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 will 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. This TechDebt is collected as a variable in the Sonar Measures table, its name is “new_sqale_debt_ratio”. More concretely, this variable contains the ratio between the cost to develop the code changed on New Code and the cost of the issues linked to it.

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.

Our success criteria is to obtain low errors for the models developed. Thus, our models will be able to predict the TechDebt for new records that could be new projects that want to prevent these issues. We will be using errors like the Mean Absolute Error or the RMSE that measure the difference between the predicted values made by a model and the observed values in the dataset.

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.

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.

More detailed data exploration can be checked in the Data Exploration notebooks in the repository. However, we will be using as target variable the “new_sqale_debt_ratio” instead of the “DEBT” one because of their range of values.

```
sonar_issues['DEBT'].describe()

count    992704.000000
mean         18.539828
std         138.776940
min           1.000000
25%           2.000000
50%           5.000000
75%          14.000000
max       92297.000000
Name: DEBT, dtype: float64
```

```
sonar_measures['new_sqale_debt_ratio'].describe()

count    66698.000000
mean         3.391989
std          1.308268
min          0.000000
25%          2.753026
50%          3.201967
75%          3.772247
max         80.606061
Name: new_sqale_debt_ratio, dtype: float64
```

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.
In our case, performing the data exploration we have checked for the quality of the data tables we will be using for our analysis. We have observed that few records contain some missing value in one of their variables. Nevertheless, this is not a major problem because the records with at least one missing value are insignificant and they represent an extremely low percentage from the whole data. These proportions are better detailed in the Data Preparation section with numeric values and how we tackle this problem. Our decision has been to delete these rows from the data.

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.

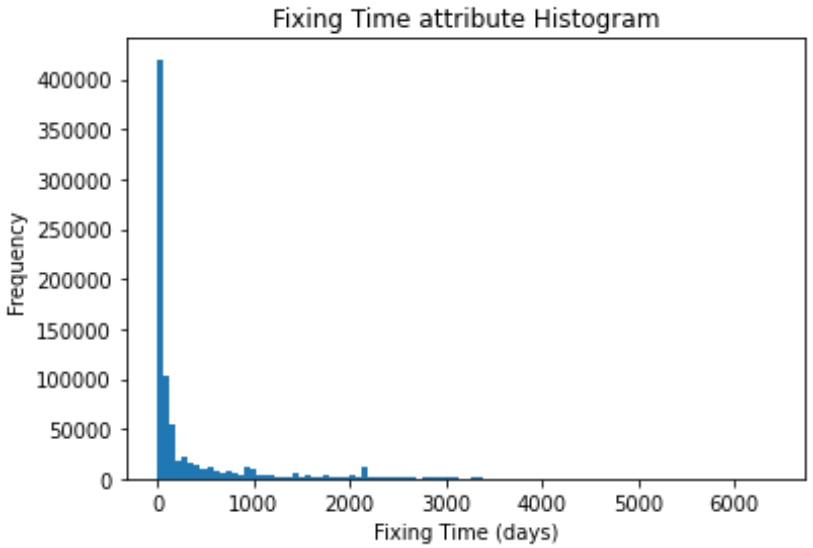

Fig.1. Fixing Time Histogram

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

The modeling assumptions made are related to the models applied to satisfy our data mining goal. Since we want to be able to predict the technical debt for a given data related to some project, we developed a regression model after performing PCA. In addition, we developed a Ridge Regression model due to the high correlation present in the Sonar measures data that we work with. Because of these models, they can only treat numerical data. We tackle this issue by encoding categorical variables like the type of issue and its severity using One Hot Encoding. Once this transformation is done, we can proceed with the modeling steps. Moreover, there can not be missing values in the attributes of the data columns. But this is not a problem anymore because we have previously deleted the very few records that had missing values during the Data Preparation phase.

3.4.2. Test design

Given that we are working with regression models, we need a mechanism to measure the quality and performance with the data they are trained with. As the predicted output is a value that tries to predict the possible TechDebt, we will be using error rates that take into account the difference between the predicted values made by our models and the observed values present in the corresponding column of the Sonar data. Such error measures are the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE), they are the average of the errors. For that reason, we would like to make models with low error values, which turn out to be more accurate when predicting the TechDebt for a project.
For this test step, we need to split the data in two parts, training and testing, the first one to build the model based on this data and the second to estimate the quality and performance on new data. Our decision has been to make this separation randomly by using seeds and fragmenting 70% of the data for the training and the remaining 30% for the testing step.

