

Stage réalisé à la recherche quantitative
d'Exane BNP Paribas

:

Software ingenieur

Guillaume Rousseaux
Tuteur: Yann Ait Mokhtar

Remerciements :

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Je tiens à remercier vivement mon maître de stage, **Mr AIT MOKHTAR, responsable de la recherche quantitative au sein d'Exane**, pour son accueil, le temps passé ensemble et le partage de son expertise au quotidien. Grâce aussi à sa confiance j'ai pu m'accomplir totalement dans mes missions. Il fut d'une aide précieuse dans les moments les plus délicats.

Je remercie également toute l'équipe présente pour leur accueil, leur esprit d'équipe et en particulier **Mme Eli**, qui m'a beaucoup aidé à comprendre certaines problématiques.

Plan du document :

1) PRESENTATION DU LOGICIEL

- 1.1. OBJECTIF
- 1.2. APPRENTISSAGE STATISTIQUE
- 1.3. PRISMES UN OUTIL NOUVEAU ET PROMETTEUR
- 1.4. FONCTIONNALITES A LA DISPOSITION DE L'UTILISATEUR
- 1.5. AUDIT TECHNIQUE
- 1.6. ENJEUX

2) TACHES REALISEES

- 2.1. DEMONSTRATION DE L'INTERET DE PRISMES DANS UNE GESTION DE PORTEFEUILLE

2.2. SEGMENTATION

2.2.1. *PARTITION SIMPLE*

2.2.1.1. METHODES

- 2.2.1.1.1. PREMIERE METHODE RIGIDE
- 2.2.1.1.2. DEUXIEME METHODE ADAPTATIVE
- 2.2.1.2. SIMPLIFICATION DE LA SERIE
- 2.2.1.3. MOVING AVERAGE
- 2.2.1.4. CALCUL VOLATILITE

2.2.2. *PARTITION GLOBALE*

2.3. FABRICATION DES ELEMENTS D'ENTREE DES ALGORITHMES

2.3.1. *CONSTRUCTION DE LA MATRICE DE PREDICTEURS*

2.3.1.1. A L'ECHELLE D'UN STOCK

- 2.3.1.1.1. OPTION QUI N'UTILISE PAS DE SEGMENTATION
- 2.3.1.1.2. OPTION LAST PRICE
- 2.3.1.1.3. OPTION DAILY CHANGE
- 2.3.1.1.4. OPTION MONTHLY RETURN
- 2.3.1.1.5. RETURN SANS PARTITIONS

2.3.1.1.2. OPTIONS AVEC PARTITIONS

- 2.3.1.1.2.1. OPTION DAILY RETURN PARTITION
- 2.3.1.1.2.2. OPTION DAILY VOLATILITE PARTITION

2.3.1.2. CONCATENATION DES MATRICES FORMATIONS MATRICE PREDICTEUR

2.3.2. *VECTEUR TARGET*

- 2.3.2.1. OPTION PERFORMANCE
- 2.3.2.2. OPTION SPREAD
- 2.3.2.3. OPTION SEGMENT
- 2.3.2.4. OPTION SHARP RATIO
- 2.3.2.5. OPTION SHARP RATIO AVEC FENETRE CONTRAINTE

2.3.3. *REUNIFICATION COHERENCE TEMPORELLE*

2.4. LEARNING

2.4.1. *ALGORITHME RANDOM FOREST CLASSIFIER*

- 2.4.2. *KNEIGHT CLASSIFIER*
- 2.4.3. *SUPPORT VECTOR MACHINE*
- 2.4.4. *ALGORITHME EXPERIENTALE*
- 2.4.5. *ASTUICES POUR AMELIORER L'EFFICACITE*
- 2.5. BACKTESTING
 - 2.5.1. *PRINCIPE*
 - 2.5.2. *HITScore*
 - 2.5.3. *INTERVALLE DE CONFIANCE*
 - 2.5.4. *RANKING PREDICTEUR*
- 2.6. *PREDICTION*
- 2.7. *CONCLUSION*

1)PRESENTATION DU LOGICIEL EXISTANT

1.1) objectif

PRISMS est un outil permettant aux investisseurs d'effectuer des décisions en gestion de portefeuille. L'objectif est d'avoir de l'avance sur l'évolution du marché en effectuant des prédictions dans un horizon de 5 à 50 jours. Il s'agit de prédictions de type binaire qui prédit le caractère montant ou descendant du prix d'un titre, ou d'un indicateur financier. **Son fonctionnement repose sur l'apprentissage statistique.**

1.2Apprentissage statistique

Les fonctions principales de l'apprentissage statistique sont :

- Apprendre des données.
- Analyser une grande quantité de données.
- Créer des dépendances entre structures à partir des observations.
- Déetecter les motifs de régularités au sein d'un grand groupe de données. Technique qui permet de se focaliser sur la précision de la prédiction et l'optimisation de la performance statistique.

Deux types de méthodologie:

Apprentissage supervisé : on considère les données à prédire organisées en groupe distincts représentés par un chiffre. On veut ici bâtir la relation entre les données d'observation sous formes de matrices et les groupes de données à prédire. L'objectif est de prédire la sortie y pour une nouvelle entrée x. Si Yi est binaire on parlera de classification, si Yi est dans R on parlera de régression.

Apprentissage non supervisé : on veut repartir en groupe un ensemble de données en réalisant un clustering. Il n'y a pas de sortie y le seul but est de trier les vecteurs entrés Xi.

1.3) Prisms un outil nouveau et prometteur

PRIMS s'est montré efficace :

- Hit ratios corrects (taux de réussite des prédictions), 65 pourcents de résultats corrects à 20 jours.

-L'essentiel repose sur des modèles non paramétriques ce qui permet une liberté absolue dans la réalisation des tests

-Les facteurs de marchés émergents sont capturés efficacement.

PRISMS est entièrement piloté par les données :

Chaque donnée d'entrée est analysée de manière indépendante, sa participation à la construction du modèle n'est pas fixée initialement. Le modèle est établi uniquement de façon statistique et non paramétrique.

-PRISMS est construit de façon à extraire le maximum d'informations d'un gros paquet de données

-PRISMS repose sur des concepts de statistique learning de derniers cris, qui se sont montrés efficaces dans de nombreux domaines.

Pas de biais :

Tous les processus de calcul reposent sur des données réelles, et les modèles sont construits de manière adaptive avec des algorithmes d'apprentissages. La méthode est non paramétrique, ce qui signifie que les modèles ne reposent pas sur des hypothèses de départ, de distribution statistiques et de dépendances.

Au service de la qualité de prédiction

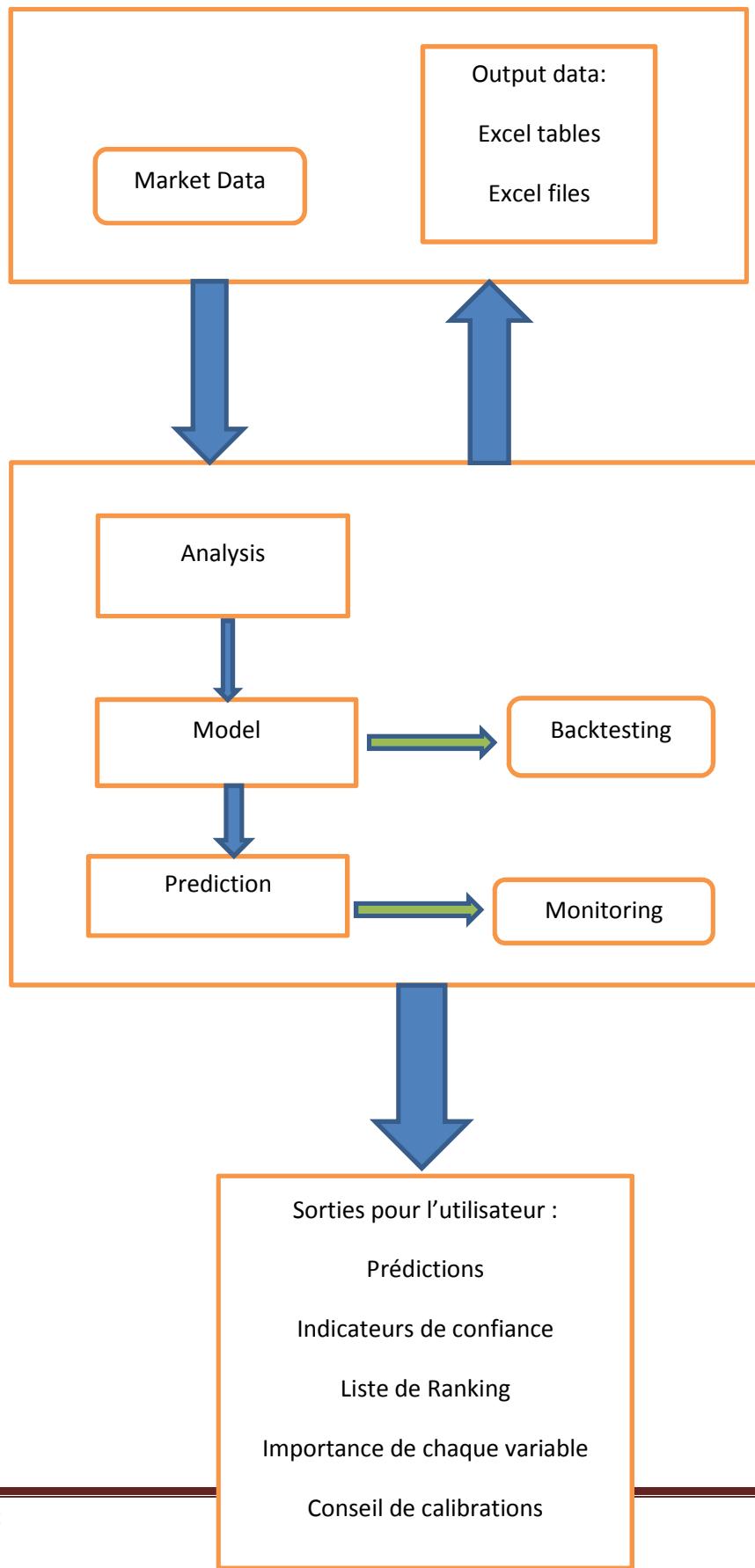
L'objectif principal est de maximiser le hit ratio. Les modèles ont pour but de donner la bonne prédiction, leur but n'est pas de dévoiler la relation entre les actifs, le résultat est la priorité absolue. La performance de l'apprentissage est directement réglable par l'utilisateur. Ce domaine de la modélisation permet de faire participer l'utilisateur.

Mises à jour permanentes :

Les valeurs financières utilisées sont mises à jours et synchronisées avec Bloomberg. C'est donc un outil dynamique qui prend en compte les dernières évolutions du marché. Il peut être utilisé quotidiennement dans un but de compétitivité.

classification des prédicteurs

Les bons prédicteurs sont les actifs dont les variations significatives amènent des changements en termes de prédictions du return du stock Target. Un prédicteur a une importance significative si son utilisation dans les données d'entrée accroît fortement le hitscore. Cette classification de prédicteurs permet à l'utilisateur d'avoir une vision large et complète des mécanismes financiers qui impactent le cours d'une valeur.



UNE METHODOLOGIE QUI REPOSE SUR L'ICA

Pourquoi ne pas utiliser le CAPM ?

La représentation du marché est discutable, les estimations comme celles des betas sont instables, le CAPM repose sur un équilibre LT qui n'est pas pris en compte pour les variations locales.

Dans le cadre du CAPM, tous les investisseurs ont un portefeuille de Markowitz car ils perçoivent toutes les actions en termes d'aspect moyenne variance.

Les investisseurs peuvent prêter ou emprunter de l'argent au taux sans risques.

$$E(R_{actif}) = R_f + \beta_{actif}[E(R_m) - R_f]$$

Avec $\beta_{actif} = \frac{cov(R_m, R_{actif})}{var(R_m)}$

L'indépendant component analysis paraît mieux adapté :

L'ICA détecte les chocs de marchés et identifie les facteurs comme portefeuille d'arbitrage.

Les facteurs systématiques sont plus robustes que ceux du CAPM. Cela consiste à chercher une transformation linéaire qui minimise la dépendance statistique entre prédicteurs. Il s'agit d'une séparation aveugle des sources, qui consiste à remplacer un espace de variables par de nouvelles variables de variance maximale, non corrélées deux à deux qui sont des combinaisons linéaires des variables d'origine. Ainsi une quantité maximale de données statistiques est extraite.

1.4 Modules et fonctionnalités disponibles par l'utilisateur :

Analyse profonde des données de marchés :

- analyse d'un gros paquet de données.

- Identifie les différents états passés du marché : indication sur la configuration actuelle du marché.

Prédictions et recommandations :

- possibilité pour un stock donné X de donner son scenario futur : une tendance respectée ou une inversion.
- détermine l'horizon qui doit être utilisé pour considérer la prédition comme fiable.
- publie des recommandations et prédictions d'un portefeuille d'actifs.

monitoring :

- produit des alertes sur les potentiels changements de marché.

- identifie les facteurs de marchés prédominants.
- permet une calibration automatique.
- en cas de changement de marché, quels sont les indicateurs à cibler ?
- détecte des changements de configurations de marchés.

Backtesting:

- test des stratégies sur le temps
- permet de statuer sur la performance d'une stratégie

Analyse du risque :

- identification des facteurs systémiques, détermine les risques et les betas.
- propose des stratégies d'Hedging et d'optimisation.

Comment Se sert on de l'apprentissage statistique dans Prisms ?

Toutes les tâches réalisées dans PRISMS peuvent être considérées comme des cas d'apprentissage statistiques.

SEGMENTATION-un problème d'apprentissage non supervisé :

Les données sous forme de signaux ont besoin d'être traitées :

- afin de donner une bonne représentation du marché,
- afin de créer des données exploitables pour les algorithmes d'apprentissage statistiques. Les Returns daily de chaque signal sont clustérés en catégories similaires.

Chaque signal est segmenté en régime de trend avec un clustering de dates en groupes homogènes.

Le processus d'identification de régime assigne à chaque période un numéro de cluster, le régime est ainsi caractérisé par un numéro.

Une indexation de ce type donne une description du marché.

PREDICTION-une tache d'apprentissage supervisé :

Cette étape clé construit les modèles depuis des données historiques avec des algorithmes d'apprentissages statistiques. Une fois construit, de nouvelles configurations de marché sont ensuite entrées dans les modèles afin de prédire les valeurs futures, avec des indicateurs de confiance.

On classe les types de prédictions de façon binaire (up=1, down=-1). Il y a cependant une possibilité de faire 4 classes (strong up/up/down/strong down).

A partir de toutes les informations sur le marché à l'instant t, on cherche à classifier le return futur que l'on souhaite prédire dans ces catégories. **Ainsi en se basant sur les observations et la relation entre le marché et le return futur, un modèle de classification est créé.**

Présentation des algorithmes utilisés pour la segmentation

Prédire le comportement d'un portefeuille à partir du marché nécessite sa décomposition en facteurs prédominants.

Ces facteurs doivent être indépendants, comme la valeur du portefeuille est la combinaison de plusieurs signaux.

Algorithme Dyadic Coifman Wickerhauser :

- a)initialisation: on considère toutes les dates séparées
- b) principe : fusion de dates consécutives si cela fait baisser le critère de Gini
- c)itération : on réitère cette opération jusqu'à ce qu'on arrive plus à faire baisser le critère

Le **coefficent de Gini** est une mesure statistique de la dispersion d'une distribution dans une population donnée, c'est le critère utilisé ici. Le critère d'entropie peut aussi être utilisé.

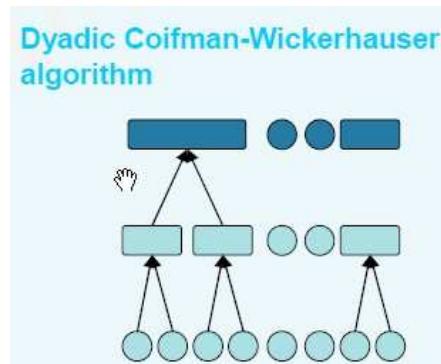


Figure 1 au fil du temps les périodes fusionnent en fonction du critère de Gini

Unsupervised CART (classification and regression Trees ,top-down):

- a)initialisation: on considère tout l'historique comme une seule période.
- b) principe : trouver la meilleure division afin de faire diminuer le critère.
- c) on réitère le processus précédent jusqu'à ce que le critère ne puissent plus descendre.

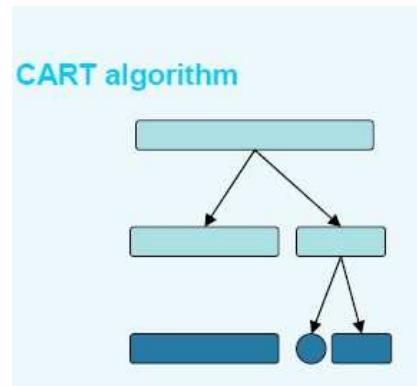


Figure 2: procédé inverse du dcw

IDENTIFICATION DES REGIMES DE MARCHE

Utilisation de l'algorithme Kmeans :

On cherche à minimiser la distance intra distances. On détermine les centres de chaque cluster et les composants de ce cluster associé.

Algorithme utilisé pour l'apprentissage statistique

Classification avec la random Forest BASE CLASSIFIER : decision trees (CART)

Objectif : trouver la partition optimale de l'espace d'entrée (prédicteurs) définie par un arbre

Critère à faire croître : critère de l'impureté, connu comme le critère de GINI, qui rend compte de la diversité d'un groupe.

MONTE CARLO ET OPTIMISATION

On utilise un grand nombre d'échantillon. A chaque la meilleure variables permettant de construire une nouvelle feuille est choisi à partir d'un grand ensemble de variables.

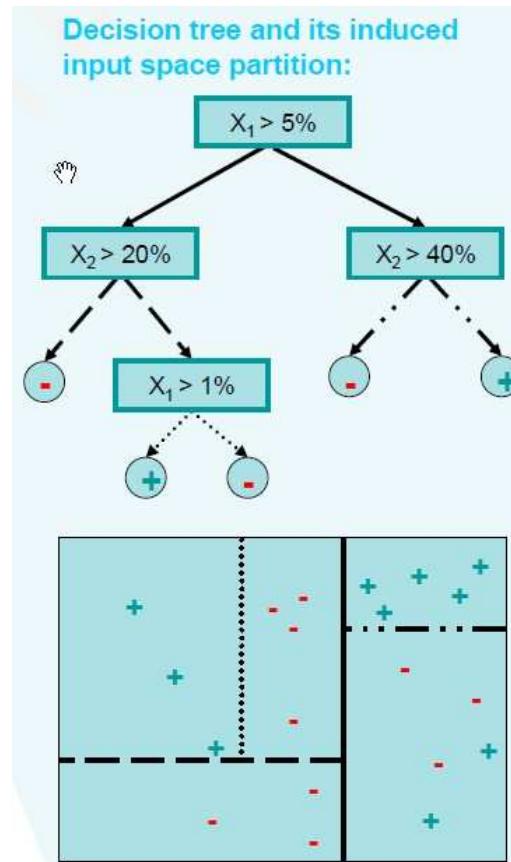


Figure 3: partitionnement de l'espace des prédicteurs, ici un espace deux dimensions les zones délimitées correspondent à une tendance future de la valeur à prédire. On peut adapter ce type de figure en fonctions de la dimension de l'espace des prédicteurs.

