Cornier Alexandre
January 10th, 202

université de BORDEAUX

Classification of images with K-NN

Introduction

In this project, we are interested in the implementation of the k-Nearest Neighbor (K-NN) classifier algorithm in JavaScript, as a plugin for ImageJ. This is a supervised learning algorithm that can be used for both classification and regression in machine learning with a high level of accuracy according to the configuration.

The principle of this model consists in selecting the data closest to the point studied in order to predict its class. This method doesn't use any statistical model, it's "non-parametric" and finally it's based only on training data. That's why K-NN is called lazy learner because it does not learn anything in the training period, which makes it faster in comparison with algorithms that require long training. It's also easy to implement because there are only two parameters needed:

- The value of k corresponding to the k instances of the dataset closed to our observation
- The Euclidean distance corresponding to an interval separating two points in a standardized space

The main purpose is being able to predict the class of a symbol in an image, with a good accuracy, even with some degradation of this image.

2020

Cornier Alexandre
January 10th, 202

Plan

1 Material & Methods

To realize this project, the only material needed is a computer running an operating system compatible with ImageJ, to build and to run the K-NN algorithm.
The programming language chosen in this case is JavaScript because it allows to interact completely with ImageJ. This is a multi-platform, Open Source image processing and analysis software developed by the National Institutes of Health in 1971. It is written in Java and allows the addition of new features via plugins and macros.

First and foremost, it is important to update Java with the JDK v9.0.4 needed by the extension we will add to ImageJ. Also, it's better to update to the last version ImageJ in v1.52s to have a good support of Java 9 by the launcher. Then we need to install the DataFrame extension of ImageJ that is Nashorn_polyfill.js and Tml.js. Once all these steps have been completed, we can now focused in the development of the K-NN algorithm and its integration in ImageJ to try it on different types of measurements.

1.1 Methodology

Before any analysis, we have written a k-NN algorithm in JavaScript, with ImageJ interactions to facilitate the run using different parameters. The full code is provided in the appendix.

The first dataset to try are the various measurements of Shapes in this image. The corresponding labels are used to train and test the k-NN algorithm.

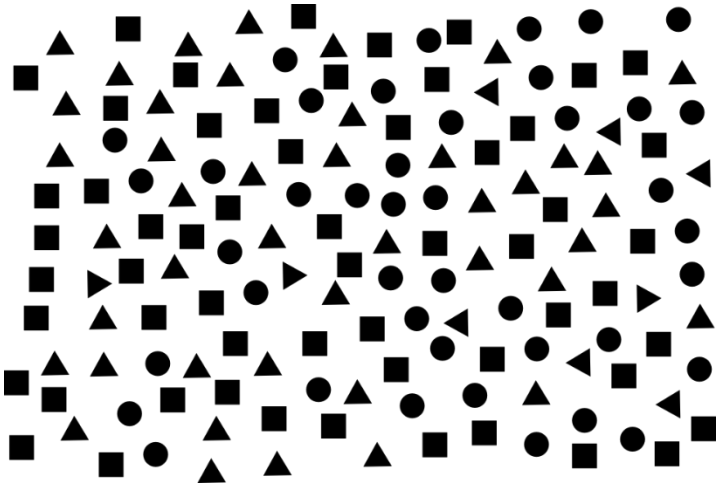


Figure 1 : Image with corresponding 151 shapes (Square, Triangle or Circle)

```
4,0,0,3,4,0,4,0,3,3,
4,3,3,0,4,3,3,4,3,4,
4,4,0,4,3,0,0,3,3,4,
0,4,0,3,0,0,4,4,4,3,
0,4,3,4,4,3,3,3,3,3,
0,0,3,3,0,3,0,4,3,0,
0,4,0,3,0,3,4,4,4,4,
0,3,3,4,4,3,3,4,4,4,
0,3,4,3,4,3,0,0,4,3,
0,3,0,3,4,3,4,0,4,3,
3,4,4,0,3,4,0,4,4,4,
0,0,4,3,3,0,3,3,3,0,
4,4,4,4,0,3,4,3,0,4,4,
3,0,0,4,0,4,3,3,4,4,
0,0,4,4,4,4,0,3,4,3,4,
3
```

Figure 2 : Labels corresponding to the figure 1

In this case, to check the accuracy of our classifier it is important to label each measurement ok shapes with its class, as a number for sake of convenience:

- The number **4** represents **square**
- The number **3** represents **triangle**
- The number **0** represents **circle**

Once our classifier is working on the first set of data, we can try it on more complex shapes in another image, and to start dealing with noise addition to compare the level of accuracy in different contexts. The second dataset used is a combination of circles:

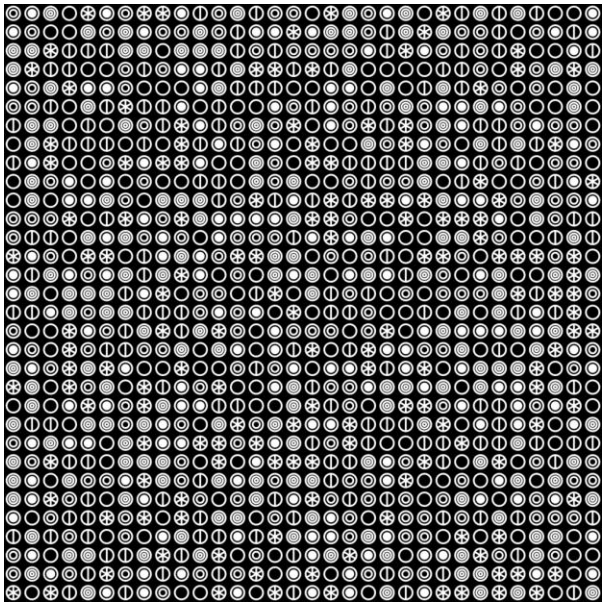


Figure 3: Images with corresponding 1024 circles

3, 5, 4, 6, 2, 5, 3, 2, 2, 5, 1, 3, 4, 1, 3, 3, 6, 2, 4, 3,
5, 4, 6, 2, 1, 4, 1, 4, 1, 6, 6, 5, 5, 3, 6, 6, 4, 3, 4, 5,
3, 3, 4, 3, 1, 5, 5, 2, 5, 4, 4, 3, 1, 4, 2, 4, 3, 3, 1, 6,
3, 5, 1, 5, 4, 4, 2, 1, 1, 1, 4, 4, 6, 4, 4, 4, 1, 3, 1, 3,
3, 3, 3, 5, 1, 2, 5, 3, 1, 3, 1, 2, 6, 6, 5, 1, 4, 2, 1, 1,
6, 6, 3, 1, 3, 5, 5, 1, 4, 2, 2, 1, 2, 1, 4, 6, 6, 6, 6, 1,
1, 3, 6, 2, 3, 4, 2, 4, 5, 3, 4, 2, 5, 5, 3, 6, 6, 6, 5, 4,
1, 1, 1, 6, 6, 5, 5, 6, 4, 6, 1, 4, 1, 2, 3, 6, 3, 6, 4, 6,
3, 3, 1, 6, 4, 1, 2, 1, 1, 5, 6, 1, 6, 6, 5, 3, 1, 5, 3, 1,
3, 4, 3, 5, 5, 4, 5, 3, 6, 6, 4, 6, 1, 4, 4, 6, 1, 6, 4, 3,
1, 2, 5, 1, 3, 4, 3, 2, 6, 5, 3, 2, 1, 2, 3, 4, 5, 1, 1, 3,
5, 3, 3, 6, 6, 4, 2, 1, 1, 1, 6, 1, 6, 2, 1, 5, 1, 3, 5, 6,
2, 6, 6, 4, 3, 2, 5, 3, 6, 4, 1, 4, 1, 2, 5, 3, 1, 5, 2, 6,
6, 3, 2, 5, 2, 2, 5, 6, 6, 4, 3, 6, 2, 2, 1, 1, 1, 1, 4, 4,
5, 1, 3, 1, 6, 3, 3, 6, 6, 4, 3, 5, 6, 5, 6, 3, 6, 6, 1, 1,
6, 4, 3, 4, 6, 6, 3, 1, 3, 3, 4, 6, 1, 2, 6, 6, 3, 1, 5, 2,
6, 4, 6, 5, 5, 4, 3, 6, 4, 4, 4, 1, 3, 2, 1, 5, 1, 2, 1, 2,
2, 5, 2, 4, 5, 5, 2, 3, 4, 3, 3, 5, 3, 3, 3, 2, 6, 1, 2, 5,
3, 2, 4, 5, 5, 5, 5, 4, 2, 2, 3, 6, 6, 2, 6, 2, 2, 2, 5, 6,
4, 3, 1, 1, 3, 1, 1, 6, 4, 5, 4, 3, 5, 3, 6, 5, 3, 3, 3, 1,
5, 2, 5, 4, 5, 6, 6, 4, 4, 2, 3, 6, 6, 1, 4, 1, 2, 5, 3, 6,
2, 2, 6, 1, 5, 4, 5, 3, 2, 2, 4, 4, 6, 3, 6, 3, 1, 6, 2, 2,
3, 6, 2, 2, 2, 3, 2, 6, 5, 1, 4, 5, 3, 5, 4, 1, 4, 2, 5, 6,
3, 6, 4, 5, 4, 6, 5, 4, 5, 1, 4, 3, 6, 5, 4, 6, 6, 4, 2, 4,
5, 4, 6, 4, 4, 4, 1, 5, 2, 6, 3, 3, 6, 1, 2, 6, 4, 1, 3, 1,
6, 3, 6, 3, 3, 3, 4, 2, 1, 2, 2, 3, 1, 1, 5, 4, 3, 4, 4, 1,
1, 1, 3, 5, 3, 5, 6, 2, 6, 1, 1, 3, 6, 6, 1, 6, 4, 6, 4, 1,
5, 1, 2, 6, 3, 6, 6, 2, 5, 3, 1, 4, 2, 1, 4, 2, 3, 6, 5, 3,

