tecnun Universidad de Navarra

Proyecto Fin de Máster

MÁSTER EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

New features for Codex app

María Serrano Martínez

San Sebastián, abril de 2021

Pº Manuel Lardizabal, 13. 20018 Donostia-San Sebastián, Gipuzkoa
Tel. 943 219 877 · Fax 943 311 442 · www.tecnun.es

Contents

List of Figures

Abstract

This project is set within the context of the Mentor research line, inside the project of the Codex app. Codex is an online learning tool developed at Tecnun, which aims to become a reference tool for both students and teachers to improve their learning and teaching experiences. This project develops some new features for the app, more precisely: a functionality for enabling live time events and some algorithms for data analysis.

1 Introduction

1.1 Motivation

This project is motivated by the will of increasing the capabilities that the Codex tool offers. As an online tool aimed for education, it is natural that new features are added as different needs and opportunities are discovered. Covid-19 has shown the importance of having valuable material and practical resources for both students and professors to learn, practice and evaluate knowledge.

During the time that Codex has been up and used by students at Tecnun, several needs and new opportunities have been found, and this project's scope is solving two of those new functionalities.

The first of these is aimed at covering a need found by professors: interactively evaluating and solving exercises with students. In recent years, tools that allow professors to examine students in real time have been actively introduced in many subjects. These tools consist on a question bank that the teachers chose questions from and display them for students to see during class. The professor has the ability to display or take down the questions whenever they want, and that way they can set the time for answering them. These tools also have a real time evaluating feature included, and teachers can use it to display results during the activity. This allows them to see real quick whether the students understand the concepts or if they should just go on with other questions.

Codex does have internal question banks for professors, from where they now create the notebooks for students to answer. But a tool that enables them to engage in this kind of real time activities was missing. The requirements for the development of such a tool included the ability of selecting the already created items manually, and for these to be displayed on the students' notebooks, whether they were already opened by them or if they were entering the notebook later.

On the other hand, a new opportunity for developing new features was spotted: data analysis and machine learning algorithms for improving the teaching and learning experience. Everytime a student answers a question, data is being collected and saved on the Codex database. All this information can be used to learn about the student's learning practices and help them where needed. It could also be used to give professors information about their evaluation system, whether their questions are significant, etc.

1.2 Overview of new functionalities

This project has developed these functionalities and their implementation is the following.

Regarding the live elements functionality, it has been developed and implemented on the Codex app, and the result is a working tool that enables professors to easily activate items for students to answer during class (or whenever the professor considers it), and items being displayed live-time on students' notebooks. Any professor can designate a notebook as a "live notebook" and students will have some visual notes to see that the notebook is in "live mode" and, thus, new items may appear on their screen. The tool also gives the professor the option of seeing and displaying the results of the questions after evaluating the answers, and it also allows to see the students who did better on the questions (the ones who achieved a higher grade in the least amount of time). In the following points the technical resolution of this functionality is explained, along with several captions that show the final view.

On another note, a machine learning algorithm has been developed for predicting grades of students based on their past completed items and the information available from other years' students. This model is a linear regression model that takes completed items' grades as inputs and calculates next items' grades and final course grades as outputs. The model has not been implemented in the Codex app yet, but a proposal on how this could be done to achieve the maximum advantage of it is given. Apart from developing the model, a study of different possible applications for machine learning models is given in the following points, and it can serve as a starting point for future functionalities related with data analysis and machine learning algorithms.

2 Codex

2.1 Project summary

Codex is a project from the Department of Industrial Organization Engineering of Tecnun-University of Navarra, inside the Mentor research line. Its objective is to help both teachers and students achieve the best results during the teaching-learning process. For doing so, the project focuses on the research, analysis and development of online learning tools. These online tools offer different learning resources such as plain content, practice exercises, evaluation tests, simulators, etc.

In this context of online based learning apps, Codex aims to be the reference tool for teachers to create and edit online content and for evaluating students' knowledge.

2.2 Current application

Codex currently works as an online tool where professors can create question banks with multiple types of questions, such as multiple choice, short paragraph, numbers and even coding scripts. After introducing the evaluation method, all the answers from students get automatically evaluated and they can receive feedback on them (if the teacher has enabled this option).

Professor have control over many things: evaluating methods, timing for answering the questions, automatically randomly selecting questions for students, data from exercises being created randomly (so as to having different exercises for each student), etc.

After creating the questions (or items, as they will be called throughout the document), professors then create notebooks where they select which items are contained in. These notebooks are then made available for students to see and answer. There is also the possibility for notebooks to contain other notebooks within.

Students have a main page where they can access their different subjects, and inside each of these subjects, they find a list with all the available notebooks for them to complete. Once completed, they also have the possibility of reviewing them by accessing them through a different page on the application.

Whenever a student opens a notebook and send the answer to any of the items, it will receive back a confirmation message to know the answer was correctly received. Depending on the configuration that the professor has set for the item and notebook, the student will have the ability to answer the question several times or just one. Also, the tool allows teachers to enable or disable showing the grade to the students.

On a more technical note, Codex is developed following the MVC (model-view-controller) design, which separates the functions that each part of the application does. Whenever the user makes an action on the UI (User Interface), this triggers a controller in the back-end which manipulates the model and updates the view based on that model, and this new view is shown to the user.

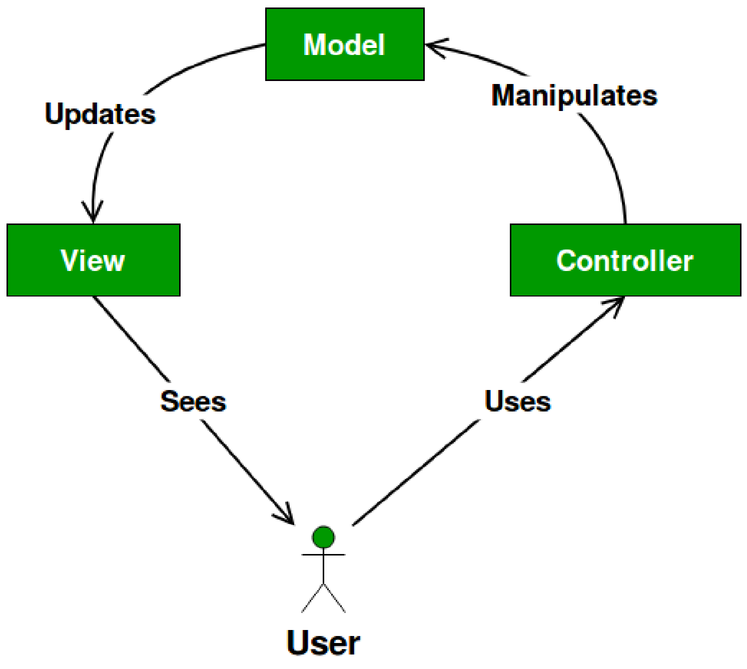


Figure 1: Model View Controller

The back-end of the application is developed using Java, and the front-end is done with HTML (for the structure), CSS (for the style) and JavaScript (for functionality). The development of the new functionalities has been done using these languages, for the live items component, and using Python for the machine learning model. This is more thoroughly discussed in the following points.

3 Live elements

3.1 Problems found and solution proposed

During the use of the Codex app, a few problems and desired missing tools where encountered regarding the possibility to add new items to a notebook once the students were already completing it. This could be due to two main reasons:

1. The teacher had forgotten to add the item in the first place. If the teacher adds it once a students has already opened the notebook, this student will not see the new item unless he closes the notebook and creates a new instance. The professor will have to tell the students to re-start the notebooks, which will make them loose their answers, so it is not a really efficient way to do it.
2. The teacher wants to engage in a dynamic activity with students, such as the ones with *Socrative* and *Kahoot*.

Socrative and *Kahoot* are online tools used by professors to test students in a quick way, with questions appearing on the screen as the teacher activates them. Both apps give instant feedback on results and participation, and in the last years they have been introduced in different subjects. They are easy to use and students find them entertaining and fun.

The problem is that, for professors using Codex on a daily basis, using either *Socrative* or *Kahoot* means having to replicate all the questions and items that they already have on their question banks in Codex. Besides, none of the mentioned apps support evaluation criteria that differs from true/false, multiple choice or simple text answers, so it impoverishes the multiple opportunities that Codex gives.

The solution is to create a new functionality, referred here to as "live items", that enables professors to use Codex in a dynamic way with their students. This functionality solves both problems tackled on the first paragraph. The requirements set for this new function are the following:

1. Easy to use, both for professors to add items and for students to see them.

2. Re-using as much user interface elements as possible, so minor changes are perceived by users and less code is introduced.

3. Adding the possibility to see the results of the answers instantly.

4. Having the possibility of adding or deleting items in live mode while the student has the notebook open.

The following points explain the technology used for achieving this and the actual implementation on Codex.

3.2 Technology used

The main issue found when regarding this problem is finding a way to notify the client (student) that changes have been made on the server side (new items have been added to the notebook). There are two main options to solve this:

1. Option 1: have the client ask the server for changes (items added or deleted) every X seconds through an AJAX call. The server will send a true or false response to the client. This solution is not the most effective, as most of the time the client will be making requests that are not necessary. Furthermore, if the time set between client requests is high, some students may receive the new item information before other students, who could be at a disadvantage.

2. Option 2: using Server Sent Events (SSE). SSE is a server-push technology where the client receives automatic changes occurring in the server. It is a one-way communication between client and server, where the server sends a message to the client each time it has an update. It can be understood as the client listening for changes in the server, rather than asking it for updates.

The following figure (Figure 2) simply demonstrates the difference between both options. SSE is an advance respecting AJAX. AJAX already is an advance in full-stack development as it enables to load dynamic elements in the page without having to reload the whole page. SSE improves this functionality in the way that the usual ask-answer way is not necessary anymore.

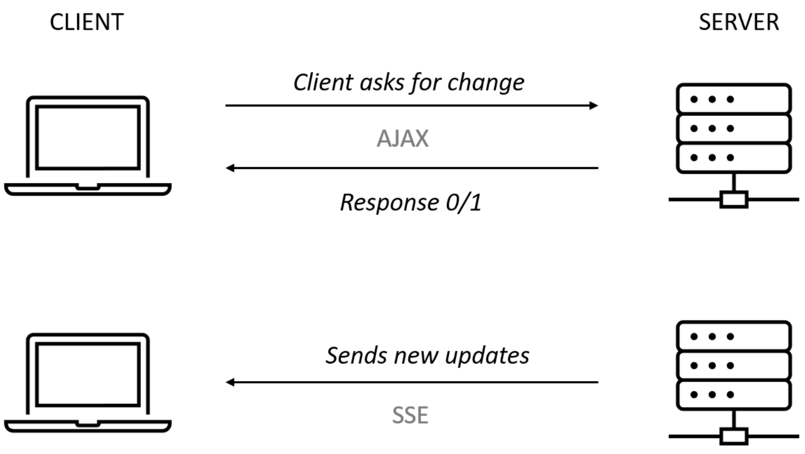


Figure 2: Difference between AJAX and SSE for this solution

To understand how the SSE technology is used in this context, here is a brief explanation on how the Codex app works when a student opens a new notebook to complete it.

First of all, the professor creates the notebook specifying which items it contains, and whether these items are enabled (the student sees them) or disabled. Once he saves it, the information is stored in the database. When the student opens a new notebook (or creates a new instance on an already opened notebook), it calls for a new notebook instance to be created. A notebook instance is a request for the server to both create empty records for each item in the notebook, so the student's answers will be stored in those records; and for printing the page with the notebook items. Once this instance is created, the student will answer the questions and, after sending them, they will be stored in the database. If the teacher were to add a new item to the notebook, students who already have their notebook instance created will not see any changes, as they do not check the notebook items' tables to load the notebook (once created, the notebook information is loaded from the useranwer's and NotebookInstances' tables). Only student who are creating a new instance will see those changes, as the request for creating the instance loads the items from the database table where the professor saves the updates. Figure 3 explains it in a visual way.

Now that this is clear, the implementation of the SSE method is easily explained. The steps behind the implementation are:

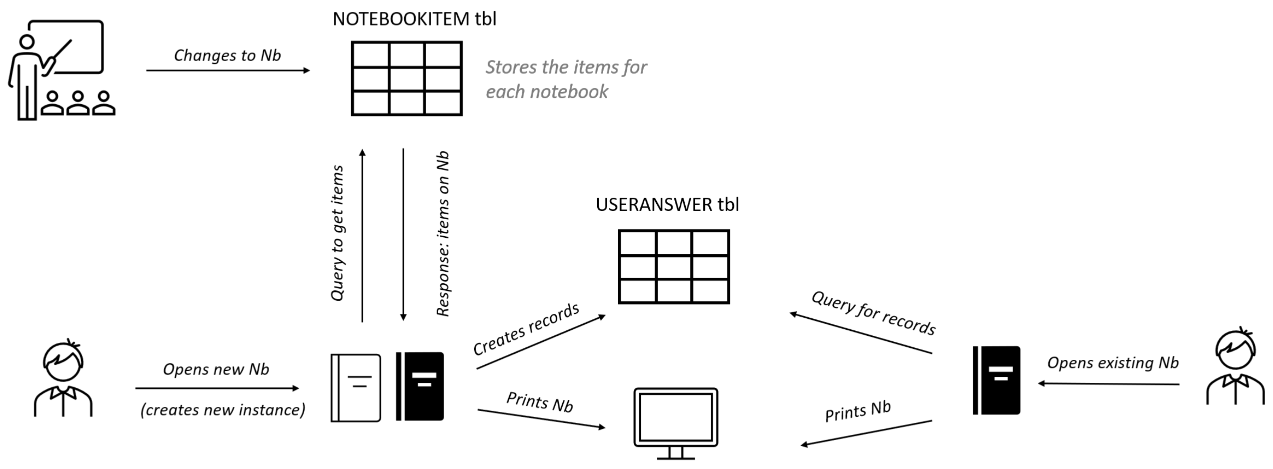


Figure 3: Notebook Instance explained

1. The professor, who has access to the subject's items, can enable and disable them just by checking and unchecking the boxes in the "Enabling" column of the Items page. When he checks/unchecks a box, it calls the EnableNoteBookItem servlet, with the arguments: Notebook ID, Item ID, Enable/Disable.

2. The EnableNoteBookItem servlet then performs the following actions:

   - If the item was already defined in the notebook, but disabled, it will enable it (and viceversa).

   - If the item was not defined in the notebook, it will insert it and it will call a function to create new records on the useranswer table for that item for every student who has the notebook opened.

3. The EnableNoteBookItem servlet will then create a new message with the information of the changes, and it will add it to a list of static messages that all the students have access to. When this list changes, the students receive the new information and the items are printed.

The structure of these messages and the list is explained in Figure 4.
We have two different classes:

1. **Msg class**: contains the information about the message type (add item, delete item) and the item ID.

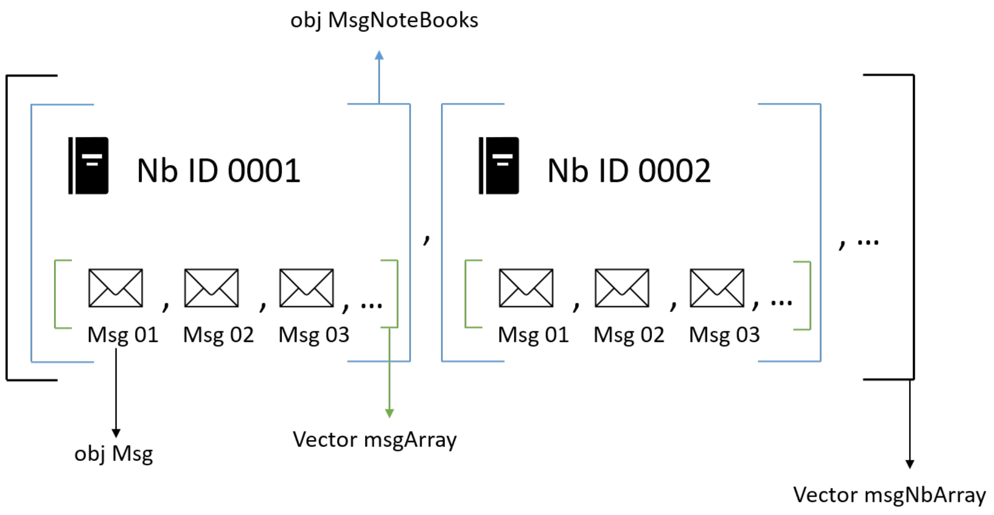


Figure 4: Structure of messages

2. **MsgNoteBooks** class: contains information about the notebook ID and all the messages regarding that notebook, stored in an array called msgArray.

The message created when the professor adds a new item will be of class Msg, and it will be introduced in the msgArray array of the corresponding MsgNoteBooks class (depending on the notebook it is contained in).

The msgNbArray array is defined as a static Vector and, thus, all the sevlets have access to it. When a student opens a notebook that the professor has defined as "live mode", the JavaScript code that loads the page will start a SSE connection with the checkMsg servlet (see Figure 5. This servlet simply has a variable for the student where it stores the value of the last message read by the student. The servlet checks every X seconds for changes in the messages list (msgNbArray) and, if it finds a new message, it sends it to the client as a JSON line containing both the type of message (add or delete) and the item's ID. The javascript file will then either make an AJAX call to another servlet (SSEInstance) to get the necessary data for printing the added item, or find the item to delete from the HTML code.

It is important to note the difference between this method and the first option mentioned at the beginning of the section. Even though using SSE the checkMsg servlet checks for changes every X seconds, it is not making requests to the server as the servlet is already on

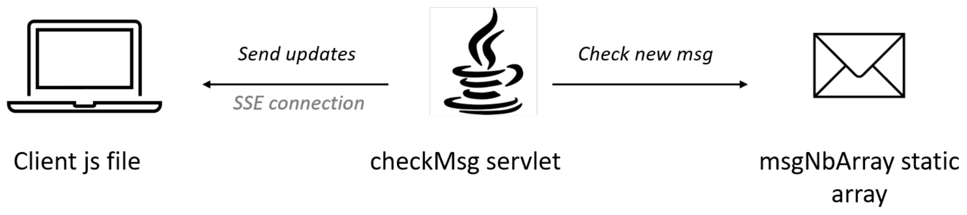


Figure 5: SSE with checkMsg

the server side and is just comparing two variables, which is not computationally expensive. In the case of using the AJAX call of the first option, we would be making requests from client to server.

The javascript file has been modified to add some miscellaneous elements, such as a banner telling the student that the notebook is in live mode and a moving Tecnun logo appearing when no items are enabled (as a way of showing the student that he should wait for new items to arrive). In the next section there are figures with real life pictures of it.

A new HTML file has been created to show the results of the live elements, to imitate the tools that are being used now. The pictures of the results page are in the next section.

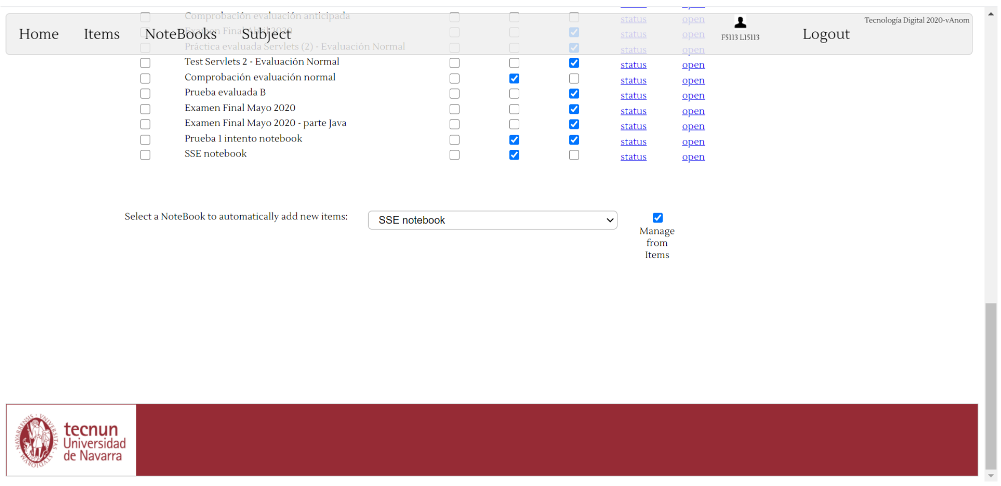
3.3 Actual implementation

This section contains a series of pictures that show the actual implementation of the "Live items" tool.

