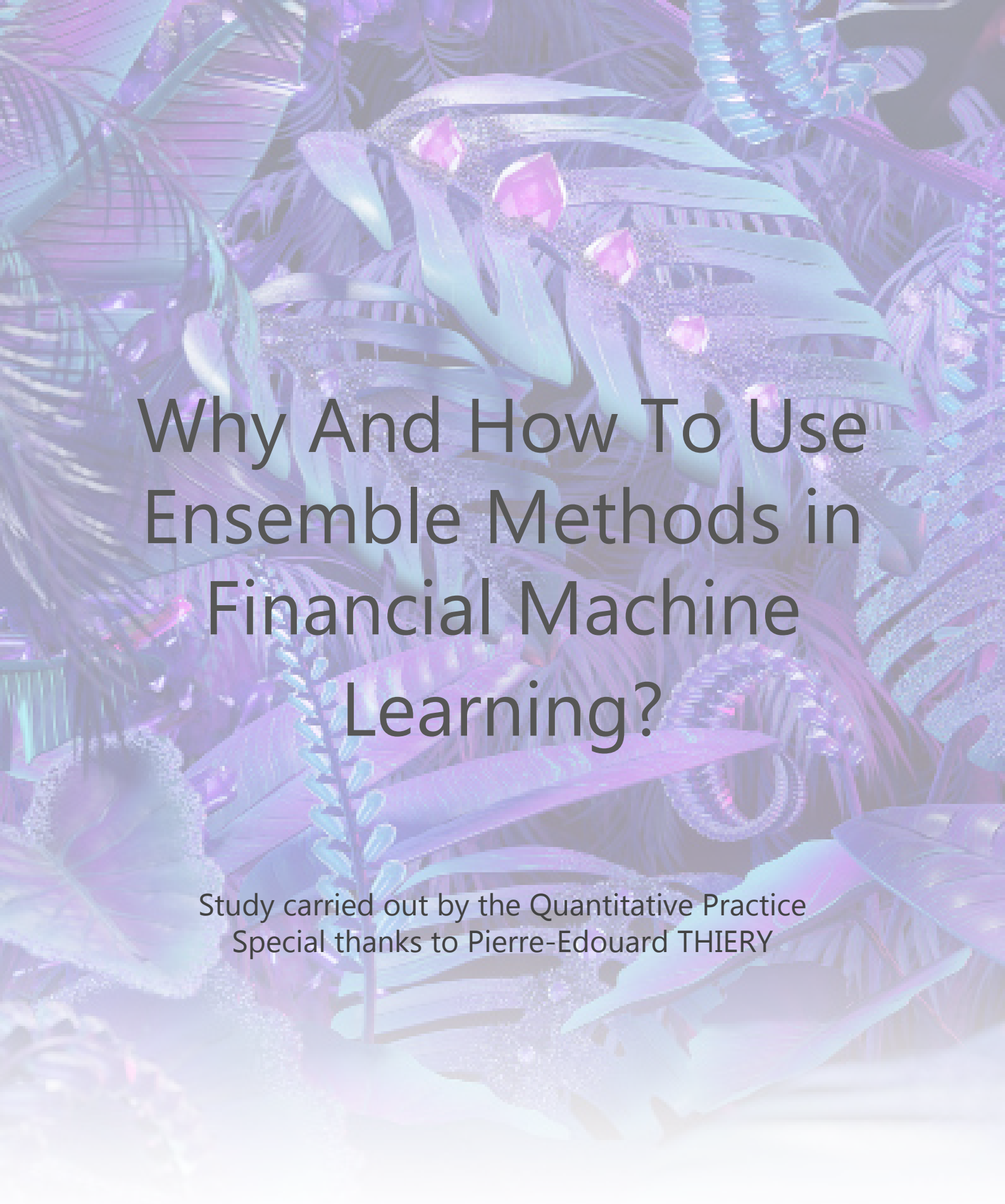

Why And How To Use Ensemble Methods in Financial Machine Learning?

