Python for Data Analysis

Block Classifier

Yanis RABIA | Dylan RAKOTOARIVELO

Table of contents

01
Introduction
Problematic
and Dataset

02
Visualisations
Correlation and
Dimensional reductions

03
Applications
Modeling and application
of PCA

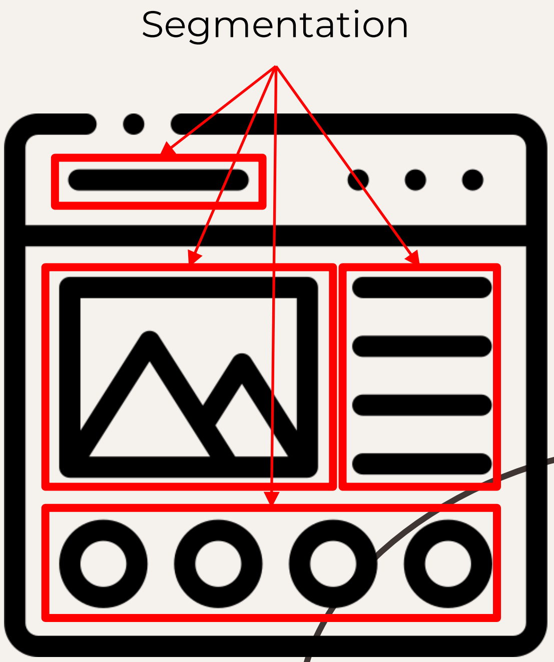
04
Conclusion
Final decision

01 - Introduction

In this project, we aim at **predicting**, according to the input data (**variables**), what the output (**target value**) will be.

In our case, the problem consists in **classifying** all the **blocks** of the page layout of a document that has been detected by a segmentation process, in order to separate text from graphic areas. Indeed, the five **classes** are :

- Text (1)
- Horizontal line (2)
- Picture (3)
- Vertical line (4)
- Graphic (5)



01 - Introduction

Thus, this segmentation has created a dataset of **5473 rows**, each with **10 variables**, more or less independent, and 1 column for the **labels** determining for such block, its class. The 10 variables are therefore the following:

- ***height*** : Height of the block
- ***length*** : Length of the block
- ***area*** : Area of the block (height * length)
- ***eccen*** : Eccentricity of the block (length / height)
- ***p_black*** : % of black pixels within the block (blackpix / area)
- ***p_and*** : % of black pixels after the application of the RLSA (blackand / area)
- ***mean_tr*** : Mean number of white-black transitions (blackpix / wb_trans)
- ***blackpix*** : Total number of black pixels in the original bitmap of the block
- ***blackand*** : Total number of black pixels in the bitmap of the block after RLSA*
- ***wb_trans*** : Number of white-black transitions in the original bitmap of the block

*RLSA : Run Length Smoothing Algorithm

02 - Visualisation

Description :

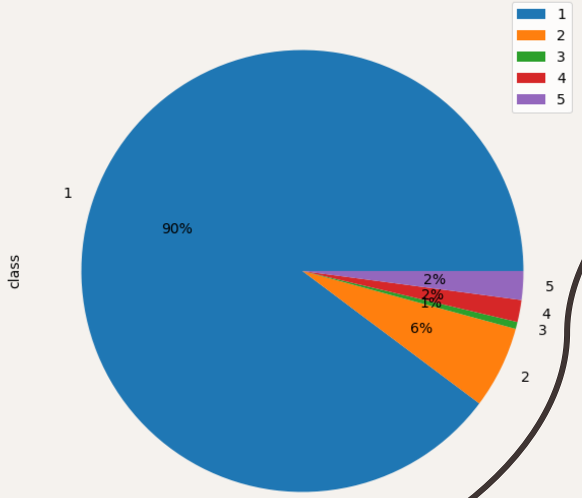
First, we wanted to do an overview of our dataset.

We started by using the **describe** method on our DataFrame. With that, we found that there is a wide range of values for some of these variables.

Then, we wanted to examine the repartition of our data between each class. For this, we did a **pie plot** and we found that **90%** of our dataset are class 1.

