Universitatea POLITEHNICA din București
Facultatea de Automatică și Calculatoare

Predicţia performanţei studenţilor unui curs folosind modele de Învăţare Automată

Autor:
Marin Cristian-Emil

Coordonator științific:
Şl.dr.ing. Popovici Dan-Matei

Septembrie 2016

University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers

Predicting Student Course Performance with Machine Learning Models

Author:
Marin Cristian-Emil

Scientific Coordinator:
Şl.dr.ing. Popovici Dan-Matei

Bucharest
September 2016

Contents

Abstract

Scriem ceva abstract aici, maxim 150 cuvinte.

Keywords: grade prediction; machine learning; data analysis; neural networks; random forests; data classification; model entropy

Chapter 1

Introduction

1.1 Motivation

Lucrarea este structurată astfel: un fisier de configurare poate fi găsit în anexa A.1.

Mai jos urmează niște text de umplutură și un algoritm realizat cu algortihm2e.

Duis consectetur tempor volutpat. Nullam molestie, nibh fringilla egestas euismod, mi eros euismod augue, eu rutrum mi nibh vitae sapien. Sed ultricies vehicula est a tincidunt. Aenean nec ligula purus, quis porttitor velit. Donec ultrices, velit sed facilisis condimentum, ligula ligula facilisis mi, ac accumsan arcu tellus eu tortor. In congue mauris ac est aliquet a ullamcorper dui fermentum. Donec venenatis nulla ut lectus cursus elementum. Phasellus vulputate aliquet dolor convallis placerat. Nam pharetra, nibh ut mollis venenatis, purus nisl adipiscing diam, quis dignissim mauris risus non velit. Aliquam et mauris non tortor malesuada luctus. Integer quis justo turpis.

Textul următor este citat din.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce auctor ultrices sollicitudin. Ut sed lobortis nunc. Proin adipiscing erat vel dolor pharetra sed semper felis malesuada. Quisque imperdiet augue lectus. Nulla facilisi. Fusce tortor quam, pharetra et blandit et, pulvinar convallis risus. Nam id molestie dolor. Phasellus aliquam laoreet dolor, vel semper neque blandit sit amet. Quisque

```
1  function eval():
2  begin
3      if bubu = null then
4          return null
5      end if
6      bubu ← bibi
7      foreach foo in bar do
8          if foo is zaz then
9              bubu ← bubu['lili.lala']
10         else if term is Call then
11             bibi ← call(cucu)
12         end if
13     end foreach
14     return crtContext
15 end
```

Figure 1.1: Algoritmul de evaluare al crocobazilor.

quis tellus ipsum. Fusce fringilla, neque vitae tincidunt placerat, turpis metus dignissim justo, quis blandit justo est nec tellus. Nunc pretium, felis sed mattis euismod, metus leo tempus risus, a sodales augue lorem id libero.

1.1.1 Lorem ipsum

Duis consectetur tempor volutpat. Nullam molestie, nibh fringilla egestas euismod, mi eros euismod augue, eu rutrum mi nibh vitae sapien. Sed ultricies vehicula est a tincidunt. Aenean nec ligula purus, quis porttitor velit. Donec ultrices, velit sed facilisis condimentum, ligula ligula facilisis mi, ac accumsan arcu tellus eu tortor. In congue mauris ac est aliquet a ullamcorper dui fermentum. Donec venenatis nulla ut lectus cursus elementum. Phasellus vulputate aliquet dolor convallis placerat. Nam pharetra, nibh ut mollis venenatis, purus nisl adipiscing diam, quis dignissim mauris risus non velit. Aliquam et mauris non tortor malesuada luctus. Integer quis justo turpis.

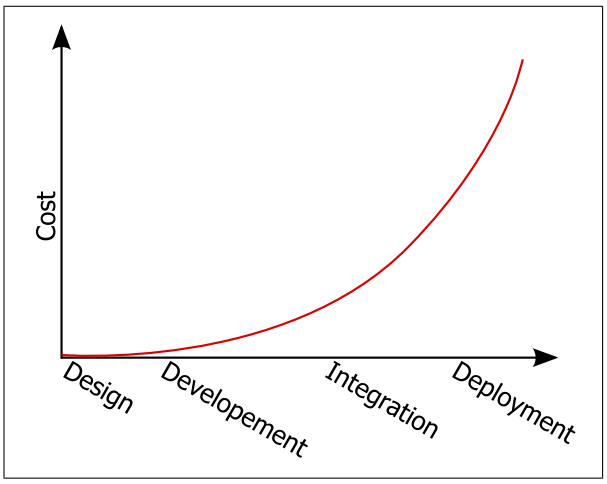

Figure 1.2: Variația costului de remediere a dudelor cu momentul descoperirii lor

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce auctor ultrices sollicitudin. Ut sed lobortis nunc. Proin adipiscing erat vel dolor pharetra sed semper felis malesuada. Quisque imperdiet augue lectus. Nulla facilisi. Fusce tortor quam, pharetra et blandit et, pulvinar convallis risus. Nam id molestie dolor. Phasellus aliquam laoreet dolor, vel semper neque blandit sit amet. Quisque quis tellus ipsum. Fusce fringilla, neque vitae tincidunt placerat, turpis metus dignissim justo, quis blandit justo est nec tellus. Nunc pretium, felis sed mattis euismod, metus leo tempus risus, a sodales augue lorem id libero.

Chapter 2

Background

2.1 State of the Art

2.1.1 A very short history of Machine Learning

"You have to know the past to understand the present." - Carl Sagan

The early beginnings of Machine Learning come not after the first electronic computers or after the first computer programs, as many may believe so. The fundamentals of this field took shape centuries ago with the discovery of the so-called Conditional Probability theories. **Bayes Theorem**, named after Thomas Bayes who first proposed a mathematical model for infering probabilities of conditioned events, and further developed by Pierre-Simon Laplace in his essay [1] in 1812, can be seen as one of the first models that can "learn" from given data and predict events based on correlated past events. Another old discovery that is the basis of today's regression models (e.g.: Linear Regression) is the "least squares method", credited to Carl Gauss, but first published by Adrien-Marie Legendre in 1805. This method was first applied in astronomy and allowed explorers to navigate oceans by aproximating the movement of celestial bodies.

