Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2018-2019

Project title: Insight into the grades of Romanian students
Tool: Scikit-Learn

Name: Csaba-Laszlo Gabor
Group: 30434
Email: csabagabor97@gmail.com

Contents

Chapter 1

Overview

1.1 Installing the tool

1. Make sure **Python** is installed on the computer(check it with *python –version*). If it is not, install it from the official Python website([7]).
2. Make sure **pip** is installed on the computer. Pip comes pre-installed with new Python releases.
3. To install the tool as a standalone version, just type `pip install scikit-learn` in the terminal
4. These tools: **numpy**, **pandas**, **matplotlib**, **seaborn** need to be installed to process data, to plot functions and to use mathematical functions so for every tool just type `pip install {name of the tool}`
5. Also it is highly recommended to install **Jupyter Notebook** which is a code editor in which separate pieces of code can be executed, commented, plots can be made etc. To install it, type: `pip install jupyter` in the terminal

1.2 Overview of the tool

Scikit-learn is an open-source machine learning library for the Python programming language. It includes several classification and regression algorithms including support vector machines, random forests etc.

Machine learning is all about predicting some labels in unseen data like predicting the height of new children if their age, gender and health condition is known(in this case height is a label while age, gender and health condition are features which help to predict labels). **Scikit-learn** helps in this by automizing some algorithms so the end programmer does not have to know how these algorithms work under the hood (of course it is recommended to have a basic understanding of them - but the programmer does not have to implement these functions from scratch) so he can start working on a problem from the beginning.

Chapter 2

Main Features

2.1 Machine learning Algorithms

There are 2 big types of algorithms:

1. **Classification**: it means that the label we want to predict can have limited possible values. For example classifying if an apple is red or green is a classification problem.

2. **Regression**: regression methods are used to predict a continuous value which can have an unlimited number of values like predicting the cost of a house(this can be 1000 dollars but it can also be 1000.0001 dollars...).

For both of these two categories, **Scikit-learn** comes with predefined algorithms.

1. **Classification**: DecisionTreeClassifier(), RandomForestClassifier(), KNeighborsClassifier(), SVC() etc.

2. **Regression**: LinearRegression(), KNeighborsRegressor(), RandomForestRegressor() etc.

These algorithms behave differently for different datasets, so there isn't a best choice between them. Another constraint which needs to be considered is time. There are some really good algorithms like the **Random Forest** but requires more time to run than other simpler algorithms such as **LinearRegression**.

2.2 Other Algorithms

We don't need just machine learning algorithms. We also need algorithms for other tasks such as:

1. splitting the dataset into training and test datasets with **train_test_split()**

2. cross-validation with **cross_val_score()**

3. testing the accuracy of our predictions with **accuracy_score()**

Scikit-Learn comes in handy because it already contains algorithms for these subtasks, so no need to develop our own ones.

Chapter 3

Algorithm - Details

3.1 General Description

The algorithm I've chosen is **Linear Regression**. Probably it is the easiest one to understand and the most intuitive. Imagine that we are given a number of features and we have to predict a label. In this case we want to assign a weight to each feature which shows how much the label depends on it.

$$y' = b + w_1 x_1 + w_2 x_2 + w_3 x_3$$

The picture shows that **y'** is the label we want to predict, **x1**, **x2**, **x3** are the features and **w1**, **w2**, **w3** are the weights. b is called bias.

In the first phase we assign a random value to each weight. Next, based on those values we calculate the predicted label values for all entries in the dataset and compare the results with the correct results. The average of the absolute difference between the correct results and our results is the **training error**. In other words this is our **loss** . Our goal is to minimize this loss. Of course there are several types of errors like:

1. **Mean Absolute Error**: this was discussed previously. This error is calculated easily but has one problem: for this error calculation it does not matter how far the predicted values are from the real values. This is when the Mean squared error helps.

$$MAE = \frac{\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid}{n}$$

2. **Mean squared error**: which calculates the power of 2 of the loss difference. It favors closer values over further values from the real values. **This method is usually used**.

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

3. **Mean Bias Error**: Same as Mean Absolute Error only that we don't take the absolute value of the difference. It is not used often.

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

There are other types of errors which are not discussed here.

We have a loss function now which depends on the weights. Our goal is to find the minimum of this function. The algorithm calculates the new values for the weights using the **gradient descent method**. The general idea of this method is to calculate the partial derivatives which indicate in which direction we have to move to minimize the loss. We can make a big step in that direction or a smaller step. This is called the **learning rate**. A small learning rate will give us the correct solution the same way the bigger one calculates the correct solution but it needs more calculations and more time. On the other hand a too big learning rate can make the algorithm fail to converge. So it won't find the minimum error(the minimum of the error function). Having a very good learning rate is hard to achieve.

3.2 Algorithm in Scikit-Learn

Scikit-Learn comes with a built-in function for Linear regression. This function initializes a LinearRegression object:

LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None)

1. The **fit_intercept** is a *boolean* parameter. It specifies if we want to include an intercept(bias) to the model. By default it is set to *True*.
2. The **normalize** is a *boolean* parameter. If set, it normalizes the data set.
3. The **copy_X** is a *boolean* parameter. If set, it copies the original `X` passed to the function. If it is *False* then the original `X` may change after running the algorithm.
4. The **n_jobs** is an *int* parameter. It is the number of jobs to use for the computation(to use more cores in a CPU). By default it is set to *None*.

