3. Team The Data Miners

Authors: Samuel Jairo Cruz Gomez, Armand Alarcón Román, Àlec Lagarde Teixidó.

Abstract: Short Introduction

The following pages will explain the process of a data mining project from the initial goals and objectives until the obtained conclusions going through the selection of variables and models used to actually perform the data mining process. Firstly we are going to establish our business and data mining goals, then we are going to set our success criteria and schedule a project plan. Then we are going to select the data that we will use on the actual data mining project. In our case we will use SonarQube variables to predict the technical debt, the type of Sonar issue or the number of bugs generated. Once we have the final dataset we will build models like PCA or regression models to finally deploy the final product and draw conclusions. All throughout performing good data science practices to ensure proper results.

3.1. Business understanding

3.1.1. Business objectives

- Background

The Technical Debt dataset has collected data from several projects with the help of different tools in order to obtain a very wide range of metrics to help researchers to study the causes of the Technical Debt by establishing a ground truth. One of the tools used to collect the data is SonarQube, an open-source static code that analyses the quality of the code.

The process of obtaining metrics with SonarQube for the projects took a lot of time as it scans the whole code and takes metrics such as the number of lines, the complexity of the function and classes and the number of duplicated lines or files. It is because of the huge amount of metrics we obtain thanks to SonarQube that we want to detect the patterns that the issues can create throughout the project in order for future developers to predict and to avoid making the same mistakes.

Another type of issues were collected by the creators of the dataset, Jira issues. We won't study this type of issue in this project as we want to study the impact of the structure of the code in the Technical Debt. Jira software tracks even more kinds of issues that can be anything from a bug to a project task and we are focusing more on the code only.

- Business goals and success criteria

Our main objective in this project will be to study the impact the metrics obtained by the SonarQube software can make and detect the pattern that can be produced throughout the duration of the projects. This way we'll be using some variables present in the Sonar tables to predict the TechDebt.

The final result should be a script explaining all the patterns that were produced by the SonarQube issues and how they impact on the Technical debt in means of time and resources.

We believe that by the end of this project finding the correlation between the SonarQube issues and the patterns they can create will help a lot of new developers to prevent problems in their ongoing projects by solving issues before they evolve into something bigger.

3.1.2. Assessment of the current situation

Inventory of resources:

- Team: our team is formed by three Data Science students that have a background in developing projects related to that discipline.

- Data: the data we will be working with is the one presented in the Technical Debt Dataset which consists of a curated dataset containing measurement data from the analysis of the commits from several GitHub project' repositories using some tools.

- Tools: we will be using Python as our main programming language and sometimes some SQL language when querying data. We will have a GitHub repository where all our code and files will be saved in order to leave a trace of the evolution and changes. Many tools from Google will be used when communicating with the members of the group, such as Meet, Gmail or Drive to share documents and write the necessary reports.

- Hardware: every member of the team has a laptop or a computer to work with.

Requirements:

- The main requirement is that the project with its results and report has to be presented and delivered by the 29th of October, so we'll be planning our work taking into account the due date.
- Since we're working on a project in order to improve the quality of the data, our mission is to act responsibly and with the best practices to obtain high quality results that must be easy to follow and able to be re-executed. To

foster that correctness and reproducibility, it has to follow the project structure seen in class, namely the “Cookiecutter Data Science” template. This model will help us to collaborate more easily on the analysis.
- As presented in the theory lessons in class, an agile methodology must be followed to be able to adapt our work if any immediate change is needed, so our project can be successful in the end.
- Another requirement that might seem quite obvious but needs to be written down is that our solution has to be useful and functional for the goal defined at the beginning. It has to add value to what we already have in the industry.

Assumptions:
We assume that with the time provided we’ll have enough time to complete the demanded task satisfying the requirements stated above. Another thing we assume is that with the hardware and software we have in our hands we will be able to develop the different tasks without any problem.

Constraints:
- The main constraint we have is that we have to hand in and present our solution on the 29th of October.
- We have to limit our solution to the data and attributes provided by the TechDebt dataset v2.0. This dataset is our unique source of data to get the necessary information from the analysis and study of its tables.

Risks and contingencies
When carrying out the project some issues might appear like needing more power or resources to complete a task, then, we’ll need to be agile enough to redefine that task and redirect our project to the correct path. It might also happen that finding issue patterns in the data become more difficult than expected initially, if this occurs, a redefinition of the business goal will have to be done.