Figure 4: Extract from the list of Labels corresponding to the figure 3

The difficulty in Figure 3 is that the circular form is predominant but there are six very distinct inside pattern corresponding each to one label, as we can see on the following pictures :

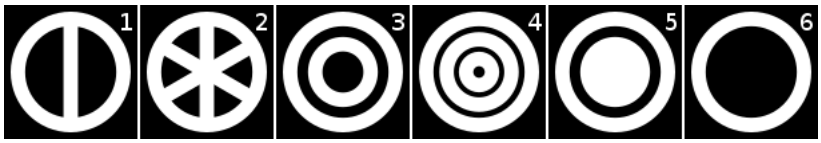


Figure 5: Six different shapes according to the Figure 3

For both contexts, Shapes and Circles, to extract the datasets from each image we use the ImageJ software. It allows us to transform an image into a data table, as shown in figure 6 representing the first 15 measurements of the image with the circles, triangles and squares. This image will be named for the following project "shapes image" and for the other one “circles image”.

	Area	X	Y	Circ.	AR	Round	Solidity
1	1156	423	39	0.813	1.000	1.000	1.000
2	908	935.751	43.249	0.919	1.003	0.997	0.960
3	906	815.696	46.135	0.927	1.007	0.993	0.963
4	662	348.441	51.678	0.592	1.017	0.983	0.937
5	1156	183.000	56.000	0.813	1.000	1.000	1.000
6	908	731.430	56.119	0.929	1.009	0.991	0.963
7	1155	635.986	60.014	0.820	1.002	0.998	1.000
8	907	594.295	71.905	0.928	1.007	0.993	0.962
9	659	89.671	80.212	0.575	1.024	0.977	0.922
10	661	265.114	82.179	0.564	1.027	0.974	0.923
11	1156	527.000	78.000	0.813	1.000	1.000	1.000
12	662	463.970	83.711	0.592	1.029	0.971	0.939
13	658	687.948	92.188	0.574	1.026	0.975	0.925
14	899	397.658	98.399	0.910	1.008	0.992	0.958
15	1188	876.972	102.500	0.825	1.029	0.971	0.999

Figure 6: The first measurements corresponding to the figure 1

To obtain this kind of table, we perform a protocol that may be different depending on the image. For the shapes image:

- We must first convert the image into a 8-bit image
- Then invert it in order to get a black background with white shapes
- Pad the image in a larger one of size (1000 x 700)
- Apply a Threshold using the Otsu method
- Set up the analyze by checking Area, centroid and shape descriptors in the set measurements
- Finally run the analyze with Analyze Particles

For the circles image it's a little bit different because we must create a stack from this image. Like shapes image we need to convert the image in a 8-bit image first, then we create the stack with 32 for the columns and 32 for the rows as shown in figure 7 and 8, which allows us to obtain a series of 1024 images containing each, a single type of label. At the end of this protocol we also get a data table.

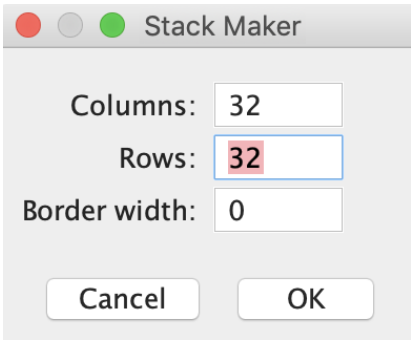


Figure 7: Creation of a stack

Figure 8 Stack circles image

For the two protocols that we just presented, we have automated them subsequently in the form of a code in JavaScript implemented in ImageJ as a plugin. It avoids much manual tasks with a risk of error during sequences. Both scripts can run in standalone and are also called by the main script for the k-NN. They are also provided in the appendix.

1.2 Code in JavaScript for ImageJ

Initially, we worked on the core algorithm of the k-NN. The main steps are the below:

1. **Datasets preparation**
    a. We need two sets of data, one for the training and one for the testing, both being associated with their targets, or Labels for each row
2. **Training, or "fit"**
    a. For such algorithm, it's more about loading the reference Data rather than "training"
3. **Prediction**
    a. At this stage, the testing data are used to calculate the Euclidean Distance with the training Data, row by row, keeping the "k" nearest neighbors
    b. The result is returned as an array of prediction for each row (one row corresponding to one symbol)
4. **Accuracy**
    a. The accuracy of the prediction is calculated by comparing the prediction of the testing Data with their known labels. Dividing the count of right answer by the count of rows gives the percentage of accuracy

As our goal was to run many times the script, using different parameters and changing the Data, we have added a couple of improvements that make the usage more friendly:

- Selection of the source image with a dialogue box
- Selection of the CSV file containing the Labels
- Choice of the ImageJ process to perform
- Noise addition
- …

Also adding user interactions avoids many direct changes into the code for each run, for instance:

- Selection of the features, or columns to keep in the Dataset for the k-NN
- Value for "k"
- …

Since the full code is written in JavaScript, most of the functions relating to ImageJ actions are using the ImageJ API "IJ", and the code exploits as well the DataFrame extension installed as plugin for ImageJ.

Finally, for convenience, we decided to keep outside the main .js script the two scripts used to prepare the context for the k-NN. We use them to process images outside the k-NN, for testing or troubleshooting purpose for instance.

- The script called proc_a_part.js process the Shapes image with Analyze Particles
- The script called proc_stack.js process the image Circles with Montage to Stack

To produce one single script, they can be easily integrated in the core script – in the if statement line 284.

2 Results

For both images, shapes and circles, we have performed several runs of the k-NN using different parameters to figure out the impact of the changes, and to identify the most relevant relationships between the Data and the results.

2.1 Shapes image

Regarding the results, we started to analyze the shapes image. First, we looked at the distributions of each shape according to the different features (Area, Shape X and Y, Circularity, AR, Round and Circularity).

2.1.1 Data distribution

We started with the Area, Circularity and Solidity features that relate easily to circles, triangles and squares.

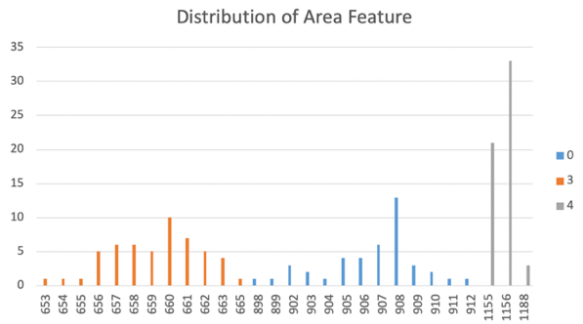


Figure 9: Distribution of Area feature

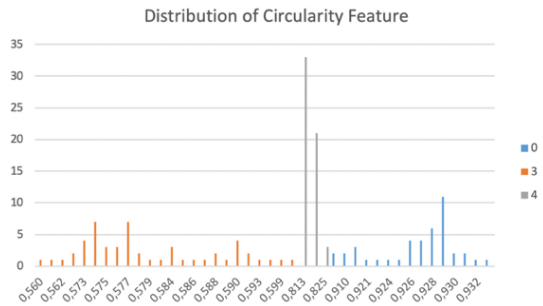


Figure 10: Distribution of Circularity feature

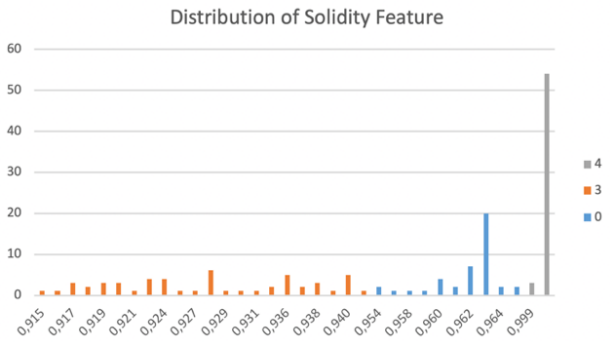


Figure 11: Distribution of solidity feature

Without surprise, figures 9, 10 and 11 show a distribution representing the 3 shapes in a distinct way present in the image. If we take the example of the Area feature, we get:

- The possible values for the triangle shape are between 653 and 665 (orange)
- The possible values for the circle shape are between 898 and 912, there is a gap between the first shape but also with the square shape (blue)
- the possible values for the square shape are between 1155 and 1188 (grey)

If we look at the results for the other 2 features (Circularity and Solidity), the distribution is similar with obviously different values for each shape. Now let's investigate the 4 other characteristics whose results are different:

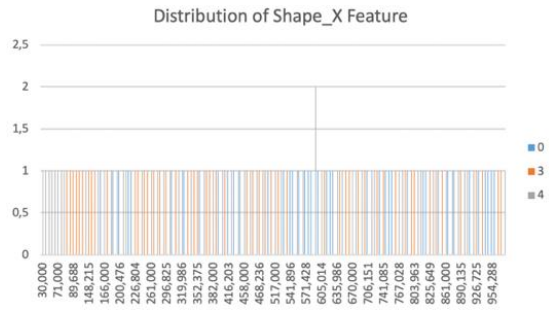


Figure 12: Distribution of Shape X feature

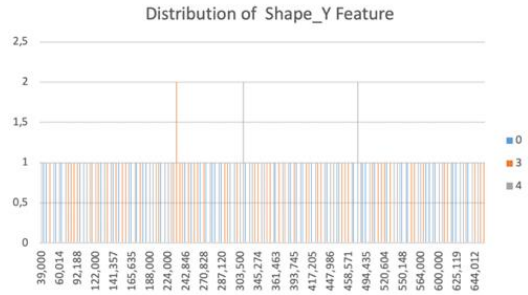


Figure 13: Distribution of Shape Y feature

Looking at the graphs in Figures 12 and 13, we observe that each of the 151 shapes presents in the image have a specific number for the features shapes X and Y, with one exception. But the distribution of the following figures is much more interesting:

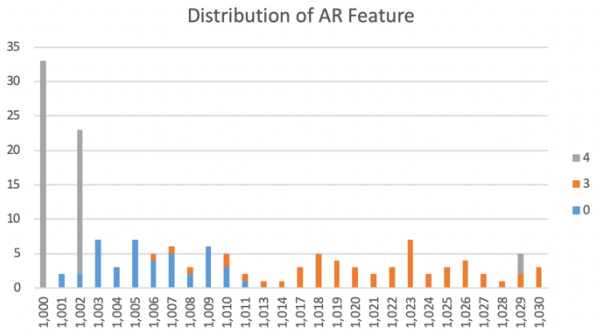


Figure 14: Distribution of AR feature

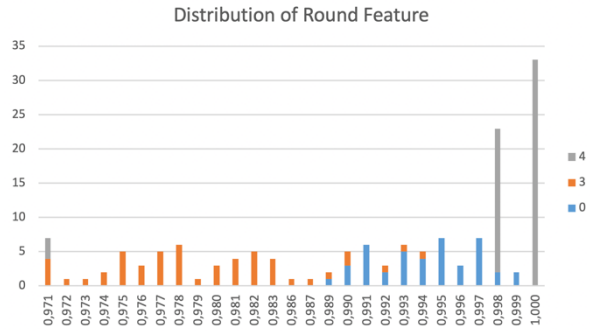


Figure 15: Distribution of Round feature

Figures 14 and 15 represent a distribution similar to Figures 9, 10 and 11. We see the 3 kinds of shapes, but we observe that some measured value can represent two shapes. For an example, for the the AR feature (Figure 14):

- The values between 1.006 and 1.008 may correspond to either a triangle or a circle
- The value 1,029 can be a triangle or a square

These results may impact the accuracy of our classifier. To evaluate this, we measured the accuracy of our model.

2.1.2 Number of neighbors, the value for k

Going further, we also implemented an incrementation of the number of neighbors by varying the number for k from 1 to 20, to compare the results. We started our analysis by giving our algorithm the dataset corresponding to the 7 features:

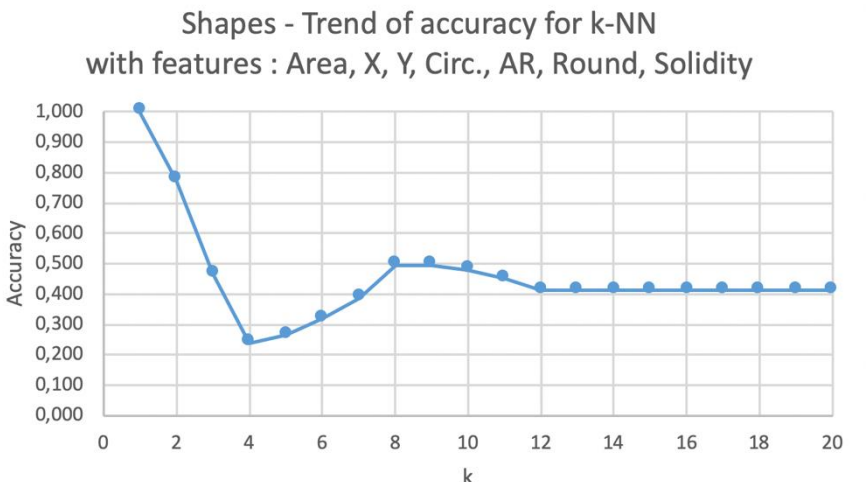


Figure 15: Graphics representing the trend for k-NN with the 7 features

The first thing we observe is that for k=1 we obtain an accuracy of 100%. So, we have all the necessary training data to classify the shapes. On the other hand, by increasing k up to 4, the rate of accuracy falls below the 30% threshold.

In the further increases for k to 20, the accuracy will reach 50% and then arrives at a tray from k = 12, with an accuracy rate at about 41%. It means there is a bias that appears from a certain value of k, then we talk about overfitting. It happens because the Machine Learning model captures too much “aspects” and details from the training dataset. In other words, the predictive model will capture generalized correlations and the noise generated by the data.

Now if we give our classifier only the values corresponding to the AR and rounds features, we should expect an accuracy less than 100% because, as we saw in figures 14 and 15, multiple symbols can match one value.

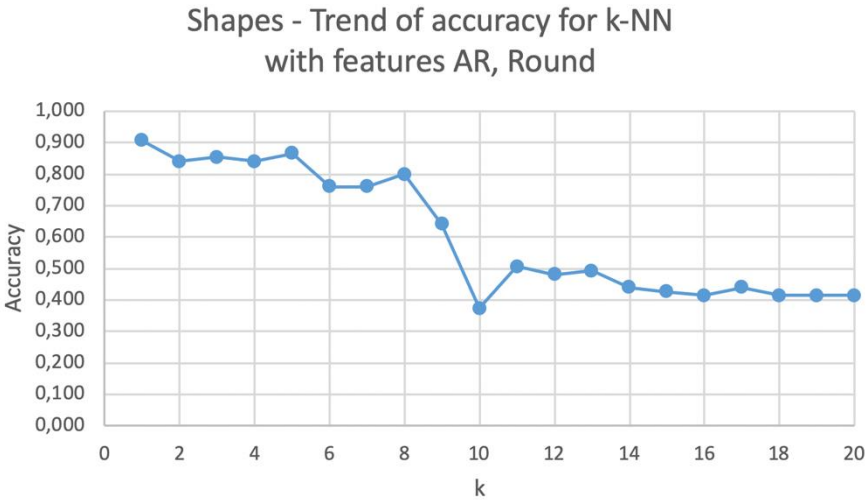


Figure 16: Graphics representing the trend for k-NN with the AR and Round features

Actually, for k=1, we get an accuracy of 90% which may fall to about 38% for k= 10, then get stabilized around 40% for a k between 10 and 20 (Figure 16). There is indeed an impact on accuracy when we increase the value of k, using a dataset with some uncertainty on values.

