POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Machine learning applications to credit risk analysis

Relatori
prof. Patrizia Semeraro
firma dei relatori

. . . . . .
. . . . . .

Candidato
Fabrizio Santoriello
firma del candidato

. . . . . .

Anno Accademico 2021-2022

Contents

List of Figures

List of Tables

Introduction

We can define Machine learning as a discipline whose purpose is to develop methods and algorithms that can "learn from experience", or more precisely, improve their performance concerning some tasks using suitable datas.

It has been an ever-expanding field over the past fifty years, and has found application in countless domains, improving every-day life. The economic one is no exception.

One of the most common ways to apply machine learning in finance is to investigate credit scores of loan applicants.

Credit scores were created to predict the relative risk of default among borrowers [Coffman and Chandler,], and is a measure of credit-worthiness of individuals. A correct estimate of them it is crucial, as evidenced by the 2008 sub-prime crisis, which can be reconducted to a failing in the credit risk estimation [Shi et al., 2022], among many other causes. Given these conditions, it is obvious that further development of machine learning techniques is fundamental.

In this work, two different dataset are taken in consideration: the first one is the well known German Credit Dataset, already used in several paper ([Wang et al., 2011, Tsai and Wu, 2008, Khashman, 2010] just to cite some). The second one is LoanP2P dataset, a collection of P2P loans made through the Lending Club platform (an American peer-to-peer lending company), from 2007 to 2020 . Both are made available by Kaggle.com, an online community of data scientists and machine learning practitioners.

Concretely the work is structured as follows below.

The first chapter has a theoretical cut, since it presents formally the statistical learning framework, and five of the most common machine learning algorithms: Logistic Regression, Naive Bayes, K-NN , SVM and Boosted-Bagged trees. Then, in the second chapter, they are applied to the previously outlined data-sets, and their performance compared, along with some considerations.

The chapter 3 covers a central role in this thesis, and may be considered the focal point, since ensemble classifiers are introduced.

Ensemble classifiers (**EC**) is an adaption of the well-known universal concept of *wisdom of crowds*, summarized as: "the Many Are Smarter Than the Few" [Suroweci, 2005].

EC is an emerging branch in artificial intelligence field, as it consists in combining different and heterogeneous predictions, generated by several methods,

into a single one.

The way outputs can be mixed has a crucial importance in the third chapter.

In the last one a concrete application of these procedures will be made, and the consequent results analyzed.

Finally, some general comments on the work done are made.

Chapter 1

Statistical learning

1.1 Theoretical introduction

[1] Speaking informally, the scope of statistical learning is to predict a quantitative or a categorical outcome measurement, starting from the value of different features. To do so, a model is constructed using a set of data, called training set, for which are known both the value of that quality and the value of the features used. In particular, using the training set, it's possible to infer the dependence of the input (the features used) on the output (the quality to predict), and then applying this model to new data, it's hopefully possible to make an accurate prediction on them.

More formally, in the statistical learning framework, we consider an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, and the pairs (X, Y) that belongs to $\mathcal{X} \times \mathcal{Y}$, where Y is a given response variable, and $X = (X_1, X_2, ..., X_p)$ a set of predictors.

Moreover we assume that $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ are random variable whose distribution is unknown, linked by a relationship that can be very generally expressed as:

$$Y = f(X) + \epsilon$$

where ϵ is a random error term with zero mean. Now it's clear that the purpose of statistical learning is to construct $\hat{f} : \mathcal{X} \to \mathcal{Y}$

$$\hat{f}(X) = \hat{Y}$$

which predicts $\mathcal{Y}$ from $\mathcal{X}$. Obviously a criterion for choosing f is needed, and seems natural to use a loss function $L(Y, f(X))$ that penalizes errors in prediction. Among the others, the most common one is the squared error loss:

$$L(Y, f(X)) = (Y - f(X))^2$$

[1] Section 1.1 and 1.2 written with reference to [Hastie et al., 2009]

This choice, since we are dealing with random variables, leads to define the expected squared prediction error:

$$EPE(f) = E(Y - f(X))^2 = \int [y - f(x)]^2 Pr(dx, dy)$$

Conditioning on X, allows us to write EPE as:

$$EPE(f) = E_X E_{Y|X}([Y - f(X)]^2|X)$$

So that is sufficient to minimize it point-wise:

$$f(x) = argmin_c E_{Y|X}([Y - c]^2|X = x)$$

the solution, as we should expect, is the conditional expectation:

$$f(x) = E(Y|X = x)$$

known as the regression function. Obviously we don't know the distribution of (X, Y), e so how to minimize analytically this expected condition.

Moreover, we can only access a small subset of the input space. Thus, the best prediction of Y given X = x is the conditional mean, when best is approximated by average squared error, measured on training set.

1.1.1 Bias-Variance trade-off

Assume that $\hat{f}(x)$ is the approximation of $f(X) = E(Y|X = x)$ obtained on some training data TR, then for a generic (x_0, y_0) it holds:

$$MSE(x_0) = E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + E(y_0 - \hat{f}(x_0))^2 + Var(\epsilon)$$

First, It can be notice that exist a lower bound for the MSE, since $Var(\epsilon)$ is an irreducible quantity.
Then, let's put in evidence that MSE is the quantity that we would get averaging the test MSE, obtained repeatedly estimating f using a large number of training sets, and tested at each x_0.

In this scenario the

$$Bias[\hat{f}(x_0)] = E(\hat{f}(x_0) - f(x_0))$$

refers to the error we make estimating a very complex , "real-life" problem with a simpler model. On the other hand $Var(\hat{f}(x_0))$ represent a measure of how much our estimate $\hat{f}(x)$ changes depending on the training set we use. Ideally we want both a low bias and a low variance, but in reality as one of the two decrease, the other increase. With a low variance-high bias method we risk to miss some important relations between training data and the response variable, while with high variance-low bias the risk is to model also the noise in the data. Concluding, the goal is to choose a model that captures important relations between X and Y, but generalizes well to new data.

1.1.2 The Classification Setting

In several applications, the response variable Y has a qualitative nature, that can be represented over a discrete set of classes. In this case the purpose of the model is to construct a classifier $C(X)$ that assign a class to unlabeled data X.

Suppose classes are $1, 2, ...K$, and define:

$$p_k(x) = P(Y = k|X = x), k = 1, 2, ..., K$$

the conditional class probabilities.

In this setting it's natural to define as loss function the *error rate*:

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

The *test error* associated with a set of observation $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ is given by:

$$Err_{Test} = Ave_{i \in Test} I[y_i \neq \hat{C}(x_i)]$$

In order to minimize this quantity it seems natural to assign each sample to the most likely class, given the relative features. Moreover we define the **optimal Bayes classifier** :

$$C(x) = \underset{j}{argmax}\ p_j(x)$$

Which precisely minimizes the *test error*. Apart from these differences, the concepts tackled in the previous sections still holds in a classification settings.

From now on, we will analyze this special case of statistical learning.

1.2 Model validation

As already outlined, it's obviously necessary to have a way to test the performance of the model generated, both for having an idea of the quality of the model and for choosing some suitable hyper-parameters.

Since the model ideally is capable of capturing trends and regularities in the training set, but also to generalize to new samples, it seems reasonable to avoid testing that model on the same dataset used to train it, since it would result probably in over-fitting the data.

A correct approach consists in testing on a dataset that the model has never seen before, the testing set.

In statistical learning selecting the hyper-parameters in such a way to minimize a certain measure or error is fundamental, and as we will see different options are possible.

1.2.1 Hold out set

If the purpose is just testing the performance of the model, a suitable approach is the following one. The dataset is split between train-set, the subset used as input for the model, and the test-set, where the algorithm is used to predict the class of each element, that consequently is confronted with the correct classification, so that it's possible to obtain a measure of the precision of the model.

In particular the dataset is divided randomly assigning samples to the train-set with probability equal to the percentage that we would like to split the dataset with. It's very fundamental to take stratified test and train-set, i.e. a subdivision that maintains the proportion between the two classes.

Commonly, several algorithms need to tune out one or more hyper-parameters, to achieve the best possible performance.

Using the the test-set to choose the parameters would result in an over-fitting, so a different option is needed.

One possible way to do that is to split the dataset into 3 subsets, instead of 2: the train-set, the test-set and the validation-set, using the latter to check how different parameters affect classification.

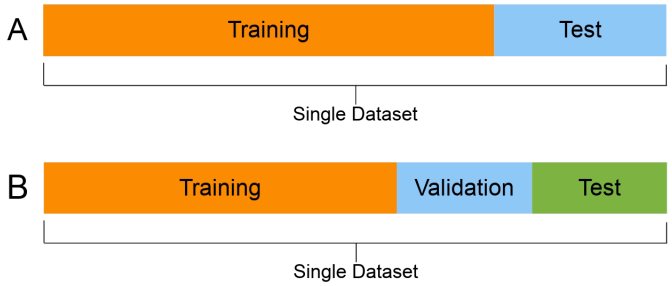


Figure 1.1: Hold-out and Validation set

1.2.2 Cross validation

A different way, that lowers variance (the variability in performance between different datasets) at the cost of greater computation time, is to use a K-fold validation.

It basically consists in splitting the train-set in K subsets, and use K_i subset for validation while the remaining parts for training. Then we iterate the process for $i = 1, 2, 3 \ldots, k$ and take a mean value for the different performance values obtained.

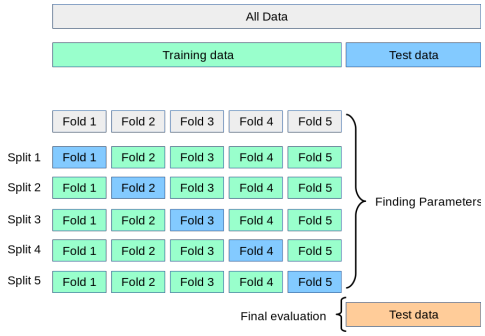


Figure 1.2: Cross validation

In a classification framework, we have that the K-fold cross-validation estimate of the error is:

$$CV_{(k)} = \frac{1}{K} \sum_{n=1}^{k} Err_i$$

where $Err_i = I(y_i \neq \hat{y}_i)$

This approach may be preferable if the dimension of the dataset aren't too large.

1.2.3 Performance measurement

In order to measure the quality of a model, a way to measure the classification performance is needed. In the classic case, where only 2 classes are present, a way to completely describe the performance of an algorithm, is the so called **confusion matrix**, that is a table representing the number of predicted class 0 and 1 elements, against their real class.

PREDICTION	STATUS	
	0	1
0	True Negative (TN)	False Negative (FN)
1	False Positive (FP)	True Positive (TP)

Table 1.1: Confusion matrix example

Moreover,there exists several measures that can summarize the information contained in that table. Probably the most intuitive one is the **accuracy**, that is the percentage of correctly classified samples.

$$Accuracy = \frac{TN + TP}{TP + FP + FN + TN}$$

but other way to do so are possible:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Precision. = \frac{TP}{TP + FP}$$

$$FalsePred. = \frac{TN}{TN + FN}$$

$$F_1 = \frac{2}{Precision^{-1} + Sensitivity^{-1}}$$

In general it's not correct to say that a measure is more complete that the others or vice-versa, since each one has his peculiar features and defects, that emerges especially when the dataset is unbalanced. For example, let's consider a case where out of 10 samples, 9 belong to *class* 0, and 1 to *class* 1. In such case, an algorithm that assign each element to *class* 0, would have an accuracy of:

$$Accuracy = \frac{9}{9+1} = 0.9$$

In other words, a performance that appears to be very good. Obviously an algorithm like that, can be considered acceptable, this example shows the limitations of this measure. Another possible choice is the so called **receiver operating characteristic curve**. The ROC curve represents intuitively the trade-off between *true positive rate* and *false positive rate* by plotting one against the other, as they vary in a particular setting.

In order to make things clearer, let's consider an algorithm whose output is the probability that a samples belongs to *class* 1, and not only the class (an example is the Logistic regression). In this case, the sample is assigned to *class* 1 is the probability is greater than a given threshold. That threshold is arbitrary, and varying it we obtain different *true positive rate* and *false positive rate* values. The ROC curve does exactly this, it plots the values obtained varying the threshold. Other than the Logistic regression that outputs exactly this probability, for other cases a way to obtain it are needed.

In addition we can intuitively summarize the information contained in the ROC curve, by using the AUC, that is the area under the ROC curve

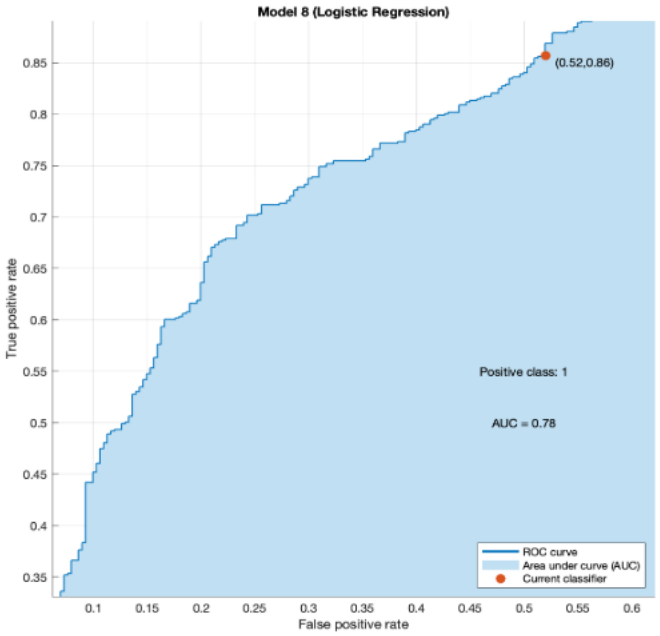


Figure 1.3: ROC curve example

1.2.4 Unbalanced data

Like already seen in the previous accuracy example, unbalanced data, can generate some bias in the model. To solve that, there are mainly to approaches:

- Under-Sampling: choose randomly a subset of the over-represented class,in such a way to equates the two. In this way some information is lost.
- Over-Sampling: duplicate samples from the class with less instances or simulate a instances based on the data available, so that the number of samples in each class are matched. There's the risk of overfitting the model.

Depending on the situation, both can be valid. A particular way to oversample datas, is the SMOTE method. SMOTE generates new samples in this way:

- Randomly picks a point x_1 among the minority class.
- Starting from that point, picks uniformly a K-neighbor (for example a point among the 5 nearest point), called x_2.
- Generates a new sample with a convex combination of the two points

  $$x_{new} = x_1 + \lambda(x_2 - x_1)$$

  With:

  $$\lambda \sim U(0, 1)$$

1.3 Classification

1.3.1 Logistic regression

[2] The first classification algorithm used is Logistic Regression which aims to model the probability that an observation belongs to certain class or not, or if it has a certain quality or not, in general any response variable with a binary nature, that in this situation we'll denote $\mathbf{Y}$.

Long story short, Logistical Regression describes

$$p\left(X\right) = ¶\left(Y = 1|\, X\right)$$

To do so, it models linearly (in fact is a generalization of the linear model) a transformation of $p\left(X\right)$, whose inverse in particular "squishes" $(-\infty, \infty)$ into $(0, 1)$, so that the probabilistic meaning on $p(X)$ can make sense.

Formally Logistic Regression estimates (using maximum likelihood, as a normal linear model) β_i such that [3]:

$$logit(p) = log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

more explicitly:

$$P(class = 1|X) = \frac{e^{\beta_0+\beta_1 x_1+...+\beta_n x_n}}{1 + e^{\beta_0+\beta_1 x_1+...+\beta_n x_n}}$$

Finally, in order to predict the class a data belongs, a threshold p_T can be set, then we label samples as follows:

If for a sample X the corresponding $P(X)$ is greater that P_T, we assign it to class 1, otherwise to class 1.

The natural choice for p_T may seems $p_T = 0.5$, but as we will see other options are possible, depending on the dataset analyzed. The parameters of the model, i.e the β, can be estimated using a Maximum Likehood Estimation (MLE) approach.

Since the logarithm is a strictly increasing function, and we are interested in maximizing the MLE, we can indifferently consider the $log - likehood$, obtained just taking the logarithm of the MLE, since this transformation won't change the $argmax$. For a generic model, depending of some parameters θ, the log-likehood $l(\theta)$ is:

$$l(\theta) = \sum_{i=1}^{N} \log p(x_i; \theta)$$

where $p(x_i; \theta) = Pr(y = 1|X = x_i; \theta)$. In the particular case of logistic regression, it becomes:

[2] section written with reference to [Hastie et al., 2009]

[3] we will indicate the number of factor with n to avoid using p, already used for denoting $p(X)$

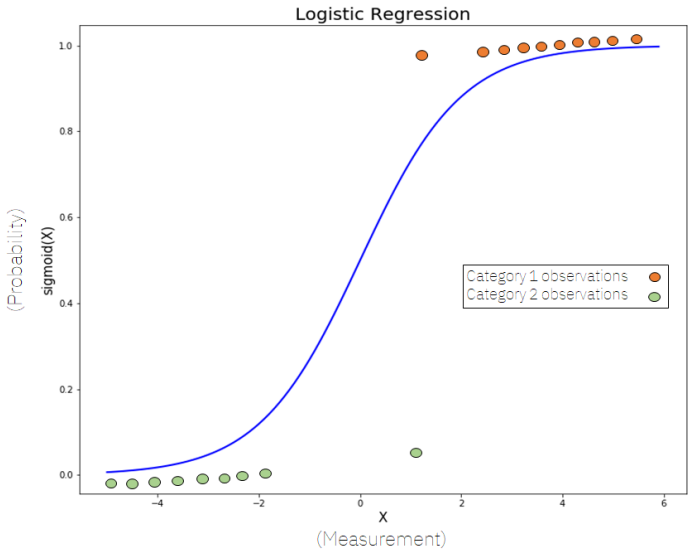


Figure 1.4: An example of sigmoid function

$$l(\beta) = \sum_{i=1}^{N} y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta)) = \sum_{i=1}^{N} y_i \beta^T x_i - \log(1+e^{\beta^T x_i})$$

To maximize it, it's sufficient to set the derivative to zero:

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i (y_i - p(x_i;\beta)) = 0$$

That implies, since the first components of x_i is 1:

$$\sum_{i=1}^{N} y_i = \sum_{i=1}^{N} p(x_i;\beta))$$

In this case, an analytic solution do not exists, so it's necessary to use a numerical method. One possible method is the Newton-Raphson one, which is quite straight-forward.

After initializing $\hat{\beta}_0$=0, we recursively update them in this way:

$$\hat{\beta}_t = \hat{\beta}_{t-1} - \left(\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T}\right)^{-1} \frac{\partial l(\beta)}{\partial \beta}$$

Writing the score (the first derivative with respect to β) and the Hessian in a matrix form, we get:

$$\frac{\partial l(\beta)}{\partial \beta} = \mathbf{X}^T(y-p)$$

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = -\mathbf{X}^T\mathbf{W}\mathbf{X}$$

where $\mathbf{p}$ is the vector of fitted probabilities $p(x_i;\beta_{t-1})$, $\mathbf{W}$ is an $N \times N$ diagonal matrix whose i-th element on the diagonal is $p(x_i;\beta_{t-1})(1-p(x_i;\beta_{t-1}))$, and y is the vector of y_i. In this framework the Newton-Raphson step becomes:

$$\beta_t = \beta_{t-1} + \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T(y-p)$$

$$= \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\left(\mathbf{X}\beta_{t-1} + \mathbf{W}^{-1}(y-p)\right)$$

$$= \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}z$$

where $z := \mathbf{X}\beta_{t-1} + \mathbf{W}^{-1}(y-p)$. Finally, at each step β_t is updated as follows:

$$\beta_t \longleftarrow arg\min_{\beta}(z-\mathbf{X}\beta)^T\mathbf{W}(z-\mathbf{X}\beta)$$

1.3.2 Naive Bayes

[4] Naive Bayes is a probabilistic classifier based on Bayes theorem, and on the hypothesis of strong independence of features.