Terminology

- pattern: When there’s a pattern in your data, it means that some variables from your data are correlated and therefore you have a relationship that enables you to predict them. Consequently, you can have a good idea when or where something will happen before it actually happens.

- TechDebt: “Technical debt” is a term used in software development to describe delayed maintenance costs caused by initial tradeoffs between quality and speed.

- SonarQube: It is an open-source platform developed by SonarSource for the inspection of code quality to perform automatic reviews to detect bugs, code smells and security vulnerabilities on different programming languages.

Costs and benefits

When these issue patterns are found, projects will be able to be carried out more efficiently as data miners working on these new projects will be able to detect these patterns and thus, when an issue may occur next. Then, these errors may be prevented and time, money and resources will be saved instead of wasting this “energy”.

Given that our project and work is not remunerated and no investments have to be done, economically it hasn’t got any cost at all. Then, we might consider as its major cost the time that our team will be dedicating to proceed with the project.

3.1.3. Data mining goals and success criteria

The main goal in this data mining project is to obtain a conclusive result whether the issues such as code smells or anti-patterns can affect the performance of the project in the long run by generating future issues or the issues are actually brought by other factors. We want to see which features from the Sonar tables help to explain the most of the final TechDebt they generate. Thus, in the end we will have a model that may be able to predict the TechDebt value depending on specific features from the Sonar tables metrics. Another interesting goal we have is to be able to predict which kind of issue (code smell, bug or vulnerability) will be raised depending on these variables.

As we just said, the focus of this project will be on the structure of the codes thus we are going to study all the issues that were collected on the technical debt dataset. These issues are mostly the ones collected by SonarQube, metrics such as the number of lines of code and code complexity, compliance rules.

In order to detect if the issues actually can influence the behaviour of research projects and generate future issues we are going to compare the evolution of the SonarQube metrics and the commit that was uploaded to the repository. As the SonarQube table has collected a high number of metrics we will have to perform a selection of variables of the table to actually use data that can be significant for our analysis.

We want to find if the non meticulously written lines of code in a development project will cause issues in the future that will take a lot of time to solve or if those “mistakes” will be the same as a perfect code when trying to fix a bug. One variable that we especially want to take into account is the amount of time between commits is used on fixing a bug on a code with code smells and other imperfections and a well-written code. Another variable that we want to focus on is the difference in the code between commits on issues that were caused only by structural failures.

The objective is to find which are the most determinant attributes that can affect the performance of the project after some time in order to help future developers to take these factors into consideration. We would also like to present solutions to the issues, find the causes (not in the code but in the way of developing it itself) and find a way to fix them. If it were the case that the metrics collected by SonarQube could not predict in a proper way if a future issue on the development process is going to appear we will try to find the reason why they are not explicative enough.

3.1.4. Project plan

While working on this project we will follow the CRISP-DM phases, these being *Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation and Deployment*.

Phase 1	**Business understanding**
Description	In this first phase, we will determine the business objectives, assess the situation, determine data mining goals and produce the project plan. This is the initial phase, thus, we will define the project, its goals and its plan. It is important that we have a clear idea of what we want to do as this phase will condition the rest of the project. Nevertheless, we can modify some aspects of this phase later on.
Duration	1 week.
Constraints	None.
Resources	None.
Deliverables	Initial delivery. Delivery of the project goals and project plan. Initial presentation.

Phase 2	**Data understanding**
Description	After having a clear idea of what we want to do, we can start collecting the initial data, describing the data, exploring the data and verifying data quality.

	In this phase we will get to know the data we are going to work with, and we will come up with a plan for preparing the data and getting it ready for modeling.
Duration	2 weeks.
Constraints	We need to be able to access the database.
Resources	Computers with python installed.
Deliverables	None.

Phase 3	**Data preparation**
Description	Next, we will prepare the data. That is, selecting the data that we need, cleaning, constructing, integrating and formatting data. At the end of this phase, the result should be the data that we need, cleaned and fully ready and optimized for modeling.
Duration	1 week.
Constraints	Data understanding.
Resources	Computers with python installed.
Deliverables	None.

Phase 4	**Modeling**
Description	This phase will be the longest and most important, as here we will develop the model for pattern detection, the goal of this project, using the data that we have prepared from the previous phase. Here we will select the modeling technique, generate the test design, build the model and assess the model.
Duration	3 weeks.
Constraints	Data preparation.
Resources	Computers with python installed.

