Project Python 2023-2024

Andreas Ball

Guillaume Awoukou

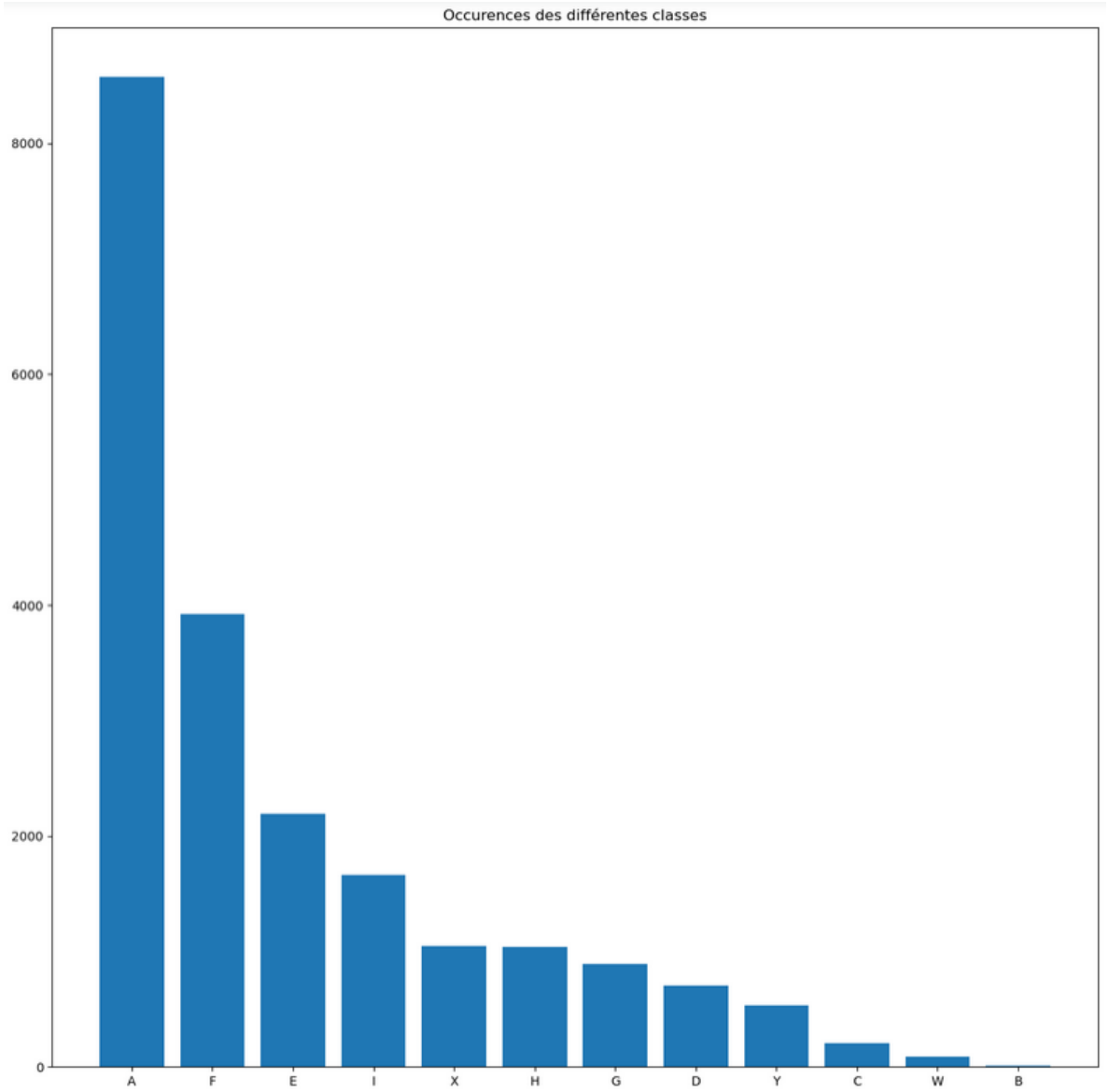
The avila bible dataset

Data :
- upper margin
- lower margin
- exploitation
- row number
- modular ratio
- interlinear spacing
- weight
- peak number
- modular ratio/ interlinear spacing

Target :
- Author

Data set context

- Using the data decribing the layout of the Avila Bible we would like to predict who wrote each page.
- The writers are here represented as letter.