So it might be difficult for our model to distinguish between the classes from 2 to 5

	height	length	area	eccen	p_black	p_and	mean_tr	blackpix	blackand	wb_trans
count	5473.000000	5473.000000	5473.000000	5473.000000	5473.000000	5473.000000	5473.000000	5473.000000	5473.000000	5473.000000
mean	10.473232	89.568244	1198.405628	13.753977	0.368642	0.785053	6.219278	365.930751	741.108167	106.662891
std	18.960564	114.721758	4849.376950	30.703737	0.177757	0.170661	69.079021	1270.333082	1881.504302	167.308362
min	1.000000	1.000000	7.000000	0.007000	0.052000	0.062000	1.000000	7.000000	7.000000	1.000000
25%	7.000000	17.000000	114.000000	2.143000	0.261000	0.679000	1.610000	42.000000	95.000000	17.000000
50%	8.000000	41.000000	322.000000	5.167000	0.337000	0.803000	2.070000	108.000000	250.000000	49.000000
75%	10.000000	107.000000	980.000000	13.625000	0.426000	0.927000	3.000000	284.000000	718.000000	126.000000
max	804.000000	553.000000	143993.000000	537.000000	1.000000	1.000000	4955.000000	33017.000000	46133.000000	3212.000000



02 - Visualisation

Generally, the first step before the visualization, consists in performing a first **preprocessing** with the purpose of **digitizing** all the data of a dataset in order to manipulate them and represent them easily in graphs. That said, by chance, the data of our dataset are **all numerical**, so this step is to be neglected.

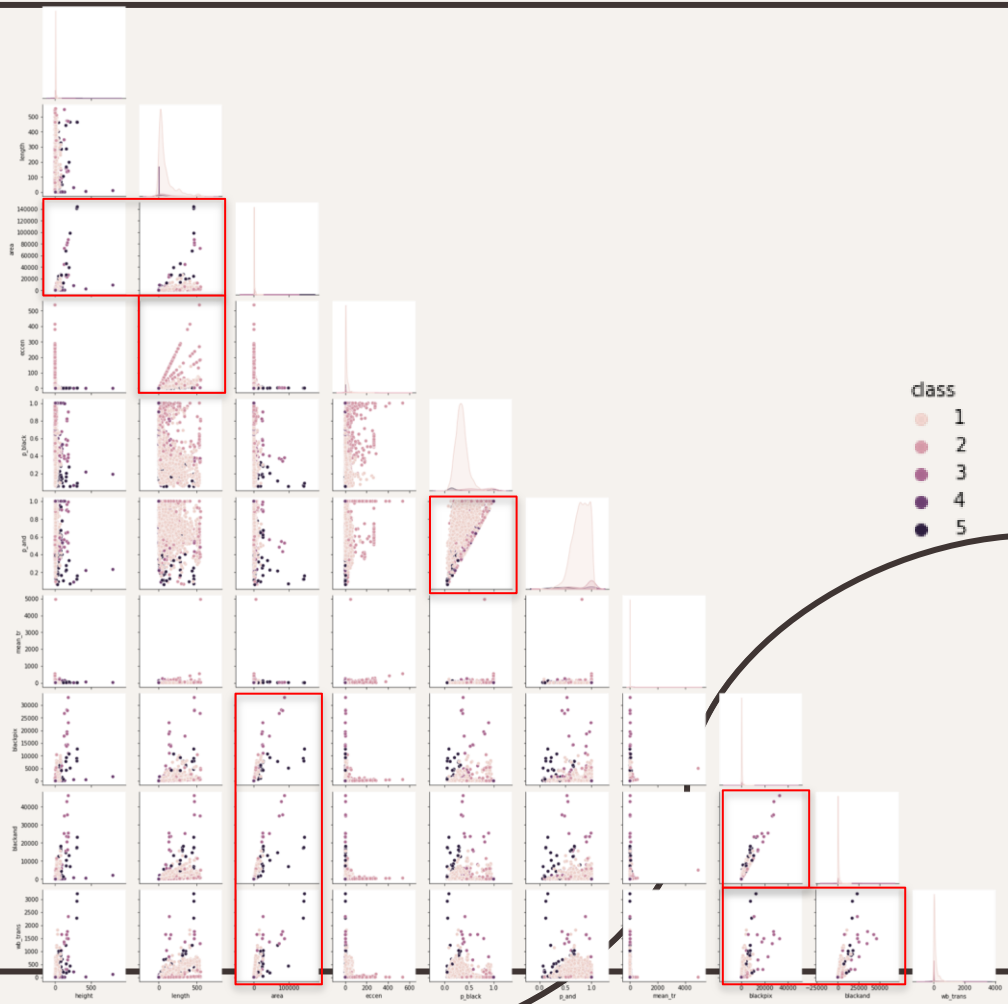
```
RangeIndex: 5473 entries, 0 to 5472
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   height    5473 non-null   int64
 1   length    5473 non-null   int64
 2   area      5473 non-null   int64
 3   eccen     5473 non-null   float64
 4   p_black   5473 non-null   float64
 5   p_and     5473 non-null   float64
 6   mean_tr   5473 non-null   float64
 7   blackpix  5473 non-null   int64
 8   blackand  5473 non-null   int64
 9   wb_trans  5473 non-null   int64
 10  class     5473 non-null   int64
dtypes: float64(4), int64(7)
memory usage: 470.5 KB
```