Deliverables	None.

Phase 5	**Evaluation**
Description	Once we have the model done, we will need to test it. In the case that the tests we do return results that we are not content with, we could return to the previous phase and change the model. In this phase we will evaluate the results, review the results and determine the next steps.
Duration	1 week.
Constraints	Modeling.
Resources	Computers with python installed.
Deliverables	None.

Phase 6	**Deployment**
Description	Once we have the results that we want for the model, we will begin with the last phase, deployment, consisting of planning of the deployment, planning the monitoring and maintenance, producing the final report and reviewing the project.
Duration	1 week.
Constraints	Evaluation.
Resources	None.
Deliverables	Final delivery. Delivery of the final report, the presentation slides and the final presentation.

- Initial assessment of tools and techniques. To do everything we need to do on this project we expect to use the python programming language through Google Colab. To store the files, this report, and more, we will use Google Drive. On top of all this, we will probably work using SQL at some point of the project.

3.2. Data understanding

3.2.1. Initial data collection

In order to work better with the dataset, we decided to download the tables as csv files directly using the following link. This way we can directly work with the tables opposed to having to deal with a dataset and a whole python library needed to navigate through it. At the end though we will still use a python script to explore the data in depth for an exploratory analysis to extract conclusions on the data that we are going to use.

3.2.2. Data description

Technical Debt Dataset is a curated dataset containing measurement data from four tools executed on all commits to enable data scientists to work on a common set of data and thus compare their results. In total there are 10 tables of data in csv format, but the most interesting ones for our project are the SonarQube-related tables as we will detect patterns that are produced by the SonarQube issues and find out the impact they have in the Technical debt in means of time and resources.

What follows is a description of all SonarQube-related tables:

SONAR_ANALYSIS.csv

This table contains the analysis keys to join with the SONAR_MEASURES and SONAR_ISSUES tables as well as the analysis date and the revision (i.e. commit hash) that it belongs to. This table allows the integration of the different SonarQube tables with the Git information offered by tables like GIT_COMMITS.

SONAR_ISSUES.csv

This table lists all of the SonarQube issues, as well as the anti-patterns and code smells detected in the analysed projects. The attributes of a SonarQube issue are:

1. **COMPONENT**
   File containing the issue.
2. **RULE**
   SonarQube rule that has been triggered.
3. **TYPE**
   Type of the issue, which can be Bug, Vulnerability or Code Smell.

4. **SEVERITY**
   Severity of the issue, which can be Blocker, Critical, Major, Minor or Info.
5. **STATUS**
   After creation, issues flow through a lifecycle, taking one of the following statuses: Open, Confirmed, Resolved, Reopened or Closed.
6. **EFFORT**
   Estimated effort required to fix the issue.
7. **CREATION_DATE**
   Creation date of the issue.
8. **START_LINE**
   Start line of the issue in the file.
9. **TAGS**
   Custom tags of the issue.

SONAR_MEASURES.csv

SonarQube is one of the most common open-source static code analysis tools for static quality analysis. This table contains the different measures SonarQube analyses from the commits. The measures analysed correspond to:

1. **vulnerabilities**
   Number of vulnerability issues.
2. **security_rating**
   A = 0 Vulnerabilities
   B = at least 1 Minor Vulnerability C = at least 1 Major Vulnerability D = at least 1 Critical Vulnerability E = at least 1 Blocker Vulnerability
3. **security_remediation_effort**
   Effort to fix all vulnerability issues. The measure is stored in minutes in the DB. An 8-hour day is assumed when values are shown in days.
4. **code_smells**
   Total count of Code Smell issues.
5. **sqale_index**
   Effort to fix all Code Smells. The measure is stored in minutes in the database. An 8-hour day is assumed when values are shown in days.
6. **sqale_debt_ratio**
   Ratio between the cost to develop the software and the cost to fix it. The Technical Debt Ratio formula is:
   Remediation cost / Development cost
7. **sqale_rating**
   Rating given to your project related to the value of your Technical Debt Ratio.
8. **duplicated_lines_density** = duplicated_lines / lines * 100

9. **duplicated_lines**
   Number of lines involved in duplications.
10. **duplicated_blocks**
    Number of duplicated blocks of lines.
11. **duplicated_files**
    Number of files involved in duplications.
12. **Complexity**
    It is the Cyclomatic Complexity calculated based on the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1. This calculation varies slightly by language because keywords and functionalities do.
13. **cognitive_complexity**
    How hard it is to understand the code's control flow.