We then carry out the study of the trend curve of the Area feature alone, which has strict values for each form:

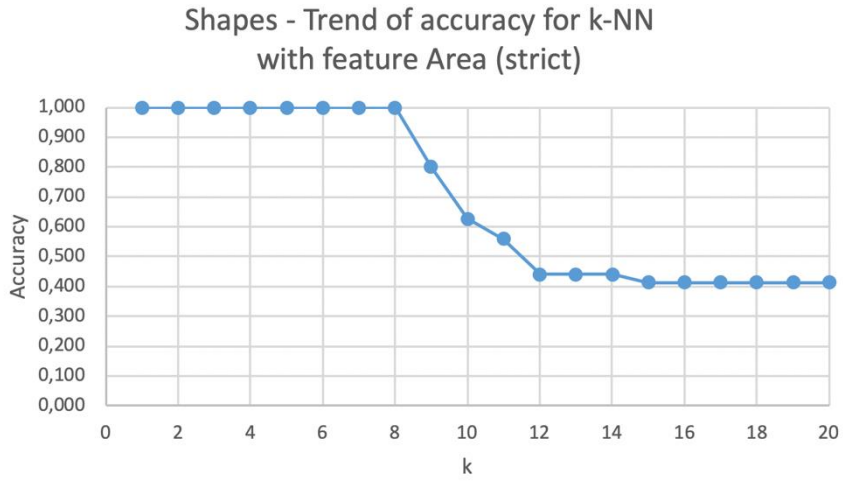


Figure 17: Graphics representing the trend for k-NN with the Area feature

As shown in Figure 17, we have an accuracy of 100% for k between 1 and 8 then a significant decrease until reaching a tray between k= 12 and k= 20, with an accuracy rate around 40% as we seen for previous experiments with the term “overfitting”. The 100% accuracy is justified by the dataset having no uncertainty. It helps the algorithm predicting right each time.

The bias is introduced by the number of neighbors and we can explain this by the needs to get more neighbors than relevant ones in the dataset. That way, the k-NN is overfitting itself, taking more options than needed, and then, predicting wrong sometimes.

To finish on the results concerning shapes image, we combined the features Area, AR and Round to evaluate the impact of not relevant features when associated to a relevant one:

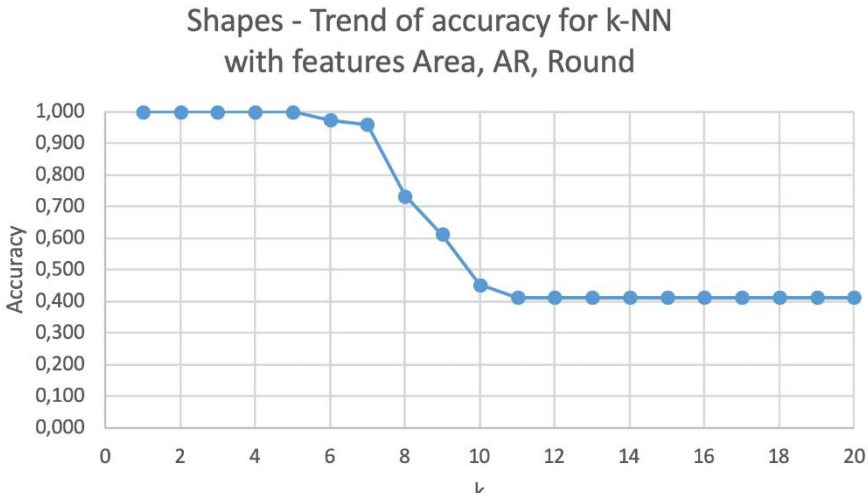


Figure 18: Graphics representing the trend for k-NN with the Area, AR and Round features

If we compare Figure 18, which added the data set Area, with the results in Figure 16 we find that we obtain more or less the same results as Figure 17. This shows the importance of data analysis in advance and the preparation of the dataset, to have to use in priority the most relevant Data, and to reduce as much as possible the not relevant one.

2.2 Circles image

Thanks to the results seen above, we were able to focus more on the data of interests relating to the analysis of the Circles image. The trend of accuracy for k-NN is drawn directly with all the data corresponding to the 7 features (figure 19):

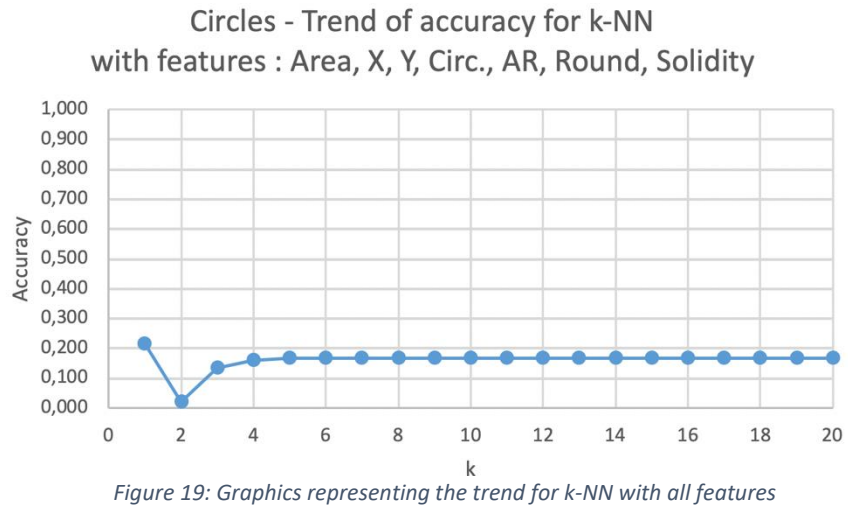


Figure 19: Graphics representing the trend for k-NN with all features

2.2.1 Features analysis

Contrary to the results of image shapes we do not get 100% accuracy and we can even say that the results overall are bad because the average accuracy rate is 16%. It corresponds to 1 chance out of 6 possible symbols. These results can be explained easily looking at the table of our dataset coming from the results from ImageJ.

Index	Area	Mean	Min	Max	X	Y	Circ.	Slice	AR	Round	Solidity	Label
1	1089	86,438	0	255	16,5	16,5	0,785	1	1	1	1	3
2	1089	102,019	0	255	16,5	16,5	0,785	2	1	1	1	5
3	1089	105,404	0	255	16,5	16,5	0,785	3	1	1	1	4
4	1089	56,69	0	255	16,5	16,5	0,785	4	1	1	1	6
5	1089	96,843	0	255	16,5	16,5	0,785	5	1	1	1	2
6	1089	102,019	0	255	16,5	16,5	0,785	6	1	1	1	5
7	1089	86,438	0	255	16,5	16,5	0,785	7	1	1	1	3
8	1089	96,843	0	255	16,5	16,5	0,785	8	1	1	1	2
9	1089	96,843	0	255	16,5	16,5	0,785	9	1	1	1	2
10	1089	102,019	0	255	16,5	16,5	0,785	10	1	1	1	5
11	1089	72,221	0	255	16,5	16,5	0,785	11	1	1	1	1
12	1089	86,438	0	255	16,5	16,5	0,785	12	1	1	1	3
13	1089	105,404	0	255	16,5	16,5	0,785	13	1	1	1	4
14	1089	72,221	0	255	16,5	16,5	0,785	14	1	1	1	1

Figure 20: Table representing the first 14 measurement for all the features

The figure above shows more than the 7 features we had for the shapes images because we have added Mean, Min and Max measurements to find a relevant value for the k-NN.

We notice that the values for each column is identical, which is normal because we have similar shape circles, except for one which is Mean. So, for the circles image the Mean feature is the most (and only) relevant data that we decided to use for the next experiments.

2.2.2 Number of neighbors, the value for k

Using Mean only, we performed the test, then draw the trend of accuracy curve:

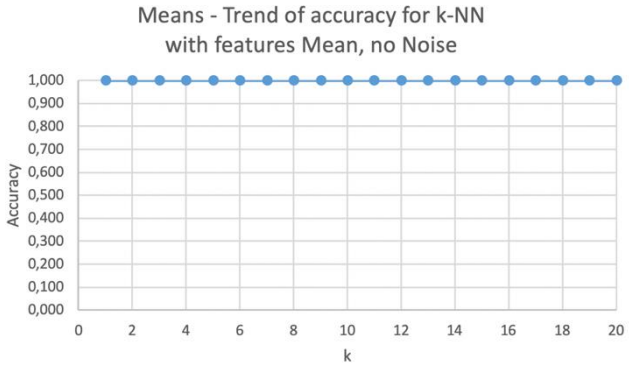


Figure 21: Graphics representing the trend for k-NN with the Mean feature

For a k varying from 1 to 20, we obtain a mean accuracy rate of 100%. Which seems logical because as we can see on figure 20, each mean value corresponds to a specific symbol, so we are sure to fall each time into the right shape.

