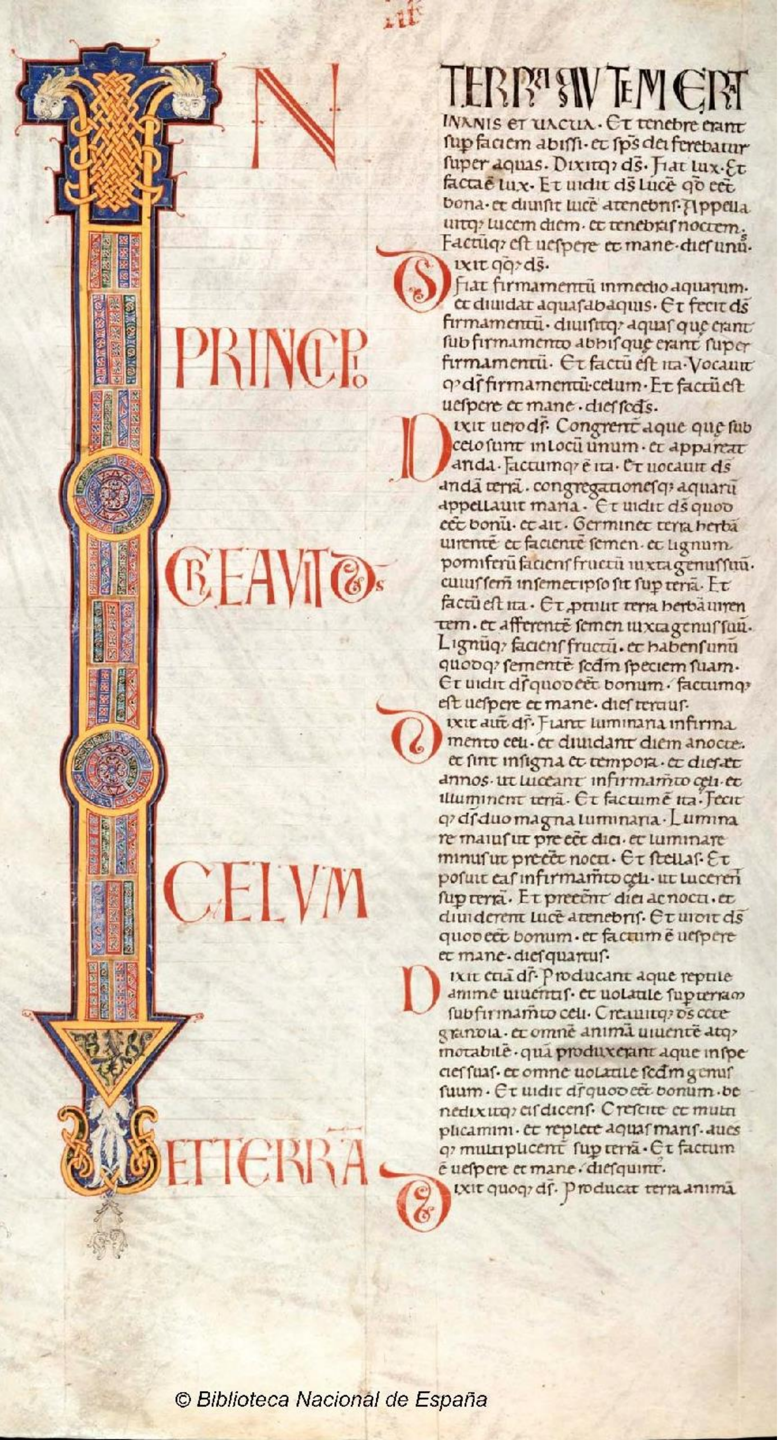
David Azoulay
Inès Nguyen
DIA1

Python For Data Analysis
Avila dataset



What we had to do

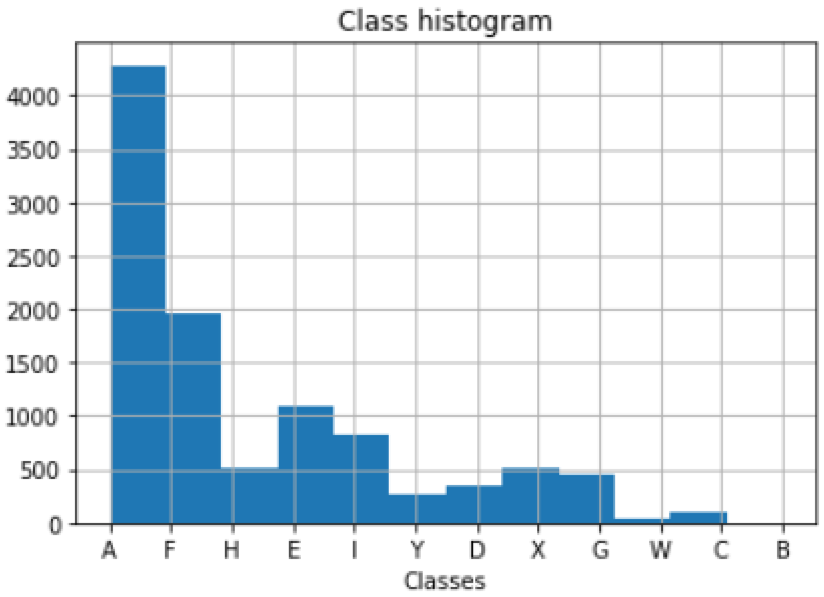
In this project, we had to explore our data set and classify some data

We were given the Avila data set which is a data from 800 images of the “Avila Bible” manuscript.

Our goal was to associate each pattern to one of the 12 copyists (labeled as: A, B, C, D, E, F, G, H, I, W, X, Y)

Exploration of the data

Repartition of the classes



CLASS DISTRIBUTION (training set)
A: 4286
B: 5
C: 103
D: 352
E: 1095
F: 1961
G: 446
H: 519
I: 831
W: 44
X: 522
Y: 266

We can see that the class A is overrepresent compared to classes W and B for example

Exploration of the data

Entrée [188]:
```
data.head()
```

Out[188]:

	intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ration	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing	Class
0	0.130292	0.870736	-3.210528	0.062493	0.261718	1.436060	1.465940	0.636203	0.282354	0.515587	A
1	-0.116585	0.069915	0.068476	-0.783147	0.261718	0.439463	-0.081827	-0.888236	-0.123005	0.582939	A
2	0.031541	0.297600	-3.210528	-0.583590	-0.721442	-0.307984	0.710932	1.051693	0.594169	-0.533994	A
3	0.229043	0.807926	-0.052442	0.082634	0.261718	0.148790	0.635431	0.051062	0.032902	-0.086652	F
4	0.117948	-0.220579	-3.210528	-1.623238	0.261718	-0.349509	0.257927	-0.385979	-0.247731	-0.331310	A

- All the attributes are decimal numbers, representing attributes of an image.
- Hard to do any supposition with such attributes, hard to say if there is any possible correlation between them.
- With these attributes all cardinals, we can’t do a lot of visualization

Exploration of the data

Entrée [189]:
```
data.describe(include = 'all')
```

Out[189]:

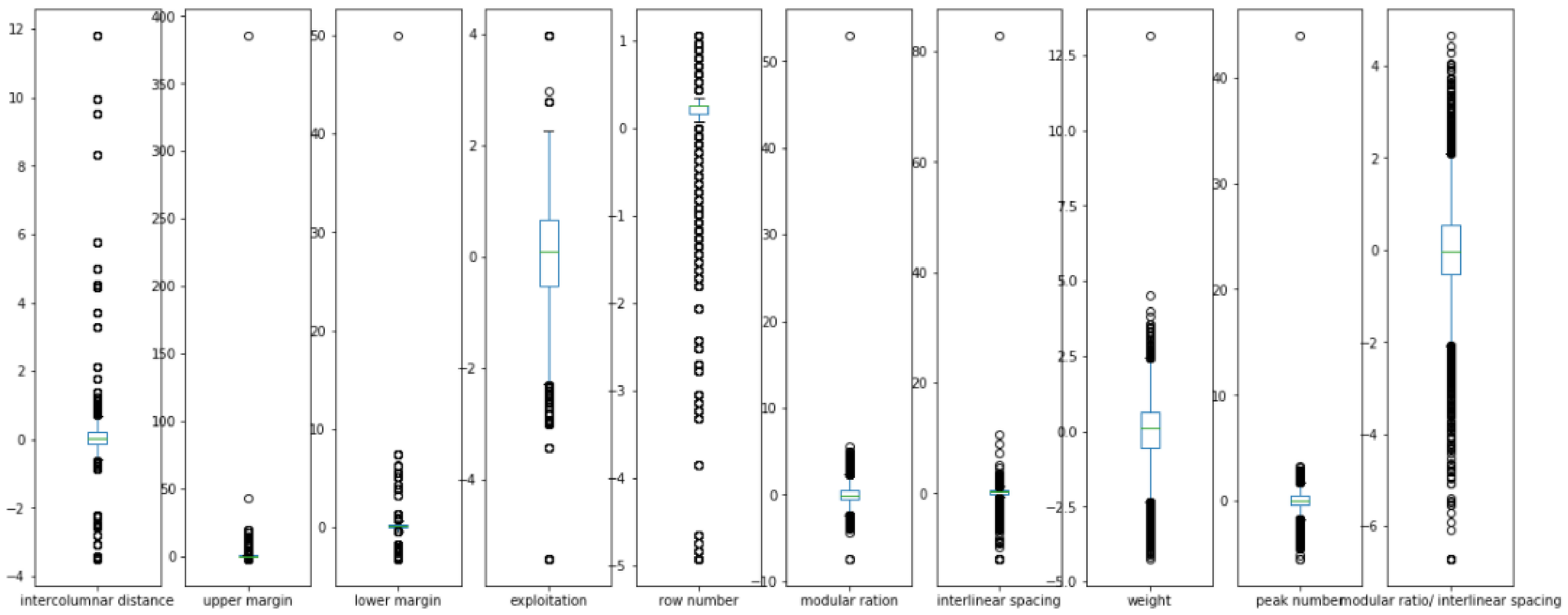
	intercolumnar distance	upper margin	lower margin	exploitation	row number	modular ration	interlinear spacing	weight	peak number	modular ratio/ interlinear spacing	Class
count	10429.000000	10429.000000	10429.000000	10429.000000	10429.000000	10429.000000	10429.000000	10429.000000	10429.000000	10429.000000	10429
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4285
mean	0.000827	0.033630	-0.000556	-0.002433	0.006354	0.013948	0.005570	0.010234	0.012891	0.000803	NaN
std	0.991475	3.921056	1.120251	1.008564	0.992100	1.126296	1.313812	1.003515	1.087715	1.007141	NaN
min	-3.498799	-2.426761	-3.210528	-5.440122	-4.922215	-7.450257	-11.935457	-4.247781	-5.486218	-6.719324	NaN
25%	-0.128929	-0.259834	0.064919	-0.528002	0.172340	-0.598658	-0.044076	-0.542001	-0.372457	-0.516103	NaN
50%	0.043885	-0.055704	0.217845	0.095763	0.261718	-0.058835	0.220177	0.111754	0.064084	-0.034621	NaN
75%	0.204355	0.203385	0.352988	0.658210	0.261718	0.564038	0.446679	0.654900	0.500624	0.530885	NaN
max	11.819916	386.000000	50.000000	3.987152	1.066121	53.000000	83.000000	13.173081	44.000000	4.671232	NaN

- With the characteristics of each attribute, it’s also hard to make any presumption.
- Some max values seem to be outliers considering min, mean and 3rd quartile like for upper margin (386) or lower margin (50) for exemple

Exploration of the data

Box Plot

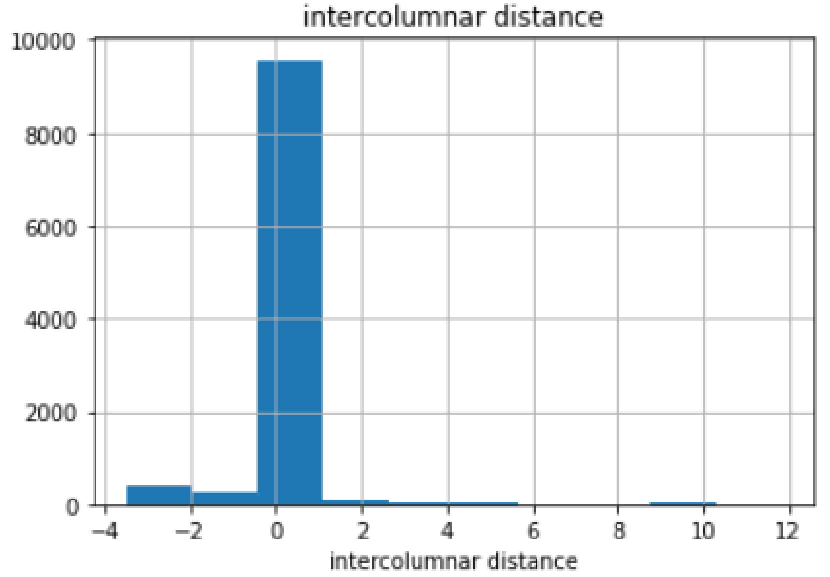
Entrée [191]:
```
#bos plot
data.plot(kind='box', subplots=True, figsize=(20,8));
```