On the professor side, he will select the notebook that he wants to work on "live mode" through the Notebooks page (Figure 6).

To enable or disable elements, he will do it from the Items page (Figure 7).

On the students' side, new items added on live mode will appear with a different border

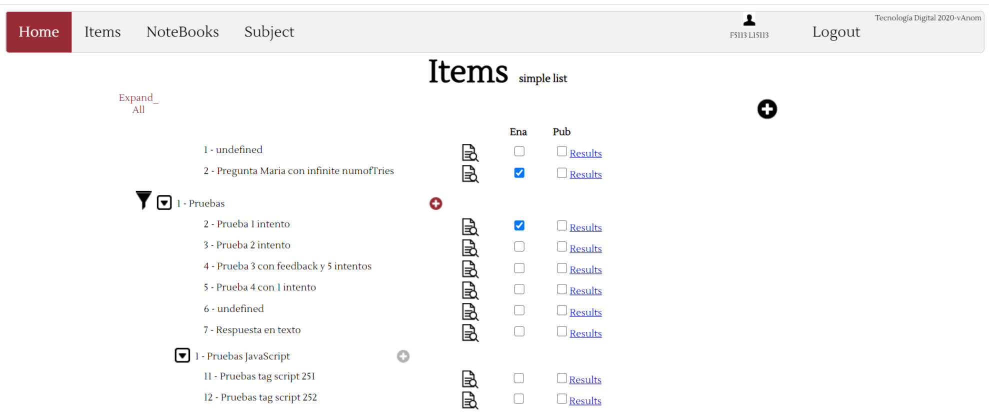


☐	Comprobación evaluación anticipada	☐	☐	☐	status	open
☐	Examen Final	☐	☐	✓	status	open
☐	Práctica evaluada Servlets (2) - Evaluación Normal	☐	☐	✓	status	open
☐	Test Servlets 2 - Evaluación Normal	☐	☐	✓	status	open
☐	Comprobación evaluación normal	☐	✓	☐	status	open
☐	Prueba evaluada B	☐	☐	✓	status	open
☐	Examen Final Mayo 2020	☐	☐	✓	status	open
☐	Examen Final Mayo 2020 - parte Java	☐	☐	✓	status	open
☐	Prueba 1 intento notebook	☐	✓	✓	status	open
☐	SSE notebook	☐	✓	☐	status	open

Select a NoteBook to automatically add new items: SSE notebook

✓ Manage from Items

Figure 6: Select notebook for "Live Mode"



Items simple list

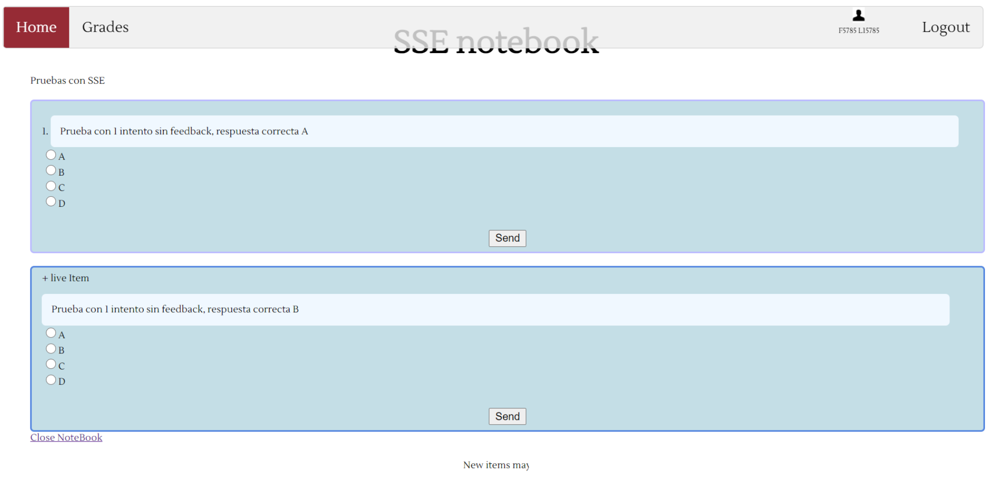
		Ena	Pub
1 - undefined		☐	☐ Results
2 - Pregunta Maria con infinite numofTries		✓	☐ Results
1 - Pruebas			
2 - Prueba 1 intento		✓	☐ Results
3 - Prueba 2 intento		☐	☐ Results
4 - Prueba 3 con feedback y 5 intentos		☐	☐ Results
5 - Prueba 4 con 1 intento		☐	☐ Results
6 - undefined		☐	☐ Results
7 - Respuesta en texto		☐	☐ Results
1 - Pruebas JavaScript			
11 - Pruebas tag script 251		☐	☐ Results
12 - Pruebas tag script 252		☐	☐ Results

Figure 7: Adding live items

to show that they have just been added (Figure 8).

If a student enters a notebook where there are no items yet, he will see a moving icon (Figure 9).

The main code written for this functionality can be found in Appendix A. It includes the servlets for checking for new messages, the structure of the messages and the requests made.



Pruebas con SSE

1. Prueba con 1 intento sin feedback, respuesta correcta A

A
B
C
D

+ live Item

Prueba con 1 intento sin feedback, respuesta correcta B

A
B
C
D

Figure 8: Notebook with live items

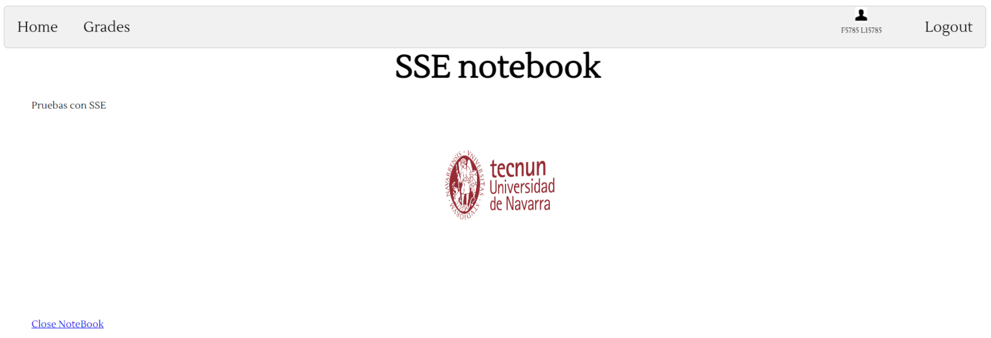


Figure 9: Live mode with no items

4 Data analytics

4.1 Objective

The Codex online tool generates a lot of information for each subject where it is used, and an objective of this project is to analyze the ways in which this data can be used to further improve Codex and the students' learning experience. The next sections propose a series of different analysis that could be made and information that could be extracted from this data and it develops one of those applications: predicting item grades based on past experiences.

This analysis has been made with the information available from the subject "Tecnología Digital" (2019-2020 course), but the algorithms and code are designed to be used for any subject. The technology used for developing this application is Python language and different Machine Learning libraries available through Python. It has been developed using Jupyter Notebooks, and the complete code and results can be found in Appendix B.

4.2 Overview of available data

The first step for undertaking the analysis of the data from Codex is to see the available data in the database. The Codex database is easily accessed through SQuirreL SQL Client, a graphical SQL client written in Java that allows us to browse and issue queries through the SQL syntax.

A summary of all the information available through this data set is presented in the following paragraph:

- For each **item**, we have the information of its content, item type, number of possible tries, correct solution, grading system and codification of the item.
- For each **notebook**, we have the information of its grading system, enabling data and publishing options, along with its proper codification.
- For each **notebook**, we have the information of the items that compose it.

- For each **student**, we have their login information (as in timestamp, remote IP and user agent)
- For each **notebook instance**, we have the information of the date it is created, the student it belongs to and the grade.
- For each **answer** (answer of 1 item from 1 student), we know the answer, grade, timestamp and number of tries (version), along with the information of which student and notebook instance the item belongs to.

Seeing all of this, the conclusion is that we are registering many data from all of the different answers to each item from every student. We also know the answer date for each item and the number of tries it takes for each student to send the final answer.

This data, which comes from using the Codex tool, can be completed or extended thanks to other tools and resources available in Tecnun. These other sources of information include the following:

- **Panopto** tool for video visualization: up until now, the professors at Tecnun have been using the Panopto software, where they upload videos and short quizzes for students to view either before or after class. Panopto creates an output file with logs of all the students and their viewing information with the following format:
  Timestamp, Start Position, Minutes Delivered, Username, User ID, Name, Email
- **Students' information**: some personal information about students, ensuring their privacy through appropriate encoding, is available, such as their origin (nationality and, if they are from Spain, their region) and their gender.

4.3 Bibliography and application proposal

After reviewing the different analysis and studies done for Machine Learning applications in the field of education, a brief summary is presented here.

Many articles write about the importance of learning analytics (LA) and educational data mining (EDM) as concepts that are evolving and becoming relevant due to the increasing amount of data that is being generated in the education field. Both are similar, although a more detailed definition will reveal that EDM revolves more around developing methods to analyze data, and LA is focused on creating applications for improving the educational practice on a daily basis. Either way, these two areas define different methods to use when studying data gathered by models and applications. Calvet, L. & Juan, Á. A. (2015) and Cristobal, C & Ventura,S (2020) describe many of the possible applications for EDM and LA. These include, but are not limited to: predicting students performance, grouping similar students based on their learning behaviour, help professors identify patterns on their students' activities, detection of outstanding or poor-performing students, monitoring students' skills over time, obtaining knowledge of the learning process through event logs, etc.

On a more practical note, on the study carried out by O. Sanchez in "Data science takes on public education: using machine learning to help analyze an education problem", the researcher used data of hundreds of students from the public school system in New York City to assess how different factors affected their achievement rates. Using data regarding their economic level, race, tests and schools' education ratings, he was able to find the factors and specific situations that were a drawback on students' success. Plus, he analyzed how different machine learning models scored on the problem. Although the approach on this last study is not the one intended in the Codex application, it does have certain similarities with it, as it uses a great amount of students' data to try and explain their performance.

Moreover, all the articles related to EDM and LA in education emphasize the need for visual tools when assessing the results of the analysis. Professors demand a tool that helps them improve their learning practices on a practical way, and dashboards and applications that clearly show the differences achieved after applying those EDM and LA techniques.

Based on this research, and the data available through Codex and other sources at Tecnun, here are the proposed analysis to be done:

- **Prediction of students' grades** and performance at each moment of the course, based on his/her behavior and that of previous years' students. This will allow the teacher to know the performance of his class, and allow the student see where he is heading.
- **Grouping students** according to the learning method they use and analyze what difficulties each of these groups encounters.
- **Analysis of forums** in the subjects and the influence of participation in learning.
- **Significance of the questions** asked by the teacher in the exams, to help the teacher eliminate questions that do not objectively evaluate the knowledge of the class as a whole.
- **Influence of various factors** (gender and origin) of students in their learning.

Out of all of this applications, this project is going to develop the first one: predict students' final and intermediate grades based on their performance throughout the year.

4.4 Basic concepts and explanations

Before explaining the development of the grade prediction model, it has been thought convenient to define and explain some basic concepts regarding machine learning and data analysis.

4.4.1 Machine learning techniques

"In the second half of the twentieth century, machine learning evolved as a subfield of Artificial Intelligence (AI) that involved self-learning algorithms that derived knowledge from data in order to make predictions. Instead of requiring humans to manually derive rules and build models from analyzing large amounts of data, machine learning offers a more efficient alternative for capturing the knowledge in data to gradually improve the performance of predictive models and make data-driven decisions." Raschka, S. and Mirjalili, V., 2017.

There are three types of machine learning techniques (see Figure 10):

- **Supervised learning**: based on training data, the algorithm creates a model to predict future unseen outcomes. This set of possible outcomes is already known (labels).
- **Unsupervised learning:** it deals with unlabeled data where the possible outcomes are unknown.
- **Reinforcement learning**: it develops a model that improves its performance based on the interactions with the environment: after defining a reward function, the model learns a series of actions that maximize the reward.

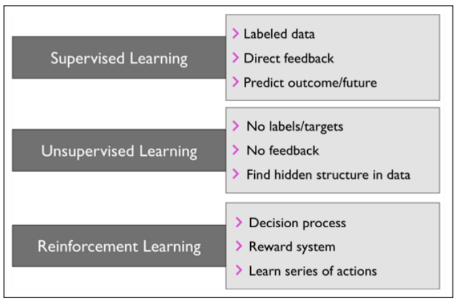


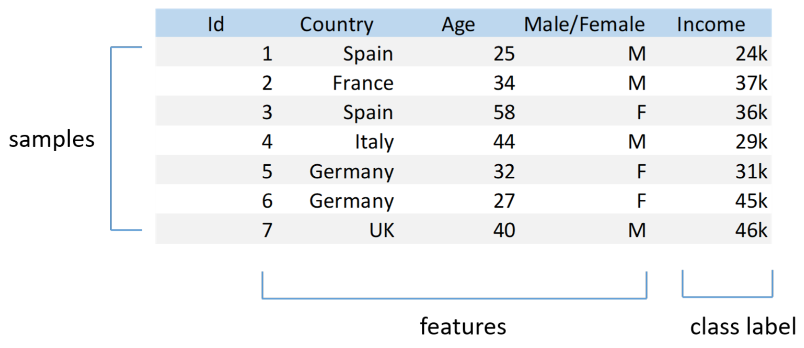
Figure 10: 3 types of machine learning techniques (from Raschka, S. and Mirjalili, V., 2017)

4.4.2 Basic terminology

In terms of terminology, a small example of a dataset will be used to define the different key concepts (Figure 11). In this example we find the basic terminology of any DataSet that is used in a machine learning problem. Each of the different rows found in the dataset are called samples (instances, observations) and each of the columns represent a feature of the database (origin, age, etc.) and the last column is the class label, the objective that we seek to know of each sample.

4.4.3 Steps for a Machine Learning model

A machine learning problem is built with four steps: data preprocessing, model learning, model evaluation and subsequent prediction with the model.



Id	Country	Age	Male/Female	Income
1	Spain	25	M	24k
2	France	34	M	37k
3	Spain	58	F	36k
4	Italy	44	M	29k
5	Germany	32	F	31k
6	Germany	27	F	45k
7	UK	40	M	46k

samples
features
class label

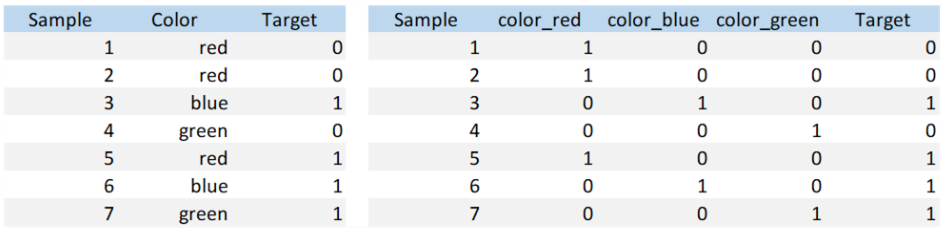
Figure 11: Simple dataset used as an example

Data pre-processing: Quite a big part of the success of a model is due to the quality and useful information of the training data. Data can come from different sources and not always with the desired format for our model. That is why data preprocessing becomes a key activity when developing the model. The first step after receiving the raw data is to delete the data that is incorrect and make sure that all the records have the complete features. The next step is to edit the format of the data to be able to feed the model.

The data can be categorical or not. Categorical data is the one that can be included within a range of known data, and this is the one that needs to be edited. The categorical data is divided into two categories: ordinal and nominal. Ordinal data is the one that has a specific order, for example, clothing sizes: XS, S, M, L. Nominal data is that which cannot be ordered under any criteria, for example, colors: blue, red, green. Most machine learning models convert the data from strings to integers, and it is good practice to treat the data with numeric values.

The simplest way to edit this data and convert it to numeric values is by using the label encoder method which simply assigns each of the various categorical values an integer value. If the data is nominal, this can lead to problems since the model can understand that the different numbers represent an order within the data. To solve this, we use the method called one hot encoding. The one hot encoding method divides the nominal properties into as many values as many different values available in that feature, and assigns each of these new values a binary number 0 or 1. An example of this method is shown in the following

Figure 12. One hot encoding has a problem, and that is that it introduces multicollinearity (a variable used for prediction can be predicted from the rest of the predictor variables). Some methods use matrix inversion and this can be a problem. The solution is simple and consists of eliminating one of the columns created with this method, thus eliminating the possibility of multicollinearity.



Sample	Color	Target
1	red	0
2	red	0
3	blue	1
4	green	0
5	red	1
6	blue	1
7	green	1

Sample	color_red	color_blue	color_green	Target
1	1	0	0	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	0
5	1	0	0	1
6	0	1	0	1
7	0	0	1	1

Figure 12: Performing one-hot encoding

Model construction: In order to build a model and test its accuracy, the data that we use is divided into two datasets: Training dataSet and test DataSet. This partition is done randomly by specifying a partition percentage. If the training DataSet is much larger than the DataSet test, we will not have a correct stimation of the model generalized error. If, on the contrary, the training Set is much smaller than the test dataSet, we will be losing valuable information that would help us to train the model and obtain a more realistic result. Thus, obtaining a good partition percentage is important when creating the model. The literature specifies that for datasets of not very large sizes, a 70: 30 split is a good guideline value. On the other hand, for very large datasets, we can move to ratios of 90: 10 or 99: 1.

Another aspect to take into account when building the model is that most of the methods used for optimizing the model work better and more efficiently with data that is within the same scale. There are two different approaches to this: normalization and standardization. In normalization we simply edit the values so that they are within a range of [0, 1]. In standardization we edit the values of each property so that they have a mean value of 0 and a standard deviation of 1, so the property column will have a normal distribution.

Once we have divided the data in the two datasets, we load the machine learning model that we have chosen to solve this problem and train it. Most of the algorithms used in machine learning are available through Python libraries. The next section explains the bases of the model that has been used in the development of the application. After training the model, we move on to the next step, where we evaluate how well the model fits the real problem.

Evaluating the model: Once the model is trained, the test DataSet is used to check the accuracy of the prediction on unseen data, and to see what the generalization error is. If the result is satisfactory, the model can be used to predict results in reality. It is important that the data in the DataSet test be given in the same format as the training DataSet. That is, if the data from the training dataset have been standardized, those from the summer test dataset will also be standardized.

Data prediction: The last step is to use the model to predict the problem outcome in reality. The model will be exported to the desired format and will be implemented in our application

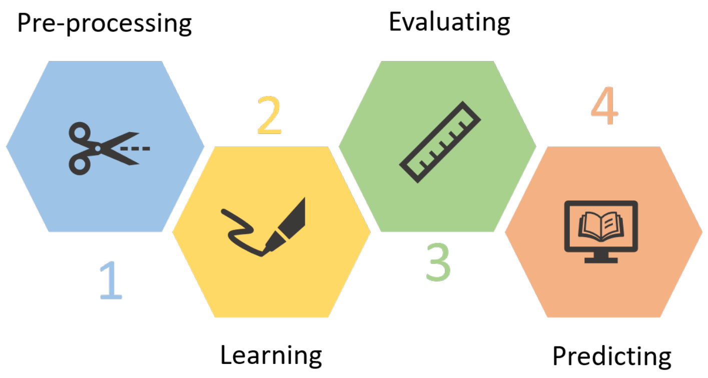


Figure 13: Building a machine learning model

4.4.4 Perceptron rule and Adaline algorithm

Finally, to be able to explain which algorithm is used for predicting the grades for the students, here is a brief summary of the basis of machine learning algorithms, which started with the Perceptron rule.