Data preprocessing

- Our dataset was split into two, one for training and one for testing, but each had 50% of the data, which is not ideal, so we merge them. Later we will use two different approaches to split the data for training and testing.

- We also checked that no data was missing. None were, so we proceeded

10427	0.229043	-0.000745	0.171611	-0.002793	0.261718	0.688613
10428	-0.301743	0.352558	0.288973	1.638181	0.261718	0.688613
10429	-0.104241	-1.037102	0.388552	-1.099311	0.172340	-0.307984

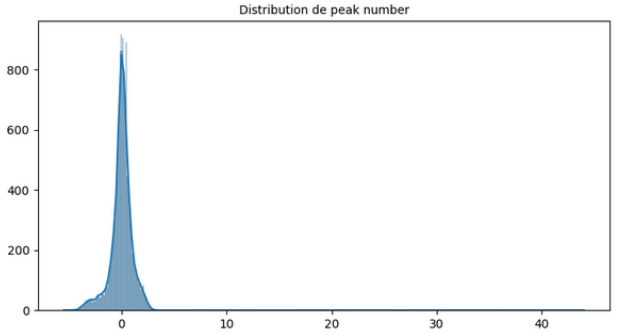
10430 rows × 11 columns

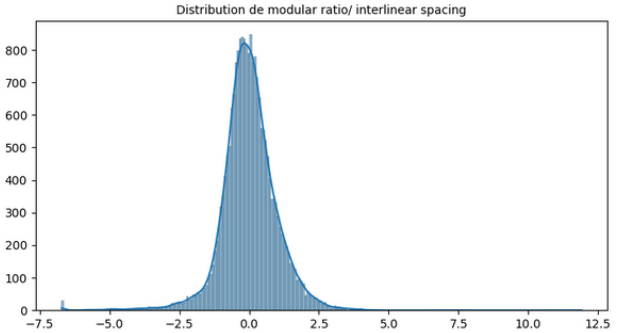
20857	0.229043	-0.000745	0.171611	-0.002793	0.261718	0.688613
20858	-0.301743	0.352558	0.288973	1.638181	0.261718	0.688613
20859	-0.104241	-1.037102	0.388552	-1.099311	0.172340	-0.307984

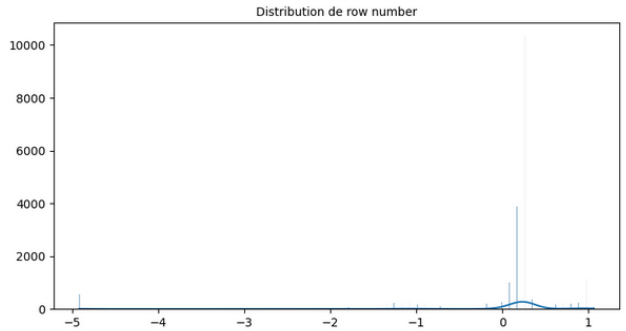
20860 rows × 11 columns

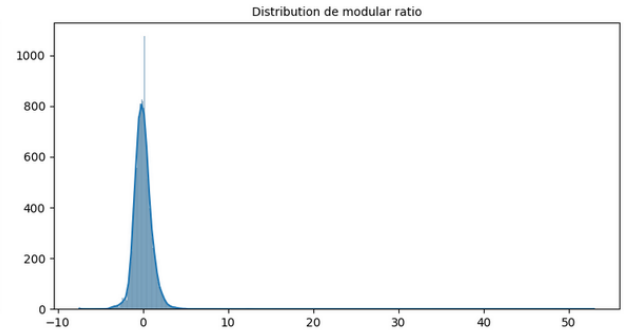
```
intercolumnar distance                0
upper margin                          0
lower margin                          0
exploitation                          0
row number                            0
modular ratio                         0
interlinear spacing                   0
weight                                0
peak number                           0
modular ratio/ interlinear spacing    0
class                                 0
dtype: int64
```

