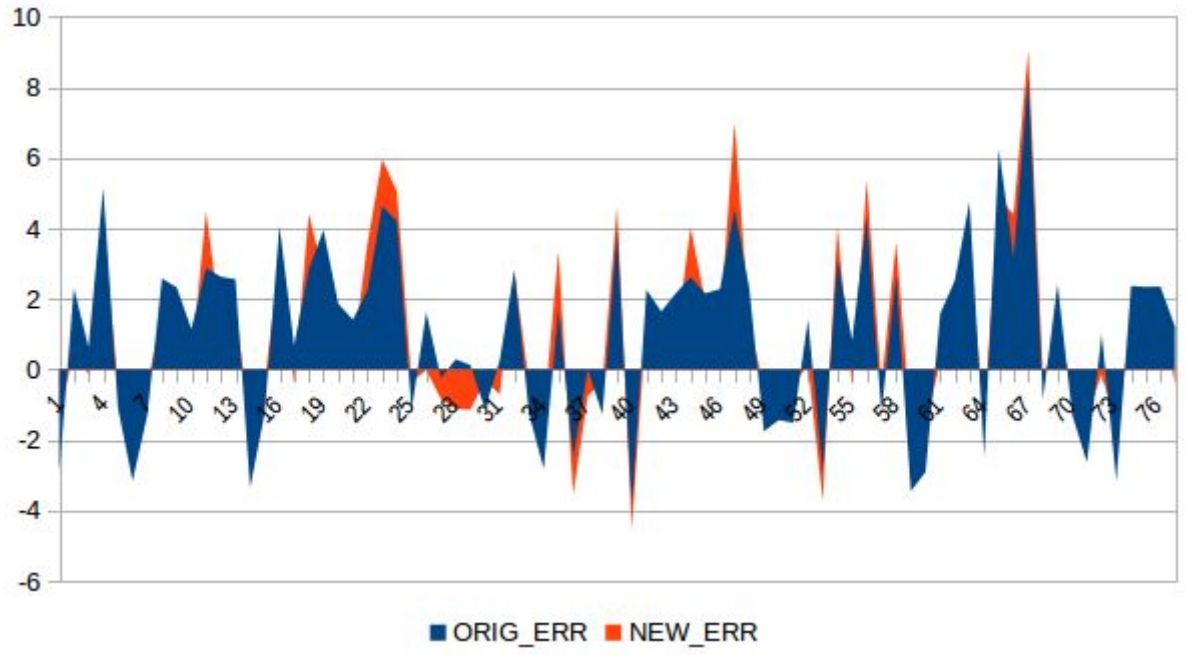


Daytime 20

Total number of improved players: 55/100. Percentage: 55%

Total number of worsened players: 22/100. Percentage: 22%

Total number of unchanged players: 23/100. Percentage: 23%



4.3.3 TESTS CONCLUSIONS

After running these tests we were able to understand that a general optimization for all the championship daytimes is fully achievable as shown from the good results obtained by the improvement of both the error average and the standard deviation. Moreover, considering that lowering down the average error was easier than lowering down the standard deviation, we decided to give more importance to the last one in the next experiments. Our idea was to force the deviation in a better way compared to the average. Furthermore it was also possible to notice that even if the general result was quite good the number of players with a prediction better than the one with the old parameters could be improved. In order to do so all the data with a lot of uncertainties (probability to play lower than 70%) should be avoided or analysed in a separate way. One example on why they could give a great impact on the model is that some players with a probability of 45% (very unlikely to play), could actually enter in the field and perform quite well. In doing so the model would find a great difficult to adapt because this situation is not easily predictable.

To sum up, after these tests I was able to understand two main things:

1. It is possible to find parameters that can optimize the original algorithm for all of the championship daytimes
2. The players with probability lower than 70% cannot be analyzed with the players with a probability bigger or equal to 70%. That is because of their variations, caused by their uncertainty to play, make it more difficult to create a good model.

4.4 ALL DAYTIME TESTS WITH FILTERS

4.4.1 INTRO

After running all the previous tests and after all the considerations taken into account, especially on the division of the dataset according to the probability to play, we decided to run more tests splitting the dataset into 2 parts: the first one with a probability to play bigger than or equal to 70% and the second one with a probability to play between 45% and 70%. Moreover we decided to give more importance to the standard deviation in the optimization because we wanted to try to lower down as much as possible this result considering that lowering down the average was quite easy. The set for the tests has been run considering the daytimes in the range from 6° to 37° with the exception of the daytimes 7, 11, 15, 19, 23, 27, 31 and 35 used as testing set.

