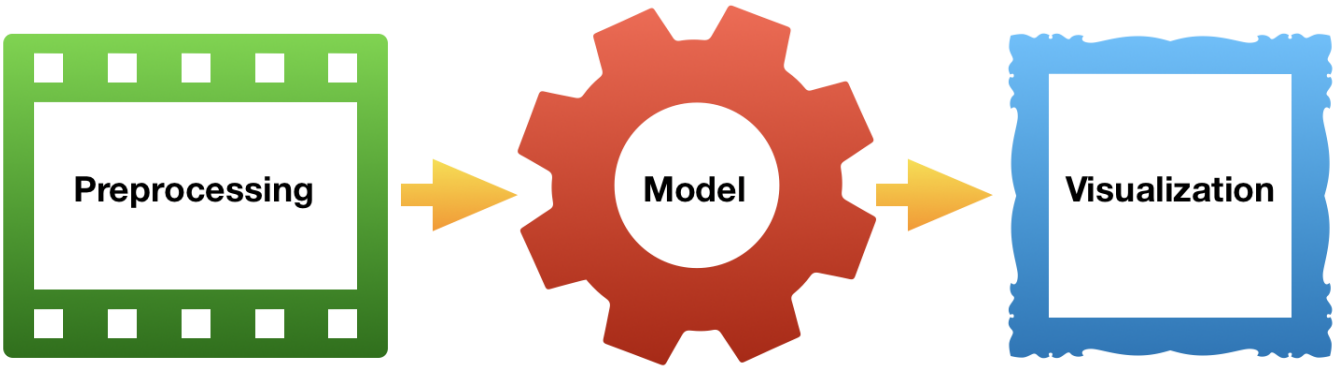
# PROPOSAL INFO232

## Analysis of cells to detect Malaria

14/03/2020




**Main steps of the project**

**Project :** MEDICHAL

**Team name :** SAHARA

**Team members :** Guillaume ABADIE < guillaume.abadie@u-psud.fr >, Louise ALLAIN < louise.allain@u-psud.fr >, Paul CATILLON < paul.catillon@u-psud.fr >, Alexandre CIORASCU < alexandre.ciorascu@u-psud.fr >, Mohamed-Fouad FELLAH < mohamed-fouad.fellah@u-psud.fr >, Nelly LAM < nelly.lam@u-psud.fr >

**Github repo of the project :** https://github.com/LouiseALLAIN/SAHARA

- *Duo 1 : Preprocessing - Paul CATILLON, Alexandre CIORASCU*
- *Duo 2 : Model - Guillaume ABADIE, Louise ALLAIN*
- *Duo 3 : Visualization - Mohamed-Fouad FELLAH, Nelly LAM*

## 0.1 Background and motivation

Malaria is one of the most considerable and impactful parasitic disease that mostly affect children and pregnant women in sub-Saharan Africa and India. We chose this project because of the actual need it may be use for preventing the propagation of the disease. The goal of the classifier is to distinguish the patients harmed by malaria of the other who are not by analysing images of stings.

## 0.2 Data and problem description

Medichal is a project involving a classification problem. We have to determine if the cells are infected or not based on their pictures provided by data science students of Paris-Saclay University. In order to classify our cells, we already have several features.

## 0.3 Description

### 0.3.1 Duo 1

We are working on the preprocessing part of the project, it means that we have to "simplify" the data given by eliminating useless features (for the training), reducing the data dimension and detecting/removing potential outliers. These tasks will help our model go faster and get better results. First of all, we delete some features because we do not want the other steps of the preprocessing to take too much time due to the useless data, we will be able to go through less features for the outlier detection for instance. We are analyzing boxplots of features independently to decide whether they are useful or not. A feature is considered as useless if there isn't any correlation between the value of the feature are the infection of cell. In this case, the feature doesn't help detecting which cells are infected, and thus we can remove it. After that, we want to reduce the number of dimension for the same reasons as the feature selection. We will use PCA and TSNE algorithms from the scikit-learn library. Finally, we had to remove outliers, which are observations that are far from the others. This step is important because we want our model to fit the region where the training data is the most concentrated, in order not to be disturb by extreme values and thus have better results with true data. For that, we used the method "Local outlier factor" from sickit-learn, which calculates the anomaly score of each sample, measuring how isolated it is with respect to the surrounding neighborhood. We compare the local density of each sample to the local densities of its neighbors, and the ones which have a lower comparative density (beyond a certain threshold) are considered outliers, so we can remove them.

### 0.3.2 Duo 2

We decided to use different scikit-learn classification models and GridSearchCV to adapt the hyper-parameters. We started to look for the best hyper-parameters for each model. Then, we compared the results using cross-validation to avoid the phenomenon of over-learning. The main difficulties we encountered, were : the hyperparameters understanding and the ≪ not too long data training ≫ models search.