Perceptron rule

In 1943, McCulloch and Pitts modeled neural cells as logic gates that, taking a series of inputs and integrating their values, generated a response signal based on whether the integrated input signal exceeded a certain threshold value or not. It was not until 1957 that American psychologist Frank Rosenblatt introduced the Perceptron learning model, an algorithm capable of automatically learning the weights that each input had to be multiplied by to obtain the correct classification results.

This Perceptron model is mathematically described as (Figure 14):

- Inputs: $x = \{x_1, x_2, x_3, ...\}$
- Weights: $w = \{w_0, w_1, w_2, w_3, ...\}$
- Bias unit: w_0
- Input function: $z = w^T x = w_0 x_0 + w_1 x_1 + ...$
- Decision function: $\theta(z) = \{1, z >= 0; -1, otherwise\}$

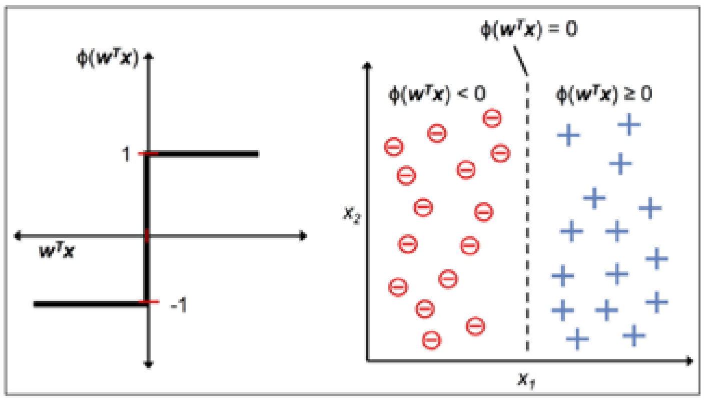


Figure 14: Perceptron rule (from Raschka, S. and Mirjalili, V., 2017)

The Perceptron weight learning algorithm is defined in the following steps:

1. Initialize the weigths w

2. For each sample in the training set x^i:
    - Compute the output value ŷ
    - Update the weights

The updating of the weights is carried out according to the rule:

$$w_j := w_j + \Delta w_j$$

where:

$$\Delta w_j = \eta(y^i - \hat{y}^i)x_j^i$$

η is the learning rate. Perceptron convergence will only be achieved if the two classes can be separated linearly and if the learning rate is small enough. See Figure 15 for clarification.

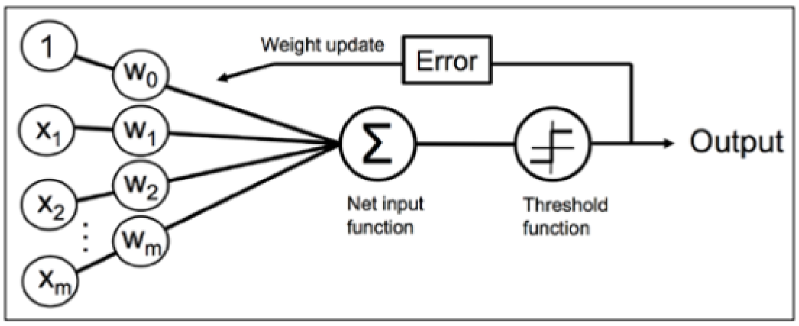


Figure 15: Perceptron algorithm (from Raschka, S. and Mirjalili, V., 2017)

Adaline (Adaptive Linear Neuron)

In 1960 Bernard Widrow and Tedd Hoff presented the Adaline model, which represented an improvement as to that of Rosenblatt's Perceptron algorithm. In addition, the idea of a cost function is introduced in classification algorithms, which serves as the basis for the ideation of new, more complex algorithms. The main difference is that the weights by which the inputs are multiplied are not updated based on the prediction, that is, the output of the threshold function, but rather are updated based on a linear function (called the linear activation function) . In Adaline, the activation function is the same as the input function $z = w^T x$.

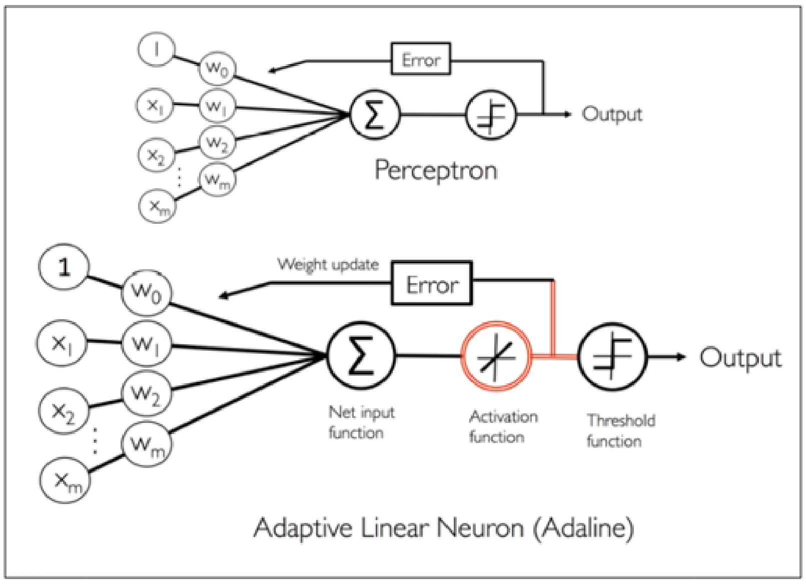


Figure 16: Adaline algorithm (from Raschka, S. and Mirjalili, V., 2017)

For classification, a step function (-1,1) is still used as in the Perceptron. See Figure 16.

Cost function and Gradient Descent method

In the Adaline algorithm, a cost function is also defined, being the objective of the algorithm to minimize said function. In the Adaline case, the cost function is the sum of the squares of the errors (the errors being the difference between the real class label and the one predicted by the algorithm (continuous value, before the threshold function)).

The cost function is mathematically described as:

$$J(w) = \frac{1}{2} \sum_i (y^i - \theta(z^i))^2$$

It has two interesting properties that make it a very effective algorithm:

1. The activation function is linear, so it is differentiable (it admits derivatives in any direction).
2. The cost function is a convex function (any local minimum we find is a global minimum).

Thanks to this, the gradient descent method can be used to effectively find the minimum error and, therefore, the best solution for the weights of the classification algorithm. The gradient descent algorithm is used to optimize the cost function as it's shown in the following point.

The gradient descent method can be intuitively understood as finding the fastest way to climb down a hill. Starting at a certain point P_0, each step we take to get to the final point P_f has to be the step that makes us go downhill faster. Mathematically, this is described as:

Take a step:

$$w := w + \Delta w$$

where the change Δw is the learning rate by the negative gradient :

$$\Delta w = -\eta \nabla J(w) = \eta \sum_i (y^i - \theta(z^i))x_j^i$$

Being the gradient of the cost function the negative partial derivatives of the function with respect each of the weights w_j. To calculate the gradient of the cost function, we calculate the partial derivative of the cost function with respect to each weight w_j as:

$$\frac{\partial J}{\partial w_j} = -\sum_i (y^i - \hat{y}^i)x_j^i$$

The learning rate parameter η needs to be properly chosen:

- If η is too small: gradient descent will take too long to converge.
- If η is too large: gradient descent will overshoot the minimum and, thus, it will fail to converge.

4.4.5 Predicting a continuous output

The Adaline and Perceptron algorithms simply predict the output of a classification problem. In our case, the output desired is a continuous value within the range 0-10 (final grade of the student). The number of possible values between those two grades is infinite, so a slight change in the algorithm is needed.

We will be using a regression model for predicting the grades of the students. Regression analysis is a subcategory of supervised learning, which predicts target variables on a continuous scale (as is the grades scale). The most simple example is the one for predicting an outcome y based on a single input variable x_1. The w_0 coefficient represents the y-axis intercept while the w_1 coefficient is the weight for the input variable.

$$y = w_0 + w_1 x_1$$

The solution for this example, the w vector (containing both w_0 and w_1 coefficients) will give us a simple 2D line as the solution to the problem.

Generalizing this simple linear regression model to multiple explanatory variables gives us the multiple linear regression model, which will take a large number of input variables (the answered items' grades) and give us the outcome as a continuous value (the grade that we want to predict). This translates to:

$$y = w_0 x_0 + w_1 x1 + ... + w_m x_m = \sum_{i=0}^{m} w_i x_i = w^T x$$

Although we cannot visualize the result of this problem, the logic is the same as the one of the simple regression model: given a set of input variables (x), we want to find the optimal coefficients (w) which will give us the best solution to the grade prediction problem (y).

The main goal of the regression model is to find the estimated coefficients for a set of input variables that produce the most accurate outcome of a continuous variable. This is done through an optimization problem as the one explained in the Adaline and Gradient descent method. As mentioned, this algorithm predicts labels for classification problems. But taking the output of the activation function (before the treshold function) as the final prediction gives us the continuous output that we are looking for.

In our case, the input variables will be the answers of the students to each item, and the outcome variable will be the final grade (or individual item grades that have yet not been answered, as we will see in this section). The model bases its predictions on the already answered items of each student and the data from past students who have already finished the course.

It is easily understandable that the model will be more accurate at predicting the final grade of the students the closer we get to the end of the course, as it will have more real data than predicted one. Thus, one of the main tasks of this application is to establish the percentage of the course needed for the predicted grade to be reliable enough, along with estimating other needed parameters for the regression model.

4.5 Development of n "Grade prediction" model

This section explains the development of the Grade prediction model, divided in the 4 steps explained before: data pre-processing, model building, model evaluation, final prediction. In the following subsections these points will be thoroughly discussed and developed. Note that the coding has been done in the Jupyter Notebooks environment, and the code and intermediate results can be accessed at the end of this document (Appendix B).

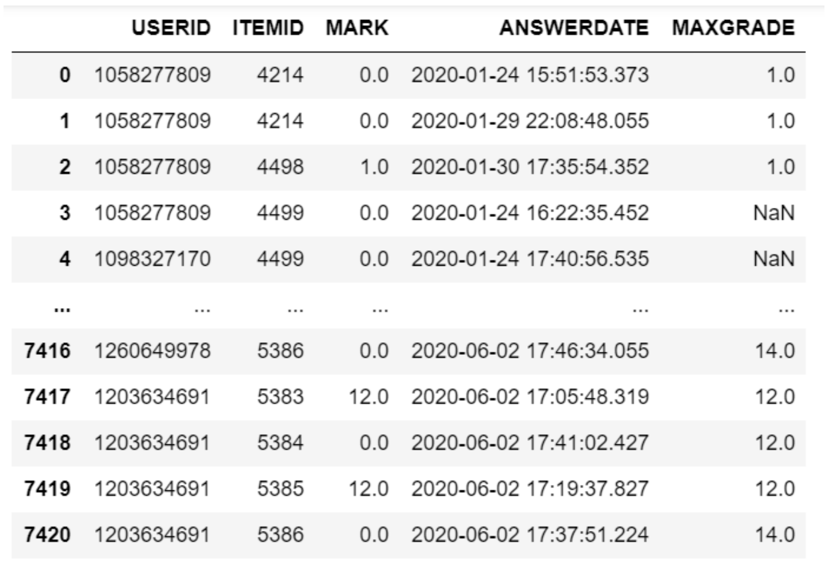
4.5.1 Data preprocessing

To begin with, we are going to use the data from the 2019-2020 course (Digital Technology) available through Codex, as it is the first course where the online tool was used continuously throughout the year. To use the data in the Jupyter Notebook environment, we first need to export it from the SQuirreL database. We are going to be using the USERID and GRADE from each student answer. To export the data from the database into a csv file, we first need to issue a query that will give us the exact data that we want to store. In our case, the SQL statement needed is the following:

SELECT USERID, USERANSWER.ITEMID, MARK, ANSWERDATE, MAXGRADE FROM USERANSWER, LOGIN, ITEM WHERE LOGIN.LOGINID = USERANSWER.LOGINID AND ANSWERDATE <=('2020-09-01 00:00:00.0') AND ITEM.ITEMID = USERANSWER.ITEMID

Note that we are only getting answers up to september 2020, as the database may contain more information above that date which corresponds to a new course. Once we have the desired table, we save it to a csv file through the SQuirreL options: Session > Scripts > Store result of SQL in file and select "Export CVS file" and Charset: UTF-8.

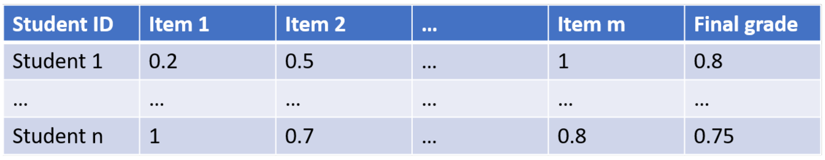
Now we can import it to our python program. After importing the pandas library, we can now import the saved table into a dataset using the pd.readcsv method. This stores the data in a new dataset with the following format (Figure 17):



	USERID	ITEMID	MARK	ANSWERDATE	MAXGRADE
0	1058277809	4214	0.0	2020-01-24 15:51:53.373	1.0
1	1058277809	4214	0.0	2020-01-29 22:08:48.055	1.0
2	1058277809	4498	1.0	2020-01-30 17:35:54.352	1.0
3	1058277809	4499	0.0	2020-01-24 16:22:35.452	NaN
4	1098327170	4499	0.0	2020-01-24 17:40:56.535	NaN
...	...	...	...	...	...
7416	1260649978	5386	0.0	2020-06-02 17:46:34.055	14.0
7417	1203634691	5383	12.0	2020-06-02 17:05:48.319	12.0
7418	1203634691	5384	0.0	2020-06-02 17:41:02.427	12.0
7419	1203634691	5385	12.0	2020-06-02 17:19:37.827	12.0
7420	1203634691	5386	0.0	2020-06-02 17:37:51.224	14.0

Figure 17: Original format of dataset

As explained at the beginning of section 4.5, the input variables to the model are the item grades and the target variables are the final grades. Recalling the structure of a machine learning model of section 4.4, our desired structure is the following one (Figure 18):

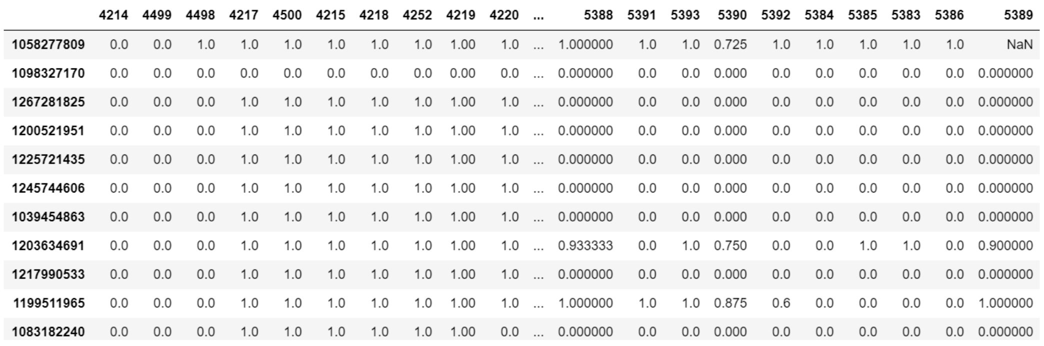


Student ID	Item 1	Item 2	...	Item m	Final grade
Student 1	0.2	0.5	...	1	0.8
...	...	...	...	...	...
Student n	1	0.7	...	0.8	0.75

Figure 18: Desired format of dataset

To achieve this final format, we first create a new blank dataframe with the appropiate design of rows and columns. Once we have that, we need to normalize the grades into a 0-1 range, as they have different formats for each item. This is not strictly necessary but it helps optimize the algorithm's performance. We do so by dividing each grade by the maximum grade for that item. This information is obtained from the original dataset in the form of MAXGRADE. If the MAXGRADE record for an item is missing, we find the maximum grade achieved by any student (through a simple loop) and use it as the maximum grade for the item.

Once we have filled the new dataframe with all the normalized values, the dataframe will look like this (Figure 19):



	4214	4499	4498	4217	4500	4215	4218	4252	4219	4220	...	5388	5391	5393	5390	5392	5384	5385	5383	5386	5389
1058277809	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	1.000000	1.0	1.0	0.725	1.0	1.0	1.0	1.0	1.0	NaN
1098327170	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1267281825	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1200521951	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1225721435	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1245744606	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1039454863	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1203634691	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.933333	0.0	1.0	0.750	0.0	0.0	1.0	1.0	0.0	0.900000
1217990533	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000
1199511965	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	1.0	...	1.000000	1.0	1.0	0.875	0.6	0.0	0.0	0.0	0.0	1.000000
1083182240	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.00	0.0	...	0.000000	0.0	0.0	0.000	0.0	0.0	0.0	0.0	0.0	0.000000

Figure 19: Final format of dataset

where the column names are the item ID's and each row represents a student.

With this new format it is really easy to train a model, but there are a few more changes to be done which will improve the algorithm's performance. First, we are going to remove the features (items) that do not give us any information. These are the features where all the records (grades) are 0.0, which means that either they were theoretical items or they were asked not to be answered by the professor. With a simple loop we can remove the columns through the pandas drop method. In this case, 12 items were eliminated. On the other hand, we have missing values, which are represented as NaN values. In this case, we are going to assume that all those who didn't answer the question and, hence, have a NaN record for that item, have a 0.0 mark. Thus, we are now going to convert all the NaN values to 0.

Now we have the data in the correct shape, and we can easily visualize a student's progress throughout the year and his grades (Figure 20):

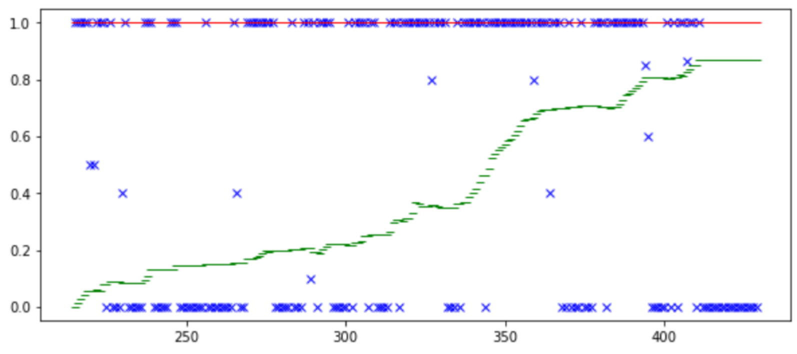


Figure 20: Random student's progress

where the blue crosses represent each item's grade, the red line represents the final grade and the red curve represents the approximate accumulated grade. Note that the green line, which represents the "accumulated" grade at that point of the year, has points where it decreases because it has been calculated based on a linear regression model where some coefficients are negative (see Figure 21).

Finally, we save this new dataframe into a csv file to access it later on.

The objective of the model that we are creating is to predict the final grades. These are the target values, and we need to ensure that they come also in the correct shape. For this, we import the final grades from a csv file as a separate series in pandas. Then we check that

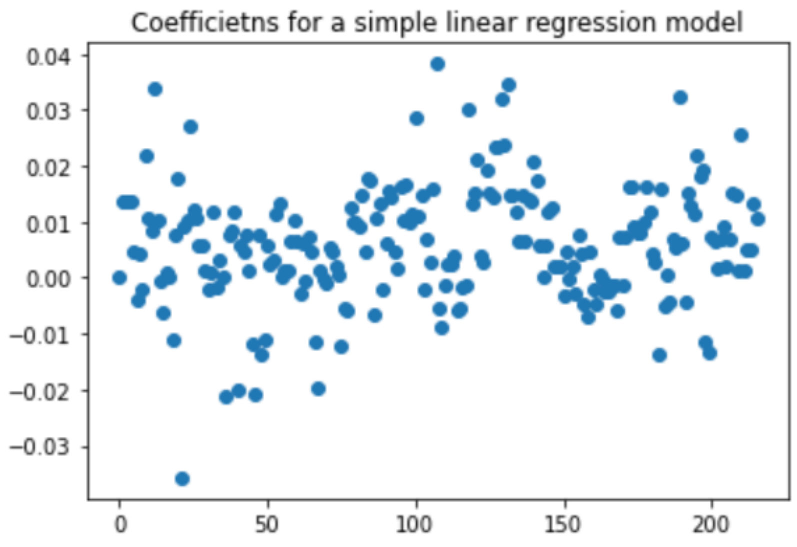


Figure 21: Coefficients for grade visualization

we have no users without final grades and we order the users' final grades in the same order as the users in the students dataframe (the dataframe that contains all the information of the item grades). If there is any user that is missing from the records, we drop it.

