Census income classification

louis TOMCZYK[1]

[1]Artificial Intelligence Department, Xidian University

December 28, 2021

Abstract

Machine learning is the study of algorithms which can automatically improve their performance at a given task through experience. The goal is to create a mathematical model that, based on experience with `training data`, can predict the output of new and previously unknown data. Among all the possible cases, the one of `labelled data` is refered as `supervised learning`. In this paper we aim to give a general methodology to address a machine learning task and more specifically a two-labels classification problem through the study of the well known dataset : *the census income*. Here we are going to show the fundamental importance of the `pre-processing` step for the `performance evaluation`. We will also see that it is not the only step needed to reach higher performance and then the importance of the `hyper-parameters optimisation` which can lead to untimely return journeys between the last three steps. Eventually we show the need to take into consideration the `running time` in order to make the algorithm quicker and more efficient.

1 Introduction

Tom MITCHELL **[1]** gives in 1998 the property of a well-posed machine learning problem as :

A computer is said to learn from experience with respect to some tast and some performance measure if increasing experiences implies an improving performance at doing the task.

In classification, samples belong to two or more classes and we want to learn, from labelled data, how to predict the class of unlabelled data. Here the task is to predict if a person earns more or less than $50,000$ $US\$/year$. This census dates back to 1994 and the authors **[2]** based their predictions on the following features:

- **age :** numerical
- **workclass :** categorical
- **fnlwgt :** standing for "final weight" : numerical
- **education.num** numerical
- **marital.status :** categorical
- **occupation :** categorical
- **relationship :** categorical
- **race :** categorical
- **sex :** categorical
- **capital.gain :** numerical
- **capital.loss :** numerical
- **hours.per.week :** numerical
- **native.country :** categorical

This census was studied using different supervised learning methods such as `Gaussian Naive Bayes (G-NB)`, `k-Nearest Neighbours (k-NN)`, `Tree Classifiers` (C4.5), or `Random Forest Models` etc. (see table 1). In order to evaluate the performance, the authors used `accuracy` which is based on the `confusion matrix` drawn at the end of the validation step of the algorithm. The error is given by equation 1 and is our reference to judge the work presented in this paper.

$$error = 16.11 \pm 2.2\ \% \tag{1}$$

Here we performed the classification using three techniques: k-NN, `Support Vector Machine (SVM)` and `Gaussian Naive Bayes`. We first give a general methodology to address a general machine learning problem. Then we give the minimum required background of each model we are using to see what are the `hyper-parameter` we can

tune to improve the performance metrics. Finally show the results we got to compare it with the previous work. We want to show the importance and the impact of the analysis and pre-processing steps of the methodoly on the performance metrics.

Algorithm	Error (%)
C4.5	15.54
C4.5-auto	14.46
C4.5 rules	14.94
Voted ID3 (0.6)	15.64
Voted ID3 (0.8)	16.47
T2	16.84
1R	19.54
NBTree	14.10
CN2	16.00
HOODG	14.82
FSS Naive Bayes	14.05
IDTM (Decision table)	14.46
Naive-Bayes	16.12
Nearest-neighbor (1)	21.42
Nearest-neighbor (3)	20.35
OC1	15.04

Table 1: Performance sum up of previous work

2 Methodology

2.1 General methodology

There is a general methodology **[3]** that we must follow when we want to solve a problem using machine learning techniques. This methodology reduces the possible sources of error and noise, which allows us to obtain results with a higher reproductibility. This methodology is made of five steps :

1/ data importation : obvious.

2/ data analysis : cleaning the data of all the possible noise sources (errors, missing elements etc.), data visualisation.

3/ data pre-processing : Normalisation, label encoding, features extraction, separation of the initial set into two subsets, one is the **training set** to train the model. The second is the **test set** for testing to improve the model.

4/ training getting the major parameters of the model and testing it by predicting the targets of the test set.

5/ performance evaluation : estimating the performance of the model and adjusting the hyper-parameters. and improve the model

However the latter is a bit more complex as shown in figure 1. Indeed, if the performance is not high enough, we have to change the training step by tuning the hyper-parameters[1] of the model. If this work is not enough, we have to change the pre-processing step by using another normalisation or features engineering methods. As a last resort, we may also go back to the analysis step and improve the method used to deal with the errors and the missing values. Eventually one may intervert the analysis and pre-procesing steps when, in order to visualise the data or make denoising (analysis step), we must encode first the data (pre-processing step).