It's interesting this time to see a stable accuracy at 100% from 1 to 20 neighbors for the k value. We did the same exercise increasing further the k value to see if we get the same behavior rather than for the shapes image, with the accuracy falling from some point.

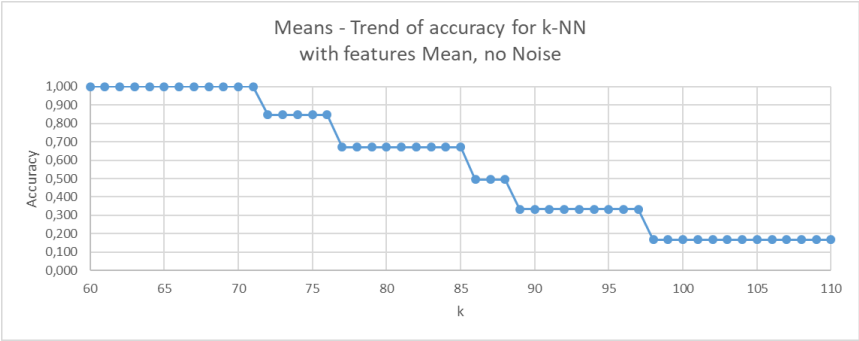


Figure 22: Graphics representing the trend of accuracy with k from 60 to 110 with feature Mean

As expected, we definitely reached out the limit for the k value to keep the k-NN 100% accurate, but the value is very high compared to k=8 for the shapes image. We can justify this by the dataset for circles which is much larger with 1024 measures for 6 distinct symbols, instead of 151 measures for 3 distinct shapes. Also, we see 5 trays corresponding to the statistical "chances" to get the right prediction:

- From k=72 to 76, we have 5 chances out of 6
- From k=77 to 85, we have 4 chances out of 6
- From k=86 to 88, we have 3 chances out of 6
- From k=89 to 97, we have 2 chances out of 6
- From k=98 the last tray is reached, with no more than 1 chance out of 6

2.2.3 Noise addition

For the next experiments, we added different level of noise to compared the results using multiple k values as usual. The noise is added by applying a standard deviation a certain number of cycles. For instance a std dev of 25, 3 time, or 50 1 time.

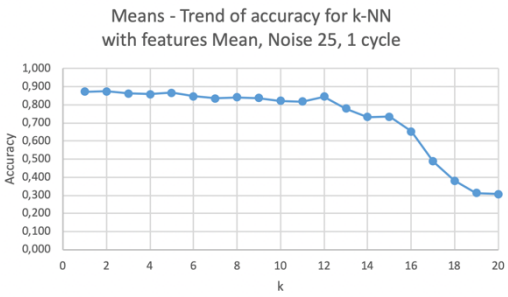


Figure 23: Graphics representing the trend for k-NN with Mean features with noise 25 cycle 1

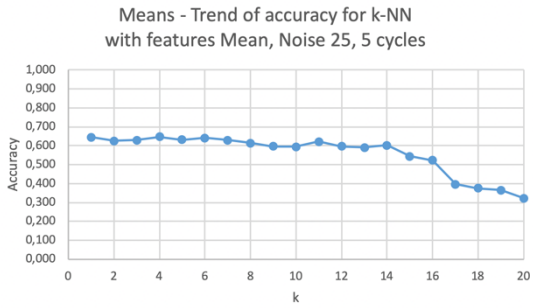


Figure 24: Graphics representing the trend for k-NN with Mean features with noise 25 cycle 5

The addition of noise to the image allows to analyze the ability of the model to provide a correct response when the image is degraded. Looking at figures 23 and 24, for a standard deviation of 25 cycled 1 time, the accuracy is between 80% and 90% with k increasing up to 12. So, adding noise directly impacts accuracy even taking mean as feature.

This phenomenon is accentuated by 5 noise cycles 25. For a k between 1 and 12 the accuracy is about 60%. For both graphs the accuracy decreases by increasing the k until reaching the minimum threshold of 30% accuracy. We explain the tendency of the precision decreases with the increase ok the value for k, which introduces a bias, due to the proximity of the values which influence the calculation of Euclidean distance as shown in the following Data distribution study.

2.2.4 Data distribution

The first analysis illustrates the natural distribution with no noise added.

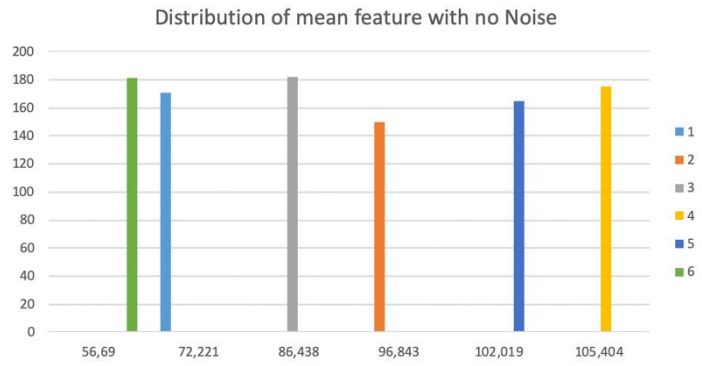


Figure 25: Graphics representing distribution of mean feature with no noise

We see the distinct distribution of values to each shape, allowing the k-NN to reach out 100% of accuracy.

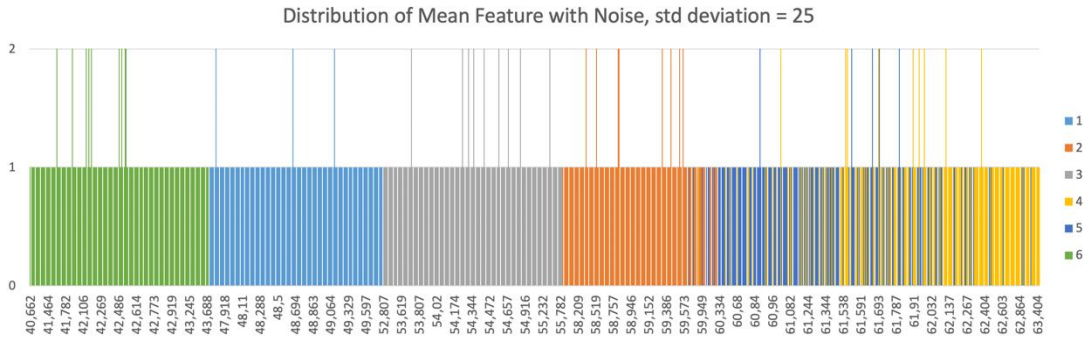


Figure 26: Graphics representing distribution of mean feature with noise, std deviation = 25

These two figures, above and below, show that the degradation of the image changes the distribution. the impact is even more important if we add noise with a standard deviation of 50, introducing more bias in the Dataset.

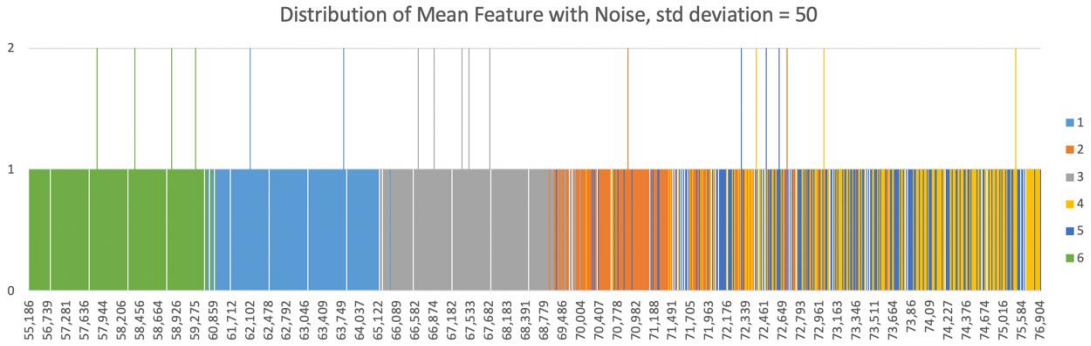


Figure 27: Graphics representing distribution of mean feature with noise, std deviation = 50

We ran the script again using this dataset with a standard deviation of 50, 1 time, to get the trend for the accuracy.

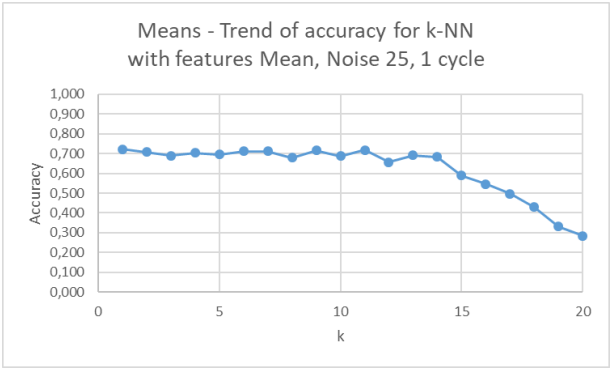


Figure 28: Graphics representing the trend for k-NN with Mean features with noise 50 cycle 1

Again, we observe an impact on the accuracy from the beginning, at 70%, until a k value of 7, with a fall starting from 15.