3.4.3. Model description

In order to achieve the goals stated in the Data mining and business goals sections, which is to find the impact of the sonar metrics in the technical debt, we will implement several models using the data collected in the sonar tables.

3.4.3.1. Principal components analysis with ‘sonar measures’ table

Our first approach was to perform a PCA with one of the datasets we generated, that is a selection of columns of the “sonar tables” (all the columns collected and the reason behind that is found in the data preparation section).

The first PCA we made was on our “new_sonar_measures” which is a selection of columns of the “sonar_measures” table that we believed were the most important. The aim of doing the PCA on this new dataset instead of the original one is to avoid the use of the large number of columns that the original dataset had and using only relevant variables that can explain what we are looking for. For this PCA we used only numerical variables and although there are techniques to use PCA with categorical variables too, we did not use them as the number of categorical variables was very small.

Once the PCA was computed we proceeded to analyze the results. We performed a heatmap to show how each variable was explained in every component. We did not see any conclusive results, on the first components almost all variables are represented in a similar way and on the last components there are variables that explain more than others, which is normal because those components explain such a small variability that they can focus on a small number of variables.

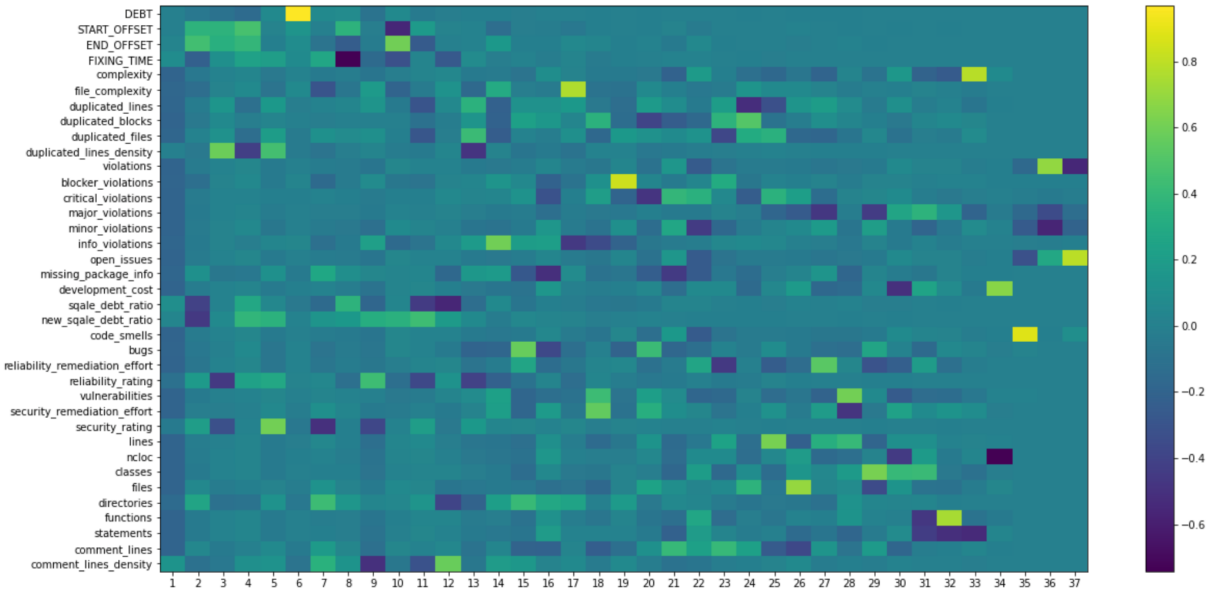

Fig.2. Heatmap components of PCA using “new_sonar_measures” table

After the light exploration of the components using a heatmap we want to know how effective the PCA was on our data and how much variability is explained by each component of the analysis. On the *”Percentage of variance explained by every component”* (Fig.3.) we can see that the first components explain almost the 100% of the variability of our data, the first component explains about a 70% of the variance, from then all the other components explain only a small percentage of it, in fact, we

can see that starting from the 14th component the percentage of variance explained for every component is lower than 1%. Following this plot we elaborated a *"Cumulative variance explained"* (Fig.4.), in this plot we can see that the 10 first components of the analysis explain about 96% of the variance. We are going to use those 10 first components in future steps in order to get a smaller dimension dataset, this will lead to easier computations and more consistent and robust results as we will work on a dataset that has almost all the information of our original dataset concentrated on only a few components. Doing this transformation will mean that we will lose a 4% of variability but this percentage is very small and we can work with this dataset by keeping in mind the error we made.