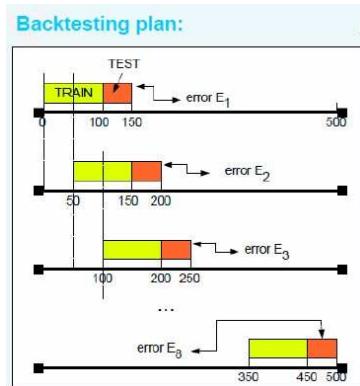
DETAILS DU BACKTESTING

Le modèle est testé sur la passé, on construit le modèle sur une période puis on le teste sur la période suivante.

Chaque modèle sélectionne les variables importantes. L'évolution du ranking des variables traduit l'évolution du contexte économique.

Les paramètres sont calibrés pour maximiser la performance attendue.

Chaque prédiction est assimilée en permanence à un intervalle de confiance, lorsque l'on réalise une prédiction. On détermine un vecteur similaire de prédicteurs et lui associe son intervalle de confiance. Tout cela permet de statuer sur la qualité du modèle créé.



Analyse du marché

- filtrer les signaux en identifiant les régimes homogènes
- cluster des actifs

Prediction

- prédire le signe de la variation, trend, reversal
- identifier les configurations de marché
- prediction des actifs

monitoring et backtesting

- classer les actifs et les prédicteurs
- backtest de modèles
- calculer les taux d'erreur

EXEMPLE CONCRET SIMPLISTE :

On considère que l'ensemble du marché est caractérisé par seulement deux variables. Le prix du Brent et l'Euro-Dollar. Le stock que l'on souhaite prédire est Total.

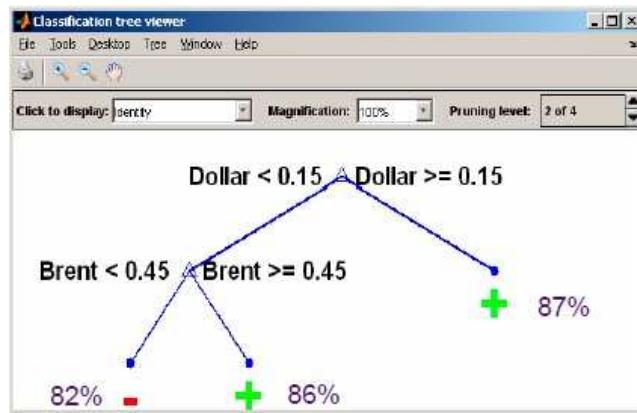
L'arbre suivant produit une « carte » du marché.

Quand le dollar est up ($>.15\%$), total est up à une fréquence de 87%.

Quand le dollar est down, il y a deux cas :

Soit le Brent est down ($<.45\%$) : alors Total est down avec une fréquence de 82%

Soit le Brent est up ($>.45\%$) : alors Total est up avec une fréquence de 86%



1.5 Audit technique de la version actuel du logiciel

Cette étape consiste à analyser les défauts du logiciel initial et proposer des solutions et innovations afin de le faire gagner en rapidité et en performance.

Le logiciel possédait 3 problèmes majeurs :

- La vitesse de calcul,

- la précision des résultats,

- le manque de méthode qui permet de mesurer l'intervalle de confiance de chaque prédiction.

Interface graphique défectueuse

Couche graphique GUI Matlab est défectueuse par endroit, (disfonctionnement visible, bouton non présent). Il faut impérativement limiter le choix des paramètres à l'utilisateur qui peuvent introduire des contre sens. L'Interface était surchargée et pas assez claire. Le GUI (graphique utilisateur interface) doit être adapté aux prochaines évolutions du code.

Trop de paramètres a régler pour l'utilisateur

L'objectif est d'obtenir une interface où les paramètres d'apprentissage et de partition se fixent automatiquement en s'adaptant en fonction des prédicteurs choisis par l'utilisateur. Il faut idéalement que l'utilisateur puisse s'en servir sans se questionner sur la signification de chaque paramètre. Le logiciel PRISMS est un prototype résultant du travail de plusieurs personnes. Cela doit être plus performant en termes de vitesse de calcul, stockage et il faut améliorer la relation entre les classes et les fonctions.

Le nombre de variables dans le Workspace et le type de données ne sont pas optimisés. Les échanges entre les différents types de mémoire de l'ordinateur ont besoin d'être limité. Cela ralentit le processus.

une meilleure utilisation de Matlab :

L'analyse du CPU nous montre qu'il y a de nombreuses pertes de temps quand Prisms tourne. Une analyse pointue profiler sous différentes conditions a été réalisée. Toutes les combinaisons d'algorithmes furent testées. Il y a dans la version actuelle des lignes ou le CPU passe trop de temps à effectuer les calculs. On a souvent des portions de script où 95 pourcents du temps est passé sur 5 pourcents des lignes.

Un des objectifs est de convertir certains objets Matlab comme les Cell ou les matrices. Matlab est réputé pour traiter les matrices avec rapidité. Il faut se servir des atouts de Matlab sans cela il n'y a aucune utilité à se servir de Matlab, étant donné que toutes les données sont numériques.

Matlab est efficace pour convertir concaténer, déplacer et réaliser des opérations entre matrices (matrix laboratory). Les objets de type structure sont lents à manipuler et mélangeant plusieurs types de variables.

Vectorisation des fonctions :

Il est nécessaire d'impérativement de vectoriser les fonctions (en évitant la superposition de boucle). Cela permet de gagner en rapidité. Le logiciel effectue plusieurs opérations à la fois dans le cas d'une bonne vectorisation. IL faut convertir des programmes scalaires en programme vectorisés. La méthodologie est la suivante:

- trouver une boucle interne qui peut être vectorisée.
- transformer les boucles et générer des codes vectoriels.

Il faut pour cela s'occuper des modules secondaires du code pour ensuite remonter dans les scripts principaux.

Le langage C un langage au service de la rapidité

Certaines parties du code doivent être convertir en C. En effet le langage C, avec sa gestion de la mémoire vive à l'aide de pointeurs, est un outil qui permet grandement de gagner en vitesse pour tout ce qui est du domaine de l'échantillonnage et des simulations de Monte-Carlo. La vitesse est un facteur clé du cahier des charges, le logiciel étant destiné à long termes à être utilisé à l'aide d'un serveur pour une clientèle plus importante

algorithme de partition

Les activités de partitionnement doivent être rattachées à celle de l'apprentissage statistique. Il n'est pas normal que le choix d'algorithme de partitionnement soit proposé à l'utilisateur qui n'est pas forcément concerné. Cela doit être réglé en fonction du type de procédure d'apprentissage envisagé par l'utilisateur. Le temps pris par les algorithmes de partition peut être remis en cause, étant donné son rôle secondaire dans l'apprentissage.

L'algorithme dcw tournait beaucoup plus rapidement que le cart. Construire un modèle de 80 prédicteurs en activant des options de learning nécessitant une partition préalable prenait plus de 20 minutes. Toutes les options d'apprentissage ne nécessitent pas forcément une segmentation.

Un code peut être trop poussé et pas forcément approprié

La programmation de la première version du logiciel a été réalisée par un thésard extrêmement compétent et travailleur. Il s'agit d'une programmation de haut niveau de 30 000 lignes. Cependant 10 000 lignes correspondent à l'interface graphique ce qui est moins intéressant au point de vue de la programmation.

Cette interface a cependant un rôle essentiel, le logiciel devant être commercialisé, l'utilisateur associe essentiellement le logiciel à son aspect graphique. La programmation était un peu surchargée par rapport à ce que réalisait réellement le logiciel. Elle présentait de nombreuses options qui éloignaient le projet de son objectif final.

La programmation objet n'est pas justifiée sur chaque partie du code. Les extractions de données et les formats des fichiers de sauvegarde d'instances de classe peuvent être modifiés. Les extractions et les échanges de mémoire sont trop nombreux au cours d'une utilisation linéaire. Il faut que le logiciel puisse tourner sur des machines pas forcément ultra performantes.

C'est un projet expérimental qui se traduit dans sa programmation, de nombreuses options sont superflues et pas forcément dans l'intérêt de faire augmenter le hit score global des prédictions. Il faut trouver le juste milieu entre la multiplication des options proposées à l'utilisateur et la performance en termes de hit score.

Un hitscore qui se doit d'être augmenté :

Initialement la performance du logiciel était de 58 pourcents de prédictions correctes à 25 jours (en résultat de type trend + ou -). Cette performance est plutôt bonne. Mais nous verrons plus tard qu'un hit score globale du logiciel permet d'augmenter le alpha d'une potentielle gestion de fond entièrement basée sur l'utilisation de Prisms.

Les deux pistes à étudier sont :

- les algorithmes, leur efficacité et leur adaptation aux données.

- le nombre et la nature des entrées de l'algorithme d'apprentissage.

Un nombre de prédicteurs qui doit être augmenté :

Pour faire augmenter le hit score, il faut trouver de nouveaux types de prédicteurs qui permettent d'apporter de l'information statistique. Initialement le logiciel exploitait trop peu de types de données pour chaque titre (principalement volatilité et return). Il faut accroître le catalogue des valeurs. L'insertion de données macroscopiques dans la base de données a été évoquée, cependant certaines ne sont pas journalières. Il faut donc au préalable traiter ces séries afin de les rendre quotidiennes. Un véritable travail de fouille de données est nécessaire, le hitscore dépend énormément de la qualité des prédicteurs et de leur choix judicieux.

2)taches réalisées.

Ma démarche de programmation fut la suivante, cela correspond à une programmation modulaire qui consiste à créer le maximum de fonctions pour clustérer au maximum le code, ce qui est très utile pour trouver les erreurs et résoudre les problèmes .Dans un second temps il faut ensuite créer une programmation de type objet afin de fluidifier le tout et simplifier pour s'y retrouver afin d'éviter la succession de plusieurs fonctions d'affilées.

```

def importeurostock4():
def affichernormalprix(prix0,sigma,hitscore):
def basel00(a):
def ploteurostock():
def afficherquartile():
def volhistnormal(listed):
def tradingabsolute(k,g):
def backtestingeuros_bck(deb,end,g):
def tradingabsolutechoisen(k,g,liste):
def backtestingliste(a,b,g,liste):
def hedgefund(k,g,long,short):
def hedgefundliste(k,g,long,short,liste):
def hedgefundlistepoid(k,g,liste,poid):
def backteststhedgefundlistepoid(a,b,g,liste,poid):
def volunivlistepoid(k,g,liste,poid):
def backtestvolunivlistepoid(a,b,g,liste,poid):
def affichagesharp():
def wait():
def afficherperiodtrading(deb,g):
def distribuniv(k,g):
def distribinvest(k,g):
def distributiongainrandom(k,g):
def gainliste(k,g,liste):
def gainlistepoid(k,g,liste,poid):
def backtestgainlistepoid(a,b,g,liste,poid):
def distributiongainrandomlong(k,g,long):
def calculvolitem(k,g):
def h():
def signalrandom():
def signalrandomyear(vol):
def volatility(liste):
def buldsignal(listemere,a,pas):
def ordernarkane(vol,corri):

```

L'IDE utilisé fut Pycharm, un IDE de développement assez performant. L'intérêt de python est de pouvoir à tout moment réaliser des tests sur la console en parallèle à la réalisation des scripts, ce qui n'est pas possible en C++ et C ou il faut faire tourner le script directement.

Cependant on peut reprocher à python d'être un langage haut niveau qui ne permet pas de gérer la mémoire de façon à gagner en vitesse. Son atout principal est la package scikit learn, qui possèdent tous les algorithmes de machine learning. Les packages conda (pandas, scipy ...) n'ont rien à envier à Matlab. L'intérêt de python est de pouvoir interférer avec Matlab en permanence. Les compilateurs utilisés par python sont un peu plus rapides que Matlab.

L'idéal aurait été de réaliser tout le logiciel en C pour gagner en vitesse, mais cela nécessite la création de beaucoup plus de fonctions et d'objets puisque celui -ci n'est pas programmé objet, ce qui peut être délicat. Ce travail aurait été bénéfique qu'en cas d'une parfaite maîtrise de la manipulation des pointeurs.

Je pense que le C++ était le compromis idéal il est assez rapide, programmé objet et comprend dans sa création déjà beaucoup de classe disponible. Le cahier des charges mettait l'accent sur la vitesse et le niveau de hit score. Il a fallu trouver le juste milieu entre les deux, ne pas sacrifier la vitesse pour la précision et l'inverse. R était une bonne solution avec sa capacité à gérer mieux les grosses quantités de données que Matlab.

Initialement, le cœur des algorithmes reposait sur la Toolbox machine learning de matlab (class reg tree). Pour comprendre et manipuler le code déjà implémenté à mon arrivé, je faisais tourner le script par petites portions en inspectant les variables Matlab stockées dans la mémoire vive. Il faut comprendre et déterminer les variables d'entrée et de sortie de chaque portion de script

nanmean	7625860	250.127 s	250.127 s	█
ismember>ismemberR2012a	6231254	572.303 s	228.857 s	█
parseArgs>create@(z)any(strncmpi(s,z,n))	7516824	209.700 s	209.700 s	█
prediction_stability	1032	3703.429 s	203.573 s	█
ClassLabel>ClassLabel.length	11098688	202.573 s	202.573 s	█
ClassLabel>ClassLabel.numel	12352144	198.506 s	198.506 s	█
TreeImpl>TreeImpl.findNode	2305628	308.459 s	188.050 s	█
...odel>ClassificationModel.get.Prior	2409860	878.761 s	161.939 s	█
glmfit>wfit	1606680	158.744 s	158.744 s	█

Figure 4: exemple d'étude permettant de visualiser les parties lentes du code et les portions à améliorer

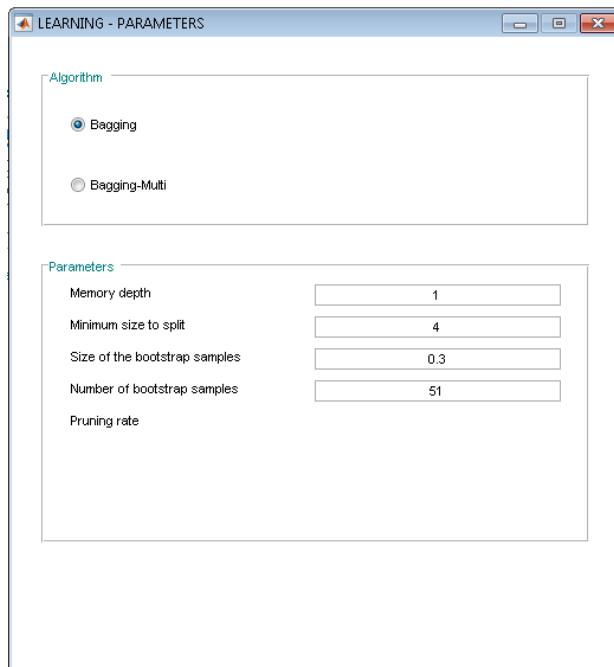


Figure 5: exemple de problème dans l'interface graphique ici manque de bouton

2.1). Démonstration de l'intérêt de Prisms dans une gestion de portefeuille

L'essentiel de la démarche est de savoir comment à partir des résultats du logiciel, on peut générer de l'alpha pour la gestion de fond en bâtissant un return stable annuel en se fiant entièrement aux prédictions du logiciel sans écouter l'intuition.

On part du principe que le logiciel donne un hit score moyen de véracité de prédiction d'un taux supérieur à 50 pourcents. Le résultat donne deux options plus ou moins. On considère que l'on souhaite étudier le résultat sur une centaine de valeurs: on les divise en deux lot le lot des titres que l'on est susceptible de shorter et le deuxième lot les titres que l'on est susceptible de longer.

On se situe à l'instant t où on fait tourner le logiciel on a donc deux options sur l'évolution du prix du titre à l'instant $t+25$. Le résultat nous est indiqué sous forme de plus ou de moins, en cas de plus cela correspond naturellement à une situation de long, en cas de moins cela correspond naturellement à une situation de short.

L'alpha sur un titre est assimilé à la plus-value réalisée après l'opération de trading choisie entre $t+25$ et t . Selon la probabilité de réussite du logiciel on est sûr de gagner de l'argent dans 58 pourcents des cas. On considère le prix à l'instant t d'un titre l'alpha correspond donc à l'écart entre la moyenne de la densité de prix à l'instant $t+25$ et le prix à l'instant t .

Dans le cas d'un long, on sait pour chaque titre qu'en moyenne on a une probabilité que le prix soit de 58 pourcents plus haut. On s'intéresse à la distribution statistique du prix à l'instant $t+25$. Celle-ci doit refléter le fait que dans 58 pourcent des cas futurs selon le logiciel, le prix($t+25$) est supérieur au prix à l'instant t .

58 pourcents est donc le ratio entre l'aire située après le prix de l'instant zéro et l'aire totale de la densité normale du prix à l'instant $t + 25$.

On se situe dans la même situation dans le cas où on short.

Cela correspond à la mise en équation suivante :

$$\frac{\int_{-\infty}^{P_0} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1(x-Pt)^2}{2*\sigma_{25days}^2}}}{\int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1(x-Pt)^2}{2*\sigma_{25days}^2}}} = \text{hitscore}$$

$$1/2 * \frac{P_0 - Pt}{N * \sigma_{25days}} * \frac{1 - e^{-\frac{(P_0 - Pt)^2}{N * \sigma_{25days}^2}}}{1 - e^{-\frac{(P_0 - Pt)^2}{N * \sigma_{25days}^2}}} = \text{hitscore-1/2}$$

$$\frac{1}{2} * \sum_{i=0}^N \frac{P_0 - Pt}{N * \sigma_{25days}} * e^{-\frac{i(P_0 - Pt)^2}{N * \sigma_{25days}^2}}$$

$$\frac{1}{2} * \int_0^{\frac{P_0 - Pt}{\sigma_{25days}}} e^{-t^2} dt = \text{hit score} - \frac{1}{2}$$

$$(1 + \operatorname{erf}\left(\frac{P_0 - Pt}{\sigma_{25days}}\right)) = \text{hitscore}$$

$$Pt = f(\sigma_{25 days}, \text{hitscore})$$

L'important est de déterminer Pt pour vérifier les conditions imposées par la représentation géométrique du hit score.

Dans le cas du short

Le hit ratio est le ratio entre l'aire située avant le prix à l'instant zéro et l'aire totale. Nous sommes en effet dans le cas où le prix est dans 58 pourcents des cas plus bas.

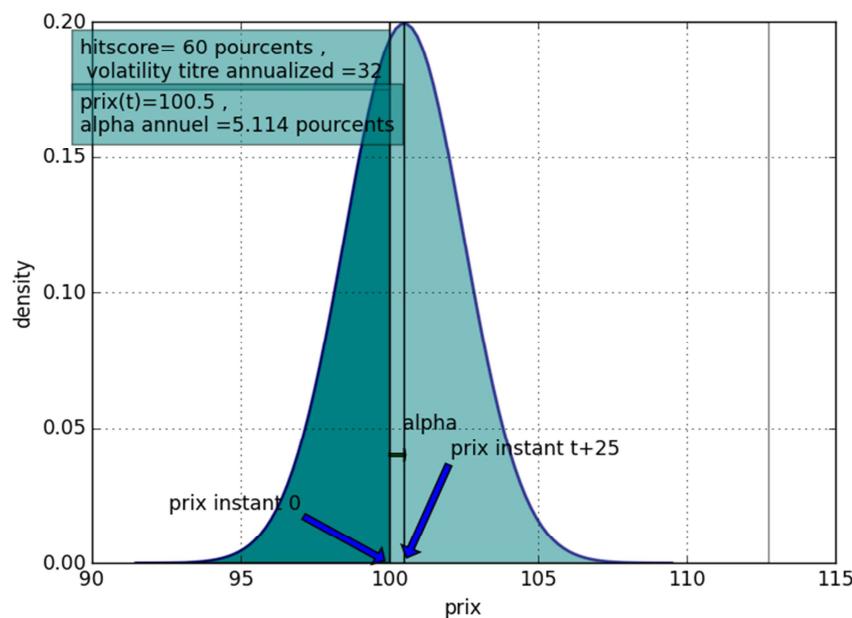


Figure 6:cas du titre long la volatilité journalière est de deux pourcents

L'alpha est conséquent pour un hitscore de 60 pourcents

Le passage de l'alpha 25 jours à l'alpha annuel se réalise de manière classique avec cette formule sur les returns:

$$\alpha_{annuel} = \alpha_{25days}^{\frac{256}{25}}$$

Le passage de la volatilité journalière à la volatilité annuelle se réalise de cette manière.

$$\sigma_{port market annuel} = \sigma_{portmarket25days} * \sqrt{\frac{256}{25}}$$

$$sharpanuel = \frac{\alpha_{annuel}}{\sigma_{port market annuel}}$$

Il faut donc déterminer le prix dans 25 jours qui permettra à la densité normale de répondre à ces conditions statistiques. Cette loi normale a pour moyenne le prix futur et la variance est la volatilité journalière du titre calculée sur un intervalle de temps précédent. Plus la volatilité est faible, plus l'alpha est petit.