Later on, in 1950, Alan Turing proposed in his paper[2] a *learning machine* that is able to learn and become intelligent and do well in the **Imitation Game** (now

[1] Théorie Analytique des Probabilités

[2] Turing, A.M. (1950). Computing machinery and intelligence. Mind, 59, 433-460.

generally called the **Turing Test**).

After this, the discovery of the Percetron and the **Neural Networks** around 1960s drew some attention in the field, but their current limitations had put Machine Learning on an impeding state for almost 10 years. It was only with the invention of the backpropagation algorithm in 1974 by Paul Werbos and the demonstration of its generalization by Geoffrey Hinton in 1986, that allowed it to be applied in multi-layered artificial neural networks. This also gave birth to a new sub-field of Machine Learning that today is called **Deep Learning.**

Along with the research in neural networks, some other models that were developed in that period are worth to mention. The most important ones are Support Vector Machines and kernels (models used for data classification and regression, that can be more time-efficient than neural networks[3] are and provide good performance from a data perspective) and Decision Trees. The latter, in combination with Ensemble Methods helped researchers invent models like **Random Forests** and **Adaptive Boosting** that are now state-of-the-art algorithms for tree models used for a lot of tasks.

2.1.2 Current interests in the field

Coming back to Deep Learning, which is today's main subject of interest of the Machine Learning community, it is a general approach that combines several state-of-the-art models of Machine Learning to solve problems such as image classification, AI for computer games, natural language, etc. Its constituents include neural networks with many hidden layers, convolutional networks, deep belief networks and recurrent networks. Also, the Q-Learning algorithm[4] and the Monte-Carlo search used in combination with convolutional networks allowed researchers to build semi-supervised learning programs that could learn to play computer games by themselves[5] or beat professional human players at games, such as Go (Google AlphaGo's program first beat Lee Sedol in October 2015).

[3]Some say that SVMs actually subsume Neural Networks, because of the flexibility of kernel functions

[4]Watkins, C.J.C.H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England

[5]https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf

2.1.3 Trends in educational learning

All advances in the Machine Learning field also conducted in an incresing interest in educational learning and assessment. With the name of **Educational Data Mining**, this newly emerging discipline deals with studying machine learning and data mining models in order to gain important knowledge about the structure of an educational system data (final grades, performance indicators, course drop-outs). Educational data can be taken from schools and universities (the classical way and also, the way that this thesis explores), online courses (such as MOOCs[6]), or even collaborative learning.

With the use of Machine Learning techniques, students can better decide on what courses they could take (based on past related grades), whether they have a chance or not to pass an exam before taking it and what indicators are relevant to their final asessment. The course department also benefits from the "learned" data because it helps them to better plan the structure of their courses, whether they are on-line or taken at the university.

2.2 Related Work

This part of the chapter will focus on the work on other people about the study on student performance prediction and analysis. Some of their research is similar to that of this thesis, and some others treat only related issues.

(Mehdi Sajjadi et al., 2016)[1] did some work on approximating final grades of students in a course on algorithms. In the grading process they used the peer grading method[7], and then applied Machine Learning (both supervised and usupervised) to aggregate those grades into a final grade. Their results were not so good compared to the simple method of just using the mean of all peer grades per an assessment as the final grade.

(Siddharth Reddy et al., 2015)[2] worked on developing a representational model of combined students and educational content (assessments and lessons). This

[6]Massive Open Online Courses

[7]A process in which students grade work of other students based on a given guideline

representation is actually a semantic space[8] and it can be used to study the relation between course content and students. Several conclusions can be drawn from these representations, such as: probability of passing an assessment or course and knowledge gained from completing a lesson. This article aimed mostly at MOOCs platforms, like Coursera, EdX and Khan Academy, and the model described was tested on synthetic student data and also, on real data from Knewton. Their model's results can be used to personalize the learning process of a course for each student in order to maximize the educational performance. Also, this model successfully predicted assessment results.

(Michael Wu, 2015)[3] wrote an interesting Master Thesis in which describes a Machine Learning Model that simulates MOOC data. Working with data gathered from EdX, the model once trained, can be able to synthesize student data. The model was trained to learn about student types, habits and difficulty of course materials. One of the main results of the thesis was being able to classify students in 20 important clusters.

(Saeed Hosseini Teshnizi and Sayyed Mohhamad Taghi Ayatollahi, 2015)[4] did a comparasion between Logistic Regression and ANNs[9] on a dataset composed of 275 undergraduate students and 16 student characteristics (e.g.: age, gender, parent education, employment status, place of residence, etc.) in order to predict academic failure. They concluded that the neural network models had a better accuracy than Logistic Regression (84.3% versus 77.5%). They tested 9 ANNs from which the one with 15 neurons in the hidden layer provided the best results, so ANNs methods were appropiate to be used in their problem.

Other references[5],[6],[7] also treat prediction of student academic performance mostly with neural networks and provide good accuracy with this model (the accuracy is also dependent of the structure of the dataset, number of examples, number of characteristics and noise in the data).