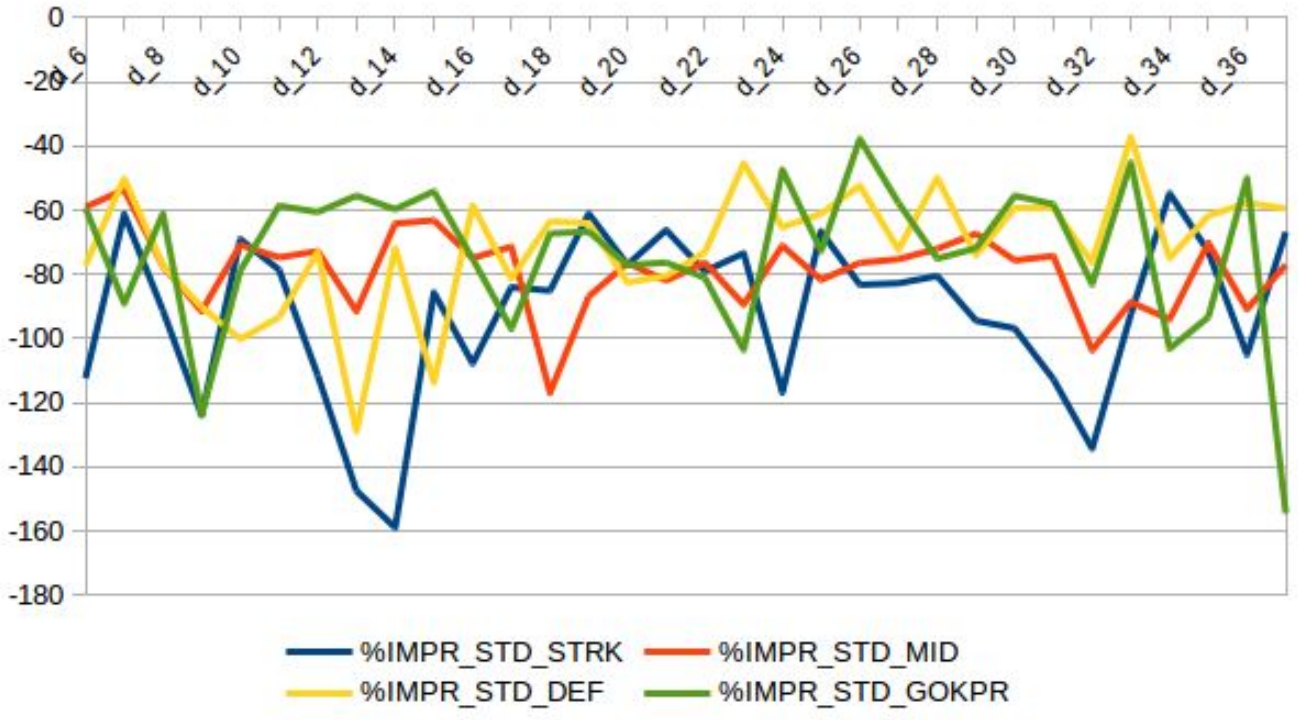
4.4.2 PROBABILITY LARGER THAN 70%

The results for these groups are very positive, it is possible to see that a very strong improvement has been made for all the roles with a very high success percentage. As a matter of fact a lot of players' performances forecasts are better than the ones with the original parameters. Moreover, thanks to the more importance given to the standard deviation in the fitness function the improvements achieved for this specific error are very significant. On the other hand the error average has been improved but a bit less compared to all the previous tests. In the following charts all these statistics are shown.

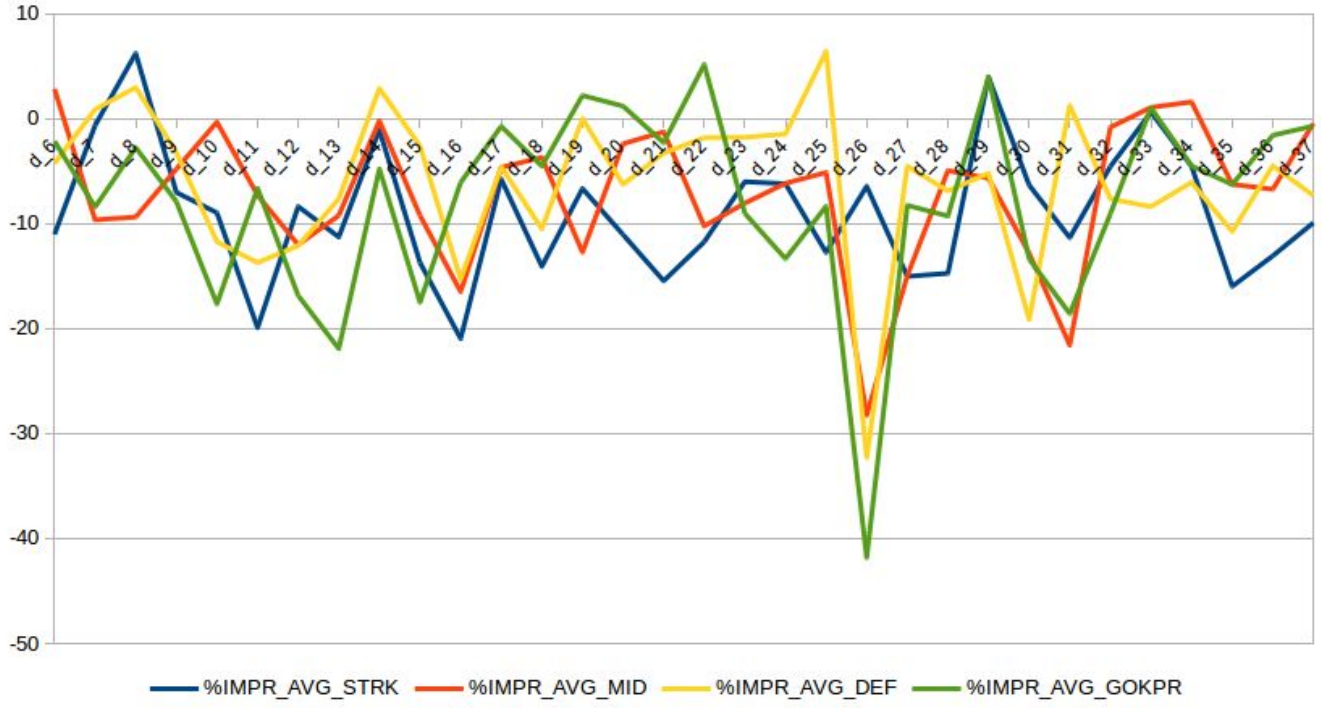
Error mean and standard deviation

The following two graphs show the percentage improvements for the standard deviation and the error average for all the roles. Negative values describe a reduction of the error compare to the ones obtained with the default parameters (positive result) while positive values indicate an increase compare to the original error (negative results).

Improvements in the standard deviation:



Improvements in the mean error:

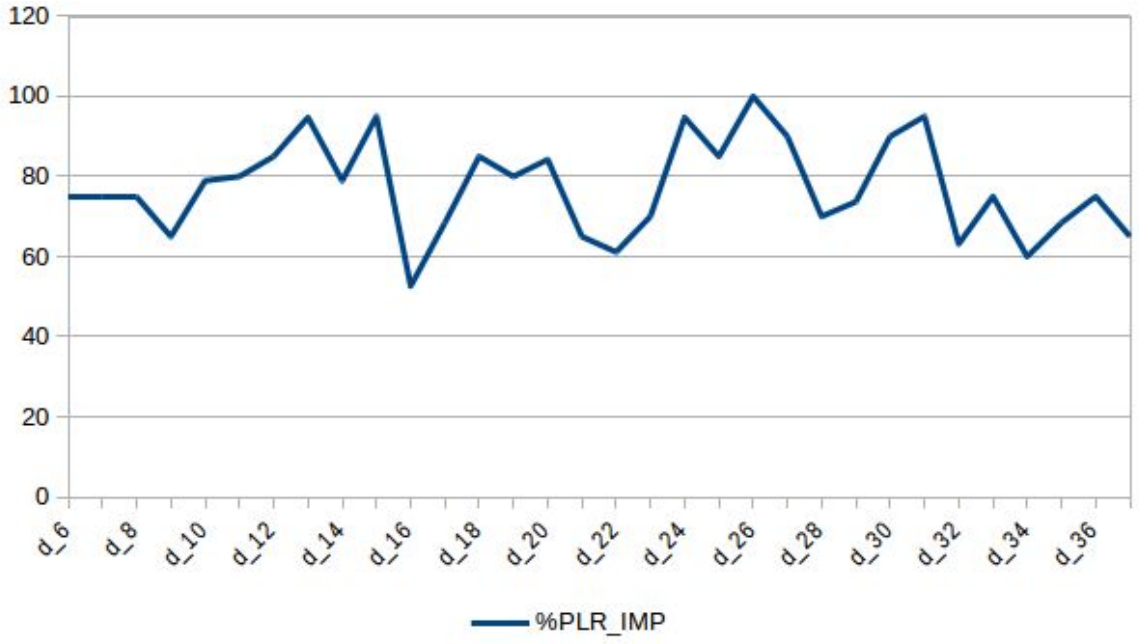


Distinct players' improvements

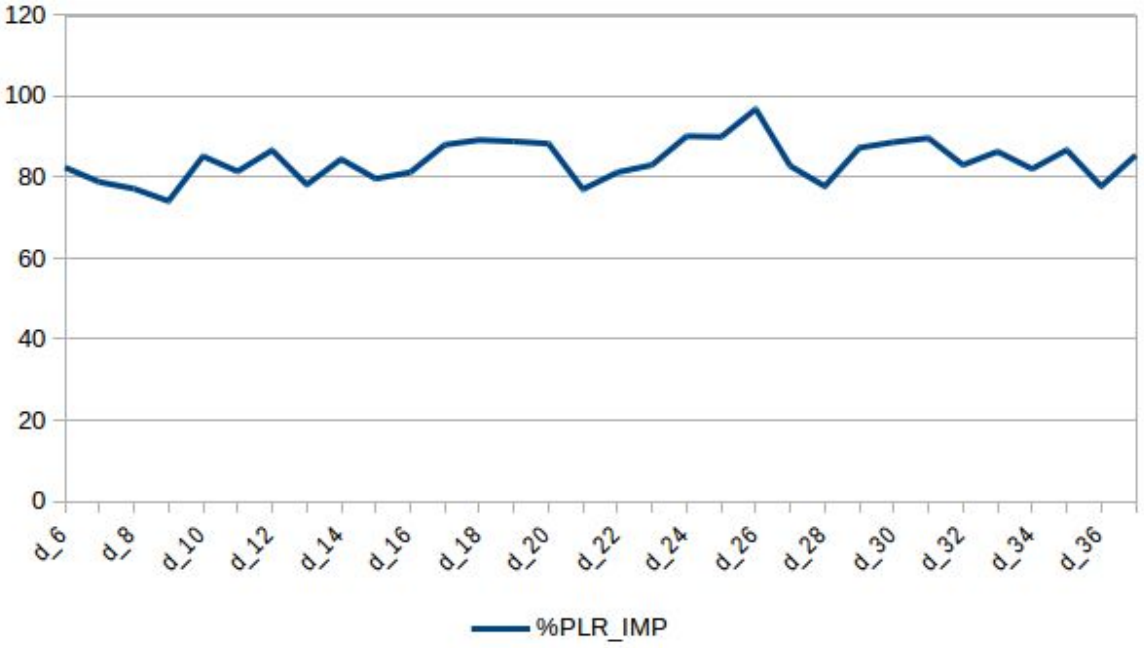
In the following chart one can see the percentage of the improvements for the players with a better forecast compared to the old ones obtained with the original parameters for all the roles.

It is possible to see that all the roles have improvements over the 60%-70%.