On comprend que la valeur de la volatilité est fondamentale car plus elle est grande plus la distribution est large et moins centrée vers sa moyenne, ce qui induit un décalage plus important du prix ($t+25$) pour satisfaire les conditions statistiques. En moyenne la volatilité journalière est de 2 (32 % annuel) pourcents, **ce qui correspond dans des bonnes conditions de hit score à un alpha de 3 pourcents. Plus la volatilité est importante plus la normale est évasée.**

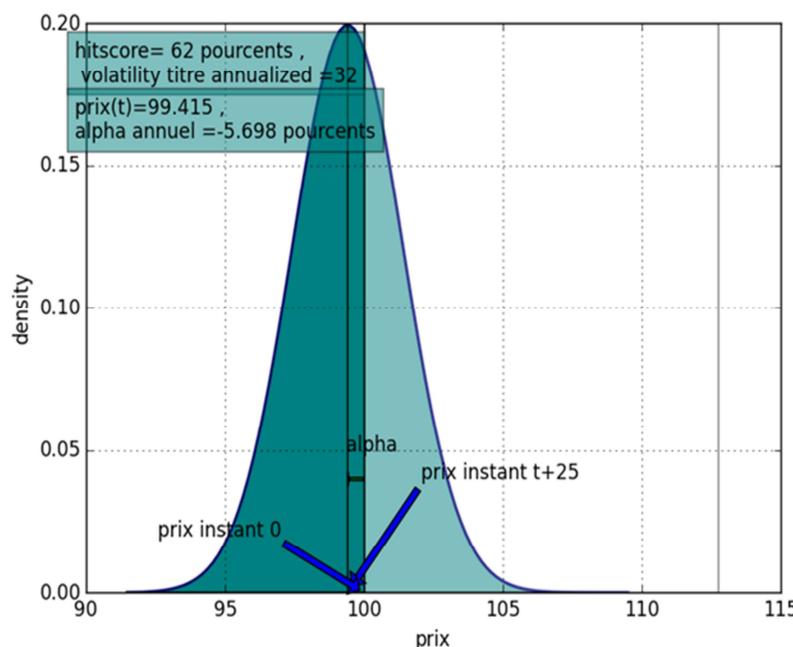


Figure 7 cas du titre short on voit bien la symétrie avec le cas long

Le cas du long et du short sont ici **réunis puisque l'on considère que la distribution de prix dans les deux cas est symétrique, et la volatilité se comporte de façon symétrique.**

L'alpha est ensuite calculé en prenant une base 100 à l'instant t et assimilé à la différence entre ce prix futur et la valeur du prix présent.

On trouve l'alpha à 25 jours de la façon suivante :

$$\alpha_{daily25days} = \frac{P_t - P_0}{P_0}$$

Nous étions précédemment à l'échelle du titre pour illustrer la notion d'alpha. Nous allons maintenant passer à l'échelle de portefeuille, ce qui est plus réaliste pour des investisseurs décident

d'utiliser le logiciel. On se sert maintenant de la volatilité du portefeuille composé de titres dont le comportement a été prédit par Prisms. On dispose donc d'une prédiction à 25 jours pour chacun des titres qui le compose. Les cas de short et de long ne sont pas distingués puisqu'on a vu précédemment que l'alpha était généré de manière symétrique dans le cas d'un long et d'un short.

A l'échelle d'un portefeuille on s'intéresse à **deux objets fondamentaux** pour calculer ses notions.

- a) La notion de **portefeuille univers d'investissement** composé de titres longs uniquement
- b) La notion de **portefeuille d'investissement** composé de titres longs et de titres shortés uniquement.

La volatilité moyenne annuelle de ce genre de portefeuille est de 20 pourcents. Il faut avoir en tête que selon la formule de la volatilité d'un portefeuille dans le cas d'une equipondération, celui-ci est égale à 2/3 de la volatilité moyenne de ces composants (soit 25 pourcents annuel pour un élément de l'euro stock,).

$$volport = \sum_1^{20} \sum_1^{20} \frac{1}{20} * \frac{1}{20} * coor(i,j) * vol(i) * vol(j)$$

La valeur moyenne de $coor(i,j)$ est de .63 pour des index on comprend en effet que cette volatilité va faire 2/3 de la volatilité moyenne de chaque élément inclu dans le portefeuille. La volatilité moyenne d'un élément type Eurostock 50 est de 25 pourcents. On comprend ainsi la valeur de 20 pourcents annoncée précédemment.

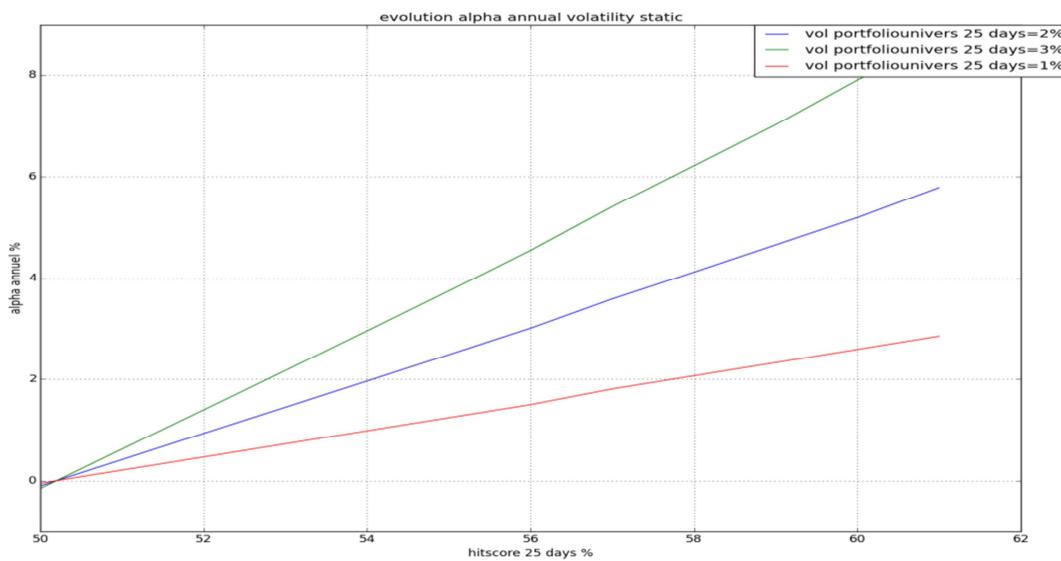


Figure 8: évolution du alpha avec trois volatilités journalières fixes en fonction du hit score on voit bien la croissance

De celui ci

Il faut ensuite le comparer à la volatilité du portefeuille d'investissement. Sa volatilité est d'une dizaine de pourcents. Le portefeuille d'investissement est lui un portefeuille ou les ordres de bourse qui sont passés à partir des titres qui le composent sont orientés vers une stratégie commune dont le but est d'augmenter le return et de baisser la volatilité.

$$\sum_{i=0, i \text{ long}}^{20} \sum_{j=0, j \text{ long}}^{20} \frac{1}{l^2} * cov(i, j) * vol(i) * vol(j) + \sum_{i=i \text{ short}}^{20} \sum_{j=j \text{ long}}^{20} \frac{1}{l*s_0} * cov(i, j) * vol(i) * vol(j) \\ + \sum_{i=0, i \text{ short}}^{20} \sum_{j=0, j \text{ short}}^{20} \frac{1}{s^2} * cov(i, j) * vol(i) * vol(j)$$

I est le poids des titres longs est le poids des titres short.

On considère ici que chaque titres long et short ont la même pondération entre eux.
Etant donné que cov(i,j) est en moyenne de .63 et que la pondération des short correspond à la valeur négative de son poids on comprend par simplification que l'on aura :

$$V_{\text{port}} = 1/3 * 2/3 * Vol - 1/3 * 2/3 * Vol + 1/3 * 2/3 * Vol = 1/3 * VOL$$

Le ratio de Sharp est le rapport entre l'alpha et la volatilité du portefeuille d'investissement. Celui-ci est l'indicateur de performance de la gestion de fond.

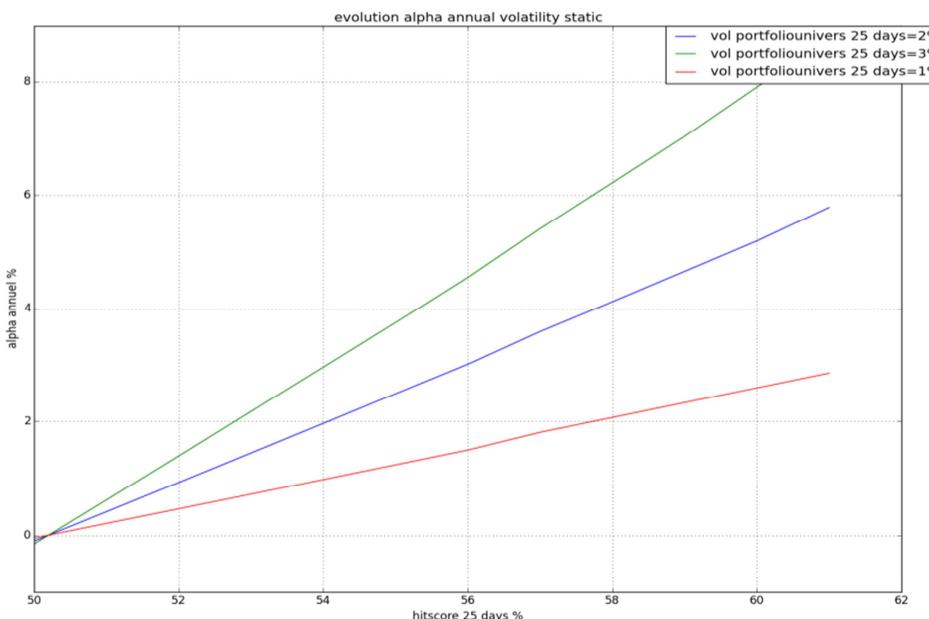


Figure 9: évolution de l'alpha généré en fonction du hitscore de Prisms pour trois valeurs de volatilités journalières choisies

Une des parties de mes activités fut de démontrer la possibilité de l'obtention de l'alpha sur des périodes de test, à partir du logiciel .Il s'agit d'une démarche commerciale visant à illustrer les bienfaits de l'utilisation de ce logiciel pour la gestion de portefeuille global, et surtout la possibilité de se servir de Prisms dans une gestion quotidienne. Cette étape est cruciale pour démontrer que celui-ci n'est pas un gadget mais un véritable outil pour créer de la valeur.

Une des inconnues est donc de calculer la volatilité du portefeuille univers .Plus celle-ci est importante plus l'alpha est élevé. L'alpha varie en effet linéairement avec la volatilité.

Cependant on comprend bien qu'une forte volatilité du portefeuille entraîne pour un utilisateur non confirmé une augmentation du portefeuille d'investissement ce qui est contraire à l'augmentation du ratio de Sharp. La clé est de faire baisser la volatilité de son portefeuille d'investissement tout en gardant une forte volatilité en portefeuille univers.

La démarche a donc été la suivante :

A partir de l'euro stock 50 on crée un portefeuille de 20 titres sélectionnés aléatoirement. On réalise cette opération un nombre suffisant pour représenter le comportement du portefeuille moyen de l'euro stock. On se fixe à un période de départ en jour sur l'année sur l'année précédente.

On calcule la volatilité journalière du portefeuille investissement univers (portefeuille comportant uniquement des longs), sur les 25 jours précédents. Il faut utiliser lors de ce calcul classiquement la volatilité de chaque élément annualisée calculée sur 25 jours précédents.

Il ne faut pas oublier d'annualiser pour des questions d'homogénéité des volatilités. Il faut se fixer une convention et s'y fier pendant toute la durée du calcul. Un autre élément en jeu est la matrice de

corrélation entre chaque élément du portefeuille calculée sur un an précédent la date où l'on se trouve. On obtient après avoir déterminé ces différents éléments la volatilité annualisée du portefeuille univers calculée sur 25 jours.

En fonction de cette valeur de volatilité, pour un hit score donné on calcule directement la valeur du alpha en pourcent (valeur comprise entre 0 et 5 pourcent d'après mes calculs). L'obtention du alpha consiste à mettre en place un algorithme qui trouve le centre idéal de la gaussienne pour obtenir les conditions sur les surfaces d'air évoquées précédemment. C'est une sorte de recherche dichotomique.

```
def affichernormalprix(prix0,sigma,hitscore):
    i=0
    while i <5:
        a=norm(prix0+i,sigma).cdf(prix0)*100
        if np.abs((100-a)-hitscore)<.25:
            break
        else:
            i=i+0.005
    alphaannuel=pow(1+i/prix0,10.25)-1
    alphaannuel=alphaannuel*100
    return (prix0+i-prix0)/prix0*100,prix0+i,alphaannuel
```

On a donc accès à une plage de valeur d'alpha en fonction des valeurs du hit score.

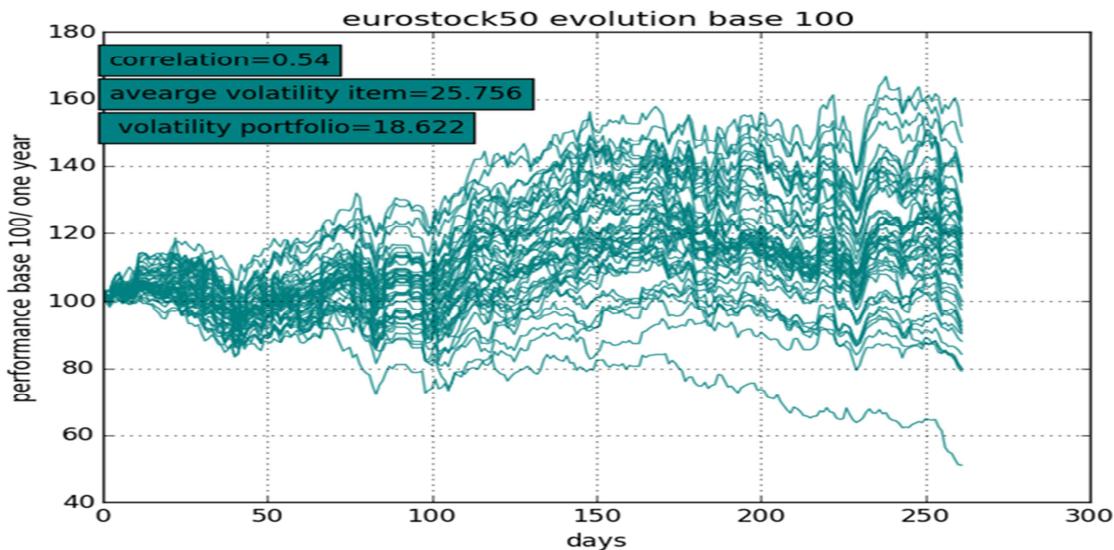


Figure 10 ensemble des titres de l'eurostock sur un an soit 256 jours, les valeurs ont été mises en base 100 au premier jour

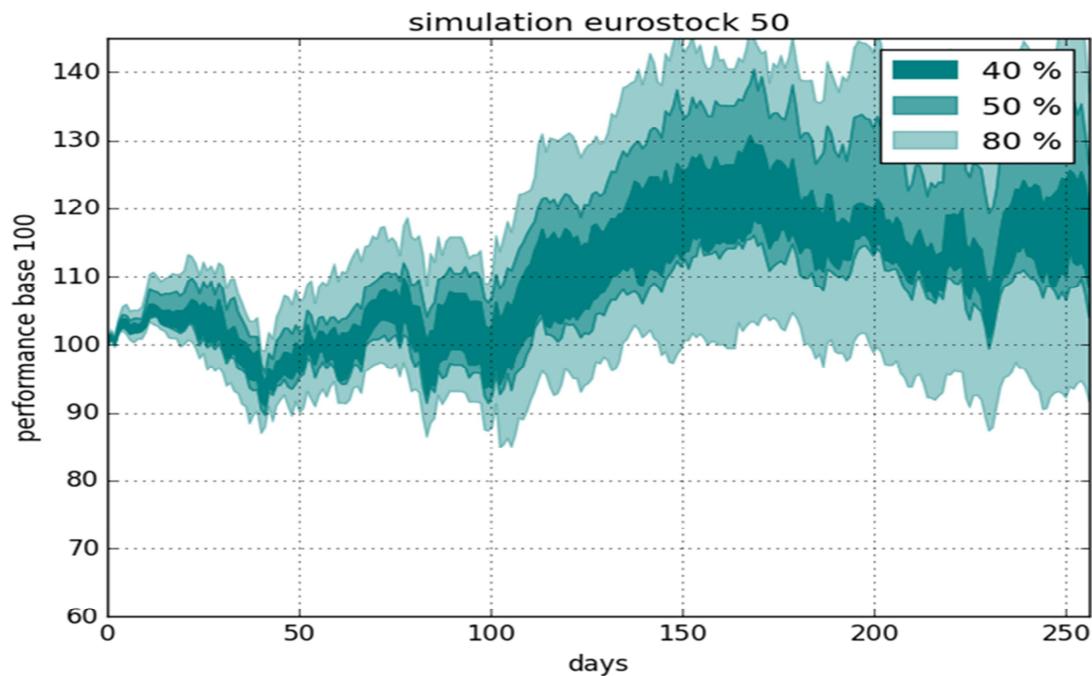


Figure 11 : densité des valeurs de l'eurostock

Le deuxième élément clef à prendre en compte est le portefeuille investissement. On va calculer celui –ci de manière aléatoire en le composant de 10 shorts tirés au hasard et de 10 longs. On tient a ici créer **des portefeuilles d'investissement pas forcément étudiés** pour baisser en volatilité (celui va en effet en moyenne se retrouver vers 11 pourcents, ce qui n'est pas très bien, les hedges funds arrivent à avoir des volatilités de 6 pourcents).

```

def tradingabsolute(k,g):
    mat,liste=np.corrcoef(g,[])
    for i in range(50):
        a=g[i][225-k:250-k]
        liste.append(a)
    liste,invest,univers,sharp=base100(liste),[],[],[]
    liste2=[]
    for i in range(50):
        liste2.append(volhistnormal(liste[i]))
    for i in range(1000):
        listel,som,voluniv,volinvest=list(np.random.choice(50,20,replace=False)),0,0,0
        somme=0
        for p in listel:
            for l in listel:
                somme=somme+1
                r=liste2[p]
                s=liste2[l]
                if listel.index(p)<10 and listel.index(l)>=10 :
                    volinvest=-.05*.05*r*s*mat[p,l]+volinvest
                if listel.index(p)>=10 and listel.index(l)<10:
                    volinvest=-.05*.05*r*s*mat[p,l]+volinvest
                else:
                    volinvest=.05*.05*r*s*mat[p,l]+volinvest
                    voluntiv=.05*.05*r*s*mat[p,l]+volumiv
        univers.append(np.sqrt(voluniv))
        invest.append(np.sqrt(volinvest))

```

Figure 12: calcul des deux types de volatilité

Cette sélection aléatoire des titres du portefeuille d'investissement doit être réalisée de nombreuses fois afin de rendre compte du comportement moyen d'un portefeuille d'investissement géré de façon aléatoire. Il faut réaliser assez de tirage pour que la moyenne actualisée des volatilités de ces portefeuilles soit stable.