We save this new list into a csv file too.

4.5.2 Model construction

The next step is to train the linear regression model with scikit-learn's LinearRegression model. As mentioned previously in section 4.4.5, the multiple linear regression model takes m input variables and tries to find the w coefficients to produce the y outcome that best fits the actual solution. But what do we mean with the best-fitting solution and how do we find it?

Linear Regression and Gradient Descent

A widely used metric for measuring a good-fitting behaviour of the solution is the Ordinary Least Squares (OLS) method, which is the sum of the squared residual errors of each sample point from the training set. We define this function as the cost function:

$$J(w) = \frac{1}{2}\sum_{i=0}^{n}(y^i - \hat{y}^i)^2$$

The term $1/2$ is added for convenience, as it will ease the calculation of the gradient in the next steps. This cost function becomes the objective function that is to be optimized during the learning process of the model.

The optimization algorithm that the `LinearRegression` model from scikit-learn uses is the gradient descent method. In each iteration, gradient descent takes a step in the opposite direction of the gradient of the cost function, until it finds the global minimum (or meets a certain criteria for stopping the iteration process). The step size is determined by the value of the learning rate, as well as the slope of the gradient. An illustrated explanation is given in Figure 22.

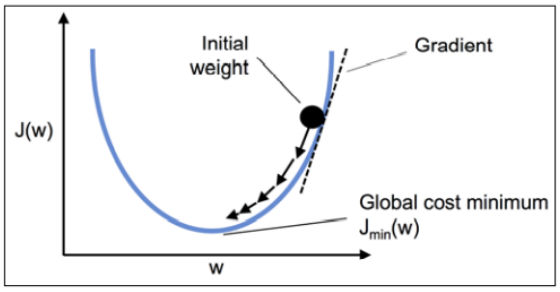


Figure 22: Gradient descent method

Implementing the algorithm to predict final grades

Implementing this algorithm through the scikit-learn library of Python is easy and quick, as it can be seen in the Jupyter Notebooks in the Apppendix B. After importing the edited dataframe into the program, we first need to split our data into a training and test dataset. Using scikit-learn's `train_test_split` method, we define the percentage of the samples that are going to make it to the test dataset. The `random_state` variable allows us to plant a seed to replicate the same split in future iterations of the model.

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
random_state=1)
```

We are going to try and train the model to predict the final grade based on 150 items completed, out of all the 216 items that compose the course. Once both training and test datasets are created, we load the `LinearRegression` model from scikit-learn and fit the train set to it. This step might take a few seconds to compute, as the algorithm has to iterate to find the minimum error through the GD method. By calling the `coef_` and `intercept_` methods from the `LinearRegression` object, we get the slope and intercept values of the fitted model. As we are working on a high-dimensional feature space, it is not really helpful for visualizing the solution, so we will skip it.

We can now use the trained model to predict the final grades for the test set, and see how well it fits the students' marks through the `predict` method of the `LinearRegression` object. Here the blue crosses represent the actual grade that the students achieved, whereas the red cross is the predicted value.

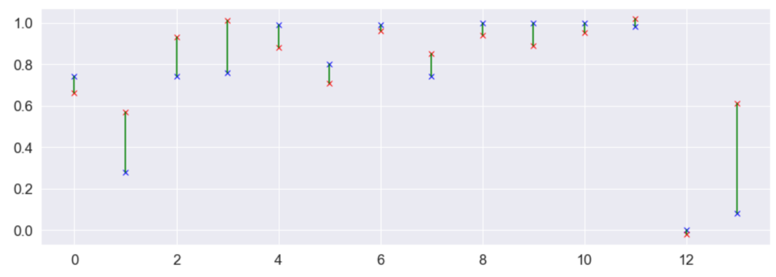


Figure 23: Prediction of final grades for the test set

Figure 23 shows a general good accuracy for most of the students, but it fails to predict correctly two of those students (the second and last one) by a large error.

Following this problem, we checked to see if those two students had anything in common between them and whether they differed much from the rest of the students. Plotting a random student's progress throughout the year (in this case, the first student from the test set) and both the "outliers'" progress showed the following:

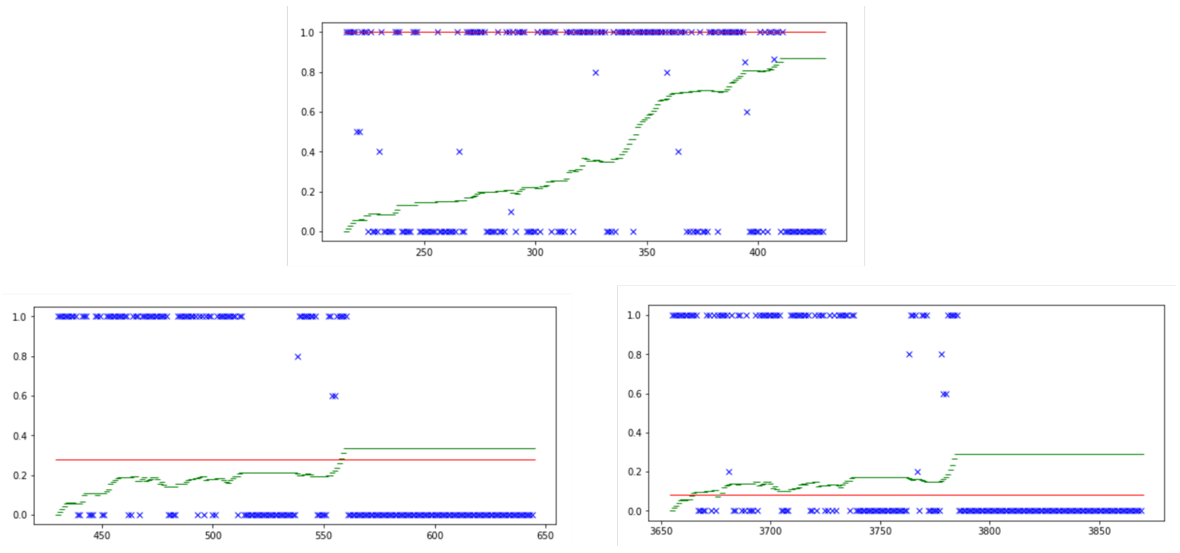


Figure 24: Performance of 3 students

It appears that both the students with a greater error at the prediction are students who clearly dropped the course at the middle of the semester. It is understandable that this situation causes the error from Figure 23, with both students having a much higher predicted grade than the achieved one. This is because the model is only trained with the first 150 items, where the students had a better performance than the one in the second half of the course. As the model does not know that, it predicts a much higher value.

This matter needs to be assessed properly as it is important for two different reasons:

1. Finding a way to see which students are going to drop the course at the middle of the semester to help prevent it
2. Detecting those outliers and leave them out of the training dataset so they do not affect the model's accuracy

At the moment, we are going to tackle this problem by automatically selecting those students who are outliers and leaving them out of our dataframe. The criteria chosen to decide whether a student is an outlier or not is by seeing his performance in the second half of the course. After seeing the percentage of 0.0 gotten by the outliers, a 40% of 0.0 grades denotates that he droped the course. But, for students who got a good grade along the whole course but did not have to answer as many items in the second half not to be detected as outliers, a final grade below 0.3 is stated.

After dropping the outliers out of the database, the result of the linear regression model is the following (see Figure 25):

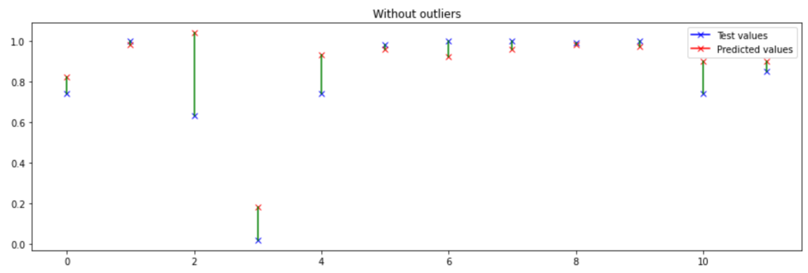


Figure 25: Prediction of final grades without outliers

In this new prediction, we can observe that, although there is one case which is predicted as a higher grade than real, overall the prediction is quite good. It is important to note that the predicted items in this case are different from the one with outliers because the training and test datasets are different. When we dropped the students that were considered outliers, we had to re-split the dataset into training and test sets, so the groups are now different.

4.5.3 Evaluating the performance of the Linear Regression model

One of the most important part of a machine learning model is using good evaluation metrics for performance. As mentioned before, this problem has multiple explanatory variables, so we cannot visualize the linear regression hyperplane. Instead, we use some other metrics for understanding the model's perfomance. These are explained in the following points.

- **Explained variance score**: it measures the proportion to which a mathematical model accounts for the variation (dispersion) of a given dataset:

$$ExplainedVariance(y, \hat{y}) = 1 - \frac{var(y - \hat{y})}{var(y)}$$

- **Max error:** it is the maximum residual error (worst case scenario):

$$MaxError(y, \hat{y}) = max(|y_i - \hat{y}_i|)$$

- **Mean absolute error**: it is a risk metric corresponding to the expected value of the absolute error loss or L1-norm loss:

$$MAE(y, \hat{y}) = \frac{1}{Nsamples} \sum_{i=0}^{N_s - 1} |y_i - \hat{y}_i|$$

- **Mean squared error**: it is a risk metric corresponding to the expected value of the squared error:

$$MSE(y, \hat{y}) = \frac{1}{Nsamples} \sum_{i=0}^{N_s - 1} (y_i - \hat{y}_i)^2$$

- **Median absolute error**: it is robust to outliers. The loss is calculated by taking the medan of all absolute differences between target and prediction:

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y}_1|, ..., |y_n - \hat{y}_n|)$$

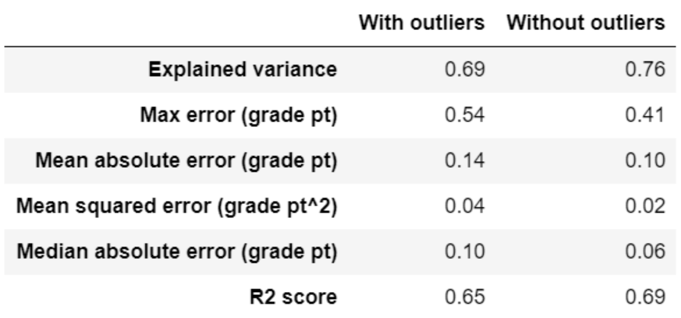
- R^2 **score:** it is the coefficient of determination. It represents the proportion of variance that can be explained by the independent variables of the model. It provides an indication of goodness of fit, so it allows to know if it will perform good or not on unseen samples. R^2 variance is dataset dependent, so it is not meaningful for comparing models with different datasets.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y}_i)^2}$$

where:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

After calculating the performance according to these metrics, the results are the following (see Figure 26):



	With outliers	Without outliers
Explained variance	0.69	0.76
Max error (grade pt)	0.54	0.41
Mean absolute error (grade pt)	0.14	0.10
Mean squared error (grade pt^2)	0.04	0.02
Median absolute error (grade pt)	0.10	0.06
R2 score	0.65	0.69

Figure 26: Performance metrics of the model

It is easily observable that the model without outliers obtains better metrics in all cases. The explained variance and R^2 score are higher in both cases, which means that the model will perfom better on unseen data, and the errors that we get in the model without outliers are smaller.

We can see that there is still room for improvement in the model, and there are some methods that are intended for better performance in prediction. These are called regularized methods, and they are explained in the next point.

Regularized methods for regression

Regularization is one approach used for tackling the problem of overfitting. This occurs when the model selects the coefficients by overfitting the curve of the regression, and it then fails to predict correctly unseen records. Regularization methods add additional information to the model, thus shrinking the parameters values of the model to induce a penalty against complexity.

There are three main methods: Ridge Regression, Least Absolute Shrinkage and Selection Operator (LASSO) and Elastic Net.

- **Ridge regression**: it is a model where we simply add the squared sum of all the weights to our least-squares cost function. By increasing the λ parameter, we increase the regularization strength (penalizing the weights) and the weights tend to decrease.

$$J(w)ridge = \sum_{i=1}^{n}(y^i - \hat{y}^i) + \lambda||w||_2^2$$

where:

$$\lambda||w||_2^2 = \lambda \sum_{i=1}^{m} w_j^2$$

- **Least Absolute Shrinkage and Selection Operator (LASSO)**: with this method, depending on the regularization strength, certain weights can become zero, which also makes LASSO regression useful as a supervised feature selection technique. LASSO selects at most n variables, so if $n > m$ we can loose important information.

$$J(w)lasso = \sum_{i=1}^{n}(y^i - \hat{y}^i) + \lambda||w||_1$$

where:

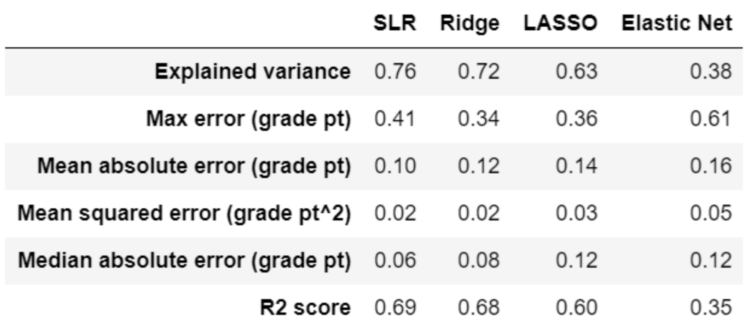
$$\lambda||w||_1 = \lambda \sum_{i=1}^{m} |w_j|$$

- **Elastic Net**: it is a mix between Ridge and LASSO regression: it has both penalties.

$$J(w)ElasticNet = \sum_{i=1}^{n}(y^i - \hat{y}^i) + \lambda_1 \sum_{i=1}^{m} w_j^2 + \lambda_2 \sum_{i=1}^{m} |w_j|$$

After several iterations with different lambda values for each regularization method, the best obtained results are shown here. Note that the comparisons between different regularization methods are done for the model trained without outliers, as in the previous point it was already stated that the results were better.

Regularized methods are not particularly useful for our case of study, as all of them give the same or worse results as the Simple Linear Regression (SLR) model (which does not have



	SLR	Ridge	LASSO	Elastic Net
Explained variance	0.76	0.72	0.63	0.38
Max error (grade pt)	0.41	0.34	0.36	0.61
Mean absolute error (grade pt)	0.10	0.12	0.14	0.16
Mean squared error (grade pt^2)	0.02	0.02	0.03	0.05
Median absolute error (grade pt)	0.06	0.08	0.12	0.12
R2 score	0.69	0.68	0.60	0.35

Figure 27: Performance metrics for different regularization methods

any method applied). We can assume that the linear regression model that we create is not overfitting and, thus, we do not need to apply any of these models, because it will only give us worse results.

To sum up, the final model that we are choosing is the one without outliers and without using any regularization methods.

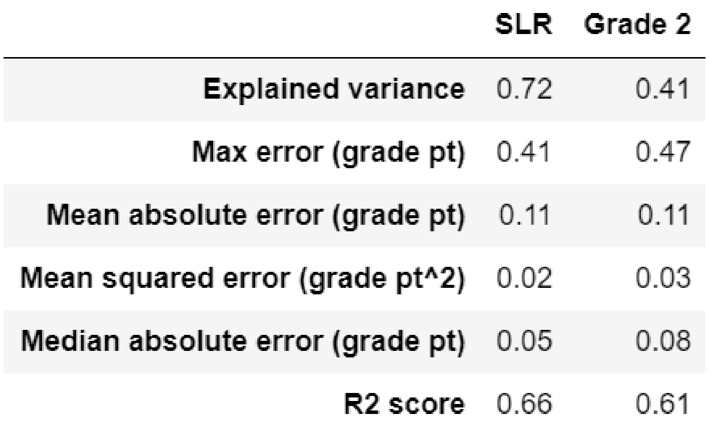
But there is still another possibility for building the model that has not been tackled, and that is building a polynomial regression model (that is, with a higher grade than 1).

Polynomial Regression

A trial with a 2 degree regression model was done but the results showed that it was not beneficial for the prediction. The detailed model creation and training algorithm can be found in Appendix B. Here is presented the performance metrics of SLR and polynomial regression.

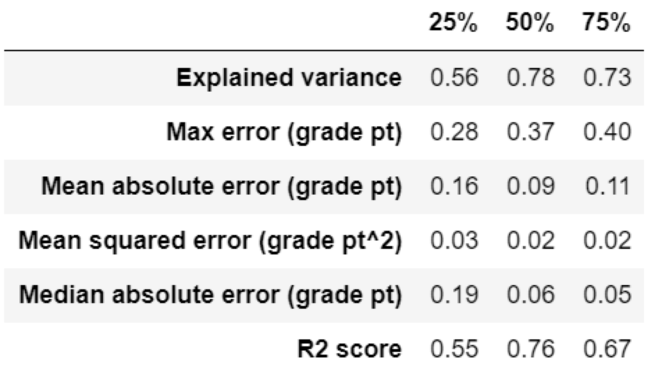
Tuning the parameters

This model has been constructed to predict the final grade based on the first 150 items of the course, which represents an approximate 70% of the course. Tuning these parameters is important for getting the best possible model. After training various models with different percentages of completed items, these are the results obtained:



	SLR	Grade 2
Explained variance	0.72	0.41
Max error (grade pt)	0.41	0.47
Mean absolute error (grade pt)	0.11	0.11
Mean squared error (grade pt^2)	0.02	0.03
Median absolute error (grade pt)	0.05	0.08
R2 score	0.66	0.61

Figure 28: Performance metrics for polynomial model



	25%	50%	75%
Explained variance	0.56	0.78	0.73
Max error (grade pt)	0.28	0.37	0.40
Mean absolute error (grade pt)	0.16	0.09	0.11
Mean squared error (grade pt^2)	0.03	0.02	0.02
Median absolute error (grade pt)	0.19	0.06	0.05
R2 score	0.55	0.76	0.67

Figure 29: Performance metrics for different percentages of done items

It appears that from 50% onward, the performance metrics are much better and, thus, the prediction will be more reliable.

Predicting individual item grades

As valuable as predicting final grades can be, it is also possible to use these same algorithms to predict intermediate grade items, so as to be capable of tackling possible problems or difficulties as students are going through the course.

This is done by training, for each item that has not been done yet, a model that takes into account the already completed items and predicts individual intermediate grades as if they were final grades. The Figure 30 is quite explanatory.

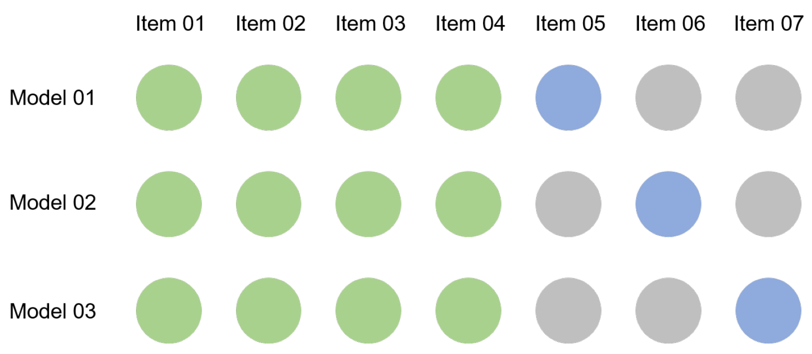


Figure 30: How to predict intermediate items

Imagine a moment in time were students have already completed 4 out of 7 items of the course. We can take those 4 completed items as our features and, for each item that needs to be predicted (5, 6 and 7), create a model were the target is said item (for item 5, our target label will be the grade of item 5, etc.). This way, we can have a way of predicting the progress throughout the course, instead of just knowing the final grade. This can help professors take measures earlier and with a more informed view.

With this new approach, for a student (record) it is necessary to edit a bit the code that generates the final grade and change it so that we can get its whole course grades (with real and predicted grades). The result of one student as an example is shown in the following Figure 31.

In the example, the blue crosses are the real grades that the student obtained and the red ones are the predicted values. The X-axis has the numbers of the items predicted. This example was done with 150 items already done by the student. This model also has an approximation applied. Most grades have either a 0.0 or 1.0 grade, but the model does not know that so many predicted grades are not exactly what they should be (i.e., a 1.0 grade gets predicted as 0.8). To account for that, a condition for approximating the grades is introduced, thus giving us the shown result. In the next Figure 32, the change of applying the condition is seen:

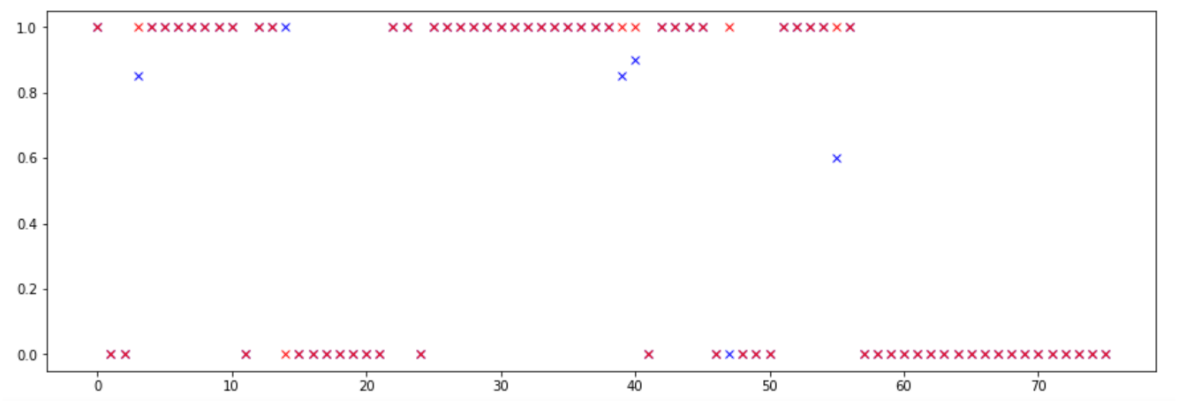


Figure 31: Example of prediction of individual items for a student

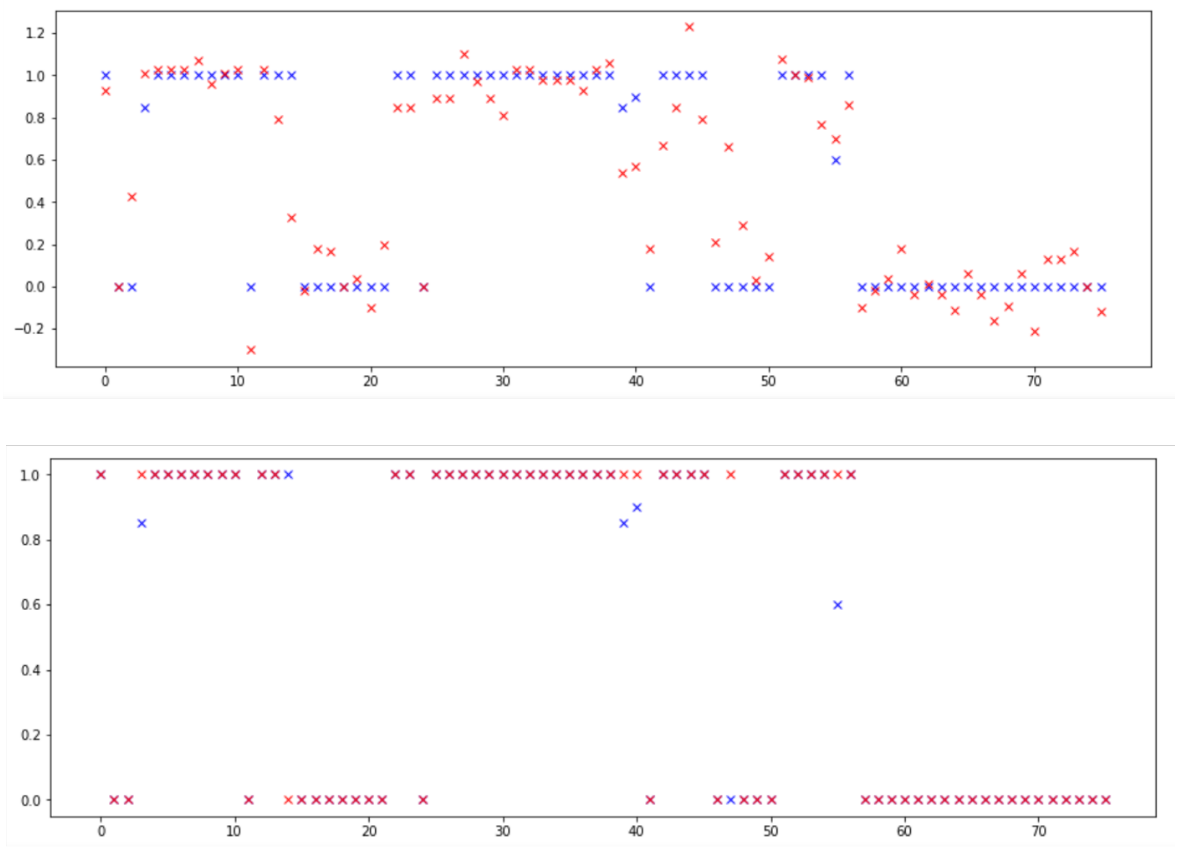


Figure 32: Example of applying the approximation

5 Conclusions

This project has been divided into two different parts, the live elements functionality development and the data analysis study and first development. Before judging the achievements and conclusions of each of these parts, it has been found convenient to remark the importance of both developments, as they are taking Codex a step further in the online learning field. Being able to collect data, in its various forms, and making it available for professors to evaluate how well the experience meets the learning requirements is key to improving such learning.

The scope of this project included the development of a functionality for making items available for students to answer when the professor wanted. This had to be done by giving the professor an easy way to activate and deactivate the items during class, and it should be responsive enough to not have a noticeable delay between the activating and visualization time for students, and it had to be scalable to many students. Plus, the structure for this development had to respect and reuse as much of the previous code from the application as possible.

After developing the functionality and testing it in real life, it can be stated that the objective has been met. The tool is now available for professors and students and it is working well. The professor can activate and deactivate the items easily by checking and unchecking some boxes right near the items. They can do this from two different menus to give the professor the possibility of using the one that suits him better: either activate and deactivate the items from the items main menu, where he can see all the items from his subject, or activate and deactivate them from the menu for each individual notebook, so he can have them prepared beforehand. Either way, the tool now achieves to show the newly added items to students as soon as the professor activates them. Both students who already have the notebook opened and students who open the notebook after the professor has activated the item can view them, so it is not a problem if a student enters the notebook after the professor has sent the item to students.

As explained in point 3, this functionality has been developed using a Javascript-back end communication technology, Server Sent Events, that introduces an important advantage over the other two possibilities mentioned, AJAX and web sockets. The live items functionality does not need a two way communication technology as the only messages that need to be sent are from server to client (student), so having the client just listen for messages (one way communication) saves much computing time and makes performance better, and makes the application more easily scalable to more users.

Besides, the use of SSE is quite new in such an application, as there is not much bibliography that refers to it. This project aims to be a reference guide for future developers who need to do a software with these characteristics. For that purpose, the code used for SSE is found in the appendix, plus a reference to a github page with a simple example of this technology.

The other part of the project was to study the possibilities of applying data analysis and machine learning algorithms to the data recovered during the use of Codex to obtain valuable information and new tools to improve the app. An initial study and a first machine learning algorithm has been developed, that serves as a basis for future improvements and other algorithms to be developed. The first step of this part of the project was to do research on what machine learning and data analysis were being applied in the field of education. The results of this research can be found in point 4.3, which throws light on the many possibilities that are available. Another important step was to do some research on the different algorithms and methods used for creating those models and predicting the different solutions. Although this project does not explain the different possibilities in depth, it does define and explain the model used for the actual development of the project: linear regression.

The second step of the development was to actually build the model for predicting grades. The obtained model has an accuracy that makes it possible to use it for academic purposes, as professors can predict their students' outcome once they have reached at least the 50% of the course. Although the model has not been implemented on Codex yet, it is a first step for this new dimension of the app. Regarding the implementation of the app, a whole point is dedicated to it in the next point as a future development. Also, there are many more studies and models to be developed, out of the list presented in point 4.3, that can be achieved with the actual data that Codex has.

Another conclusion of this part of the project is the information that could be useful for data analysis and that Codex does not recover yet. During the study of the different possibilities of machine learning models for learning, it was seen that information regarding the time spent in answering each question could be very valuable, but it is not yet captured by the app. This is highly difficult data to recover. Right now, the information we get is the time at which the notebook was opened (by the student) and the time at which the answers were sent. Just taking the difference between these times is not reliable, as there are many factors that can affect the time for sending an answer: Did the student answer the items in order? Did he shut down is PC for a while, but the notebook was still open? Did someone tell him the answers so he did not really spent time on them? So a solution for measuring this is still to be found. This data could help us understand what type of contents the students struggle most with, which one they focus more on, etc.

6 Future developments

Regarding future developments of this project, here are presented two lists with different proposals for improving Codex and the new functionalities created in this project.

Future developments for the live items functionality:

- Make sure that the connection between client and server is correctly shut down when the student leaves the notebook. During the implementation of the functionality, this caused some trouble as the SSE connection would remain open when the student closed the notebook, and a new one would be created when he created a new instance. This caused the server to have many connections to clients and it saturated it.
- Removing messages from the server once all the students have read them. This would liberate memory and performance would improve.
- Adding a results page that is configurable by the professor. Right now, the results window is only shown for a single item, and the winners are determined by correct results and time spent for answering. It would be good to develop a way of showing the results of all the items done in live mode, not just one by one. Also, the professor should have the possibility of choosing the evaluation method that he wants for selecting the winners. The design of the results window could be improved too.
- Adding more live functionalities, like changing the time for answering. The professor should be able to edit the closing time for the notebook and the student should get a notification telling him how much time he has left. This can be easily done with the structure of the live items functionality, as it has been designed to get different types of messages (i.e., add items, delete items, change time, etc.). The main thing left to do would be the front-end development for getting this message and translating it into a functionality.

Future developments for the machine learning functionality:

- Development of a visualizing tool in Codex, both for professors and students. At first this was an objective of the project but due to lack of time it was left out. It would be very useful to have a visual tool that could show students and professors the grade predictions and advise students on what they should focus on in order to improve their grades. With the intermediate grade prediction, where we are able to predict individual item's grades, this could be done by selecting the most weight and worst predicted grade and showing it to the student.

- Development of other models for Codex. Based on point 4.3, many more models could be developed with the available data: student-group forming according to learning behaviour, study of knowledge and learning according to origin and gender of the students, how well tests' questions evaluate knowledge, analysis of forums and its impact on students' grades, etc.
- Introduction of neural networks with multiple layers.

7 Budget

Here follows the detailed budget of this project. It is divided into two parts:

• Software: due to the free availability of all the programs used in this project, this part's total budget does not add anything to the final budget.

• Workforce

Task	Duration	Unitary cost (€/hour)	Total cost
Student	600	6	3600
Professor	100	50	5000
Total Workforce			**8600**

8 References

[1] CALVETE, L., & JUAN, Á. A. (2015). Educational Data Mining and Learning Analytics: differences, similarities, and time evolution. RUSC. Universities and Knowledge Society Journal, 12(3). pp. 98-112. doi: http://dx.doi.org/10.7238/rusc.v12i3.2515

[2] CARÍAS ÁLVAREZ, J. F. - Análisis, Diseño y Prototipo de un Sistema Automatizado de Evaluación. Proyecto Final de Máster, Máster en Ingeniería Industrial. 2017.

[3] RASCHKA, S., & MIRJALILI, V. - Python Machine Learning. Machine Learning and deep Learning with Python, scikit-learn, and TensorFlow. Packt, 2017.

[4] ROMERO, C., & VENTURA, S. (2020) Educational data mining and learning analytics: An updated survey. WIREs Data Mining Knowl Discov. 10:e1355. https://doi.org/10.1002/widm.1355

[5] SANCHEZ, O. - Data Science Takes on Public Education. https://towardsdatascience.com/data-science-takes-on-public-education-f432910ea9f0

[6] U.S. Department of Education, Office of Educational Technology, Enhancing Teaching and Learning Through Educational Data Mining and Learning Analytics: An Issue Brief, Washington, D.C., 2012.

[7] W3schools - Learn Java

[8] W3schools - Learn JavaScript

[9] W3schools - Learn Python

[10] W3schools - Learn SQL

A Appendix 1

Code for the Live Elements functionality

Appendix 1

Industrial Engineering Master Thesis

María Serrano Martínez

Tecnun

University of Navarra

1 Msg class