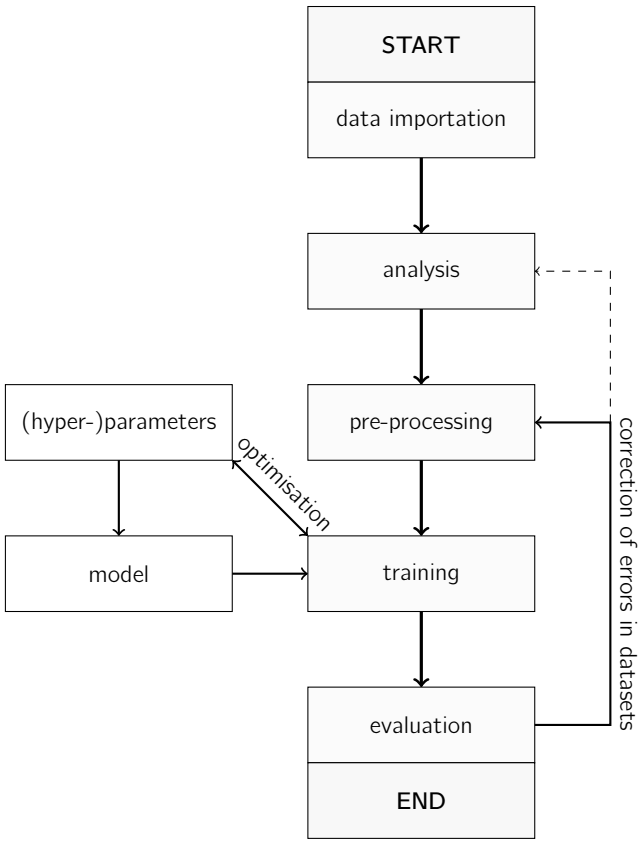


Figure 1: Methodology for addressing problem using machine learning techniques

2.2 Testing methodology

In order to perform the training and validation steps, we have to split our dataset into two sets. One for training, the other one to test our model. Intuitively in the former

[1] parameters not learned by the model, given by the user

case, its size should be as large as we can and larger than the testing set. A common rule is two take two-thirds for the training and one-third of the testing sets.

However, some machine learning techniques use random procedures as initialization steps. Thus, depending on the realisation, the performance can be more or less good. If we take only one realisation as training leading to a bad performance result, it does not necessarily means that the algorithm is not working. It might only mean that the randomized intialisation was not good. But it has nothing to do with the general performance of the algorithm. In order to face this issue, the *k-folds cross-validation* was created **[4]**. It consists in dividing the dataset into (k) folds. Then the $(k-1)$ folds are used for training and the last one for testing. Afterward, we modify the test fold $(k-1)$ times and repeat the training, in total $(k-1)$ times. Finally, we derive statistics from these trials and we can obtain more meaningful conclusions about the effectiveness of the algorithm.

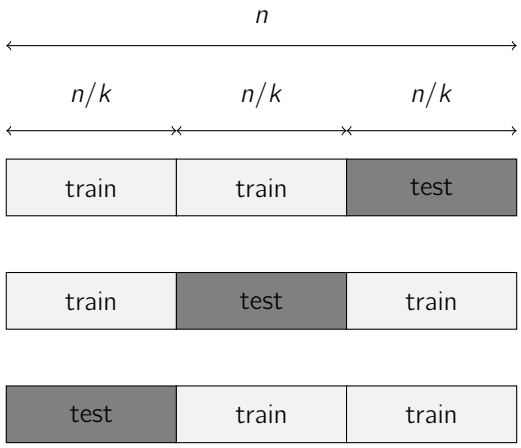


Figure 2: Schematic principle of the k-fold cross-validation with $k = 3$

2.3 Performance evaluation methodology

In classification, according to the number of targets, we may use different metrics. However many of them are using the confusion matrix. The most basic case is the two-labels classification. We look for making predictions equal to the actual values. Here the two labels are $\{\leq 50K, > 50K\}$ we encode by $\{0, 1\}$. Thus we define four main classes :

True Positive (TP)**:** number of samples correctly identified as positive. More precisely, they are the samples identified as equal to $1 \equiv > 50K$ which are actually equal to 1.

True negative (TN)**:** number of samples correctly identified as negative. More precisely, they are the samples identified as equal to $0 \equiv \leq 50K$ which are actually equal to 0.