In particular is a conditional probability model, where given a sample $x = (x_1, x_2, \ldots, x_p)$ with p features, it assign for each class k, $P\left(x \in \text{Class}_k | x_1, x_2, \ldots, x_p\right)$, i.e. the probability that the sample belongs to class k, given x.

To do so, it uses Bayes theorem:

$$P\left(x \in \text{Class}_k | x_1, x_2, \ldots, x_p\right) = \frac{P\left(x \in \text{Class}_k\right) P\left(x_1, x_2, \ldots, x_p | x \in \text{Class}_k\right)}{P\left(x_1, x_2, \ldots, x_p\right)}$$

Furthermore the "important" part is the numerator, since the denominator does not depend on the *Class* and remains constant for each of them.

Now, using the strong independence hypothesis the model becomes:

$$P\left(x \in \text{Class}_k | x_1, x_2, \ldots, x_n\right) \propto P\left(x \in \text{Class}_k\right) \prod_{i=1}^{k} P\left(x_i | x \in \text{Class}_k\right)$$

[4] section written with reference to [VS, 2022]

Obviously the sample is assigned to the class that maximizes $P(x \in \text{Class}_k | x_1, x_2, \ldots, x_n)$:

$$\text{Class}_k = \underset{k}{\text{argmax}}\, P(x \in \text{Class}_k | x_1, x_2, \ldots, x_n)$$

.

It's easy to see also that:

$$P(x \in \text{Class}_k | x_1, x_2, \ldots, x_n) = \frac{P(x \in \text{Class}_k) \prod_{i=1}^{k} P(x_i | x \in \text{Class}_k)}{Z}$$

with

$$Z = \sum_k \left(P(x \in \text{Class}_k) \prod_{i=1}^{k} P(x_i | x \in \text{Class}_k) \right)$$

as normalization constant.

Among several choices, usually Gaussian Kernel are used for continuous feature. It means that the unknown distribution of attributes is supposed to be a Gaussian one (we remark also we consider them independent). For each one of the p features, the estimates $\hat{\mu_p}$ and σ_p for μ_p and σ_p are computed with a MLE approach.

Regarding categorical features, the situation is very similar to the continuous case. Often, for the sake of simplicity, they are supposed to follow a Multinomial distribution. Apart from this, the procedure is analogous, with the parameters estimated through MLE.

1.3.3 Support vector machine

[5] SVM is a supervised learning algorithms, whose purpose is to find a suitable hyper-plane expressed as $wx + b$, that divides "correctly" the multi-dimensional vectors of the dataset in the two classes, represented by $y \in \{-1; 1\}$.

Ideally, the plane divides perfectly the two classes. That constraint for a generic sample can be express by :

$$y_i(w \cdot x_i + b) \geq 0$$

.

With some manipulations, equivalently (in then sense that the solution describes the same hyper-plane)

$$y_i(w \cdot x_i + b) \geq 1, \text{ with } y \in \{-1; 1\}$$

[5] section written with reference to [Shalev-Shwartz and Ben-David,]

Usually, it's not possible to divide perfectly samples, so it's necessary to allow some violations. The constrain becomes:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

trying to minimize the errors ξ.

More precisely, SVM does not only try to find a plane that divides data, but it also tries to maximize the "margin", i.e the distance of the nearest points from the hyper-plane , while minimizing the error, i.e the penalty associated to data on the wrong side of the margin. Moreover, the corresponding penalty increases with the distance of the wrongly classified points from the plane.

Formally, we can express SVM as:

$$\min_{w,\epsilon} \frac{1}{2}\|w\|^2 + C\sum \xi_i$$

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0, \forall i \in I$$

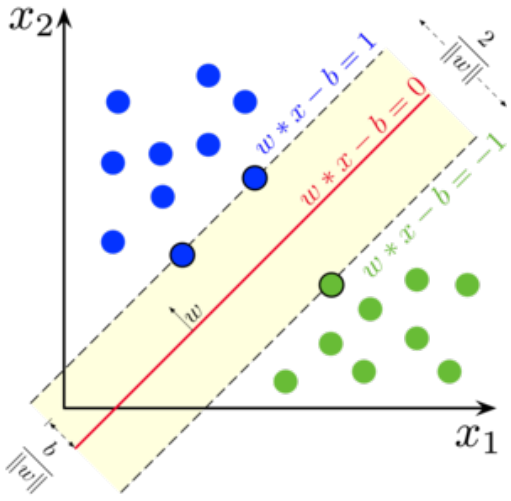


Figure 1.5: 2-D example of SVM

Non-Linear Kernel

In some cases, hyper-plane cannot divide properly the two classes. A possible way to overcame that, is to represent data in a higher dimensional space, and then find a suitable hyper-plane in that space. Here's an intuitive image that shows the usefulness of that process.

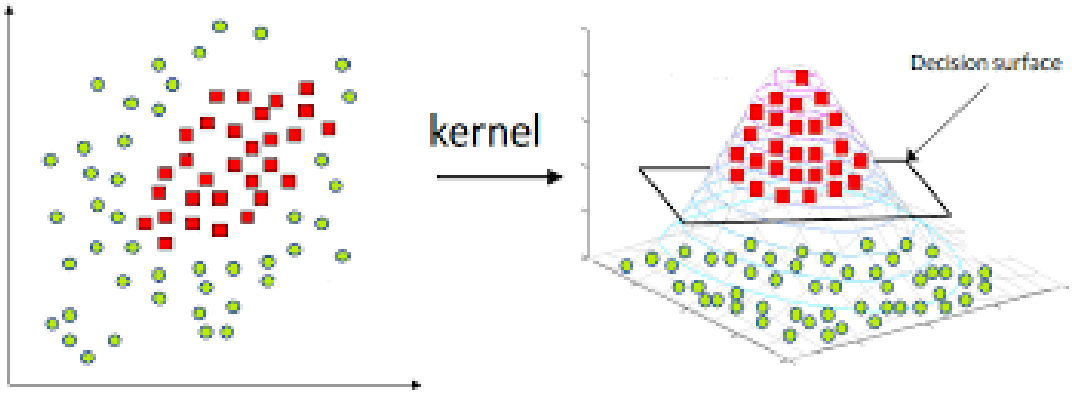


Figure 1.6: Kernel trick. credit: medium.com

Observing the dual formulation of the Soft Margin SVM (that is the case that allows for some violations):

$$\min_{\vec{w},\vec{b}} \max_{a\geq 0} \frac{1}{2}\|\vec{w}\|^2 - \sum_j \alpha_j \left[(\vec{w}\vec{x_j} + b)\, y_j - 1\right]$$

Or equivalently:

$$\max_{\alpha\geq 0, \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \, (\vec{x_i}\vec{x_j})$$

it's clear that the objective function depends on the dataset only by the dot product of his vectors.

That allows for a great simplification when representing samples in higher dimensional spaces, that can become computational unfeasible when working with lots of features.

The point is that it's not necessary to calculate explicitly the samples in the new dimensions, since only the dot product between them in the selected space is needed (the so called Kernel Trick). In other words, it's not required to explicitly write the mapping

$$\Phi : U \longrightarrow V$$

with $Dim(U) < Dim(V)$, but it's only needed the kernel function

$$K(\overrightarrow{x_i}, \overrightarrow{x_j}) = \langle \Phi(\overrightarrow{x_i}), \Phi(\overrightarrow{x_j}) \rangle_V$$

with V an Hilbert space.

Some examples are:

- **Polynomial Kernel**

$$K(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\gamma \overrightarrow{x_i} \overrightarrow{x_j} + c)^d$$

- **Radial kernel**

$$K(x, y) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - y_{ij})^2)$$

  with d, c, γ as hyper-parameter to be optimized.

Platt scaling

Contrary to logistic regression, the output of the model is not $\Pr(Y = 1|X)$, but it's just a label (a point lies on one side of the hyper-plane, or on the other). For lots of applications it's useful to associate a probability, or more generally a score, as a result of the classification, instead of a label. It can be done in the following way, known as Platt scaling:

$$P(y = 1|x) = \frac{1}{1 + \exp(\mathrm{A} f(x) + B)}$$

with $f(x)$ being the output of the classification, A and B hyperparameter to be optimized.

In this case the obvious choice for $f(x)$ is some sort of distance (that must also be suitable for categorical data).

The interpretation is quite straightforward: the farther the point is from the separating hyperplane, the more sure the classification is.

Notice that, it's meaningless to associate a threshold to this value, since a point can be on one side side of the hyperplane, or on the other.

1.3.4 K-nearest neighbors

[6] KNN method is a very straight-forward method quite easy to understand.

It basically assign an element to a class if that class represents the majority part of the K nearest neighbors. Formally:

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \Omega} I(y_i = j)$$

where Ω is the set of the K-nearest point to x_0 in the dataset.

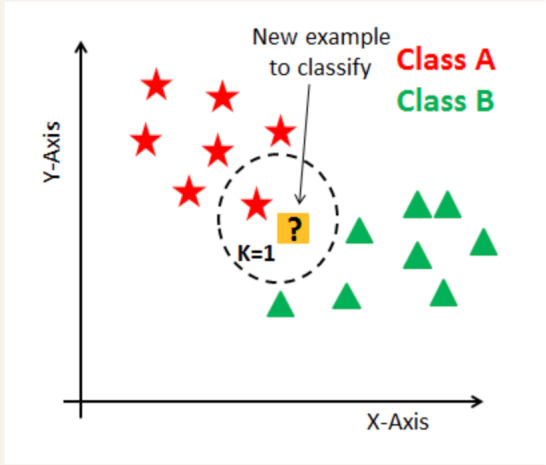


Figure 1.7: KNN example. Credit: www.datacamp.com

We recall what we said in the first section about statistical learning:

$$f(X) = E(Y|X = x)$$

With $f(X)$ indicating the hypothesis we want to infer (in that case it was a regression problem, in this case is a classification one, but conceptually in this situation they're interchangeable).

If we think about that, we can notice that what K-NN is trying to do, is to approximate $f(X) = E(Y|X = x)$ as

$$\hat{f}(x) = Ave_{i \in \Omega}(y_i|x_i)$$

It's important to precise that the definition of "neighborhood" depends upon the chosen formulation of distance.

Among the different possibilities, a way to deal with categorical data, is to transform them into numerical ones, creating for each categorical attribute and for each value of the level of the category considered a dummy variable.

[6] section written with reference to [Hastie et al., 2009]

If we consider a feature with N levels, and a sample whose level for that category is i, then after that procedure we would obtain N new binary variable, whose value is one for the i-th dummy variable and zero for all the other.

$$\begin{bmatrix} Grade \\ 7 \\ 7 \\ 8 \\ 9 \\ 7 \\ 9 \\ 8 \\ 7 \end{bmatrix} = \begin{bmatrix} Grade_7 & Grade_8 & Grade_9 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Figure 1.8: Creation of a dummy variable

In the example above is possible to observe how this process acts on a categorical variable *grade* (for the sake of simplicity let's assume that its levels are $7, 8, 9$).

1.3.5 Classification tree

Tree based method are ML algorithms broadly used for classification and regression.

Briefly, they stratifies the predictor space, creating several simple subregions, defined by certain "range" of the attributes.

In our case, this is what a portion of the tree looks like:

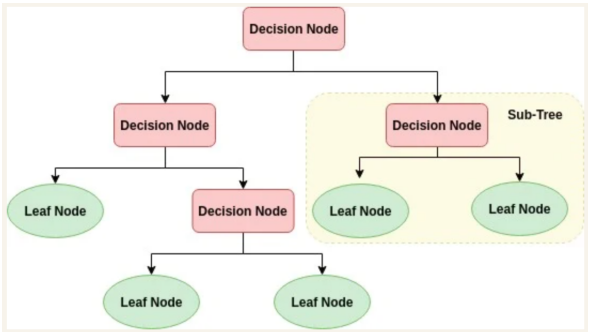


Figure 1.9: Decision tree example. Credit:www.datacamp.com

The fundamental element of a tree is the node, i.e. the single bisection; a decision rule for a node has a formulation like $X_i \leq t$, or in case of categorical may be a splitting of the various level.

For each leaf, we have a certain number of samples of the two classes,and we "assign" that terminal node to the most represented one.

Ideally, the best situation is the one where a particular decision rule relative to a certain feature, splits the samples in such a way that in the following node only one class is represented, because that means we can probably successfully use that attribute to predict classes.

It comes naturally to use some measure of "how well" the attributes separates classes.

The most common choices are:

- GINI INDEX :$\sum_{i=1}^{k} 1 - p_k^2$
- ENTROPY : $\sum_{i=1}^{k} p_k \log_2 (p_k)$

p_k is the proportion of elements of the class k in the leaf of interest:

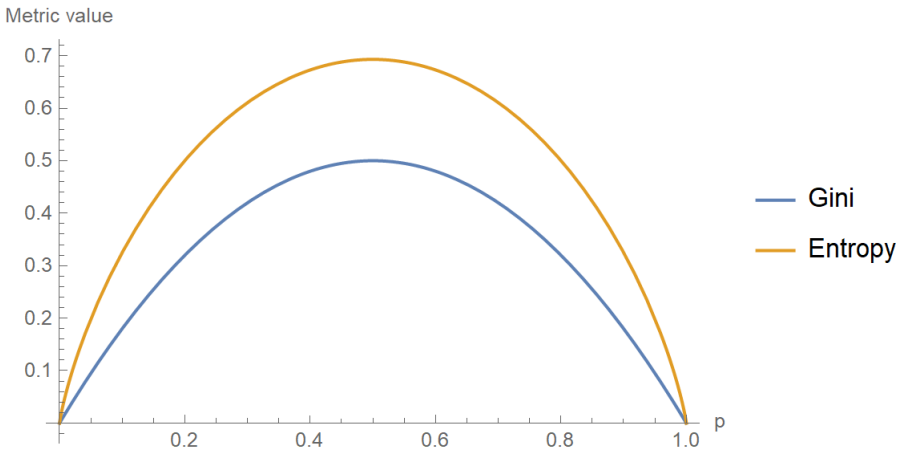


Figure 1.10: GINI index and Entropy

As we can see, nodes whose composition is quite homogeneous (i.e. for values are close to 0.5) are "penalized".

The algorithm keeps dividing elements choosing attributes (and particular value of the chosen attribute) in such a way to minimize "greedily" one of the possible index until some stopping criteria are met.

Some example of stopping criteria are:

- In order to attempt a split, there must be at least a certain number of observations
- A minimum number of samples is required in each terminal node.
- The number of split must be less than a certain value.

With decision trees, is not so easy to associate a "score" as a result of a classification instead of a label. What can be done is associating to each terminal leaf a score equal to the proportion of classes present, for example if a samples belongs to a leaf that in the training phase was composed by 31 element of class "0" and 5 of class "1" would have a corresponding probability of belonging to class "0" of $\frac{31}{31+5} = 0.86$. while this can be useful for calculating ROC and AUC, it makes little sense if the scope is to choose a suitable threshold, since other method like misclassification cost, under-sampling or over-sampling are much more intuitive

1.3.6 Bootstrap aggregation and boosted trees

Classification trees have a great interpretability, but the classification performance leaves some room for improvement. In fact, decision tree usually have lots of variance relatively to their performance. In order to limit this aspect bagging

method were introduced. The idea behind bootstrap aggregation (i.e.bagging) methods is to combine the classification results of different "weak" classifiers into a "strong" one, lowering in this way the variance, since what we are really doing is averaging different observations. A weak classifier, in binary classification case, is a classifier whose accuracy is only slightly better than 50%. Usually, it's not possible to have access to different training sets, so a solution is to generate several bootstrapped training sets starting from the original one. This process is done by sampling with replacement several new datasets (bootstrapped) from the original dataset.

Concretely, B bootstrapped training set are created, then B over-fitting trees are build upon it, and then the results are averaged.

The tree is an over-fitting one, so that the relative bias is low; the variance will be reduced with then averaging step.

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b^*(x)$$

the most natural choice is to classify a sample following the **majority vote**.

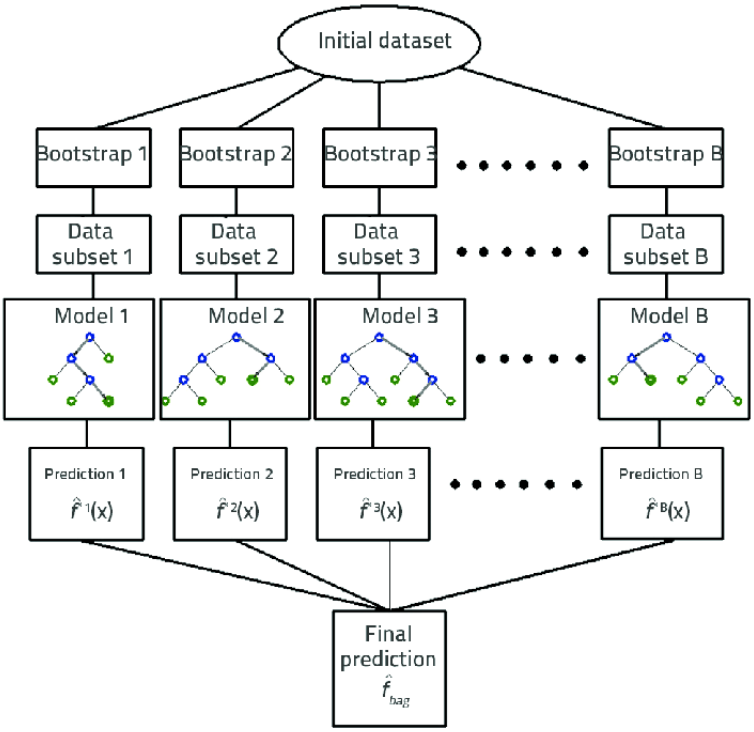


Figure 1.11: Bagging process. Credit: medium.com

Boosting method works in a similar way. The main difference is that each

tree is grown sequentially using informations of previous trees.

Random forest adds a step to bagging, by introducing an improvement that decorrelates trees. The first part is equivalent to the preceding one, the classification trees are build upon bootstrapped subsets, but in the case of random forest the features are chosen randomly. More precisely m predictors are picked among the full set of p elements, typically $m \approx \sqrt[2]{p}$.

AdaBoost

[Freund and Schapire, 1999] AdaBoost in a Boosted trees algorithm formulated by Yoav Freund and Robert Schapire in 1995, and it's based on the use of weak-learners, i.e. methods who performs only slightly better that random guessing, into a single model.

In this case the "weak learner unit" is a *stump*, a tree with only one split. They're built progressively, in fact each stump is build using information obtained in the previous round; moreover samples and stump do not have all the same weight like in the bagging case, but "good classifiers vote" weights more, and misclassification cost of misclassified samples is greater, so that hopefully in the next round they will be classified correctly.