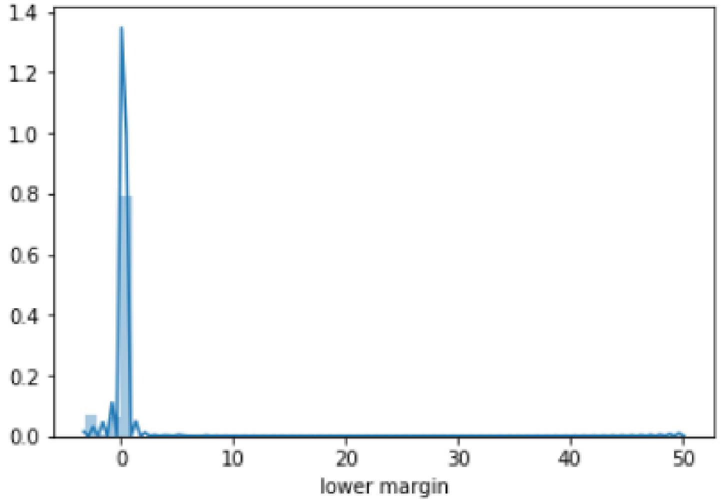


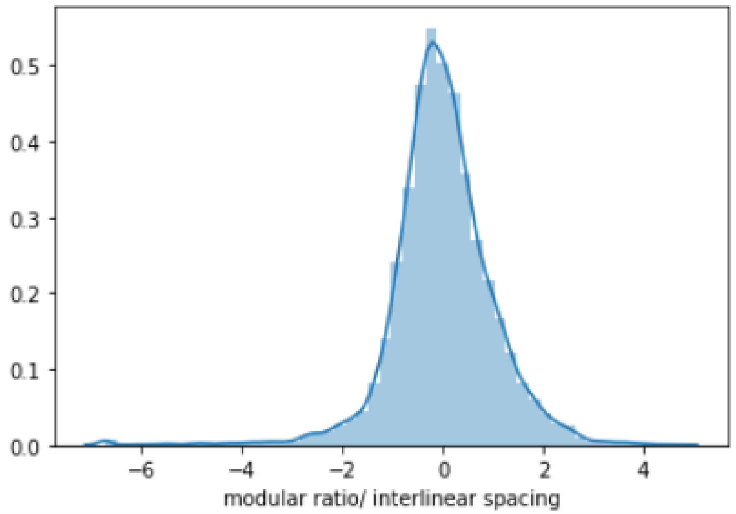
The box plots confirms that there is an outlier for some attributes like upper margin, lower margin, modular ratio, interlinear spacing or peak number for example, but with the amount of data we have, it should not have a big interference to determine our models

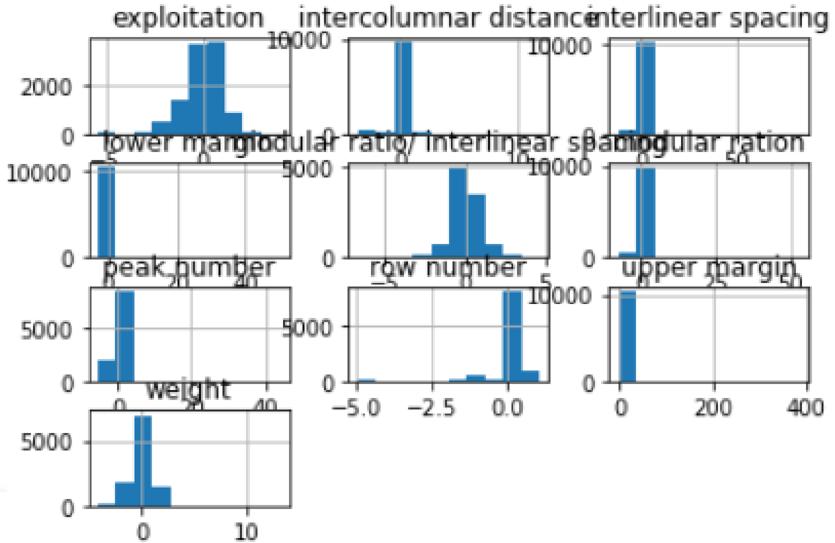
Exploration of the data

With the following histograms and plots representing the repartition for each attribute, the values seem very close, and this is sometimes due to the upper outlier which give us a biased scale. However, it will not impact our research to find the best model and do the prediction