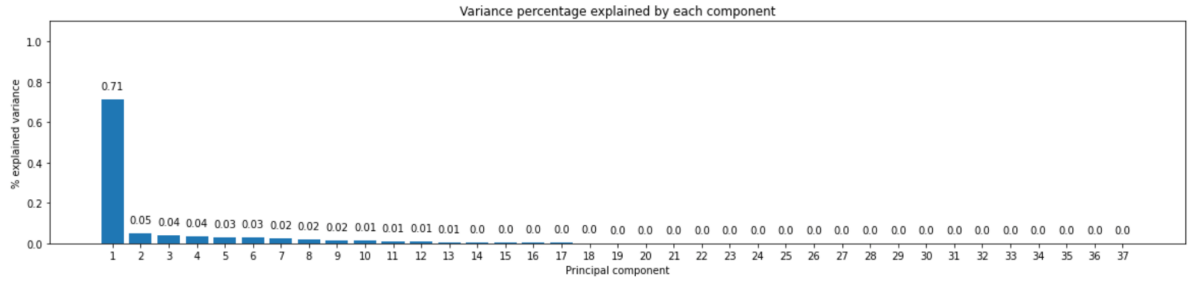

Fig.3. Percentage of variance explained by every component (sonar_measures data)

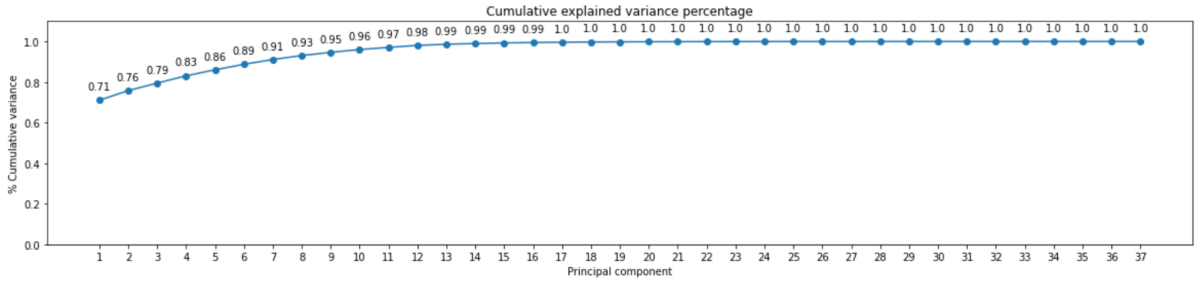

Fig.4. Cumulative variance explained (sonar_measures data)

The only problem with this PCA is that the fact that the first component explains such a high amount of variability means that our data is not very independent and we can explain a high amount of variability on the data with only a small number of components. In order to achieve a better PCA with more uniform components we need a less dependent dataset. We will show how to accomplish this in the next sections.

The next step after the principal component analysis is done is to elaborate a principal component regression. The objective of this regression is to be able to predict our target variable based on the results we got on our PCA analysis. In our case the target variable had to be related to the technical debt. At first we were not really sure which variable or variables to use as our target variable so we tried a few variables related to technical debt to compare results and choose the best one. The variables we were not sure could work as target variables were the *'new_sqale_debt_ratio', 'file complexity'* and *'code_smells'.* We took these variables

into account as they were the ones that were most related with the technical debt concept; *'code smells'* is a variable that keeps track of the number of code smells in our dataset; *'file complexity'* keeps a record of the the added complexity in the file at every new commit and *'new_sqale_debt_ratio'* is a measure that keeps track of the added sqale debt ratio at every commit, 'sqale_debt' is a measure created by sonar and it is a really good factor to assess aspects of the technical debt like the complexity or the quality of the code.

What we will also achieve with this Principal Components Regression is to find the correct number of components that we will use to transform our dataset. We will first decide which variables behave the best as a target variable (by evaluating the error, the shape of the plot and the estimation of the regression) and once we find the correct target variable we will study the correct number of principal components by performing a cross validation on them.