Let's see the step that AdaBoost is composed by:

1. We take as input a Test Set with m samples :$(x_1, y_1), ..., (x_m, y_m)$, a number of rounds T, and a Weak Learner WL
2. We initialize each samples weight $W_1 = (\frac{1}{m}, ..., \frac{1}{m})$, and the learner weights.
3. We create a decision stump for each variable, and observe the classification results obtained. More importance is assigned to "good classifiers", following this criteria

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - TotalError_m}{TotalError_m} \right)$$

Where α_m is the importance of the stump, and $TotalError$ is just the sum of misclassifications.

Moreover, weight is assigned to misclassified samples, in that way :

$$w_i = e^{\pm \alpha} \times w_{i-1}$$

Where the sign of α depends by the label of the sample (correct or incorrect). Then weight are normalized up to 1

4. Now a new dataset is generated, by "tweaking" the original one in such a way that the samples with the highest way "appears" more time than the other.

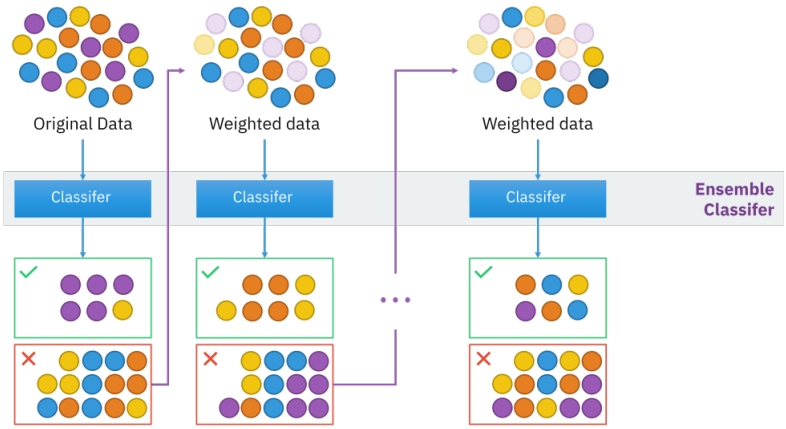


Figure 1.12: Ensamble classifiers. Credit: towardsdatascience.com

Finally the steps are repeated until a stopping criteria is met.

Concluding, in order to classify a sample, we observe the output generated by each learner, and we assign the sample itself to the majority class, following a weighted vote criteria.

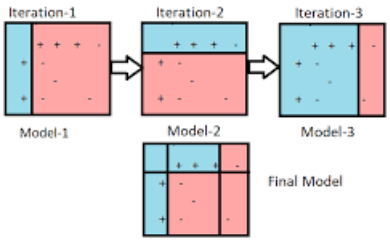


Figure 1.13: Ensamble classifier. Credit: towardsdatascience.com

1.3.7 Hyper-parameters tuning

[7] In the previous section, we've shown some problems linked to the various performance measurements, nevertheless choosing a way to assess the model performance is crucial in order to proceed and optimize hyper-parameters.

This process is far from being easy. In fact if we observe the problem from a more abstract point of view, we can interpret the model as a "function" , whose inputs are the samples, and the output is a value that measure the classification performance. The point is that the function we want to optimize, is not differentiable, or convex, and moreover a single evaluation may be very time-expensive. So special approaches are needed.

Furthermore, the performance metric chosen in both datasets is just the **Misclassification Error**.

Grid search

Grid search is a model-free black-box optimization method. It basically consists in an exhaustive search on set that consists of a Cartesian product of different sets. Each set is formed by a discretization of the set of a given hyper-parameter if it is continuous, or the whole itself, if it is discrete.

The most notable problem linked to this approach is the **curse of dimensionality**, since the number of evaluations needed grows exponentially with the number of parameters to be optimized.

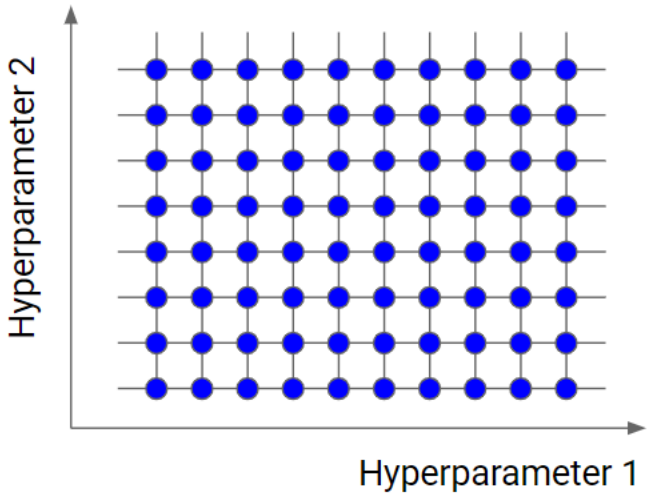


Figure 1.14: Grid search. Credit : researchgate.net

[7] Written with reference to [Feurer and Hutter, 2019]

Random search

The second approach, which is still a model-free black-box optimization method, is the random search, which replaces an exhaustive enumeration with a random selection of a certain umber of N-tuple parameters. Despite his simplicity,it offers some advantage over the grid search.

- given a grid, adding a point will destroy the grid structure (except for some trivial cases).
- The search is parallelizable very easily.
- It's possible to include some prior knowledge, since we can choose the density we sample from.

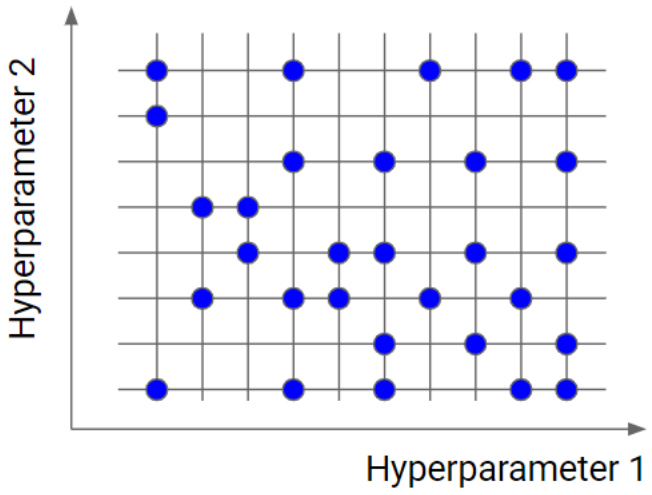


Figure 1.15: Random search. Credit : researchgate.net

Bayesian optimization

Bayesian optimization is an approach very useful for black-box optimization, when evaluations are expensive, and we can't calculate derivatives.

Bayesian optimization includes prior information about the function to be optimized, in our case a performance measure, and updates posterior information, which helps reduce loss and maximize the model's accuracy.

Moreover Bayesian optimization it's not an exhaustive search, in fact it tries to optimized a very complicated function by approximating it with a simpler one, a **surrogated function**.

A common choice is to use a **Gaussian process**.

To define the next point to query we define an acquisition function, that models how good a point we believe could be.

Ideally, we want to evaluate a function in a point which:

- we expect it to be a good optima candidate
- we have a lot of uncertainty about

so the acquisition function should be high in the points we expect to be good and in the points where uncertainty is high, and low in the already explored points.

A suitable function is the **Expected Improvement**

$$\mathbb{EI}(\lambda) = \mathbb{E}\left[max(f_{min} - y, 0)\right]$$

is common choice since it can be computed in closed form if the model prediction y at configuration λ follows a normal distribution, i.e a we are using a Gaussian process:

$$\mathbb{EI}(\lambda) = (f_{min} - y)\phi\left(\frac{f_{min} - \mu(\lambda)}{\sigma}\right) + \sigma\phi\left(\frac{f_{min} - \mu(\lambda)}{\sigma}\right)$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and standard normal distribution function, and f_{min} is the best observed value so far.

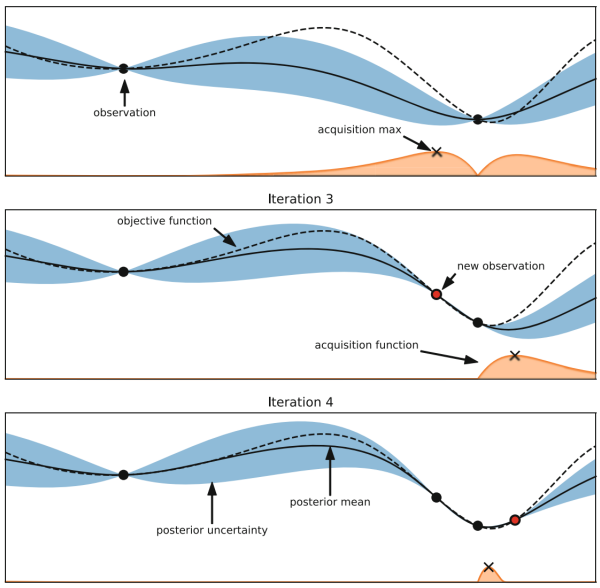


Figure 1.16: Bayesian optimization process.Credit:"Automated ML",Hutter,Kotthoff,Vanschoren.

The blue shaded area represents the uncertainty linked to the Gaussian process, the dotted line is the real mean of the process, and the black one is the estimated one.

As we can see the query point is the one that maximizes the acquisition function, which intuitively is high in the promising zones (i.e near the best point so far) and in the unexplored zones.

A generic algorithm should look like this.

Algorithm 1 Basic pseudo-code for Bayesian optimization

Place a Gaussian process prior on f
Observe f at n_0 points according to an initial space-filling experimental design. Set $n = n_0$.
while $n \leq N$ **do**
  Update the posterior probability distribution on f using all available data
  Let x_n be a maximizer of the acquisition function over x, where the acquisition function is computed using the current posterior distribution.
  Observe $y_n = f(x_n)$.
  Increment n
end while
Return a solution: either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean.

In this section some corners were cut along the way, since they're beyond the scope of this thesis.

Chapter 2

Algorithm application

2.1 German Credit Dataset

2.1.1 Dataset Description

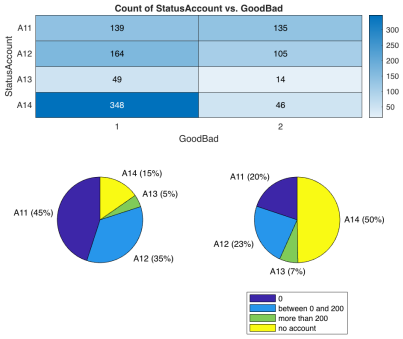
The first dataset introduced is a collection of 1000 samples and 20 attributes. It aims to describe the loans emitted from the bank, and the economic situation of the borrower.

It is made available by UCI Machine Learning Repository [1], and it's generated by Hans Hofmann, professor at Institut fur Statistik und "Okonometrie Universit" (Hamburg).

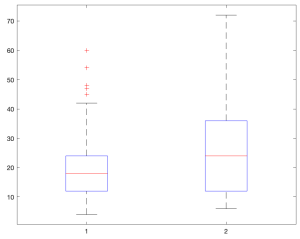
The response variable is binary, and is **GoodBad**, 1 for good, 2 for bad. Before applying any machine learning algorithm it might be useful a first, more general inspection of the dataset. Ideally, for the purposes of classification, for each attribute should be possible to see some "in-homogeneity" in the distribution of values, depending on whether the sample belongs to class 1 or 2. By way of example, a possible difference in the average age of risky and non-risky creditors would make Age a highly significant attribute for classification. Therefore a graphic survey can highlight these aspects. To do so, the dataset is split into two different ones, based on the response variable. In particular the dataset is composed as follows:

- **Status of existing checking account**:
  - A11 : ... < 0 DM
  - A12 : 0 <= ... < 200 DM
  - A13 : ... >= 200 DM salary assignments for at least 1 year
  - A14 : no checking account

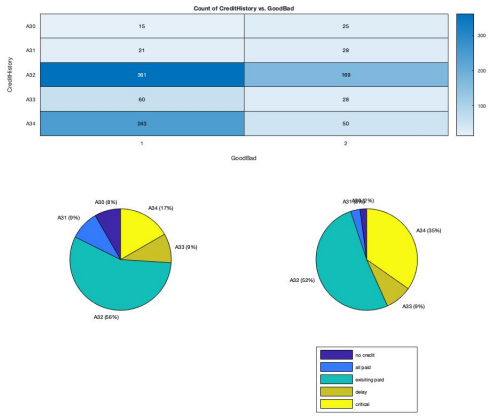
[1] https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/



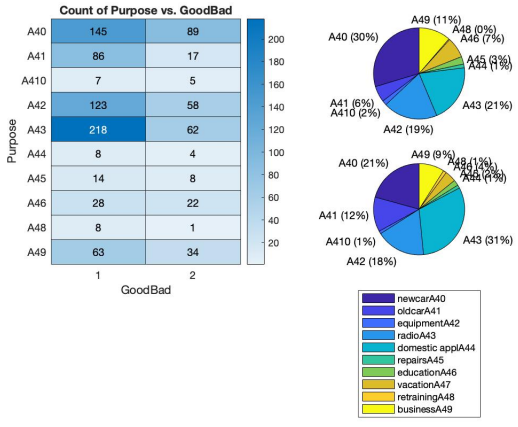
- **Duration in month: duration in month**



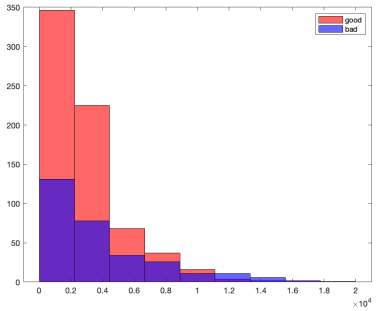
- **Credit history:**
  - A30 : no credits taken/ all credits paid back duly
  - A31 : all credits at this bank paid back duly
  - A32 : existing credits paid back duly till now
  - A33 : delay in paying off in the past
  - A34 : critical account/other credits existing (not at this bank)



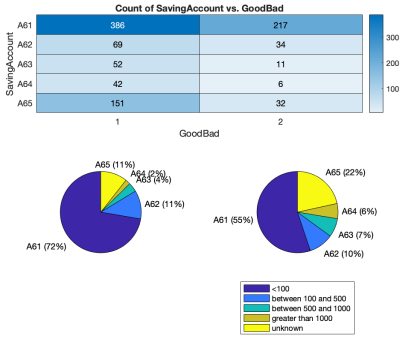
- **Purpose**: purpose of the loan
  - A40 : car (new)
  - A41 : car (used)
  - A42 : furniture/equipment
  - A43 : radio/television
  - A44 : domestic appliances
  - A45 : repairs
  - A46 : education
  - A47 : (vacation - does not exist?)
  - A48 : retraining
  - A49 : business
  - A410 : others



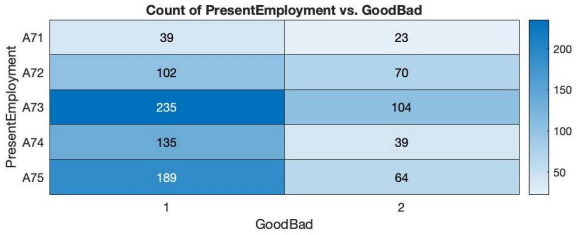
- **Credit amount**



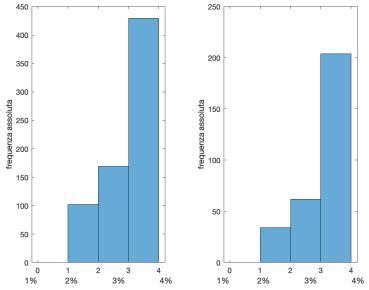
- **Savings account**
  - A61 : ... < 100 DM
  - A62 :100 <= ... < 500 DM
  - A63 :500 <= ... < 1000 DM
  - A64 : .. >= 1000 DM
  - A65 : unknown/ no savings account



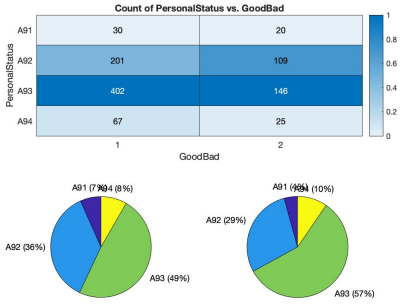
- **Present employment since**
  - A71 : unemployed
  - A72 : ... < 1 year
  - A73 : 1 <= ... < 4 years
  - A74 : 4 <= ... < 7 years
  - A75 : .. >= 7 years



- **Installment Rate**: Installment rate in percentage of disposable income



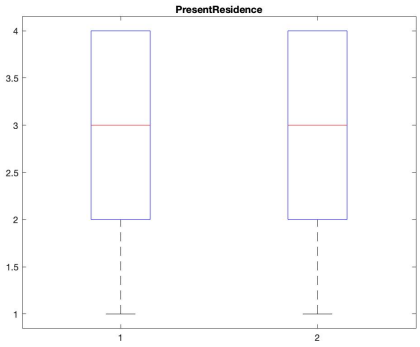
- **Personal status and sex**:
  - A91 : male : divorced
  - A92 : female : divorced/separated/married
  - A93 : male : single
  - A94 : male : married/widowed
  - A95 : female : single



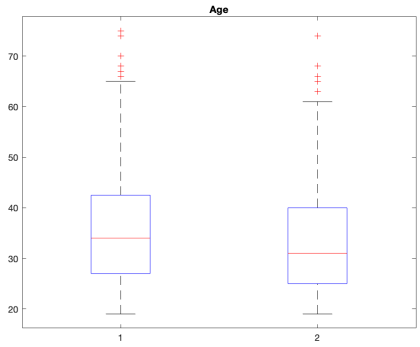
- **Other debtors / guarantors**:

  - A101 : none
  - A102 : co-applicant
  - A103 : guarantor

- **Present residence since**:

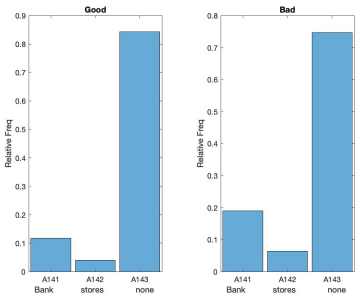


- **Property**: Asset with higher value owned
  - A121 : real estate
  - A122 : if not A121 : building society savings agreement/ life insurance
  - A123 : if not A121/A122 : car or other, not in attribute 6
  - A124 : unknown / no property
- **Age**

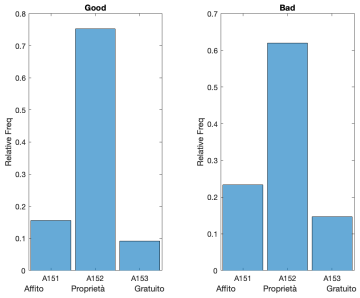


- **Other installment plans** : other credits in progress
  - A141 : bank

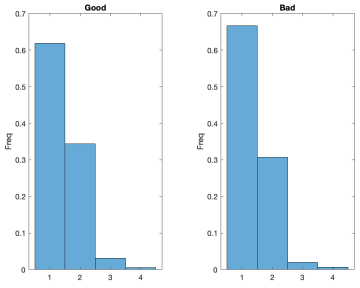
- A142 : stores
- A143 : none



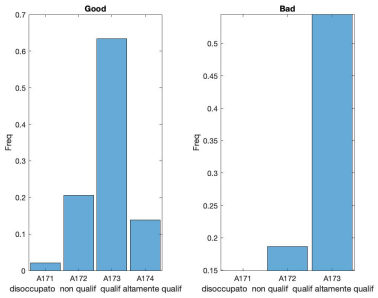
- **Housing**:
  - A151 : rent
  - A152 : own
  - A153 : for free