Study carried out by the Quantitative Practice
Special thanks to Pierre-Edouard THIERY

awalee

JANVIER 2021

Summary

Introduction

Machine Learning techniques are gaining currency in finance nowadays; ever more strategies rely on Machine Learning models such as neural networks to detect ever subtler signals. Nonetheless this rising popularity does not come without any shortcoming, the most widespread one being the so-called "overfitting", when models tend to learn by heart from the data and are thus unable to face unknown data. In our opinion, using Machine Learning algorithms in finance without a deep understanding of their inner logic is highly risky: promising initial results are often misleading, the real-life implementation being much disappointing due to the lack of comprehension of what is really happening.

In this paper we decide to focus on a specific category of Machine Learning meta-algorithms: the ensemble methods. The ensemble methods are called meta-algorithms since they provide different ways of combining miscellaneous models of Machine Learning in order to build a stronger model. Those techniques are well-known for being extremely powerful within many areas; however we believe that it is important to understand what their advantages are from a mathematical point of view to make sure that they are used purposefully when dealing with a financial Machine Learning problem.

First we set forth how ensemble methods work from a general point of view. We then present the three sources of errors in Machine Learning models before explaining what the advantages of bagging over boosting are in finance, and how to use efficiently bagging.

1 From A Single Model To Ensemble Methods: Bagging and Boosting

Machine Learning is mainly premised on predictive models. Once devised, a model is then trained thanks to available data; its purpose is to predict the output value, also known as the outcome, corresponding to new input data. Formally we can define a predictive model in the following manner:

Definition 1 (Predictive Model)
A predictive model is defined as an operator $\mathcal{M}$, based on meta-parameters denoted M, and on parameters denoted P. It uses a set of inputs, denoted $\vec{x} \in \mathbb{R}^m$, to compute an output, denoted $O \in \mathbb{R}$, seen as the predicted value. We can write:

$$\begin{array}{rccc} \mathcal{M}(M;P;\bullet) & : & \mathbb{R}^m & \to & \mathbb{R} \\ & & \vec{x} & \mapsto & O = \mathcal{M}(M;P;\vec{x}) \end{array}$$

Thus the idea of a predictive model is only to predict a value based on several features which are the inputs. If $\mathcal{M}$ is considered to be "the machine", the learning part consists in estimating the parameters P that enable us to use the model. The metaparameters M are chosen, and often optimized, by the user.

For instance, a neural network is a predictive model. The shape of the neural network, i.e. the number of layers and the number of neurons in each layer as well as the functions within each neuron, forms the metaparameters M. The parameters of the neural network are the weights for each link between two neurons from two consecutive layers. Those parameters are estimated thanks to a training set $\mathcal{D}$: formally $P = P(\mathcal{D})$. As of now, since the training sets which are used are of the utmost importance, we always write $P(\mathcal{D})$ to clearly mention which training set is used to find the parameters of a given model.

The gist of ensemble methods is fairly simple: we combine several weak models to produce a single output. As of now and for the rest of this paper, the number of models is denoted N.

The ensemble methods can be divided into two main sets: the parallel methods, where the N models are independent, and the sequential methods, where the N models are built progressively.

• A Parallel Method: Bootstrap Aggregating

In this section, we set forth the bootstrap aggregating method, also known as "bagging", which is the most widespread of the parallel methods [1]. As of now, we assume a training set, denoted $\mathcal{D}$, is at our disposal:

Definition 2 (Data Set)
A data set $\mathcal{D}$ is a set of couples of the following form

$$\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^m \times \mathbb{R},\ 1 \leq i \leq n\}$$

where n is the cardinal of $\mathcal{D}$. For the i-th element in the data set, $\vec{x}_i \in \mathbb{R}^m$ is called the vector of the m features, and y_i is called the output value.