SONAR_RULES.csv

This table lists the rules monitored by SonarQube. The features of this table are:

1. **NAME**
   Summary of the rule that serves as its name.
2. **DEF_REMEDIATION_FUNCTION**
   Remediation function used to estimate the effort to solve the issue.
3. **DEF_REMEDIATION_BASE_EFFORT**
   Estimated time to solve the issue.
4. **PLUGIN_NAME**
   Family of rules where the rule belongs to.
5. **PLUGIN_RULE_KEY**
   Rule identifier.
6. **TYPE**
   Type of the issue, which can be Bug, Vulnerability or Code Smell.
7. **SEVERITY**
   Severity of the issue, which can be Blocker, Critical, Major, Minor or Info.
8. **SYSTEM_TAGS**
   Tags used by SonarQube to better group the rules.
9. **DESCRIPTION** Description of the rule.

3.2.3. Data exploration

In order to perform a good exploration on the data we elaborated a python notebook. Since we already know the structure of the tables of the technical debt database we

wanted to give a first look at the actual table values in order to see which variables can be useful for our project goal.

There was nothing useful on the “Projects” table as it only contains information about the project key, the git-hub repository and some other links and keys to the Sonar and Jira tools.

We thought, at first, that the information on the “Jira_issues” table will be useful for our project as it contains information about the issues collected with the Jira software. It turns out, though, that the Jira issues are more wide than the SonarQube ones and do not focus only on the code measures as SonarQube does. The table contains the key of the Jira issue, the id of the project as well as the creation resolution and the commit date. The column hash is the one that connects with the commit table that can help us to track the issues in each commit.

The “Sonar measures” table is the one that will help us the most as it contains all the metrics obtained with the SonarQube software. The measures taken go from the complexity (complexity in file, in classes, in functions, etc) to the number of critical issues or the number of violations. We could see that there were a lot of NA’s in the data so we have to study what to do with the data. We can either do an estimation to fill the gaps or we can remove them. The tables “Sonar Issues” and “Sonar Analysis” also give us information about the type of issues encountered. In “Sonar Issues” we can see aspects like type of the issue, the severity, the resolution the debt generated and even a message whereas in the “Sonar Analysis” we can see if the issues have been analysed and revisited. As all the tables contain information about SonarQube issues the three tables are connected by the project _id.

The table “Git Commits” contains basic information about the commits performed and will be useful in the future when we intend to find patterns of the SonarQube issues that relate to these commits. One interesting variable in this table is the “commit message” which can be used to find any commit that solves issues related to the structure of the code. This table can be joined with the “Sonar” tables by the hash.

The “Refactoring Miner” table gives us information about the changes performed in the code and can actually be helpful in our goal to find patterns in the code that can generate future problems and obstacles in a project. We have yet to study the meaning of each refactoring type to see if it can actually help our goal. This table can be joined again with all the other tables by the commit hash.

Finally the “Szz Fault Inducing Commits” table gives us information about the fault fixing commit and the fault inducing commits and we do not know yet if we are going to use this information in the future or not.

3.2.4. Data quality

As the creators of the Technical Debt Dataset mentioned in their paper, some commits couldn't be analyzed properly since Ptidej or SonarQube raised some exceptions when a commit did not compile correctly. That's why they created a table called COMMITS with all the information concerning all the commits while in the other tables they only stored the ones that had a successful analysis without errors. Having this fact in mind, we'll need to be cautious when reaching conclusions since tracking of the issues might not be accurate at all due to these missing values in the general tables.

3.3. Data preparation

In this step of the process we are going to prepare the original data available to keep the desired attributes and tables with the wanted format convenient transformations or aggregations. Once we have the data prepared, we'll be able to proceed with the modeling process.

3.3.1. Data selection

The Technical-Debt Dataset consists of 10 csv files. As our project focuses on a specific region of the dataset, most of these tables will not be needed to carry out our project.
So, in this part of the data preparation process, we decide which of these tables we need. On top of that we will also select the columns that are interesting to us and leave out those which are not.

The first logical step is to select which tables we are going to use. As we are focusing on detecting patterns from the Sonar issues, we consider that the tables that we need are "SONAR_MEASURES" and "SONAR_ISSUES". The reasons for that were explained in the Data Exploration section.