After initialization, the **fit()** method needs to be called on the training set. This function trains the model.

fit(X, y, sample_weight=None)

`X` is the set of features and `y` is the set of labels. With `sample_weight` we can add individual weights for each sample.
After training our model, we have to verify its accuracy. To do this, we run the **predict()** function.

predict(X)

`X` is the feature set in the test set. This function returns the set of the predicted labels.

To check the mean square error of our model on the test set, we can use the `mean_squared_error(y_test, y_pred,...)` function where `y_pred` is the set returned by the `predict(X)` function and `y_test` is the original label set for the test set obtained after splitting our data set(it contains the correct results).

We can also check the accuracy using the `accuracy_score` function.

Simple usage:

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
y_pred = regressor.predict(X_test)

#To retrieve the intercept:
print(regressor.intercept_)

#For retrieving the slope:
print(regressor.coef_)

print('Mean Squared Error:', mean_squared_error(y_test, y_pred))
```

Chapter 4

examples

4.1 Provided Examples

The next example is taken from [8] and [6]. It uses the Heart Disease UCI dataset from Kaggle[5]. The data set includes features like *age, cholesterol level of the blood or gender*. The example uses the **Random Forest Classifier** which is a decision tree, but replicated multiple times with a randomly taken set of features and the final result is the average of the partial results in the trees. The example predicts if a person has a heart disease or not.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error

#GET THE DATA
df = pd.read_csv('heart.csv')

#get the first 5 rows of data
df.head()

#information for the values
df.info()

# We must clean the NaN values, since we cannot train
# our model with unknown values.
# Luckily, there is nothing to clean.
df.isna().sum()

# First of all let's see how many zeros and ones do we have...
negative_target = len(df[df.target == 0])
```

```
positive_target = len(df[df.target == 1])
sns.countplot(x = "target", data = df, palette = "pastel")
plt.xlabel("Target (0 = no, 1= yes)")
plt.ylabel("count")
plt.show()

# It is a classification problem, and more precisely a BINARY
    CLASSIFICATION.
# we divide into two subsets: the training data and the testing data
# X are the explanatory variables, Y is the response variable
    ('target'):
X = df.drop('target', axis=1)  # everything except target.
Y = df['target']               # only target.

# Since the amount of data is not extremely large, we will use a
    small test_size (0.10-0.15).
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
    test_size=0.15)

# Firstly, let's take Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100)

#train the model
rfc.fit(X_train, Y_train)

#predict unseen data
rfc_predictions = rfc.predict(X_test)
test_error = accuracy_score(rfc_predictions, Y_test)
test_error

# get importances from RF
importances = rfc.feature_importances_

# then sort them descending
indices = np.argsort(importances)

# get the features from the original data set
features = df.columns[0:13]

# plot them with a horizontal bar chart
plt.figure(1)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b',
    align='center')
plt.yticks(range(len(indices)), features[indices])
plt.xlabel('Relative Importance')
```

After running the program we get an accuracy score of 89%. The example also makes a plot which shows the relative importance of every feature, the most important being **thelach** feature which is *the maximum heart rate achieved*.

4.2 Own Examples

We are going to predict the final baccalaureate grades of students if we know their grades from 8th Grade(*Evaluare Nationala*). So we'll have only one feature(the grade from 8th Grade) and one label(final baccalaureate grade). This time we will run a simple Linear Regression.

After importing the necessary modules, we read the data and check some of its attributes such as size.

```
df = pd.read_csv('note.csv')
df.keys()
df.shape
df.head()
df['romana final']
df['nota la limba romana']
```

Now we split the data in training and test set and train our model:

```
X = df['nota la limba romana'].values.reshape(-1,1)
Y = df['romana final'].values.reshape(-1,1)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
    test_size=0.20)
regressor = LinearRegression()
regressor.fit(X_train, Y_train) #training the algorithm
```

Now we predict unseen data and show the error and accuracy score:

```
Y_pred = regressor.predict(X_test)
mean_squared_error(Y_test, Y_pred)
mean_absolute_error(Y_test, Y_pred)
regressor.score(X_test, Y_test)
res = pd.DataFrame({'Note evaluare':X_test.ravel(),
'Predicted':Y_pred.flatten(), 'Actual': Y_test.flatten()})
res
```

We get a mean squared error of 2.18 and an absolute error of 1.15. The accuracy score is 45%. Showing the intercept and the coefficient:

```
print(regressor.intercept_)
print(regressor.coef_)
```

We get 0.794 for the coefficient and 0.88 for the intercept. This is an interesting result and shows that the results from the Baccalaureate are worse than the ones from the previous exam. Of course this was expected because the Baccalaureate is a harder exam.
Before judging the results, let's run another example. This time we replace every value in the feature column with the average in that column.

```
avg = df[' nota la limba romana'].mean()
df[' nota la limba romana'] = avg
X = df[' nota la limba romana'].values.reshape(-1,1)
Y = df['romana final'].values.reshape(-1,1)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
    test_size=0.20)
regressor = LinearRegression()
regressor.fit(X_train, Y_train) #training the algorithm
Y_pred = regressor.predict(X_test)
mean_squared_error(Y_test, Y_pred)
mean_absolute_error(Y_test, Y_pred)
```

This time we get a mean squared error of 3.86 and an absolute error of 1.63. This shows that, despite the fact that in our previous example, we achieved a low accuracy score, there is a high correlation between this feature and this label.