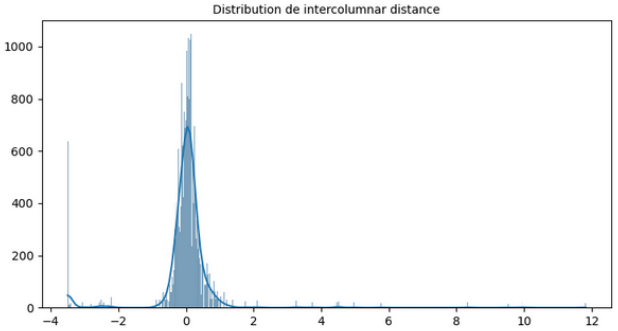
Data distribution

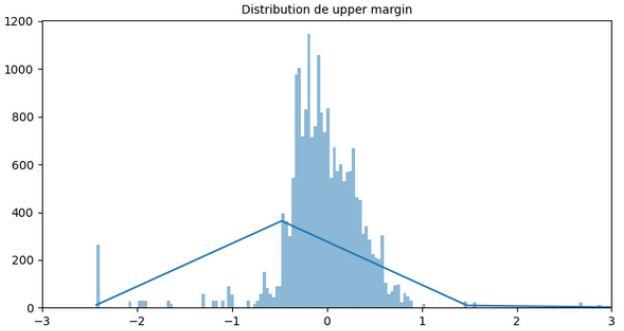


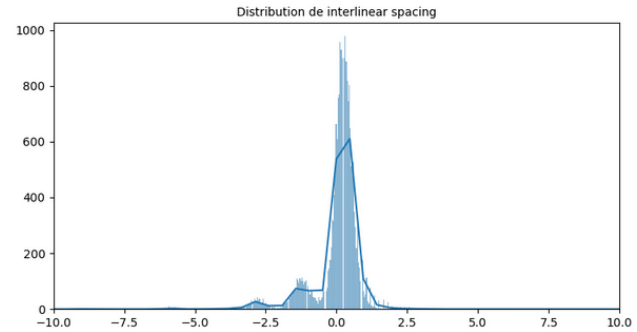


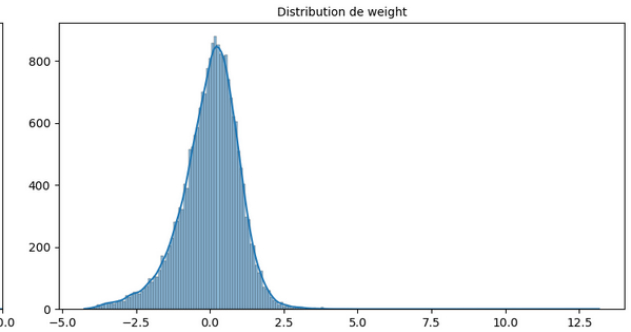


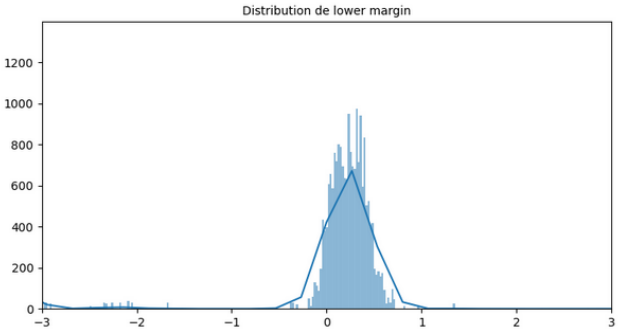


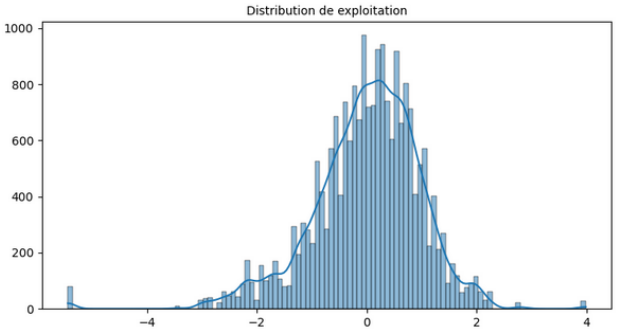


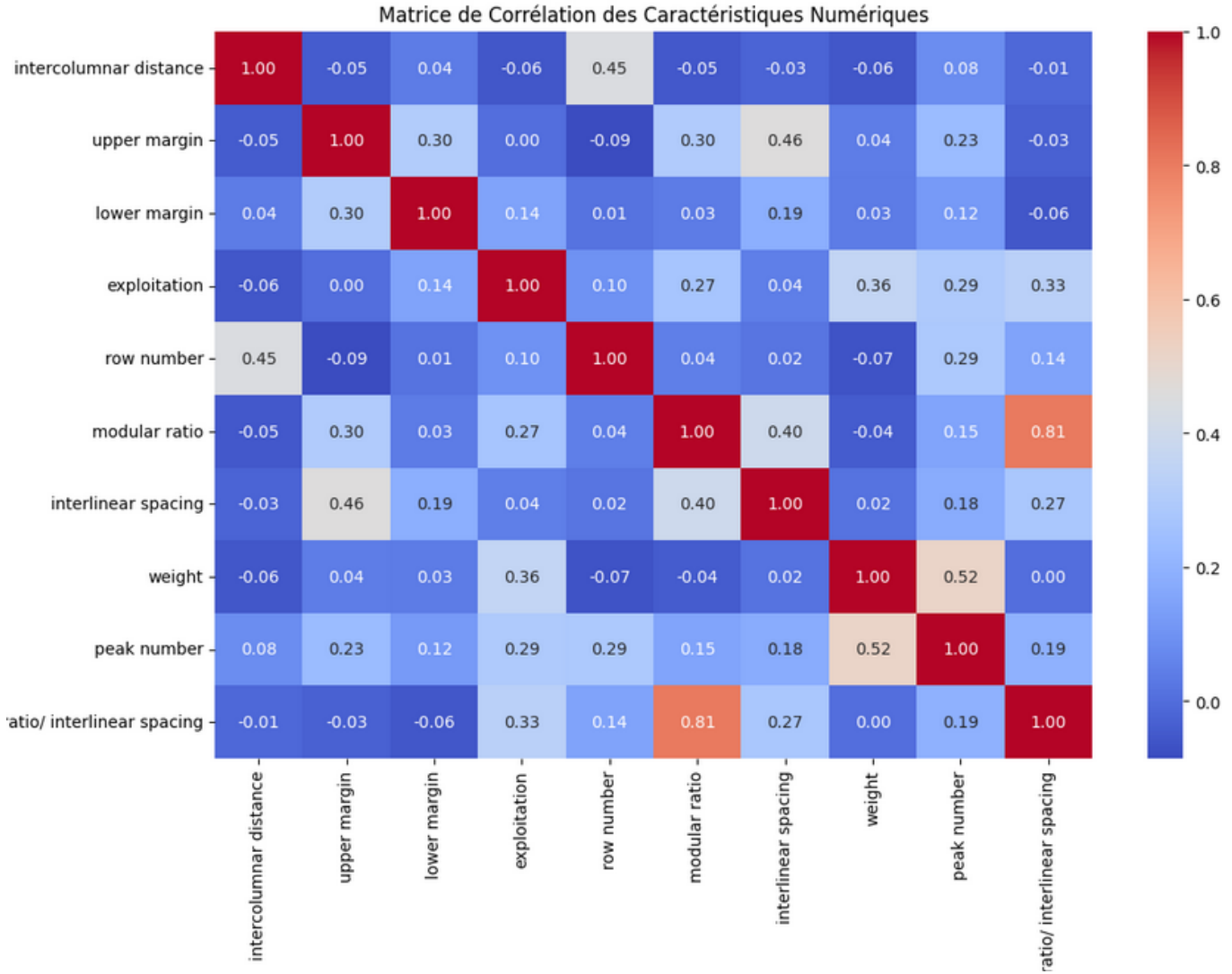












Data correlation

We found that the variable modular ratio/interlinear had a high correlation with other variables. This makes sense because it's just a ratio of two other variables we have in our data set. We then decided to drop it to avoid data redundancy.

Our Model

Multinomial Logistic Regression :
Perfect for multinomial classification cases. It models the probability of belonging to each class. Which is exactly what we are trying to do.

KNN :
None parametric that is useful to find natural group or tendancy inside dataset.