To carry out the bagging, we construct N models $\mathcal{M}^j$ with $1 \leq j \leq N$. To do so, we consider a generic model $\mathcal{M}(M;\bullet;\bullet)$, i.e. a predictive model whose metaparameters are fixed, for instance a neural network with a given shape. In order to get N models, the generic model will be trained with N different training sets $\mathcal{D}_j$:

$$\mathcal{M}^j = \mathcal{M}(M; P(\mathcal{D}_j); \bullet)$$

Thus, $\mathcal{M}^j$ is now a function which, for every input vector $\vec{x} \in \mathbb{R}^m$ output a real value $y = \mathcal{M}^j(\vec{x})$.

The N models are different since they are not trained on the same training set; it means that the set of parameters will be different; therefore we will have different output values for the same input vector of features $\vec{x}$.

The N training sets are created thanks to the data set $\mathcal{D}$. The size of the training sets is chosen by the user and denoted K with $K < n$, otherwise the training sets would necessary contain redundant information. The K elements of the training set $\mathcal{D}_j$ for $1 \leq j \leq N$ are sampled in $\mathcal{D}$ with replacement: for $1 \leq j \leq N$

$$\mathcal{D}_j = \left\{\left(\vec{x}_{u(j,k)}, y_{u(j,k)}\right),\ 1 \leq k \leq K\right\}$$

where $u(j,k)$ is a uniform random variable in $[1,n]$.

Once the N models $\mathcal{M}^j$ have been trained, they are combined into the final model $\mathcal{M}^f$. For instance, if we consider a regression problem, meaning that the output value y does not belong to a predetermined finite set of values:

$$\begin{array}{rccc} \mathcal{M}^f & : & \mathbb{R}^m & \to & \mathbb{R} \\ & & \vec{x} & \mapsto & \frac{1}{N}\sum_{j=1}^{N}\mathcal{M}^j(\vec{x}) \end{array}$$

If we consider a classification problem, meaning that the output value y belongs to a finite set of values $\mathbb{S}$, the output value of the final model is determined by a vote of the N models $\mathcal{M}^j$: the outcome which appears the most among the N output values produced by the N models is the outcome of the final model.

Such a model can then be tested on a test set of data as it is usually done for every model of Machine Learning.

It is also worth noticing that there are many bagging approaches, which all derive from the general principle as presented above. Even though we do not delve into the details, we can for instance mention the so-called "feature bagging", where each one of the N models is trained using only a specific subset of features.

• A Sequential Method: Boosting

Sequential methods consist no longer in using a set of N independent models, but instead a sequence of N models, where the order of the models matters:

$$\underbrace{\{\mathcal{M}^1,\dots,\mathcal{M}^N\}}_{\text{a mere set: no order}} \to \underbrace{(\mathcal{M}^1,\dots,\mathcal{M}^N)}_{\text{a sequence}}$$

So we have to construct the sequence of the N models, beginning with the first one, which will then sway how the second one is defined, and so on and so forth. In the rest of this section we present some of the principal ideas of boosting.

First, as with bagging, we assume we have a training set made of n elements and denoted $\mathcal{D}$. If we choose to consider a generic model $\mathcal{M}(M;\bullet;\bullet)$, we can train a first model:

$$\mathcal{M}^1 = \mathcal{M}(M;P(\mathcal{D});\bullet)$$

For every element within $\mathcal{D}$ we can compute $\mathcal{M}^1(\vec{x}_i)$ and compare it to the outcome y_i.

To devise the second model $\mathcal{M}^2$, we are going to train the generic model $\mathcal{M}(M;\bullet;\bullet)$ on a new training set $\mathcal{D}_1$: the new training set derives from $\mathcal{D}$. It contains $K < n$ elements, as will all the subsequent training sets.

We attribute to each element within $\mathcal{D}$ a weight depending on how far y_i is from $\mathcal{M}^1(\vec{x}_i)$. The more important the error, the higher the weight associated with an element. We then use those weights to randomly sample $\mathcal{D}$ in order to generate the new training set $\mathcal{D}_1$.

$$\mathcal{M}^2 = \mathcal{M}(M;P(\mathcal{D}_1);\bullet)$$

The process is then exactly similar. Thanks to the error of the second model, we can compute for each element within $\mathcal{D}$ a new weight. Those weights are used to create a new training set: we sample $\mathcal{D}$ using the new weights to get $\mathcal{D}_2$, which will then be used to train model $\mathcal{M}^3$ and so on.