Une fois le portefeuille composé il faut calculer sa volatilité. J'ai considéré dans un premier temps que chacun des titres du portefeuille est de même poids, ainsi le poids total des titres shortés est le même que celui des titres long.

On calcule la volatilité de la même manière que précédemment cependant ici les titres shortés sont associés à un poids négatif. On utilise aussi la volatilité de chaque élément et la matrice de corrélation. Il faut savoir que cette volatilité est de 11 pourcents ce qui correspond à un tiers de la volatilité des titres annuelle. On comprend cela comme les poids sont les même au point de vue de la formule. On a ici calculé la volatilité du portefeuille d'investissement.

On arrive à une étape où on dispose de l'alpha et de la volatilité du portefeuille d'investissement. On peut donc calculer la volatilité du fameux ratio de Sharp, l'indicateur global de l'efficacité de la stratégie. Notre Sharp sera donc en moyenne de 0.4. Cela n'est pas bien au premier abord. Cela est dû au fait que le portefeuille d'investissement n'est pas du tout optimisé. Il n'y a pas de répartition de poids adéquates, le choix des titres à long et short n'est pas réalisé. On peut donc aisément avec des techniques classiques faire baisser celui-ci.

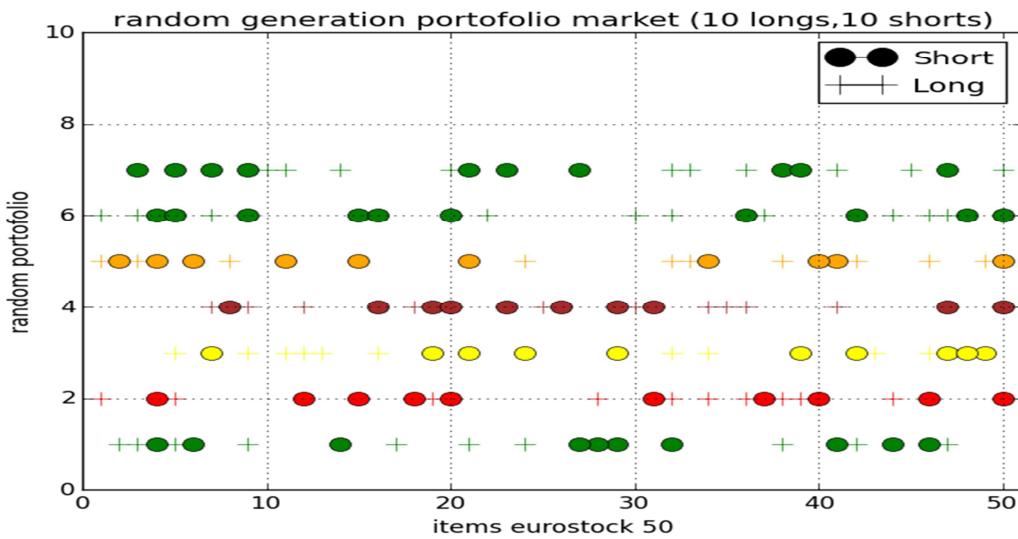


Figure 13: illustration de la génération aléatoire de portefeuille sur les 50 titres présents au départ 10 long et 10 short sont sélectionnés. Il faut utiliser les simulations de Montecarlo un nombre de fois nécessaire pour atteindre le comportement moyen du portefeuille moyen

J'ai étudié une autre valeur. Pour chaque portefeuille d'investissement créé il m'a semblé bon d'étudier le gain réalisé par celui-ci au bout de 25 jours en réalisant les opérations prévues au bout de 25 jours. Sur l'ensemble des portefeuilles simulés le gain est évidemment centré en zéro. La variance moyenne est de un euro pour un portefeuille total de 100 euros.

Nous nous sommes situés ici à une période précise.

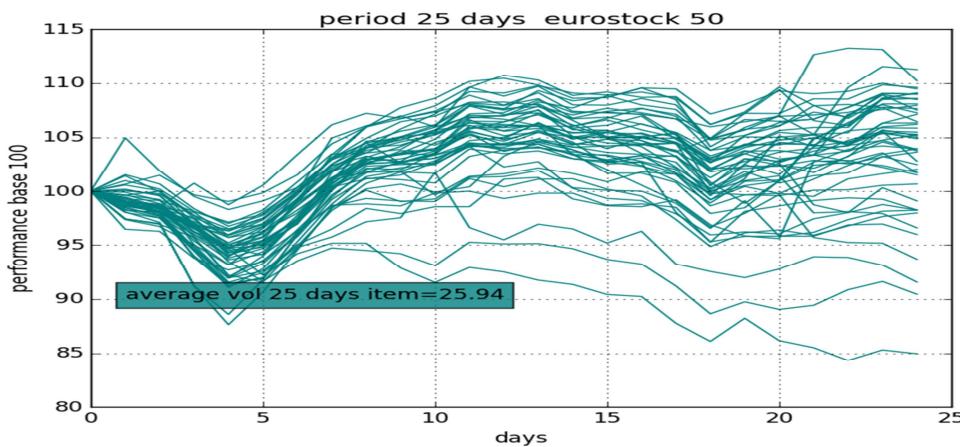


Figure 14 période isolée de 25 jours mise en base 100 afin de simplifier les calculs et la représentation de plusieurs séries en même temps

Ainsi j'ai calculé :

-la distribution des volatilités univers des portefeuilles aléatoires

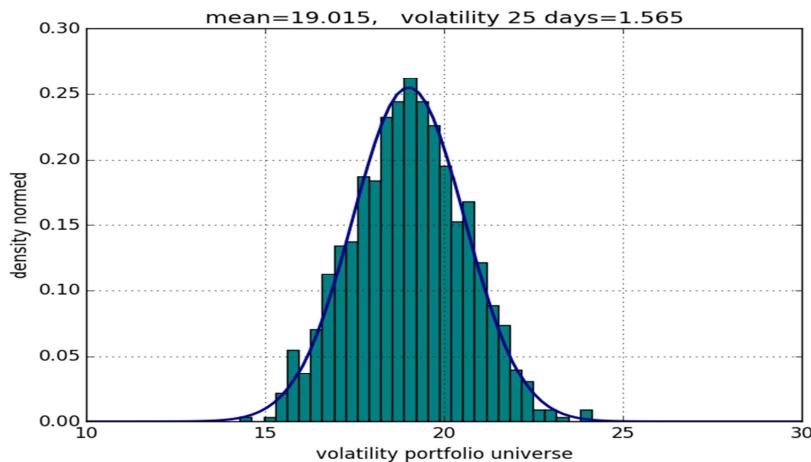


Figure 15 distribution des volatilité des portefeuilles univers sur une période 25 jours précise de l'Eurostock, la volatilité moyenne est de 19 pourcent ce qui est cohérent et fiable à exploiter

-la distribution des volatilités investissement des portefeuilles aléatoires

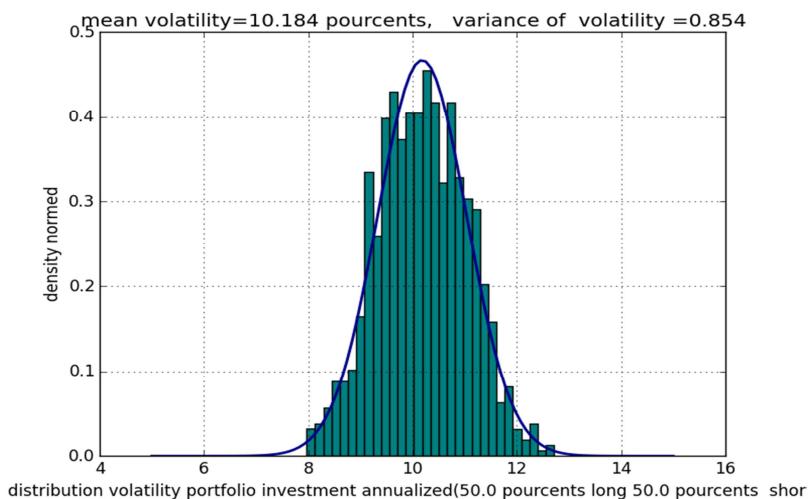
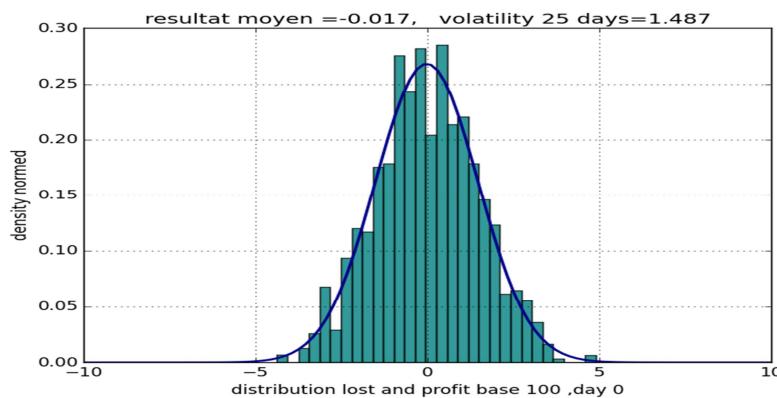


Figure 16 distribution des volatilité des portefeuille Investment sur une période 25 jours précise de l'Eurostock, la volatilité moyenne est de 10 pourcent ce qui est cohérent et fiable à exploiter

-la distribution des gains des portefeuilles d'investissement aléatoire



-la distribution des alphas avec différents hitscore.

La distribution des gains suit la loi suivante, on voit que l'on shorte 10 valeurs et on long 10 valeurs

$$somme = \sum_{i=1}^{10} S_i(25) - S_i(0) - \sum_{i=10}^{20} S_i(25) - S_i \sim N(0, \sigma)$$

Un de mes autre script à permit de réaliser un processus de sliding windows sur le temps en avançant d'un jour à chaque étape du processus et en réalisant les 4 opérations statistiques évoquées précédemment. Pour chaque journée, on calcule :

-la moyenne des volatilités univers des portefeuilles aléatoires

-la moyenne des volatilités investissement du portefeuille aléatoire

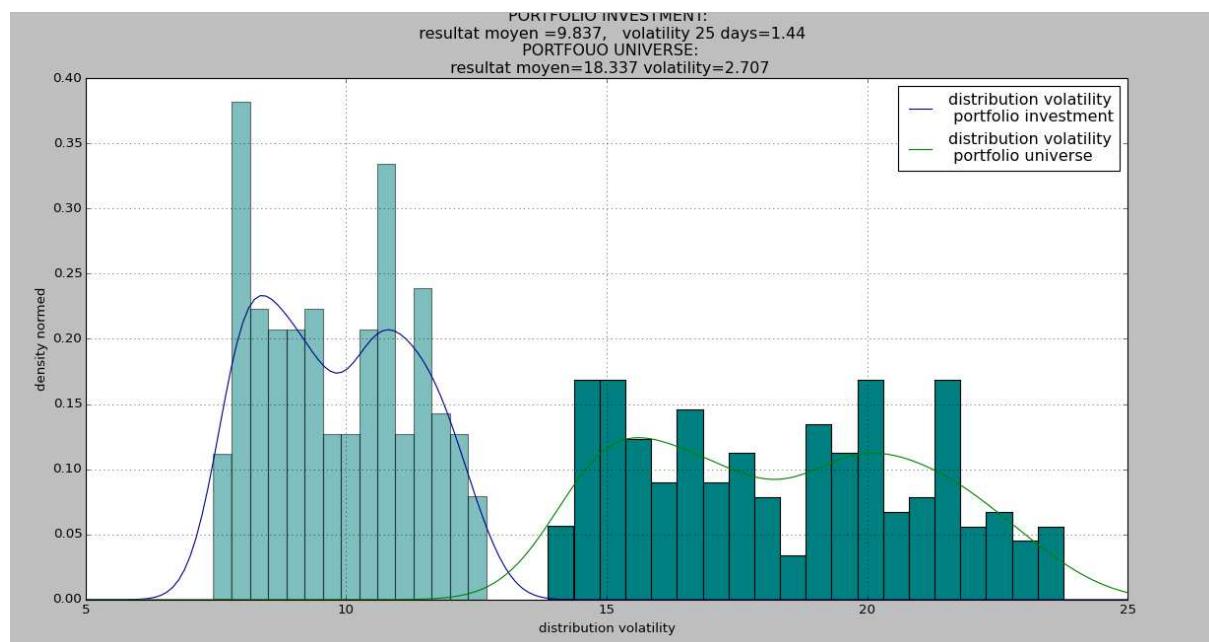


Figure 17: résultat de backtesting sur 200 jours historiques de l'Eurostock ; on voit la distribution des moyennes des volatilités Investment et univers, en génération de portefeuille aléatoire

-la moyenne des gains des portefeuilles d'investissement aléatoire

-la moyenne des alphas

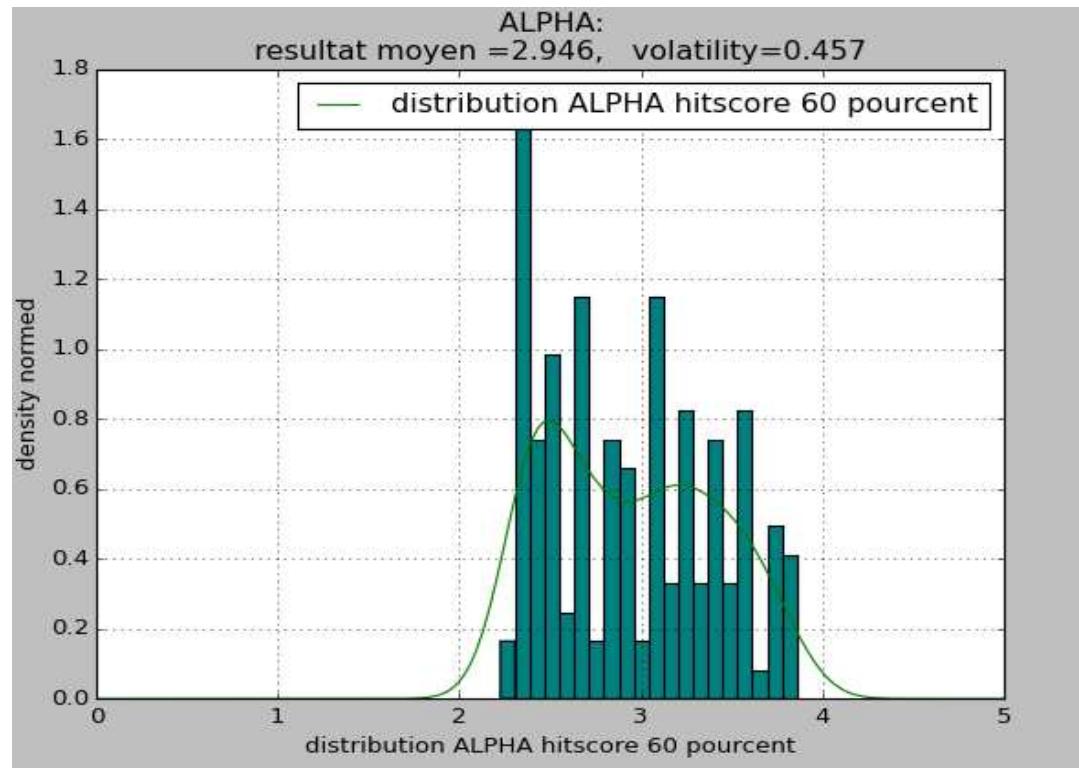


Figure 18: distribution des alphas obtenus dans le cas où le hitscore est de 60 pourcent