### 0.3.3 Duo 3

For our part, we decided to use models which derive from Manifold Learning, (we took some of our sources of scikit-learn in particular to study the learning scores). We tried to choose the most relevant models in order to have the best possible results. We started to study populations, using research methods (Neighbors Search), with the LLE method (Locally Linear Embedding) which comes directly from Manifold Learning. This method searches for a lower dimension projection of the data which preserves the distances in the local "neighborhoods". We can consider this as a series of analyze of local principal components which are globally compared to find the best nonlinear incorporation. This method is very practical mainly for its relevance. Then we used a meta-estimator which adjusts a number of decision tree classifiers on various subsamples among all of our data (cf. RFC - Figure 4). This method combines the result of multiple predictions which aggregates many

decision trees (RandomForestClassifier). We chose this method because it allowed us to obtain more precise prediction results according to our estimates. Finally, we used a representation of learning scores and validation scores of an SVM in order to study the learning scores and validation scores of an SVM for different values of the $\gamma$ parameter of the nucleus. This method was chosen because support vector machines (SVM) are a set of supervised learning methods which have several advantages, in particular the fact that it is a very effective method in large spaces. . In addition, this method uses a subset of learning points in the decision function (called support vectors), so it is also efficient in memory. Then we made the observations that for very low $\gamma$ values, we see that the training score and the validation score are low. This is called under-adjustment. Average $\gamma$ values will result in high values for both scores, this shows that the classifier is working quite well. On the other hand, if the $\gamma$ is too high, the classifier will be overloaded, so the training score will be good except that the validation score will be bad.(cf.Figure 3). We have reused the graph representing the plot of the ROC curve, this plot is a good indicator for us to find a good threshold for our model, indeed if we really want something close to 100% of true positives, we could consider a model giving us 40% false positive rate, so we could consider this threshold as a medical diagnosis.

## 0.4 Possible improvements

### 0.4.1 Duo 1

We would like to use the PCA method to reduce the dimensions

### 0.4.2 Duo 2

In order to improve our part of the project, we first wanted to test other models. We compared the most exhaustive list possible of the classification methods proposed by scikit-learn. We would also like to adapt the hyper-parameters better. Indeed, until today, we have for each model only modified two to three hyper-parameters (reasons : hyper-parameters understanding and time). In fact, each combination of hyper-parameters is tested by GridSearchCV : adding a hyper-parameter of which we test five values multiplies by five the number of combinations to try. In addition to that, if the interval in which we vary the values of our hyperparameters is badly chosen, certain models will take a long time to train. Concerning the hyperparameters, we would also like to display a curve of visualization of the results as a function of a hyperparameter, to see the effect of this one on the result of the model with the help of the duo 3. Maybe we will (we are not sure about that) add images of cells infected or uninfected by malaria in our dataset. We now count more to work on the version RAW of the project, not the PREPROCESSED version. Indeed, we have already tried to work on the RAW version of the project and obtained scores around 0.57 (which is extremely low). To improve this, we plan to try set methods, including Boosting, such as AdaBoost or by developing our own sets of models.

### 0.4.3 Duo 3

A problem encountered with the LLE method is the regularization problem. When the number of neighbors is greater than the number of input dimensions, we can then try to solve the problem by using several weight vectors in each district, so this is what the Modified Locally Linear Embedding (MLLE) method uses , in this case the MLLE can be executed with the local_linear_embedding function or its object oriented equivalent LocallyLinearEmbedding, with the keyword method = 'modified'. We will then take n_neighbors higher than n_components.

## 0.5 First results

We observed that the most efficient model was the random forest, which obtained a score on cross-validation around 0.95. A comparison of the results obtained without having previously set the hyper-parameters, with the results obtained by having adjusted them, showed us the importance of the hyper-parameters on the score of the algorithm in cross-validation : in particular for the decision tree, the naive Bayes method and the gradient descent. However, our tests on the RAW version of the project turned out to be much less successful, with scores of 0.7, at best.

# FIGURES PAGE

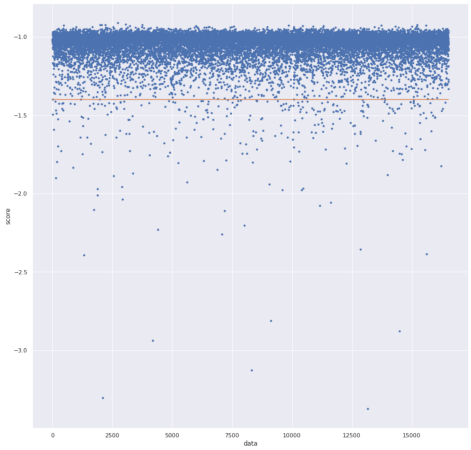


*Figure 1 : Outlier detection with anomaly score method*

In this graph, each sample is represented by a point, whose ordinate is his anomaly score. The lower a point, the lower its score is. We decided to fix the threshold at 1.4, it is represented with the red line. Samples that are below the red line have a score too low, they are too far from the others so they are considered outliers : we have to remove them.