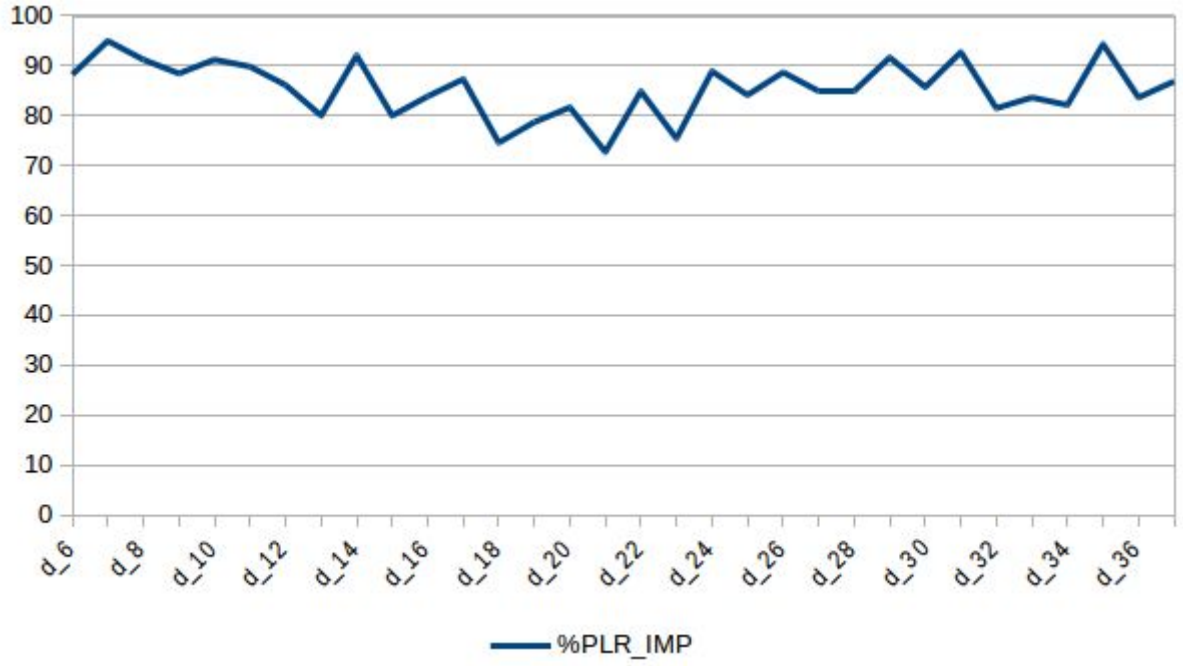
Goalkeepers



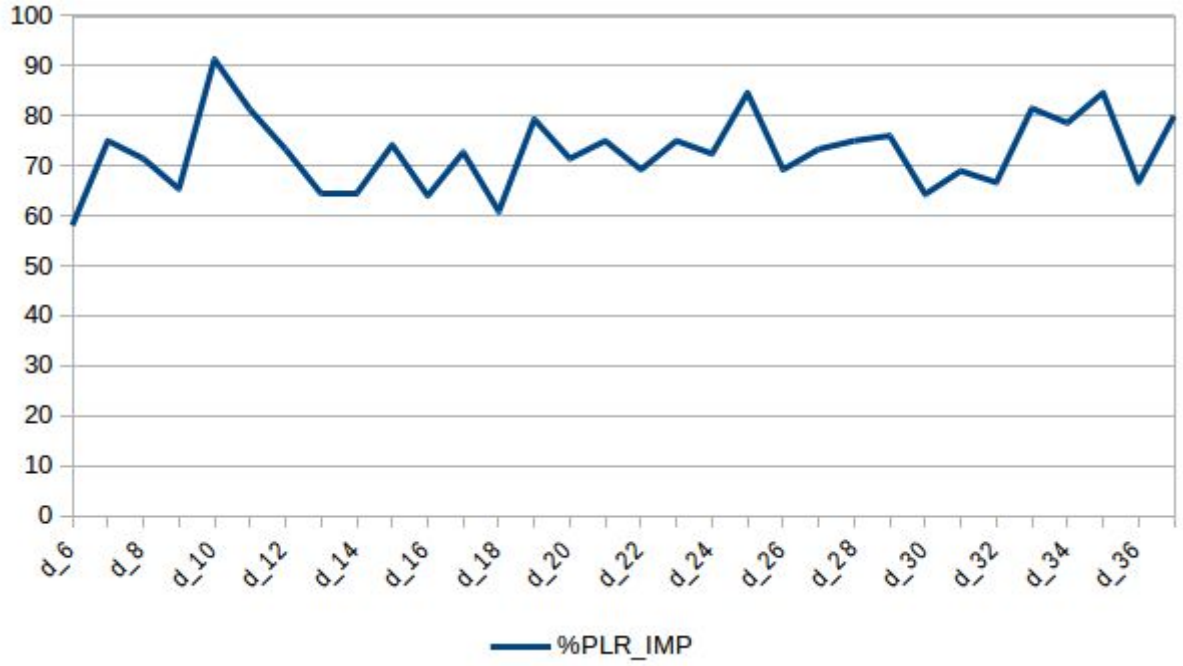
Defenders



Midfielders



Strikers



To give a general idea on the success of the model, the results for the total number of improved players of these tests were compared to the results of the tests without filters regarding the striker's role for the same daytimes previously analyzed (11, 15, 20).

Daytime 11

Total number of improved players: 26/32. Percentage: 81%

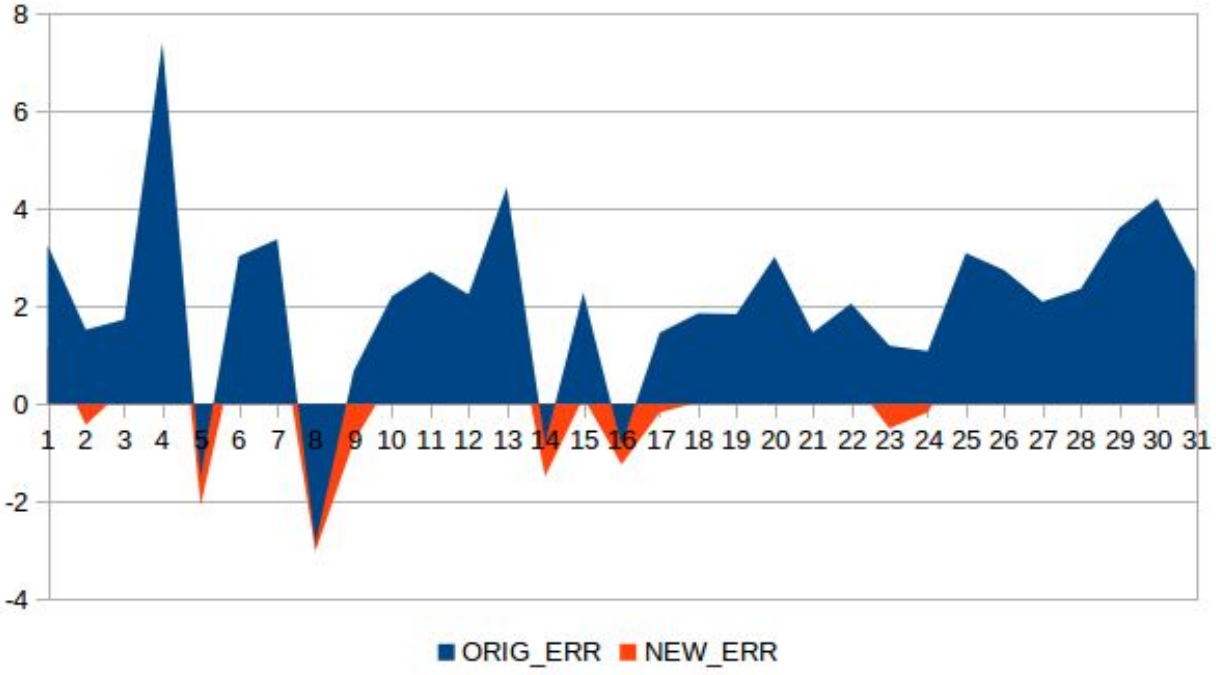
The old percentage was 57%.