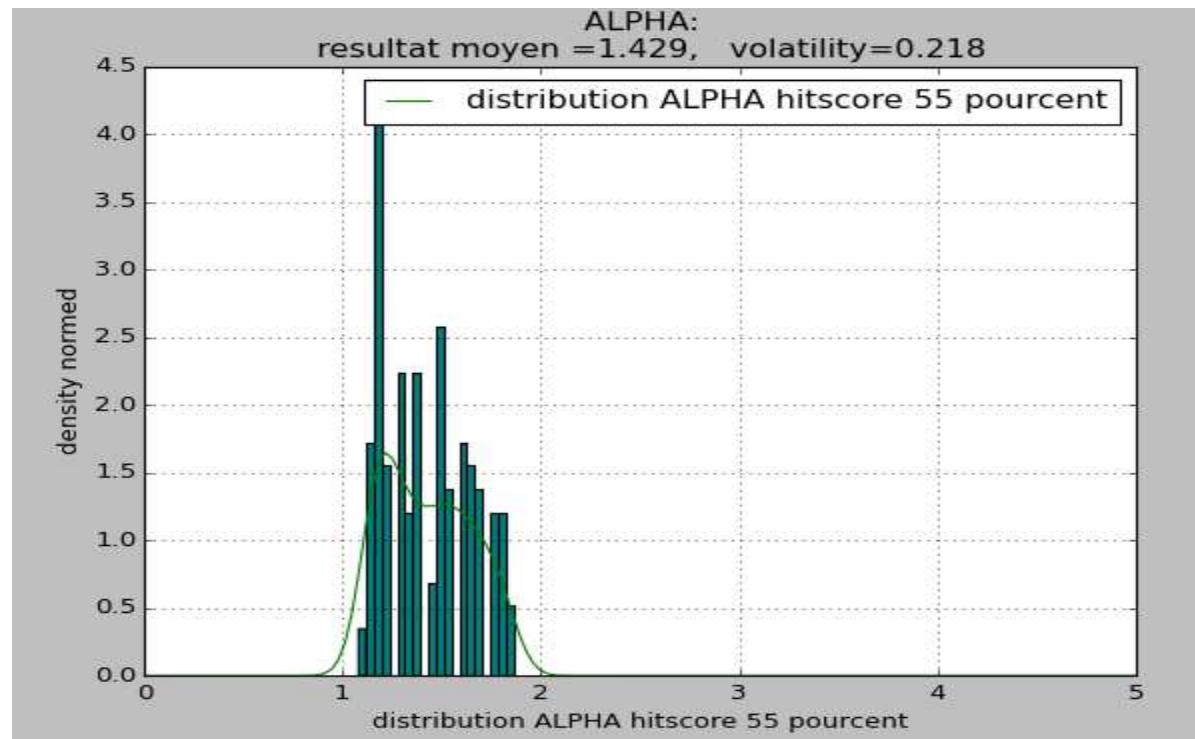


Figure 19 cas ou le hitscore est de 55 pourcents

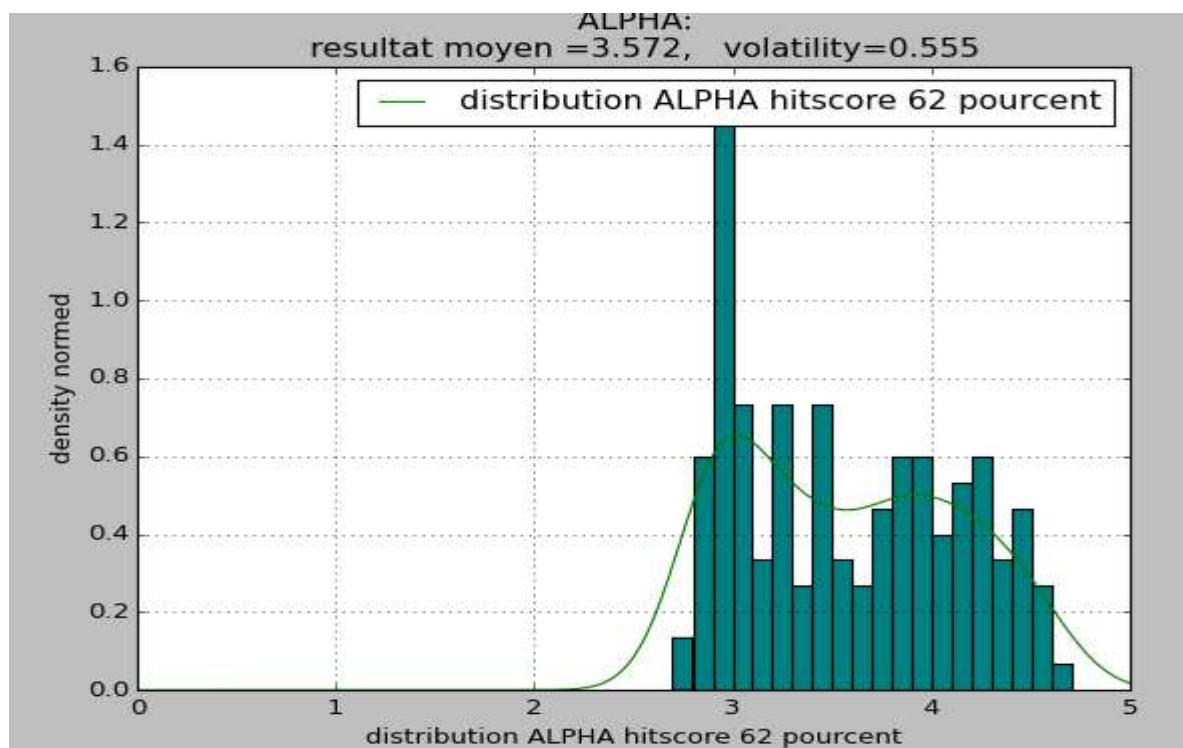
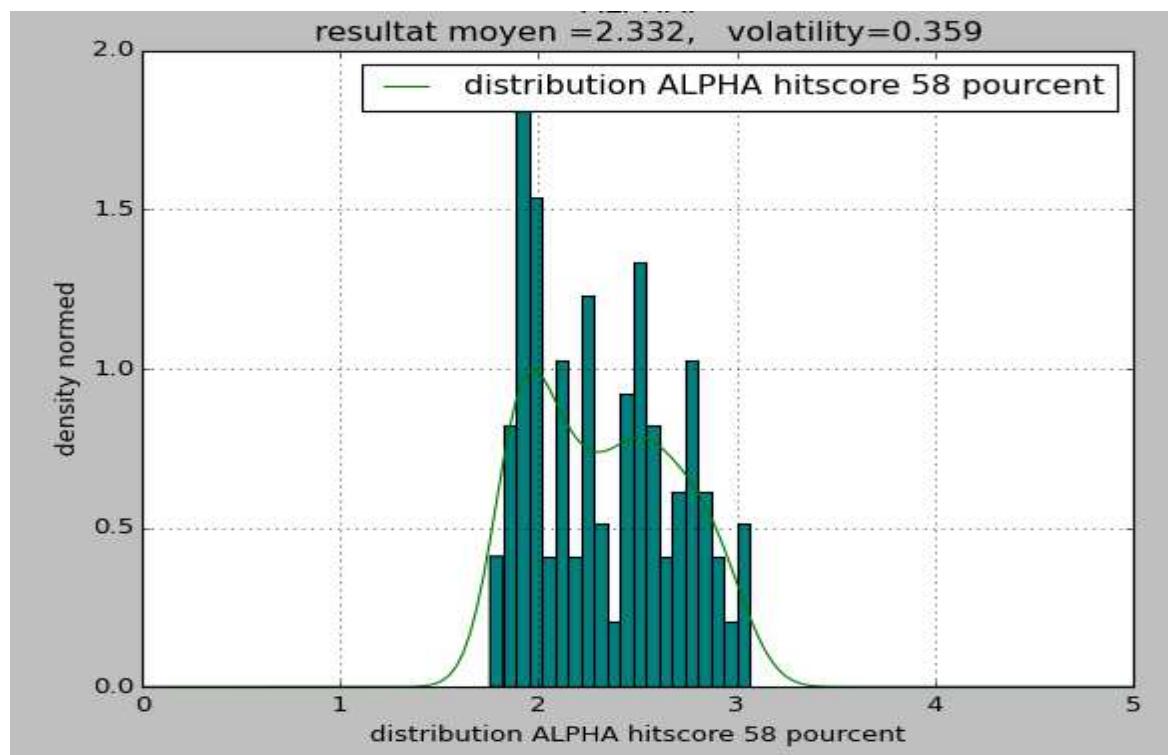


Figure 20: cas ou le hitscore est de 62 pourcents ce qui est utopique



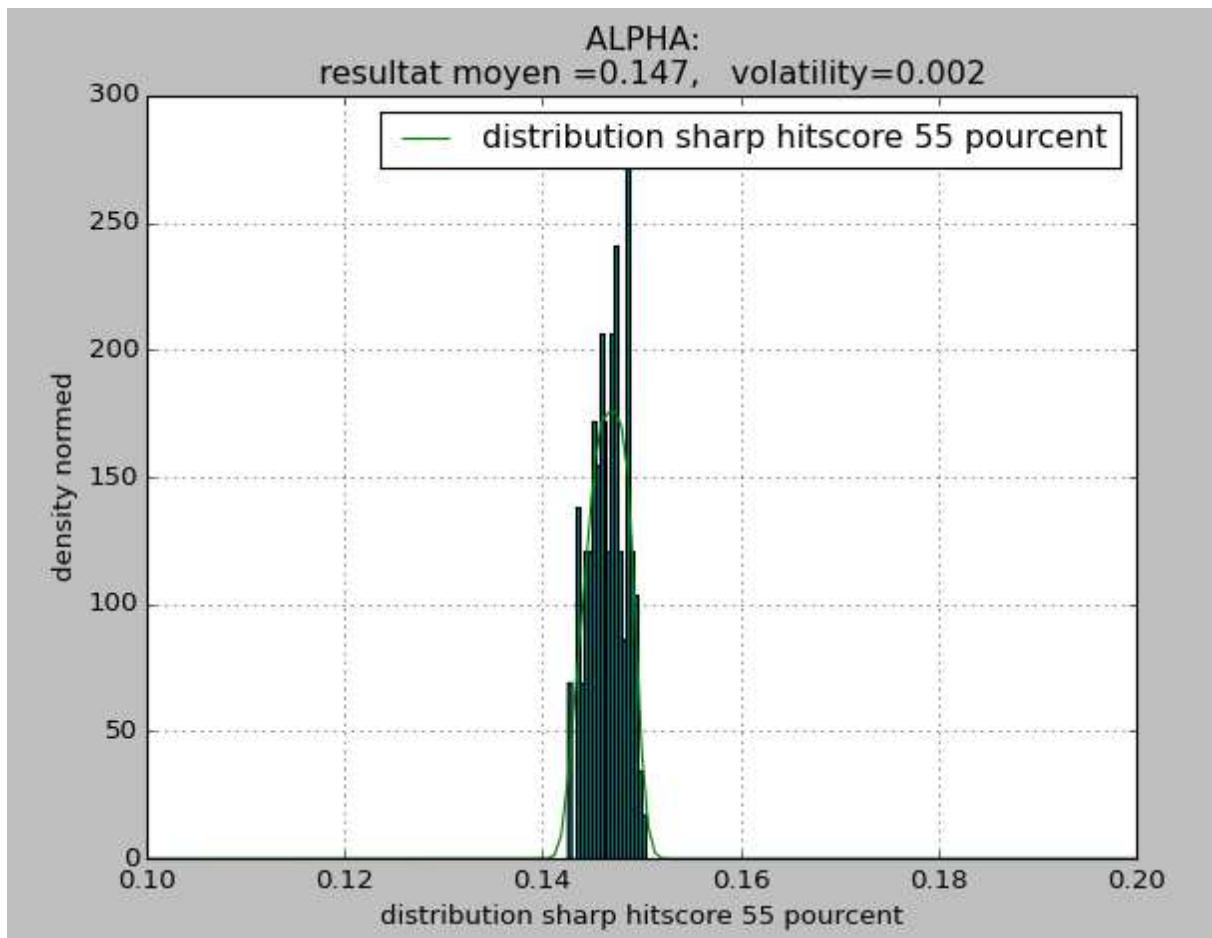
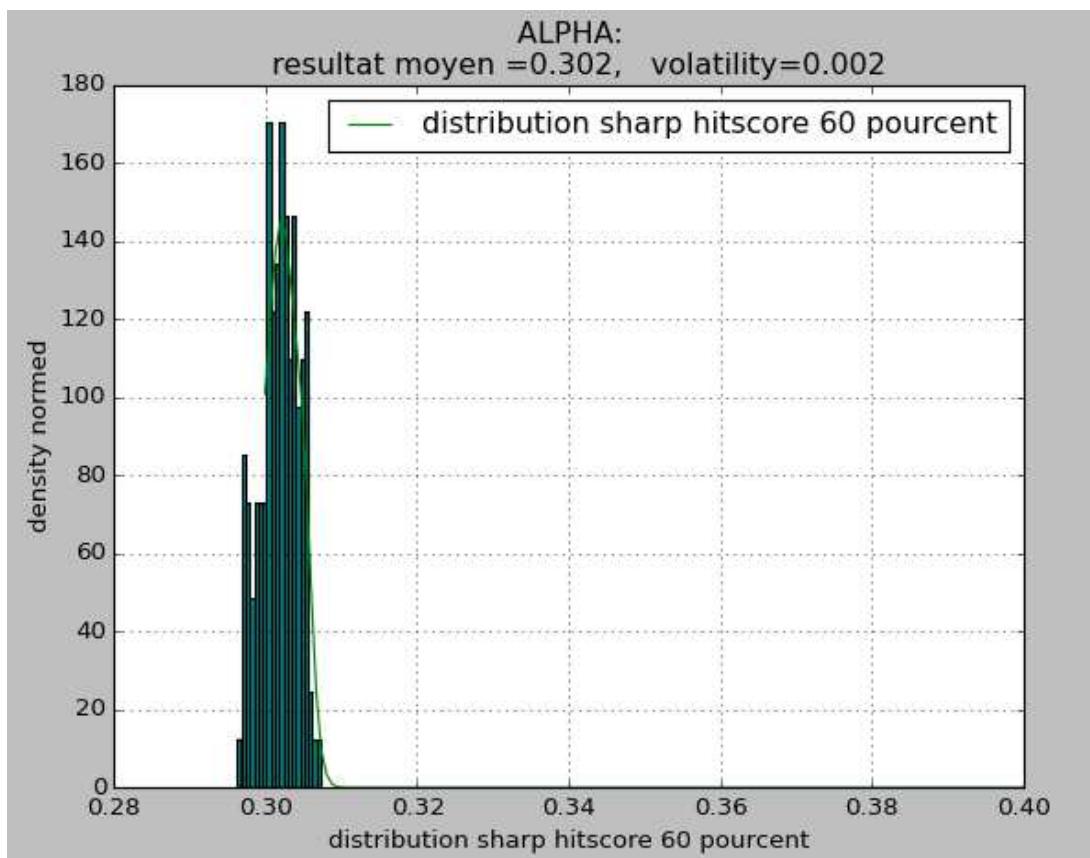
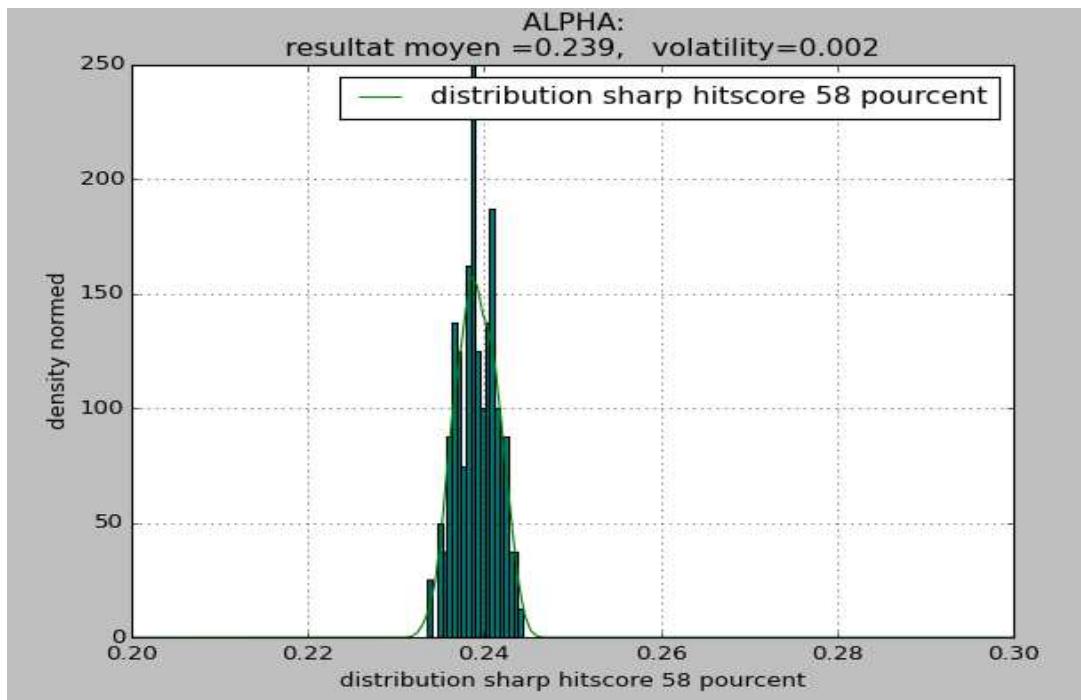


Figure 21: ratio de sharp correspondant au hitscore, celui est très bas cela est dû au portefeuille d'investissement qui n'est pas optimisé



Après avoir fait glisser cette fenêtre de 25 jours de calcul sur la durée que l'on souhaite.

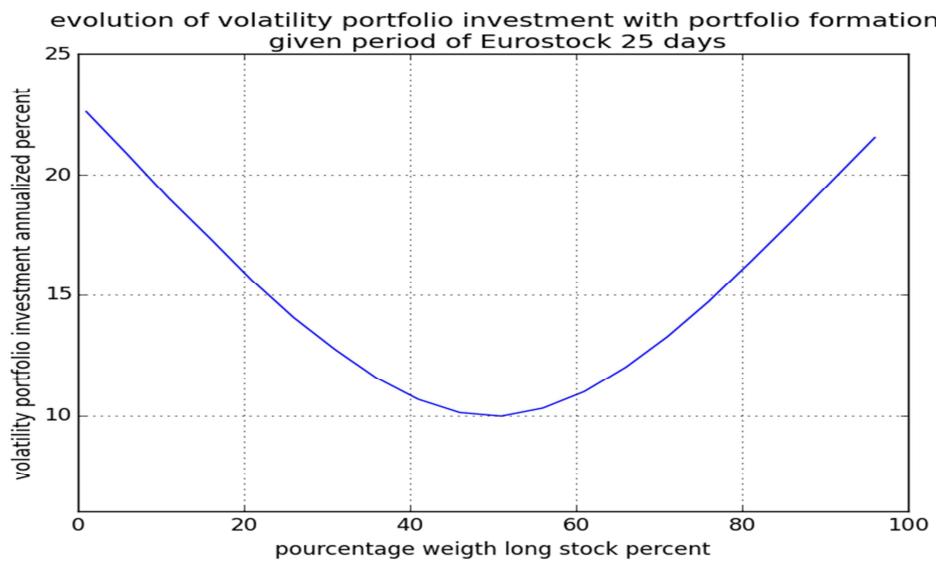
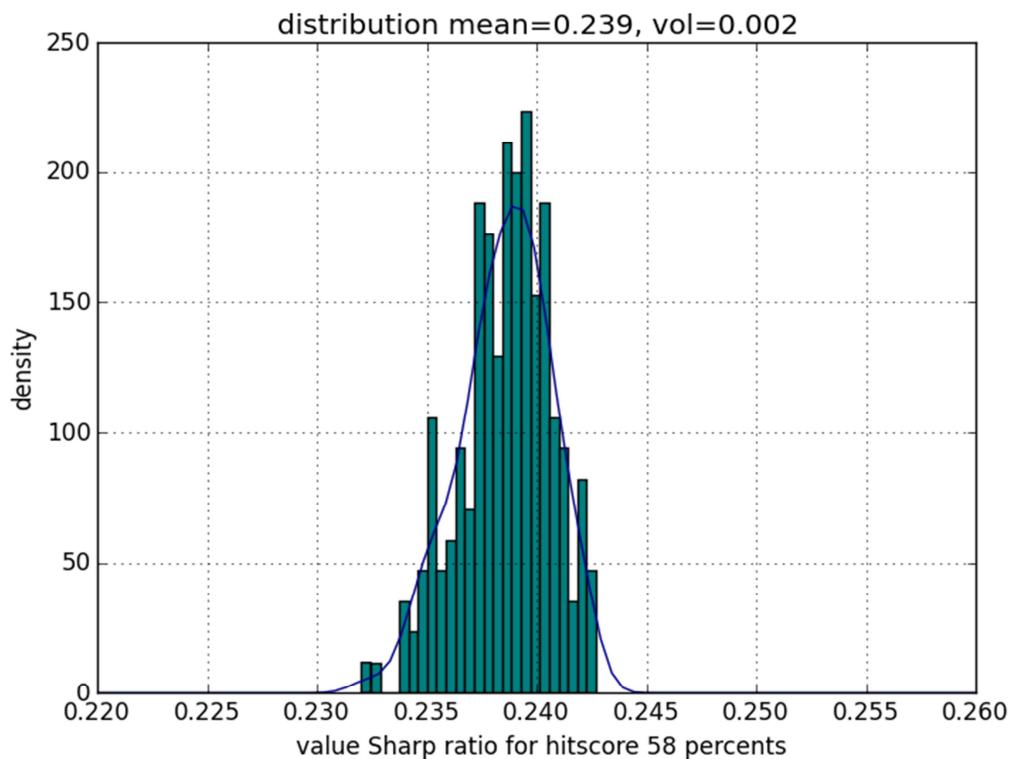


Figure 22: évolution de la volatilité du portefeuille d'investissement en fonction du rapport en proportion des poids des titres long et short. Plus on arrive vers les pondérations extrêmes, plus cette volatilité tend vers la valeur de volatilité du portefeuille univers. Cela est dû au fait que les termes en $1/s^*s$ et $1/s^*I$ tendent vers 0

A terme de ce processus de sliding windows on obtient :

- La distribution de la moyenne de la volatilité univers des portefeuilles aléatoires pour un jour dans l'intervalle
- La distribution de la moyenne des volatilités Investment des portefeuilles aléatoires pour un jour dans l'intervalle
- La distribution de la moyenne des alphas des portefeuilles aléatoires pour un jour dans l'intervalle



-La distribution de la moyenne des gains du portefeuille Investment des portefeuilles aléatoires pour un jour dans l'intervalle.