3 Discussion & Conclusion

Before to conclude, we wanted to go further by adding one more functionality that allows the k-NN to be used to classify symbols. in other words, we provide an image corresponding to one of the 6 symbols and in return, they tell us which label it belongs to.

3.1 Addition of an “exploitation” functionality to the script

Here is how to use the new functionality. The first part is the same script where we have to load the image and the list of labels in the form of a CSV. We are doing the montage to stack. we will then ask the question if we want to add noise, in this example we will do without as the Standard Deviation is kept to 0.

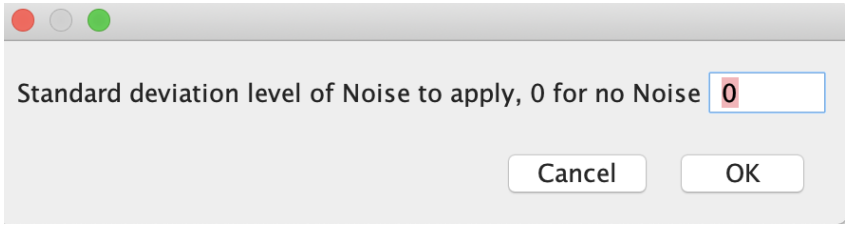


Figure 28: Add standard deviation or not

Then we are asked what features we want to use, so for picture circles we only keep Mean.

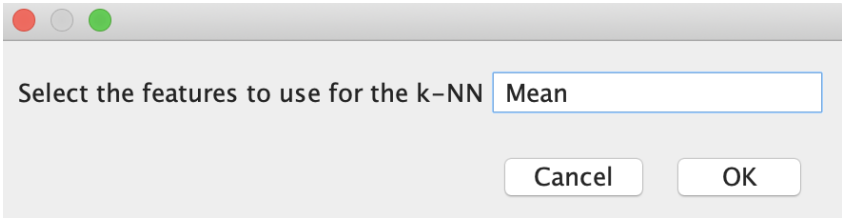


Figure 29: features selection

After the main core script starts the new “exploitation” functionality. It begins with a question to confirm if we want to predict the label of an image or not. (this is to stop here if we work on the k-NN itself for getting accuracy in many configurations).

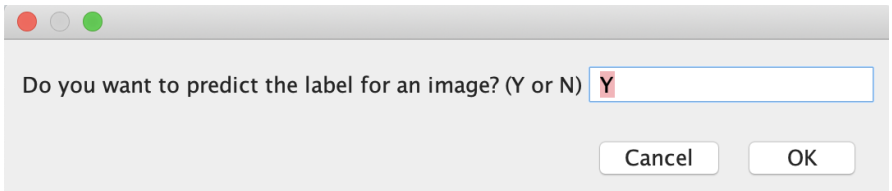


Figure 30: Prediction for an image

For this example, we will give the 3 following images, corresponding to the first 3 shapes issued by the Montage to Stack of the Circle image.

Figure 31: Selection of images

The code loops on image prediction until we decide to stop. Repeating the image selection with the question 3 times, we get the results below:

```
k value = 1, Features used: Mean
Symbol 86.43801879882812 is of type: 3
k value = 1, Features used: Mean
Symbol 102.019287109375 is of type: 5
k value = 1, Features used: Mean
Symbol 105.40403747558594 is of type: 4
```

Figure 32: Results in the console of ImageJ

As we used images from the Stack, the observed results are easily verifiable by comparing with the table containing the data set and the different labels associated:

A	B	C	D	E	F	G	H	I	J	K	L	M
Index	Area	Mean	Min	Max	X	Y	Circ.	Slice	AR	Round	Solidity	Label
1	1089	86,438	0	255	16,5	16,5	0,785	1	1	1	1	3
2	1089	102,019	0	255	16,5	16,5	0,785	2	1	1	1	5
3	1089	105,404	0	255	16,5	16,5	0,785	3	1	1	1	4

Figure 33: Dataset with labels associated to measures

In the framework of this project the k-NN classifier is an interesting algorithm to develop because it fits perfectly with the subject we deal with, that is to predict the label of a symbol, even when degrading it, as soon as the symbol is kept simple.

3.2 Conclusion

Based on the results we have observed throughout this project, we can conclude 3 important principles concerning the k-NN classifier:

- First, the very high dependency of accuracy on **quality** and **quantity** of the data provided to k-NN in training

- Then the quite **good efficiency** of the classifier as soon as the tuning is well done, and the Euclidean distances are significant. At this point, we have measured that the accuracy can exceed 80%, even by adding some Noise (the features must remain relevant enough)

- Finally, attention should be paid to the **risk of bias**, which increases when the number of measures decreases or when the gap between the discriminant values is too small

Finally, this algorithm works, and it can be used in operations to classify "simple" symbols. This model could be extended to other types of measurements, for instance to colors and dimensions.

4 References

Writing Our First Classifier - Machine Learning Recipes #5 <https://www.youtube.com/watch?time_continue=6&v=AoeEHqVSNOw&feature=emb_logo>

'Crazybiocomputing/Crazytijs', *GitHub* <https://github.com/crazybiocomputing/crazytijs>

'*K*-Nearest Neighbors Algorithm', *Wikipedia*, 2020 <https://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=934101283>

Benzaki, Younes, 'Introduction à l'algorithme k Nearest Neighbors (KNN)', *Mr. Mint : Apprendre le Machine Learning de A à Z*, 2018 <https://mrmint.fr/introduction-k-nearest-neighbors>

'KNN Using JavaScript and Underscore' <http://16cards.com/2013/06/06/knn-in-parse-cloud/>

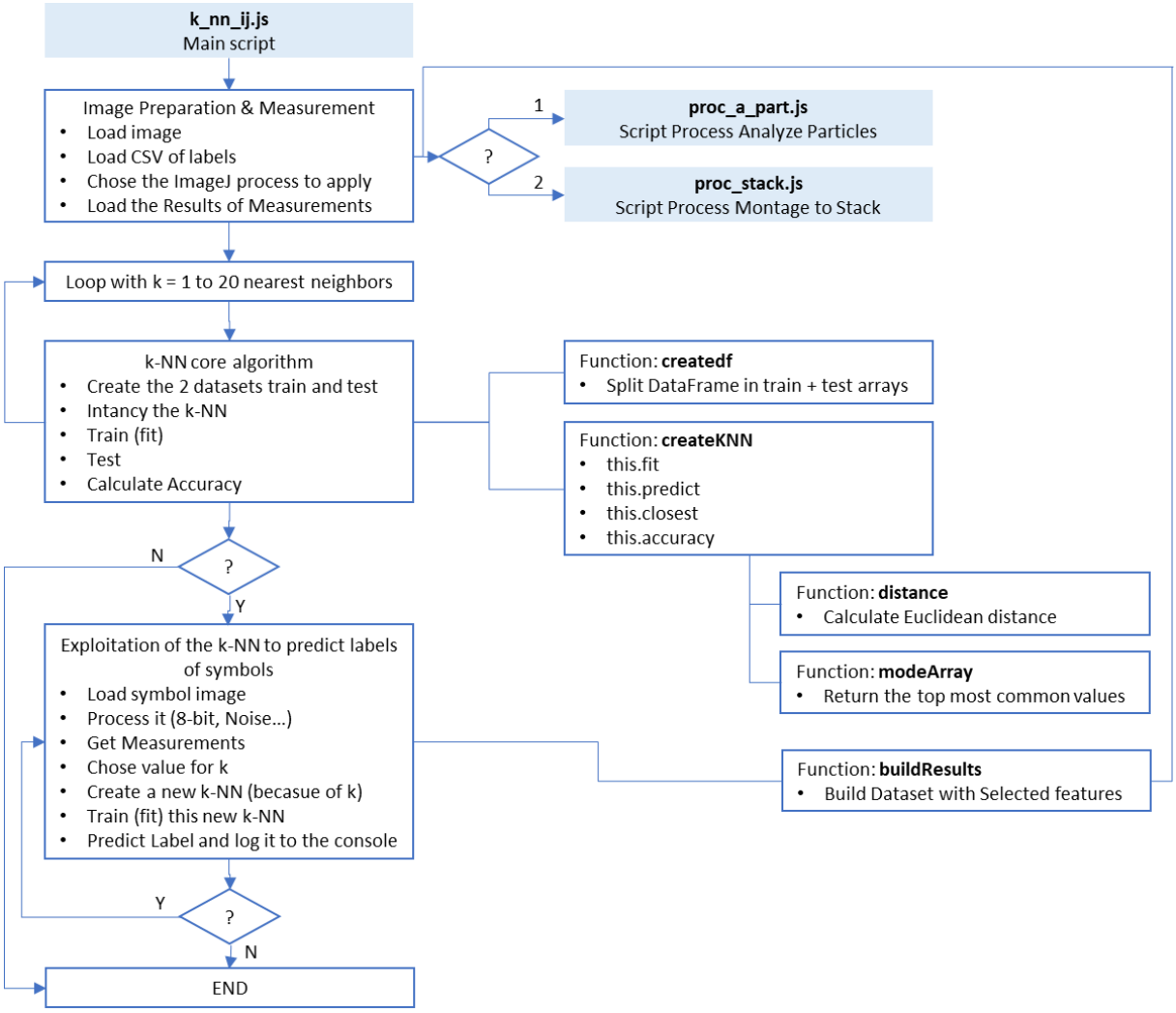
'IJ (ImageJ API)' <https://imagej.nih.gov/ij/developer/api/ij/IJ.html>