Exploration of the data

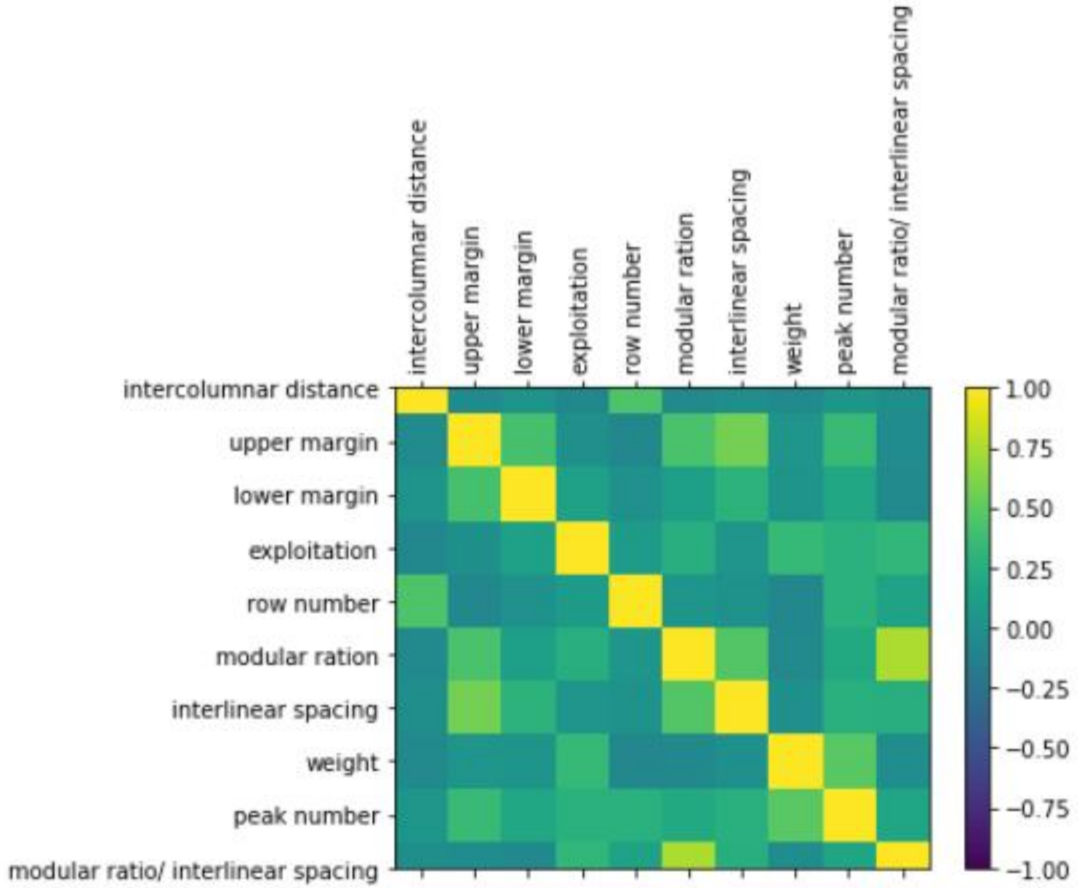
Entrée [194]:
```
#we check for missing values
data_null = data.copy()
data_null.isnull().sum()

# 1 correspond to 1 value missing
#we don't have any missing values because we only see 0
```

Out[194]:
```
intercolumnar distance                 0
upper margin                           0
lower margin                           0
exploitation                           0
row number                             0
modular ration                         0
interlinear spacing                    0
weight                                 0
peak number                            0
modular ratio/ interlinear spacing     0
Class                                  0
dtype: int64
```

- We can see that the training data is good quality because there is not any missing values

Exploration of the data

Correlation matrix:



- We can see that there is no correlation between the different attributes, with a correlation near to 0 each time (except for intercolumnar spacing/ upper margin).

Looking for the best model

5 models were adapted to a classification problem (Multi-Class classification):

- KNN
- Random forest
- Decisions trees
- Gaussian Naive Bayes
- Gradient boosting

Our method:

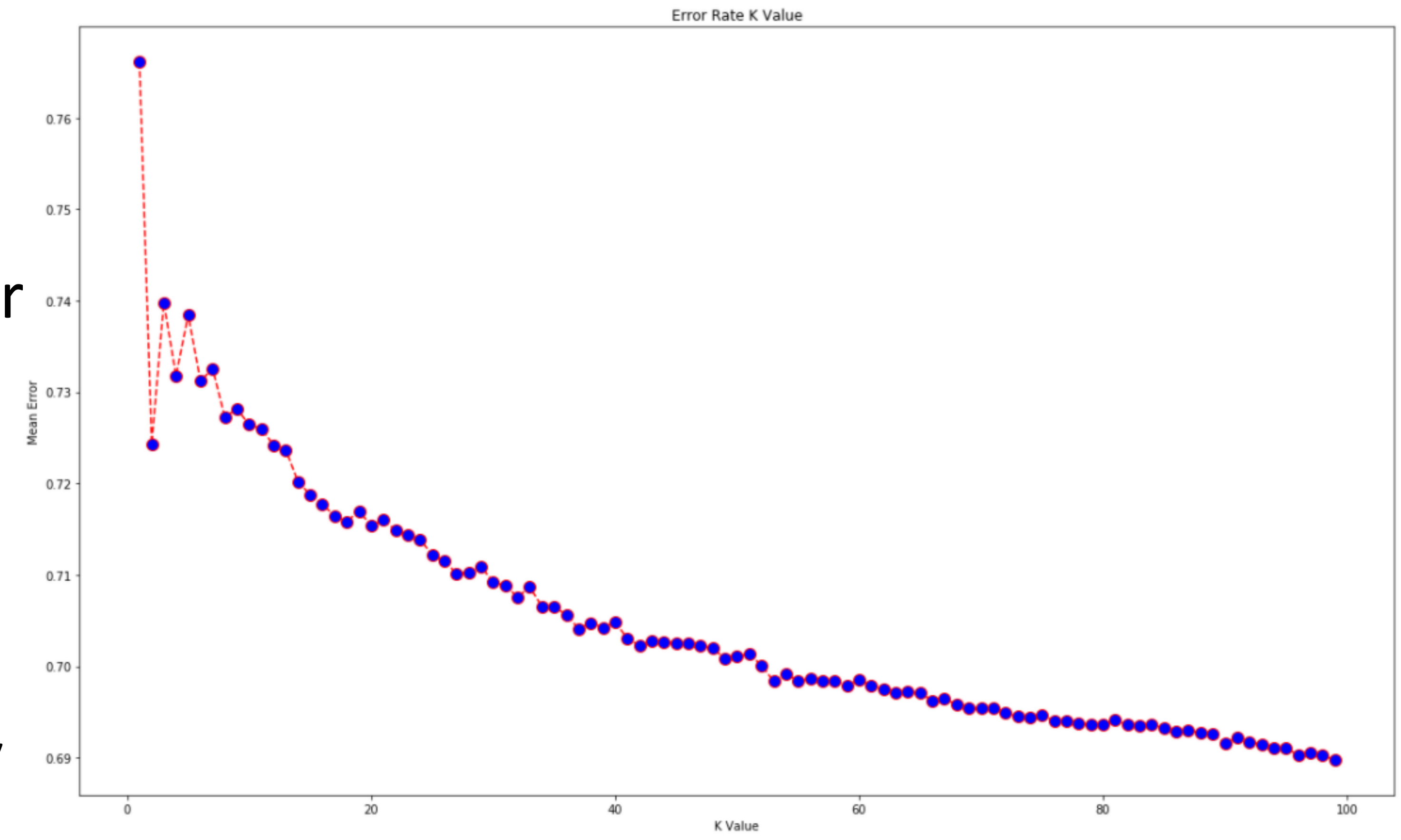
- Split our training data into a training set and a testing set (with a ratio 75/25%)
- Train our prediction models with the training set
- Apply models on our testing set
- Compare the model to find the best(s), use of cross-validation
- Boosting the best model by tunning hyperparameters thanks to our gridsearch
- Do the final prediction, save the results in a .txt file

Looking for the best model: KNN

We use the error rate K value to find the best K for our model.

We see that higher is K, better is the error K value but with no big difference after K around 20.

We obtain an accuracy of around 66% which is still very low.



Looking for the best model

For the 4 others models, we need to encode the Classes values to make it an integer instead of a string.

To do this, we use the Label Encoder and it works like this: A becomes 0, B become 1, etc.