Dans un second temps j'ai voulu sortir de la filière de génération de portefeuille aléatoire, pour s'intéresser à des cas concrets ce qui est plus marquant et d'avantage illustrateur pour un utilisateur.

En laissant à l'utilisateur le choix de composer son propre portefeuille cependant le choix est toujours de 10 titres short et 10 titres longs. Il peut fixer l'identité des titres et leur pondération dans leur portefeuille global.

A partir de l'interface, l'intérêt est donc à l'utilisateur de réaliser des tests sur des données historiques. L'intérêt est de savoir calculer un alpha ponctuel sans utiliser une génération de portefeuille aléatoire .l'utilisateur peut cibler ses choix et faire baisser sa volatilité de portefeuille pour effectuer un calcul de Sharp réel. Il est à noter que chaque période est initialisée en base 100 au début de la période afin de simplifier les calculs et uniformiser les gains.

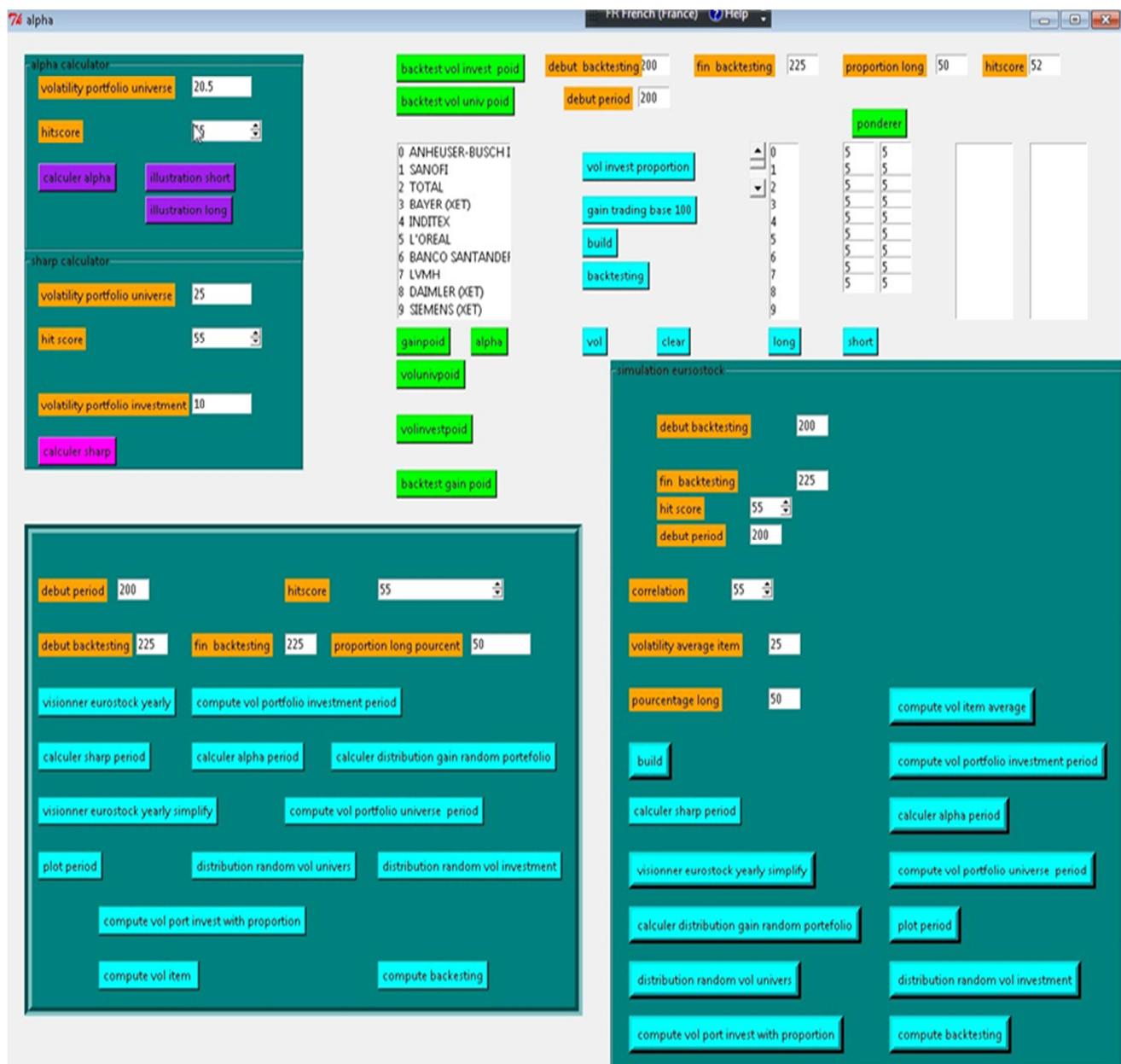


Figure 23: interface graphique créée pour illustrer l'intérêt de PRISMS dans le cadre d'une gestion de portefeuille

J'ai réalisé la répartition des poids de la manière suivante j'ai considéré que l'utilisateur avait à sa disposition un poids total à répartir de 100. Il assigne une valeur à chacun de ses titres sélectionnés. Le poids total des Shorts ne doit pas forcement être égal à celui des longs. A partir de cela la formule de la volatilité est appliquée.

Les mêmes opérations que précédemment sont à la disposition de l'utilisateur.

On peut ainsi faire backtester les gains, les volatilités portefeuille et investissement du portefeuille constitué initialement.

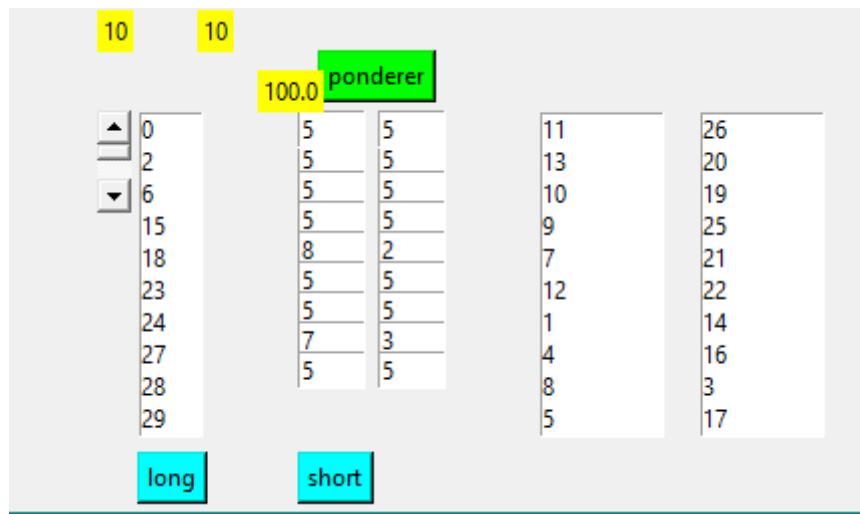


Figure 24 : interface qui permet de fixer les poids sur les titres

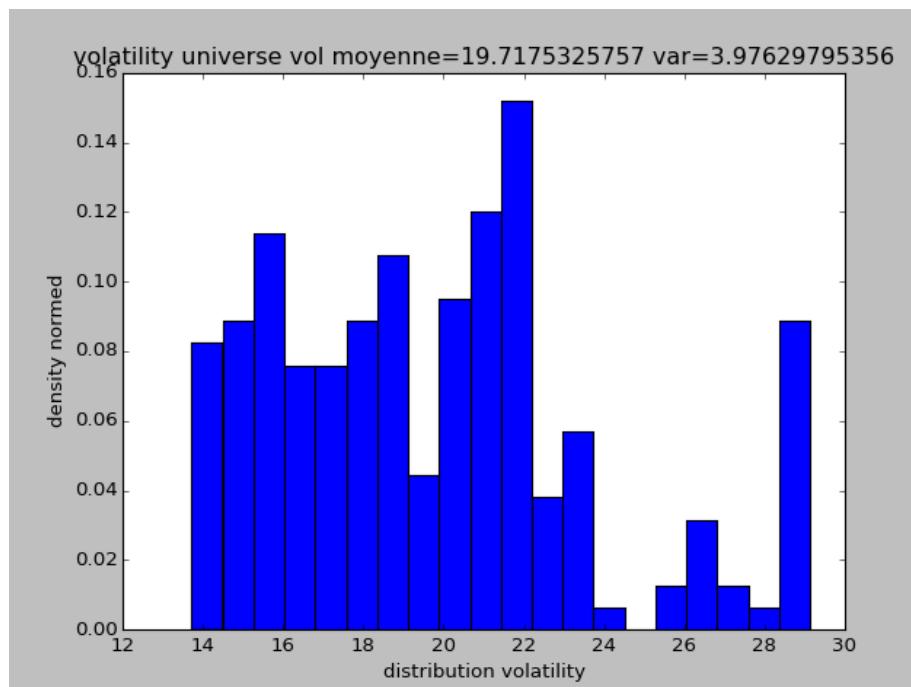


Figure 25: backtesting de la volatilité portfolio universe sur un portefeuille créé au préalable

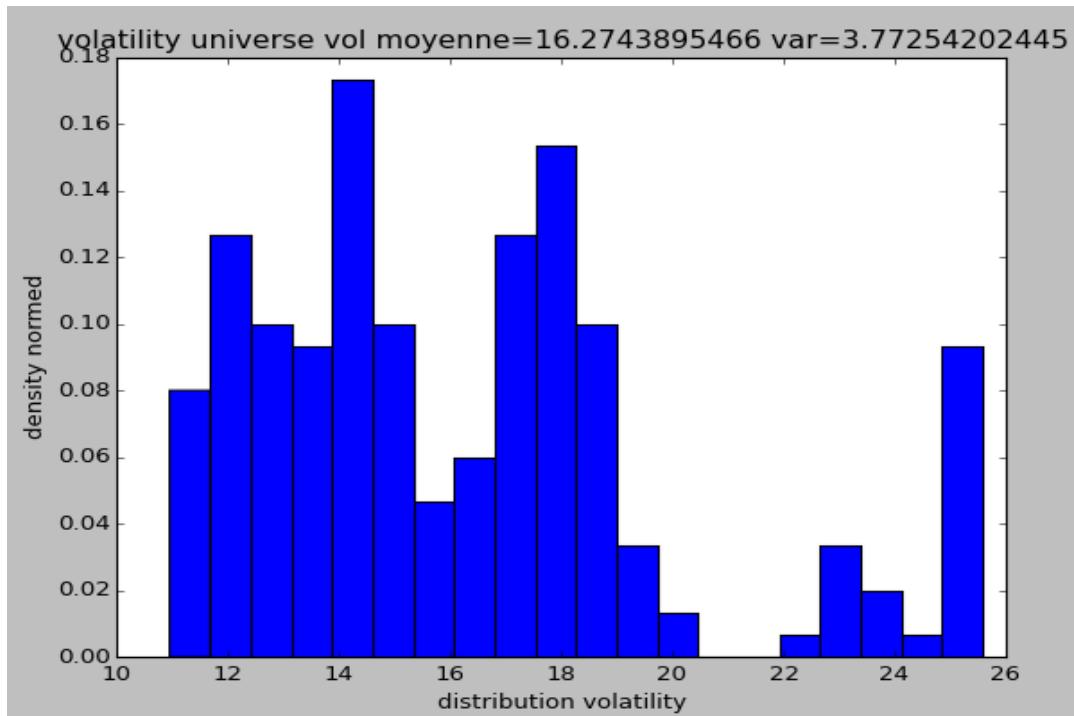


Figure 26 :backtesting de la volatilité portfolio investment sur un portefeuille créé au préalable

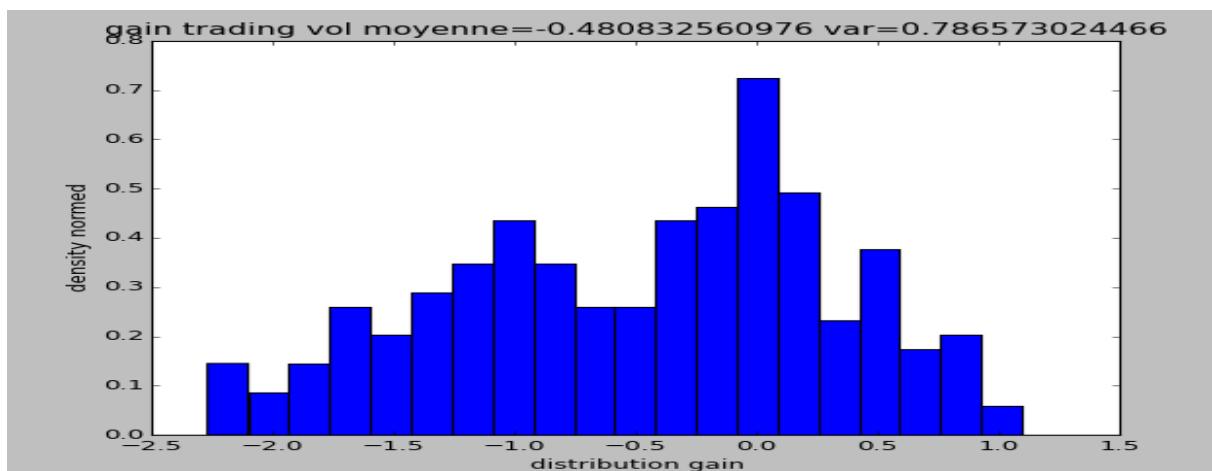


Figure 27: backtesting du gain d'un portefeuille créé au préalable

Une troisième partie du module consiste à générer des séries de 50 signaux aléatoires en influant sur deux paramètres :

- la corrélation des séries entre elles

- la volatilité moyenne d'un élément de la série. Le principe a été le suivant une fois les paramètres de l'utilisateur rentré. On génère un signal de 256 point mère de volatilité correspondant à celle rentrée.

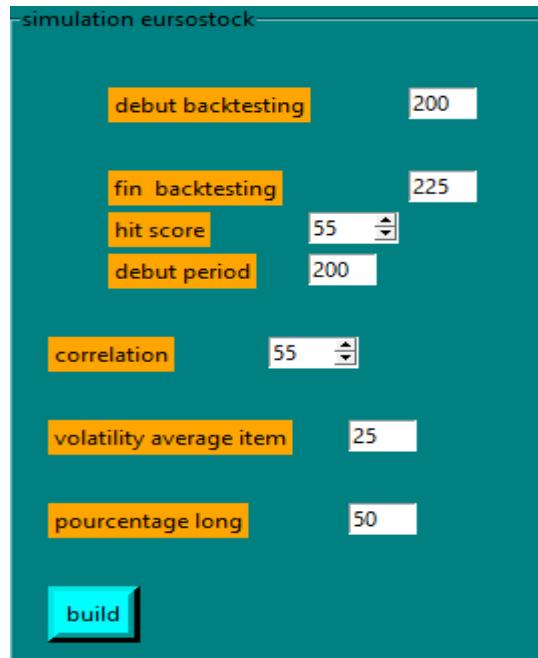


Figure 28: interface permettant de régler la corrélation et la volatilité moyenne des titres à créer

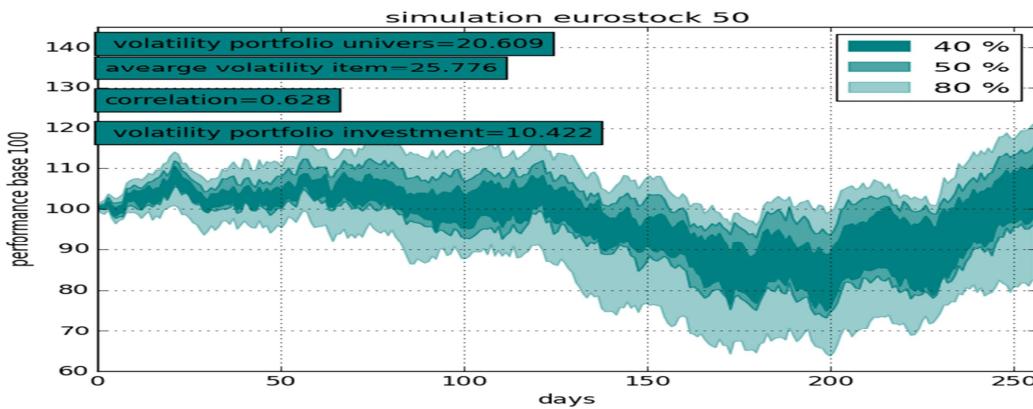


Figure 29 ici le signal générée convertit en base 100 possède une volatilité de 25 pourcent en moyenne par constituant et une corrélation de .62

On s'assure que son return annuel est compris entre .8 et 1.2 afin d'obtenir des séries réalistes avec des comportements de marchés. Une fois cette série mère créée, on crée une multitude de séries filles à partir de la série mère en récupérant la valeur de la corrélation désirée que l'on multiplie au valeur de la série mère une fois cela réalisé, on génère une partie aléatoire en plus .

```

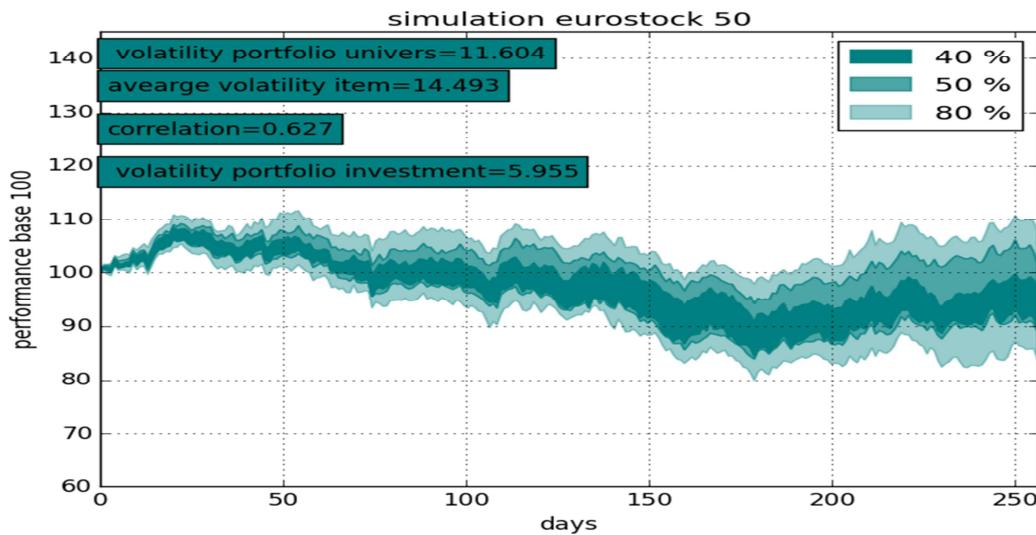
def buldsignal(listemere,a,pas):
    newsignal=[100]
    for i in range(1,len(listemere)):
        newsignal.append(newsignal[i-1]+(listemere[i]-listemere[i-1])*a+(1-a)*np.random.normal(0,pas))
    return newsignal
def orderpackage(vol,corr):
    pas=.120
    test=0
    mere=[]
    while np.abs(test-vol/corr)>2:
        mere=signalrandomyear(pas)
        test=volatility(mere)
        pas=pas+0.01
    returnmax=mere[len(mere)-1]/mere[0]
    while np.abs(returnmax-1)>.2:
        mere=signalrandomyear(pas)
        returnmax=mere[len(mere)-1]/mere[0]
    test2=0
    pas2=.40
    euro=[]
    while np.abs(test2-corr)>.02:
        euro=[]
        for i in range (50):
            euro.append(buldsignal(mere,pas2,pas))
        test2=np.mean(np.corrcoef(euro))
        pas2=pas2+.01
    return euro
def reglagebase(vol,corr):
    correl,volat,a=0,0,0
    while np.abs(correl-corr)>.01 or np.abs(volat-vol)>1:
        a=orderpackage(vol,corr)
        correl=np.mean(np.corrcoef(a))
        data=[]
        for i in range(50):
            data.append(volatility(a[i]))
        volat=np.mean(data)
    return a,correl,volat

```

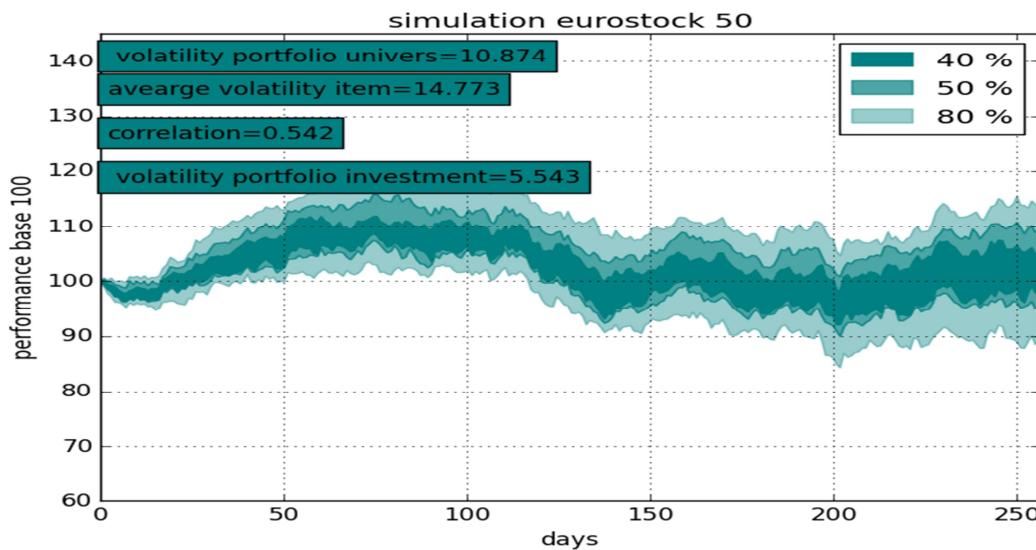
PEP 8: indentation is not a multiple of previous level's

PEP 8: missing whitespace around operator

Figure 30 processus pour générer les signaux



Il a bien sûr été question d'une période d'ajustement où chaque variance de mouvement brownien était réglée afin d'obtenir les caractéristiques du signal à obtenir.