02 - Visualisation

Correlation between variables:

We started by looking for correlation between our columns. For visualising them, we used the **pairplot** function from **seaborn**

From that, we were able to observe several relationships that we framed on this visualisation using red squares

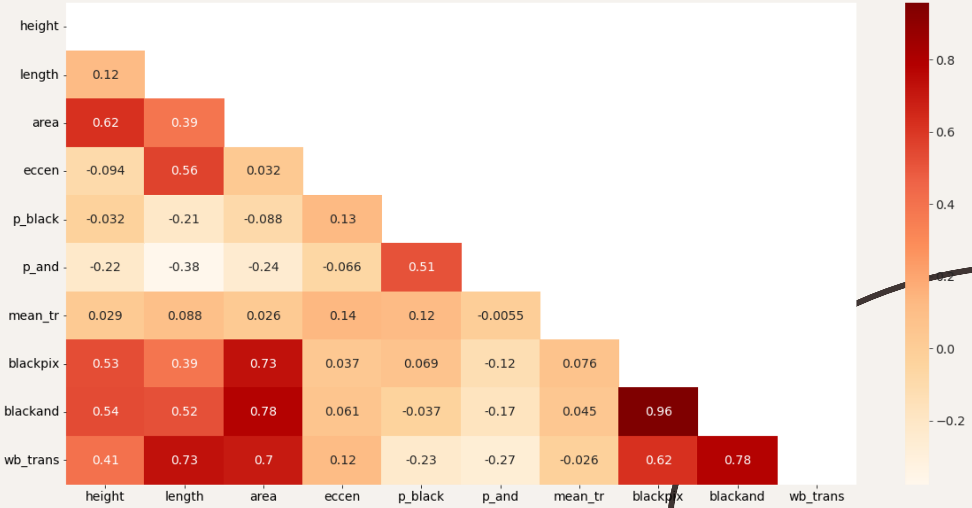


02 - Visualisation

<u>Correlation between variables:</u>

Then we plotted **the heatmap of the correlation matrix** to compare with our observations.

We found that most of our observations were right and we discovered more correlations between our columns