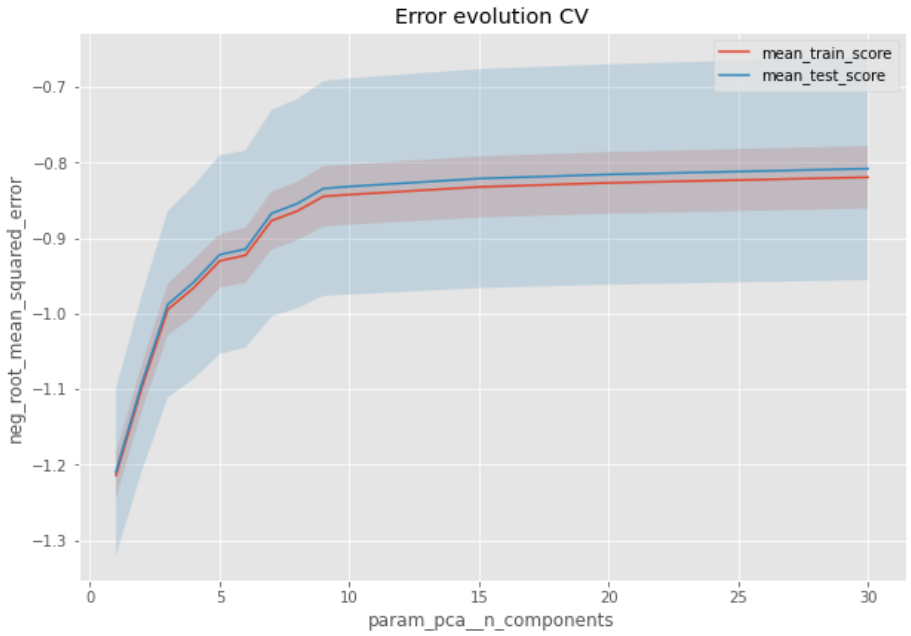

Fig.5. Results of grid search with cross validation

As we can see in the figure above, after doing a grid search with cross validation to find out the correct number of parameters, which in this case is the number of principal components, we can see that we achieve a good regression with a small error with only 10 components. The figure represents the error we achieve on our principal components regression using different values for the number of components. We can see both the training and test errors, as well as their confidence interval. The error we used to measure the difference between the predictions and the actual value of the debt was the rooted mean square error. It is obvious that as we have more components used to do the regression, we will have less error, given that the regression will have more values to fit the model, but we

can see that after 10 components used, the error actually starts to grow in a very slow manner, the improvement on the use of more components is that small that we only need 10 components in order to do a good prediction.

3.4.3.2. Principal components analysis with all “sonar data”

After doing a PCA analysis with only numerical data from one of the sonar tables we had available we have realized that the data we had in that table had a very high correlation factor, that is why we are going to do a second principal component analysis with more sonar data. This time we are going to use the columns from the *“sonar_measures”* table and “sonar_issues table”, this will give us more variables to work with and we expect a lower correlation factor between the variables of the dataset. This dataset can be found in our repository as *“sonar.csv”*

First of all, as our new dataset contains not only numerical variables but categorical variables also, we need to transform the categorical variables into a numerical representation using the one hot encoding technique.

With this new data, we do PCA, minimum squares and PCR over again and see whether there has been an improvement.

On the new heatmap (Fig.6), we can see on the first components that, for example, “TYPE_CODE_SMELL” stands out, which as we said, didn’t happen with our previous data, where all variables were represented in a similar way.

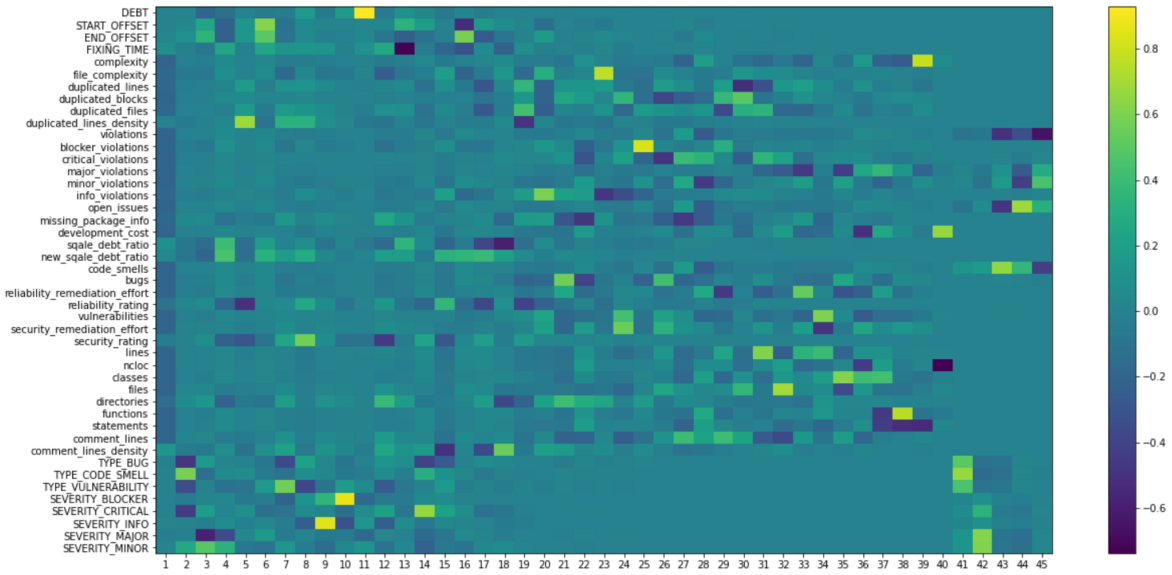

Fig.6. Components’ heatmap (one hot)