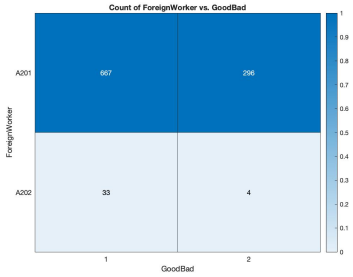
- **Number of existing credits**



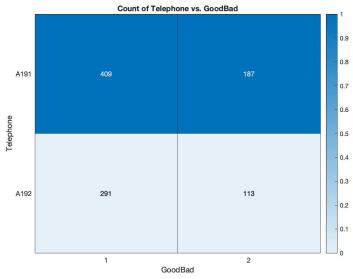
- **Job**:
  - A171 : unemployed/ unskilled - non-resident
  - A172 : unskilled - resident
  - A173 : skilled employee / official
  - A174 : management/ self-employed/ highly qualified employee/ officer qualificated



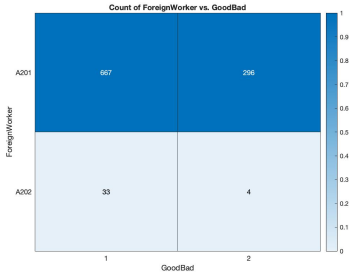
- **Number of people being liable** : Number of people entitled to maintenance
  - 1: from 0 to 2
  - 2: 3 or more



- **Telephone**: binary attribute



- **Foreign Worker**: binary attribute



2.1.2 Algorithm application

In this paragraph we will apply the algorithms introduced in the previous sections. A part of the dataset (30%) is reserved to the final evaluations, in the last chapter Moreover performance is estimated using a 5-fold validation. Then, when possible, hyper-parameters are optimized using some of the methods outlined before, namely Grid-search or Bayesian optimization, while trying to minimize **Minimum Classification Error**

Logistic regression

The first algorithm applied is the logistic regression. Here we can the β in decreasing order, with the corresponding $p - value$:

		Estimate	pValue
1	StatusAccount_A14	1.7119	0
2	Purpose_A41	1.6665	0
3	InstallmentRate	-0.3301	0.0002
4	Purpose_A43	0.8916	0.0003
5	SavingAccount_A65	0.9467	0.0003
6	CreditHistory_A34	1.4358	0.0011
7	Purpose_A42	0.7916	0.0024
8	Duration	-0.0279	0.0027
9	CreditAmount	-0.0001	0.0039

Table 2.1: Logistic regression results

The most significant indicators of a risky creditor seems to be the purchase of a used car and the absence of a checking account. The results of the classifications are:

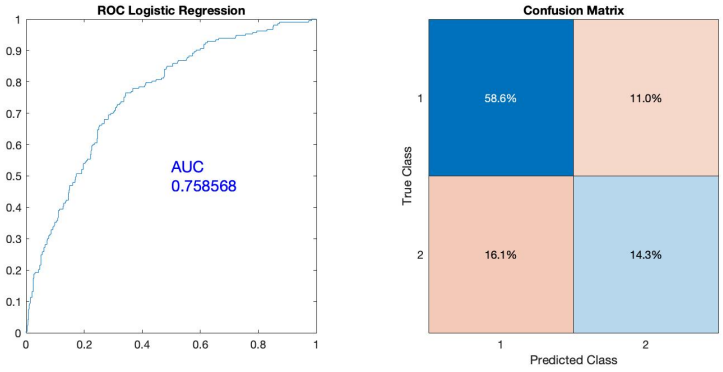


Figure 2.1: Results of Logistic regression

Plotting each samples against the corresponding probabilities, we can get a glimpse of classification performance:

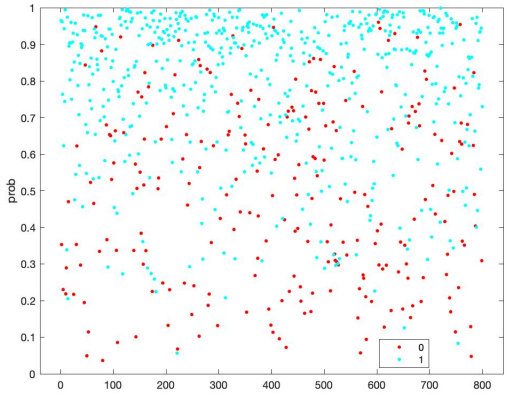


Figure 2.2: Depiction of each sample against the corresponding predicted probability to belong to class 1

Naive bayes

Naive Bayes gives the following results:

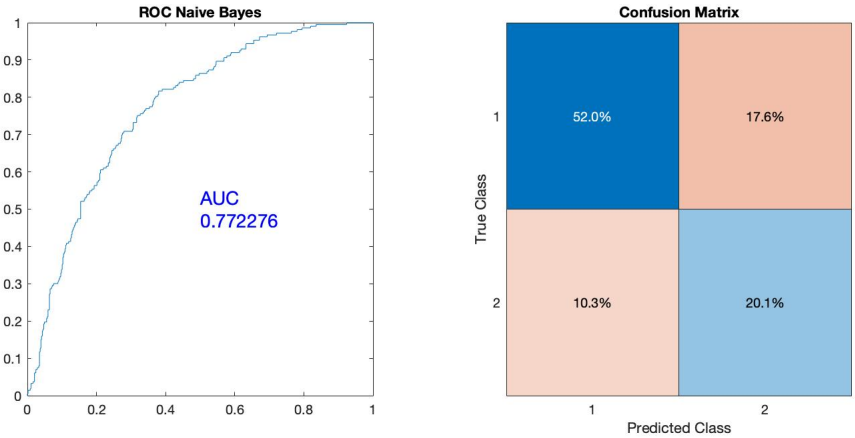


Figure 2.3: Results of Naive Bayes

It has to be remarked that Naive bayes achieves good results in just $1,01s$.

SVM

Firstly a linear kernel SVM is applied. It gives the following results:

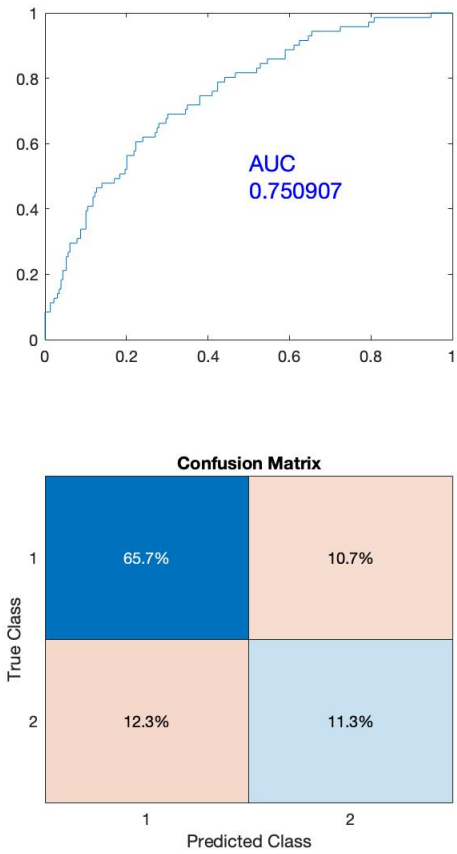


Figure 2.4: Results of Logistic regression

Here's an example on how SVM splits datas, in a CreditAmount against Duration plan:

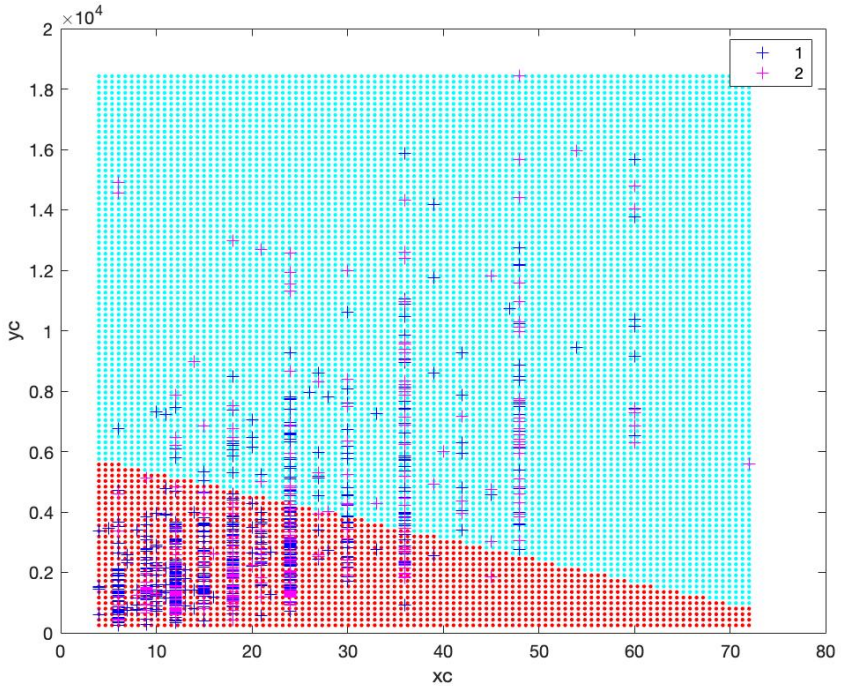


Figure 2.5: Linear kernel SVM

The next step consists in optimizing the parameters with a Bayesian optimization. Heuristically, the optimal parameters appears to be :

- Kernel : medium Gaussian
- KerneScale: 4.5 (How much data should be scaled).
- Standardize: true (If it's better to standardize datas or not).
- BoxCostraint: 1 (How much classifications should be penalized).

Here we can see a snippet of the Bayesian optimization process:

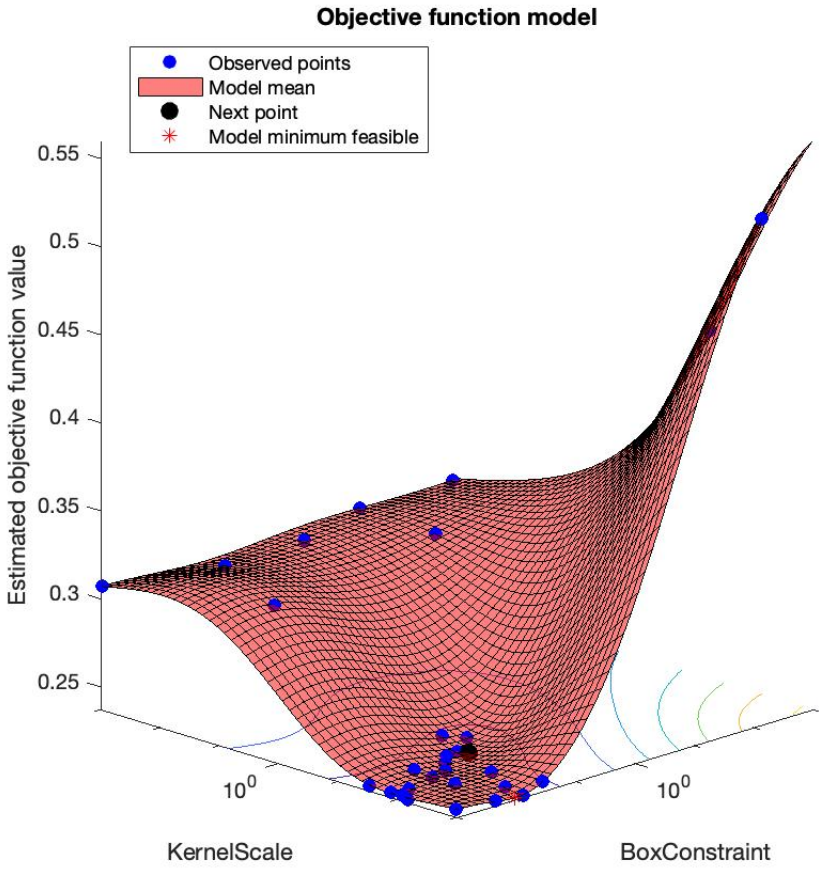


Figure 2.6: Surrogated function of Bayesian optimization.

Optimizing parameters, increases the quality of results. For example using non-linear kernel as a Gaussian one, namely:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

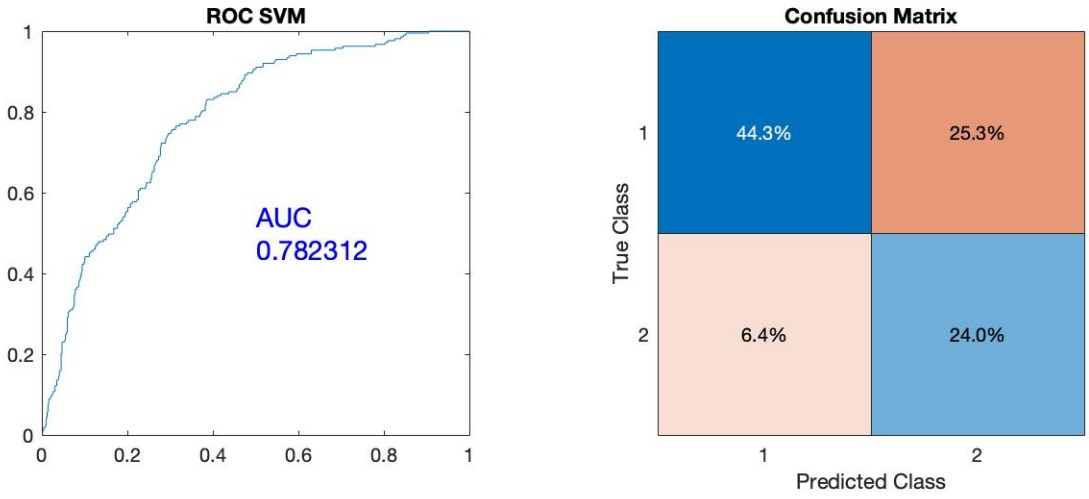


Figure 2.7: Results of SVM

Here we can observe the non linearity introduced by the Gaussian kernel:

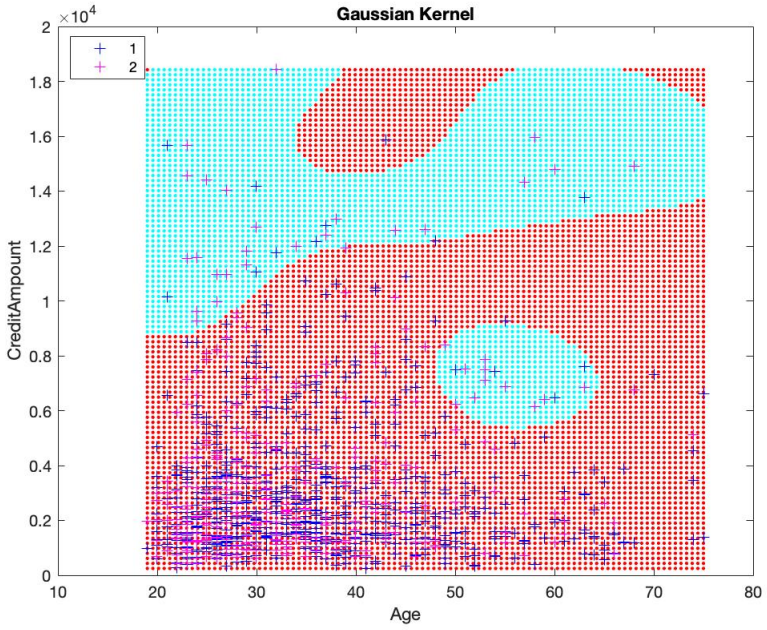


Figure 2.8: Gaussian Kernel SVM

Knn

In knn there are mainly two parameters to be optimized: the number of neighbors and the distance used. After standardizing datas, using a grid search, we can optimize both of them. In the plot we can see the estimated classification error for each iteration:

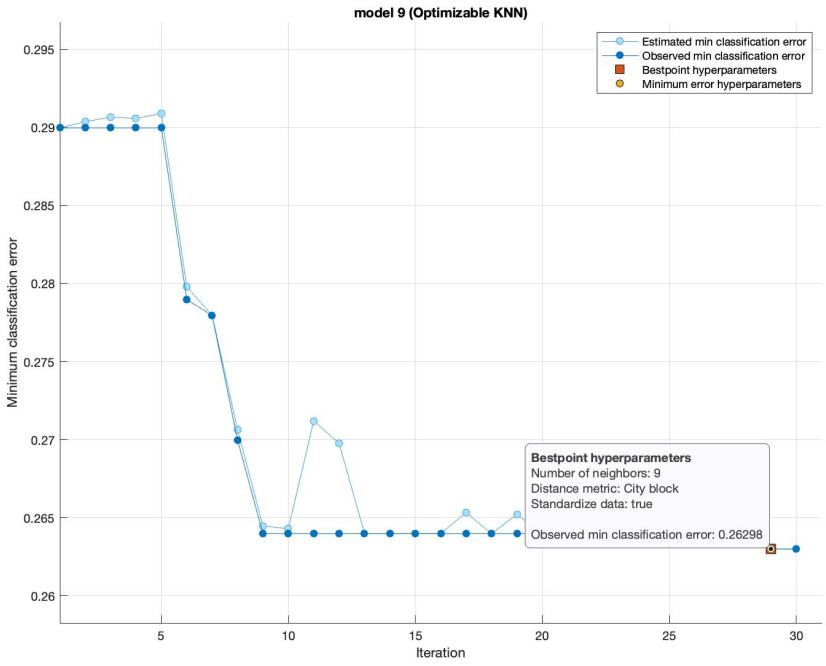


Figure 2.9: Minimization of classification error process

The best results are obtained using $k = 9$, and a *City block* distance, defined as follows:

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

Below, we can see an image that intuitively explains the concept.