```
public class Msg{

   int msgType; //Type 0: add item, Type 1: delete item, Type 2: time changes
   int msgValue; //Item id or time

   public Msg(int value, int type){
      this.msgValue=value;
      this.msgType=type;
   }
}
```

2 MsgNoteBooks class

```
public class MsgNoteBooks{

   int msgNbid=0;
   Vector<Msg> msgArray = new Vector<Msg>();
   static int cont=0; //Contador de MsgNoteBooks
   static int pos=0; //Para saber en qu nb aadir el msg

   static Vector <MsgNoteBooks> msgNbArray = new Vector<MsgNoteBooks>();

   public MsgNoteBooks(int num, Msg msg){
      msgNbid = num;
      msgArray.addElement(msg);
      cont ++;
   }

   public static boolean nbIdExists(int num){
      boolean a=false;
      for (int i=0; i<cont; i++){
         if(msgNbArray.elementAt(i).msgNbid == num){
            a=true;
```

```
          pos = i;
          i=cont;
        }
      }
    return a;
  }
}
```

3 EnableNoteBook servlet

```
public class EnableNoteBookItem extends HttpServlet {
  Connection conn = null;

   public void init (ServletConfig config) throws ServletException {
      super.init(config);
      conn = ConnectionManager.createConnection();
   }

   public void destroy () {
      super.destroy();
      ConnectionManager.destroyConnection(conn);
   }

   public void doPost (HttpServletRequest req, HttpServletResponse resp) throws
       ServletException, IOException {
      doGet(req, resp);
   }

   public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
       ServletException, IOException {
      Connection conn = ConnectionManager.getConnection();
      resp.setContentType("text/html");
      PrintWriter out = null;
      try {
         out=resp.getWriter();
```

```
    } catch (IOException io) {
        System.out.println("Exception creating PrintWriter");
    }
```

```
    nid = req.getParameter("nid");
    iid = req.getParameter("iid");
    c = req.getParameter("c");
    System.out.println("nid="+nid);
    System.out.println("iid="+iid);
    int n;
  long loginid = (long)session.getAttribute("loginid");

    if(c.equals("true")){ //MARIA
        n = NoteBookManager.enableNoteBookItem(nid, iid, conn, loginid);
     Msg msg = new Msg(Integer.parseInt(iid), 0); //type 0: add item
     boolean nbidstate = MsgNoteBooks.nbIdExists(Integer.parseInt(nid));
     if(nbidstate == true){
        MsgNoteBooks.msgNbArray.elementAt(MsgNoteBooks.pos).msgArray.addElement(msg);
     }else{
        MsgNoteBooks msgNbs = new MsgNoteBooks(Integer.parseInt(nid), msg);
        MsgNoteBooks.msgNbArray.addElement(msgNbs);
     }
     System.out.println("--------------------------------------");
     System.out.println("MESSAGES ON QUEUE");
     for(int i=0; i<MsgNoteBooks.cont; i++){
        System.out.println("--------------------------------------");
        System.out.println("NB id: " +
            MsgNoteBooks.msgNbArray.elementAt(i).msgNbid);
        for (int j=0; j<MsgNoteBooks.msgNbArray.elementAt(i).msgArray.size();
            j++){
          System.out.println("ITEM id: " +
              MsgNoteBooks.msgNbArray.elementAt(i).msgArray.elementAt(j).msgValue
              + " Type:" +
              MsgNoteBooks.msgNbArray.elementAt(i).msgArray.elementAt(j).msgType);
        }
     }
    }else{
        n = NoteBookManager.disableNoteBookItem(nid, iid, conn);
     Msg msg = new Msg(Integer.parseInt(iid), 1); //type 1: delete item
     boolean nbidstate = MsgNoteBooks.nbIdExists(Integer.parseInt(nid));
     if(nbidstate == true){
        MsgNoteBooks.msgNbArray.elementAt(MsgNoteBooks.pos).msgArray.addElement(msg);
     }else{
        MsgNoteBooks msgNbs = new MsgNoteBooks(Integer.parseInt(nid), msg);
        MsgNoteBooks.msgNbArray.addElement(msgNbs);
     }
     System.out.println("---------------------------------------");
     System.out.println("MESSAGES ON QUEUE");
     for(int i=0; i<MsgNoteBooks.cont; i++){
        System.out.println("---------------------------------------");
        System.out.println("NB id: " +
            MsgNoteBooks.msgNbArray.elementAt(i).msgNbid);
        for (int j=0; j<MsgNoteBooks.msgNbArray.elementAt(i).msgArray.size();
            j++){
          System.out.println("ITEM id: " +
              MsgNoteBooks.msgNbArray.elementAt(i).msgArray.elementAt(j).msgValue
              + " Type: " +
              MsgNoteBooks.msgNbArray.elementAt(i).msgArray.elementAt(j).msgType);
        }
     }

    }
    out.print(n);
    out.flush();
```

```
        out.close();
        ConnectionManager.closeConnection(conn);
    }
}
```

4 JavaScript for starting the SSE connection

```
function startLiveState(x){ //MARIA
     var url = "checkMsg?nid=" + nid + "&stname=" + stname;
         var source = new EventSource(url);
         source.onmessage = function(event){
            var jsonData = event.data;
            var newData = JSON.parse(jsonData);
            var newItemSSE = newData.message;
            var newItemType = newData.type;
            if(newItemType==0){       //Type 0: add new live item
               getItemData(newItemSSE,x);
            } else if(newItemType==1){     //Type 1: delete live item
               deleteItem(newItemSSE);
            }
         }
}

function getItemData(newItemSSE,x){ //MARIA
   var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
         if (this.readyState == 4 && this.status == 200) {
            var newItemSSEData = JSON.parse(this.responseText);
            printItems(x, newItemSSEData);
         }
        };
        xhttp.open("GET", "SSEInstance?itemid=" + newItemSSE + "&instanceid=" +
            instanceId, true);
        xhttp.send();
}
```

```
function deleteItem(itemId){ //MARIA
   var element = document.getElementById(itemId);
   var htmlButton = document.getElementById("button"+parseInt(itemId));
   htmlButton.disabled = true;
   document.getElementById(itemId).style.width = "0%";
   setTimeout(()=> removeItem(element), 800);
}

function removeItem(element){
   element.parentNode.removeChild(element);
   checkNumberOfItems();
}
```

5 checkMsg servlet

```
public class checkMsg extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{

      // Initialize lastValue for each student running the servlet
      int lastValue=0;
      int positionNb=-1;
      int temp = 0;

      // Set response content type
         response.setContentType("text/event-stream");
      response.setCharacterEncoding("UTF-8");
      System.out.println("SSE mode started");

        PrintWriter out = null;

      String nid = request.getParameter("nid");
      String stname = request.getParameter("stname");
```