After some exploring of the components with the heatmap, we see how much of the variance is explained by each component and compare it with our previous tests. As

we see in Fig.7., the variance explained by the first component has dropped from 71% to 58%, while the rest of components have gained importance.

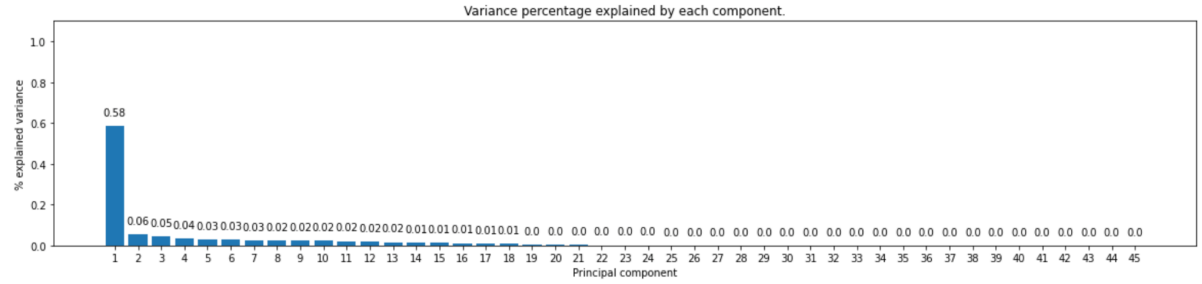

Fig.7. Percentage of variance explained by every component (one hot)

Now, looking at the cumulative explained variance of the components, we observe that in order to get to 96% of the explained variance, we need to take into account the first 15 components, as opposed to the only 10 that we needed before.

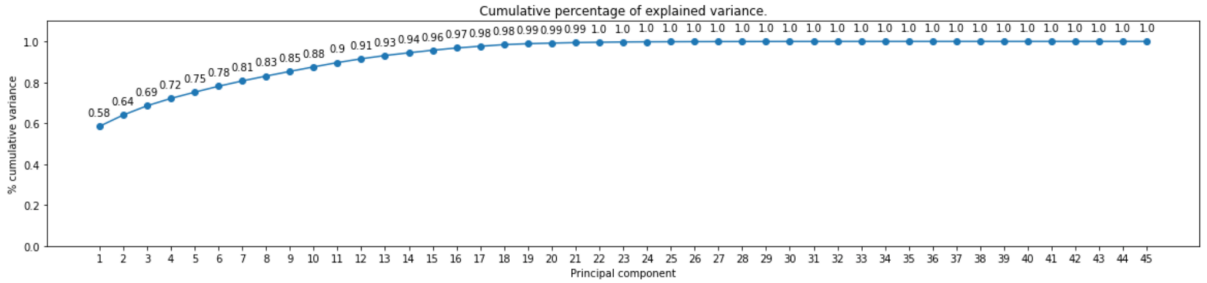

Fig.8. Cumulative variance explained (one hot)

3.4.3.3. Multilayer Perceptron to predict error type

Another objective stated in the Data Mining and Business goals was to be able to predict the error type of the sonar issue based on other sonar metrics. To that we are going to use a classifier using neural networks, more precisely we are going to use a multilayer perceptron.

We have chosen a multilayer perceptron to classify the error type or the sonar issue because we wanted a non-supervised procedure that was able to predict with a low percentage of error the type of issue we commit. The use of neural networks is an unsupervised method that, with the right parameters, achieves very good results, with a very low error and works well with any type of dimension.

The first thing we have to do is to define the data we are going to use, in this case we are going to use the data from the *“sonar.csv”* file we created (it contains columns form the “sonar issues” and “sonar_measures” tables that we found interesting). The target variable we are going to classify this time is the *“type”* column but we could actually have used other variables like the *“severity”* of the issue, as the architecture used for all the predictions would be the same, we would only need to change the

target variable. We have eliminated some of the categorical variables we had in our *“sonar-csv”* as neural networks work with numerical data only. Because of the same reason, we had to transform our target variable into a numerical variable, we changed the three error types into 0,1,2 values.Even if the variables are now numbers the multilayer perceptron keeps seeing them as a categorical variables as we have to label them as classes.