False Positive (FP)**:** number of samples identified as positive while they are negative. More precisely, they are the samples identified as equal to 1 while they are actually equal to 0.

False Negative (FN)**:** number of samples identified as negative while they are positive. More precisely, they are the samples identified as equal to 0 while they are actually equal to 1.

From these four classes, we can draw the confusion matrix, given by the figure 3.

		actual	
		1	0
predictions	1	TP	FP
	0	FN	TN

Figure 3: Confusion matrix for two-labels classification

Given this confusion matrix, the most general metric to globally evaluate a classification algorithm is the accuracy defined as the ratio of correctly identified labels.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

3 Analysis and Pre-processing

3.1 Errors and noise sources

First, as we do not have any information on how the data were collected, we assume that the census does not contain any error. However, it does not avoid the missing values which can appear in all the features. In this work, we choose to simply remove all the data samples for which any feature is missing. The original dataset contains $32,561$ ssamples, and there are $2,399$ samples with missing features. Which reduces the dataset size by 7%. Unless specified, all the results were obtained by performing 5-folds cross-validation. This means that four of the five folds contain $6,032$ and the last one contains $6,034$ samples.

The figure 4 shows the statistics of the raw data by using box plots. We can clearly see that the *fnlwgt* distribution

[2]The sum of the the values of this feature gives six billions which is, indeed approximately the world population in the 90s **[5]**

shows a much higher mean value around $2 \cdot 10^5$. Indeed, this feature aims to give the number of people in the world (in the 90s) represented.[2] To illustrate it, we can consider sample 11.

sample	age	fnlwgt	education	income
11	45	172,274	Doctorate	>50K

Here the final weight means that the authors estimate that there are 172, 274 people in the world with more or less the same characteristics. It explains then why the values are much higher for this feature compared to the other ones which have a maximum mean value around $1 \cdot 10^3$ (for the *capital.gain* feature). We can also notice a large number of outliers for these two features (circles) which is likely to cause issues when training the data.

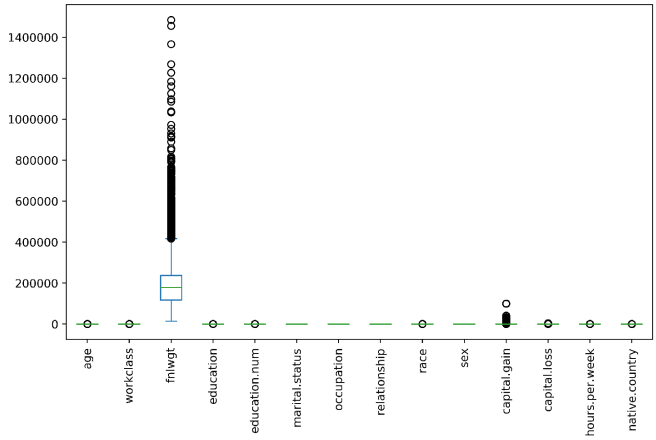


Figure 4: Statistics of the raw data

3.2 Scaling

3.2.1 Need

Scaling refers to a method used to normalize the range of independent variables or features of data.

According to the machine learning algorithm, the hidden mathematical foundations can vary significantly. For some, scaling or not data can have various impact on the performance evaluation or the `running time`.

For example the $k - NN$ algorithm is based on distance calculations. As features like *fnlwgt* shows much higher values than in any other feature, the distance (d) between those will be not change much :

$$d(x, x_0) \underset{x \to \infty}{=} x + \circ\left(\frac{x_0}{x}\right) \tag{3}$$

And the other features impact will be null. In the case the `neural networks` using `gradient descent`-like algorithm to compute the `loss function`, the figure 5 shows the role of scaling the data on the convergence speed of the algorithm. It represents a view from the top of an loss surface. In the scaled features case, the algorithm converges quicker, because of fiewer iterations, than on the unscaled case or not converge at all.

3.2.2 Methods

There are many ways to scale the data **[5]**. In practice we do not consider the statistical distribution shape and scaling boils down to centering, by removing the mean value ($\langle \bullet \rangle$), then normalising by dividing by the standard deviation ($\sigma(\bullet)$).