Total number of worsened players: 5/32. Percentage: 16%

The old percentage was 21%.

Total number of unchanged players: 1/32. Percentage: 3%

The old percentage was 22%.



Daytime 15

Total number of improved players: 23/31. Percentage: 74%

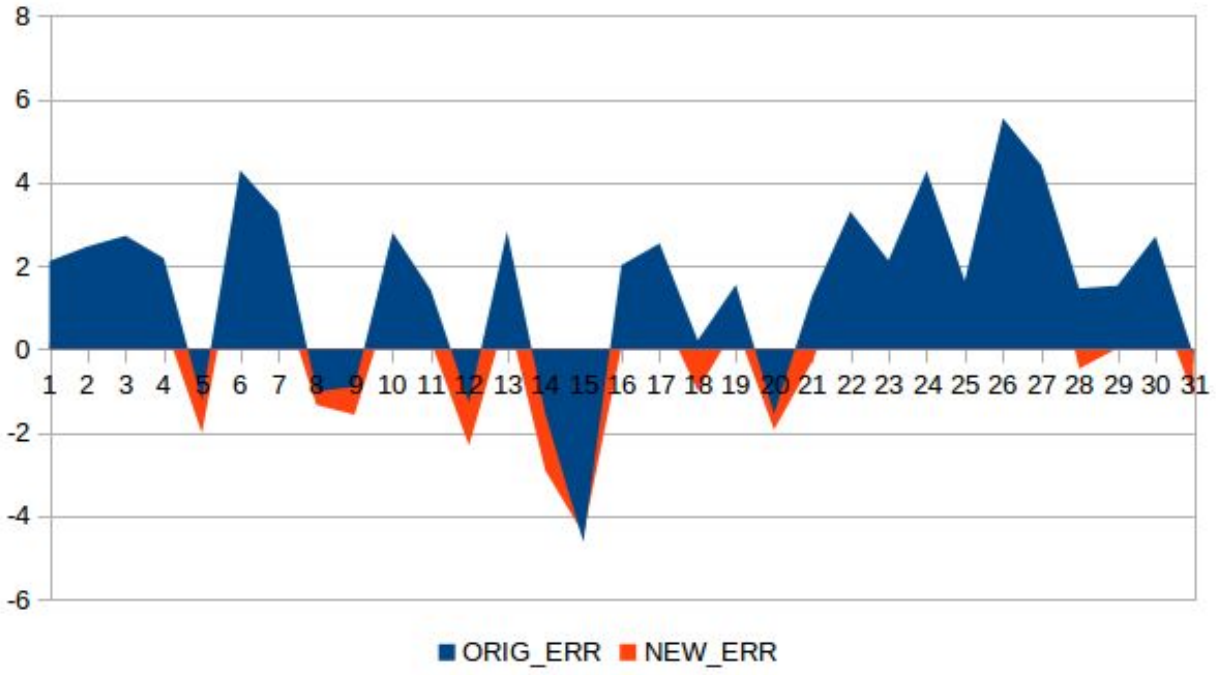
The old percentage was 55%.

Total number of worsened players: 8/31. Percentage: 26%

The old percentage was 25%.

Total number of unchanged players: 0/31. Percentage: 0%

The old percentage was 20%.



Daytime 20

Total number of improved players: 20/28. Percentage: 71%

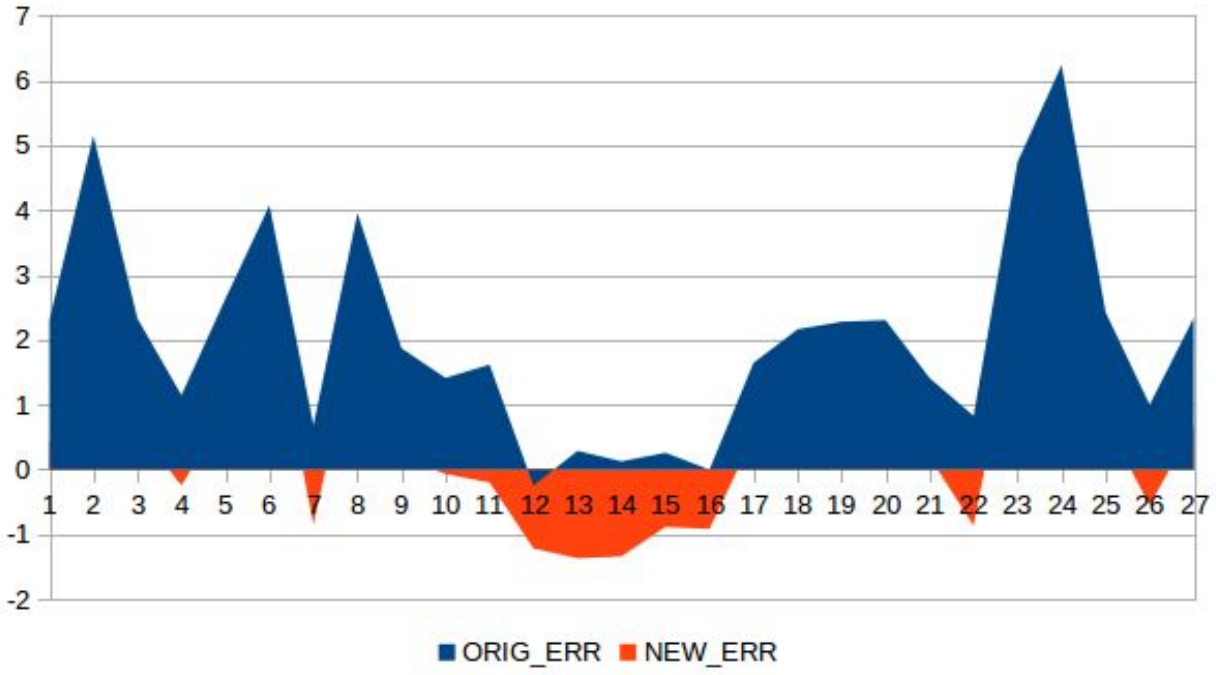
The old percentage was 55%.

Total number of worsened players: 7/28. Percentage: 25%

The old percentage was 22%.