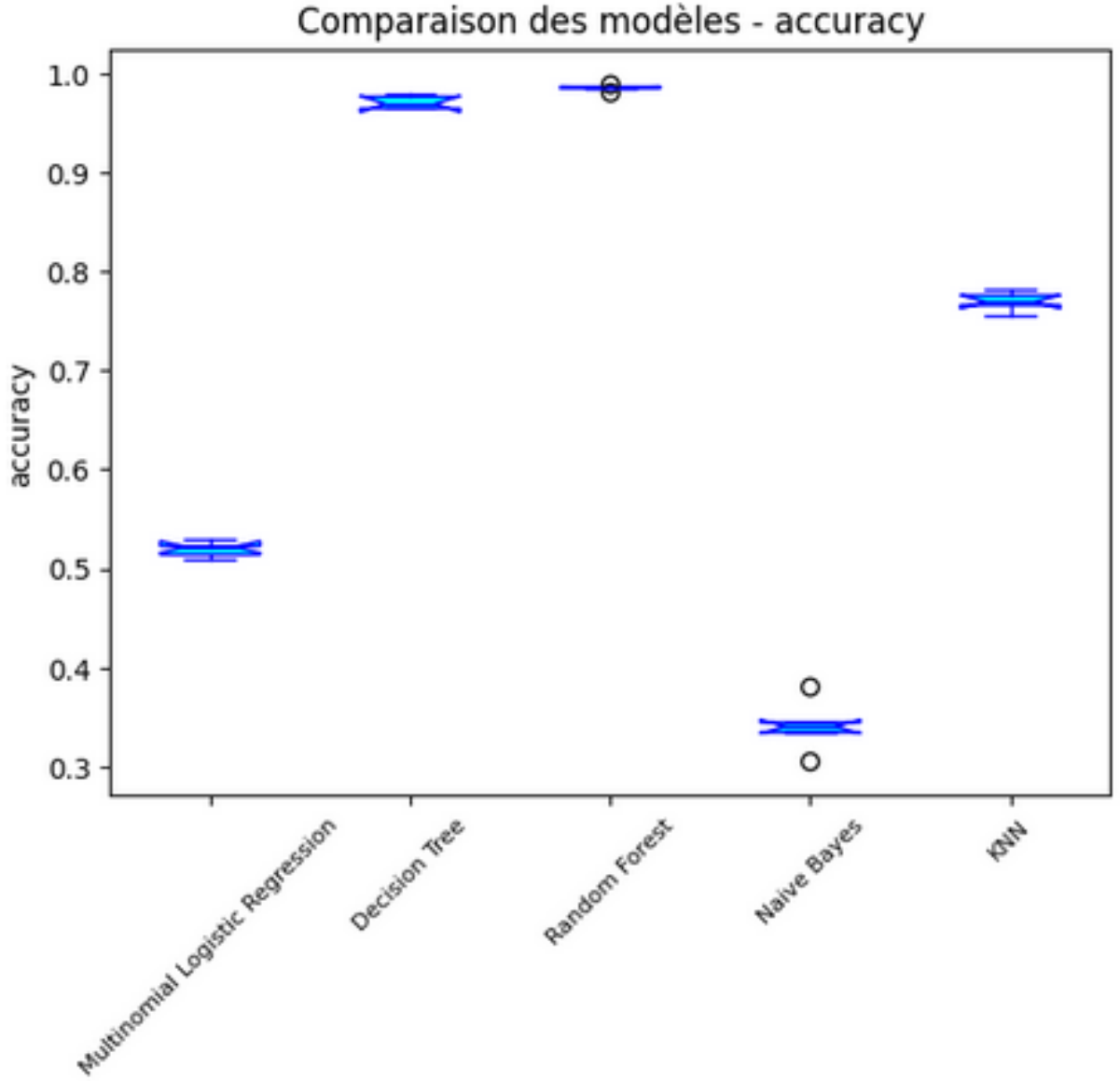
Naive Bayes :
Efficient et fast adapted to independant carateristic, which seems to be the case in your dataset.

Decision Tree :
Ideal for non linear relation. Easy to read and interpret. Should work on our data since it seems there is no linear relation.

Random Forest :
Upgrade from desision tree, reduce the risk of overadjustement et increase precision