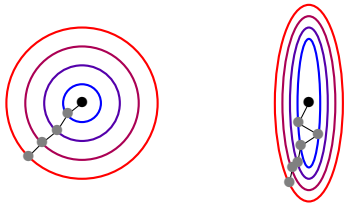
$$X_{scaled} = \frac{X - \langle X \rangle}{\sigma(X)} \tag{4}$$

This scaling is called `standard scaling`. It can be done when the ranges of some features are very different compared to others, like the *fnlwgt* case. However in other cases, like in image processing if we want to improve the contrast, we might just want to rescale the features (pixel value in the example) between a minimum and maximum.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5}$$

This scaling is called `min-max scaling`. However these two techniques cannot deal with `outliers`. Indeed, the outliers can significantly modify the empirical means and standard deviations. As they are measurement points, it would be wrong to remove them.

A way to deal with these data samples, is to perform a `robust scaling` which consists in centering, and standardize according to the `quantile range` : between the $25th$ and $75th$ quantile. Quantiles are less subject to outliers as long as the number of outliers is small compared the dataset size. All these operators belong to the `affine transformers` family. Other `non-linear transformers` exists, like the `Tuckey ladder of powers` but are not studied here.



scaled features unscaled features

- • : minimum of the loss surface.
- • : loss function calculation at each iteration.
- – : contour lines of the loss surface.

Figure 5: Schematic view of impact of (un)scaled features on the gradient descent convergence.

3.2.3 Materials

The results given in this paper were obtained with a computer with the following characteristics.

programming language : python 3.8.10 - 64bit

memory : 15.5 GiB

processor : Intel® Core™ i7-10510U CPU @ 1.80GHz x 8

OS name - type: Ubuntu 20.04.1 LTS - 64-bit

3.2.4 Experiments and results

To illustrate the scaling effect, we took three different algorithms and we tunned one hyperparameter using cross-validation, whose informations are given in table 2.

algorithm	hyperparameter	values
KNN	number of neighbours	$[\![2, 15]\!]$
SVM	kernel - poly degree	$[\![2, 4]\!]$

Table 2: Details of algorithms used to show impact of scaling.

The table 3 gives the results of the impact of the scaling on the running time. The question mark "**?**" means that the algorithm does not converge or takes more than two hours.

Firstly we can see that the impact is globally the same on both algorithms.

1/ the accuracy is equivalent for all the algorithms, around 0.75

2/ the running time can be whether higher or lower when using scaled data or not

3/ the deviation can also be whether higher or lower when using scaled data or not

		scaled		not scaled
	times [ms]	robust	standard	
k-NN	fit time	315 ± 9	209 ± 5	66 ± 5
	accuracy	0.738	0.73	0.73
SVM	fit time	**?**	$(34.1 \pm 5.7) \cdot 10^3$	$(63.1 \pm 11.7) \cdot 10^3$
	accuracy	**?**	0.75	0.75
G-NB	fit time	4.3 ± 0.4	4.7 ± 0.9	4.6 ± 0.2
	accuracy	0.748	0.747	0.748

Table 3: Results of running time and accuracy for different scaling methods

The first point clearly means that if we care only about the performance, scaling is useless for this dataset. But the second and third points need to be more precise. We can see, using the k-NN, that scaling the data increases the running time by almost 4 :

$$\frac{\text{fit time}_{scaled}(kNN)}{\text{fit time}_{unscaled}(kNN)} = \frac{315 + 209}{2 \cdot 66} = 3.97 \qquad (6)$$

Scaling has no impact using G-NB however for SVM it depends on the scaling techique.

$$\frac{\text{fit time}_{scaled}(GNB)}{\text{fit time}_{unscaled}(GNB)} = \frac{4.3 + 4.7}{2 \cdot 4.6} = 0.97 \qquad (7)$$

The standard scaler improves the performance for the SVM in term of mean and deviation for the fit time by almost 2.

$$\frac{\text{fit time}_{scaled-standard}(SVM)}{\text{fit time}_{unscaled}(SVM)} = \frac{34.1}{63.1} = 1.85 \qquad (8)$$

But the robust scaler degrades significantly the performance as the algorithm does not, or too slowly, converge. Indeed it took at least two hours while using the standard scaler took less than twenty minutes. Eventually for both accuracy and running time, the Naive Bayes algorithm is the best.

3.2.5 Conclusion

We can understand the negative impact of the scaling for k-NN as it is based on distance calculations. Due to the specific range values of $fnlwgt$, using the equation 3, we can conclude that distance computation behaves as if there was only one dimension.

However for the SVM we see improvement by using standard scaler as the margin calculations also use distance. Which does not work with the robust scaler as it keeps the

outlier values which are the cause of margin calculation issues.