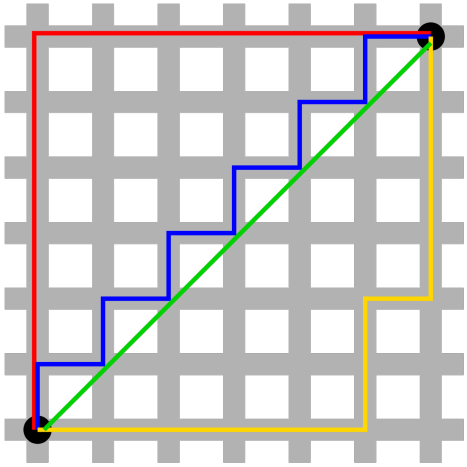


Figure 2.10: The green one is a representation of the euclidean distance, the others of Manhattan distance

The results obtained are the following:

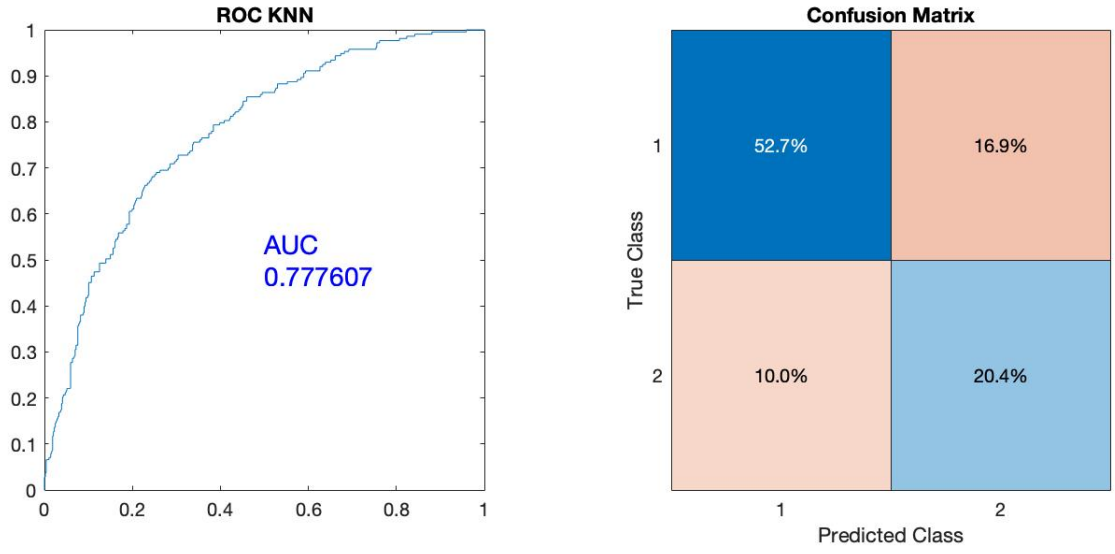


Figure 2.11: Results of KNN

Optimized and bagged tree

The great advantage of trees over the boosted or bagged counterpart, is that they are very easy to interpret as we can see from a snippet of the model built:

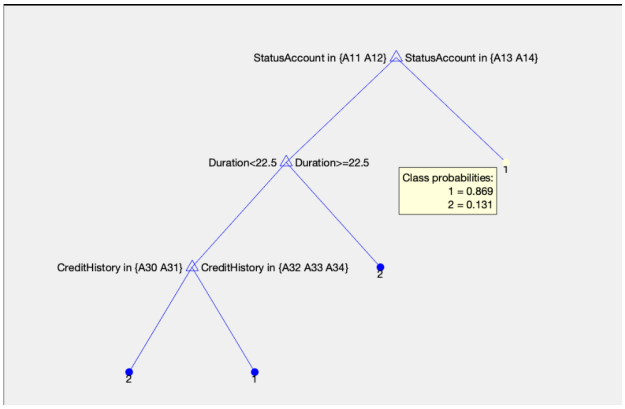


Figure 2.12: Decision tree of German dataset

Using Bagging, i.e building trees on bagged subset taken from the dataset or Boosting, interpretability is lost, but the performance improves usually by a lot. The best results are obtained with AdaBoost:

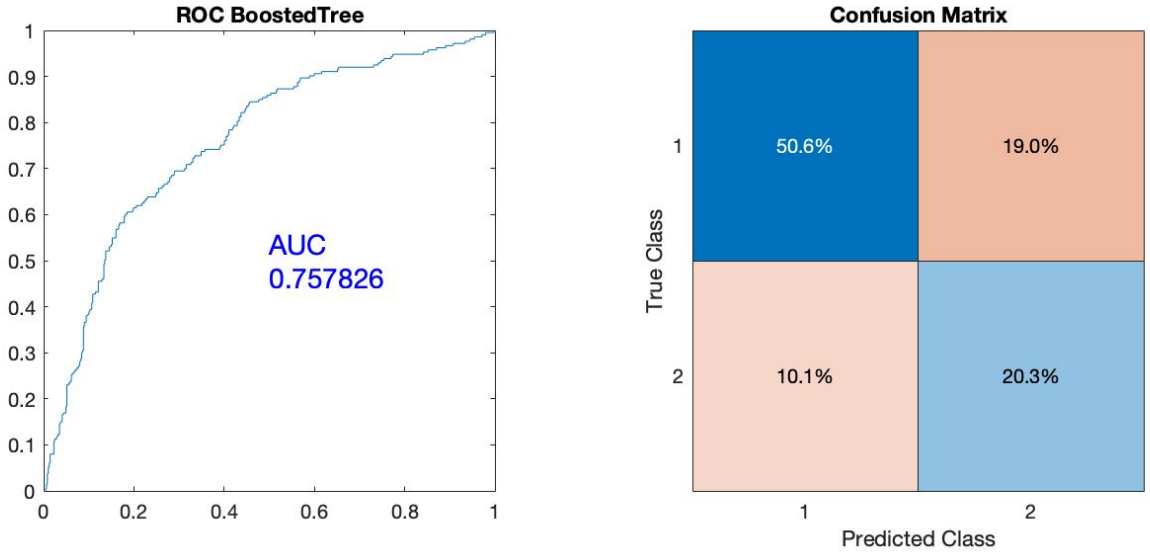


Figure 2.13: Results of Boosted trees

Summary of results

The AUC of the methods used are:

	Logistic Reg	Naive Bayes	KNN	SVM	Boosted Trees
AUC	0.7585	0.7726	0.7776	0.7823	0.7578

Table 2.2: AUC results

Here we can see all the ROC curves:

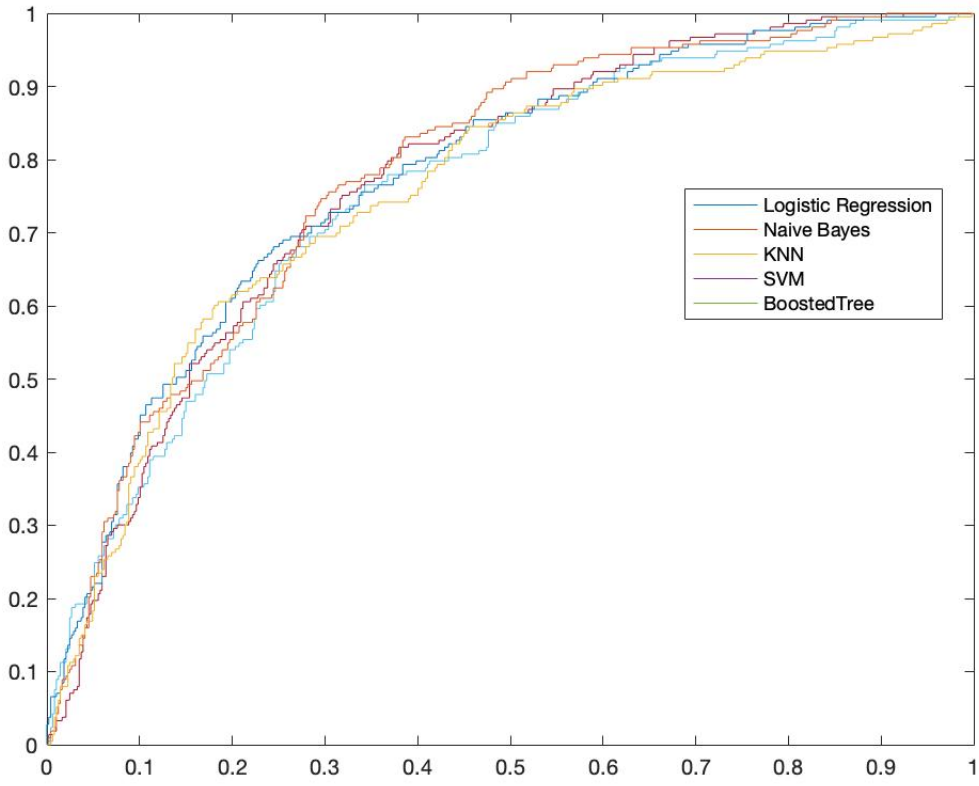


Figure 2.14: ROC curve confrontation

We remark that a classifier whose ROC curve lies above another classifiers ROC curve, has a better performance. Other than AUC there exists other performance measures:

	Accuracy	Sensitivity	Specificity	Precision	FPR	F1score
Logistic Reg	0.7300	0.6064	0.7754	0.8429	0.4978	0.7053
Naive Bayes	0.7114	0.5467	0.8273	0.7219	0.6900	0.6222
KNN	0.6929	0.5230	0.8232	0.6921	0.6943	0.5958
SVM	0.6457	0.4779	0.9114	0.5244	0.8952	0.5001
Ensemble Trees	0.6971	0.5275	0.8312	0.6900	0.7118	0.5979

Table 2.3: German dataset classification performance

Basing our choice on AUC, which we remark to be a good performance measure, the best method seems to be the **SVM**. But as it can be seen, other measure would lead to different options. In particular, also **Logistic regression**'s performance looks quite valid.

2.2 Loan P2P

The dataset in question is a collection of P2P loans made through the Lending Club platform, from 2007 to 2020. LendingClub is an American peer-to-peer lending company, the first P2P lender to register its offerings as securities with the Securities and Exchange Commission, and to offer loan trading on a secondary market. The dataset is made available by Kaggle [2] , an online community of data scientists and machine learning practitioners. There are 887380 samples and 74 attributes, some of which must be discarded as textual attributes, so it is difficult to obtain valid and reliable information.

Other than that, the dataset needs some pre-processing, so in the following section it will undergo a step-by-step data-cleaning procedure

2.2.1 Dataset pre-processing

Text Attributes

The first step, is to remove text attributes, since it's difficult and outside the scope of this thesis to extract information from these, so:

- **Desc** Loan description provided by the borrower.
- **URL** Which is a web address.
- **Emp_title** Job described by the borrower when applying for the loan.

Leaking future datas

The first step, consists in removing those attributes whose value was not available at the moment of lend concession. In fact, since we are interested in classifying reliable and unreliable borrower, we cannot use data that we will obtain only in the future. So we will remove:

- **Funded_amount** The total amount committed to that loan at that point in time.
- **Funded _ amount_inv** The total amount committed by investors for that loan at that point in time.
- **Last_pymnt_d** Last month payment was received
- **Last_pymnt_amnt** Last total payment amount received
- **Total_pymnt** Payments received to date for total amount funded
- **Total_pymnt_inv** Payments received to date for portion of total amount funded by investors
- **Total_rec_prncp** Principal received to date

[2] https://www.kaggle.com/datasets/ethon0426/lending-club-20072020q1

- **Total_rec_late_fee** Late fees received to date
- **Out_prncp** Remaining outstanding principal for total amount funded
- **Out_prncp_inv** Remaining outstanding principal for portion of total amount funded by investors

Since them contains informations obtained past the loan emission.

- **Issued_d** The month which the loan was funded

Since it means that the platform granted a loan.

- **Recoveries** post charge off gross recovery
- **Collection_recovery_fee** post charge off collection fee

Since the presence of these means that loan defaulted. Moreover since we're interested in testing our ML algorithm, it seems reasonable to drop off all FICO scores related attribute. FICO score is a value that banks uses to measure the creditworthiness of the issuer.

- **fico_range_high** The upper boundary range the borrower's FICO at the moment the loan had been granted
- **fico_range_low** The lower boundary range the borrower's FICO at the moment the loan had been granted.
- **last_fico_range_high** The FICO upper boundary range at the present moment.
- **last_fico_range_low** The FICO lower boundary range at the present moment.

Non-informative attributes

The next step, is to drop off those attribute that do not influence the classification:

- **id** — randomly generate identifier.
- **member_id** — randomly generated identifier
- **addr_state** since contains the same information of **zip_code**
- **Grade**, in fact the same informations are contained **Subgrade**

2.2.2 Missing data, unique values

Here we can have a look at the attributes that presents missing datas:

Attribute	Missing data	Attribute	Missing data
"total_acc"	29	"open_acc_6m"	866007
"next_pymnt_d"	252971	"open_il_6m"	866007
"last_credit_pull_d"	53	"open_il_12m"	866007
"collections_12_mths"	145	"open_il_24m"	866007
"acc_now_delinq"	29	"open_rv_12m"	866007
"emp_length"	44825	"mths_since_last_delinq"	454312
"annual_inc"	4	"mths_since_last_record"	750326
"delinq_2yrs"	29	"open_acc"	29
"earliest_cr_line"	29	"pub_rec"	29
"inq_last_6mths"	29	"revol_util"	502
"open_rv_24m"	866007	"inq_last_12m"	866007

Table 2.4: Missing values of Loan dataset

we can remove these attributes as they miss too much values:

- **open_rv_24m**
- **inq_last_12m**
- **open_acc_6m**
- **open_il_6m**
- **open_il_12m**
- **open_il_24m**
- **open_rv_12m**
- **mths_since_last_delinq**
- **mths_since_last_record**

For other attributes, we need some care, because those ones could introduce some bias in our models. But since only the 5% of entries has one or more missing attributes, we can remove them safely without loosing too much information. Also, we can remove **Policy_code**, **pymnt_plan**, **application_type** as every samples has the same value, once the dataset is pre-processed.

Response variable

The response variable is **status**, that has 10 different levels with several samples belonging to each one:

- **Charged off**: considered as probable loss
- **Current**: in progress
- **Default**: default
- **Does not meet the credit policy-Charged off**: the loan is considered as probable loss, the loan application would no longer meet the credit policy and would not have been approved in the market.
- **Does not meet the credit policy-Fully paid**: the loan was repaid, but the loan application would no longer meet the credit policy and would not be approved in the market.
- **Fully paid**: paid
- **In grace period**: in grace period of 15 days
- **Issued**: loan granted
- **Late (16-30 days)**: delayed for 16-30 days
- **Late (31-120 days)**: delayed for 31-120 days

Some categories need to be deleted or modified:

- **charged off**: has no elements present
- **Issued**: these are newly granted loans for which no information is available

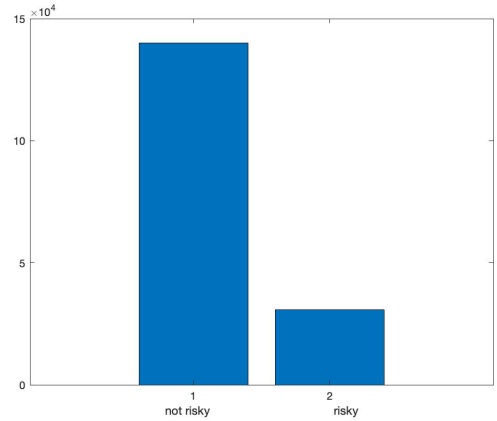
Finally, it is possible to remove attributes for loans still in progress:

- **In grace period**
- **Late (16-30 days)**
- **Late (31-120 days)**
- **Does not meet the credit policy-Charged off**

The 3 levels remaining are:

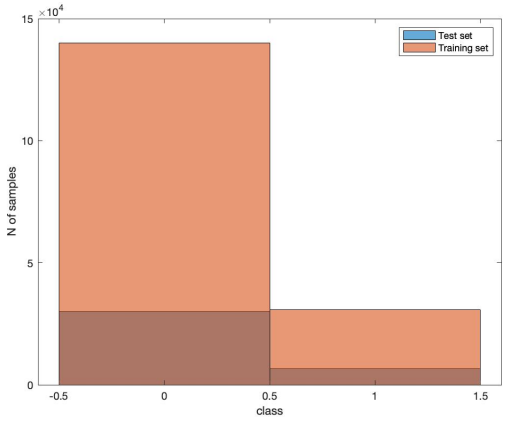
Level:	N of samples:
Default	1219
Fully paid	207723
Charged off	45248

We can organize those levels in 2 different ones: **risky** (defaulted and charged off) and **not risky** (Fully paid).



Hold out-set

Since the dimension of the dataset are large, it's possible to partition it into a **training set**, **test set**, and **hold-out set**. To do so each samples is assigned with probability 0.7 to the first one, and probability 0.15 to each of the other one. In this way the dataset obtained are stratifies, i.e. they maintain the same proportion of risky and non risky elements.



Principal component analysis

In order to further reduce dimensionality, we can apply PCA to the remaining numerical attributes of the dataset. The dataset is centered, so that each feature has the same weight. Plotting the cumulative variance explained (which can be interpreted as a measure of information retained) , in relation to the number of components, we can see that the 99% of variance is retained with the first 4 principal component:

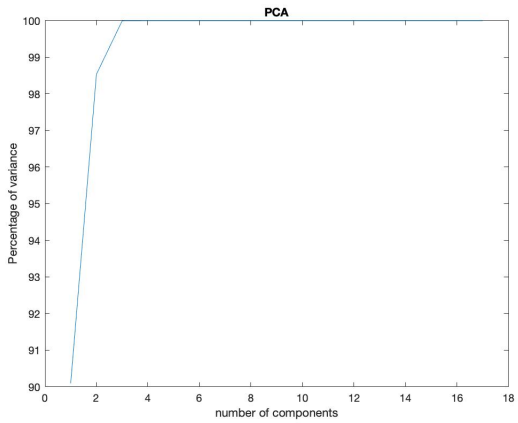
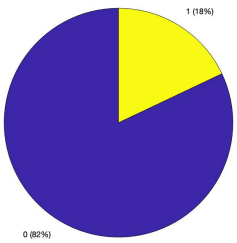


Figure 2.15: Variance explained and PCA

Unbalanced datas

As we already seen we have a quite unbalanced situation:



Since the samples after the pre-processing are **243974**, which is a great number, in order to deal with the outbalancing we can apply an undersampling, by constructing a new training set in this way

1. Selecting all the elements of class **risky**

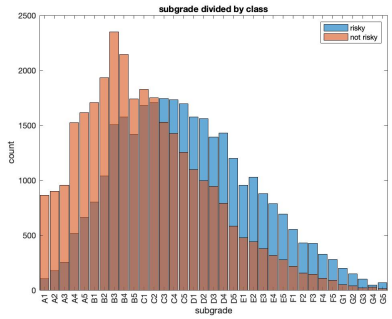
2. Selecting an equal number of elements of **not risky** class, randomly

By doing so we obtain a perfectly balanced training set.

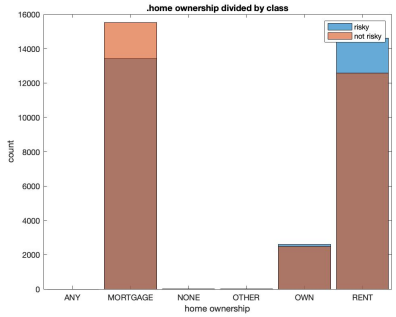
2.2.3 Algorithm application

Now the dataset presents **61209** and 6 **attributes**, plus the **class label**:

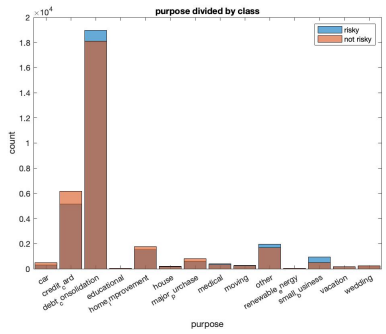
1. **Subgrade** LC assigned loan subgrade



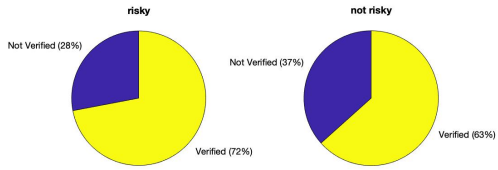
2. **Home_ownership** The home ownership status provided by the borrower during registration



3. **Purpose** A category provided by the borrower for the loan request.

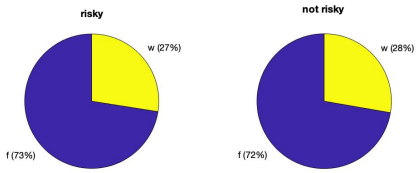


4. **Verification_status**

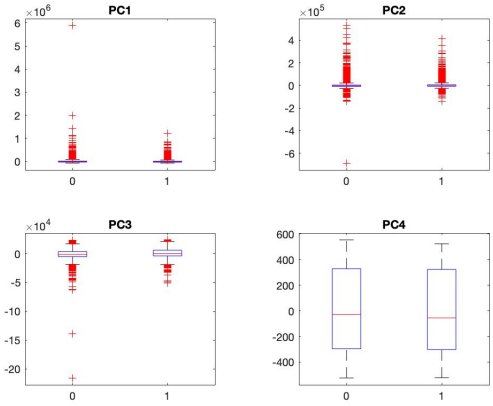


5. **addr_state** The state provided by the borrower in the loan application

6. **Initial_list_status** The initial listing status of the loan (binary).