Then we have to divide the dataset into training and test sets. The training set consists of 70% of the data and the test set contains the remaining 30%. Once the division is done, we can start training the network, update the weights, repeat for each batch an epoch and evaluate the results.

The network consists of 37 input neurons (we have 37 variables in our entry dataset), a hidden layer with 128 neurons and an output layer with three neurons (the number of classes of our target variable).

3.4.4. Model assessment

3.4.4.1. Principal components analysis with ‘sonar measures’ table

The first thing we do is look at the correlation between numeric columns. The results show a lot of highly correlated variables, which could cause a problem while performing regression.
We then adjust two linear models, one with the predictors and another one with some of the components we got from PCA. To be able to evaluate them, we split the models into train (70%) and test (30%). Once we have our models, we start modelling.

We first try least squares, and get a test error (rmse) of 46.339700523860635.
After this, we move on to PCR, creating a pipeline to combine it with linear regression. We first evaluate the model including all components, and get a test error (rmse) of 46.339700523860635.
Then, we perform search by grid search with cross validation and obtained the following results:

	param_pca__n_components	mean_test_score	std_test_score	mean_train_score	std_train_score
9	35	-43.146723	3.145347	-43.251165	0.800481
8	30	-43.156466	3.143974	-43.261069	0.799808
7	20	-43.246556	3.134308	-43.350964	0.797417
6	15	-43.313969	3.130715	-43.418237	0.796400
5	10	-43.334815	3.127786	-43.439454	0.795495
4	8	-43.410351	3.119344	-43.514304	0.793075
3	6	-43.490775	3.113649	-43.594197	0.791861
2	4	-43.492906	3.114307	-43.596555	0.792028
1	2	-43.555424	3.108452	-43.658815	0.790412
0	1	-43.575975	3.109215	-43.679475	0.790816

Fig.9. Results of grid search with cross validation

So, pretty bad results overall.
Lastly, we try training the regression model after PCA with scaling, and we again get a bad result (test error = 46.677738428966215).

3.4.4.2. Principal components analysis with all ‘sonar data’

We then look at the correlation between numeric columns once more and get the same results as before, highly correlated variables.

After this, we split our new data as we did before and perform the least squares. This time, the test error (rmse) is 0.8738696816951032, which is quite low and a huge improvement compared to the previous experiment.

We then move on to PCR, this time, similar to what happened with minimum squares, we get a low test error (0.8738770022725192). Which confirms we are on the right track.

We then again perform a search by grid search with cross validation, but this time we get good results:

	param_pca__n_components	mean_test_score	std_test_score	mean_train_score	std_train_score
9	40	-0.893936	0.019255	-0.893922	0.004815
8	30	-0.902572	0.018962	-0.902582	0.004740
7	20	-0.911313	0.019368	-0.911445	0.004831
6	15	-0.933302	0.018991	-0.933419	0.004628
5	10	-1.017101	0.012053	-1.016219	0.008217
4	8	-1.027124	0.014361	-1.027143	0.003696
3	6	-1.034170	0.014273	-1.034222	0.003616
2	4	-1.052296	0.014221	-1.052370	0.003543
1	2	-1.080177	0.013637	-1.080251	0.003415
0	1	-1.082915	0.013541	-1.082990	0.003380

Fig.10. Results of grid search with cross validation (one hot)

We conclude that 4/6 components seem sufficient, and we plot the cross validation results for each hyperparameter to see the evolution of the cross validation error:

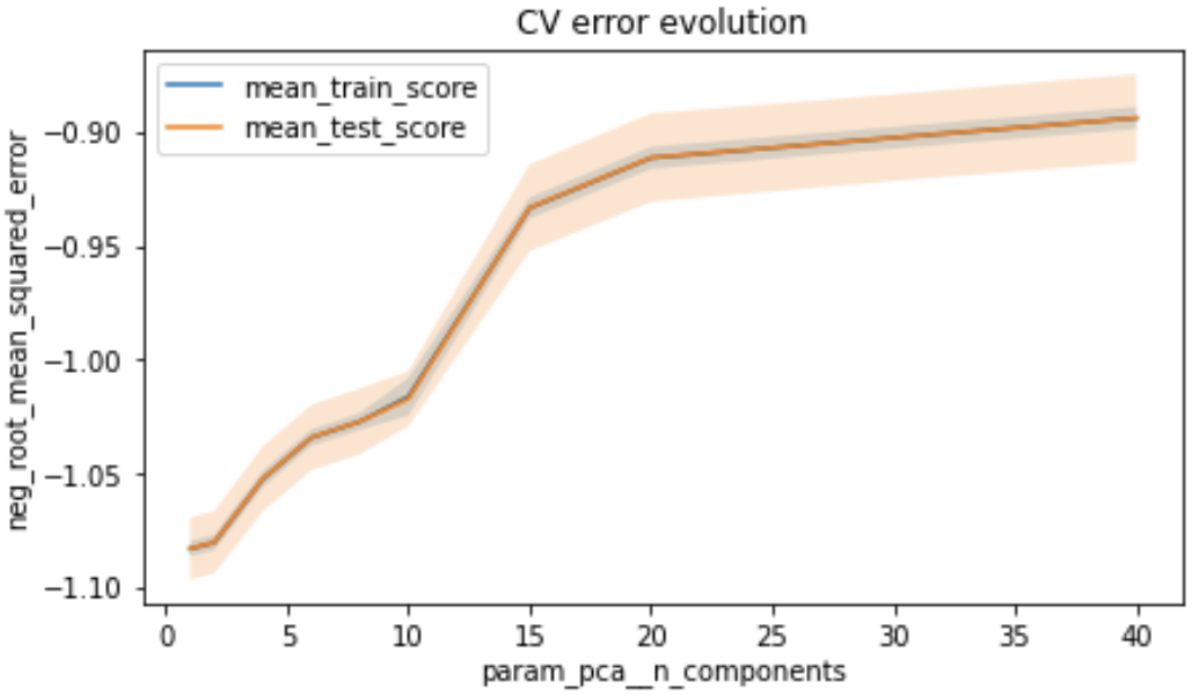

Fig.11. CV error evolution (one hot)

Finally, we train a regression model after PCA with scaling and the test error this time is 1.0300173587437191.

Given that the explanatory variables are highly correlated and they are very dependent, we also developed a Ridge Regression model, which performs better in these cases. We use all the data in our generated datasets, including the categorical variables encoded using One Hot Encoding. We performed a Grid Search using Cross Validation for the hyperparameters tuning, this is only one hyperparameter, the alpha that controls the weighting of the penalty used in the loss function. Our best

results were obtained using an alpha equal to 0.99, which can be considered as 1. The MAE for an alpha with value 1 is around 0.352, which is a low error.

3.4.4.3. Multilayer Perceptron to predict error type

The best parameters we found for the execution of the neural network were a batch size of 200, since smaller values generated an overfitting on the target variable; ten epochs to arrive to more robust results, a learning rate of 0.001 which we considered was a good value that allows small changes in the optimization process. The function we used to evaluate the loss of the classification process was the negative log likelihood.

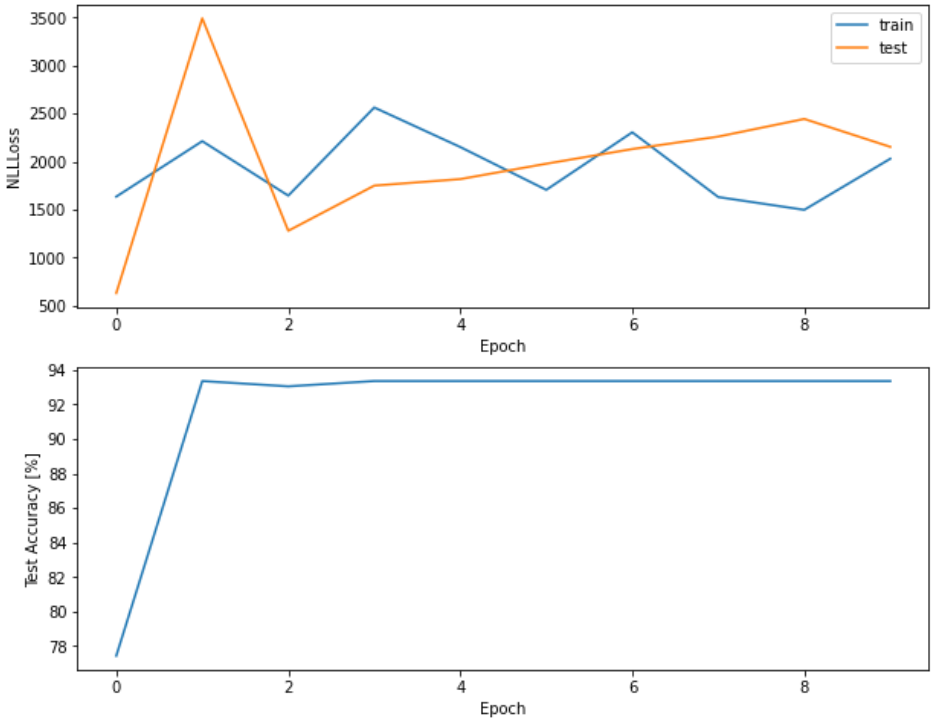

Fig.12. Loss and accuracy of the neural network.