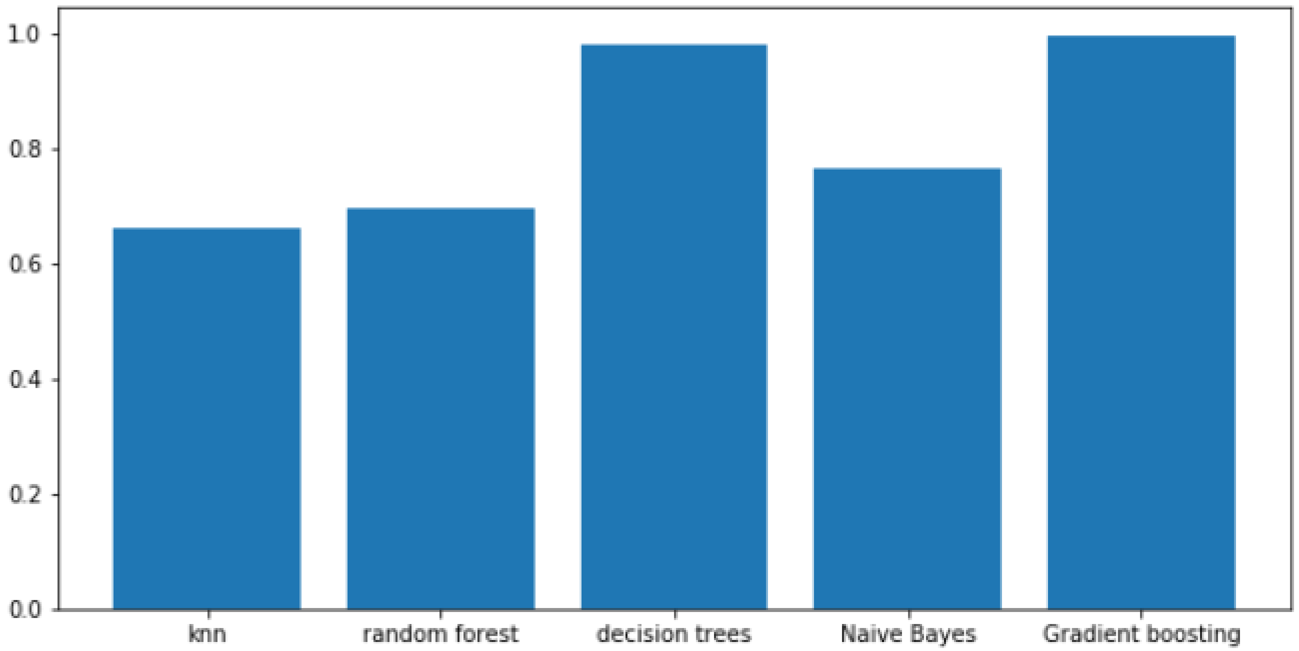
A	B	C	D	E	F	G	H	I	W	X	Y
⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕
0	1	2	3	4	5	6	7	8	9	10	11

We do the reverse operation once the prediction is done to have our labels back when we save our predictions in a new text file

Looking for the best model

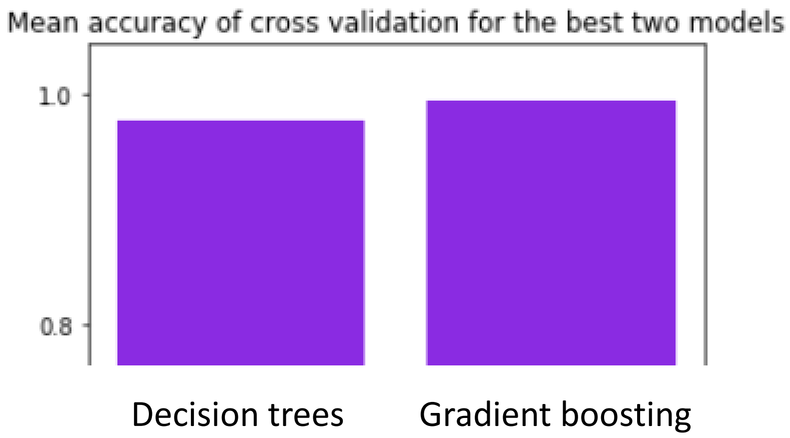
After the KNN method, we do the 4 following and we obtain these results :

- Random forest: 69.56%
- Decisions trees: 98.04%
- Gaussian Naive Bayes: 76.32%
- Gradient boosting: 99.46%



Looking for the best model

Decision trees and Gradient boosting method have both a high accuracy and are very close, so we do a cross-validation to be sure that the Gradient boosting is indeed the best model. We split the training set in 5 for the cross-validation



The Decision trees model obtain an accuracy of 97.6% with a standard deviation of 0.3%

The Gradient boosting model obtain an accuracy of 99.4% with a standard deviation of 0.2%

We can be sure that the gradient boosting model is the best for our dataset

Making the prediction

Now that we know that the Gradient boosting model is the best, we can use it to predict the classes on our test set.