Il faut trouver le juste milieu entre l'utilisation de while intempestif et le réglage de brownien, car lorsqu'il s'agit de variables aléatoires en théorie toute valeur peut être atteinte avec une boucle while en fonction de la distribution statistique à laquelle elle répond. Cependant ce temps d'atteinte peut être long, il faut donc calibrer la variance des browniens. Une fois ces signaux créés, on réalise les mêmes études que précédemment en analysant l'impact de la corrélation et de la volatilité choisie sur les volatilités des portefeuilles générés. Il faut cependant rester dans des valeurs de paramètres réalistes.

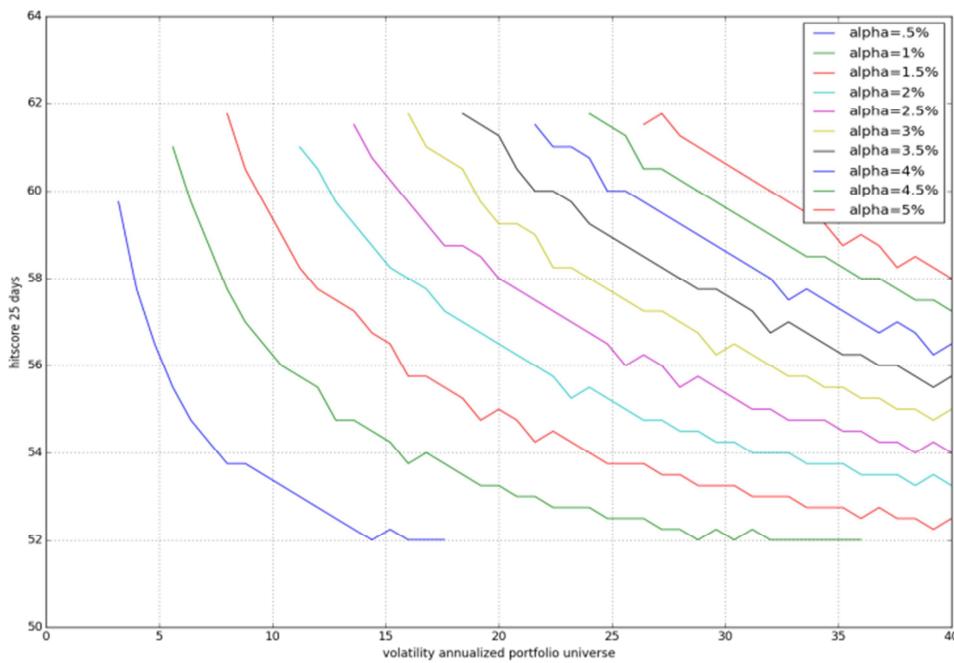


Figure 31 abaque de lignes de niveau chaque ligne de niveau représente l'hyperplan dans l'espace (volatilité, hitscore) qui correspond à l'alpha indiqué dans la légende

Ces développements informatiques ont un rôle commercial, car le projet devant être vendu et exporté doit clairement mentionner l'étape nécessaire à l'utilisateur de voir une application concrète de l'utilisation de Prismes dans sa gestion de portefeuille. Des présentations sur ce point ont été réalisées.

Mon travail était divisé en quatre parties bien distinctes :

- une partie segmentation, lavage des données,
- une partie préparation des données d'entrée et de sortie avant la modélisation,
- une partie construction du modèle,
- une partie backtesting et mise en place des intervalles de confiances.

La partie segmentation peut être vue comme un module à part son utilisation n'est pas forcément lié aux autres activités. On peut s'en servir pour avoir une approche macroscopique du signal considéré.

2.2partie segmentation

2.2.1)partition simple

Les données sont issues de Bloomberg, elles sont donc sous formes de séries temporelles de prix last Price. En fonction de la valeur concernée, on dispose donc d'une série de valeurs représentant les prix à la fin de chaque jour de bourse depuis la création de la valeur concernée. Ces séries représentent donc l'évolution des prix depuis la création de la valeur. Chaque série rentrée est caractérisée par son vecteur date et son vecteur valeur. La construction d'un modèle de machine Learning se déroule au moins sur 200 jours.

L'étape de segmentation est réalisée sur une série temporelle sélectionnée, elle consiste à partitionner le signal, en différent régimes temporels, qui est caractérisé par le même return pour chaque valeur de ce régime et la même volatilité.

La partie segmentation a 3 buts :

-permettre à l'utilisateur de **visionner son signal « débruité »** en dégageant pour une période historique les tendances réelles et le partitionnement de l'évolution de cette valeur.

-le signal segmenté va être utilisé pour créer des prédicteurs en vue de l'apprentissage statistique, dans un second temps, les fichiers segmentation seront sauvegardés pour la suite du processus.

Certaines options de calcul de return et de volatilité ont besoin du signal segmenté pour être activées.

-intérêt scientifique et cohérence de scientifique de l'ensemble du projet. Les algorithmes reposent aussi sur du machine learning.

J'ai créé un algorithme de segmentation. Ma méthodologie de création fut la suivante. Le logiciel officiel dispose initialement de deux algorithmes pour réaliser cette activité :

-dcw

-cart.

Le processus est le suivant :

On considère que l'on souhaite réaliser une segmentation sur une période T de 256 jours d'un signal (1 an de jours de bourse). Les données Bloomberg sont sauvegardées dans des fichier.mat, à partir des dates de début et de fin de partition, on extrait les valeurs nécessaires à être traitées en recherchant les indices du vecteur valeur correspondant aux dates souhaitées

On assimilera cela à un vecteur de 256 valeurs journalières représentant le prix de la valeur.

- 1) pour chaque jour on calcule le Daily change (la différence du cours entre aujourd’hui et la veille). On obtient donc un vecteur de 255 valeurs.
- 2) Il faut être conscient que dans ce vecteur de 255 valeurs. Certaines valeurs qui se succèdent sont proches, des périodes de jours successifs se dégagent.

On considère **une période [T0Tn] de jours successifs ayant une un return similaire** chaque return va ainsi être remplacé par la valeur $(\text{Return}_{T0-Tn})/n$, pour ainsi créer un signal linéaire sur cette période

1 2,2 2,3 1,9 2,1 8 15 -5 -6 -8 -7,2 -6,8 -8,2



2.2.1méthode

2.2.1.1PREMIERE METHODE POUR DETECTER UN REGIME

On définit un taux qui correspond à la différence maximale entre la première valeur d'une potentielle zone et les valeurs suivantes de la zone. Par exemple on considère dans la zone 1 un taux de 1.5. Toutes les valeurs sont à moins de 1.5 de la première valeur de la zone seront enregistrées comme appartenant à cette zone.

```
def detectvraisemblance(X,taux):
    temp,i=[],1
    while (i<len(X)):
        j,ind,a=i+1,0,0
        while(j<len(X) and np.abs(np.float64(X[j]-X[i]))<float(taux)):
            a=a+1
            if controldrift(X[i+1:i+a+1],taux,0.9,X[i])==0:
                break
            ind,j= ind+1, j+1
        temp.append(i)
        temp.append(ind)
        i=i+ind+1
    return temp
```

On considère les valeurs dans l'ordre chronologique. Ainsi on regarde les valeurs suivantes et voir si successivement elles sont dans l'intervalle fixé par le taux. Il y a deux possibilités ou bien il n'y a pas de valeurs, on passe donc à la valeur suivante dans l'ordre chronologique et on réitère ce procédé de recherche, ou bien on détecte une zone (composée d'au moins 2 dates).

Dans le cas d'une détection de zone, on comprend bien que **cette détection traduit le fait que chaque valeur de la zone n'est pas trop éloignée de la première valeur de la zone, puisque sa différence avec la première valeur est inférieure au pas fixé.**

On comprend que si chaque valeur est la limite de cet intervalle, on peut assister à une légère dérive, par rapport à la réalité du signal. **En effet cette détection est rigide puisque tout est basé sur la première valeur.** Il faut donc éviter cela. Pour chaque vecteur d'une zone, on calcule le vecteur de somme cumulée de ce vecteur. On vérifie que chaque valeur de ce vecteur somme cumulée ne dépasse une valeur caractérisée par une somme du taux à la puissance. On vérifie donc que cette hypothèse est vérifiée. Si celle-ci est vérifiée on a donc obtenu une zone.

```
def controldrift(liste_taux,drift,val0):
    res=1
    if len(liste)>1:
        listel,borne=np.cumsum(liste),[]
        borne.append(taux)
        for i in range(1,len(liste)):
            borne.append(borne[i-1]+pow(drift*taux,i))
        for i in range(len(liste)):
            if np.abs(listel[i]-val0)>np.abs(borne[i]):
                res=0
    return res
```

Une fois les régimes détectés il faut s'assurer de la continuité temporelle dans la succession de régimes en y incorporant les valeurs qui sont laissées solitaires.

2.2.1.1.2DEUXIEME METHODE DE DETECTION DE ZONE

On définit un taux qui correspond à la différence maximale entre la moyenne des valeurs déjà présentes dans la zone et la valeur suivante testée pour son intégration ou non dans le régime. Par exemple on considère dans la zone 1 un taux de 1.5. On considère qu'on a déjà identifié trois dates successives pouvant former une zone. La quatrième date sera intégrée dans cette zone seulement si la différence entre la moyenne des trois valeurs de Daily change déjà présente dans cette zone et la valeur de la quatrième date est inférieure à ce pas. Si cette condition est vérifiée la quatrième date intègre ce régime sinon le régime s'arrête ici, et on réitère un nouveau processus de recherche de période à la 5 eme date.

La zone est caractérisée par l'index de la première valeur et l'index de la dernière valeur. On réitère ensuite le procédé de recherche de zone en commençant par la valeur après l'index de la dernière valeur de la zone trouvée précédemment.

On réalise ce procédé pendant toute la longueur du signal à partitionner.

A la fin de l'étape de partition, on a donc les coordonnées en dates d'un ensemble de zones que l'on appellera régimes.

2.2.1.2 SIMPLIFICATION DE LA SERIE ;

Une fois la partition du signal réalisée, il faut se servir de ces régimes déterminés précédemment.

Pour chaque régime ; on accède à la valeur réelle du signal à l'index du début du régime et à l'index de la fin du régime. Chaque valeur à l'intérieur du régime est associée à la valeur de l'interpolation linéaire des deux valeurs précédemment citées. On a donc un signal linéaire à l'intérieur d'un régime.

Il s'agit donc d'une simplification et d'un debruitage. On réalise cela pour chaque régime de la partition réalisée. On obtient donc un signal partitionné.

```
def affichercourbesimplifie(X,taux):
    data,bucket=detectvraisemblance(computedist(X),taux),[]
    newsig,temp=[None]*len(X),len(data)/2
    newsig[0]=np.float64(X[0][0])
    bucket.append(0)
    for i in range(2,temp+1):
        newsig[data[2*i-2]-1]=X[data[2*i-2]-1][0]
        bucket.append(data[2*i-2]-1)
        newsig[len(X)-1],newsig[len(X)-2]=X[len(X)-1][0],None
    newsig=cleansheet(newsig)
    bucket.append(len(X)-1)
    if len(X)-2 in bucket:
        bucket.remove(len(X)-2)
    return newsig,bucket
```

Une des parties importantes fut le réglage du taux, l'idéal est d'avoir un signal où on détecte 5 régimes qui se succèdent sans laisser des valeurs seules pour 256 jours. Il faut trouver le juste milieu.

En cas d'une valeur de pas inappropriée on assistera à des partitions simplificatrices ou trop segmentées avec des régimes ne comportant pas assez de valeurs. Les signaux à traiter sont variés : leur aspect est différent et les valeurs que prennent les séries sont différentes. Il faut donc s'assurer que le pas est choisi en fonction du signal.

L'algorithme comprend un ajustement du pas pour avoir un nombre de régime cohérent. Une question qui se pose est de savoir comment choisir le taux en fonction du signal à partitionner. Je fixais le taux à un pourcent de la moyenne des valeurs du signal dans la période considérée. A l'aide d'une boucle while certaines valeurs de pas sont testées et associées à un nombre de régimes créé.

Le pas est retenu en fonction du nombre de régimes créés.

```

def detecterpasoptimal(reel,index1,index2):
    pas=89
    i=10
    if index2-index1>80:
        while i>0:
            pas=np.mean(reel)/100*i
            reg=creerregime(affichercourbesimplifie(reel,pas)[1])
            if len(reg)>3:
                break
            else:
                i=i-1
    if index2-index1>=50 and index2-index1<80:
        while i>0:
            pas=np.mean(reel)/100*i
            reg=creerregime(affichercourbesimplifie(reel,pas)[1])
            if len(reg)>2:
                break
            else:
                i=i-1
    if index2-index1>=30 and index2-index1<50:
        while i>0:
            pas=np.mean(reel)/100*i
            reg=creerregime(affichercourbesimplifie(reel,pas)[1])
            if len(reg)>1:
                break
            else:
                i=i-1
    if index2-index1<30:
        pas=np.mean(reel)/100*5
    return pas

```

Figure 32: réglage du pas pour avoir un nombre de régimes adéquats en fonction de la durée de la partition envisagée.

2.2.1.3 MOVING AVERAGE

J'ai incorporé une option qui consiste à calculer la moyenne mobile d'un signal afin de réaliser un lissage et de recueillir des prédicteurs efficaces. J'ai mis un entrée un paramètre P. Le calcul à partir de la série brute s'effectue de la manière suivante. Soit le signal en entrée [R1...Rend]

Pour les valeur i dans l'intervalle temps [1,p] :

$$\text{On calcule } \sum_1^{2i} \frac{R_j}{2*i}$$

Pour les valeur i dans l'intervalle temps [P,Tend-P] :

$$\sum_{i-P/2}^{i+P/2} \frac{R_j}{2 * i}$$

Pour les valeurs de i dans l'intervalle temps [Tend-P,Tend] :

$$\sum_{i=P/2}^{i+Tend-} \frac{R_j}{2 * i}$$

```
def movingaverage(liste,pace):
    mov=[]
    mov.append(liste[0][0])
    for i in range(1,pace):
        mov.append(np.mean(liste[:2*i]))
    for i in range(pace,len(liste)-pace):
        mov.append(np.mean(liste[i-pace:i+pace]))
    for i in range(len(liste)-pace,len(liste)):
        mov.append(np.mean(liste[i-(len(liste)-i):]))
    return mov
```

Figure 33 :implémentation de la moving average



Figure 34: exemple de segmentation

2.2.1.4 CALCUL DE LA VOLATILITE

Une des raisons d'effectuer une partition est le calcul de la volatilité. En effet chaque régime doit être assimilé à une volatilité qui est unique et qui lui est propre. Pour ce calcul de volatilité on va utiliser deux sources de données. Le signal réel et le signal simplifié.

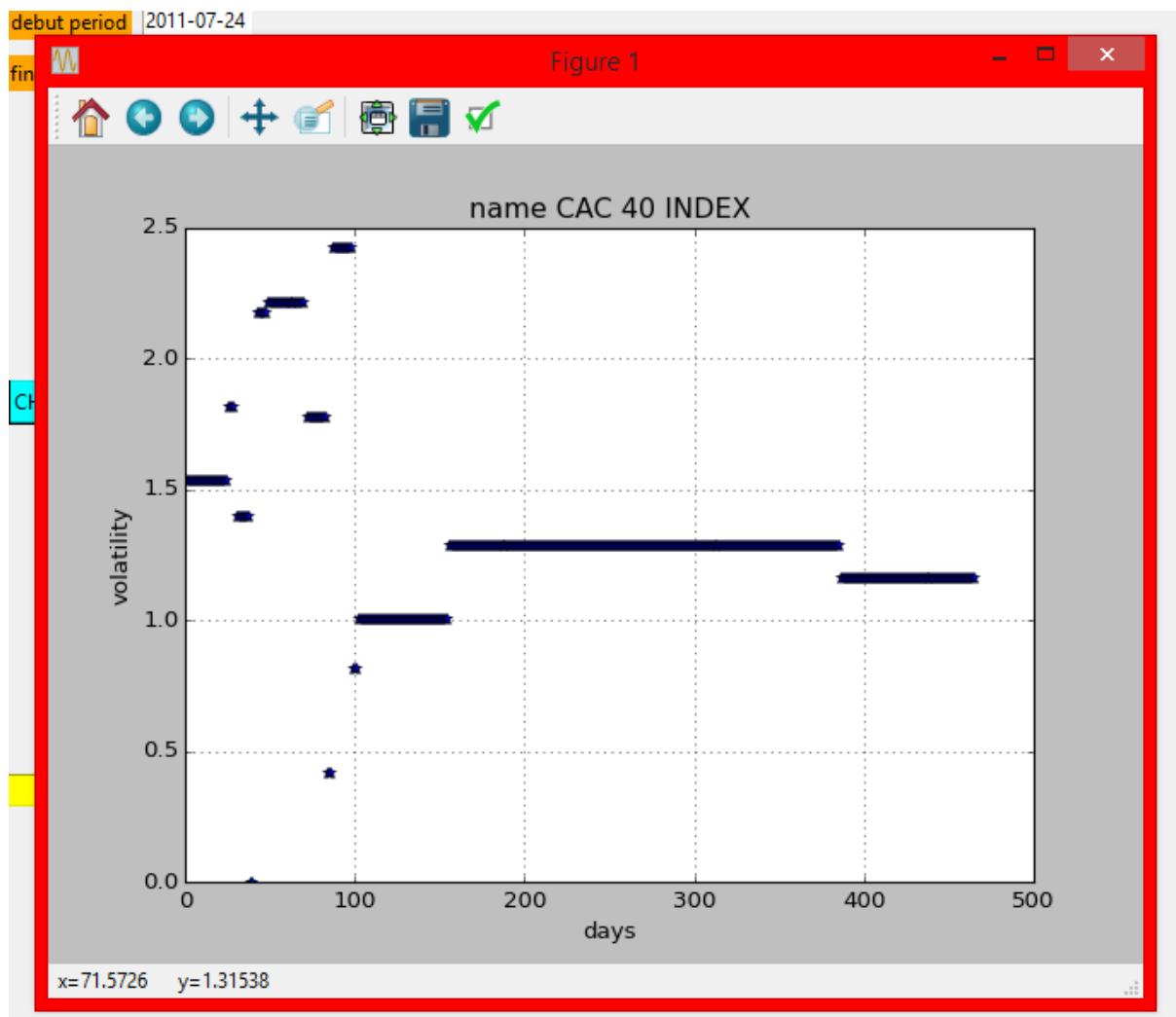


Figure 35: volatilité en fonction des régimes

Au sein de chaque régime on calcule le return du signal simplifié c'est donc le même pour chaque date du régime, par définition de la partition. On calcule le return pour chaque jour de la courbe réelle.

$$\text{vol daily seg}^2 = \frac{\sum_{i=1}^T (R_{\text{real}} - R_{\text{seg}})^2}{T} \quad \text{avec } T \text{ nombre de jour du régime}$$

Une fois ces données calculées, on calcule la volatilité en mettant au carré l'écart de chaque return réel et le return simplifié. Il faut donc que le régime est un nombre suffisant de jours sans cela le calcul de volatilité est erronée.