The "SONAR_MEASURES" table contains a lot of metrics collected with SonarQube for every commit in the projects. As SonarQube is a very robust tool, the quantity of metrics collected is too big for us to take into account every single one of them. Out of the 240 columns of the table we will only keep 36 variables (counting the two primary keys of course). We chose all those variables because we thought that they were the most interesting and explicative out of all the others but we know that it is possible that not all the variables have the same impact in the final technical debt.

You can find a list and a description of all the variables we chose on this google spreadsheet file. Even though you can find all the information on that spreadsheet we will explain a few of the variables of the sonar measures table and the reason behind their selection.

Our initial hypothesis consists in studying variables such as the technical debt, the type of issue and number of bugs using other variables. We want to show if variables like the number of duplicated lines, blocks or files as well as a non-time dependent variable (duplicated lines density) can have an effect on the technical debt. We are also going to use the number of rules violations and the complexity of the code.

The variables we are going to use from the “SONAR_ISSUES” table, which at the beginning had 23 columns, are only a few related the type of issue, the resolution, the debt it generated and some other variables that maybe we will use in other type of analysis like the message of the issue and some auxiliary variables to compute the size of the offset.

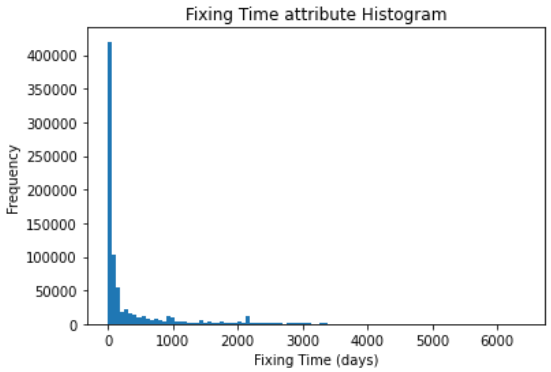
3.3.2. Data cleaning

Once we have a clear idea of which variables we want from each table, we can start the cleaning process. That means, removing all the columns we don’t need from the tables and transforming those which we will use if necessary.

As mentioned in the previous section, we will select only the desired columns. Then, we will only pick those features and leave the rest by removing them. During this process we also analyze the values from the different features that we keep by doing plots and other visualizations. We also check for the missing values of some records, which we finally discard as explained in the Data Construction step due to their low representation in the entire table.

3.3.3. Data construction

During this phase, new attributes from the different variables of the Sonar tables are derived or simply the original data column is transformed into the format desired to ease the modeling process that will be carried out later on.
In this case, an attribute that contains the needed fixing time for a Sonar issue in the table “SONAR ISSUES” is derived from the original attributes “CREATION_DATE” and “CLOSE_DATE”. As expected, this value can only be computed for records that have “FIXED” as their “RESOLUTION” column, in other words, issues that have not been resolved can’t have an attribute “FIXING_TIME”. The proportion of fixed Sonar issues is about 77.96% of the total table, we go from 1,024,614 rows to 798,745. Then, to compute the derived attribute “FIXING_TIME” we filter the original table to preserve the ones that have been closed. To be able to subtract both dates, we need to convert them from string type to timestamp class with the correct format.
In the following figure we can observe the distribution of needed days to fix a Sonar issue.



Other decisions that have been taken are to discard the records from the Sonar Issues table that contained an empty value or string (NaN) for a column that we were going to preserve. Thus, we got rid of approximately 8.8% of the records from the fixed Sonar issues, which hadn't got a value in their "DEBT", "START_OFFSET" or "END_OFFSET" column.

Regarding the Sonar Measures table, we have selected the features mentioned above, and from these we've discarded the records that had a non-available value for at least one column. This is not a major problem because the discarded rows are a total of an insignificant 0.05% of the overall table.

The resulting datasets of these transformations and selections are the "sonar_issues_fixed" and "new_sonar_measures".

3.3.4. Data integration

Having cleaned and prepared both tables ("sonar_issues_fixed" and "new_sonar_measures"), we here proceed to integrate them.

Before merging the tables, some renaming of they key columns had to be done, as we are merging the "CREATION_ANALYSIS_KEY" column from "sonar_issues_fixed" and the "analysis_key" from "new_sonar_measures". In order to simplify this, we just called both of them "key".

Then, once we have our key columns, we easily join the tables with the *merge* function from *pandas*. The result of this join is explained in the following section.