7. and the first **4 principal component**



A first graphical inspection confirms that there is some dishomogeneity in the data distribution, especially in **Sub_grade, Home_ownership and Purpose**

Logistic regression

The results of logistic regression confirms what the visual inspection suggested:

		Estimate	SE	tStat	pValue
1	sub_grade_E4	2.7159	0.1989	13.6530	0
2	sub_grade_E5	2.7463	0.2035	13.4939	0
3	PC1	-4.9723e-06	3.6930e-07	-13.4642	0
4	sub_grade_E3	2.6260	0.1952	13.4520	0
5	sub_grade_E2	2.5679	0.1919	13.3816	0
6	sub_grade_E1	2.5571	0.1929	13.2592	0
7	sub_grade_D5	2.4778	0.1871	13.2429	0
8	sub_grade_D3	2.4018	0.1850	12.9842	0
9	sub_grade_D4	2.3894	0.1844	12.9565	0

Table 2.5: Logistic regression results

The table depicts the most influential attributes, along with the corresponding β. Also $p - value$, $standard - error$, and $t - statistic$

Subgrade is the most indicative attribute. Moreover is interesting to notice that PC1, even is very significative, the corresponding β is very close to 0. The classification performance is the following:

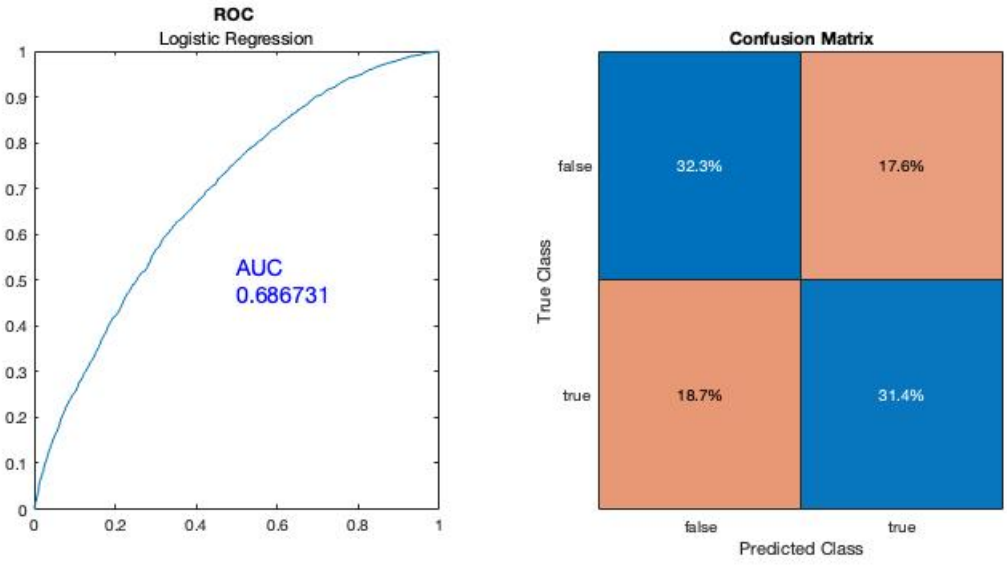


Figure 2.16: Logistic regression results

Plotting each samples against the corresponding probabilities, we can see intuitively the classification performance:

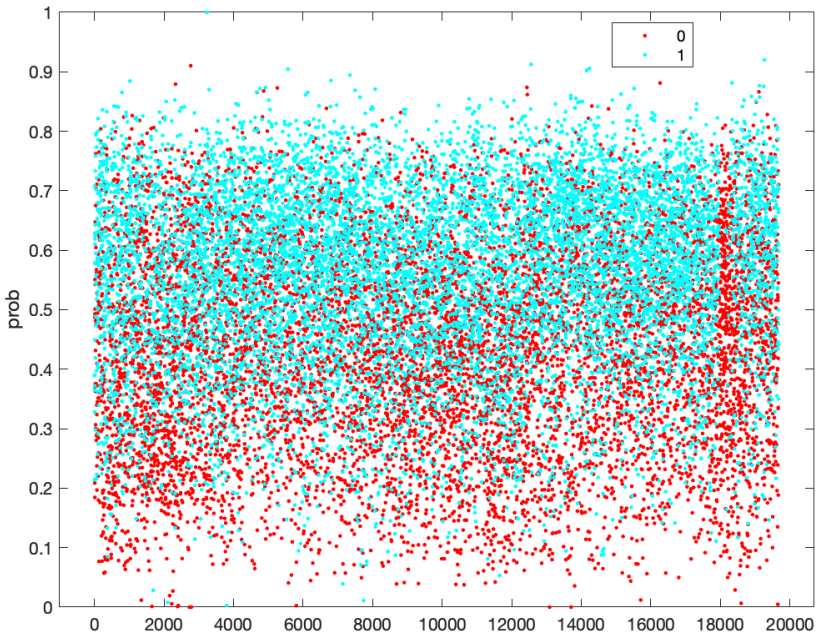


Figure 2.17: Depiction of each sample against the corresponding predicted probability

Regarding Logistic regression, recalling what we said in section 1.2.3 we may assume intuitively that we should assign a sample to class 0 if $p \leq 0.5$, and to class 1 in $p > 0.5$, but in case of highly unbalanced case, this may not be the optimal choice, in fact several one are possible.

Let's consider two trivial case: the first one, with that given threshold T equal to one, would led to classify each sample to class 0, so obviously it holds:

$$Precision_{T=1} = 1$$

$$Sensitivity_{T=1} = 0$$

And, in a symmetrical way, for $T = 0$:

$$Precision_{T=0} = 0$$

$$Sensitivity_{T=0} = 1$$

For values of T between 0 and 1 we will get several different performance measure's values. The increase in one, will eventually make the others decrease,

so it's not obvious which the optimal choice is (we recall that this one is exactly the meaning of the ROC curve, as explained in section 1.2.3).

In order to grasp better the concept, we can resort to a graphic interpretation. What we should do, is to choice a threshold (which is just an horizontal line) that splits well the blue points from the red one. Varying this threshold gives different confusion matrix.

Therefore the best threshold depends on different factors, anyway the ROC curve can overcame this aspects by summarizing effectively the effect that every choice of T would have.

SVM

the best SVM hyperparameters seems to be:

- KernelFunction': 'polynomial'
- 'PolynomialOrder': 3
- 'KernelScale': 1
- 'BoxConstraint': 0.001018900420527312
- 'Standardize': true

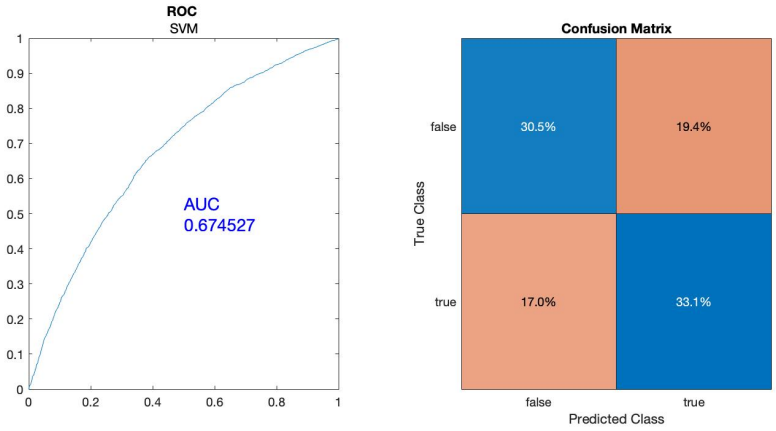


Figure 2.18: SVM results

Naive Bayes

Naive Bayes, is by far the quickest algorithm. Kernel Naive Bayes gives this performance:

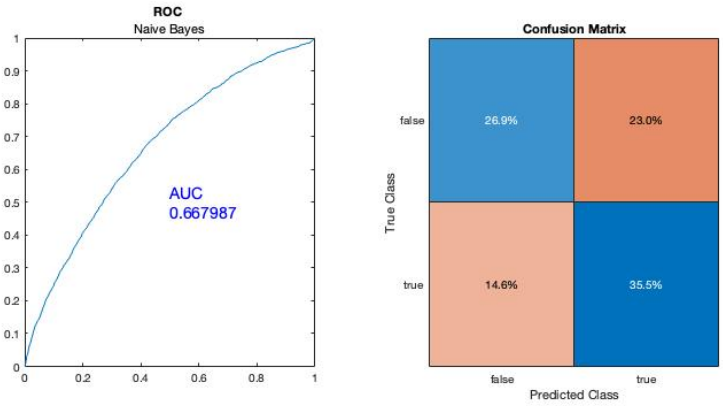
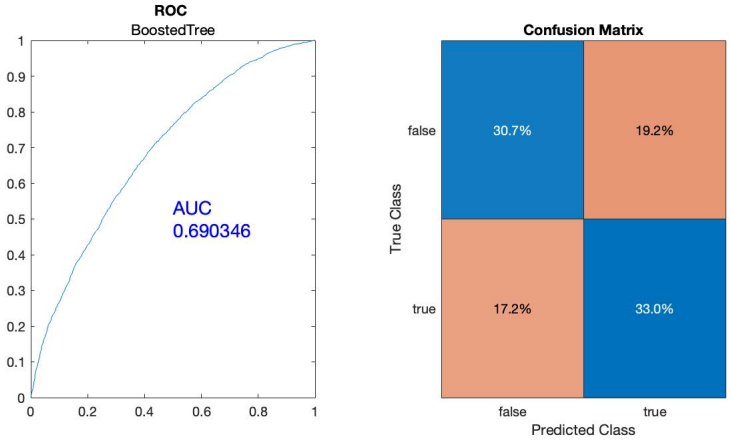


Figure 2.19: Naive Bayes results

Bagged trees

Optimization gives these results:

- 'Method': AdaBoost
- 'NumLearningCycles': 330
- 'LearnRate': 0.1355696826739945



Knn

Before applying K-NN, since there are several categorical attributes, it's necessary to transform them into numerical ones. Since Sub-grade has ordinal features, it's possible to map them into integers

'A1'		1
'A2'		2
'A3'		3
'A4'		4
'A5'		5
'B1'	$\longrightarrow$	6
'B2'		7
'B3'		8
'B4'		9
'B5'		10

Table 2.6: Subgrade: from categorical to numerical

Other attributes, do not present this characteristic, so it's necessary to transform them into dummy attributes. The hyper-parameters optimization gives as optimal:

- 'Distance' : Spearman
- 'NumNeighbors' : 321
- 'Standardize' : true

Given two sample $(X_1, X_2, .., X_n)$ and $(Y_1, Y_2, ..., Y_n)$ we can calculate the **Spearman distance** by converting each entries of the samples in the corresponding rank $R(X_i)$ and $R(X_2)$, then the distance is just:

$$r_s = \rho_{\mathrm{R}(X),\mathrm{R}(Y)} = \frac{\mathrm{cov}(\mathrm{R}(X), \mathrm{R}(Y))}{\sigma_{\mathrm{R}(X)}\sigma_{\mathrm{R}(Y)}},$$

The results obtained are:

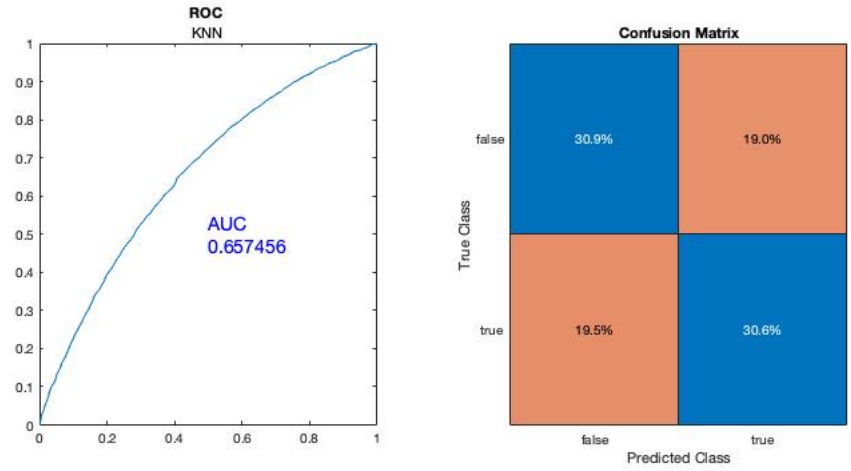


Figure 2.20: KNN results

Summary of results

The best performance seems to be achieved by logistic regression, but the small difference with Boosted trees suggest their performance is more or less equivalent, and may depend by the particular instances we are analyzing.

	Logistic Reg	Naive Bayes	KNN	SVM	Boosted Trees
AUC	0.6909	0.6679	0.6574	0.6745	0.6903

Table 2.7: AUC results

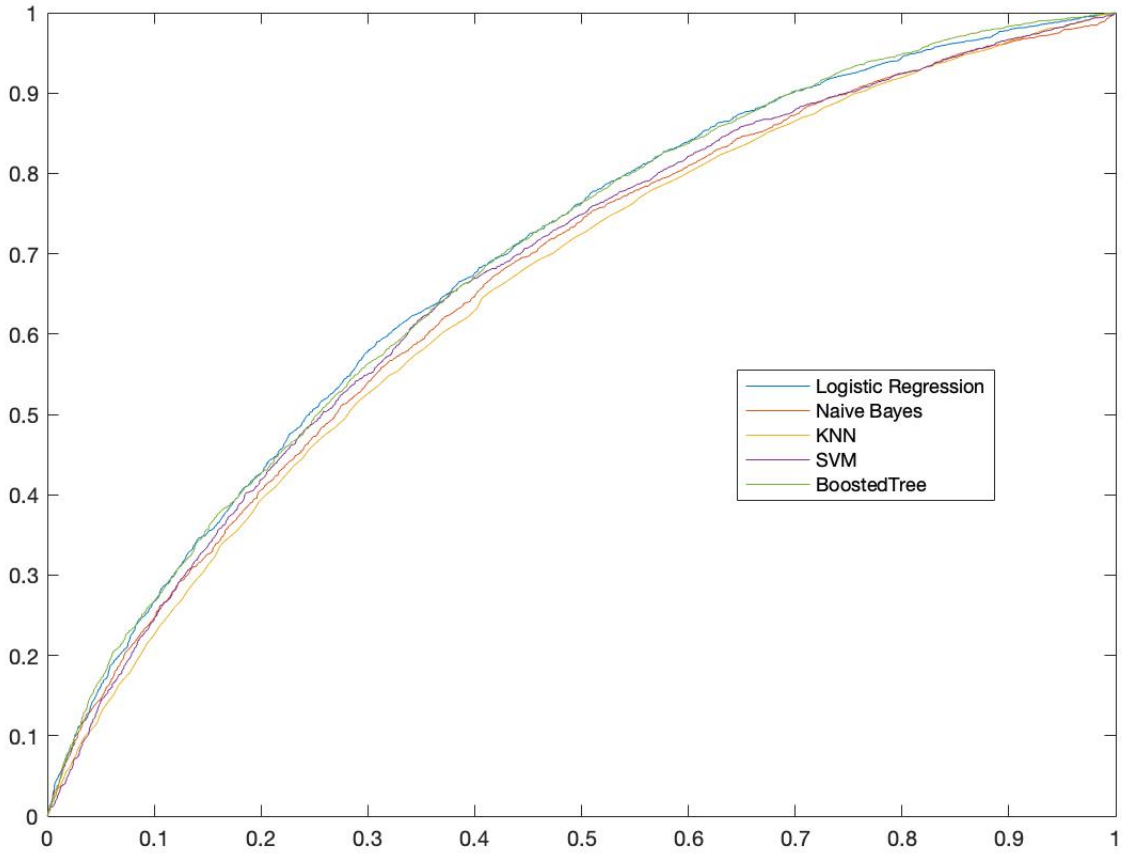


Figure 2.21: ROC curves confrontation

For the sake of completeness we can also report other performance measures:

	Accuracy	Sensitivity	Specificity	Precision	NPV	F1score
Logistic Reg	0.6327	0.6284	0.6374	0.6119	0.6534	0.6200
Naive Bayes	0.6156	0.6037	0.6308	0.5531	0.6777	0.5773
KNN	0.6154	0.6164	0.6144	0.6152	0.6156	0.6158
SVM	0.6304	0.6208	0.6419	0.5865	0.6742	0.6032
Ensemble Trees	0.6358	0.6284	0.6441	0.6033	0.6681	0.6156

Table 2.8: Loan dataset classification performance

As always, different measures lead to different definition of "the best method". But overall we can say that logistic regression gives good results, moreover it offers generally a greater interpretability than the others alternatives so it may be the preferred alternative.

Chapter 3

Ensemble classifiers

3.0.1 Model complexity

Now we will return to a topic already treated previously, the bias-variance trade-off, but from a slightly different point of view. We recall that in a statical learning framework, for a generic (x_0, y_0) it holds:

$$MSE(x_0) = E(y_0 - \hat{f}(x_0)^2) = Var(\hat{f}(x_0)) + E(\hat{f}(x_0) - f(x_0)) + Var(\epsilon)$$

so the error is composed by different parts: one that is irreducible, i.e $Var(\epsilon)$, and other two, the "bias" an the "variance" of the model:

$$Var(\hat{f}(x_0))$$

$$Bias[\hat{f}(x_0)] = E(\hat{f}(x_0)) - f(x_0)$$

We will try to grasp in an intuitive way, what these terms means, and more generally their trade-off.

Conceptually, the **error due to bias** is the difference between the expected prediction and the correct label. We talk about "expected prediction" because the model builded depends on the sample we have, as well as other factor. So, repeating the model building process ex novo, would lead to different results. After repeating this process several times, we can start talking about an average prediction.

On the other side, in this framework, the **error due to variance** is taken as the variability linked to the prediction relative to a given point. In simple words, the variance is how much the prediction changes depending on the realizations of the model.

Graphically we can represent those error as follows:

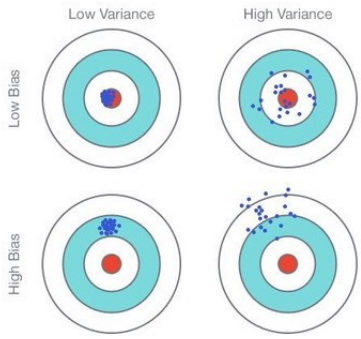


Figure 3.1: Bias-variance trade-off. Credit: medium.com

In this bulls-eye diagram, the center of the target represents the perfect model, and the variance is obviously symbolized as the dispersion of the points.

Over-fitting

In this scenario, we are intuitively led to think that we should reduce the bias as much as possible, no matter what, but it is not correct, in fact this would lead to an **over-fitting** of data.

We can define intuitively over-fitting as a model that too closely fits a given dataset, and it ends up not fitting other dataset as well. It essentially starts to fit not only the data structure, but also the noise of the samples.