```
        try {
            out = response.getWriter();
        } catch (IOException io) {
            System.out.println("Exception creating PrintWriter");
        }


    while(positionNb== -1){
       if(MsgNoteBooks.cont==0){temp=1;}
       for(int i=0; i<MsgNoteBooks.cont; i++){
          if
              (nid.equals(Integer.toString(MsgNoteBooks.msgNbArray.elementAt(i).msgNbid))){
             positionNb = i;
          }
       }
    }
    if(temp==0){
       lastValue = MsgNoteBooks.msgNbArray.elementAt(positionNb).msgArray.size();
       System.out.println("My last value is: " + lastValue);
    }

    while(true){
       for (int i=(lastValue);
           i<MsgNoteBooks.msgNbArray.elementAt(positionNb).msgArray.size(); i++){
          String next = "data: {\"message\": \"";
          next +=
              Integer.toString(MsgNoteBooks.msgNbArray.elementAt(positionNb).msgArray.elementAt(i
          next += "\", \"type\": \"";
          next +=
              Integer.toString(MsgNoteBooks.msgNbArray.elementAt(positionNb).msgArray.elementAt(i
          next += "\"}\n\n"; //SSE doesn't work without \n\n
          System.out.println("next: " + next);
          out.write(next);
          out.flush();
          lastValue++;
```

```
      }

      try{
         Thread.sleep(100); //Change for faster replies
      } catch(InterruptedException ex) {
         Thread.currentThread().interrupt();
      }

    }
  }
}
```

6 SSEInstance servlet

```
public class SSEInstance extends HttpServlet {
   Connection conn = null;

    public void init (ServletConfig config) throws ServletException {
        super.init(config);
        conn = ConnectionManager.createConnection();
    }

    public void destroy () {
        super.destroy();
        ConnectionManager.destroyConnection(conn);
    }

    public void doPost (HttpServletRequest req, HttpServletResponse resp) throws
         ServletException, IOException {
        doGet(req, resp);
    }

    public void doGet (HttpServletRequest req, HttpServletResponse resp) throws
         ServletException, IOException {
        Connection conn = ConnectionManager.getConnection();
        resp.setContentType("text/html");
        PrintWriter out = null;
        try {
            out=resp.getWriter();
        } catch (IOException io) {
            System.out.println("Exception creating PrintWriter");
        }
        HttpSession session = req.getSession(false);

        String iid = req.getParameter("itemid");
      String nbinstid = req.getParameter("instanceid");



      String itemInstance = NoteBookManager.getItemData(Integer.parseInt(nbinstid),
          conn, iid);

      String html = itemInstance;

      out.write(html);
      out.flush();
      out.close();




        ConnectionManager.closeConnection(conn);
    }
}
```

B Appendix 2

Data PreProcessing_Predict grades based on past items

April 18, 2021

1 Getting data into shape

1.0.1 Import the data to Python

Import the useranswer table from the database from a csv file. To create the cvs file with data from Squirrel database: 1. Run a query to obtain the table 2. Session > Scripts > Store result of SQL in file 3. Select "Export CVS file" and Charset: UTF-8

SQL: SELECT USERID, USERANSWER.ITEMID, MARK, ANSWERDATE, MAXGRADE FROM USERANSWER, LOGIN, ITEM WHERE LOGIN.LOGINID = USERANSWER.LOGINID AND ANSWERDATE <=('2020-09-01 00:00:00.0') AND ITEM.ITEMID = USERANSWER.ITEMID

[1]:
```python
# Import the dataset as a pandas dataframe (original_df)
import pandas as pd
import numpy as np

original_df = pd.read_csv('C:/Users/María/Documents/Tecnun/Master/PFM/
 ↪01_DataPreprocessing/Useranswer')
max_grade_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\01_DataPreprocessing\UseranswerWithMaxGrade')
item_dates = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\01_DataPreprocessing\ItemDates')
item_type_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\ItemType')

display(max_grade_df)
```

```
          USERID  ITEMID  MARK              ANSWERDATE  MAXGRADE
0     1058277809    4214   0.0  2020-01-24 15:51:53.373       1.0
1     1058277809    4214   0.0  2020-01-29 22:08:48.055       1.0
2     1058277809    4498   1.0  2020-01-30 17:35:54.352       1.0
3     1058277809    4499   0.0  2020-01-24 16:22:35.452       NaN
4     1098327170    4499   0.0  2020-01-24 17:40:56.535       NaN
...          ...     ...   ...                      ...       ...
7416  1260649978    5386   0.0  2020-06-02 17:46:34.055      14.0
7417  1203634691    5383  12.0  2020-06-02 17:05:48.319      12.0
7418  1203634691    5384   0.0  2020-06-02 17:41:02.427      12.0
7419  1203634691    5385  12.0  2020-06-02 17:19:37.827      12.0
```

```
7420  1203634691    5386   0.0  2020-06-02 17:37:51.224      14.0

[7421 rows x 5 columns]
```

1.0.2 Create the new DataFrame (with the appropiate design)

We group data according to USERID (each row will contain the data from each student). For doing that, we first create a blank dataframe with the appropiate design of rows and columns.

[5]:
```python
items = item_dates.ITEMID.unique()
item_type = np.zeros(len(items))
#print(items)

a=0

for x in items:
    for i in range(item_type_df.shape[0]):
        if x == item_type_df.loc[i, 'ITEMID']:
            item_type[a] = item_type_df.loc[i, 'ITEMTYPEID']
            a = a+1
            break

item_type = item_type.astype(int)
#print(item_type)
```

[6]:
```python
# Get unique values of Items and UsersId
items_cat = max_grade_df.ITEMID.unique()
users_id = max_grade_df.USERID.unique()
#print(items_cat)

# Using list comprehension to initialize matrix
N = users_id.size # Rows
M = items_cat.size  # Columns
values_matrix = [ [ 0 for i in range(M) ] for j in range(N) ]

# Create blank dataframe
students_df = pd.DataFrame(values_matrix, index=users_id, columns=items)
users_df = pd.DataFrame(users_id, columns=['USERID'])
#print(users_df)
#print(students_df)
```

We now proceed to fill in the empty cells for each student.

There are a few items with no Max_grade property, so we are going to calculate it based on the maximum grade obtained by the students.

[8]:
```python
number_null = np.zeros(max_grade_df['MAXGRADE'].isnull().sum(), dtype=int)
import math
```

```
i=0
for x in range(max_grade_df.shape[0]):
    if math.isnan(max_grade_df.loc[x, 'MAXGRADE']):
        number_null[i] = max_grade_df.loc[x,'ITEMID']
        i=i+1

null_items = list(set(number_null))
#print(null_items)
```

[9]:
```
# Calculate the maximum grade achieved by students for the items in null_items
for x in null_items:
    max_temp = 0.0
    for i in range(max_grade_df.shape[0]):
        if max_grade_df.loc[i, 'ITEMID'] == x:
            if max_grade_df.loc[i, 'MARK'] >= max_temp:
                max_temp = max_grade_df.loc[i, 'MARK']
    for j in range(max_grade_df.shape[0]):
        if max_grade_df.loc[j, 'ITEMID'] == x:
            max_grade_df.loc[j, 'MAXGRADE'] = max_temp
    print('MAX GRADE FOR ITEM ' + str(x) + ' is ' + str(max_temp))
```

```
MAX GRADE FOR ITEM 4448 is 1.0
MAX GRADE FOR ITEM 4518 is 0.0
MAX GRADE FOR ITEM 4499 is 0.0
MAX GRADE FOR ITEM 4694 is 0.0
MAX GRADE FOR ITEM 4537 is 5.0
MAX GRADE FOR ITEM 5049 is 1.0700000524520874
MAX GRADE FOR ITEM 4414 is 1.0
MAX GRADE FOR ITEM 4447 is 1.0
```

[11]:
```
# Fill each value (student-item) with its matching mark
i=0
for x in max_grade_df.USERID:
    mark = max_grade_df.loc[i, 'MARK']
    itemid = max_grade_df.loc[i, 'ITEMID']
    max_grade = max_grade_df.loc[i, 'MAXGRADE']

    if max_grade == 0.0:
        actual_grade=0.0
    else:
        actual_grade = mark/max_grade

    i=i+1

    for j in range(items_cat.size):
        if items_cat[j] == itemid:
            students_df.loc[x,(items_cat[j])] = actual_grade
```

```
cols = students_df.columns
for col in cols:
    students_df[col] = students_df[col].astype(float)
#print(students_df)
```

Once we have all the information for each student, we import the target values array (final grade) as a separate series

```
[14]: # Import from csv file
grades_df = pd.read_csv('C:/Users/María/Documents/Tecnun/Master/PFM/
 ↪01_DataPreprocessing/grades.csv', sep=';')
#print (grades_df)
```

```
[16]: # Check to see if we have users without final grades
print('Number of students from dataset: ' + str(students_df.shape[0]))
print('Number of students with actual grade: ' + str(grades_df.shape[0]))

# Compare both series, first order each one
users_df = users_df.sort_values(by=['USERID'])
grades_df = grades_df.sort_values(by=['USERID'])

# Find missing element
users_vector = users_df.USERID.unique()
grades_vector = grades_df.USERID.unique()
#print(users_vector)
#print(grades_vector)

for x in range(users_vector.size):
    if users_vector[x] != grades_vector[x]:
        missing_element = users_vector[x]
        print('Element missing: ' + str(missing_element))
        break
```

```
Number of students from dataset: 45
Number of students with actual grade: 44
Element missing: 1058277809
```

```
[17]: # Drop the missing record from the students_df
students_df_final = students_df.drop([missing_element])
students_df_final = students_df_final.sort_index()
#print(students_df_final)
```

1.0.3 Removing and imputing missing values from dataset

We have missing values, which are represented as NaN values. In this case, we are going to assume that all those who didn't answer the question and, hence, have a NaN record for that item, have a 0.0 mark. Thus, we are now going to convert all the NaN values to 0.

```
[18]: students_df_final = students_df_final.fillna(0.0)
      #print(students_df_final)
```

1.0.4 Save the new DataFrame and the target values vector

We save the edited DataFrame and the target values vector in different csv files to be able to load them from different jupyter notebooks. Note that we do save the values of students_df_final index to be able to convert them into a new column when we import it again in the next step.

```
[19]: students_df_final.to_csv(r'C:
       ↪\Users\María\Documents\Tecnun\Master\PFM\01_DataPreprocessing\students_df_final',␣
       ↪sep=',')
      grades_df.to_csv(r'C:
       ↪\Users\María\Documents\Tecnun\Master\PFM\01_DataPreprocessing\grades_df.
       ↪csv', sep=',',index=False)
```

```
[20]: display(students_df)
```

```
            4214  4499  4498  4217  4500  4215  4218  4252  4219  4220  ...  \
1058277809   0.0   0.0   1.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1098327170   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  0.00   0.0  ...
1267281825   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1200521951   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1225721435   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1245744606   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1039454863   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1203634691   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1217990533   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1199511965   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1083182240   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   0.0  ...
1256116316   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   0.5  ...
1143849676   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   0.5  ...
1062206845   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1010498129   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1177419780   0.0   0.0   0.0   1.0   1.0   1.0   1.0   0.6  0.00   0.5  ...
1226900861   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1187887289   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1195822211   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1230106397   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1001878881   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1264826547   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  0.50   1.0  ...
1042775892   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  0.00   0.0  ...
1011762717   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1115820224   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1009074790   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1161766484   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1077598052   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1183300975   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
```

```
1059658959   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  0.75   0.5  ...
1255647652   0.0   0.0   0.0   1.0   1.0   1.0   1.0   0.0  0.00   0.0  ...
1034515793   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1109305111   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1263822766   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1246874373   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1069422119   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1087192798   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1260649978   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1008562537   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  0.50   0.5  ...
1123907581   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1236980620   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  1.00   1.0  ...
1009358503   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  0.75   0.5  ...
1151697612   0.0   0.0   0.0   1.0   1.0   1.0   1.0   1.0  0.50   1.0  ...
1181253520   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  0.00   0.0  ...
1177614143   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  0.00   0.0  ...

                 5388  5391  5393   5390  5392  5384  5385  5383  5386  \
1058277809  1.000000   1.0   1.0  0.725   1.0   1.0   1.0   1.0   1.0
1098327170  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1267281825  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1200521951  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1225721435  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1245744606  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1039454863  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1203634691  0.933333   0.0   1.0  0.750   0.0   0.0   1.0   1.0   0.0
1217990533  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1199511965  1.000000   1.0   1.0  0.875   0.6   0.0   0.0   0.0   0.0
1083182240  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1256116316  0.800000   0.0   1.0    NaN   0.0   0.0   0.0   0.0   0.0
1143849676  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1062206845  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1010498129  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1177419780  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1226900861  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1187887289  1.000000   1.0   1.0  0.725   0.0   0.0   0.0   0.0   0.0
1195822211  1.000000   NaN   1.0  0.725   0.0   1.0   1.0   1.0   0.0
1230106397  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1001878881  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1264826547  1.000000   1.0   1.0  0.875   0.0   0.0   0.0   0.0   0.0
1042775892  1.000000   1.0   1.0  0.800   0.0   0.0   0.0   0.0   0.0
1011762717  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1115820224  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1009074790  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1161766484  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1077598052  0.933333   1.0   1.0  0.000   1.0   0.0   0.0   0.0   0.0
1183300975  1.000000   1.0   0.6  1.000   1.0   0.0   0.0   0.0   0.0
1059658959  1.000000   0.0   1.0  0.725   0.0   0.0   0.0   1.0   0.0
```

```
1255647652  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1034515793  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1109305111  1.000000   NaN   1.0  0.475   0.0   0.0   0.0   0.0   0.0
1263822766  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1246874373  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1069422119  1.000000   0.0   1.0  0.475   0.0   0.0   0.0   0.0   0.0
1087192798  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1260649978  0.933333   1.0   0.0  0.675   0.0   1.0   1.0   1.0   0.0
1008562537  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1123907581  0.933333   0.0   1.0  0.275   0.0   0.0   0.0   0.0   0.0
1236980620  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1009358503  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1151697612  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1181253520  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0
1177614143  0.000000   0.0   0.0  0.000   0.0   0.0   0.0   0.0   0.0

                5389
1058277809       NaN
1098327170  0.000000
1267281825  0.000000
1200521951  0.000000
1225721435  0.000000
1245744606  0.000000
1039454863  0.000000
1203634691  0.900000
1217990533  0.000000
1199511965  1.000000
1083182240  0.000000
1256116316  0.266667
1143849676  0.000000
1062206845  0.000000
1010498129  0.000000
1177419780  0.000000
1226900861  0.000000
1187887289  1.000000
1195822211  1.000000
1230106397  0.000000
1001878881  0.000000
1264826547  0.800000
1042775892  0.600000
1011762717  0.000000
1115820224  0.000000
1009074790  0.000000
1161766484  0.000000
1077598052  1.000000
1183300975  1.000000
1059658959  1.000000
1255647652  0.000000
```

```
1034515793  0.000000
1109305111  0.800000
1263822766  0.000000
1246874373  0.000000
1069422119  0.800000
1087192798  0.000000
1260649978  0.000000
1008562537  0.000000
1123907581  0.900000
1236980620  0.000000
1009358503  0.000000
1151697612  0.000000
1181253520  0.000000
1177614143  0.000000

[45 rows x 227 columns]
```

Visualize Data_Predict grades based on past items

April 18, 2021

1 Visualize data to find relationships

In this step we are going to visualize the data and try to perform dimensionality reduction. As a first step, we noticed during DataFrame construction that many features (items) had a 0 mark for all the students in the class. This can be due to those items being non-gradable, rather than all of them getting them wrong. As it won't give the model any information, we can start by eliminating those features.

[29]:
```python
# Import necessary libraries and files
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

students_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\01_DataPreprocessing\students_df_final',sep=',',␣
 ↪header=0)
grades_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\01_DataPreprocessing\grades_df.
 ↪csv',sep=',')

students_df = students_df.rename(columns={"Unnamed: 0": "USERID"})
grades_df.loc[:,'GRADE'] = grades_df.loc[:, 'GRADE'] / 10
#print(students_df)
#print(grades_df)
```

For each column feature in students_df, check if all elements are 0. If they are, then drop the column and save the item information in a different vector.

[30]:
```python
#### We build an array with the column headers (items)
items_cat = students_df.columns.values.tolist()
items_cat.pop(0)
items_cat = [ int(x) for x in items_cat ]
#print('Original items')
#print(items_cat)

num_eliminated=0
items_eliminated = np.zeros(len(items_cat,), dtype=int)
# For each column, check the number of elements with 0 value
```

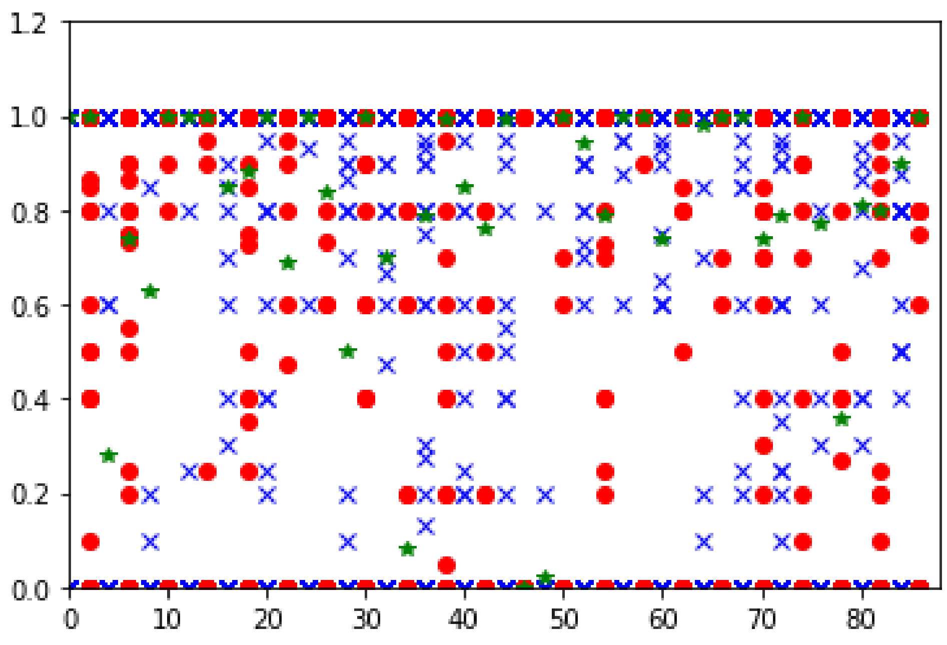
```
for j in range(len(items_cat)):
    num_zeros=0
    for x in range(students_df.shape[0]):
        if students_df.loc[x,str(items_cat[j])] == 0:
            num_zeros = num_zeros + 1
        if x == students_df.shape[0]-1:
            if num_zeros == x+1:
                # Eliminate column
                students_df.drop(str(items_cat[j]), inplace=True, axis=1)
                #del students_df[str(items_cat[j])]
                # Save item number
                items_eliminated[num_eliminated] = items_cat[j]
                num_eliminated = num_eliminated + 1

items_eliminated = items_eliminated[0:num_eliminated]
#print('Items eliminated ' + str(num_eliminated))
#print(items_eliminated)
#print('Students records without irrelevant items')
#print(students_df)
```

Once we have all the neccessary data in the correct format, we are going to visualize the data for each student. Firstly, we are going to visualize it just as points, not taking into account the answer date.

[31]:
```
# Items_cat has changed, we have to eliminate the irrelevant items
items_cat = students_df.columns.values.tolist()
items_cat.pop(0)
items_cat = [ int(x) for x in items_cat ]
#print(items_cat)
```

[32]:
```
# Import the necessary libraries and plot X(students) and Y(mark)
import matplotlib.pyplot as plt

X = students_df.loc[:,'USERID']
X = X.values.tolist()
print(X)

axes = plt.gca()
axes.set_xlim([0,44*2])
axes.set_ylim([0,1.2])

a=-1
for i in X:
    a=a+1
    for j in items_cat:
        if a%2 == 0:
            colBR= 'blue'
            mar = 'x'
        else:
            colBR='red'
            mar = 'o'
        plt.plot(a*2, students_df.loc[a, str(j)], color=colBR, marker=mar)
    plt.plot(a*2, grades_df.loc[a, 'GRADE'], color = 'g', marker = '*')
```