Indeed, the SVM optimization problem is the calculation of the best margins equations. In an appropriate vector space, their equation can be written :

$$\langle w, X \rangle + b = 0 \tag{9}$$

where (w, b) are the coordinates of the normal vector of the margin, and (X) the features vector. The objective if to give (w^*) the optical vector in order to minimise the Lagrangian. We can show [7] that :

$$w^* \propto \langle X, y \rangle \tag{10}$$

where (y), the target vector. The optimal Lagrangian (L^*) is given by :

$$L(w^*) \propto \sum y_i \cdot y_j \cdot \langle X_i, X_j \rangle \tag{11}$$

Thus we can understand with brief uncomplete formula, that if the features are not scaled, or at least, still with the outliers, the dot product $\langle X_i, X_j \rangle$ can have two consequences.

The first is to give coefficient leading to non-convex surfaces and consequently with local minima. The second is that this dot product will give values in a wide range in all $(\mathbb{R})$. In this case can appear the floating-point arithmetic issues leading to dramatic calculation errors.

Finally for the GNB algorithm, we see that scaling does not impact much the performance. Indeed, it is based on the probability theory. As statistcal studies aim to derive probability laws, scaling the data will not change the distribution shape. No changes in the latter lead to no changes in the probabilities.

Thus this dataset appears to be the best example to show the deep impact of scaling or not the data.

4 Training and Evaluation

4.1 Chosen algorithms

We decided to fully study three algorithms :

k-Nearest Neighbours : to compare our results with those given in the dataset description The hyperparameter to tune is the number of neighbours

Gaussian Naive Bayes : same purpose. The `Lidstone` (or smoothing) hyper-parameter is tuned.

Support Vector Machine : to see if we can reach better performance with another algorithm also based on distance calculations. The hyperparameters we tune are : the kernel type and its depending parameters.

4.2 Implementation

For all the training work, we used `SciKit Learn` libraries which offers a wide range of functions to perform machine learning tasks. This choice is done to avoid any problem linked to the coding and implementation of the algorithms as the choice is to concentrate on the data processing and the hyper-parameters role in the performances. However the performance evaluation are custom functions as well as results display and report generation.

We tuned the hyper-parameters by performing a `grid-search`. This technique aims to perform an exhaustive search over specified parameter values for an estimator. It provides the running and scoring time needed at each step, and the performance metrics we desire, here only the accuracy. The uncertainty will be simply given by the standard deviation. This will avoid any discussion about the `Student law` and the precision degree desired.

4.3 Results

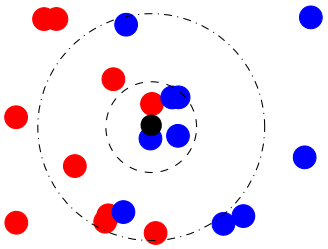
4.3.1 Use of scaling

All the global results about the scaling comparisons are summarized in the table 3. We can already state that the use of k-NN does not need the scaling. While the SVM needs standard scaling and GNB needs robust scaling.

4.3.2 k-NN.

The k-NN is one of the simplest classifcation algorithms which relies on the neighbourhood study of the point we want to classify. If the majority of its neighbourhood belongs to a certain class, then it will be identified as belonging to this class, see figure 6.

This decision is only depending on the number of the neighbours. We can see that taking the 5 neigbours will lead the data sample to belong to the red class. Whereas if we take 13 neigbours, we will classify it as blue.



$-.$: $k = 13$
$--$: $k = 5$

Figure 6: Schematic principle of the k-NN algorithm.

Here we performed the grid search two times, leading to ten runs of the algorithm for each value of (k). The accuracy uncertainty is the same for all (k) and is inferior to 10^{-3}. The table 4 give the results of the number of neighbours tuning. We can see that both mean accuracy and mean running time show small deviation rate (< 5 %) which is defined as :

$$\text{deviation rate} = \frac{\sigma(\bullet)}{\langle \bullet \rangle} \tag{12}$$

We can deduce from the results that the optimal value for the number of neighbours is $k = 14$ as it gives at the same time, the mean and deviation of the fit time : 63.1 ± 3 $[ms]$. As well as the best accuracy value 0.75.

4.3.3 SVM.

The main idea of SVM is to find `hyperplanes` which discriminates all the classes, see figure 7. In one dimension, the hyperplane is a point. The goal is to find the optimal hyperplane among the possible infinity of hyperplanes (in green).