Total number of unchanged players: 1/28. Percentage: 4%

The old percentage was 23%.



4.4.3 PROBABILITY BETWEEN 45% AND 70%

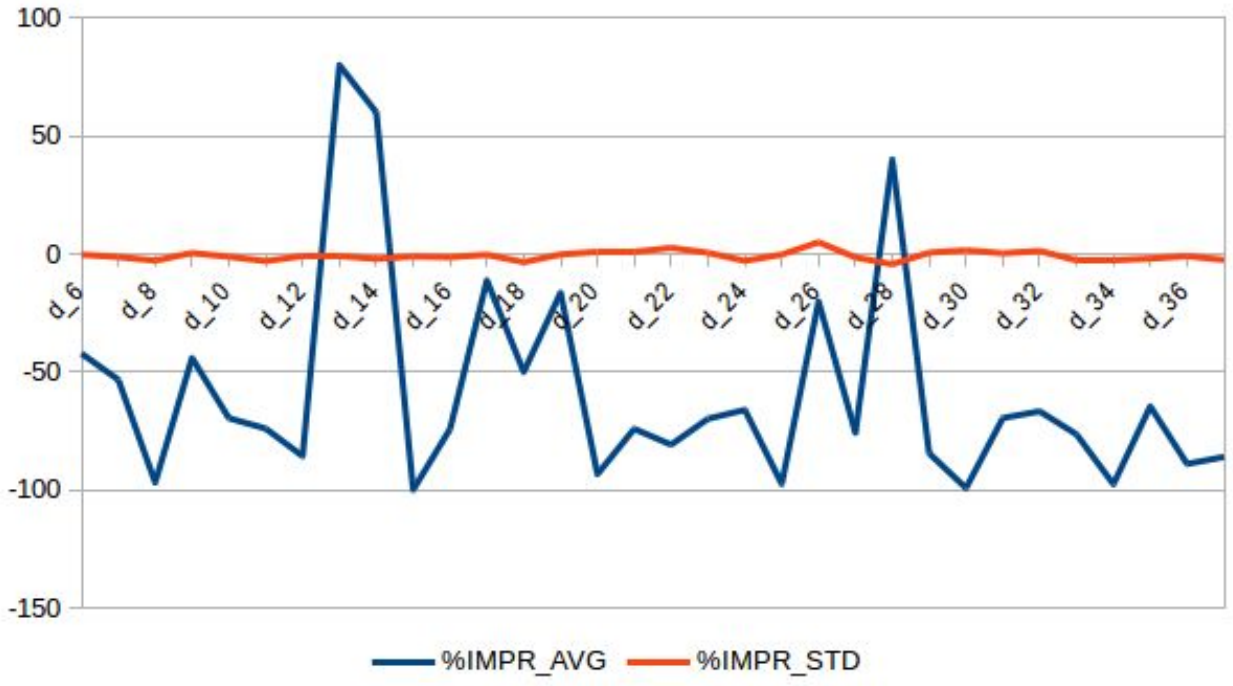
Running the same tests used for the range with a probability larger than equal to 70% (giving more importance to the standard deviation compare to the error average) for the dataset ranging from 45% to 70% did not give the same good results. The problem is that in this range of values players are very unlikely to play so, usually, the best advice is not to line them up for the match. But of course during the matches there are a lot of possible, unpredictable, variations and if some of these players enter the field and take quite good grades the forecasts errors can become rather high. Considering that I decided to run tests giving more importance within the fitness function to the error average instead of the standard deviation. The reason is that I wanted to try to lower down the error which could become quite big if a lot of players did not perform as they usually do.

The result were quite good, there were improvements for almost all the daytimes in compare to the original forecast.

Error average and standard deviation

The following chart reports the percentage improvements for both the error average and the standard deviation compare to the original results.

Negative values describe a reduction of the errors compare to the ones obtained with the default parameters (positive result) while positive values indicate an increase compare to the original errors (negative results).



Distinct players' improvements

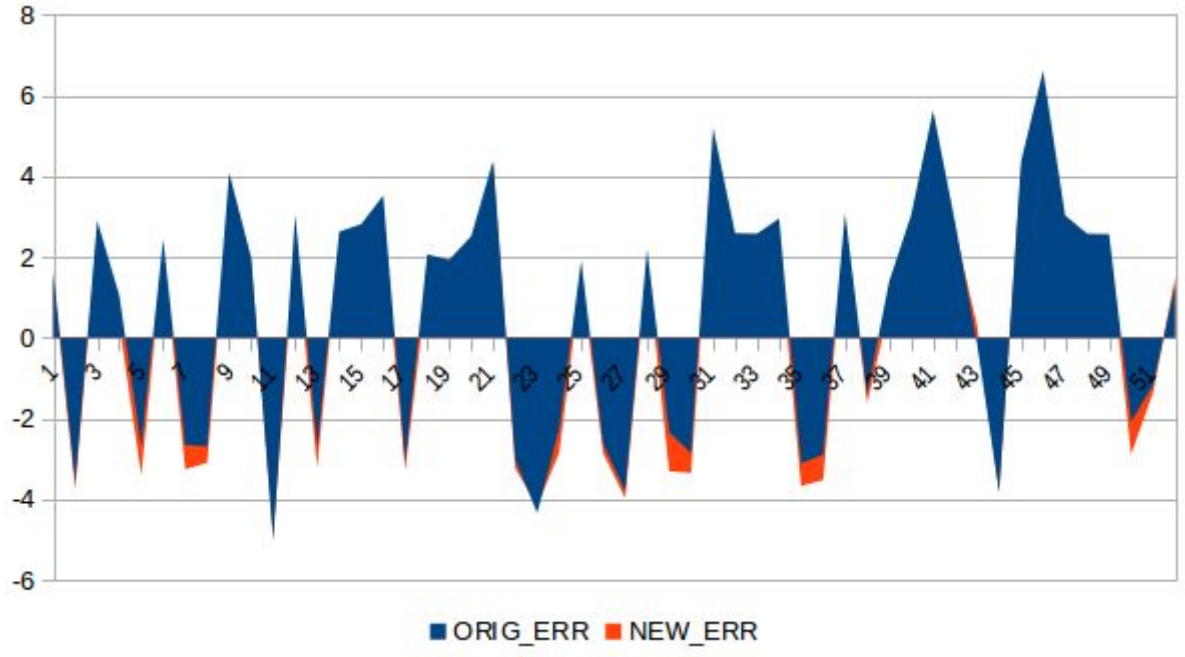
In the following pages the results are reported which show the percentage of improved, worsen and equal distinct players errors compared to the original values. The results are improved but the percentage of success is not so high considering, as already said before, that these players are very difficult to model with a proper generalization function. To show the percentage results the test daytimes are used.