```
[1001878881, 1008562537, 1009074790, 1009358503, 1010498129, 1011762717,
1034515793, 1039454863, 1042775892, 1059658959, 1062206845, 1069422119,
1077598052, 1083182240, 1087192798, 1098327170, 1109305111, 1115820224,
1123907581, 1143849676, 1151697612, 1161766484, 1177419780, 1177614143,
1181253520, 1183300975, 1187887289, 1195822211, 1199511965, 1200521951,
1203634691, 1217990533, 1225721435, 1226900861, 1230106397, 1236980620,
1245744606, 1246874373, 1255647652, 1256116316, 1260649978, 1263822766,
1264826547, 1267281825]
```



[33]:
```
# Students percentages
students_userid = students_df['USERID'].values.tolist()

students_percentages = np.zeros([len(percentages), 2])
for x in range(len(percentages)):
    students_percentages[x, 0] = students_userid[x]
    students_percentages[x, 1] = percentages[x]

students_percentages_df = pd.DataFrame(students_percentages, columns=['UserID',␣
 ↪'Percentage'])
```

```
students_percentages_df = students_percentages_df.astype({"UserID":'int',␣
↪"Percentage":'float'})

print(students_percentages_df)
print('HIGHER THAN 40')
print (students_percentages_df[(students_percentages_df >= 40.0).all(1)])
```

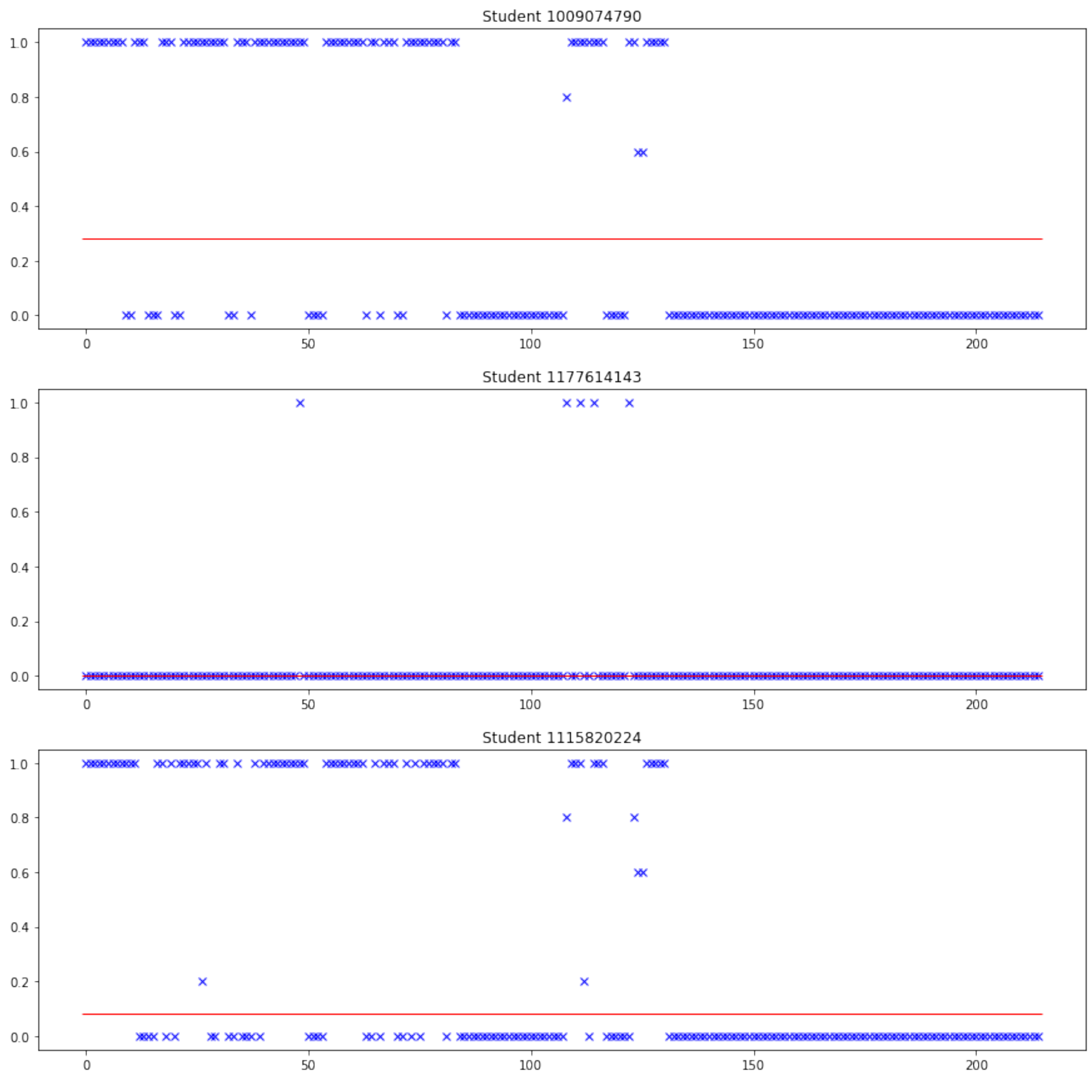
```
        UserID  Percentage
0   1001878881   14.883721
1   1008562537   18.604651
2   1009074790   41.860465
3   1009358503   21.860465
4   1010498129   18.604651
5   1011762717   17.209302
6   1034515793   15.813953
7   1039454863   15.348837
8   1042775892   17.674419
9   1059658959   19.069767
10  1062206845   16.279070
11  1069422119   42.325581
12  1077598052   12.093023
13  1083182240   16.279070
14  1087192798   28.372093
15  1098327170   16.279070
16  1109305111   22.790698
17  1115820224   42.790698
18  1123907581   17.674419
19  1143849676   16.744186
20  1151697612   15.813953
21  1161766484   14.883721
22  1177419780   16.744186
23  1177614143   48.372093
24  1181253520   46.046512
25  1183300975   33.953488
26  1187887289   13.953488
27  1195822211   14.418605
28  1199511965   12.558140
29  1200521951   18.604651
30  1203634691   14.418605
31  1217990533   13.488372
32  1225721435   21.395349
33  1226900861   19.534884
34  1230106397   17.674419
35  1236980620   18.139535
36  1245744606   16.744186
37  1246874373   18.139535
38  1255647652   30.232558
39  1256116316   41.395349
```

```
40  1260649978   33.953488
41  1263822766   17.674419
42  1264826547   15.813953
43  1267281825   13.953488
HIGHER THAN 40
        UserID  Percentage
2   1009074790   41.860465
11  1069422119   42.325581
17  1115820224   42.790698
23  1177614143   48.372093
24  1181253520   46.046512
39  1256116316   41.395349
```

[34]:
```
# Plot x students

student_array = [2, 23, 17]

fig, axs = plt.subplots(len(student_array), figsize=(15,5*len(student_array)))

i=0

for student_index in student_array:
    a=0
    perc0=0
    for x in items_cat:

        axs[i].plot(a, students_df.loc[student_index, str(x)], color='blue',␣
 ↪marker='x')
        axs[i].plot(a, grades_df.loc[student_index, 'GRADE'], color='red',␣
 ↪marker='_')
        a=a+1

        if a>= len(items_cat)/2:
            if students_df.loc[student_index, str(x)] == 0.0:
                perc0=perc0 + 1

    print("Student: " + str(students_df.loc[student_index, 'USERID']) + "␣
 ↪Number of 0 in second half: " + str(perc0) + ' Final grade: ' +␣
 ↪str(round(grades_df.loc[student_index, 'GRADE'], 2)))
    perc0=perc0/len(items_cat)*100
    axs[i].set_title('Student ' + str(students_df.loc[student_index, 'USERID']))
#    axs[i].set_title("Student " + str(students_df.loc[student_index,␣
 ↪'USERID']) + "      Percentage: " + str(perc0) + "%")
    i=i+1
```

```
Student: 1009074790 Number of 0 in second half: 90 Final grade: 0.28
Student: 1177614143 Number of 0 in second half: 104 Final grade: 0.0
```

```
Student: 1115820224 Number of 0 in second half: 92 Final grade: 0.08
```



We are going to eliminate the students that drop the course at the middle of the semester. These are the ones who have a percentage >= 40.0 and whose final grade is >=0.05 (the ones below are the ones who are failing the course but who do not drop at the middle of the year).

```
[35]: # Save the id of the students to eliminate

students_higher_than_40_df = students_percentages_df.copy()
students_higher_than_40_df = students_higher_than_40_df.
 ↪drop(students_higher_than_40_df[students_higher_than_40_df.Percentage < 40.
 ↪0].index)

students_higher_than_40_ix = students_higher_than_40_df.index
```

```
students_higher_than_40_grades = grades_df.loc[students_higher_than_40_ix,␣
 ↪'GRADE']

a=0
for x in students_higher_than_40_grades:
    if x<0.03:
        ix_drop = students_higher_than_40_ix[a]
        students_higher_than_40_df = students_higher_than_40_df.drop([ix_drop])
    a=a+1

students_to_drop = students_higher_than_40_df.index
print(students_higher_than_40_df)
print(students_to_drop)
```

```
        UserID  Percentage
2   1009074790   41.860465
11  1069422119   42.325581
17  1115820224   42.790698
39  1256116316   41.395349
Int64Index([2, 11, 17, 39], dtype='int64')
```

```
[36]: # Copy students_df and drop the outliers
students_no_outliers_df = students_df.copy()
grades_no_outliers_df = grades_df.copy()

students_no_outliers_df = students_no_outliers_df.drop(students_to_drop)
grades_no_outliers_df = grades_no_outliers_df.drop(students_to_drop)
#print(students_no_outliers_df)
#print(grades_no_outliers_df)
```

We save the edited data in different cvs files to be used on step 3 (model).

```
[37]: # Save to csv files
students_df.to_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\students_df.
 ↪csv', sep=',', index=False)
grades_df.to_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\grades_df.
 ↪csv', sep=',',index=False)

students_no_outliers_df.to_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\students_no_outliers_df.
 ↪csv', sep=',', index=False)
grades_no_outliers_df.to_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\grades_no_outliers_df.
 ↪csv', sep=',',index=False)
```

Model_Predict grades based on past items-Copy1

April 18, 2021

1 Create model for predicting final grades

In this step we are going to load the pre-processed data and construct the model with a regression analysis.

[63]:
```
# Import the data and the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

students_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\students_df.
 ↪csv',sep=',', header=0)
grades_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\grades_df.
 ↪csv',sep=',')

students_no_outliers_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\students_no_outliers_df.
 ↪csv',sep=',', header=0)
grades_no_outliers_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\grades_no_outliers_df.
 ↪csv',sep=',')

nb_items = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\NotebookItems')
item_type_df = pd.read_csv(r'C:
 ↪\Users\María\Documents\Tecnun\Master\PFM\02_DataVisualization\ItemType')
#display(students_df)
```

[64]:
```
items_cat = students_df.columns.values.tolist()
items_cat.pop(0)
#print(round(25*228/100,0))
```

[64]:
```
'USERID'
```

1.0.1 Estimating coefficient of a regression model via scikit-learn

For items [0:150] we are going to estimate the final grade using a Linear Regression model

[89]:
```
# Import metrics
from sklearn.metrics import explained_variance_score
from sklearn.metrics import max_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import r2_score
```

[90]:
```
# Splitting the data into training and test samples
number_items = int(round(75*228/100,0))
#number_items=150
#print(number_items)

X = students_df.loc[:,(items_cat[0:number_items])]
y = grades_df.loc[:, 'GRADE']

X_no = students_no_outliers_df.loc[:,(items_cat[0:number_items])]
y_no = grades_no_outliers_df.loc[:, 'GRADE']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=1)
X_no_train, X_no_test, y_no_train, y_no_test = train_test_split(X_no, y_no,␣
↪test_size=0.3, random_state=1)

train_students_index = X_train.index
test_students_index = X_test.index

train_no_students_index = X_no_train.index
test_no_students_index = X_no_test.index
```

[91]:
```
# Results for linear regression without regularization
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_train, y_train)
y_pred = slr.predict(X_test)
y_pred = y_pred.round(2)

slr.fit(X_no_train, y_no_train)
y_no_pred = slr.predict(X_no_test)
y_no_pred = y_no_pred.round(2)

y_slr = y_no_pred
```

[92]:
```
# Accuracy metrics
data = np.zeros([6,2])

exp_var = round(explained_variance_score(y_test,y_pred),2)
exp_var_no = round(explained_variance_score(y_no_test,y_no_pred),2)
data[0,:] = [exp_var, exp_var_no]

max_err = round(max_error(y_test, y_pred), 2)
max_err_no = round(max_error(y_no_test, y_no_pred), 2)
data[1,:] = [max_err, max_err_no]

mean_err = round(mean_absolute_error(y_test, y_pred), 2)
mean_err_no = round(mean_absolute_error(y_no_test, y_no_pred), 2)
data[2,:] = [mean_err, mean_err_no]

mean_sq_err = round(mean_squared_error(y_test, y_pred), 2)
mean_sq_err_no = round(mean_squared_error(y_no_test, y_no_pred), 2)
data[3,:] = [mean_sq_err, mean_sq_err_no]

med_err = round(median_absolute_error(y_test, y_pred), 2)
med_err_no = round(median_absolute_error(y_no_test, y_no_pred), 2)
data[4,:] = [med_err, med_err_no]

r2_sc = round(r2_score(y_test, y_pred), 2)
r2_sc_no = round(r2_score(y_no_test, y_no_pred), 2)
data[5,:] = [r2_sc, r2_sc_no]

print('Regularization method: slr')
index = ['Explained variance', 'Max error (grade pt)', 'Mean absolute error␣
↪(grade pt)', 'Mean squared error (grade pt^2)', 'Median absolute error␣
↪(grade pt)', 'R2 score']
accuracy_metrics = pd.DataFrame(data, columns=['With outliers', 'Without␣
↪outliers'], index=index)
display(accuracy_metrics)
```

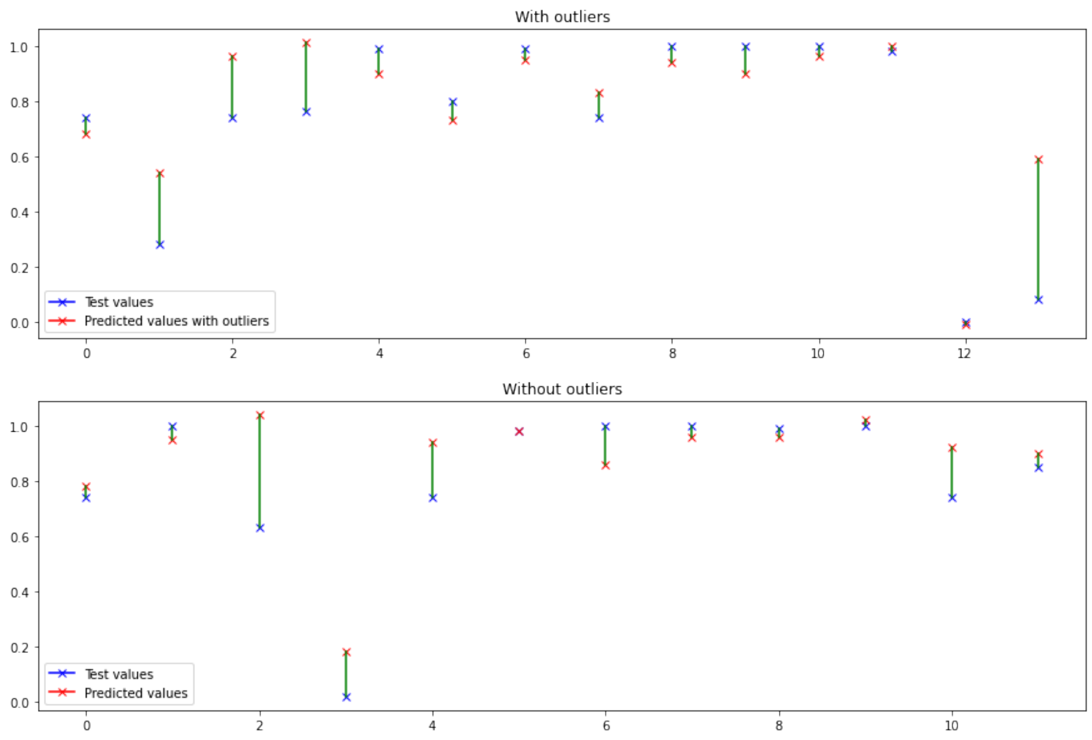
```
Regularization method: slr

                                  With outliers  Without outliers
Explained variance                         0.73              0.72
Max error (grade pt)                       0.51              0.41
Mean absolute error (grade pt)             0.13              0.11
Mean squared error (grade pt^2)            0.03              0.02
Median absolute error (grade pt)           0.08              0.05
R2 score                                   0.70              0.66
```

[69]:
```
# Plot with and without outliers
fig, axs = plt.subplots(2, figsize=(15,5*2))
```

```
print(regularization)

# With outliers
y_test = y_test.reset_index(drop=True)
y_test.tolist()
errors = np.zeros(y_pred.size)
a=0
for i in range(y_pred.size):
    if(i==1):
        axs[0].plot(a, y_test[i], color='blue', marker='x', label='Test values')
        axs[0].plot(a, y_pred[i], color='red', marker='x', label='Predicted␣
↪values with outliers')
        errors[i] = y_test[i] - y_pred[i]

        axs[0].legend()
    else:
        axs[0].plot(a, y_test[i], color='blue', marker='x')
        axs[0].plot(a, y_pred[i], color='red', marker='x')
        errors[i] = y_test[i] - y_pred[i]


    x_values = [a, a]
    y_values = [y_test[i], y_pred[i]]
    axs[0].plot(x_values, y_values, color='green')

    a=a+1

axs[0].set_title("With outliers")

# Without outliers
y_no_test = y_no_test.reset_index(drop=True)
y_no_test.tolist()
errors = np.zeros(y_no_pred.size)
a=0
for i in range(y_no_pred.size):
    if i==1:
        axs[1].plot(a, y_no_test[i], color='blue', marker='x', label='Test␣
↪values')
        axs[1].plot(a, y_no_pred[i], color='red', marker='x', label='Predicted␣
↪values')
        errors[i] = y_no_test[i] - y_no_pred[i]
        axs[1].legend()
    else:
        axs[1].plot(a, y_no_test[i], color='blue', marker='x')
        axs[1].plot(a, y_no_pred[i], color='red', marker='x')
        errors[i] = y_no_test[i] - y_no_pred[i]
    x_values = [a, a]
    y_values = [y_no_test[i], y_no_pred[i]]
    axs[1].plot(x_values, y_values, color='green')

    a=a+1

axs[1].set_title("Without outliers")
```

```
Elastic Net
```

[69]:
```
Text(0.5, 1.0, 'Without outliers')
```



1.1 Regularization methods

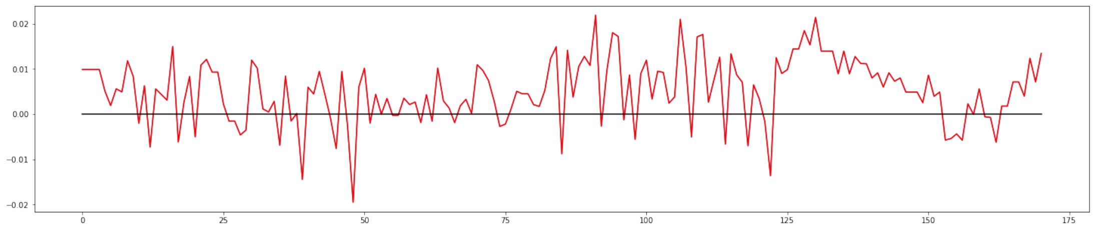
[70]:
```
# Splitting the data into training and test samples
number_items = int(round(75*228/100,0))
#number_items=150
#print(number_items)

X = students_no_outliers_df.loc[:,(items_cat[0:number_items])]
y = grades_no_outliers_df.loc[:, 'GRADE']