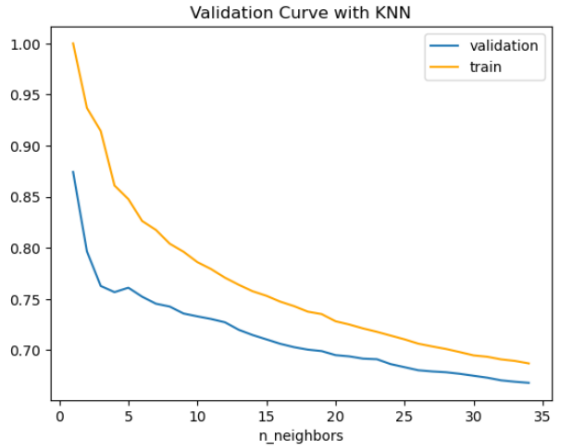
Model accuracy comparison

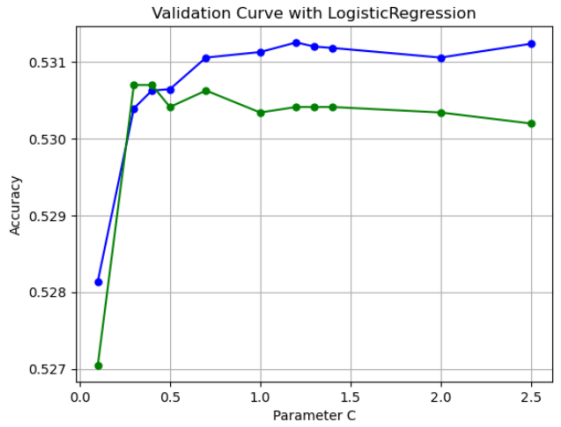
- We can clearly see difference of performance between the different model with the tree being clearly better than the other.
- Also, we can see that Naïve Bayes perform quite badly this could because of underfitting.

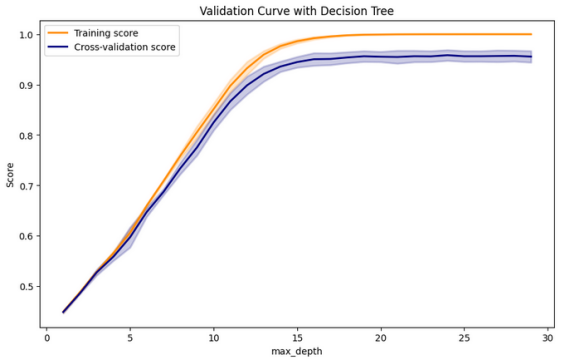


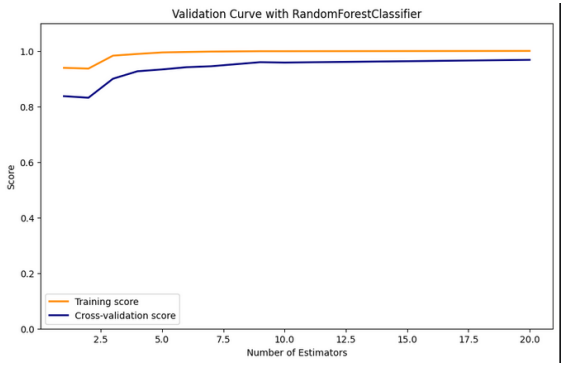
Validation Curve

- Here we are checking on the model that perform quite well if there is a case of overfitting.
- We observe that both tree model are overfitting at a certain depth.
- We can see that KNN overfit a lot at low number of neighbors, but it reduce significantly at higher number while retaining a good accuracy