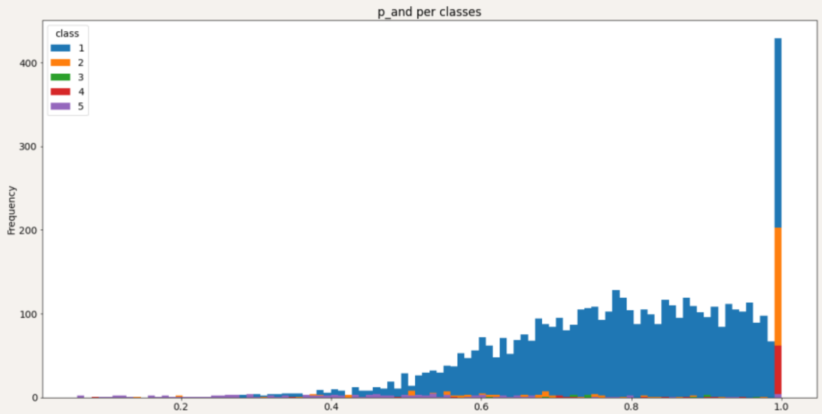


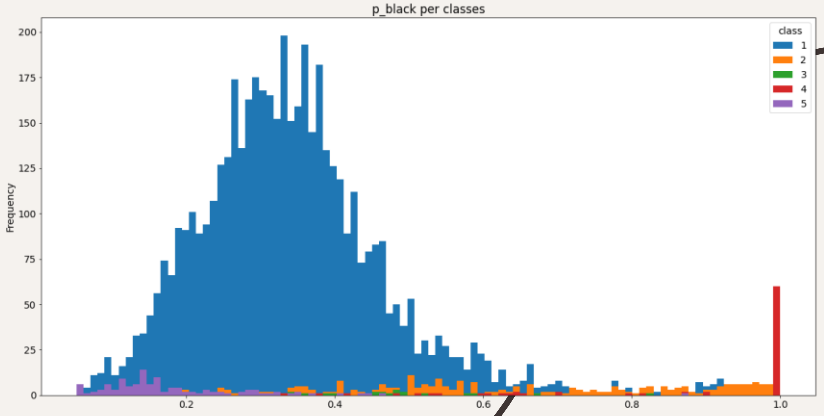
02 - Visualisation

Correlation between variables and classes:

We also tried to found correlations between our variables and the classes. So we plotted **histograms** of our columns using the classes as colors.

However, due to the high number of individuals in class 1 compare to the others, we couldn’t make any conclusion of this





02 - Visualisation

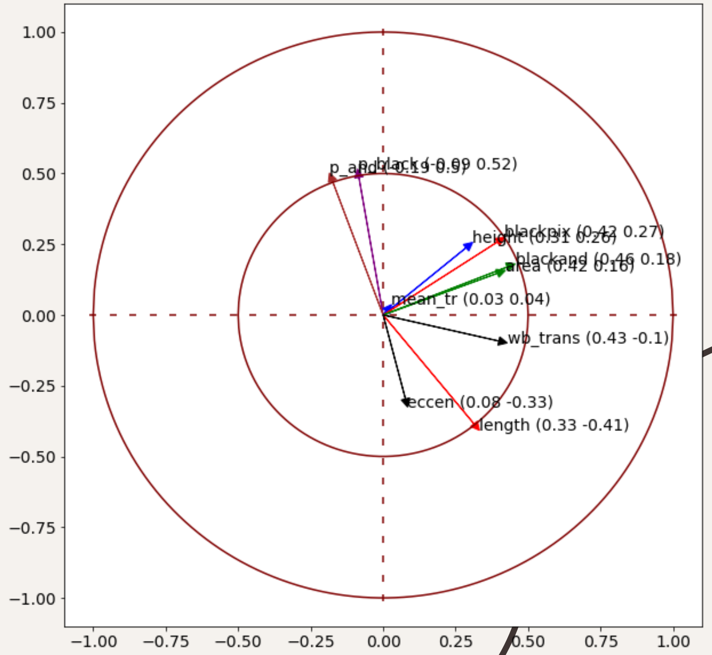
Dimensional reduction (PCA) :

To **optimize our model**, it is often advisable to do a **dimensional reduction** when we have a large number of variables, especially when many of them are strongly correlated. That said, removing variables also leads to **losing information** on a block, but can however **avoid redundancy**, i.e. the so-called "useless" columns. So, we used the ***sklearn*** library, to apply a PCA to our dataset.