Daytime 7

Total number of improved players: 33/54. Percentage: 61%

Total number of worsened players: 19/54. Percentage: 35%

Total number of unchanged players: 2/54. Percentage: 4%

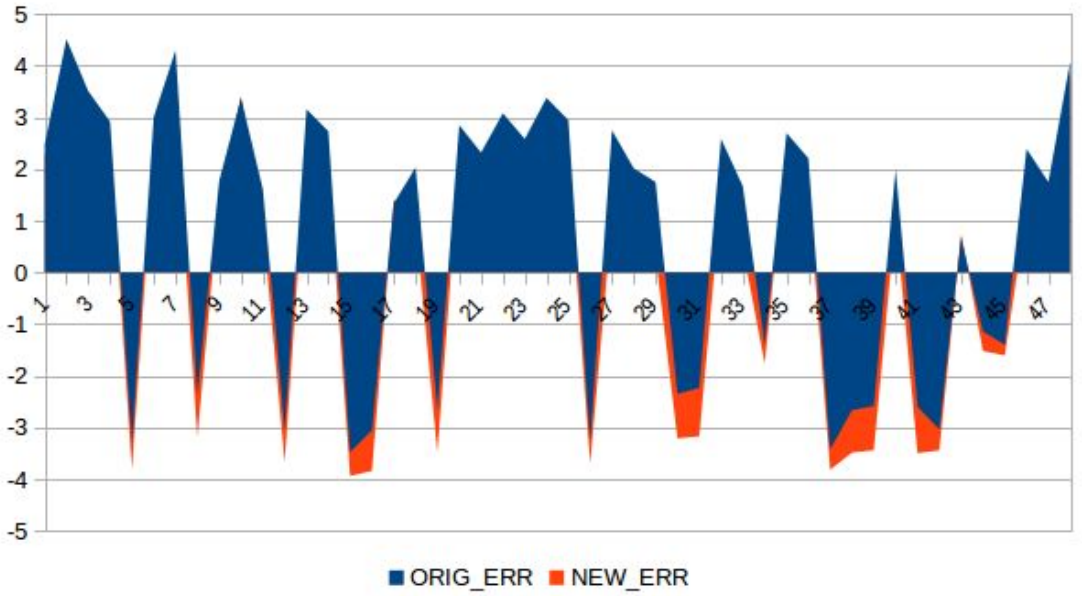


Daytime 23

Total number of improved players: 28/51. Percentage: 55%

Total number of worsened players: 20/51. Percentage: 39%

Total number of unchanged players: 3/51. Percentage: 6%

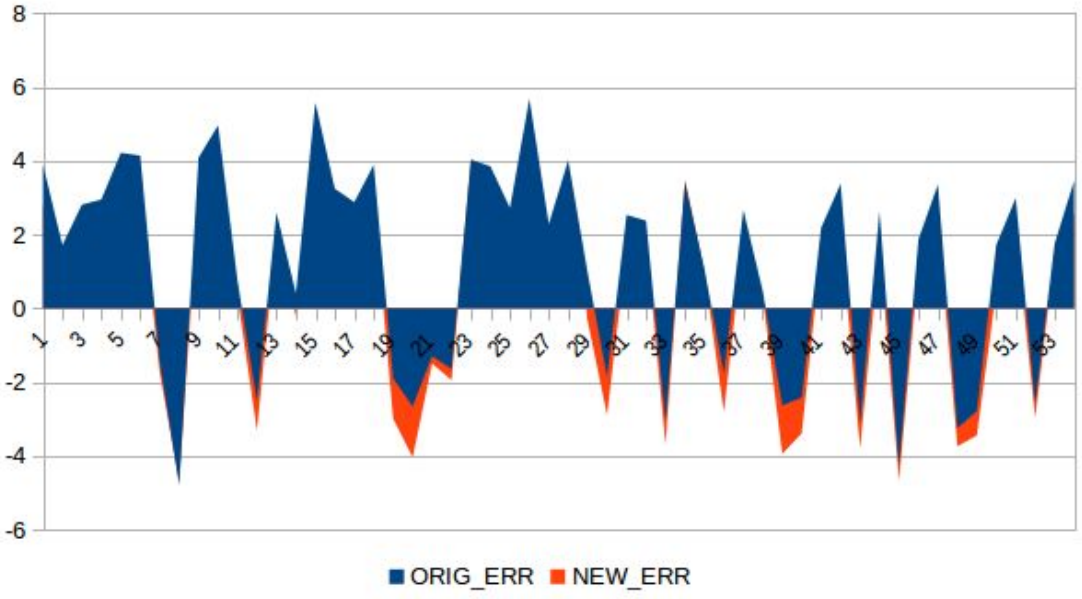


Daytime 35

Total number of improved players: 37/54. Percentage: 69%

Total number of worsened players: 17/54. Percentage: 31%

Total number of unchanged players: 0/54. Percentage: 0%



CONCLUSIONS

5.1 TESTS RESULTS

After all the tests carried out it is possible to observe that all the errors with the original parameters were improved with the new ones. The mean error and the standard deviation have significantly lower down their values; moreover, the percentage of players with a better forecast has strongly increased.
For this reason is definitely possible to say that the goal of improving the original forecasts has been successfully achieved. The original parameters have been improved allowing the Teamies' algorithm to provide its customers with better forecasts.

For the players with higher probability to play (bigger than equal to 70%) the forecasts have been strongly improved. This is positive not only for the better precision but also because these players are the ones who need a more precise forecast compared to other players with a lower probability. The reason is that these are the footballers which impact the most on the total amount of points that a fantasy manager can gain during a championship daytime, so having high precise results on them is far more important compared to players with lower probabilities who usually have a very small impact on the total final result.