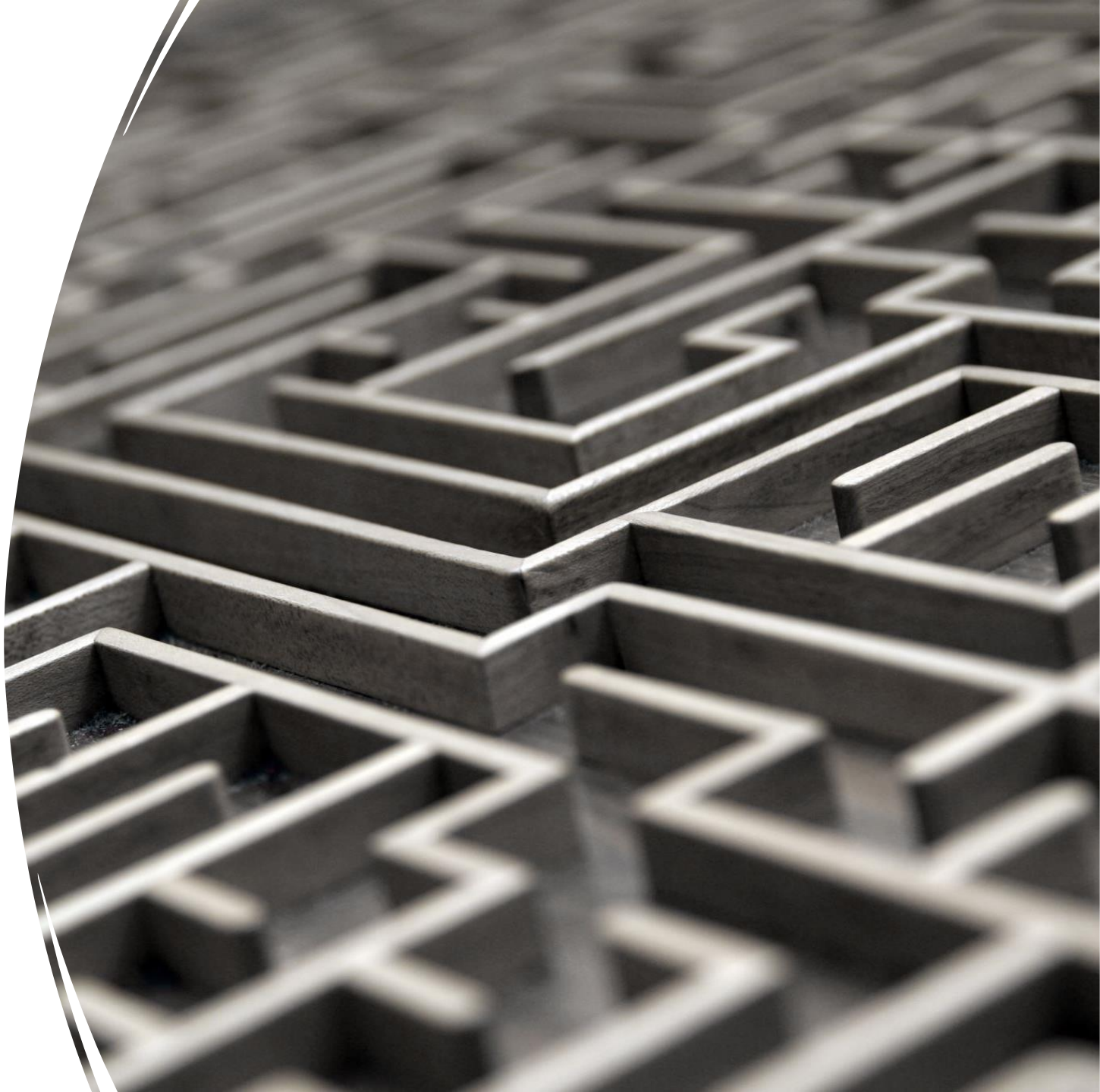






Model Optimization with Grid Search

- In our pursuit to fine-tune and enhance the performance of our machine learning models, we employed the Grid Search technique. Grid Search is a systematic approach to tuning hyperparameters, ensuring that we explore a wide range of possible configurations. This method involves defining a set of potential values for each hyperparameter and then exhaustively trying out all possible combinations of these values.
- The process can be computationally intensive, but it is instrumental in finding the most effective combination that maximizes our model's performance. By applying Grid Search to all our models, including Multinomial Logistic Regression, Decision Trees, Random Forest, Naive Bayes, and KNN, we were able to pinpoint the best hyperparameters for each.
- This methodical approach not only enhanced our models' accuracy but also provided deeper insights into how different hyperparameters impact model performance. In essence, Grid Search played a pivotal role in ensuring our models are robust, efficient, and finely tuned to our specific dataset, leading to more reliable and precise predictions



Optimizing Models with Grid Search - Key Results

In our pursuit of optimal model performance, we employed the Grid Search technique across various models. This approach systematically explored a multitude of hyperparameter combinations, ensuring a comprehensive search for the best model settings. Here, we highlight two notable results from our Grid Search:

Decision Tree Classifier:

Search Summary: Evaluated 108 different combinations over 5 folds, totaling 540 fits.

Best Parameters: Criterion: 'entropy', Max Depth: 30, Min Samples Leaf: 1, Min Samples Split: 2.

Best Accuracy: Achieved an impressive accuracy of 97.04%.

Random Forest Classifier:

Search Summary: Tested various configurations over 5 folds, amounting to 540 fits.

Best Parameters: Bootstrap: False, Max Depth: 30, Max Features: 'log2', Min Samples Leaf: 1, Min Samples Split: 2, Number of Estimators: 200.