02 - Visualisation

PCA 1 :

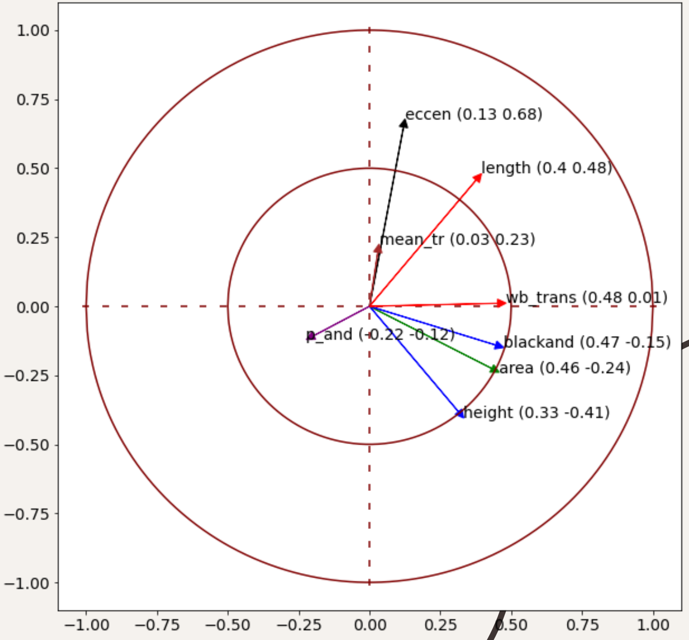
Here, this **biplot** includes all the variables of the initial dataset, and we can already state that many variables have **strong correlations** with others. We first chose to remove ***p_black*** and ***blackpix***, because they have a strong correlation with *p_and* and *blackand* respectively, and because significantly speaking, they **represent the same thing** (or even worse) given their definitions.



02 - Visualisation

<u>PCA 2 :</u>

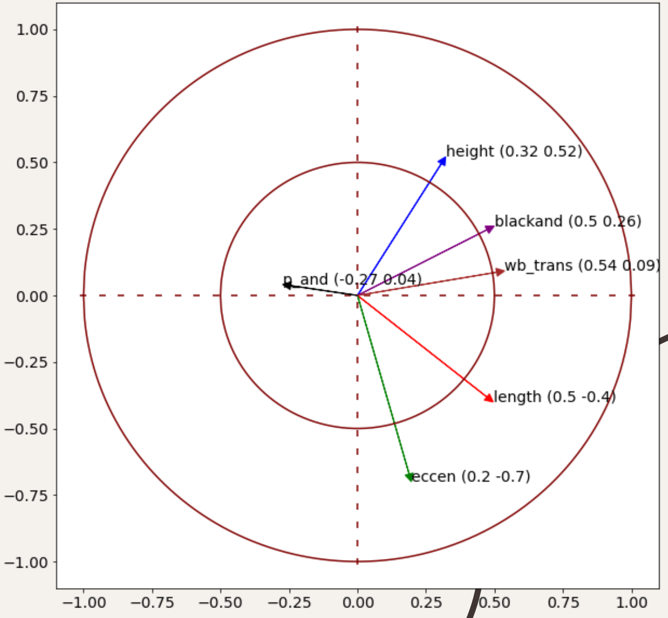
Once *p_black* and *blackpix* are removed, we have this **biplot**. From this, we decided for our second dimensional reduction, to remove ***area*** and ***mean_tr*** because of their strong correlations with *blackand* and *eccen* respectively.



02 - Visualisation

PCA 3 :

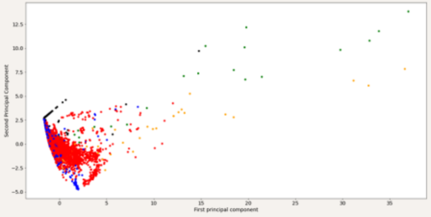
Finally, once *area* and *mean_tr* are removed in addition to *p_black* and *blackpix*, we have this **biplot**, whose variables seem more or less well **decorrelated**. However, we decided to perform a final dimensional reduction but by replacing ***height*** and ***length*** by ***area*** and removing ***eccen*** since this variable had a strong correlation with *area*.