'ImageJ' <https://imagej.nih.gov/ij/macros/js/>

5 Annexes

5.1.1 Overview of Code Structure

Here is an overview of the structure of the final code:



5.1.2 k_nn_ij.js - Main Script

```
/*
k-NN algorithm for ImageJ
This program can use either analyze particule or montage to stack process
main steps are:
  -preparation of the Data with ImageJ
  -training (fit)
  -testing
  -accuracy measurement
```

```
  -exploitation (using stack images to predict)
requirements:
  -ImageJ v1.52s, Java SDK 9.0.4
external js scripts as plugin:
  -proc_stack.js, proc_a_part.js
  -
Alexandre Cornier - Jan. 2020
*/

// Import helper functions (used for instance for DataFrame() and console.log(
))
const IJ_PLUGINS = IJ.getDir('plugins');
load(`${IJ_PLUGINS}/javascript/nashorn_polyfill.js`);
load(`${IJ_PLUGINS}/javascript/tml.js`);

/*-------------------------------------------------------------------------
Set Global const and var
-------------------------------------------------------------------------*/

// Set the 4 DataFrames needed for Training & Test with Data & Labels
const df_train_data = [];
const df_train_label = [];
const df_test_data = [];
const df_test_label = [];

// Set the subdirectory hosting proc_a_part.js & proc_stack.js plugins
const pluginsHome = "0Home/";

/*-------------------------------------------------------------------------
Functions
-------------------------------------------------------------------------*/

/*
Create the 4 Dataframes needed from the loaded dataset
Argument:
- dataset: array of arays containing all the Data inclusing Labels
Return nothing but fillin the 4 df_* const with the appropriate Data
-------------------------------------------------------------------------*/
function createdf(label, dataset) {
  // Check lenght of both arrays for equality

  if (label.length != dataset.length) {
    console.log("ERROR: both tables for labels and data don't have the same si
ze: label=" + label.length + ", data=" + dataset.length);
  }

  // Use first half of the data for training
  // Math.trunc is not supported by ImageJ so Math.floor is used instead
  let isplit = Math.floor(dataset.length / 2);
```

```
  for (let i = 0; i <= isplit; i++) {
    df_train_data.push([dataset[i]]);
    df_train_label.push(label[i]);
  }
  isplit = isplit + 1;
  // Use second half of the data for testing
  for (i = isplit; i < dataset.length; i++) {
    df_test_data.push([dataset[i]]);
    df_test_label.push(label[i]);
  }
}

/*
Get Euclidean distance between two points
Arguments:
- first point: coordintates as array
- Second point: coordinates as array
Return distance or -1 if the two points don't have the same dimension
------------------------------------------------------------------------------*/
function distance(a, b) {
    let total = 0, diff = 0, fDist = 0;
    if (a.length != b.length) {
      console.log("ERROR: Distance - both points don't have the same dimension
")
      console.log(a);
      console.log(b);
      return -1;
    }
    for (let i = 0; i < a.length; i++) {
        if (a[i] > b[i]) {
            diff = a[i] - b[i];
          } else {
            diff = b[i] - a[i];
          }
        total += diff * diff;
    }
    fDist = Math.sqrt(total);
    return fDist;
}

/*
Get the most common value found in an array
Argument:
- arrLabel: array of values
Return one single value that is the most common label
------------------------------------------------------------------------------*/
function modeArray(arrLabel) {
  let most_freq = 1;
```

```
  let icount = 0;
  let label = arrLabel[0];
  for (let i = 0; i< arrLabel.length; i++) {
    for (let j = i; j< arrLabel.length; j++) {
      if (arrLabel[i] == arrLabel[j]) {
        icount++;
      }
      if (most_freq < icount) {
        most_freq = icount;
        label = arrLabel[i];
      }
    }
    m=0;
  }
  return label;
}

/*
Build the table of results with the selected features from a DataFrame
Argument:
- df: DataFram from ImageJ Results
- ft_str: CSV String containing selected features separated by comma
Return an array with Measurements in Row and features in Columns
------------------------------------------------------------------------*/
function buildResults(df, ft_str) {
  let features = ft_str.split(",");

  // Create the new Dataset with selected columns (features)
  let data_arr = [];
  if (features.length < 2) {
      // Case of one single feature, no need to combine columns
      let df_single = new DataFrame();
      df_single = df.select(features[0]);
      data_arr = df_single.array();
  } else {
      // Case of multiple features
      let df2 = [];
      let df2_arr = [];
      // Get the columns corresponding to features selected
      for (let f = 0; f < features.length; f++) {
          df2[f] = df.select(features[f]);
          df2_arr[f] = df2[f].array();
      }

      // Create new rows and push them in the final table
      for (let i = 0; i < df2_arr[0].length; i++) {
          let arr_temp = [];
          for (let j = 0; j < features.length; j++) {
              arr_temp.push(df2_arr[j][i]);
```

```
            }
            data_arr.push(arr_temp);
        }
    }

    return data_arr;
}

/*
createKNN - main "Class" containing the fit and predict functions
Argument:
- k: number of closest neighbors
------------------------------------------------------------------------*/
function createKNN(k) {
    this.k = k; // number of neighbors to consider
    this.prediction = []; // array of labels predicted using the test Data
    /*
    initialize DataFrame for Training
    Arguments:
    - df: DataFrame containing the Data
    - labels: array of numbers containing the labels of the observations in df
    = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
    this.fit = function(df, labels) {
        this.x_train = df;
        this.y_train = labels;
    }

    /*
    predict Labels
    Argument:
    - df: DataFrame of unlabeled test Data
    Return an array of predicted Labels
    = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
    this.predict = function(df) {
        for(let i = 0; i < df.length; i++) {
            let k_labels = this.closest(df[i]);
            let label = modeArray(k_labels);
            this.prediction.push(label);
        }
    }

    /*
    Get closest neighbors
    Arguments:
    - row: data to evaluate
    Return an array of k labels from closest neighbors
    = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
    this.closest = function(row) {
        let dist = 0;
```

```
    let best_dist = [], best_index = [], k_labels = [];
    // set arrays of resulting distances & indexes
    for(let n = 0; n < this.k; n++) {
      best_dist.push(distance(row, this.x_train[0]));
      best_index.push(0);
      k_labels.push(0);
    }
    // compute distance with each row in training Dataframe
    for(let i = 0; i < this.x_train.length; i++) {
      dist =  distance(row, this.x_train[i]);
      // record and sort the distance - if shorter - in array of results
      for(n = 0; n < this.k; n++) {
        if (dist < best_dist[n]) {
          best_dist[n] = dist; // new best distance as result
          best_index[n] = i; // corresponding index into the training Datafram
e
          break;
        }
      }
    }
    //get array of labels for closest neighbors
    for(let n = 0; n < this.k; n++) {
      k_labels[n] = this.y_train[best_index[n]];
    }
    return k_labels;
  }

  /*
  Get accuracy of prediction
  Arguments:
  - test_labels: array of known Labels for the test DataFrame
  Return percentage of accuracy in a range[0..1]
  = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
  this.accuracy = function(test_labels) {
    let label_match = 0, acc_percentage = 0;
    if (test_labels.length != this.prediction.length) {
      console.log("ERROR: Prediction and Test tables don't have the same lengt
h.")
      acc_percentage = -1;
    } else {
      for (let i = 0; i < test_labels.length; i++) {
        if (test_labels[i] == this.prediction[i]) {
          label_match++;
        }
      }
      acc_percentage = label_match / test_labels.length;
    }
    return acc_percentage;
  }
```

```
}

/*------------------------------------------------------------------------
Core algorithm - ImageJ - Image preparation & Measurement
------------------------------------------------------------------------*/

// Get the image fullpath from a dialog box
let path = IJ.getFilePath("Select the image to open");

// Open image
let image = IJ.openImage(path);
image.show("Image"); // display it

// Get the CSV file fullpath containing labels from a dialog box
path = IJ.getFilePath("Select the CSV file containing the labels for this imag
e");

// Open file
let CSV_labels = ResultsTable.open(path);
CSV_labels.show("Results");

// Load DataFrame from the Result Window
let table = ResultsTable.getResultsTable();
let df_label = new DataFrame();
df_label.fromIJ(table);

// Close the Result Window used to load the CSV Labels
IJ.selectWindow("Results");
IJ.run("Close");

// Create an array containing the Labels and load the Data from the DataFrame
let label_arr = df_label.column('Label').array();

// Get the process to apply, to choose between image to stack and analyze part
icles
let iprocess = IJ.getNumber("ImageJ process to apply to the image: 1 for analy
ze particles, 2 for montage to stack, ", 1);

// Get the plugin directory
let plugin_path = IJ.getDirectory("plugins");
let plugin = "";

// Run the selected process script
if (iprocess == 1) {
  // Case of Shapes measured using Analyze Particles
  plugin = plugin_path + pluginsHome + "proc_a_part.js"
  IJ.runMacroFile(plugin);
} else if (iprocess == 2) {
  // Case of Circles measured using Montage to Stack
```

```
  plugin = plugin_path + pluginsHome + "proc_stack.js"
  IJ.runMacroFile(plugin);
}

// Load DataFrame from the Result Window after Measurement
let table_results = ResultsTable.getResultsTable();
let df_results = new DataFrame();
df_results.fromIJ(table_results);

// Get the features to use
let q_str = df_results.headings;
let features_str = IJ.getString("Select the features to use for the k-
NN", q_str);

// Build the target Dataset containing only selected features
let results_arr = buildResults(df_results, features_str);

/*-------------------------------------------------------------------------
Core algorithm of k-NN using values [1..20] for k
-------------------------------------------------------------------------*/

// Create the 4 DataFrames for train & Test with Data & Labels in distincts ar
rays
createdf(label_arr, results_arr);

let faccuracy = 0;
for (let k = 1; k < 21; k++) {

  // Create the k-NN object
  let classifier = new createKNN(k);

  // Load the Data & Labels for the training into the k-NN object
  classifier.fit(df_train_data, df_train_label);

  // Predict Labels using test Data (without providing Labels)
  classifier.predict(df_test_data);

  //Measure accuracy by comparing test & predicted labels
  faccuracy = classifier.accuracy(df_test_label);
  console.log(faccuracy);
}

/*-------------------------------------------------------------------------
Exploitation of the k-NN to predict labels of symbols
-------------------------------------------------------------------------*/

// Get answer for predicting an image or not
let bpredict = IJ.getString("Do you want to predict the label for an image? (Y
 or N)", "Y");
```

```
if (bpredict == "Y") {
  // Close the Stack window
  IJ.selectWindow("Stack");
  IJ.run("Close");

  do {
    // Close the Result window
    IJ.selectWindow("Results");
    IJ.run("Close");

    // Get the image fullpath from a dialog box
    let path_img = IJ.getFilePath("Select the image to open for prediction");

    // Open image
    let img_test = IJ.openImage(path_img);
    img_test.show("Img_test");

    // Instance current image
    let imp = IJ.getImage();

    // Convert to Gray 8-bit
    IJ.run(imp, "8-bit", "" );

    // Get noise level to apply
    let fnoise = IJ.getNumber("Standard deviation level of Noise to apply, 0 f
or no Noise", 0);

    // Add noise if the deviation entered is > 0
    if (fnoise > 0) {
      // Get the number of cycles to apply Noise
      let fnb = IJ.getNumber("Number of cycles to apply Noise", 1);

      // Divide by 2 the pixels intensity
      IJ.run(imp, "Divide...", "value=2");

      // Cycle noise
      let std_noise = "standard=" + fnoise;
      for (let i = 0; i < fnb; i++) {
        IJ.run(imp, "Add Specified Noise...", std_noise);
      }
    }

    //Set Measurements
    IJ.run(imp, "Set Measurements...", "area mean min centroid shape redirect=
None decimal=3");

    // Get results
    IJ.run(imp, "Measure Stack...", "");
```

```
    // stop the console.info to not confuse the logs with the Columns loaded i
n DataFrame from Results
    let consoleOrg = console.info;
    console.info = function () {};

    // Load DataFrame from the Result Window after Measurement
    table_results = ResultsTable.getResultsTable();
    df_results = new DataFrame();
    df_results.fromIJ(table_results);

    // Reactivate the console.info
    console.info = consoleOrg;

    // Build the Dataset containing only selected features
    let results_arr = [];
    results_arr.push([buildResults(df_results, features_str)]);

    // Get the number of nearst neighbors for the k-NN
    let k = IJ.getNumber("How many neighbors for the k-NN?", 1);

    // Create the k-NN object
    let classifier = new createKNN(k);

    // Load the Data & Labels for the training into the k-NN object
    classifier.fit(df_train_data, df_train_label);

    // Predict Labels using test Data (without providing Labels)
    classifier.predict(results_arr);
    console.log("k value = " + k + ", Features used: " + features_str);
    console.log("Symbol " + results_arr + " is of type: " + classifier.predict
ion);

    // Get answer for predicting another image or not
    bpredict = IJ.getString("Do you want to predict the label for another imag
e? (Y or N)", "Y");

    // Close the previous image
    imp.close();

  } while (bpredict == "Y");
}

// Close images and results windows to clean up
IJ.run("Close All");
IJ.selectWindow("Results");
IJ.run("Close");

//------------------------------- END -----------------------------------
```