Figure 7: example of dataset for SVM

However in some cases it is impossible to find even one hyperplane with the raw data, see figure 8. We can then transform the data using a `projection function` (Φ)[3].

which will project the data into a space of higher dimension, in which we might find an hyperplane.

neighbours	fit time [ms]	accuracy
2	73.1 ± 16	0.72 ± 0
3	64.5 ± 4	0.68 ± 0.01
4	64.4 ± 3	0.73 ± 0
5	64.7 ± 3	0.70 ± 0
6	66.5 ± 5	0.73 ± 0
7	65.7 ± 4	0.72 ± 0
8	64.2 ± 3	0.74 ± 0
9	66.6 ± 6	0.73 ± 0
10	67.4 ± 5	0.74 ± 0
11	64.3 ± 5	0.73 ± 0
12	67.6 ± 7	0.74 ± 0
13	63.9 ± 3	0.74 ± 0
14	63.1 ± 3	0.75 ± 0
15	64.7 ± 3	0.74 ± 0
deviation rate %	3.8	2.5

Table 4: Results of the k-NN hyper-parameter tuning.

If we have (p) scalar features, and (n) samples, the projection function is defined by :

$$\forall (n,p) \in (\mathbb{N}^*)^2 \quad \Phi : \begin{pmatrix} \mathbb{R}^{n,p} & \to & \Phi(\mathbb{R}^{n,p}) \\ X & \mapsto & \Phi(X) \end{pmatrix} \tag{14}$$

In the figure 9, we solved the issue by applying a polynomial projection of degree 2, which enables to find this time an hyperplane of dimension 1 (in green).

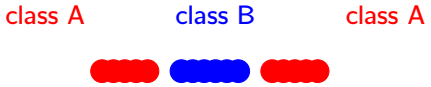


Figure 8: example of dataset for SVM showing the need of a kernel

[3] In the litterature we can get confused with the names leading to calling *kernel* the function which transform the data. More exactly the kernel function comes from the optimization problem **[8]** which is a `Lagrangian minimisation problem`. The optimal hyperplane, in the projected space, can be written as $(\mathcal{H}) : \langle w, X \rangle + b = 0$, where (w) is linked to the kernel function defined as :

$$k : \begin{pmatrix} \Phi(\mathbb{R}^p) \times \Phi(\mathbb{R}^{n,p}) & \to & \mathbb{R} \\ \Phi(X_i), \Phi(X) & \mapsto & \langle \Phi(X_i), \Phi(X) \rangle \end{pmatrix} \tag{13}$$

Where (n) is the number of samples, (p) is the number of features and $i \in [\![1, n]\!]$

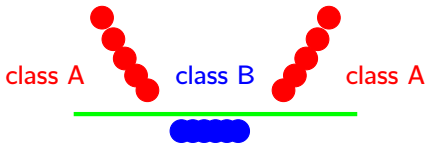


Figure 9: Same dataset as in figure 8, with a polynomial kernel (degree 2).

For the training we performed only one grid search with a 5-folds cross validation. The support vector machine hyper-parameter is the kernel. Here we choose to study the three major.

The `polynomial` projection function has three hyper-parameters (γ, c, d).

$$\Phi(X) = (\gamma \cdot X + c)^d \tag{15}$$

Here due to too long computation times, we only tunned the degree, and $(\gamma, c) = (1, 0)$ The results are given in table 5. We see that the a second-order polynome suits best for the task by showing both best running time and accuracy. We also see that changing the degree can significantly change the running time, up to three times ($d = 5$) even if the accuracies values are equivalent.

The `Radius Basis Function (RBF)` is a gaussian transformation with only one hyper-parameter similar to a standard deviation (γ). The results are given in table 6.

$$\Phi(X) = exp\Big(-\gamma \cdot X^2 \Big) \tag{16}$$

We see that $\gamma = 0.1$ suits best for the task by showing best running time, and a slightly smaller accuracy. Once again we notice similar accuracy values. This time the running time can up to two times bigger ($\gamma = 10$).

Eventually, a possible `sigmoid` projection function is the hyperbolic tangent[4] which shows two hyper-parameters (γ, r).

$$\Phi(X) = tanh\Big(\gamma \cdot X + r\Big) \tag{17}$$

Again due to too long computation times, we only tunned the (r) coefficient, and $\gamma = 1$. The results are given in table 7. This time the accuracy values are significantly changing, up to 20 % from $c = 1$ to $c = 10$. But running times are quite similar. We see that $c = 10$ suits best for the task.

degree	fit time [ms]	accuracy
2	$(4.5 \pm 0.1) \cdot 10^3$	0.749 ± 0
3	$(5.9 \pm 0.4) \cdot 10^3$	0.749 ± 0.01
4	$(6.0 \pm 0.2) \cdot 10^3$	0.746 ± 0
5	$(12 \pm 0.9) \cdot 10^3$	0.748 ± 0

Table 5: SVM polynomial kernel degree tuning results.