Chapter 5

Problem specification

5.1 General specification

My project consists of predicting the grades of students in Romania at the Baccalaureate. Also other predictions will be carried out such as predicting the failure of students, the chance to have an exam in a minority language based on the county, predicting the grades for optional subjects for those who had an exam at the same subject 4 years earlier and for those who didn't have etc. Plots will be drawn to visualize the data better. The aim of this problem is not just to have a high accuracy but to carry out as many predictions as we can and to analyze those predictions.

5.2 Data

The data from the 2012's Baccalaureate is used. This data is taken from an Open sourced source [10]. It contains two hundred thousand rows. But we also need data for the grades from 4 years earlier at the *Evaluare Nationala* which will be scraped from the well-known *http://web.archive.org* domain. The extracted data needs to be merged with the first one to form a single unit. The merge will be done on the Name(*nume*) column. We want to have a single data unit for a single person with his/her grades from both exams. The second dataset also contains the average grades for the 4 years in school from grade 5 to grade 8. The final dataset has the following columns: *nume, judet8, scoala, media la admitere, media teze nationale, media de absolvire, nota la limba romana, nota la matematica, optional8, optional8 nota, limba materna8, limba materna8 nota , Unnamed: 13, pozitia in ierarhie pe judet, pozitia in ierarhie pe tara, unitatea de invatamant, judetul, promotie anterioara, forma invatamant, specializare, romana oral, romana nota, romana contestatie, romana final, limba materna, materna oral, materna nota, materna contestatie, materna final, limba moderna, limba moderna nota, disciplina profil, disciplina profil nota, disciplina profil contestatie, disciplina profil final, optional, optional nota, optional contestatie, optional final, competente digitale, media, rezultatul final*

Some columns which were redundant like the first column which contained information which remained from the scraping process were removed. The data also contained some duplicates because there were students with identical names(in this case the merging process could have merged 2 **different** students with identical names - every such student should have been checked manually and added to the data set but because there were only few of them every duplicate was eliminated).
Some info about the type of the data for every column:

```
nume                             10237 non-null object
judet8                           10237 non-null object
scoala                           10237 non-null object
media la admitere                10237 non-null object
media teze nationale             10237 non-null float64
media de absolvire               10237 non-null float64
nota la limba romana             10237 non-null float64
nota la matematica               10237 non-null float64
optional8                        10237 non-null object
optional8 nota                   10237 non-null object
limba materna8                   10237 non-null object
limba materna8 nota              10237 non-null object
pozitia in ierarhie pe judet     10237 non-null int64
pozitia in ierarhie pe tara      10237 non-null int64
unitatea de invatamant           10237 non-null object
judetul                          10237 non-null object
promotie anterioara              10237 non-null object
forma invatamant                 10237 non-null object
specializare                     10237 non-null object
romana oral                      10199 non-null object
romana nota                      10237 non-null float64
romana contestatie               1108 non-null float64
romana final                     10237 non-null float64
limba materna                    944 non-null object
materna oral                     943 non-null object
materna nota                     944 non-null float64
materna contestatie              64 non-null float64
materna final                    944 non-null float64
limba moderna                    10237 non-null object
limba moderna nota               9176 non-null object
disciplina profil                10237 non-null object
disciplina profil nota           10237 non-null float64
disciplina profil contestatie    1205 non-null float64
disciplina profil final          10237 non-null float64
optional                         10237 non-null object
optional nota                    10237 non-null float64
optional contestatie             751 non-null float64
optional final                   10237 non-null float64
competente digitale              10184 non-null object
media                            10237 non-null float64
rezultatul final                 10237 non-null object
```

We can see that *Pandas* recognized most of the columns with grades as float values and the columns with strings as object values. The other columns for which the type was inferred incorrectly will be corrected manually. Before we run any algorithm on string columns, we have to encode them with the power of *scikit*. This is a rather complex data set with 42 columns and more than 10000 rows(when merging the 2 datasets only a portion of the names were common in both datasets so the final size of the dataset is smaller than the initial ones - some students didn't finish high school, others didn't go to the Baccalaureate exam - the reasons can be many). Also, the column names are self-explanatory.

5.3 Scraping the Data

For scraping, the following *JavaScript* code was used. This code can be just copied in the console of any modern browser after injecting/importing *jQuery*.