(Emaan Abdul Majeed and Khurum Nazir Junejo)[8] had some great results using Machine Learning models for predicting student's performance. They claim that they were capable of predicting the final grade with an accuracy of 96%. With a final number of 2500 student records and about 10 attributes for each record,

[8]Semantic similarity between objects represented as a kind of "metric" in space

[9]Artificial Neural Networks

they managed to predict the value of the final Grade attribute (which is a Class Variable that can have 6 values: A, B+, B, C+, C, Fail). Four classifier models were used in their study and we can see in Figure 2.1[8] their performance:

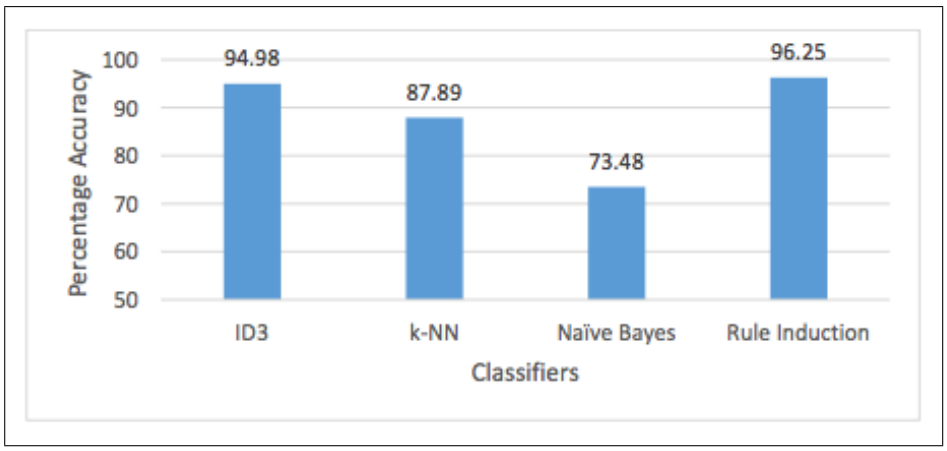

Figure 2.1: Classifiers accuracy

Some other last interesting references for this thesis[9],[10] used students online activity (course logs) to find patterns related to their overall performance. When applied to a MOOCs platform[10], the researchers found that the learners performace is strongly related to the number of videos played in the course platform, posts in course forum and, also, the total number of active days on that platform. With the use of SVMs, they achieved a 95% accuracy[10].

[10]They divided the learners in two classes: earning or not a certificate for the course

Chapter 3

Model Design and Methods Used

3.1 Getting and Pre-processing the Data

3.1.1 Choosing the dataset

First, when this project was in development, we only had access to a single students dataset, i.e. students performance grades from the Analysis of Algorithms course, which took place in the Fall 2015 semester. As progress was being made with the implementation, it became clear that the dataset could not be practically used, due to its high amount of noisy data. The noise was present in the data because of the small size of the dataset (137 students were taken into consideration) and also, the non-trivial distribution of final grades over the semester activity of students. The grading subjectivity of teaching assistants combined with a new experimental grading method for the course were also important factors in this negative result. Hence, with this much noise in the data, the models have not been able to correctly label the examples and, with more complicated models, there was a high tendency for *overfitting* the data.

Later, as the second semester was ending, we had access to another dataset of students and decided to switch to it. We also kept the students final grades for the first course to use them as attributes for the new dataset, in order to increase accuracy. In the 5^{th} chapter of this thesis, a detailed analysis will be made between these two datasets to show how they differ and what is wrong with the first one.

An important note must be made here. To increse the number of examples for our dataset, we could have combined those two above or could have aggregated students from different years into a single dataset but, there are two main reasons we did not do this: one, because the grading method and the number of semester assessments were different from one year to another and, second, because the two courses are conceptually different: one is theoretical, the other is more practical. Also, combining those two sets of examples would just have brought more noise to the second dataset.

Given this situation, the data that is currently used in this thesis comes from our Computer Science Undergraduate course, i.e. Programming Paradigms from 2^{nd} year during the Spring 2016 semester. For that semester, we had a total record of almost 200 registered students, but we removed those whose attributes were missing or were not relevant (the ones that had a very low performance). Since we are interested in classifying students based on their exam grades and final grades, also implying the classification in *passed* and *failed* classes, we kept some students that had a poor grade during the semester and could not participate at the final exam, although they were close to that point. Having said this, we are left with a total of 143 relevant student records. These records must further be split in two sets: the training set - the set that is used for building the model and the testing set - the set used for testing the accuracy and measure the performance indicators of the model.

There is not a standard recipe of how to divide these two sets, but it is recommended that the training set should have a proportion between 60% and 80% of the total dataset's number of examples. For empirical reasons, the ratio of 80-to-20 percent was chosen for our dataset and the examples were sorted alphabetically based on the student's surname (the order is independent from the features and labels). The current dataset is not uniformly distributed over the *failed/passed* classes. There are 40% students that passed the course and 60% that failed it. This proportion is helpful in studying the prediction models (they must predict the failed students with a slightly higher accuracy). In more detail, from those 143 analyzed students, 57 of them had a positive grade (>=5) and 86 a negative one (<=4). From the 86 students that did not pass the course, 55 of them failed the final exam and 31 of them failed the course during the semester (and did not take part in the final examination process). The reason we kept those 31 students in

the datased was to give the ML model more data examples, thus increasing the accuracy when predicting *failed* students.

3.1.2 Dataset Structure

To build the dataset, several sources of information were used. The first trivial one is the course semester activity of the students. The second, comes from performance of students achieved at past courses relevant to ours (chosen in different semesters in order to make them as independent from each other as possible), from a topic perspective. The courses that we got the grades from are:

- Computer Programming (PC): first year, first semester
- Data Structures (SD): first year, second semester
- Analysis of Algorithms (AA): second year, first semester

Lastly, the third subset of entire dataset was taken from our on-line e-learning course platform[1]. There, we had access to all the students activity logs recorded during the course progress. The logs were downloaded as *.csv* file format and then, aggregated in four categories for meaningful results.

In Table 3.1 there is a listing with all the attributes (features) belonging to the first subset of features - the course semester activity. There are a total of 10 initial features (will address later on this problem with the number of features). The weight characteristic of the features represents the actual maximum points that a student can get from that assignment (they are not equal, but are summing for a total of 6.0 semester points out of 10), and was not used for building the ML models.

Table 3.2 describes the third subset of our dataset's features: the Moodle logs. Activities including homework forum posts and views (x_1), active days (x_2), course resources opened (x_3) and total number of logs per user (x_4) were used along with the rest of features (together and separately) in the learning models.