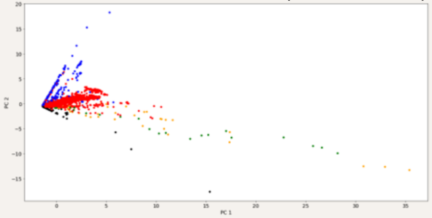
02 - Visualisation

Plot of Principal Components:

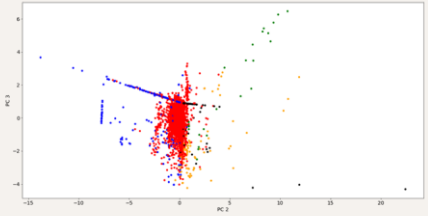
PCA on full dataset (PC1 vs PC2)



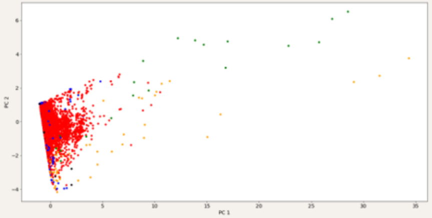
PCA on 1st reduction (PC1 vs PC2)



PCA on 2nd reduction (PC1 vs PC2)



PCA on 3rd reduction (PC1 vs PC2)



To finish with the dimensional reduction, here are the **plots** of the first 2 **principal components** of each of reductions, including the whole dataset too. However, as we can see, it is **difficult to conclude** anything since they seem rather disproportionate, so not very reliable.

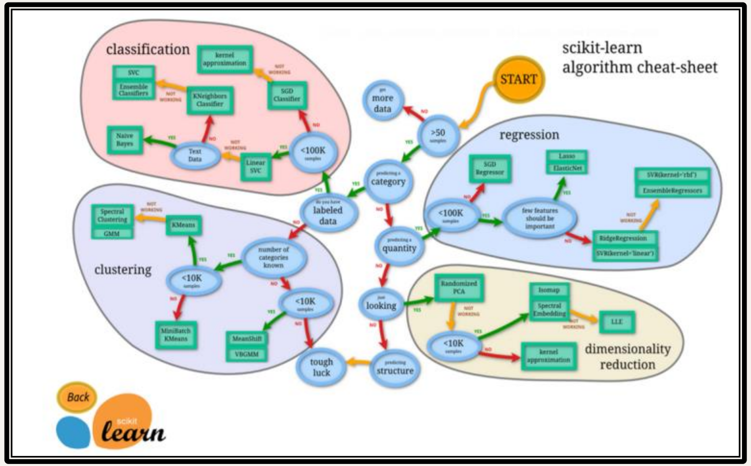
02 - Visualisation

Conclusion des PCA :

Thus, we don't really know if our dimensional reductions are **efficient**/**conclusive** or not, since we generally **lose informations** by removing variables. We have therefore decided, in a first step, to **apply models on the whole dataset**, then in a second step, **on the datasets reduced** by our PCA. We will then compare our various approaches, to know if the modeling will be more effective with or without dimensional reduction.

03 - Applications

Our dataset, having **labels**, it seemed logical to us to start on a **supervised learning** of **classification**. For this, we used the ***sklearn*** library again, which contains a variety of models in addition to the dimensional reduction. Its detailed structure is as follows:

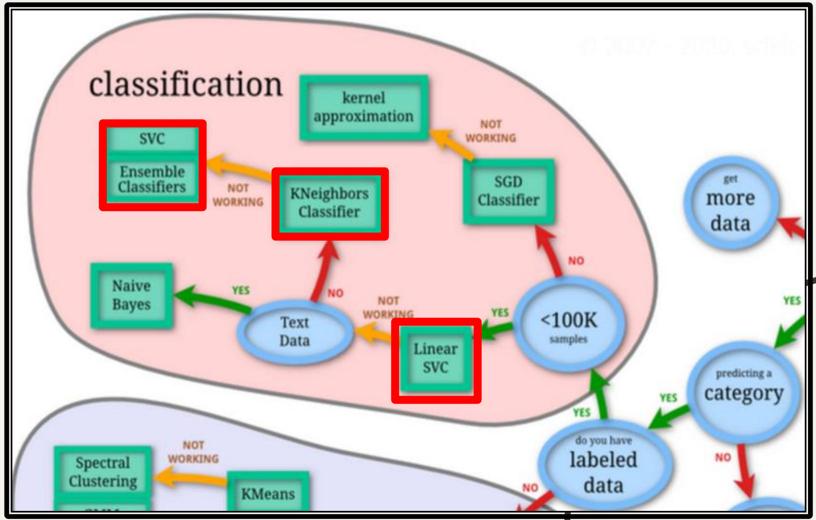


03 - Applications

So, depending on the context of our topic, we leaned towards the boxed models on the right, which are :

- **SVC**
- **KNeighbors Classifier**
- **Ensemble Classifiers**

Moreover, for each of the realized models, we applied a ***GridSearch*** function, allowing to ensure the **optimization of the hyperparameters** of the models, and thus to ensure the most optimal comparisons, thereafter, to choose the **most appropriate model**.



03 - Applications

Confusion Matrix SVC :

True \ Pred	Text	H Line	Picture	V Line	Graphic
Text	1596	8	2	4	2
H Line	27	89		2	
Pricture	8		4		
V Line	4			24	
Graphic	22			1	18

GridSearch results : C=0.5, decision_fuction_shape='ovo', degree=2,kernel='linear', shrinking=False

Precision Score : 95.57% → Too much false *Text* predictions

03 - Applications

Confusion Matrix KNeighbors :

True \ Pred	Text	H Line	Picture	V Line	Graphic
Text	1590	9	3	3	7
H Line	19	95		2	2
Pricture			8		
V Line	4			24	
Graphic	16	1		1	23

GridSearch results : metric='manhattan', n_neighbors=4, weights='distance'

Precision Score : 96.29% → All *pictures* are all well predicted

03 - Applications

Confusion Matrix Ensemble 1 (Bagging) :

True \ Pred	Text	H Line	Picture	V Line	Graphic
Text	1593	6	2	6	5
H Line	15	98		3	2
Pricture			8		
V Line	1			27	
Graphic	17			2	22

GridSearch results : bootstrap=False, max_features=5, n_estimators=25

Precision Score : 96.73% → All *pictures* are all well predicted and better predictions for *Horizontal / Vertical Lines*

03 - Applications

Confusion Matrix Ensemble 2 (Random Forest) :

True \ Pred	Text	H Line	Picture	V Line	Graphic
Text	1593	7	2	4	6
H Line	14	99		3	2
Pricture			8		
V Line	1			27	
Graphic	14	1		1	25

GridSearch results : criterion='entropy', max_features='sqrt', min_weight_fraction_leaf=0, n_estimators=150

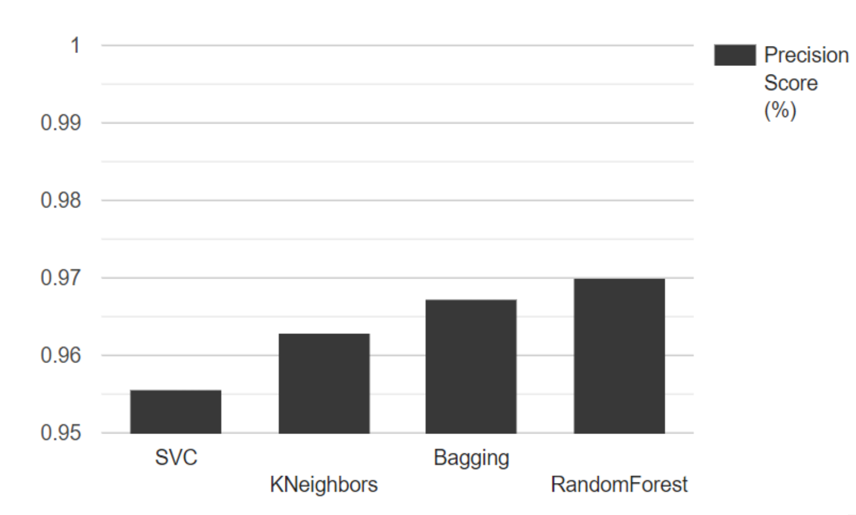
Precision Score : 96.95% → Same as Bagging but even better

03 - Applications

Model Comparaison :

Finally, we compared our results, and concluded that the **Random Forest** was the most adapted to our dataset since this model generated the **best precision score** (up to 96.95%).