After selecting the county in [1], the following piece of code extracts the links which take us to the pages of schools.

```
a = $(".tdc > .lnk");

var hrefs = new Array();
a.each(function(){
  hrefs.push($(this).attr('href'));
})
```

```
$("body").html("");


hrefs.forEach(function(item) {
        $.get(window.location.href+item, function(data){
         var t = $("<div id='" +item +  "'></div>");
         t.prepend(data);
         $('body').append(t);
         });
});
```

The following code extracts the links which take us to the tables where the students are shown in each school and loads the tables from the saved links into different *div* elements.

```
var hrefs = new Array();
$('body').children('div').each(function () {
     var id = $(this).attr('id');
      a = $(".tdc > .lnk",this);

     a.each(function(){
        hrefs.push( id + $(this).attr('href'));
     })
});


$("body").html("");

var t = $('<table width="88%" align="center" cellspacing="0"
   class="mainTable" id="table"><tbody
   id="tbody"></tbody></table>');
$('body').append(t);


hrefs.forEach(function(item) {
        $.get(window.location.href+item, function(data){
         var t = $("<div></div>");
         t.prepend(data);
         $('body').append(t);

         });
});
```

This code contains 2 functions which were taken from[4] and are used to save the data in a table into a .csv file. Also it merges the small tables into a big one.

```
function download_csv(csv, filename) {
    var csvFile;
    var downloadLink;

    // CSV FILE
    csvFile = new Blob([csv], {type: "text/csv"});

    // Download link
    downloadLink = document.createElement("a");

    // File name
    downloadLink.download = filename;

    // We have to create a link to the file
    downloadLink.href = window.URL.createObjectURL(csvFile);

    // Make sure that the link is not displayed
    downloadLink.style.display = "none";

    // Add the link to your DOM
    document.body.appendChild(downloadLink);

    // Lanzamos
    downloadLink.click();
}



function export_table_to_csv(html, filename) {
        var csv = [];
        var rows = html.querySelectorAll("table tr");

    for (var i = 0; i < rows.length; i++) {
                var row = [], cols = rows[i].querySelectorAll("td,
                   th");

        for (var j = 0; j < cols.length; j++)
            row.push(cols[j].innerText);

                csv.push(row.join(","));
        }

    // Download CSV
    download_csv(csv.join("\n"), filename);
}
```

```
bodies = $('#mainTable > tbody');

var t = $('<table width="88%" align="center" cellspacing="0"
    id="BIGtable"><tbody id="tbody"></tbody></table>');
$('body').append(t);

bodies.each(function(){
  $(this).children('tr').not(':first').each(function(){
    t.append($(this));
 })
})
```

This final code saves the table in a .csv file.

```
var html = document.getElementById("BIGtable");
export_table_to_csv(html, "table.csv");
```

Note that these snippets must be run after each other, one at a time, with some time between them because the loading of data is done using *Ajax* and it needs time to load the data.

5.4 Merging the Data

Loading the data and transforming the values in the *nume* column to lowercase and stripping the whitespace chatacters. Then merging the result into a single *DataFrame*.

```
df1 = pd.read_csv('eval.csv')
df2 = pd.read_csv('bac.csv')
df1.nume = df1.nume.str.lower()
df2.nume = df2.nume.str.lower()
df1.nume = df1.nume.str.strip()
df2.nume = df2.nume.str.strip()
df = pd.merge(df1, df2, how='inner', on=['nume'])
```

Duplicates in the *nume* column were identified and eliminated.

```
sum(df['nume'].duplicated())
df.drop_duplicates(subset='nume', keep=False, inplace=True)
```

Columns without values were identified.

```
df.isna().sum()
```

```
nume                                0
judet                               0
scoala                              0
media la admitere                  14
media teze nationale                0
media de absolvire                  0
 nota la limba romana               0
nota la matematica                  0
optional8                           0
optional8 nota                      0
limba materna8                      0
limba materna8 nota                 0
pozitia in ierarhie pe judet        0
pozitia in ierarhie pe tara         0
unitatea de invatamant              0
judetul                             0
promotie anterioara                 0
forma invatamant                    0
specializare                        0
romana oral                        48
romana nota                         0
romana contestatie               9506
romana final                        0
limba materna                    9681
materna oral                     9683
materna nota                     9681
materna contestatie             10601
materna final                    9681
limba moderna                       0
limba moderna nota               1093
disciplina profil                   0
disciplina profil nota              0
disciplina profil contestatie    9407
disciplina profil final             0
optional                            0
optional nota                       0
optional contestatie             9882
optional final                      0
competente digitale                62
media                               0
rezultatul final                    0
```

It is unacceptable to have rows in which *media la admitere* column does not have a value so these rows(14 rows in total) will be eliminated.

```
df.dropna(subset=['media la admitere'], how='all', inplace = True)
```

Also, after the scraping process, some columns have invalid data such as the *media la admitere* which has *string* in some columns. These rows will be deleted. Also, other rows with NaN values will be deleted except those from columns like *limba materna8 nota* - where the majority of rows contain NaN values anyway.

```
 df[df.columns[3]] = df[df.columns[3]].apply(pd.to_numeric,
     errors='coerce').fillna(0).astype(float)
df = df[df['media la admitere'] != 0]

df['optional8 nota'] = df['optional8 nota'].astype(float)
df = df.dropna(axis=0, how='any', subset=['competente digitale'])
df = df.dropna(axis=0, how='any', subset=['romana oral'])
```

Finally we save the corrected data set:

```
 df.to_csv("note.csv", index=False)
```