Once all the training was done we could extract these two graphics, the first one is the error evolution of the neural network through the epochs. We can see that the values of the error are quite big but that is because of the magnitude of the batch size number. Regarding the accuracy of the model, we can see that we reach a good 93% which is a very good result. At first, on the first epoch, there is a very small accuracy but from then on we can see that the accuracy stabilizes.

3.5. Evaluation

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

Our project goal was to be able to predict which type of issue may occur with the current status and measures of a project and also to predict the generated debt by the code. In addition to this, we wanted to achieve models with low error rates for the regression analysis and a high accuracy for the Neural Network model. Given the results from our models and their performance metrics, we can say that this point has been satisfied. For the MLP we have achieved a 93% of accuracy, which is quite high. For the regression models the error rates are the following ones:

Regression	Error	Type of error
Linear regression	0.890	rMSE
PCA numerical data	1.011	rMSE
PCA one hot encoding	0.874	rMSE
Ridge regression (α = 1)	0.352	MAE

Regarding these values, we can state that the project finally meets the initial objectives and that the models can be approved.

3.5.2. Review of process

We believe that the process has been done correctly, following the CRISP-DM model guide to obtain the best quality results and also for the correct development of the project to finally meet the initial requirements and goals. If we had more time to carry out the project, we would have dedicated more time to each phase of the whole process. Moreover, we would have tried more models and different techniques to enhance the results and business goals. However, we are satisfied with the results obtained.

3.5.3. List of possible action

In the future, we could try to find new data mining goals to enrich our solution with the information that JIRA reflects in the data tables, and analyze if the information that JIRA collects is more explanatory than the SonarQube measures. Furthermore, if we had some budget we would try to acquire more computational resources to be able to work with more variables in the tables that we had to discard, since perhaps, one of them is quite important when trying to predict the kind of issue raised or the generated technical debt.

On the other hand, we can close this project and move on to deployment.

3.6. Deployment

3.6.1. Deployment plan

The deployment of our model consists of the different notebooks where our models are trained and prepared with the variables indicated along this report. In these notebooks we can find the different analyses that have been done to achieve our business and data mining goals. At the end of the different models sections, we are able to call the respective function that allows us to predict the output of the TechDebt or the issue caused given a new record that can be invented or generated from analysing any project with SonarQube.

3.6.2. Monitoring and maintenance plan

We could keep track of our data mining results and the usage other researchers do with it. Afterwards, if some feedback or opinions are given by these users in order to improve our services, we would try to correct the errors and even increase the quality of our results. This monitoring could be performed by using the functionality that GitHub has to share ideas or possible issues with files or code in the repository. This way, we as developers, will be able to know what are the thoughts on our work and if some things had to be changed or considered to enhance the quality of the services that our project brings.

3.7. Final executive summary

The Technical Debt Dataset helps researchers to study the causes of the Technical Debt by establishing a ground truth. One of the tools that collects the data is SonarQube, which collects huge amounts of metrics. Another tool is Jira Issues, but we won't use it.

Our goal is 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. We want to predict the technical debt ("new_sqale_debt_ratio")

Our success criteria is to obtain a good accuracy (low errors) on the models that we train to predict the different metrics.

We first predicted the technical debt. For this we have trained a few regression models. The one that worked the best was the Ridge Regression model we performed on the one hot encoded data, which got an error of 0.352%. Then, we also predicted the issue type. For this, we trained a multilayer perceptron. As well, we could predict any other categorical variable with this method, but we'll stay on the issue type.

After training and testing the multilayer perceptron, for a high batch size we got a high loss, and setting the batch size to a lower number, we ran into overfitting. The final result was an accuracy of 93%.

In conclusion, it is fair to say that we successfully predicted the technical debt and the issue type.

Finally, one thing we could say for future data mining is that our work is replicable, we have a code file for each part of the project, which makes it easy to follow it through. That means that another data miner can use our code on his data only by changing a few lines of codes and also changing our variables to his variables.

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.