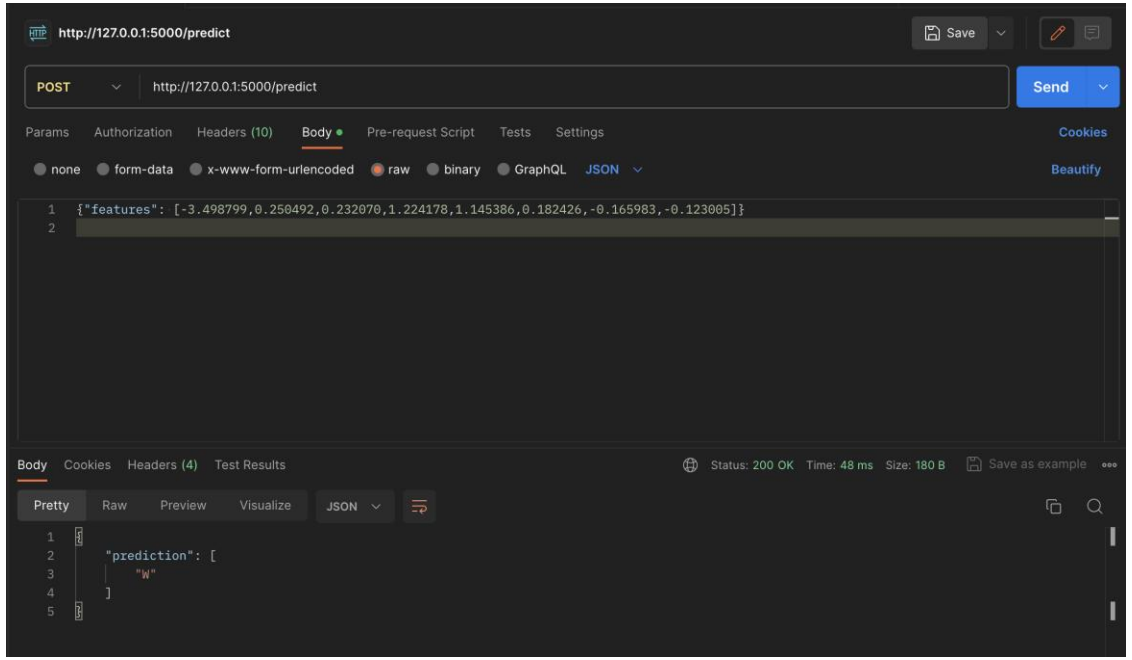
Best Accuracy: Attained a remarkable accuracy of 98.73%.

Transforming the Model into an API

- We made our optimized model accessible for practical use and real-time predictions, we transformed it into a web-based application programming interface (API)
- We chose Flask for its lightweight and straightforward functionality to transform our trained Random Forest model into a web API. Flask's simplicity allowed us to quickly set up a server that could take data inputs through HTTP requests and return the model's predictions. This approach made our model easily accessible and usable in real-world applications.

Test of our API

- To demonstrate the API's functionality, we co nducted a test using Postman, a popular tool for API testing. We selected a sample row from our test dataset and input the feature values into the request body of Postman. Upon sending the request to our Flask API, it successfully returned the correct prediction, showcasing the model's a bility to provide real-time insights. The following screen capture from Postman illustrates this process and the API's effective response to the input data



Conclusion and Future Directions

- In conclusion, our journey through the Avila Bible dataset analysis has been both enlightening and rewarding. We efficiently navigated the phases of data preprocessing and encoding, ensuring that our models received high-quality input for optimal performance. Our decisions at every step were informed and deliberate, guided by thorough research and a comprehensive understanding of the data and the tools at our disposal.
- Our models were fine-tuned to achieve maximum effectiveness, with a keen focus on avoiding overfitting. This careful balance ensured that our results were not only accurate but also truly reflective of the underlying patterns in the data.
- Looking Ahead: While we are satisfied with our current achievements, there are always avenues for further exploration. Future work could involve experimenting with more complex models, exploring deeper feature engineering techniques, or even expanding our dataset for a more extensive analysis. Another interesting direction could be the integration of our model into larger systems or applications, leveraging the API we developed to bring our insights into practical use.
- In essence, this project stands as a testament to the power of machine learning in uncovering hidden patterns and insights, even from the most intricate and historical datasets like the Avila Bible.