The final dataset contains 10112 rows and 42 columns and the column types are identified correctly (except the column *limba materna8 nota* because only some students had this exam and '-' appears in some places, but these lines will be removed when this column is included in an analysis).

```
nume                          10172 non-null object
judet8                        10172 non-null object
scoala                        10172 non-null object
media la admitere             10172 non-null float64
media teze nationale          10172 non-null float64
media de absolvire            10172 non-null float64
nota la limba romana          10172 non-null float64
nota la matematica            10172 non-null float64
optional8                     10172 non-null object
optional8 nota                10172 non-null float64
limba materna8                10172 non-null object
limba materna8 nota           10172 non-null object
pozitia in ierarhie pe judet  10172 non-null int64
pozitia in ierarhie pe tara   10172 non-null int64
unitatea de invatamant        10172 non-null object
judetul                       10172 non-null object
promotie anterioara           10172 non-null object
forma invatamant              10172 non-null object
specializare                  10172 non-null object
romana oral                   10134 non-null object
romana nota                   10101 non-null float64
romana contestatie            1103 non-null float64
romana final                  10101 non-null float64
limba materna                 944 non-null object
materna oral                  943 non-null object
materna nota                  934 non-null float64
materna contestatie           64 non-null float64
materna final                 934 non-null float64
limba moderna                 10172 non-null object
limba moderna nota            9111 non-null object
disciplina profil             10172 non-null object
disciplina profil nota        10088 non-null float64
disciplina profil contestatie 1196 non-null float64
disciplina profil final       10088 non-null float64
optional                      10172 non-null object
optional nota                 10088 non-null float64
optional contestatie          747 non-null float64
optional final                10088 non-null float64
competente digitale           10119 non-null object
media                         10111 non-null float64
rezultatul final              10172 non-null object
```

Chapter 6

Related work

As far I know nobody has published an analysis on the Baccalaureate grades in Romania but I've found related work from other countries which predict the grades of students. These are [9], [2] and [3].

[9] works on the prediction of the grades of Portuguese students in 2 subjects: Mathematics and Portuguese language. It works on a dataset which includes several features like sex, age, family size, the job of the father, the job of the mother, extra-curricular activities etc. This data set isn't similar to mine in any aspect because it just contains information about the environment of the student and wants to see in which environments students achieve better. My data set is more concrete than this because it has some concrete past data about students such as their past grades. The majority of Kernels which try to solve this problem use decision trees which seems a good choice in this case.

[2] is the most similar to my problem because it is also about the Baccalaureate grades(in another country). But the data set only contains 8 features, and the problem with these 8 features is the same as with [9] - they are too general. There is only a single Kernel working on this data set which tries to visualize the data with plots.

[3] is the most interesting one because it tries to solve the general problem of predicting grades, not just a local one. It tries using *Bayesian Linear Regression*. The data set it uses is a broad one with 33 features[11]. The good part about this tutorial is that it uses the same tools I'll be using. After following this tutorial I realized that making plots of the data, making plots which show the relation between features is a really important thing because it helps in choosing the best features for the current analysis. Overall, this related work is worth reading.

Chapter 7

Preliminary Results

Two separate experiments were carried out. Code is not pasted here but it is attached and every step is commented to better understand the reason of the steps that were done.

7.1 Predicting the target value `romana final`

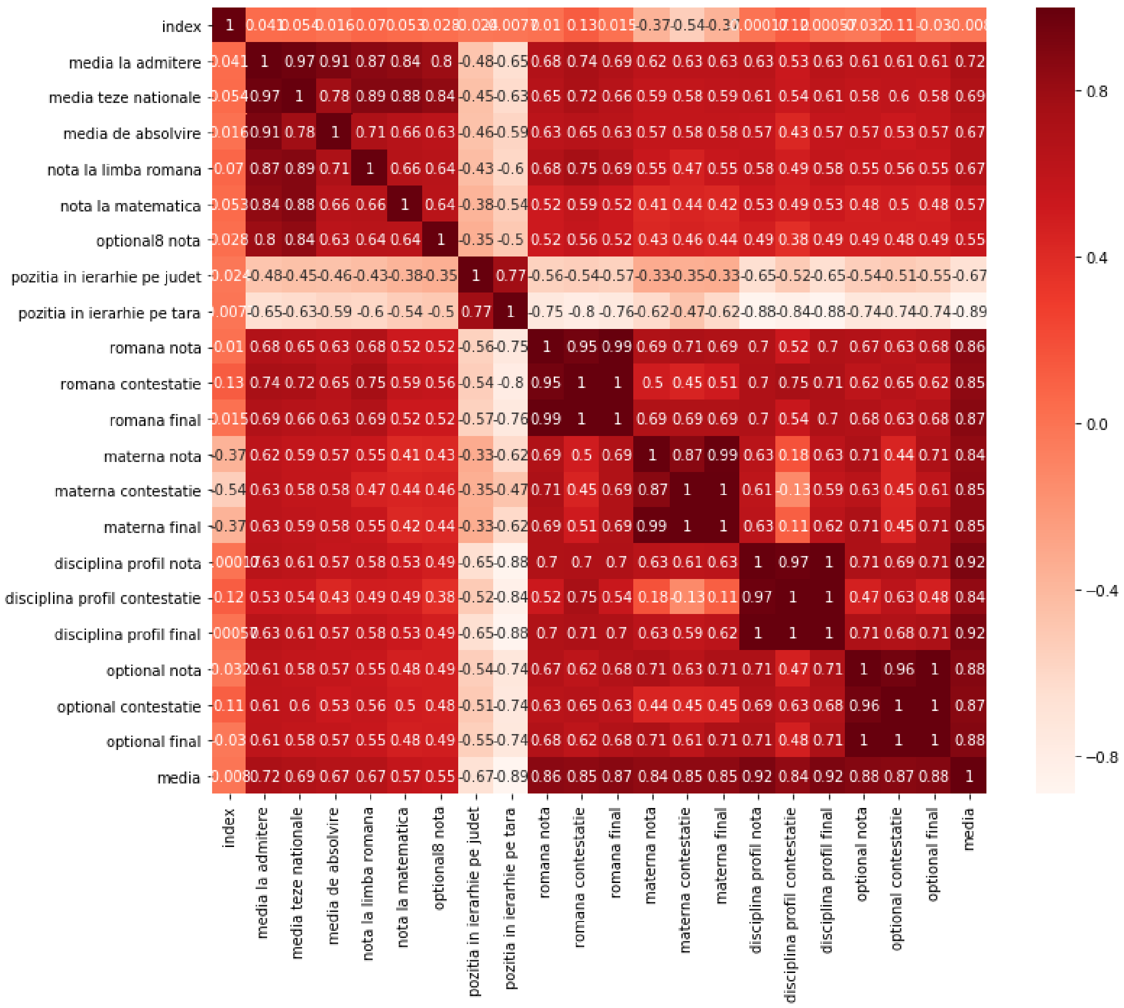
`romana final` is the grade for the final Baccalaureate grade in Romanian language. It is a continuous value ranging from 0 to 10. So, this is a regression problem. Some information about the target value:

```
count    10112.000000
mean         6.893004
std          2.040470
min          0.000000
25%          5.450000
50%          7.200000
75%          8.600000
max         10.000000
Name: romana final, dtype: float64
```