γ	fit time [ms]	accuracy
0.1	$(26 \pm 0.3) \cdot 10^3$	0.748 ± 0
1	$(37 \pm 0.4) \cdot 10^3$	0.75 ± 0
10	$(43 \pm 3.2) \cdot 10^3$	0.75 ± 0

Table 6: SVM RBF hyper-parameter tuning results.

r	fit time [ms]	accuracy
0.1	$(8.2 \pm 0.9) \cdot 10^3$	0.629 ± 0
1	$(8.4 \pm 0.5) \cdot 10^3$	0.627 ± 0
10	$(9.6 \pm 0.6) \cdot 10^3$	0.752 ± 0

Table 7: SVM sigmoid kernel hyper-parameter tuning results.

Eventually the table 8 summarizes the results by taking the best hyper-parameter of each projection function.

kernel	fit time [ms]	accuracy
polynomial	$(7.1 \pm 0.4) \cdot 10^3$	0.748 ± 0
rbf	$(35 \pm 1.3) \cdot 10^3$	0.749 ± 0
sigmoid	$(8.7 \pm 6.6) \cdot 10^3$	0.752 ± 0

Table 8: SVM kernel tuning results.

We can first notice that the sigmoid function used leads to the worst case. Both polynomial and rbf projection functions show 12 % accuracy improvement. We can also notice that the same sigmoid also leads to the worst deviation which can be up to 17 times bigger than the optimal solution. Eventually the optimal case is reached using second order polynomial function.

The last remark which can be done is the average hundred times slower than the k-NN.

[4] *Sigmoid* refers to a mathematical family of functions. This is why we use *a possible ...* instead of *the....* The hyperbolic tangent is the only one already implemented in SciKit Learn library.

4.3.4 G-NB.

The Gaussian Naive Bayes has one hyper-parameter linked to the feature statistical distribution smoothing. Assume we have $c \in \mathbb{N}^* \backslash \{1\}$ classes to predict $\mathcal{C} = \{C_1, \cdots, C_c\}$ and $p \in \mathbb{N}$ features $X = \{x_1, \cdots, x_p\}$.

The probability of a sample to be classified in $(C_k), k \in [\![1, c]\!]$, knowing its features is :

$$\mathbb{P}(C_k|X) = \frac{\mathbb{P}(C_k)}{\mathbb{P}(X)} \cdot \prod_{j=1}^{p} \mathbb{P}(f_j|C_k) \tag{18}$$

However, at first sight[5], if we have at least one feature such as $\mathbb{P}(f_j|C_k) = 0$ then the probability $\mathbb{P}(C_k|X) = 0$.

We can write this probability as :

$$\mathbb{P}(f_j|C_k) = \frac{|f_j, C_k|}{|f_j, \mathcal{C}|} \tag{19}$$

Where :

$|f_j, C_k|$: number of occurences of the feature value (f_j) in the class (C_k)

$|f_j, \mathcal{C}|$: number of occurences of the feature value (f_j) in all the classes

Thus $\mathbb{P}(f_j|C_k) = 0$ if there are no occurence of the feature value in the class we want to predict. Pragamaticaly, if we look at the class $> 50K$, there is no age value below 17. However if we take into account the case of young artists (singers for example), some can be below 17 while they make more than $50,000\ US\$/year$.

In such case, we can smooth the probability by performing a Lidstone smoothing **[10]** by adding $\delta \in \mathbb{R}^+$ such as[6] :

$$\tilde{\mathbb{P}}(f_j|C_k) = \frac{|f_j, C_k| + \delta}{|f_j, \mathcal{C}| + \delta \cdot p} \tag{20}$$

Pragamaticaly, the smoothing boils down to redistribute the probability of higly frequent classes to less frequent classes. Or, in other words, decrease the effect of outliers. Here we choose one hundred values of $\delta \in [10^{-5}, 10^5]$. And all give the same accuracy value of 0.75. The running times are showed in figure 10. We can see that, compared to the default value (when the smoothing variable is not set) of 10^{-9}, the fit times is slightly bigger. So we can deduce that the best case, for this dataset, is to not specify any value for the Lidstone parameter.