5.1.3 proc_a_part.js – script to process the Shapes image with Analyze Particles

```
/*
Process the current image in Imagej with Analyze Particles
This script is to be used as a plugin in ImageJ
Can be use alone or called by the main k_NN_js script
Process:
- convert to 8 bit
- invert
- resize to 1000x700
- set threshold to Otsu dark
- set measurements
- analyze particles to get results

Alexandre Cornier - Jan. 2020
*/

// Instance current image
let imp = IJ.getImage();

// Convert to Gray 8-bit
IJ.run(imp, "8-bit", "" );

// Invert
IJ.run(imp, "Invert", "" );

// Increase canvas size
IJ.run(imp, "Canvas Size...", "width=1000 height=700 position=Center zero");

// Set Threshold
IJ.setAutoThreshold(imp, "Otsu dark");

// Set Measurements
IJ.run(imp, "Set Measurements...", "area centroid shape redirect=None decimal=
3");

// Analyze particles
IJ.run(imp, "Analyze Particles...", "  show=Outlines display");
```

5.1.4 proc_stack.js – script to process the Shapes image with Analyze Particles

```
/*
Process the current image in Imagej with Montage to Stack
This script is to be used as a plugin in ImageJ
Can be use alone or called by the main k_NN_js script
Process:
- convert to 8 bit
- ask for a Noise Standard Deviation, if entered
  - divide by 2 the pixels intensity
  - ask for a number of cycle to apply the std deviation
  - apply n times the std deviation
- set measurements
- run the montage to stack
- measure stack to get results

Alexandre Cornier - Jan. 2020
*/

// Instance current image
let imp = IJ.getImage();

// Convert to Gray 8-bit
IJ.run(imp, "8-bit", "" );

// Get noise level to apply
let fnoise = IJ.getNumber("Standard deviation level of Noise to apply, 0 for n
o Noise", 0);

// Add noise if the deviation entered is > 0
if (fnoise > 0) {
  // Get the number of cycles to apply Noise
  let fnb = IJ.getNumber("Number of cycles to apply Noise", 1);

  // Divide by 2 the pixels intensity
  IJ.run(imp, "Divide...", "value=2");

  // Cycle noise
  let std_noise = "standard=" + fnoise;
  for (let i = 0; i < fnb; i++) {
    IJ.run(imp, "Add Specified Noise...", std_noise);
  }
}

// Set Measurements
IJ.run(imp, "Set Measurements...", "area mean min centroid shape redirect=None
 decimal=3");

// Create stack from image
```

```
IJ.run(imp, "Montage to Stack...", "columns=32 rows=32 border=0");

// Instance the stack
let stack = IJ.getImage();

// Get results
IJ.run(stack, "Measure Stack...", "");
```