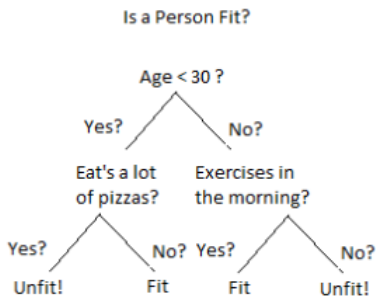



*Figure 2 : Example of Decision Tree*

A decision tree model is a non-parametric supervised learning method used for classification and regression. The series of questions and their possible answers can be organized in the form of a decision tree, which is a hierarchical structure consisting of nodes and directed edges.[...]The tree has three types of nodes :

- A root node that has no incoming edges and zero or more outgoing edges.
- Internal nodes, each of which has exactly one incoming edge and two or more outgoing edges.
- Leaf or terminal nodes, each of which has exactly one incoming edge and no outgoing edges.

In a decision tree, each leaf node is assigned a class label. The non- terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics.

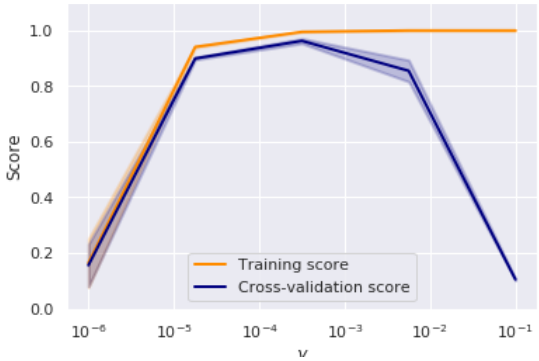



*Figure 3 : Validation Curve with SVM [2]*

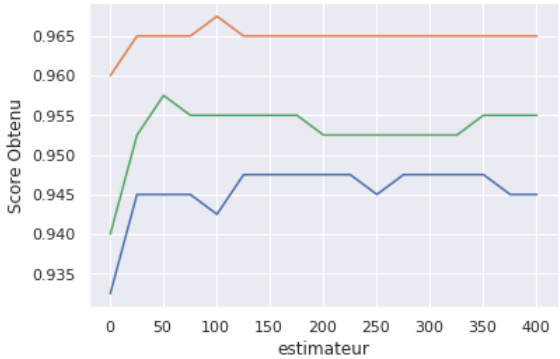



*Figure 4 : Meta-estimator (Method of RandomForrestClassifier) [2]*

# BONUS PAGE

| | modèles | score moyen | score maximal | score minimal |
|---|---|---|---|---|
| 0 | Nearest Neighbors | 0.697480 | 0.706089 | 0.682488 |
| 1 | Random Forest | 0.951350 | 0.953588 | 0.948888 |
| 2 | Decision Tree | 0.947150 | 0.950209 | 0.945229 |
| 3 | Perceptron | 0.547366 | 0.572991 | 0.500000 |
| 4 | Naive Bayes | 0.883293 | 0.903002 | 0.858672 |
| 5 | Gradient Descent | 0.670407 | 0.796540 | 0.558931 |

*Table 1 : Cross Validation score without fixing hyperparameters*

| | modèles | score moyen | score maximal | score minimal |
|---|---|---|---|---|
| 0 | Nearest Neighbors | 0.696230 | 0.699446 | 0.691218 |
| 1 | Random Forest | 0.951332 | 0.954856 | 0.947933 |
| 2 | Decision Tree | 0.926831 | 0.929515 | 0.922349 |
| 3 | Perceptron | 0.547366 | 0.572991 | 0.500000 |
| 4 | Naive Bayes | 0.801839 | 0.872104 | 0.701669 |
| 5 | Gradient Descent | 0.551693 | 0.564860 | 0.526335 |

*Table 2 : Cross Validation score with better hyperparameters*

In these two tables, we display the average, minimum and maximum scores for each model : first on a model whose hyper-parameters have been fixed and then, on the same models with non-worked hyper-parameters. We observe the incidence of hyper-parameters on the model score. The impact seems particularly strong on the decision tree, the naive Bayes method and the gradient descent, whose scores increase a lot with good hyper-parameters. In addition, we note that certain models, like the perceptron or the descent of gradient, obtain very varied scores with a minimum score around 0.5 and a maximum score around 0.82.

# ALGORITHMS AND REFERENCES PAGE

**Algorithm 1** Random Forest

**Precondition:** A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features $F$, and number of trees in forest $B$.

```
function RandomForest(S, F)
    H ← ∅
    for i ∈ 1, ..., B do
        S^(i) ← A bootstrap sample from S
        h_i ← RandomizedTreeLearn(S^(i), F)
        H ← H ∪ {h_i}
    end for
    return H
end function
function RandomizedTreeLearn(S, F)
    At each node:
        f ← very small subset of F
        Split on best feature in f
    return The learned tree
end function
```

The random forests model is an ensemble learning method. It operates by constructing a multitude of decision trees at training time and outputting the average of all the classes. This model is way better than decision trees because it corrects the overfitting problem of this method.

**References :**


[1] Mini Project L2 Data Science Course : slide by Adrien Pavao
[2] Scikit-Learn : Python library for machine learning
[3] Wikipedia : Ensemble Learning
[4] Matthew N. Bernstein Postdoctoral Fellow at the Morgridge Institute of Research : Random Forests
[5] Vipin Kumar : Classification: Basic Concepts, Decision Trees, and Model Evaluation