Before making the prediction, we boost our model with the best hyperparameters. To do so, we first need to do a Grid search to know which hyperparameters are the best. However, doing this takes a lot of time so we need to concentrate only on few parameters. We obtain these parameters as the best: Using these parameters, we have an accuracy of 99.53% instead of 99.47%

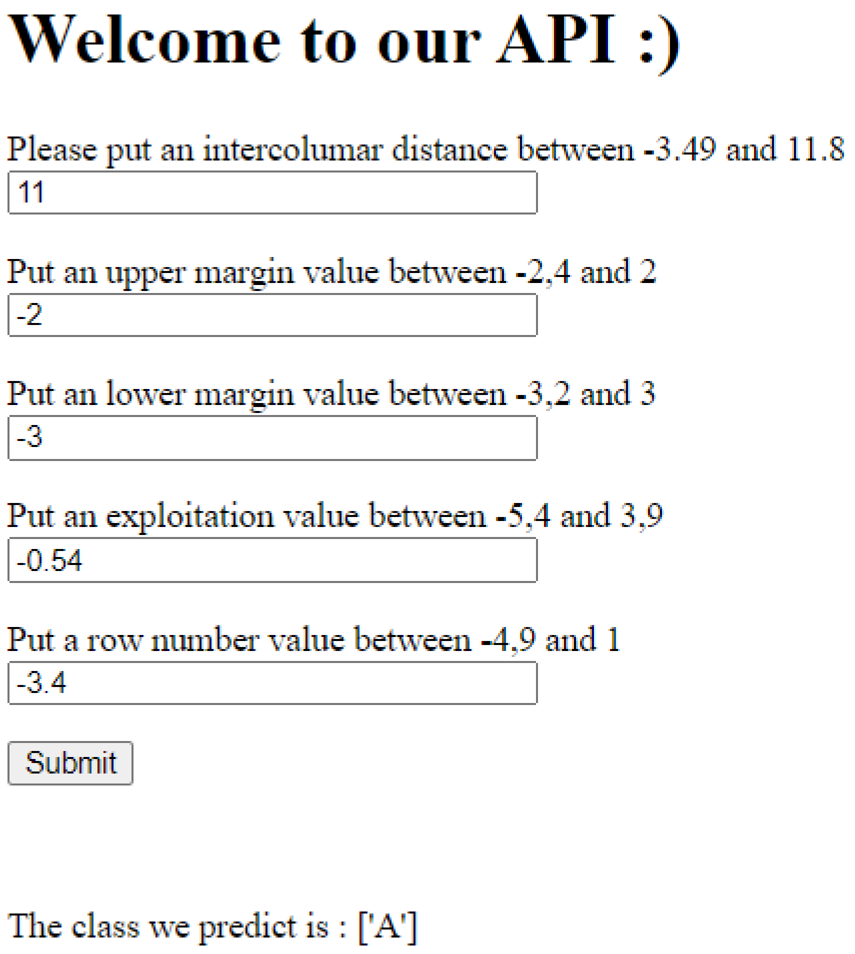
Since the testing set file provided contains the real classes, we can compare it to our prediction and we finally obtain an accuracy of 99.66% !

Then we save our predictions in a new text file!

The API

The goal of our API is to allow a real-time prediction when a user wants to predict to which class belongs an image.

As this is only an example, we ask the user only 5 of the 10 attributes because it's repetitive and it's only some random values for some normal people.



The API

To create our API, we used Flask.

Then, we wrote a python code that contains Flask API's and received the parameters through API calls. Thanks to a pickle, we applied our tunned Gradient Boosting model to those parameters and returned it.

The HTML file contains a template and allow the user to enter the parameters he wants.

```
from flask import Flask, render_templ
from wtforms import Form, TextField,
from wtforms.validators import DataRe
import numpy as np
from sklearn.ensemble import Gradient
import pickle
import os

# App config.
DEBUG = True
app = Flask(__name__)
app.config.from_object(__name__)
app.config['SECRET_KEY'] = '703841'
```

```
<title>Index1 Form Demo</title>

{% block content %}
    <h1>Welcome to our API  :) </h1>
    <form action="" method="post" novalidate>
        <p>
            {{ form.intercolumnar.label }}<br>
            {{ form.intercolumnar(size=32) }}
        </p>
        <p>
            {{ form.upperMargin.label }}<br>
            {{ form.upperMargin(size=32) }}
        </p>
        <p>
            {{ form.lowerMargin.label }}<br>
            {{ form.lowerMargin(size=32) }}
        </p>
        <p>
            {{ form.exploitation.label }}<br>
            {{ form.exploitation(size=32) }}
        </p>
        <p>
            {{ form.rowNumber.label }}<br>
            {{ form.rowNumber(size=32) }}
        </p>
        <p>{{ form.submit() }}</p>
    </form>
```