Next, we need to provide the values used for labelling the examples. Since this thesis treats both classification and regression problems, different types of labels

[1]Moodle, the Open-source learning platform

Feature	Description	Type	Range	Weight
t_1	Test 1 grade	Float	0-10	0.25
t_2	Test 2 grade	Float	0-10	0.25
t_3	Test 3 grade	Float	0-10	0.25
t_4	Test 4 grade	Float	0-10	0.25
t_f	Final test grade	Float	0-10	1.0
hw_1	Homework 1 grade	Float	0-1	1.0
hw_2	Homework 2 grade	Float	0-1	1.0
hw_3	Homework 3 grade	Float	0-1	1.0
lab	Lab activity	Float	0-0.5	0.5
lecture	Lecture activity	Float	0-0.5	0.5

Table 3.1: Dataset Semester Attribute Details

must be present in the dataset. For classification, we use the final grade (real number between 0 and 10) and the exam grade as integer values. For regression, only the exam/final grade is used and it will be represented as a float number.

Table 3.3 gives a structured view of how the students dataset labels are used in the studied models. For classifying the students into 3 classes based on their exam points or final grade, the real continuous interval $(0, 10]$ was split into 3 different-length subintervals: $(0, 5)$, $[5, 7]$ and $(7, 10]$. For each subinterval there was assigned a class label: 0, 1 and 2, respectively.

Feature	Description	Type
x_1	f(forum_posts, forum_views)	Float
x_2	Number of active days	Float
x_3	Number of course resource views	Float
x_4	Total number of logs per user	Float

Table 3.2: Dataset Logs Attribute Details

The f function from the 3.2 table is actually a linear combination of the number of forum posts and forum views, as follows:

$$f(\#posted, \#viewed) = \#posted + \frac{\#viewed}{\alpha} \tag{3.1}$$

where α is the factor representing the *viewed* to *posted* ratio of all the homework forum logs (40 in our case). This was chosen because we are more interested in forum posts (as they are more relevant), so the forum views are adjusted with this factor to have a smaller value.

Problem type	Label Description		
	Label type	Label Value	Label used
Binary Classification	Integer	0|1	Exam/Final grade
3-class Classification	Integer	0|1|2	Exam/Final grade
Regression	Float	0.0-10.0	Exam/Final grade

Table 3.3: Label Structure

3.1.3 Data Standardization

In Machine Learning, standardization of data is a common requirement for a lot of models (e.g.: neural networks and SVMs). This means that, before applying our learning models, we must first scale the features from our dataset. This scaling implies that the data should have **mean** 0 and **standard deviation** 1. This is done by subtracting the mean value of each feature, then scale the data by dividing the features by their standard deviation (or variance, because standard deviation is the square root of variance, so they are equal).

$$X_{scaled} = \frac{X - \overline{X}}{\sigma} \tag{3.2}$$

where X is a vector of values from one feature, $\overline{X}$ is the mean of X and σ represents the standard deviation of X.

In Figure 3.1[2] we can see a straight-forward visualization of how the data is represented before and after the preprocessing techniques:

Data standardization is important if we are comparing features that have different ranges, but it is also necessary for some of the machine learning models. That

[2]http://cs231n.github.io/neural-networks-2

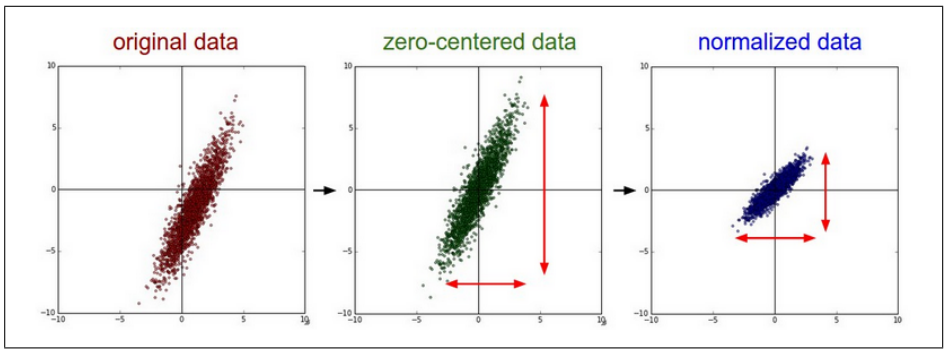

Figure 3.1: Data preprocessing pipeline

is because these models use the *gradient descent*[3] algorithm, which is sensitive to feature scaling.

It is worth noting that the mean and standard deviation values must come only from the training data and the transformation must be done on both training and testing sets, for meaningful results. The reason is that, in general, the built Machine Learning model is applied on unseen data (on real-time data), which is not available when the model is built. So, for accurate calculations on the model's performance and generalization, we must restrict the computation of mean and variance only on the training examples.

3.2 Feature generalization

3.2.1 The shape of a general dataset

Because one of the purposes of this work is to apply the learning model on future datasets (from the same course or even from other courses), we must first generalize the features of the model. To do this, the dimension of the feature space must have a constant value along multiple datasets and each feature must have the same *meaning*. Therefore, we are going to divide our current feature space discussed in subsection 3.1.2 into several categories, based on their similarity. Besides the above stated purpose, an advantage of this feature modelling is to decrease the dimension of the feature space. Having a small, but meaning-

[3] an optimization algorithm that is used to find a local minimum of a cost function

ful dimension of the input, the learning models can sometimes perform better, mainly when the number of examples in the dataset is small. The reasons for this statement are described in the following subsection.

Since almost every course has a semester grading method based on homeworks, tests, lecture and lab activity, the split visible in table 3.4 comes natural. With this, the dataset now has a constant number of features: five. If the Moodle logs features (which also have a constant dimension regardless of the dataset used) are further added to this new feature space, there will be a total of nine features. This way, we can use every student dataset, transform it to have this structure and then, apply the model.