It is then possible to define the final model $\mathcal{M}^f$ as a weighted sum of the N models $\mathcal{M}^j$, where the weight associated with a given model is derived from the error of this model on the data.

We have only presented the main ideas of boosting; the simplest implementation of those guidelines is probably the AdaBoost algorithm [2].

2 The Three Errors Of A Machine Learning Model

A Machine Learning model can suffer from three sources of error: the bias, the variance and the noise. It is important to understand what lies behind those words in order to understand why and how ensemble methods can prove to be helpful in finance.

The bias is the error spawned by unrealistic assumptions. When the bias is particularly important, it means that the model fails to recognize the important relations between the features and the outputs. The model is said to be underfitted when such a case occurs.

Figure 1 displays a model with an important bias. The dots represent the training data, which obviously do not exhibit a linear relation. If we assume that there is a linear relationship between the features and the outcomes, such a model clearly founders to recognize any relation between the former and the latter.

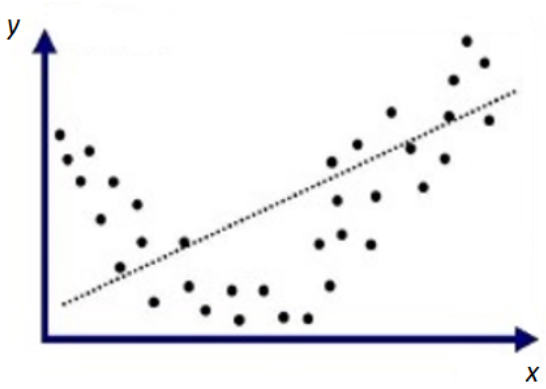

Figure 1: Underfitted model

The variance stems from the sensitivity to tiny changes in the training set. When variance is too high, the model is overfitted on the training set: there happens a "learning-by-heart" situation. This explains why even a small change in the training set can lead to widely different predictions.

Figure 2 displays an overfitted model: instead of finding a robust relationship between the features and the outcomes, the model only replicates what it has learned on the training set.

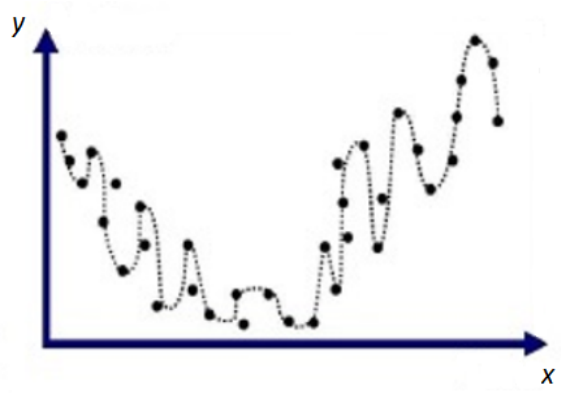

Figure 2: Overfitted model

The third source of error in a Machine Learning model is the noise, which is the error caused by the variance of the observed values, due, for instance, to measurement errors. This is the irreducible error.

From a mathematical point of view, those three types of errors can be apprehended in the following manner. We assume we have at our disposal a training set $\mathcal{D}$ made of n observations. We also assume there exists an unknown function f such that:

$$y = f(\vec{x})$$

where y is a generic outcome and $\vec{x}$ a vector of features. Since our training set exhibits some noise, we assume that, for $1 \leq i \leq n$:

$$y_i = f(\vec{x_i}) + \epsilon_i$$

ϵ is noise: it represents the fact that the available data do not perfectly reproduce the function f we would like to estimate. It is pretty natural to assume that it is independent from the feature x and that $\mathbb{E}[\epsilon] = 0$.