3.3.5. Dataset description

The resulting dataset obtained from joining our two previous tables is the dataset which we are going to use in the modeling part. It consists of 697,602 rows and 45 columns. The names of these columns are:

'key', 'ISSUE_KEY', 'TYPE', 'SEVERITY', 'RESOLUTION', 'DEBT', 'MESSAGE', 'START_OFFSET', 'END_OFFSET', 'FIXING_TIME', 'project_id', 'complexity', 'file_complexity', 'duplicated_lines', 'duplicated_blocks', 'duplicated_files', 'duplicated_lines_density', 'violations', 'blocker_violations', 'critical_violations', 'major_violations', 'minor_violations', 'info_violations', 'open_issues', 'missing_package_info', 'development_cost', 'sqale_debt_ratio', 'new_sqale_debt_ratio', 'code_smells', 'bugs', 'reliability_remediation_effort', 'reliability_rating', 'vulnerabilities', 'security_remediation_effort', 'security_rating', 'lines', 'ncloc', 'ncloc_language_distribution', 'classes', 'files', 'directories', 'functions', 'statements', 'comment_lines', 'comment_lines_density'.

We will apply PCA to this table, so the number of columns could decrease, but this will be explained in the modeling part.

3.4. Modeling

3.4.1. Modeling assumptions

3.4.2. Test design

3.4.3. Model description

3.4.4. Model assessment

3.5. Evaluation

3.5.1. Assessment of data mining results with respect to business success criteria

3.5.2. Review of process

3.5.3. List of possible action

3.6. Deployment

3.6.1. Deployment plan

3.6.2. Monitoring and maintenance plan

3.7. Final executive summary

- Summary of business understanding: background, objectives, and success criteria
- Summary of data mining process
- Summary of data mining results
- Summary of results evaluation
- Conclusions for the business
- Conclusions for future data mining

3.8. Good engineering practices

When carrying out the project we are expected to apply good engineering practices when developing software systems with machine learning components. For that reason, we are required to apply three different practices that are considered as useful in order to proceed correctly when working in this field. Next, the three different practices are explained.

3.8.1 File structure and replication package

The first good practice that we will employ is related to the organization, more specifically, it consists of following a file structure by replicating a package called Cookiecutter Data Science. This template is helpful to foster the correctness and reproducibility of the code that we are expected to achieve and that was listed in the requirements of the project. The resulting software, duly structured, will be uploaded to a repository on GitHub so we can keep track of the changes and modifications made to the different files.
Thanks to this practice we will be able to collaborate more effectively and easily on the analysis and feel confident in the conclusions at which the analysis arrives.
On the other hand, starting to use this file structure template eases the search of documents inside the directory since every file is duly classified. Moreover, if it is the first time that you visit that repository, you can expect which types of files will you find inside every folder.

3.8.2 Second good practice

The second good practice is related to the code quality, we will be using static analysis to check the code quality of our results. The intention is to avoid the introduction of code that is difficult to test, maintain or extend. Since high quality code is easier to understand, test, maintain, reuse and extend we will be making use of static analysis tools to obtain that. This good practice is interesting because ensuring high code quality we can avoid the

introduction of defects into the code and enable team members to become productive quicklier and more easily thanks to the correctness of our code.
In this case, the code analysis will be done using linters. A linter is a tool that can find undesirable patterns in program code and report them back to the programmer. These linters can be activated in a code editor or they can simply be run on the commandline.
To do this, we'll be using mllint, a linter for Machine Learning projects. It is a command-line utility to evaluate the technical quality of projects written in Python by analysing its source code, data and other configuration. Basically, mllint will help us to assess the quality of our ML project and receive recommendations on what aspects we should focus on improving.

3.8.1 Third good practice

The last good practice is related to the team communication, more concretely implies to share status and outcomes of experiments within the team. In other words, when performing different training models or experiments, the results should be communicated within the group members in order to know which advances or improvements are being made and in which direction. Even which way should be discarded if it is found out that an approach is not the accurate one. Behaving in this manner, since we won't be doing unnecessary work, we will save more resources and time than if we hadn't applied that good practice.
As we were mentioning, good communication within the team is important to facilitate the transfer of knowledge, the peer review and the model assessment. Every member of a team has different ways of working and managing experiments related to the data, then, sharing their views and thoughts on that can be extremely beneficial for the success of the whole team.
In the end, what is important is to adopt a common way of logging the results obtained from each member of the team and the progress it's being made. Obviously, these results should be understandable and accessible to all the representatives of the group.