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=1)
```

[71]:
```
# Results for linear regression without regularization
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_train, y_train)
y_pred = slr.predict(X_test)
y_pred = y_pred.round(2)

slr.fit(X_no_train, y_no_train)
y_no_pred = slr.predict(X_no_test)
y_no_pred = y_no_pred.round(2)

y_slr = y_no_pred
```

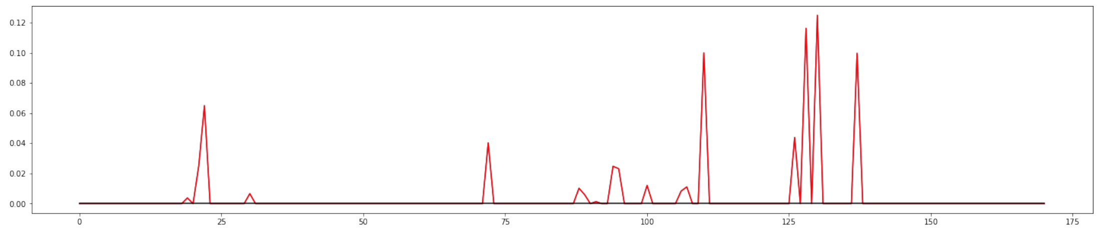
[72]:
```
# Ridge
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=10.0)
from sklearn.metrics import explained_variance_score

ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
y_pred = y_pred.round(2)
coefs = ridge.coef_

ridge.fit(X_no_train, y_no_train)
y_no_pred = ridge.predict(X_no_test)
y_no_pred = y_no_pred.round(2)
coefs_no = ridge.coef_

y_ridge=y_no_pred

regularization = "Ridge"

plt.rcParams['figure.figsize'] = [25, 5]
plt.plot(range(number_items), coefs)
plt.plot(range(number_items), coefs_no, color='red')
plt.plot(range(number_items), np.zeros(number_items), color='black')
```

[72]:
```
[<matplotlib.lines.Line2D at 0x218b85609d0>]
```



```
[73]: # LASSO
      from sklearn import linear_model
      reg = linear_model.Lasso(alpha=0.01)

      reg.fit(X_train, y_train)
      y_pred = reg.predict(X_test)
      y_pred = y_pred.round(2)
      coefs = reg.coef_

      reg.fit(X_no_train, y_no_train)
      y_no_pred = reg.predict(X_no_test)
      y_no_pred = y_no_pred.round(2)
      coefs_no = reg.coef_

      y_lasso = y_no_pred

      regularization = "LASSO"

      plt.rcParams['figure.figsize'] = [25, 5]
      plt.plot(range(number_items), coefs)
      plt.plot(range(number_items), coefs_no, color='red')
      plt.plot(range(number_items), np.zeros(number_items), color='black')
```

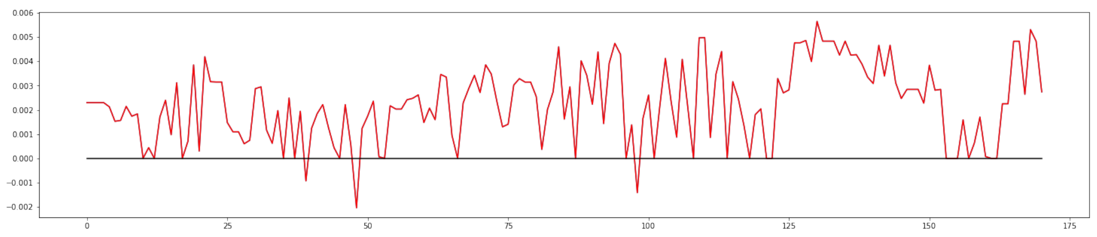
```
[73]: [<matplotlib.lines.Line2D at 0x218b85ba550>]
```



```
[74]: # Elastic net
      from sklearn.linear_model import ElasticNet
```

```
elanet = ElasticNet(alpha=5.0, l1_ratio=0.001)

elanet.fit(X_train, y_train)
y_pred = elanet.predict(X_test)
y_pred = y_pred.round(2)
coefs = elanet.coef_

elanet.fit(X_no_train, y_no_train)
y_no_pred = elanet.predict(X_no_test)
y_no_pred = y_no_pred.round(2)
coefs_no = elanet.coef_

y_en = y_no_pred

regularization = "Elastic Net"

plt.rcParams['figure.figsize'] = [25, 5]
plt.plot(range(number_items), coefs)
plt.plot(range(number_items), coefs_no, color='red')
plt.plot(range(number_items), np.zeros(number_items), color='black')
```

[74]:
```
[<matplotlib.lines.Line2D at 0x218b66b00d0>]
```



[75]:
```
# Accuracy metrics
data = np.zeros([6,4])

for x in range(4):
    if x==0:
        y_predicted=y_slr
    elif x == 1:
        y_predicted = y_ridge
    elif x == 2:
        y_predicted = y_lasso
    else:
        y_predicted = y_en

    exp_var = round(explained_variance_score(y_no_test,y_predicted),2)
    data[0,x] = exp_var

    max_err = round(max_error(y_no_test, y_predicted), 2)
    data[1,x] = max_err

    mean_err = round(mean_absolute_error(y_no_test, y_predicted), 2)
    data[2,x] = mean_err

    mean_sq_err = round(mean_squared_error(y_no_test, y_predicted), 2)
    data[3,x] = mean_sq_err

    med_err = round(median_absolute_error(y_no_test, y_predicted), 2)
    data[4,x] = med_err

    r2_sc = round(r2_score(y_no_test, y_predicted), 2)
    data[5,x] = r2_sc

index = ['Explained variance', 'Max error (grade pt)', 'Mean absolute error␣
 ↪(grade pt)', 'Mean squared error (grade pt^2)', 'Median absolute error␣
 ↪(grade pt)', 'R2 score']
accuracy_metrics = pd.DataFrame(data, columns=['SLR', 'Ridge', 'LASSO',␣
 ↪'Elastic Net'], index=index)
display(accuracy_metrics)
```

```
                                  SLR  Ridge  LASSO  Elastic Net
Explained variance               0.72   0.69   0.63         0.41
Max error (grade pt)             0.41   0.36   0.36         0.58
Mean absolute error (grade pt)   0.11   0.12   0.14         0.16
Mean squared error (grade pt^2)  0.02   0.03   0.03         0.05
Median absolute error (grade pt) 0.05   0.07   0.12         0.12
R2 score                         0.66   0.65   0.60         0.38
```

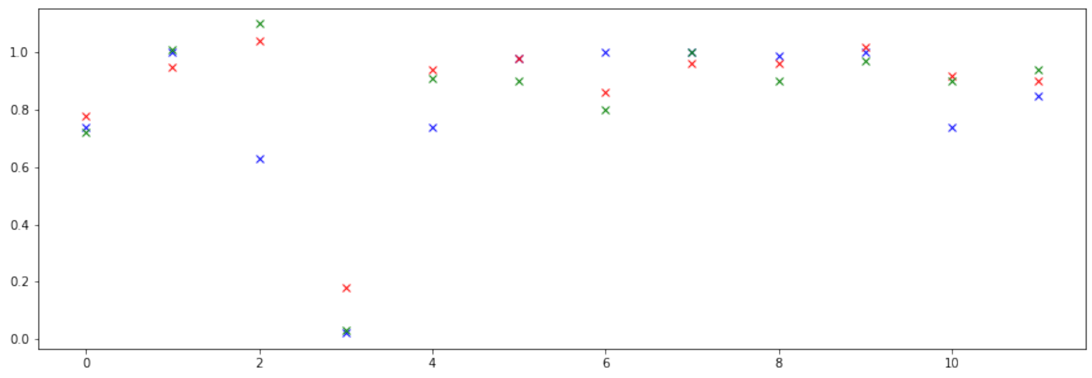
1.2 Polynomial regression

[76]:
```
# Splitting the data into training and test samples
number_items = int(round(75*228/100,0))
#number_items=150
#print(number_items)

X = students_no_outliers_df.loc[:,(items_cat[0:number_items])]
y = grades_no_outliers_df.loc[:, 'GRADE']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1)
```

[77]:
```
# Results for linear regression without regularization
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_train, y_train)
y_pred = slr.predict(X_test)
y_pred = y_pred.round(2)

y_test = y_test.reset_index(drop=True)
y_test.tolist()
```

[77]:
```
[0.74, 1.0, 0.63, 0.02, 0.74, 0.98, 1.0, 1.0, 0.99, 1.0, 0.74, 0.85]
```

[78]:
```
# Results for polynomial regression without outliers
from sklearn.preprocessing import PolynomialFeatures
quadratic = PolynomialFeatures(degree=2)
X_quad = quadratic.fit_transform(X_no)

X_train_quad, X_test_quad, y_train, y_test_quad = train_test_split(X_quad,␣
 ↪y_no, test_size=0.3, random_state=1)
pr = LinearRegression()

pr.fit(X_train_quad, y_train)
y_pred_quad_no = pr.predict(X_test_quad)
y_pred_quad_no = y_pred_quad_no.round(2)

# Plot both and compare

plt.rcParams['figure.figsize'] = [15, 5]

y_test_quad = y_test_quad.reset_index(drop=True)
y_test_quad.tolist()
a=0
for i in range(y_pred_quad_no.size):
    plt.plot(a, y_test[i], color='blue', marker='x')
    plt.plot(a, y_pred[i], color='red', marker='x')
    plt.plot(a, y_pred_quad_no[i], color='green', marker='x')


    a=a+1
```



[79]:
```
# Accuracy metrics
data = np.zeros([6,2])

exp_var = round(explained_variance_score(y_test,y_pred),2)
exp_var_q = round(explained_variance_score(y_test_quad,y_pred_quad_no),2)
data[0,:] = [exp_var, exp_var_no]

max_err = round(max_error(y_test, y_pred), 2)
max_err_no = round(max_error(y_test_quad, y_pred_quad_no), 2)
data[1,:] = [max_err, max_err_no]

mean_err = round(mean_absolute_error(y_test, y_pred), 2)
mean_err_no = round(mean_absolute_error(y_test_quad, y_pred_quad_no), 2)
data[2,:] = [mean_err, mean_err_no]

mean_sq_err = round(mean_squared_error(y_test, y_pred), 2)
mean_sq_err_no = round(mean_squared_error(y_test_quad, y_pred_quad_no), 2)
data[3,:] = [mean_sq_err, mean_sq_err_no]

med_err = round(median_absolute_error(y_test, y_pred), 2)
med_err_no = round(median_absolute_error(y_test_quad, y_pred_quad_no), 2)
data[4,:] = [med_err, med_err_no]

r2_sc = round(r2_score(y_test, y_pred), 2)
r2_sc_no = round(r2_score(y_test_quad, y_pred_quad_no), 2)
data[5,:] = [r2_sc, r2_sc_no]

index = ['Explained variance', 'Max error (grade pt)', 'Mean absolute error␣
 ↪(grade pt)', 'Mean squared error (grade pt^2)', 'Median absolute error␣
 ↪(grade pt)', 'R2 score']
accuracy_metrics = pd.DataFrame(data, columns=['SLR', 'Grade 2'], index=index)
display(accuracy_metrics)
```

```
                                   SLR  Grade 2
```

```
Explained variance                 0.72     0.72
Max error (grade pt)               0.41     0.47
Mean absolute error (grade pt)     0.11     0.11
Mean squared error (grade pt^2)    0.02     0.03
Median absolute error (grade pt)   0.05     0.08
R2 score                           0.66     0.61
```

1.3 Splitting percentages

[80]:
```
# Splitting the data into training and test samples 75%
number_items = int(round(75*228/100,0))
#number_items=150
#print(number_items)

X_no_75 = students_no_outliers_df.loc[:,(items_cat[0:number_items])]
y_no_75 = grades_no_outliers_df.loc[:, 'GRADE']

from sklearn.model_selection import train_test_split
X_no_train_75, X_no_test_75, y_no_train_75, y_no_test_75 =␣
↪train_test_split(X_no_75, y_no_75, test_size=0.3, random_state=1)

train_no_students_index = X_no_train_75.index
test_no_students_index = X_no_test_75.index
```

[81]:
```
# Splitting the data into training and test samples 50%
number_items = int(round(50*228/100,0))
#number_items=150
#print(number_items)

X_no_50 = students_no_outliers_df.loc[:,(items_cat[0:number_items])]
y_no_50 = grades_no_outliers_df.loc[:, 'GRADE']

from sklearn.model_selection import train_test_split
X_no_train_50, X_no_test_50, y_no_train_50, y_no_test_50 =␣
↪train_test_split(X_no_50, y_no_50, test_size=0.3, random_state=1)

train_no_students_index = X_no_train_50.index
test_no_students_index = X_no_test_50.index
```

[82]:
```
# Splitting the data into training and test samples 25%
number_items = int(round(25*228/100,0))
#number_items=150
#print(number_items)

X_no_25 = students_no_outliers_df.loc[:,(items_cat[0:number_items])]
y_no_25 = grades_no_outliers_df.loc[:, 'GRADE']
```

```
from sklearn.model_selection import train_test_split
X_no_train_25, X_no_test_25, y_no_train_25, y_no_test_25 =␣
↪train_test_split(X_no_25, y_no_25, test_size=0.3, random_state=1)

train_no_students_index = X_no_train_25.index
test_no_students_index = X_no_test_25.index
```

```
[83]: # Results for linear regression without regularization 75%
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_no_train_75, y_no_train_75)
y_no_pred_75 = slr.predict(X_no_test_75)
y_no_pred_75 = y_no_pred_75.round(2)
coefs_no_75 = slr.coef_

y_slr_75 = y_no_pred_75
```

```
[84]: # Results for linear regression without regularization 50%
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_no_train_50, y_no_train_50)
y_no_pred_50 = slr.predict(X_no_test_50)
y_no_pred_50 = y_no_pred_50.round(2)
coefs_no_50 = slr.coef_

y_slr_50 = y_no_pred_50
```

```
[85]: # Results for linear regression without regularization 25%
from sklearn.linear_model import LinearRegression
slr = LinearRegression()

slr.fit(X_no_train_25, y_no_train_25)
y_no_pred_25 = slr.predict(X_no_test_25)
y_no_pred_25 = y_no_pred_25.round(2)
coefs_no_25 = slr.coef_

y_slr_25 = y_no_pred_25
regularization = "No regularization"
```

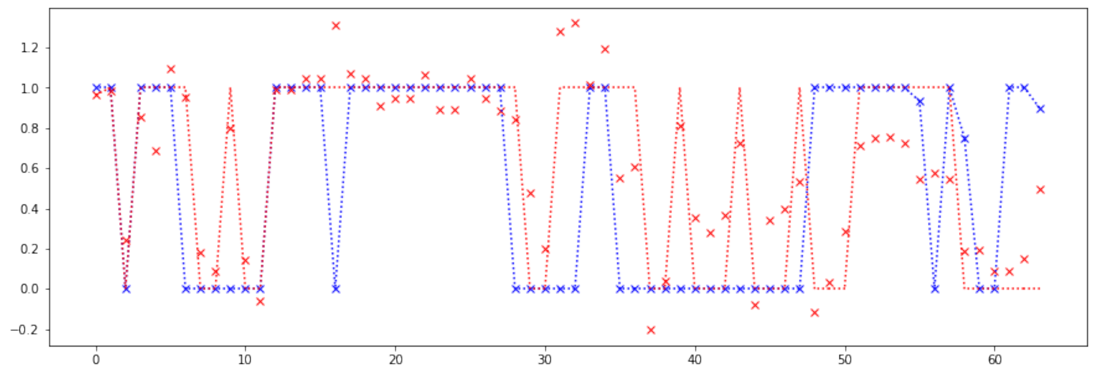
```
[86]: # Accuracy metrics for different %
data = np.zeros([6,3])

exp_var_25 = round(explained_variance_score(y_no_test_25,y_no_pred_25),2)
exp_var_50 = round(explained_variance_score(y_no_test_50,y_no_pred_50),2)
```

```
exp_var_75 = round(explained_variance_score(y_no_test_75,y_no_pred_75),2)
data[0,:] = [exp_var_25, exp_var_50, exp_var_75]

max_err_25 = round(max_error(y_no_test_25, y_no_pred_25), 2)
max_err_50 = round(max_error(y_no_test_50,y_no_pred_50), 2)
max_err_75 = round(max_error(y_no_test_75,y_no_pred_75), 2)
data[1,:] = [max_err_25, max_err_50, max_err_75]

mean_err_25 = round(mean_absolute_error(y_no_test_25, y_no_pred_25), 2)
mean_err_50 = round(mean_absolute_error(y_no_test_50,y_no_pred_50), 2)
mean_err_75 = round(mean_absolute_error(y_no_test_75,y_no_pred_75), 2)
data[2,:] = [mean_err_25, mean_err_50, mean_err_75]

mean_sq_err_25 = round(mean_squared_error(y_no_test_25, y_no_pred_25), 2)
mean_sq_err_50 = round(mean_squared_error(y_no_test_50,y_no_pred_50), 2)
mean_sq_err_75 = round(mean_squared_error(y_no_test_75, y_no_pred_75), 2)
data[3,:] = [mean_sq_err_25, mean_sq_err_50, mean_sq_err_75]

med_err_25 = round(median_absolute_error(y_no_test_25, y_no_pred_25), 2)
med_err_50 = round(median_absolute_error(y_no_test_50,y_no_pred_50), 2)
med_err_75 = round(median_absolute_error(y_no_test_75, y_no_pred_75), 2)
data[4,:] = [med_err_25, med_err_50, med_err_75]

r2_sc_25 = round(r2_score(y_no_test_25, y_no_pred_25), 2)
r2_sc_50 = round(r2_score(y_no_test_50,y_no_pred_50), 2)
r2_sc_75 = round(r2_score(y_no_test_75, y_no_pred_75), 2)
data[5,:] = [r2_sc_25, r2_sc_50, r2_sc_75]

print('Regularization method: ', regularization)
index = ['Explained variance', 'Max error (grade pt)', 'Mean absolute error␣
 ↪(grade pt)', 'Mean squared error (grade pt^2)', 'Median absolute error␣
 ↪(grade pt)', 'R2 score']
accuracy_metrics = pd.DataFrame(data, columns=['25%', '50%', '75%'],␣
 ↪index=index)
display(accuracy_metrics)
```

```
Regularization method:  No regularization

                                  25%   50%   75%
Explained variance               0.47  0.72  0.72
Max error (grade pt)             0.30  0.40  0.41
Mean absolute error (grade pt)   0.18  0.11  0.11
Mean squared error (grade pt^2)  0.04  0.03  0.02
Median absolute error (grade pt) 0.20  0.06  0.05
R2 score                         0.45  0.65  0.66
```

1.4 Predicting individual items

[88]:
```python
plt.rcParams['figure.figsize'] = [15, 5]
done = 150;
X = students_df.loc[:, (items_cat[0:done])]
a = 0
for j in range(done+1, len(items_cat)):
    y = students_df.loc[:, (items_cat[j])]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)


    slr = LinearRegression()
    slr.fit(X_train, y_train)
    y_pred = slr.predict(X_test)

    y_test = y_test.reset_index(drop=True)
    y_test.tolist()

    #plot student from test set
    student = 2

    plt.plot(a, y_test[student], color='blue', marker='x')
    plt.plot(a, y_pred[student], color='red', marker='x')

    if a==0:
        temp_test = y_test[student]
        temp_pred = y_pred[student]


    if a>0:
        if y_pred[student] > 0.5:
            y_pred[student] = 1.0
        else:
            y_pred[student] = 0.0

        x_values = [a, a-1]
        y_values_test = [y_test[student], temp_test]
        y_values_pred = [y_pred[student], temp_pred]
        plt.plot(x_values, y_values_test, color='blue', linestyle=':')
        plt.plot(x_values, y_values_pred, color='red', linestyle=':')
        #plt.plot([a,a], [y_test[0], y_pred[0]], color='green')

        temp_test = y_test[student]
        temp_pred = y_pred[student]

    a=a+1
```


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50
60