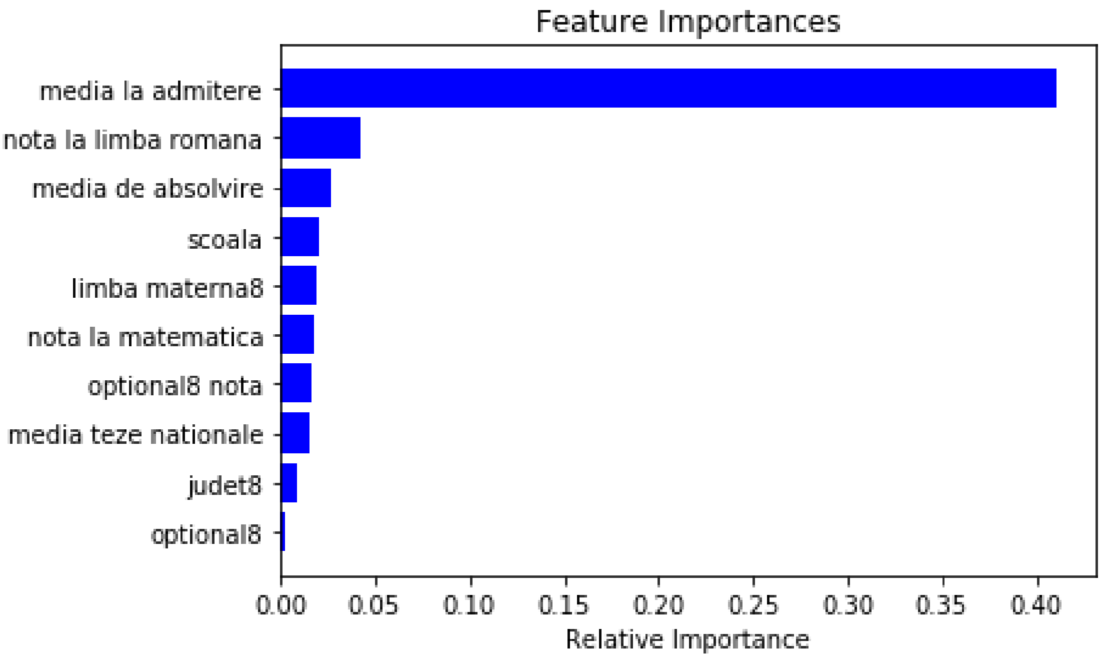
This was a tough experiment because a lot of feature engineering needed to be done, namely:

1. Removing highly correlated columns, because there are 42 columns in total with many categorical features and we must speed up the algorithms somehow. Besides this, some algorithms don't work well with highly correlated features such as *Linear Regression.* Several features were excluded because of correlation such as '..nota' and '..contestatie' for every type of grade because the '...final' column contains the final grade which is of our interest. Yes, some observations could have been made regarding who goes to dispute the grade(*contestare*), but because the columns contain sparse data I excluded them to speed up the algorithm. We can see that 'media teze nationale' and 'media la admitere' are highly correlated(because one is used in the calculation of the other one). Also, 'nota la limba romana' and 'optional8 nota' and 'nota la matematica' are highly correlated with both the 'media la admitere' and 'media teze nationale' These are obvious because some of the columns are just the average of other columns, so these will be removed. But there is a neat trick here: we won't include the 'limba materna8 nota' feature because it has many empty values, but instead we include both the 'media teze nationale' and the

grades for other subjects because in this way 'media teze nationale' will contain in itself the value for 'limba materna8 nota'.