Thanks to our training set, we estimate a function $\hat{f}$ that best fits the unknown function f in the sense that it makes the variance of the estimation error minimal.

$$\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[(f(x) - \hat{f}(x) + \epsilon)^2\right]$$

$$= \mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2 + \epsilon^2 + 2\epsilon\left(f(x) - \hat{f}(x)\right)\right]$$

$$= \mathbb{E}\left[\left\{\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right] + \mathbb{E}\left[\hat{f}(x)\right] - f(x)\right\}^2 + \epsilon^2\right]$$

$$+ 2\mathbb{E}[\epsilon]\,\mathbb{E}\left[f(x) - \hat{f}(x)\right]$$

Using the above-mentioned assumptions regarding ϵ, the second term in the last equality is zero.

$$\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$$

$$= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right]$$

$$+ 2\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)\right]$$

$$+\mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)^2\right]$$

In the above computations, x is a mere constant; the randomness is carried by $\hat{f}$, which is built thanks to the training set, whose components can be seen as random variables. So, in the last term, $\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)^2$ is a constant. Besides:

$$\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\underbrace{\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)}_{constant}\right]$$

$$= c_0\,\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\right] = 0$$

Thus we can write:

$$\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right]$$

$$= \underbrace{\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)^2}_{bias} + \underbrace{var\left\{\hat{f}(x)\right\}}_{variance} + \underbrace{\mathbb{E}\left[\epsilon^2\right]}_{noise}$$

It is important that the reader bears in mind that the randomness in the above calculations is carried by the estimated function $\hat{f}$: $\hat{f}$ depends on the elements within the training set, and those elements can be seen as random variables. We draw new variables by measuring the features and the associated outcomes. We can draw a parallel with what is done to estimate the cumulative distribution function of an unknown random variable X when we have n independent drawings of the variable $X : \{X_1, \ldots, X_n\}$. The cumulative distribution function can be estimated by:

$$\hat{F}(t) = \frac{1}{n}\sum_{i=1}^{n} 1(X_i \leq t)$$

So, for a given t, $\hat{F}(t)$ can be seen as a random variable depending on n iid random variables denoted X_i. The philosophy is the same when we estimate the function f in the above calculations.

3 Why Is It Better To Rely On Bagging In Finance?

Bagging is a technique which significantly reduces the variance of a model. To see that, let us assume we consider a regression problem. We have performed the bagging with N models. So for a fixed vector of features $\vec{x}$, we have N estimations of its outcome:

$$\mathcal{M}^j(\vec{x})$$

and the final estimation is given by:

$$\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^i(\vec{x})$$

This last quantity plays the role of $\hat{f}(\vec{x})$. So we compute its variance:

$$var\left\{\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^i(\vec{x})\right\} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} cov\left(\mathcal{M}^i(\vec{x}),\mathcal{M}^j(\vec{x})\right)$$

As of now, for the sake of clarity, we denote $\sigma_{ij} = cov\left(\mathcal{M}^i(\vec{x}),\mathcal{M}^j(\vec{x})\right)$ and $\sigma_i^2 = var\left(\mathcal{M}^i(\vec{x})\right)$.

$$var\left\{\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^i(\vec{x})\right\} = \frac{1}{N^2}\sum_{i=1}^{N}\left(\sum_{j\neq i}\sigma_{ij}+\sigma_i^2\right)$$

We denote $\bar{\sigma}^2$ the average variance of a single model:

$$\bar{\sigma}^2 := \frac{1}{N}\sum_{i=1}^{N}\sigma_i^2$$

We would like to compare $var\left\{\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^j(\vec{x})\right\}$, which is the variance of the bagging estimator, with $\bar{\sigma}^2$, which is the expected variance when using a single model.

$$var\left\{\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^i(\vec{x})\right\} = \frac{\bar{\sigma}^2}{N} + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j\neq i}\sigma_{ij}$$

We also define $\bar{\rho}$ the average correlation between the N predictions of the N models:

$$\underbrace{\bar{\sigma}^2\bar{\rho}}_{average\ covariance} := \frac{1}{N(N-1)}\sum_{i=1}^{N}\sum_{j\neq i}\sigma_{ij}$$