A disadvantage of this generalization is that we lose information about features that get aggregated together in a new one. Sometimes this is useful, for example when we want to see what homework or what test was the most relevant in the student's exam or final grade.

Feature	Description	Type	Range
hw_agg	Homework points	Float	0-1
t_agg	Test points	Float	0-10
past_agg	Past results score	Float	0-10
lab	Lab activity	Float	0-0.5
lecture	Lecture activity	Float	0-0.5

Table 3.4: Dataset Semester Aggregated Features

Note: the homework, test, and past results aggregated features are calculated as a weighted arithmetic mean between their components.

3.2.2 Curse of Dimensionality

In general, the number of examples and the dimensionality of each example (the number of features per example) are correlated, taking into consideration the accuracy of the trained model. The *Hughes phenomenon*[11] tells us that if we have a constant number of training examples, the ability of the model's prediction decreses when the dimensionality increses over an optimal value. This is also

called the **Curse of Dimensionality** and it can lead to *overfitting* the dataset - the model has a low power of generalization. Finding the best number of features can be a very hard problem (as it requires a lot of manual testing). Actually, this is an *intractable* problem, because we need to generate all possible combinations of features and find the optimal one. This could easily be avoided now by using Feature Selection tools. For example, **Random Forests** are very good models at selecting features that provide the best accuracy to the model. This will be analyzed with more details in the next sections of the thesis.

So, supposing that we have M number of examples in the training set and the feature tensor is one-dimensional, if we add another dimension to the tensor (another feature), ideally, we need to square the training examples. By induction, the number of training example grows exponentially with the dimension of the feature tensor. The reason behind this, is that when adding more features to the dataset by keeping the same number of examples, the space where our examples are distributed becomes sparser. So, in order to keep the same sparseness of the space, we must fill the higher dimensional space with more data, and that data must grow exponentially as the dimension of the space increases. Keeping the same size of the dataset will result in overfitting the data, which is bad for a model.

Figure 3.2[4] shows a representation of how the number of feature dimensions affects the quality of the learning model. It can be seen that keeping the training examples constant and only increasing the number of features, the accuracy drops by an exponential rate. Finding the optimal dimension of the feature space requires a lot of work and testing - there is not an universal recipe that does this yet, but methods such as Feature Selection or Feature Extraction are a good place to start.

3.2.3 Adding Complexity to the Model

There are situations in which is better to add complexity to our data. For example, when our dataset in not linearly separable using one, two, or more features, we can add extra dimensions to the feature tensor in order to make that data separable.

[4] http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/

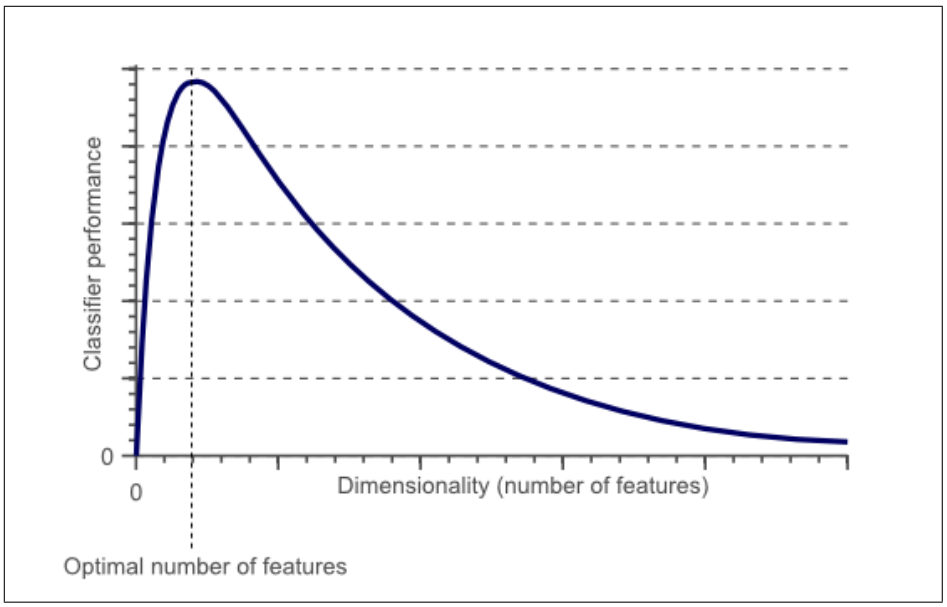

Figure 3.2: The Curse of Dimensionality

SVMs make use of this approach, by using multiple *kernel functions* that have the role to transform an input space in order to easily process data. Intuitively, a kernel is a "shortcut" that allows us to do certain computations, but without being directly involved in higher-dimensional calculations. Kernels can be linear functions, polynomial functions or even sigmoid functions. Using SVMs, we implicitly add complexity to our model.

Having tested some simple models on the above dataset, such as Linear Regression or the Perceptron, adding some complexity to the models was not at all a bad idea. A good method was the one of using **Polynomial Features**, that combines the initial dataset's features into new nonlinear features. Polynomial Features add more dimensions to the feature space, but the key here is that they are correlated, so this can help in achieving better prediction accuracy. Here, there are two different options to consider:

- The first one is generating a list of features (a polynomial of a certain degree) from the current features. Example: If we have the input given as (X_1, X_2), after the polynomial tranformation the example becomes $(1, X_1, X_2, X_1 * X_2, X_1^2, X_2^2)$
- The second approach was to consider only interactions between the features for building a polynomial with the same degree as the number of initial fea-

tures. Example: The features (X_1, X_2, X_3) are transformed by the polynomial into: $(1, X_1, X_2, X_3, X_1 * X_2, X_1 * X_3, X_2 * X_3, X_1 * X_2 * X_3)$, resulting in a total of 2^N final features, where N is the original dimension of the input space.