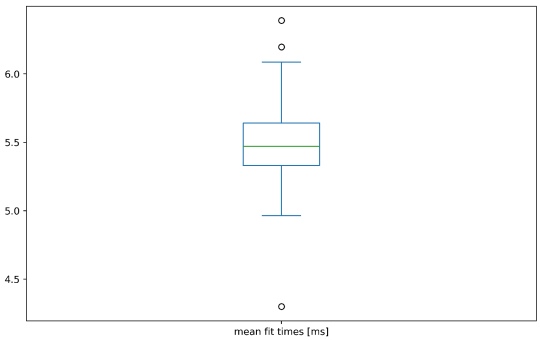


Figure 10: Fit times distribution for the GNB classifier.

5 Conclusion

In this paper we tried to show the impact of the preprocessing and hyper-parameter tuning on the performance of the classification task. Globally, all algorithm together, the average error is 25 ± 0.01 % instead of 16 ± 2 % in the previous work. It represents 50 % of difference. This difference can have many sources. The first one could be the starting assumption of ignoring all the data samples with any missing features. Keeping these values and performing some work on it, like filling by the mean values or other techniques are well known, could have changed the performances. The second could be a deeper study of some hyper-parameters like in the SVM using RBF kernel, by spending time on the Gram matrix **[8]** which gives informations on the correlation on the data samples. A third one could be the wrong scaling method. We mentionned the existance of non-linear techniques as the Tuckey ladder powers, but many other also exists.

This anormal difference also shows the importance of a machine learning important field which is the features engineering. Vice versa, the features engineering work alone, without the scaling and hyper-parameters tuning cannot give the optimal performance. The features engineering would consist in studying the importance of each features in the training. The ideal would be to get an expert in the domain, here it is economy and sociology, which will be able to help to focus our work on some features for example. However, without an expert, one can still perform the study by using methods usually used in clustering tasks, as the Principal Component Analysis (PCA) which studies the correlations between the features.

[5] *At first sight* as **[9]** shows that the model is more sophisticated
[6] The added $\delta \cdot p$ is not keep a probability inferior to one.

Bibliography

[1] **Andrew NG**, *Machine learning*, Coursera, https://www.coursera.org/learn/machine-learning, [Accessed : December 28, 2021]

[2] **Ronny KOHAVI, Barry BECKER**, *Census Income Data Set*, UCI Machine learning repository, https://archive.ics.uci.edu/ml/datasets/census+income, 1994, [Accessed : December 28, 2021]

[3] **Louis TOMCZYK**, *Homework for MultiLayers Perceptron*, Xidian University, November 2021

[4] **SciKit Learn**, *Cross-validation: evaluating estimator performance*, https://scikit-learn.org/stable/modules/cross_validation.html [Accessed : December 28, 2021]

[5] **Bureau of the census**, *The world at a glance : 1994*, Statistical brief, March 1994, https://www.census.gov/prod/1/statbrief/sb94_4.pdf [Accessed : December 28, 2021]

[6] **SciKit Learn**, *Preprocessing data*, https://scikit-learn.org/stable/modules/preprocessing.html [Accessed : December 28, 2021]

[7] **Chloé-Agathe AZENCOTT, Yannis CHAOUCHE**, *Entraînez un modèle prédictif linéaire - Maximisez la marge de séparation entre vos classes*, [French], May 2021, https://openclassrooms.com/fr/courses/4444646-entrainez-un-modele-predictif-lineaire/4507841-maximisez-la-marge-de-separation-entre-vos-classes [Accessed : December 28, 2021]

[8] **Chloé-Agathe AZENCOTT, Yannis CHAOUCHE**, *Utilisez des modèles supervisés non linéaires - Classifiez vos donnée avec une SVM à noyau*, [French], February 2021, https://openclassrooms.com/fr/courses/4470406-utilisez-des-modeles-supervises-non-lineaires/4722466-classifiez-vos-donnees-avec-une-svm-a-noyau [Accessed : December 28, 2021]

[9] **Harry ZHANG**, *The Optimality of Naive Bayes*, Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, 2004, https://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf, [Accessed : December 28, 2021]

[10] **Wikipedia**, *Additive smoothing*, Decembre 2021, https://en.wikipedia.org/wiki/Additive_smoothing [Accessed : December 28, 2021]