2. Transforming categorical features. **Random Forest** can work well with *Label Encoded* features and it is also quicker. One hot encoded features were used initially with Random Forest to see the feature importance for every subcategory. In this way, we got some interesting results such as *Harghita* county in itself is a very good predictor for the final target value. *Nume* feature was replaced with firstnames, and students can contain multiple firstnames so they were one-hot encoded to allow multiple firstnames for a single person. Also, 'limba moderna nota' was a categorical feature but was replaced with a single number which is the average of the grades got at the exam with *A1 = 1* and *B2 = 4*. For those who have taken another language exam, their grades were replaced with the maximum grade of 4.0.



After running a *Random Forest Regressor* we got an accuracy of 70%. Cross validation showed a score of 67% which is still a high value. Several other algorithms were tried. Only one was with success(**GradientBoostingRegressor**) with an accuracy score of 72%.

The next step will be to remove outliers because probably they are causing overfitting in our model.

7.2 Predicting the target value `rezultatul final`

`rezultatul final` is the column which shows if a student passed the final Baccalaureate. So, it makes sense to not include grades from the Baccalaureate subjects in the features because we'll get a high accuracy, but without any general knowledge. What we'd like to do instead is to predict this value **ONLY** knowing the grades from *Evaluare Nationala* and eventually some categorical values from the Baccalaureate such as the name of the optional subject(but **NOT** the grade itself, just the name) or the type of the High School class(*Mathematics, Computer-Science, etc.*).

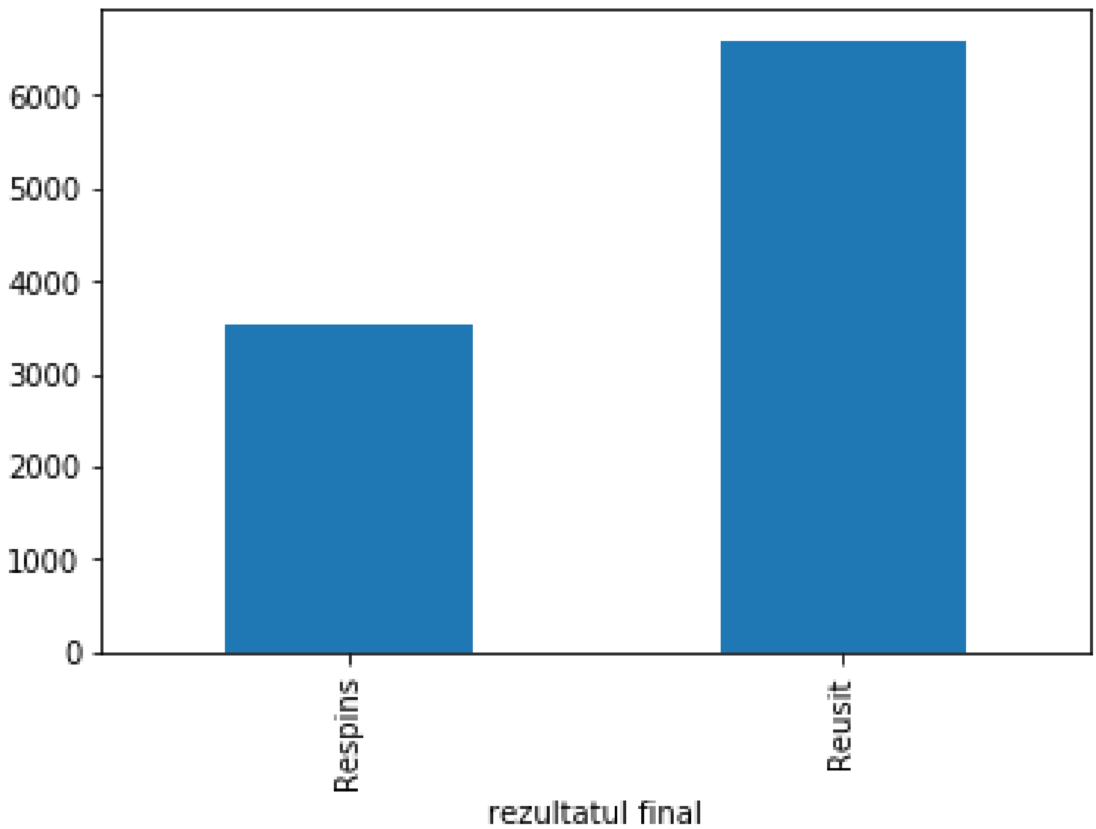
In this experiment, feature engineering wasn't so heavy, only some categorical features had to be transformed into numerical ones like 'specializare', 'limba moderna', 'disciplina profil' etc.