So, for Linear Regression, the input features are transformed using the first method (to generate non-linear functions like polynomials of degree 2 or 3). This "trick" allows us to use simple linear models that are trained on actual non-linear combinations of the data and are faster than other complex non-linear models. Supposing that we want to train our student's dataset using Linear Regression with Polynomial Features:

Let $\tilde{y}$ be the output vector of the linear model, x the input tensor, $\omega \in \mathbb{R}^{M \cdot N}$ the coefficient tensor (a two-dimensional vector) and β the vector bias. The model computes the following equation, making use of the "least squares" method for calculating ω and β:

$$\tilde{y}(\omega, x) = \omega \cdot x + \beta \tag{3.3}$$

where $x = (x_1, x_2, \ldots, x_9)$. When we add the polynomial features, x is transformed like this:

$$x_{nonlinear} = (x_1, x_2, \ldots, x_9, \ldots, x_i \cdot x_j, \ldots, x_1^2, x_2^2, \ldots, x_9^2); i, j = \overline{1, 9}, i < j$$

The size of the new feature space is: $9 + 9 + C_9^2 = 54$
Substituting $x_{nonlinear}$ in Equation 3.3, we obtain:

$$\tilde{y}(\omega', x_{nonlinear}) = \omega' \cdot x_{nonlinear} + \beta' \tag{3.4}$$

We can observe from the above equation that the linearity is still preserved and the model can fit more complicated data now.

On the other hand, the Perceptron model was tested using both methods, although the second method turned to be more appropiate, i.e. the *interaction features* method. This method was also used for the MLP[5] neural network model. With the size of the feature space of 9, adding interaction features provided a total of $2^9 = 512$ features.

Besides **Polynomial Features**, another way to add more features from existing features is to use **Trigonometric Features**, like $sin(x)$, $cos(x)$, etc. This can be useful when you only have two or three features in the dataset and want to increase the

[5]Multi-Layer Perceptron

input size a little more to see if the model (e.g. neural networks) can fit the data more precisely.

It is worth saying here that this approach of making the model more complicated is a good thing when having a big dataset of training examples. In our case, the training examples have a dimension of 80% of a total of 143 examples, which is approximately 115 examples. With this number, and with an input size of 9 features, generating polynomial features and increasing the input space to a much higher value does not scale very well, even if the features are correlated to each other: it will only make the data more complex and the model will give bad results, no matter how good it is, theoretically.

3.3 Visualizing the dataset

Machine Learning can be very counter-intuitive when we are dealing with the number of dimensions. Often, it implies working with hundreds, or even thousands of dimensions, and our human mind cannot reason easy about this (or not at all) when it comes to visualizing/imagining what is happening with the data or how it looks. Humans can think with no problem in two or three dimensions (even four, with some effort), so researchers in this field came up with some useful tools that do a *dimensionality reduction* of the data. This means transforming data from higher-dimension space to a human-meaningful lower-dimension space, with the purpose of having a view of how the dataset looks before trying to apply some Machine Learning models ot it. Also, another reasing for the reduction of the input space is to decrease the number of features in an optimal way (find the most important ones or create new fewer ones from the existing features), so the models can provide better results.

There are many tools that are used for the dimensionality reduction. Some examples are: **PCA**[6]**, MDS**[7] **and t-SNE**[8]. If visualization is what is needed, **Graph Based Visualizations** models are very helpful, as they give important insights into the structure of the data, i.e. how the points are connected to each other.

[6]Principal Component Analysis
[7]Multidimensional Scaling
[8]t-Distributed Stochastic Neighbor Embedding

3.3.1 PCA

For this work, the *PCA* technique was used to get information about our dataset and see how it looks, visually. PCA was not used for *feature extraction* purposes, since the aim was to focus on the models and current features and to get useful insights about them and their role in the student's performance.

What PCA does is to get the original input space and apply to it an orthogonal linear transformation in such a way that it will reduce the dimension of the input space and the new points will be spreaded the most in the new space. The reason behind this is to capture the most possible *variance* of the new data, so we can get a better look at them. Variance measures how much the data is distributed across the space, i.e. how far a set of points are spread out from their central point (thei mean value).

For example, considering the dataset fits in an *N-dimensional* hypercube, PCA will find the optimal angle to look at the data and then will project that data to two or three orthogonal axes, so it can be visualized. That angle will give the most variation in the data. However, the most variation does not always imply that the new dimensions can be used as new, alternative features for the learning algorithms, mostly when the dataset is not classifiable (it has a very high amount of noise).

3.3.2 Looking at the data

Since our dataset's feature space has a minimum dimension of 4 (when using just the Moodle logs features) and a maximum dimension of 17 (when using all the possible features - past results, semester grades and Moodle logs), PCA was applied multiple times for different subsets of the feature space with the goal of reducing its dimension. In this subsection, the dataset described in subsection 3.2.1 was used as an example.

In figure 3.3, it can be seen how the points are spread across the two principal axes in order to have the most variance. The first one, i.e. the horizontal one, represents the first principal component (with the highest variance) and the second one (the vertical axis) count for the second principal component (which has the second-