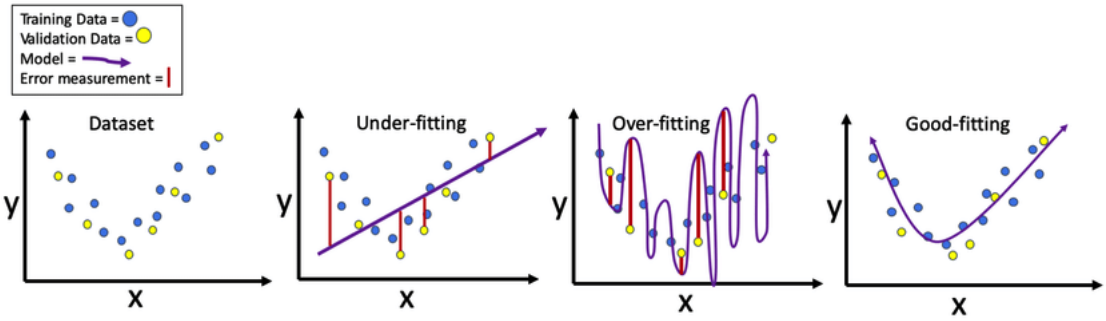


Figure 3.2: Overfitting. Credit :Deep learning approaches in food recognition

Furthermore, over-fitting can be seen under another point of view, plotting the test-error and the training error of the model against the model complexity, which can be interpreted intuitively as a property of the model that leads to an under-fit or an over-fit of the same.

As can be seen, increasing the model complexity leads almost surely to a decrease of the testing error. To convince us of this, we can take as an example a simple decision tree. It's easy to keep splitting leaves until we have a complete homogeneity of classes in each terminal node.

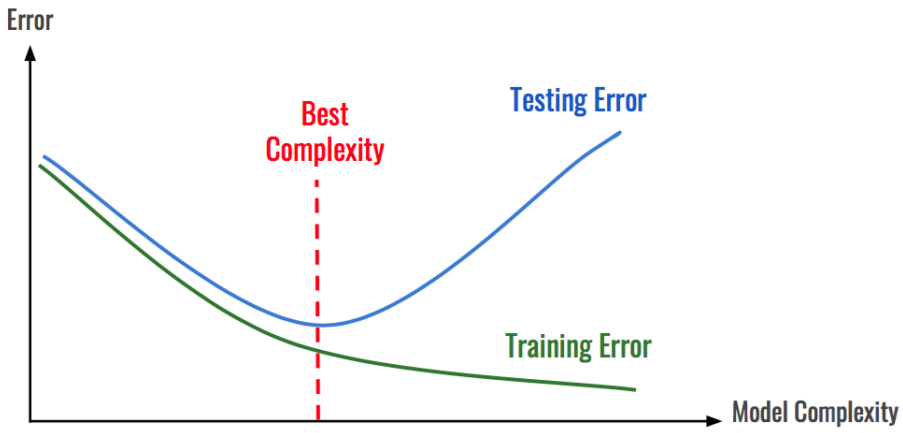


Figure 3.3: Bias-Variance and Errors. Credit: Medium.com

On the other side, an increase in complexity, past a certain point, would not increase testing performance which is the quantity that we are really interested in.

So, as we have already said, there exist an optimal complexity level, that allows us to achieve the best possible performance.

3.0.2 Lowering variance

As shown in the previous sections, the algorithm used possess some constraint that allows to lower variance. The logistic regression models linearly the log-odds ratio, and the SVM (even with the use of kernels) use a separating hyper-plane that maximizes margin, so the over-fitting is avoided by the structure itself of the model. K-nn with k greater than 1, "averages" the neighbors, and the decision trees use some pruning technique to lower complexity.

A special mention is needed for **bagging** and **boosting**, already introduced in section 3.6. Recapping briefly, bagging, i.e. bootstrap aggregating, builds several datasets by sampling with replacement from the original one, then, for each bootstrapped dataset a model is created, and finally the different predictions are aggregated. Boosting is similar, but it uses "weak learners", in other words, models that are only slightly better than random guessing. This allows to decorrelate trees and so lower further the variance.

Summing it all up, bias and variance have to be balanced, as an increase in one generally leads to an increase in the other. Each model must have a way

to control variance, and a very common way is to aggregate prediction made by different models.

Till now we combined prediction that are in a certain way "homogeneous", since they are generated always by the same algorithm. Intuitively, continuing this reasoning, we can also aggregate predictions generated by different algorithm.

This approach is a very fertile trend in the field of machine learning, and it's known as **ensemble classifiers**

3.0.3 Origins of ensemble learning

Ensemble learning is a general term for machine learning methods that unify multiple predictions to make a decision, typically in a supervised framework. It's not easy to find an exact origin of ensemble learning, since it can be seen as an evolution of the **wisdom of the crowd**. This phenomenon can be summed up as:

the average value of multiple estimates tends to be more accurate than any one single estimate; [Joshua L. Fiechter, 2021].

The famous **Condorcet's jury theorem** has many links to ensemble learning. The theorem considers a scenario where a group tries to reach a binary decision by majority vote (one of the two outcomes is obviously correct), and each voter has a probability p of voting for the correct decision which is independent of all the others.

The theorem states that if p is greater than 1/2, then adding more voters increases the probability that the majority decision is correct, and in the limit, it's sure that the majority votes correctly.

Francis Galton in 1907 gave a concrete example of wisdom of crowds.

During a show in Plymouth, in a weight-estimation game wherein 787 people were trying to estimate the weight of an ox, Galton collected responses of the participants.

The average estimate was 1197 lb; like the weight of the ox, so the crowd had perfectly assessed the weight, while for what concerns the individual estimates:

that the individual estimates are abnormally distributed in such a way that it is an equal chance whether one of them, selected at random, falls within or without the limits of -3 · 7 per cent. and +2 · 4 per cent. of their middlemost value.

In other terms, it's clear that in this case the crowd is much more precise than a single estimate.

Obviously, a single experiment, cannot guarantee that a crowd will always outperform a single expert. Anyway, according to [Suroweci, 2005] the wisdom of the crowd is likely to be better than a single "expert" when the following criteria are met :

- Independence of opinions.
- Aggregation: there exists a way to unify private judgments into a collective one.
- Diversity of opinions: Everyone should have private informations, even unorthodox interpretations of facts.
- Decentralization: Everyone can specialize and make predictions based on local information.

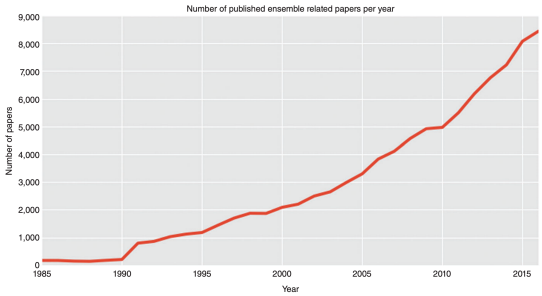


Figure 3.4: Number of papers regarding ensemble learning over the years.picture taken from: [Dong:2020tx]

3.0.4 Why ensemble learning works

Has been shown empirically many times that ensemble learning improves prediction performance. The causes are different, and mainly four can be identified [Polikar, 2006] [Sagi and Rokach, 2018]:

- **The statistical problem**: the problem that arises when the algorithm cannot span all the hypothesis space, because it's too large, and the amount of datas is not sufficient. In that case an algorithm could find several different hypothesis which yields to the same classification performance. A voting system will reduce the effects of that problem.
- **The representation problem**: The representation problem happens when the hypothesis space do not contain a good approximation of the real function f that we are trying to approximate. Hopefully a combination of classifiers could "span" a larger hypothesis space, providing a better

estimate of f. We can show this concept by taking in consideration a case where we have a classifiers that separates linearly datas, for example SVM, or single-split decision tree. In that situation, combining different "voters" can generate some non-linearity in our model. This phenomenon can be presented graphically as follows:

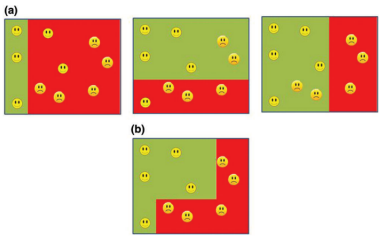


Figure 3.5: Picture taken from: [Sagi and Rokach, 2018]

This notion can be generalized to more articulated cases, where the decision boundaries are just too complex for one classifier, but not for the ensemble ones:

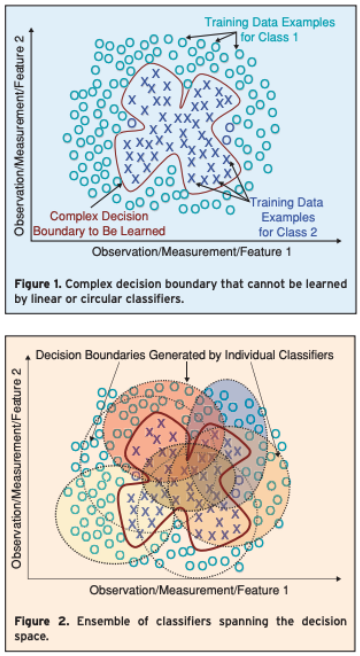


Figure 3.6: Picture taken from: [Polikar, 2006]

As it can be seen, combining prediction we allow for much more flexible

boundaries.

- **The computational problem**: sometimes the task of finding the best hypothesis is computationally intractable, so some heuristic methods are needed.

  The problem that arises in these situations is that the method risks to get stuck in local optima, while combining outputs reduces the risk of choosing the wrong local minima.

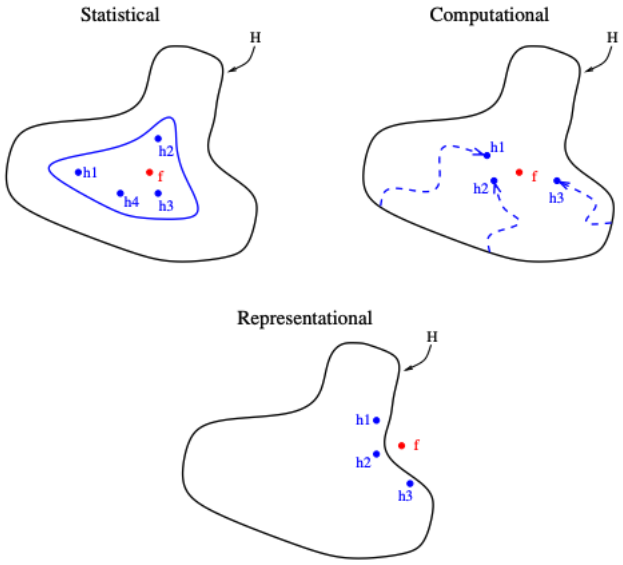


Figure 3.7: The three most relevant causes that make ensemble learning work.

The black curve represents the hypothesis space H, and the blue curve the set of hypotheses that yields to a good classification performance, meanwhile the point "f" is the true hypothesis, and the blue points the single classifiers.

Another important cause is the problem of **data fusion**: when combining different datasets, a single classification method may fail to learn the informations contained in them , if their nature is heterogeneous [Polikar, 2006]. For example, in a medical environment, it's difficult to use a single algorithm for a MRI scan, an ECG recording, a blood test..., while we can use a different methods for each on them, and then combine their outputs in a single prediction.

3.0.5 Combining voters

At this point, the already very simple scheme of an ensemble classifiers should be clear.

Starting from different output generated by the single committee, we combine them in a single output.

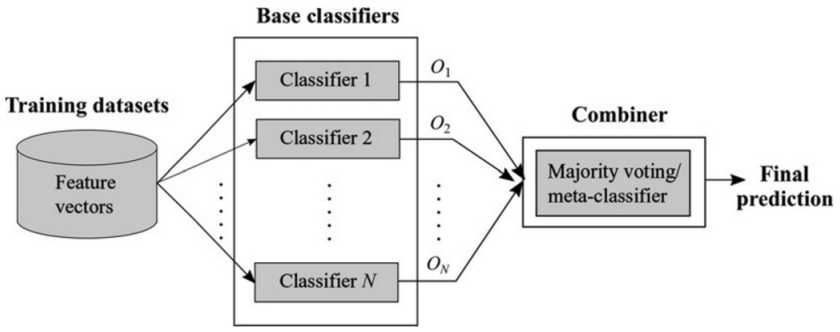


Figure 3.8: Picture taken from: [Dietterich, 2000]

The two open questions are:

- What kind of output should the voters produce?
- How can we combine these votes?

Considering a binary classification case, the first answer is quite straightforward, since the possible outputs are mainly two:

1. **The class label**.
2. **The class scores** (remembering that given one class score we can deduct the other class one).

As far as it is concerned the second question, the situation is much more complex, since there exist virtually unlimited ways for combining outputs into a single one. It's important to remark that some methods outputs probabilities, some other instead produce a score, but both are simply a measure of the confidence the model has in the prediction made.

We can anyway enumerate some of them:

- Voting ensembles:
  - **Hard voting**: each method outputs a label, then the sample is assigned to the majority class

  - **Soft voting**: the class that received the largest sum of probability or scores is chosen. The main point is that each classifier is considered equally good or important
  - **Weighted voting**: the vote classifiers whose performance is better has a greater value

- Advanced ensembling technique:
  - **Blending**
  - **Stacking**

The last two methods deserve a deeper explanation compared to the more naive voting system.

Stacking is a 2-level learning method. It consists in mainly two phases: briefly, during the first one , several machine learning algorithm (**base-models**) are implemented for generating prediction using a **cross-validation** , in a score-like form. These prediction are collected in a new training set, and finally this one is used for building another model whose features are precisely those scores (this model is known as **meta-model**).

Getting into details, the stacking method is composed by these steps:

1. The dataset is split into training-set with n samples, and test-set
2. using a k-fold validation, each of the chosen M algorithm is implemented.

   All the k-1 fold are used to generate a prediction on remaining fold (as usual this process is repeated k times, changing always the selected folds). Now we have a dataset with M features and n samples.
3. At this point, we need a test-set that obviously has the same features of the training set, which we remember to be the outputs of the base-models. To do so we have to train each base-model and make prediction on the test-set

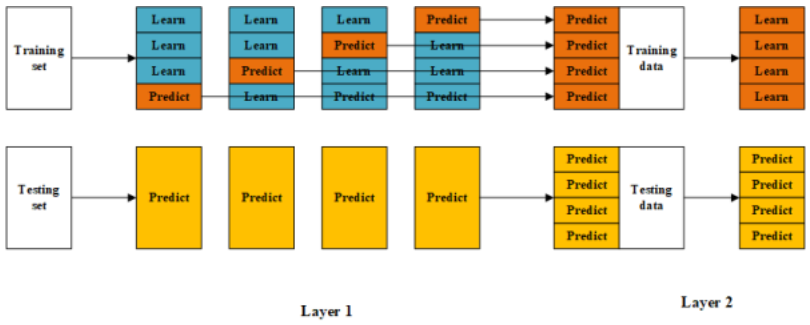


Figure 3.9: Step 3 and 4. picture taken from: CodeProject.com

4. Summing up, now we have a training set composed by n samples and M feature, that are outputs of K-fold validated base-models , and a test-set whose features are prediction generated by each model trained on the **entire training-set**, and applied to the test. Now we just have to use another algorithm, for example a logistic regression to generate the final prediction.

The **blending** method is very similar, except for the fact that the base models do not use a k-fold validation, but a hold-out set approach.

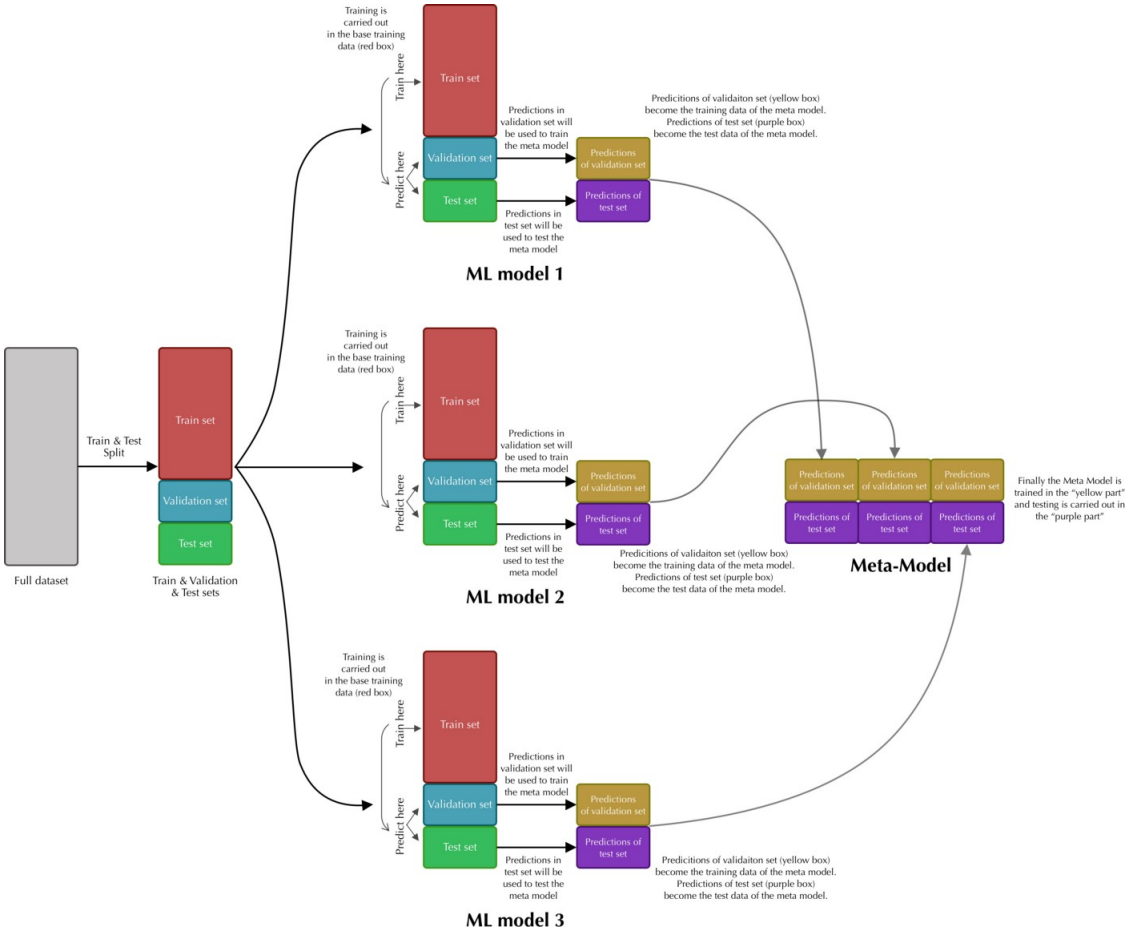


Figure 3.10: Blending method. Credit: towardsdatascience.com

The main difference is that same samples are in a certain way lost, respect the stacking method.

Chapter 4

Ensemble learning application

4.0.1 german credit dataset

Stacking

In this section, we will retrace the steps that compose the stacking method, already outlined before.

We start splitting the german credit dataset into training-set (70%) and test-set (30%).

Then, the first step is the base-models building.

Five learners are chosen, and optimized (the same use in the previous chapters):

1. Logistic regression
2. Naive Bayes
3. KNN
4. SVM
5. Boosted Trees

Each algorithm outputs a **score** (obtained through a 5-fold validation) that is a measure of the confidence of the prediction: the higher the score, the more the algorithm is sure of the prediction. For logistic regression and Naive Bayes, the score is simply the probability, for the other is a value that can be both positive and negative. In the SVM case for example, the score absolute value is the distance from the hyper-plane, and the sign represents the side of the plane where the point lies.

We could transform all these scores in a probabilistic fashion, but this process is useless since for the meta-model would make no difference if input are scores or probabilities.