After running a *Random Forest Classifier* we got an accuracy of 83%. For classification problems, it is important to calculate the confusion matrix which shows if there are many false negatives or false positives(this usually happens because the data is unbalanced). This wasn't the case here:

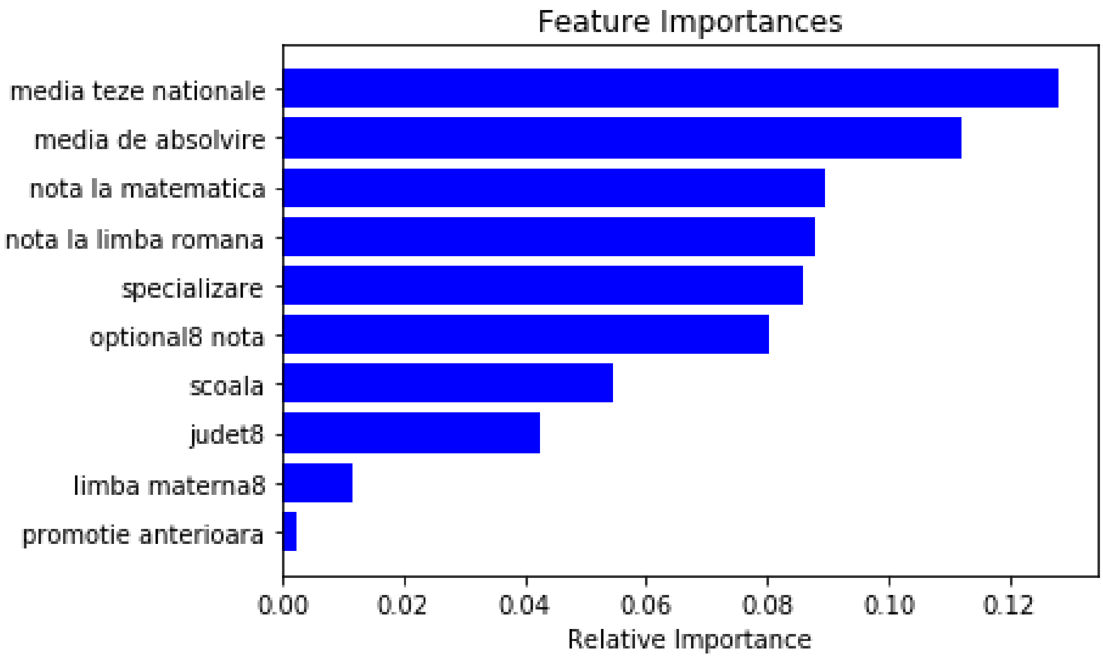
```
#in a classification problem they are really important
print confusion_matrix(Y_test, Y_pred)

print precision_score(Y_test, Y_pred, average='macro')
print recall_score(Y_test, Y_pred, average='macro')
```

```
[[ 535  171]
 [ 168 1149]]
0.8157393637656796
0.8151138629514671
```

We have a high recall and precision score, which is really nice. In fact, our data is not unbalanced indeed:



These are the feature importances. Feature importances are of great interest because we can see where we have to do more feature engineering, which features can be removed easily without having an impact on accuracy(but making the training phase faster).


Feature Importances
media teze nationale
media de absolvire
nota la matematica
nota la limba romana
specializare
optional8 nota
scoala
judet8
limba materna8
promotie anterioara
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Relative Importance

Appendix A

Your original code

app:code This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained. Including in this section any line of code taken from someone else leads to failure of IS class this year. Failing or forgetting to add your code in this appendix leads to grade 1. Don't remove the above lines.

Appendix B

Quick technical guide for running your project

Requirements

Step by step technical manual

Appendix C

Check list

1. Your original code is included in the Appendix app:code.
2. Your original code and figures are readable.
3. All the references are added in the Bibliography section.
4. All your figures are referred in text (with command ref), described in the text, and they have relevant caption.
5. The final documentation describes only your project. Don't forget to remove all tutorial lines in the template (like these one).
6. The main algorithm of your tool is formalised in latex in chapter **??**.

Bibliography

[1] Admitere 2008-grades. http://web.archive.org/web/20081201094019/http://admitere.edu.ro/2008/staticRepI/j/.

[2] Bagrut grades in israeli high schools (2013-2016). https://www.kaggle.com/emachlev/bagrut-israel.

[3] Bayesian linear regression in python: Using machine learning to predict student grades. https://towardsdatascience.com/bayesian-linear-regression-in-python-using-machine-learning-to-predict-student-grades-part-1-7d0ad817fca5.

[4] Export html table to csv. https://jsfiddle.net/gengns/j1jm2tjx/.

[5] Heart disease uci. https://www.kaggle.com/ronitf/heart-disease-uci.

[6] Kaggle example. https://www.kaggle.com/triomni/heart-disease-prediction-by-a-newbie-to-newbies.

[7] Official Python website. https://www.python.org/downloads/.

[8] Plot the importances. https://stackoverflow.com/questions/17057139/how-to-find-key-trees-features-from-a-trained-random-forest.

[9] Predict the final grade of portugese high school students. https://www.kaggle.com/dipam7/student-grade-prediction.

[10] Rolisoft : Bacalaureat-data. https://github.com/RoliSoft/Bacalaureat-Data.

[11] Student performance data set. https://archive.ics.uci.edu/ml/datasets/student+performance.