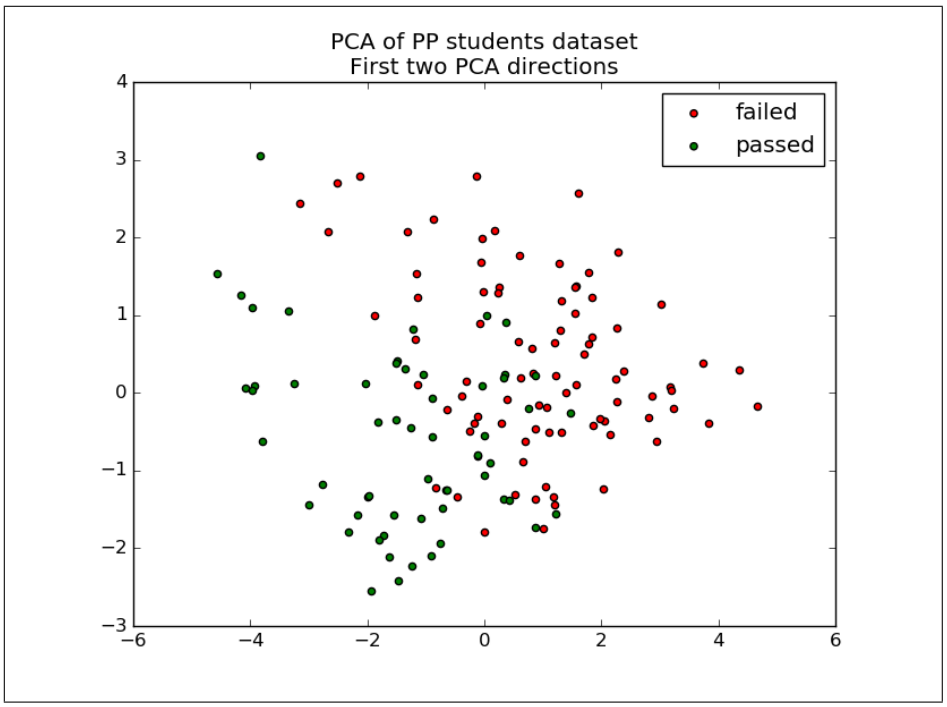

Figure 3.3: 2D PCA of the model

highest variance). After the plot was made, the points were colored based on their class membership. The red dots represent the students that failed the course and the green dots the ones that passed (for this example only the binary split of the examples was taken into consideration). We can see from the picture that the PCA algorithm manages to split the dataset without *knowing* about the label values. This is why PCA can be seen as an unsupervised learning model for classification that finds patterns in the data by its own.

Figure 3.4 show the same PCA analysis of the same dataset, but this time plotted in three dimensions, only to get another viewable perspective. The higher the number of dimensions, the more information the model has. So, using three instead of two dimensions for the PCA, we can get more insight into the structure of the data (e.g.: in three dimensions we can see if the data has a curvature, and this can be helpful).

Looking at the two visualizations made by the PCA analysis, some conclusions can be drawn about the dataset. First, we can observe that there is only a small mount of noise in the data, which is a good thing, because the models used will be able to classify students with a high accuracy. Second, PCA provides a safety

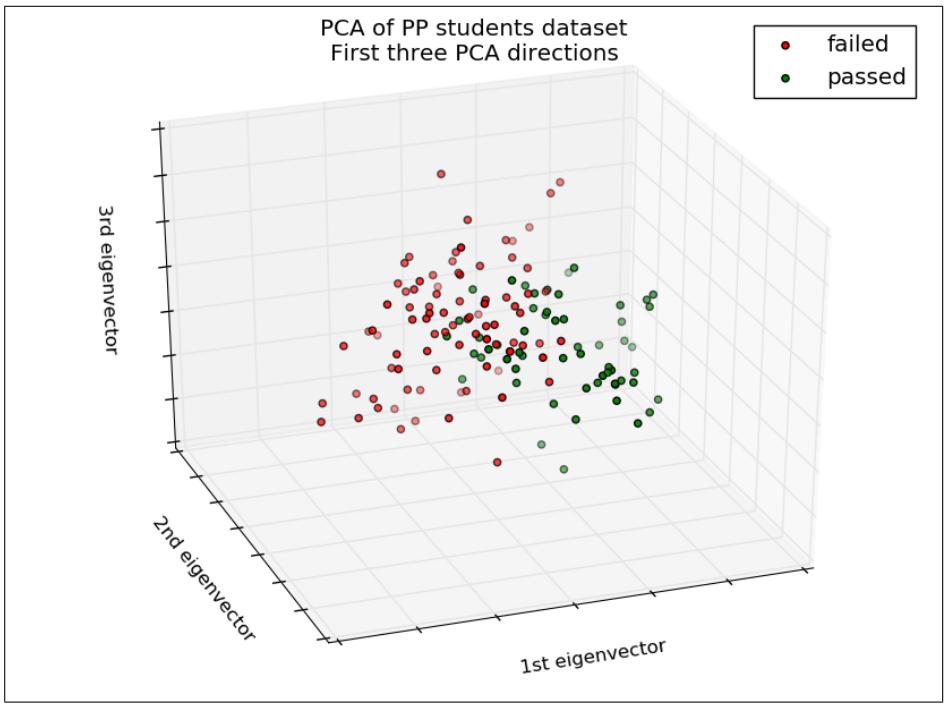

Figure 3.4: 3D PCA of the model

check before going further with building the model (if we do not get an overview of our dataset, we will not know how well is expected from it to perform).

3.4 The Models

After building the dataset, which implied choosing the examples and the subsets of features, the next step is to choose a Machine Learning model (or more) to start training the data. The first part of this section discusses the design of two simple models (the *Perceptron* and the *Linear Regression* model), while the second subsection expands the architecture and components of a more complex model: *Artificial Neural Networks*. Finally, in the third part the *Random Forest* model is analyzed, with focus on its structure and goals.

3.4.1 Simple Models

The Perceptron

The perceptron is a model used for binary classification. The algorithm is linear: it combines the input vector of features with a vector of weights that are adapted in the learning process, based on the sign value (-1 or +1) of the product between the real class label and the sign value of the linear term $\omega \cdot x + b$. When this process ends, the model can decide if an input belongs to a class or the other. Figure 3.5 shows the model's simple architecture.