Thus:

$$var\left\{\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^i(\vec{x})\right\} = \bar{\sigma}^2\left(\frac{1+(N-1)\bar{\rho}}{N}\right)$$

$$= \bar{\sigma}^2\left(\bar{\rho}+\frac{1-\bar{\rho}}{N}\right)$$

The above equation shows that bagging is effective to reduce the variance when $\bar{\rho} < 1$; if $\bar{\rho} \to 1$, then $var\left\{\frac{1}{N}\sum_{i=1}^{N}\mathcal{M}^i(\vec{x})\right\} \to \bar{\sigma}^2$. Therefore we see it is important to produce training samples which are as independent as possible in order to reduce $\bar{\rho}$. In our first section, we have set forth the simplest way of implementing the bagging: the N training sets are obtained by sampling uniformly the initial set $\mathcal{D}$ with replacement. Other techniques exist, which are particularly suited when dealing with financial data, for instance the so-called sequential bootstrapping.

• Sequential Bootstrapping

The adjective "sequential" is a bit dangerous in this context: it does not mean that the bagging will be turned into some sort of sequential ensemble method. Sequential bootstrapping only means that each training set $\mathcal{D}_i$, used to train the model $\mathcal{M}^i$, is constructed in a sequential manner: once a first observations has been added to $\mathcal{D}_i$, instead of merely replacing it in $\mathcal{D}$ before drawing uniformly the second observation which will join $\mathcal{D}_i$, the way this second observation is chosen will depend on the first observation. Then the third observation will depend on the first two ones, and so on.

The sequential bootstrapping is particularly suited to financial problems insofar it is a method which enables to take into account the particularities of financial data [3] [4], notably that the observations are made on a given time period. Let us start with our data set:

$$\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^m \times \mathbb{R},\ 1 \leq i \leq n\}$$

In finance, it is important to bear in mind that the observations are made thanks to given times t within a given stint, denoted $[0, T]$. If we consider a single element of our data set, $(\vec{x}_i, y_i)$, the features for the i-th element are computed thanks to observations within a time period $\left[t_{i,0}, t_{i,1}\right]$. For instance, if $\vec{x}$ is of size 2, the first component could be the average of the last ten daily returns, and the second component could be the historical volatility over those last ten days.

In this particular case, every t in $[0, T]$ corresponds to a given day; if the i-th element corresponds to a day t_i in $[0, T]$, then $\vec{x}_i$ is computed thanks to the observations of the price for days $t_i - 10, \ldots, t_i$, so $t_{i,0} = t_i - 10$ and $t_{i,1} = t_i$.

In order to perform the sequential bootstrapping, for each element, indexed by i, within the data set and each time t such that $1 \leq t \leq T$, we define:

$$1_{i,t} := \begin{cases} 1\ if\ \left[t_{i,0}, t_{i,1}\right] \cap [t-1, t] \neq \emptyset \\ 0\ otherwise \end{cases}$$

So $1_{i,t}$ aims at identifying when an overlap occurs : for the i-th element within the data set and a time t in our time horizon, if there is an overlap between the stint on which the features are computed and the time window defined by t and the previous time $t-1$, $1_{i,t}$ is worth one, zero otherwise.

Thanks to this framework, we can now present how a training set $\mathcal{D}_i$ is built before training the i-th model of our bagging. We start with:

$$\mathcal{D}_i = \emptyset$$

The first element which will join $\mathcal{D}_i$ is drawn uniformly in $\mathcal{D}$: each element in $\mathcal{D}$ has the same weight $\frac{1}{n}$. We denote $X_{i_1} = \left(\vec{x}_{i_1}, y_{i_1}\right)$ the first element uniformly drawn in $\mathcal{D}$:

$$\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{X_{i_1}\}$$

For the second draw, we would like to reduce the probability of drawing an element X_j which overlaps with X_{i_1}, i.e. $\left[t_{j,0}, t_{j,1}\right] \cap \left[t_{i_1,0}, t_{i_1,1}\right] \neq \emptyset$.