For the players with lower probability to play (between 45% and 70%) the results are also positive but not as much as the probability range starting from 70%. This results less optimized do not have, however, a relevant negative effect on the forecasts

provided to the Teamies' customers because selldom these players are chosen to be line up. Furthermore in the event that a regular player (player with probability larger or equal to 70%) is removed before the beginning of the match from the original formation for special occasions, such as an accident, the probability of non regular players that are going to substitute him are updates and a more reliable forecast can be calculated. So the problem with the range 45%-70% is only present when players who are not supposed to play actually enter in the fields during a match and perform quite well.

As we can understand the probability for a player to be lined up is a very important parameter to forecast the best footballers. For this reason all the fantasy managers in order to get the best predictions from the Teamies platform should wait for the last advice provided before the matches (according to the rule of the fantasy football game they are playing).

It is also very important to keep in mind that trying to forecast what could happen is always difficult especially when the aim is to try to evaluate the performance of football players. In fact as a human being a lot of non statistical variables could affect the performance in unpredictable ways.

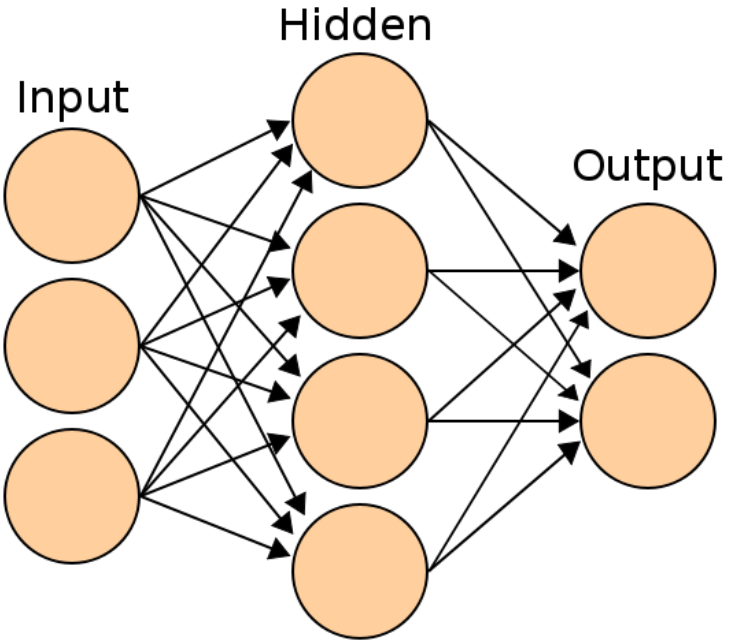
5.2 WHAT COULD BE IMPROVED

Of course a lot of improvements can be applied to this application because like in every software development situation very often there are different solutions that can be used to achieve the same goals. But in this case the most important optimization that could be applied would be to further reduce the errors for players with a probability to play ranging between 45% and 70%.

To achieve that more tests should be carried out to understand if this range could be described by a better model or if changing the algorithm and the math formulas could give better results. Of course a deeper analysis would take a lot of time and for the aim of the thesis this was not possible.
But to give a quick overview on a possible strategy to improve the results a solution could be to try to create a new algorithm specifically studied for these types of footballers. One idea could be to try to predict when a regular footballer could be substituted during a match and predict which other player could enter in the field in his place. To understand which regular footballer could be changed a possible procedure could be to collect information about his physical state (e.g: if he has some small injury) in order to forecast his ability to play for all the length of the match and, if substitute, to provide a more accurate forecast for the players who will substitute him.

5.3 NEURAL NETWORK: ANOTHER APPROACH

To solve the problem the approach was to use a evolution algorithm because Teamies already had a proprietary algorithm to evaluate players performances and they just wanted to optimize specific parameters. So the need was to have a search technique to find exact or approximate solutions to optimize the search problems and the most suitable solution in this case was to use an evolution algorithm.
But of course there are other different techniques that can be used to try to improve the forecasts. Following today's trends one approach could be to use a Neural Network. Neural Networks are brain-inspired systems which are intended to replicate the way that humans learn. Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use. They are excellent tools for finding patterns and model complex relationships which are far too complex or numerous for a human to extract and teach the machine to recognize them.



[image and information] [7]

If Teamies would have wanted to try to optimize the forecast using Neural Network the approach could have been, in short:

1. to set as input the players statistics and their teams statistics (and probably also the ones of their opponents)
2. to set as output values the real players performances.

Of course several possible strategies should be taken into consideration and tested but this could be a good starting point.

But the problem with a Neural Network is that it is notoriously difficult to figure out what they have learned, i.e. to extract the knowledge from them once trained, and reuse the knowledge. So as a first attempt, considering that Teamies wanted to understand the relationships between a player statistics and their performances, they prefered to start with an evolution approach.

BIBLIOGRAPHY

[1] Wikipedia contributors. Wikipedia, The Free Encyclopedia. **"**Cron". Retrieved on: 18 Oct. 2018, from: https://en.wikipedia.org/wiki/Cron.

[2] Various contributors, Google Puppeteer official documentation. "Puppeteer". Retrieved on: 20 Oct 2018, retrieved from: https://github.com/GoogleChrome/puppeteer/blob/master/docs/api.md.

[3] Various contributors, Mysql npm package official documentation. "Mysql-npm". Retrieved on: 22 Oct 2018, retrieved from: https://www.npmjs.com/package/mysql.

[4] Various contributors, µGP (MicroGP) official documentation. "µGP (MicroGP)". Retrieved on: 24 Oct 2018, retrieved from: http://ugp3.sourceforge.net.

[5] Various contributors, Quora answers. "What is the difference between training data and testing data?". Retrieved on: 26 Oct 2018, retrieved from: https://www.quora.com/What-is-the-difference-between-training-data-and-testing-data.

[6] Anup Bhande, Medium answers. "What is underfitting and overfitting in machine learning and how to deal with it". Retrieved on: 26 Oct 2018, retrieved from: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76.

[7] Wikipedia contributors. Wikipedia, The Free Encyclopedia. **"**Artificial neural network". Retrieved on: 26 Oct. 2018, from: https://en.wikipedia.org/wiki/Artificial_neural_network.