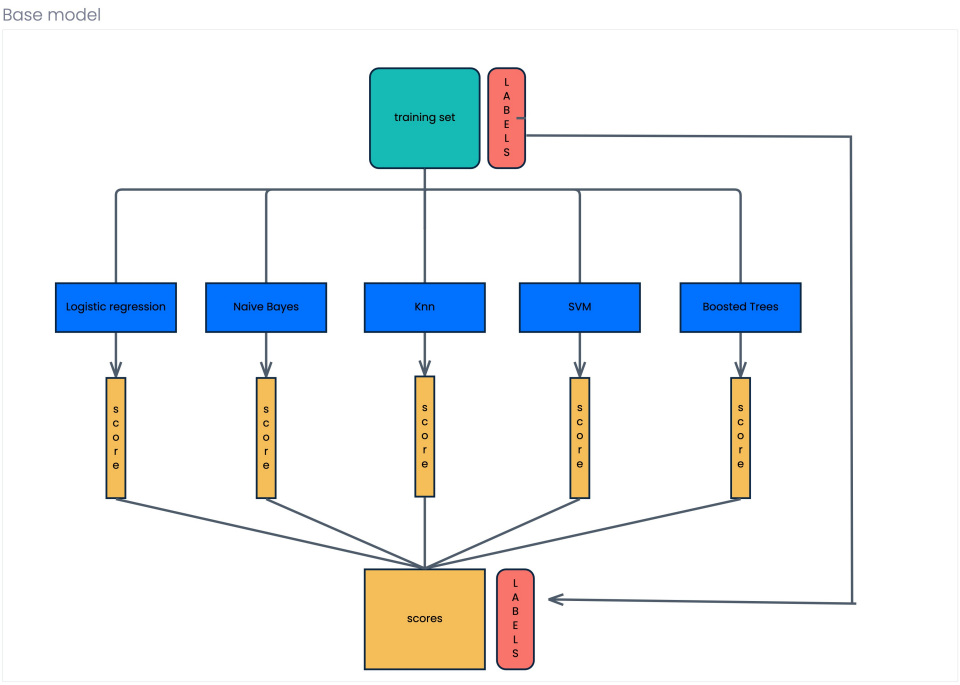


Figure 4.1: A snippet of the process

Here we can see the results produced on the first step, which is the yellow dataset in the picture above:

LogReg	NB	KNN	SVM	Boost	Class
0.0494098	0.0177445	0.155506	-0.80010524	-0.425097463680275	1
0.5710981	0.852916322	0.354706	0.482189	0.8224137384	2
0.01594290	0.01968295	0.1982274	-0.6402284	-1.117215059	1
0.2738027	0.93256559	0.367902	0.52913145	0.3513957344	1
0.831816	0.7245028045	0.391607	0.85569	0.5702024	2
0.193445	0.918067	0.2511766	-0.072189	-0.40364441	1

Table 4.1: Scores of the first 6 samples

Now basically we have to use this dataset to train a model, that will make prediction on the test set.

The point is that the test is not in the same format of this training-set, since it is composed by the original features, and not by scores.

Long story short, we have to use the models we builded to transform that

set into scores, i.e we have to predict them starting from the original attributes.

At this point, after choosing now a new algorithm (in our case SVM is picked) we have all we need to build the meta-mode:

- A training set: whose dimension is $N_{training_samples} \times N_{models}$ plus the class labels, so 700×6
- A test-set : which is$N_{test_samples} \times N_{models}$ plus the labels, i.e 300×6

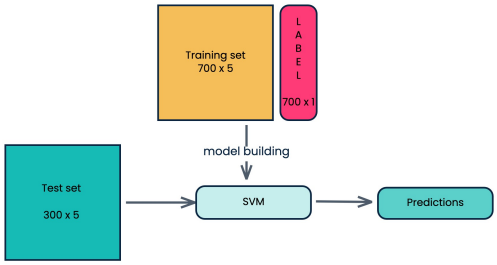
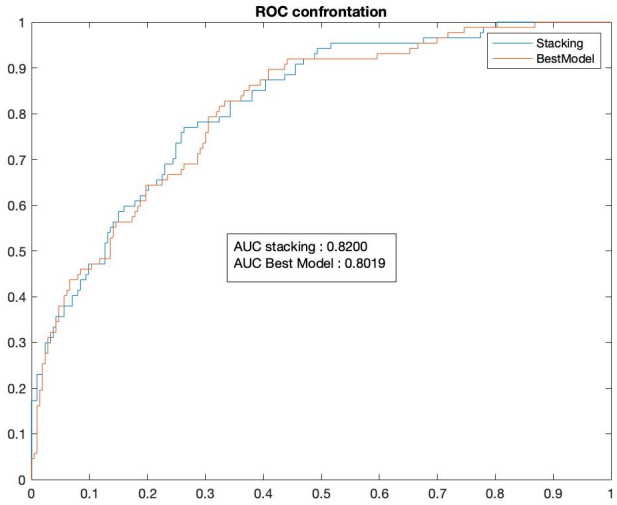


Figure 4.2: Meta-model building

Now we can see the result obtained, confronting them with the learner which **performed the best** previously.



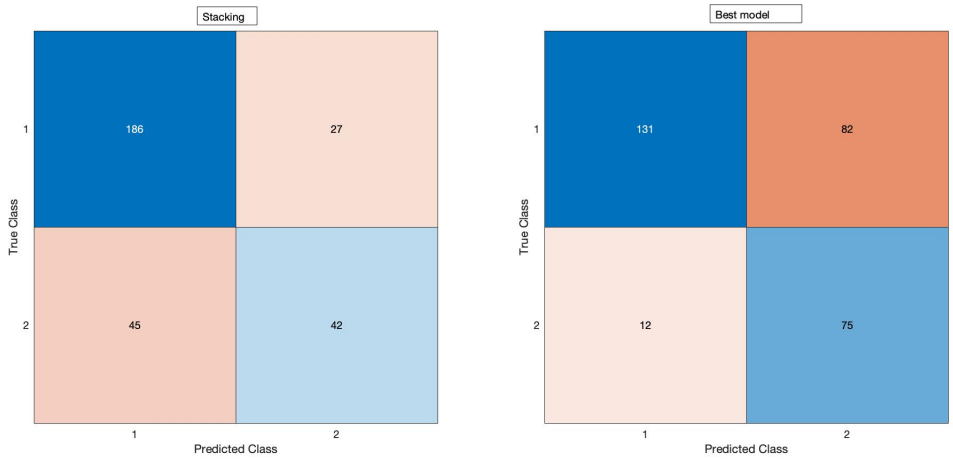


Figure 4.3: Stacking ROC curve confrontation

The improvement in performance is evident, and it remains significant even after repeating the process several times.

Another thing to pinpoint is the sensitivity:

$$Sensitivity_{Stacking} = \frac{42}{42+27} = 0.60$$

$$Sensitivity_{BestMod} = \frac{75}{75+82} = 0.47$$

Sensitivity is a very important value in our setting. It can be reformulated as: given that an individual is classified as a bad creditor, how probable is that he is a bad creditor for real?

in the last case the probability is 0.47 in the first 0.6 : a great improvement.

Hard-voting

Hard-voting is quite straight-forward: each one of the 5 committee (algorithm) outputs for each sample a predicted class, then that sample is assigned to the class that obtained the most votes.

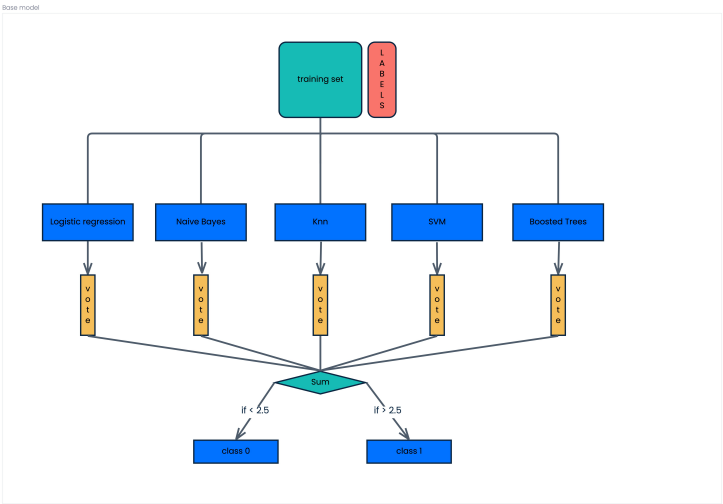


Figure 4.4: Hard voting process

the results obtained are:

Since, in a hard-voting setting it's nonsensical to produce scores (the only possible values would be : $0, 0.2, 0.4, 0.6, 0.8$ and 1) the confrontation is made only by confusion matrices:

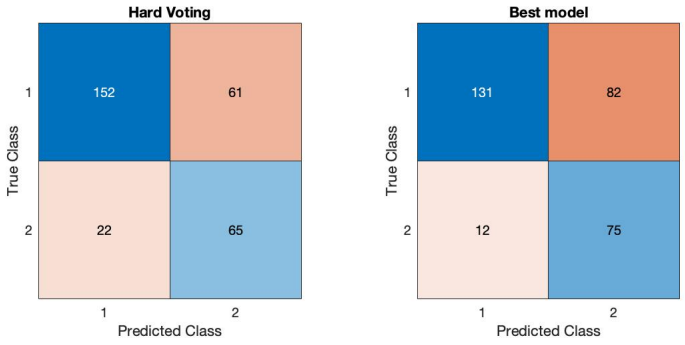


Figure 4.5: Hard voting results

Also remarking that sensitivity increases from 0.47 to 0.52. A significant improvement, even if not as good as the stacking model.

4.0.2 Loan dataset

Blending

In this section we will repeated the same steps taken in the section above, the only difference is that hold-out set approach is used, instead of a cross-validation one. The choice of using blending instead of stacking is justified by the dimensions of the loan dataset in fact it has 243974 samples, while the German one has "only" 1000 samples. In the Loan dataset case, the cross-validation is not necessary, since we have a sufficient number of samples to accurately estimate the model performance with a validation set approach, moreover the latter allows to save a lot of computational time.

First, we take from the test-set a certain number of samples, in our case 30% (same hold-out procedure outlined in the first chapters).

Then we build a model for each of the 5 learners, starting from the test-set. Successively we use those model to predict the score for each element belonging to the validation set.

Finally we build the meta-model, that has as "input" the scores previously obtained (and obviously the label), and outputs a prediction for the test set.

In particular, here Logistic regression is chosen, since it seems to achieve good performance in a short time. The figure below depicts the steps taken:

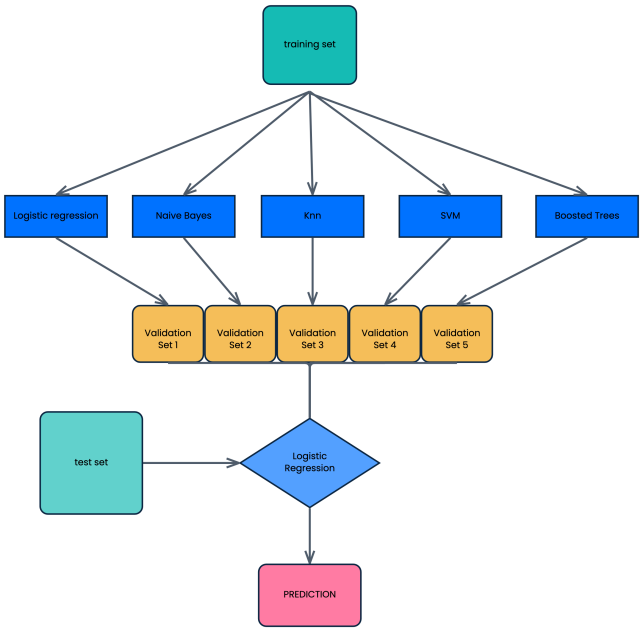


Figure 4.6: Blending process

The results obtained are the following:

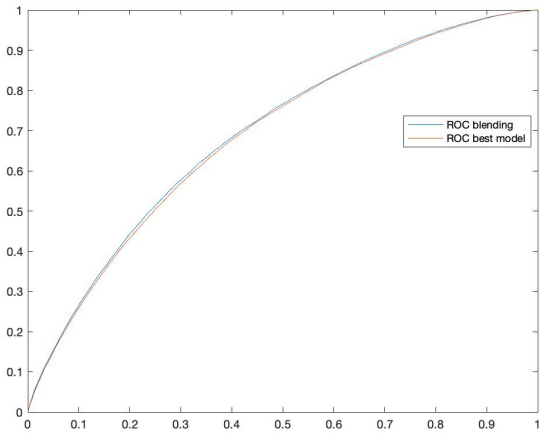


Figure 4.7: Blending ROC curve confrontation

With an increase in AUC from 0.683 to 0.692.

Normalizing the confusion matrix along the column, we can highlight the improvement of the performance, recalling that in this setup, the value at the bottom right is the Sensitivity of the model:

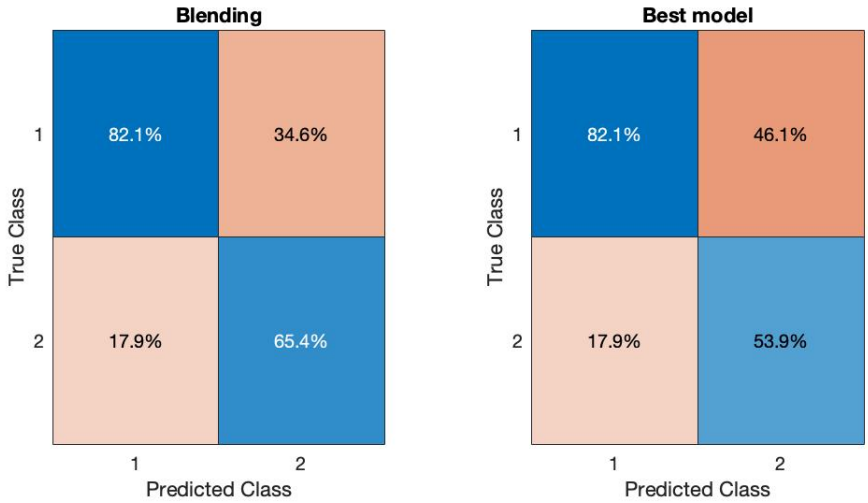


Figure 4.8: Blending results

Sensitivity also gets significantly better:

$$Sensitivity_{Blending}0.654$$

$$Sensitivity_{BestMod} = 0.539$$

Hard voting

The hard voting case is completely analogous the the German dataset case.

Figure 4.9: Results of Hard voting

$$Sensitivity_{HardVoting} = 0.619$$

$$Sensitivity_{BestMod} = 0.27$$

As can be seen the performance gets a lot better. Another thing to remark is that hard-voting is far quicker than blending, and do not need to split the original in three parts: it is sufficient to build the learners on the training-set, make prediction on the test-set, and finally average them into a single one.

As said before, in the German dataset case, it's not useful to produce scores as outputs of the model, since in the hard-voting case it's the only possible values would be : $0, 0.2, 0.4, 0.6, 0.8$ and 1.

Concluding, we can observe that in both datasets that ensemble learning improves performance:

	Accuracy	Sensitivity	Specificity	Precision	F1-score	AUC
Stacking German Dataset	0.7733	0.6462	0.8085	0.8920	0.7494	0.82
Best Model German Dataset	0.7033	0.4917	0.8444	0.7136	0.5822	0.79
Blending Loan Dataset	0.8205	0.6543	0.8206	0.9998	0.7910	0.70
Best Model Loan Dataset	0.8205	0.5393	0.8208	0.9994	0.7005	0.68

Table 4.2: Ensemble learning classification performance.

Conclusions

The conclusion of this work leads to some final considerations. First of all, the comparison between the various classification methods cannot always be exhausted in a mere numerical comparison, in fact, "performance" is strongly dependent on what are trying to optimize.

In an economical setting, we can ideally generate a misclassification cost matrix , where entries are deducted by the economic cost of false positive or false negative errors.

Beside these considerations, AUC and sensitivity are two strong indicators in our setting.

In the German credit dataset, the **SVM** was the best classifier under this point of view.

In the LoanP2P one, **Logistic regression** and **Ensemble trees** performed the best, but the grater interpretability of GLMs makes the first option much more preferable.

However, a relevant result is the **improvement in the quality of the model achieved by ensemble classifiers** (especially blending and stacking, that besides some technicalities, are conceptually equivalent). AUC of ensemble models outperformed in both cases the best out of the five learners.

Furthermore, sensitivity, which we remark to be a very important indicator in our setting, increased substantially in each situation.

However, it's important to point out that EC can increase by a lot computational time , especially with respect to building only one model. Nevertheless, considering a very common case, when we test various algorithms to choose the one that seems to perform the best, combining their outputs into a single one requires only one additional model, so not too much additional time.

In conclusion, ensemble classifiers are a very strong tool that should be considered when dealing with machine learning in an economic setting.

Bibliography

[Coffman and Chandler,] Coffman, J. Y. and Chandler, G. G. 1 applications of performance scoring to accounts receivable management in consumer credit.

[Deisenroth et al., 2020] Deisenroth, M. P., Faisal, A. A., and Ong, C. S. (2020). *Mathematics for Machine Learning*. Cambridge University Press.

[Dietterich, 2000] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Dong et al., 2020] Dong, X., Yu, Z., Cao, W., Shi, Y., and Ma, Q. (2020). A survey on ensemble learning. *Frontiers of Computer Science*, 14(2):241–258.

[Feurer and Hutter, 2019] Feurer, M. and Hutter, F. (2019). *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham.

[Freund and Schapire, 1999] Freund, Y. and Schapire, R. (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*.

[Galton, 1907] Galton, F. (1907). Vox popili. *Nature*, 75.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer.

[Joshua L. Fiechter, 2021] Joshua L. Fiechter, N. K. (2021). How the wisdom of crowds, and of the crowd within, are affected by expertise.

[Khashman, 2010] Khashman, A. (2010). Neural networks for credit risk evaluation: Investigation of different neural models and learning schemes. *Expert Systems with Applications*, 37(9):6233–6239.

[Kiourt et al., 2020] Kiourt, C., Pavlidis, G., and Markantonatou, S. (2020). Deep learning approaches in food recognition.

[Polikar, 2006] Polikar, R. (2006). Polikar, r.: Ensemble based systems in decision making. ieee circuit syst. mag. 6, 21-45. *Circuits and Systems Magazine, IEEE*, 6:21 – 45.

[Sagi and Rokach, 2018] Sagi, O. and Rokach, L. (2018). Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*, 8(4):e1249.

[Shalev-Shwartz and Ben-David,] Shalev-Shwartz, S. and Ben-David, S.

[Shi et al., 2022] Shi, S., Tse, R., Luo, W., D'Addona, S., and Pau, G. (2022). Machine learning-driven credit risk: a systemic review. *Neural Computing and Applications*, 34(17):14327–14339.

[Suroweci, 2005] Suroweci, J. (2005). The wisdom of crowds. *Nature*.

[Tsai et al., 2014] Tsai, C.-F., Hsu, Y.-F., and Yen, D. C. (2014). A comparative study of classifier ensembles for bankruptcy prediction. *Applied Soft Computing*, 24:977–984.

[Tsai and Wu, 2008] Tsai, C.-F. and Wu, J.-W. (2008). Using neural network ensembles for bankruptcy prediction and credit scoring. *Expert Systems with Applications*, 34(4):2639–2649.

[Twala, 2010] Twala, B. (2010). Multiple classifier application to credit risk assessment. *Expert Systems with Applications*, 37(4):3326–3336.

[VS, 2022] VS, M. M. D. (2022). Pattern recognition an algorithmic approach.

[Wang et al., 2011] Wang, G., Hao, J., Ma, J., and Jiang, H. (2011). A comparative assessment of ensemble learning for credit scoring. *Expert Systems with Applications*, 38(1):223–230.