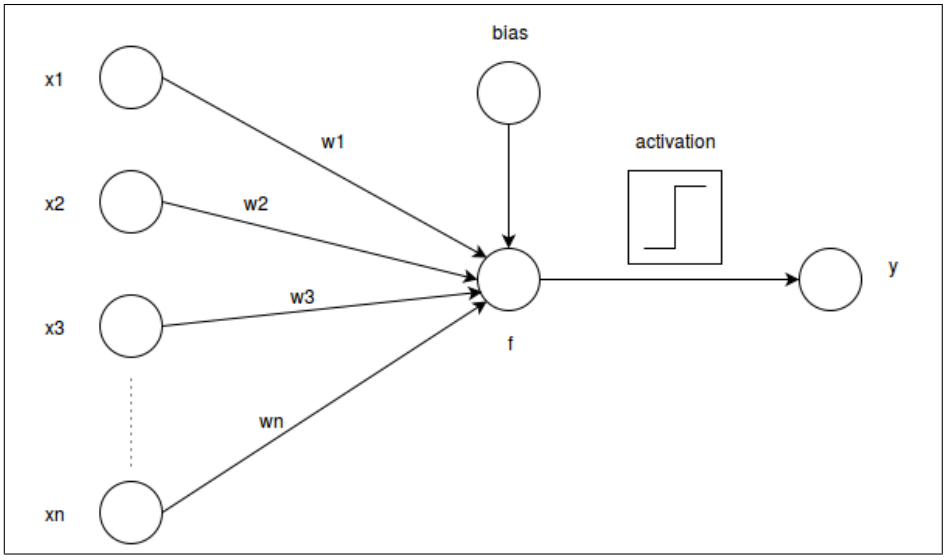

Figure 3.5: The Perceptron

This design was inspired by how a biological neuron works (it activates on certain learned thresholds) and it was the first step into developing *artificial neural networks*, which are a collection of perceptrons arranged in a layered network.

From the figure, we can see certain elements of the model: x and *omega* represents the input vector and the vector of weights, respectively; the *bias* has the role of adjusting the boundary position between the classes; f and the *activation* function represents the *perceptron algorithm* and y is its output, i.e. -1 or 1.

Linear Regression

The Linear Regression model uses the *ordinary least squares* method to solve an optimization problem that has the form:

$$min_{\omega}||\omega \cdot x - y||_2^2 \tag{3.5}$$

Starting from the standard equation $y = \omega \cdot x + \beta$, the method is used to find the ω and β parameters of the equation that satisfy the above optimization problem. The solutions are:

$$\omega = (x^T x)^{-1} x^T y;\ \beta = y - \omega \cdot x \tag{3.6}$$

Note: since this an optimization problem, *gradient descent* cand also be used to find its solution, but with an iterative approach, not analytically like the first method. The cost function used by this algorithm has a form similar to the optimization problem mentioned above in equation 3.5:

$$Loss(\omega) = \frac{1}{2}||\omega \cdot x - y||_2^2 \tag{3.7}$$

where $\frac{1}{2}$ is a factor used only as a convenient when calculating the first derivative of the cost function:

$$\frac{\partial Loss(\omega)}{\partial \omega_j} = \frac{\partial}{\partial \omega_j}\frac{1}{2}(\omega x - y)^2 = (\omega x - y)x_j, for\ all j = \overline{1, N} \tag{3.8}$$

where N is the input size of an example.

The algorithm uses this partial derivative to repeatedly update the weight vector ω, until convergence:

$$\omega_j := \omega_j - \alpha\frac{\partial}{\partial \omega_j}Loss(\omega),\ for\ every\ j. \tag{3.9}$$

where α is the learning rate of the algorithm.

Combining this with the *polynomial features* discussed in subsection 3.2.3, we can extend the model in order to perform better at fitting the data, while still preserving its linearity.

3.4.2 Neural Networks

3.4.3 Random Forest

Appendix A

Appendices

A.1 Neural networks

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <services>
    <service-include file-path="remoting-config.xml" />
    <default-channels>
      <channel ref="my-amf"/>
    </default-channels>
  </services>
</services-config>
```

List of Figures

Bibliography

[1] Mehdi S. M. Sajjadi, Morteza Alamgir, Ulrike von Luxburg. *Peer Grading in a Course on Algorithms and Data Structures: Machine Learning Algorithms do not Improve over Simple Baselines.* In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale,* pp. 369-378, 2016.

[2] Siddharth Reddy, Igor Labutov, Thorsten Joachims. *Learning Representations of Student Knowledge and Educational Content. Cornell University,* 2015.

[3] Michael Wu. *The Synthetic Student: A Machine Learning Model to Simulate MOOC Data,* MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2015.

[4] Saeed Hosseini Teshnizi and Sayyed Mohhamad Taghi Ayatollahi. *A Comparison of Logistic Regression Model and Artificial Neural Networks in Predicting of Student's Academic Failure.* In *Acta Inform Med.,* 23(5): 296–300, 2015 Oct.

[5] Jeng-Fung Chen, Ho-Nien Hsieh and Quang Hung Do. *Predicting Student Academic Performance: A Comparison of Two Meta-Heuristic Algorithms Inspired by Cuckoo Birds for Training Neural Networks*. In *Algorithms,* 7(4), 538-553, 2014.

[6] B.A., Kalejaye, O., Folorunso and O.L., Usman. *Predicting Students' Grade Scores Using Training Functions of Artificial Neural Network*. In *Journal of Natural Sciences, Engineering and Technology, Volume 14,* 2015.

[7] V.O. Oladokun, Ph.D., A.T. Adebanjo, B.Sc., and O.E. Charles-Owaba, Ph.D. *Predicting Students' Academic Performance using Artificial Neural Network: A Case Study of an Engineering Course*. In *The Pacific Journal of Science and Technology, Volume 9,* pp. 72-79, 2008.

[8] Emaan Abdul Majeed and Khurum Nazir Junejo. *GRADE PREDICTION USING SUPERVISED MACHINE LEARNING TECHNIQUES.* In *E-Proceedin*

of the 4th Global Summit on Education, pp. 224-234, 2016.

[9] Amelia Zafra and Sebastián Ventura. *Predicting Student Grades in Learning Management Systems with Multiple Instance Genetic Programming*. In *Educational Data Mining*, 2009.

[10] Bin Xu and Dan Yang. *Motivation Classification and Grade Prediction for MOOCs Learners*. In *Comput Intell Neurosci*, 2016 Jan.

[11] G. Hughes. *On the mean accuracy of statistical pattern recognizers. In IEEE Transactions on Information Theory, Volume 14*, pp. 55-63, 1968.