If $\mathcal{D}_i$ already contains κ elements, here is how we draw the $\kappa+1$-th element. To carry out the $\kappa+1$ drawing, we define the uniqueness of the element j for time t: $1 \leq j \leq n$, $1 \leq t \leq T$

$$u_{t,j}^{\kappa+1} = \frac{1_{t,j}}{1+\sum_{k:X_k\in\mathcal{D}_i} 1_{t,k}}$$

It is then possible to average the uniqueness over the time period:

$$\bar{u}_j^{\kappa+1} = \frac{\sum_{t=1}^{T} u_{t,j}^{\kappa+1}}{\sum_{t=1}^{T} 1_{t,j}}$$

This leads us to the new weights associated with each element within $\mathcal{D}$:

$$p_j^{\kappa+1} = \frac{\bar{u}_j^{\kappa+1}}{\sum_{k=1}^{n} \bar{u}_k^{\kappa+1}}$$

Using those weights, we can now draw a new element within $\mathcal{D}$: we denote it $X_{i_{\kappa+1}}$:

$$\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{X_{i_{\kappa+1}}\}$$

The main advantage of such a sampling method is that the training samples we obtain to train the N models are much closer to iid samples than the ones we obtain with a uniform sampling with replacement. This helps to reduce the average correlation $\bar{\rho}$ between the outcomes produced by the N models, meaning that the bagging is better at reducing the variance of the model.

So far we have mainly insisted on the interest of bagging; in particular we have shown how it is possible to adapt the common bagging in order to deal with financial data. What about boosting?

We do not delve into technical details when it comes to boosting since boosting is less legitimate in finance: boosting is good at correcting both bias and variance, nonetheless correcting bias comes at the cost of greater risk of overfitting.

In financial Machine Learning, there is nothing easier than to overfit a model: bagging is then a much more powerful tool in finance than boosting.

Conclusion

Ensemble methods are powerful tools in Machine Learning inasmuch as they can transform a set of weak learners into a stronger one. Those methods can be divided into two categories: the parallel methods and the sequential methods.

Due to its advantages, bagging, which is the most common of the parallel methods, is better suited to financial problems: in finance overfitting is a real plague, which can lead to disastrous results. Bagging is better at decreasing the overfitting of a model, whereas boosting, the most common technique in the field of sequential techniques, is better at preventing the underfitting.

Nonetheless, this does not mean that bagging should be implemented in finance without a preliminary reflection. Indeed financial data, which are time dependent, exhibit strong characteristics; they cannot be assimilated to iid data, as it is commonly admitted in Machine Learning. In this paper, we have set forth how sequential bootstrapping works. Machine Learning tools are not mathematical recipes which can be automatically in various situations; they demand a real comprehension of their inner logic. Therefore this paper shows how to tweak the common bagging technique into a financial bagging, which can prove to be very helpful when applied to financial data.

References

[1] Leo Breiman. Bagging predictors. *Machine Learning, 24*, pages 123–140, aug 1996.

[2] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory To Algorithms*. Cambridge University Press, 2014.

[3] Ingo Steinwart and Andreas Christmann. Fast learning from non-i.i.d. observations. *Neural Information Processing Systems Conference*, jan 2009.

[4] Martin Sewell. Characterization of financial time series. *UCL Department of Computer Science*, jan 2011.

A propos d'Awalee

Cabinet de conseil indépendant spécialiste du secteur de la Finance.

Nous sommes nés en 2009 en pleine crise financière. Cette période complexe nous a conduits à une conclusion simple : face aux exigences accrues et à la nécessité de faire preuve de souplesse, nous nous devions d'aider nos clients à se concentrer sur l'essentiel, à savoir leur performance.

Pour accomplir cette mission, nous nous appuyons sur trois ingrédients : habileté technique, savoir-faire fonctionnel et innovation.

Ceci au service d'une ambition : dompter la complexité pour simplifier la vie de nos clients.

«*Run the bank*» avec Awalee !

Contactez-nous

Ronald LOMAS
Partner
rlomas@awaleeconsulting.com
06 62 49 05 97

awalee

est une marque de

canopee group