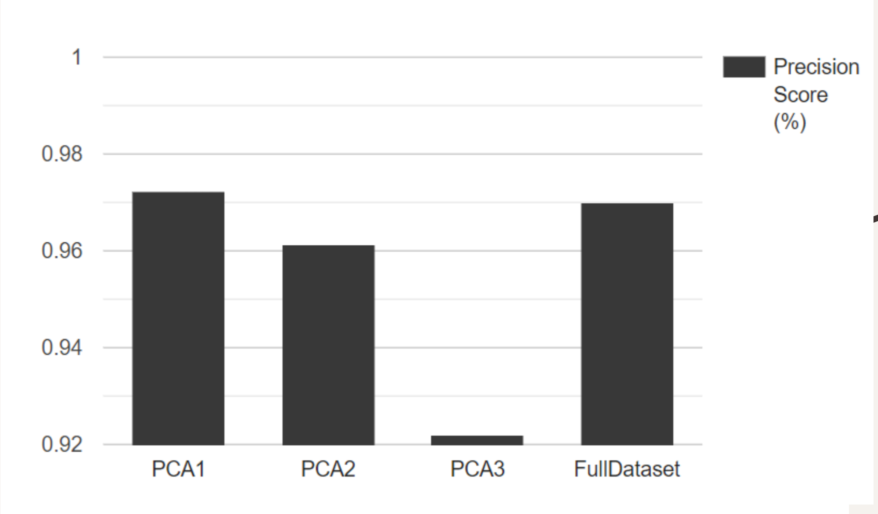
Thus, as expected, we used this model to **apply it to dimensionally reduced datasets** (with GridSearch).



03 - Applications

<u>Dimensional Reduction ?</u>

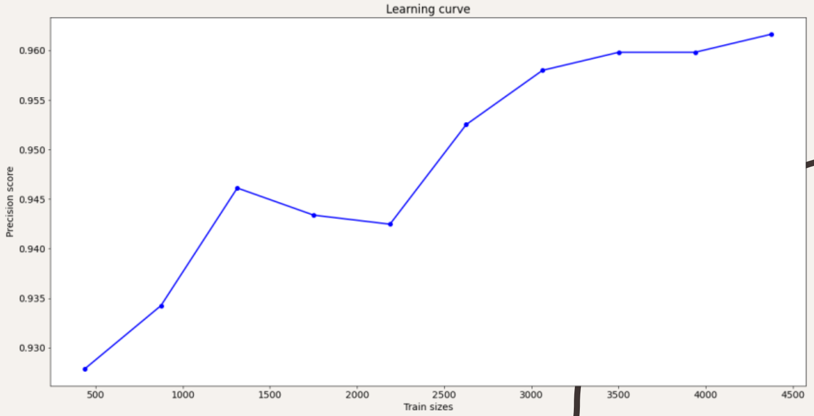
Some of the datasets generated by our dimensional reductions are **much less efficient** than the initial dataset. However, the one generated by **PCA1** (without *p_black* and *blackpixel*), gave **the best result** (up to 97.23%), so we chose to look at this data to fit our final model.



03 - Applications

Learning curve

The purpose of this curve is to show us whether we should **limit** our dataset to a **maximum number of samples** in order to make our model fit while preserving an optimal performance. However, our learning curve states that the performance of our model increases when the number of samples increases too (it **does not stabilize**). Therefore, we must use the **whole dataset** to fit our model.



04 - Conclusion

To conclude, among our 4 models, the **Random Forest** was the most accurate, and concerning our dimensional reductions we have chosen to fit our model from the dataset generated by the **PCA1**.

Despite the fact that our data is very **disproportionate** (~ 89.8% of texts), we have created, from our steps and our final dataset, a model with a satisfying precision score up to :

97,23 %