Cela est assimilable à un calcul de volatilité journalière basique sur la courbe réelle pendant un intervalle de temps assimilé aux régimes, cependant ici ce n'est pas la moyenne des return réels pendant cette période mais le return journalier du signal partitionné.

$$\text{vol daily classique}^2 = \frac{\sum_{i=1}^T (Rreel_i - E(Rreel_i))^2}{T} \quad \text{avec } T \text{ nombre de jour du régime}$$

Chaque régime est donc assimilé à trois données :

La période (segment composé de la date de début du régime et de la fin du régime)

Le return journalier associé à cette période.

La volatilité journalière associée à cette période et calculée pendant celle-ci.

Moi, personnellement je l'ai stocké dans une matrice de largeur 4 (date début de régime, date de fin de régime, return associé au régime et volatilité) et de hauteur correspondant au nombre de régimes

Une partition d'un signal est donc composée d'une succession de régimes chronologiques.

reg1	734689.0	734724.0	-1.204	2.542
reg2	734725.0	734725.0	0.0	0.0
reg3	734726.0	734732.0	1.569	2.478
reg4	734733.0	734734.0	-3.785	3.174
reg5	734737.0	734747.0	0.854	1.571
reg6	734748.0	734752.0	-3.264	2.272
reg7	734753.0	734754.0	1.986	2.584
reg8	734755.0	734758.0	-3.889	0.05
reg9	734759.0	734767.0	-0.326	4.71
reg10	734768.0	734768.0	0.0	0.0
reg11	734769.0	734802.0	0.707	4.563
reg12	734803.0	734803.0	0.0	0.0
reg13	734804.0	734807.0	-3.062	3.053
reg14	734808.0	734808.0	0.0	0.0
reg15	734809.0	734825.0	-0.183	2.885
reg16	734828.0	734831.0	-1.524	1.585
reg17	734832.0	734850.0	0.645	5.097
reg18	734851.0	734856.0	-0.871	2.395
reg19	734857.0	734933.0	0.211	1.105
reg20	734934.0	734934.0	0.0	0.0
reg21	734935.0	735266.0	0.035	1.654
reg22	735269.0	735269.0	0.0	0.0
reg23	735270.0	735385.0	0.09	1.355

Figure 36 : affichage d'une partition

Nous avons donc défini une opération de partition. Nous allons maintenant définir l'action de partition globale.

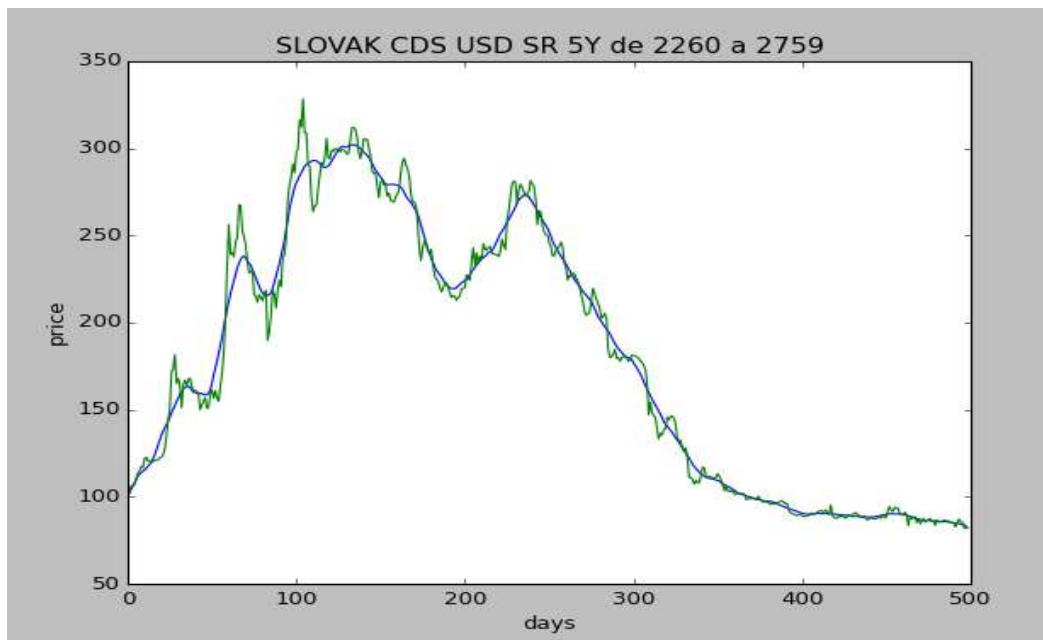
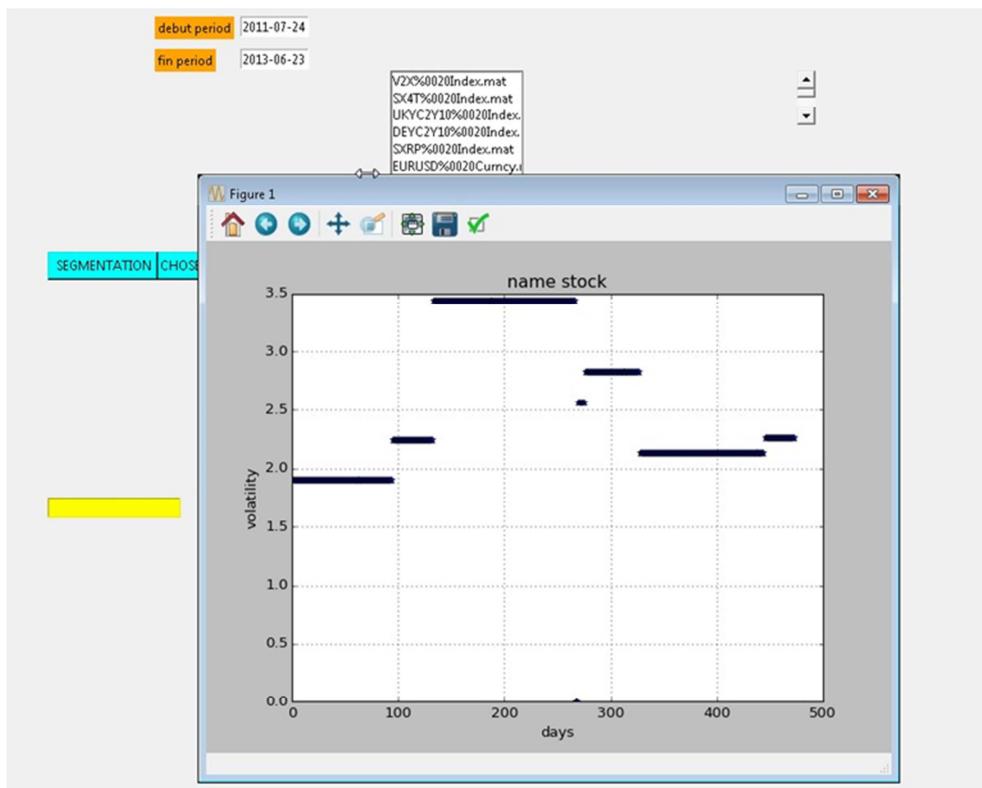


Figure 37 : moving average parameter 10

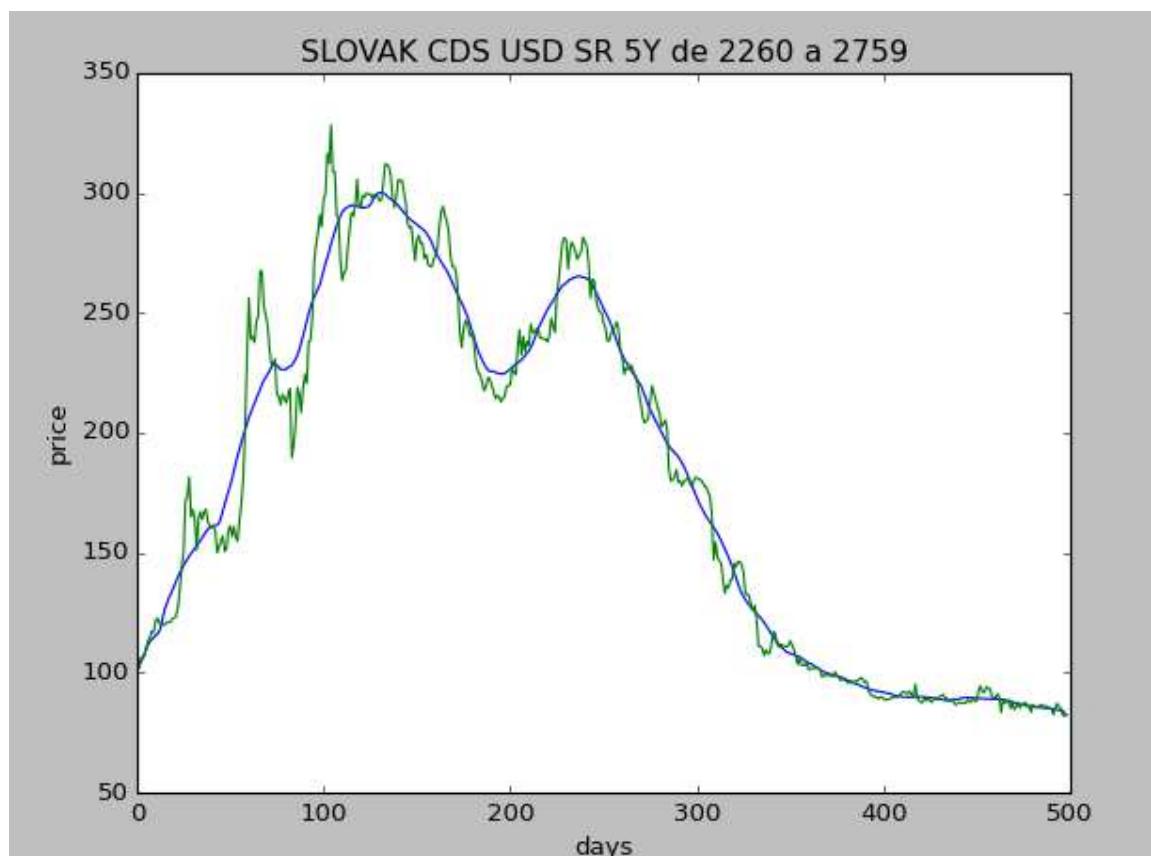


Figure 38: moving average parameter 15

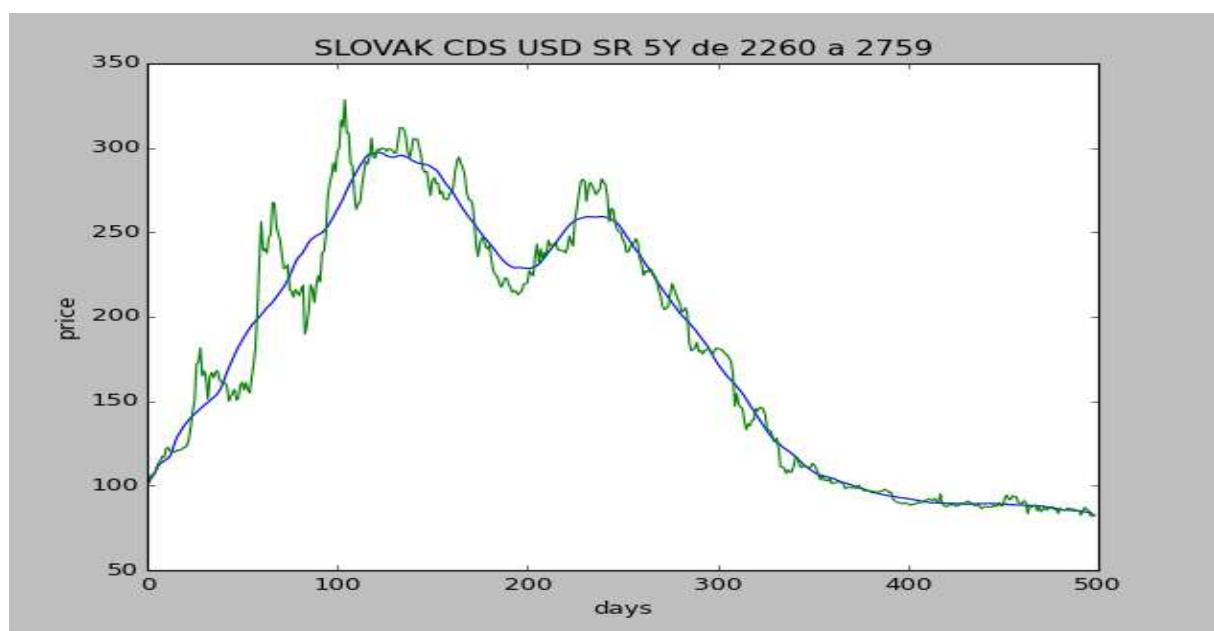


Figure 39: moving average paramètre 20

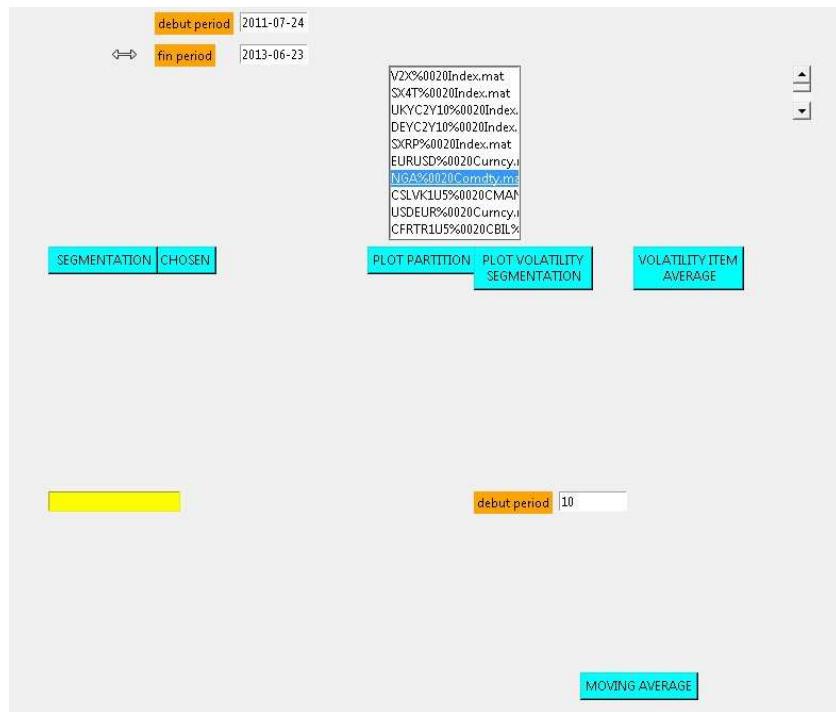
2.2.2) PARTITION GLOBALE

Cette opération peut nous entraîner dans un piège, je considère la situation suivante.

Je réalise une partition sur une période $T = [\text{ind1}, \text{ind2}]$

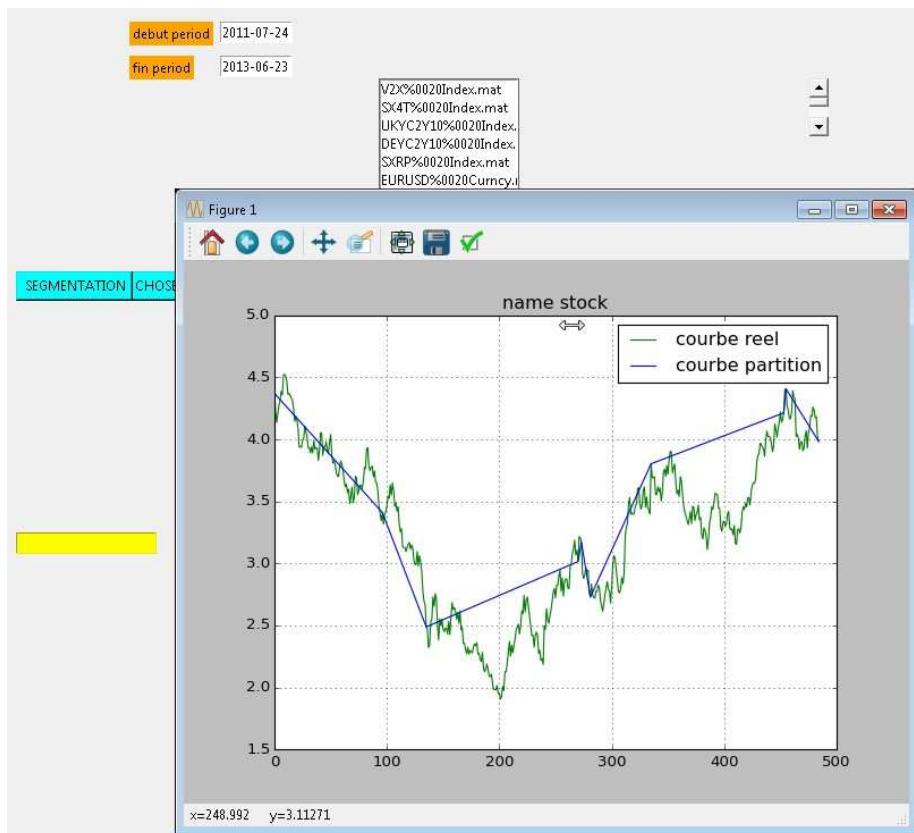
On imagine que je souhaite obtenir la volatilité journalière du signal à une date T inter représentée par un index INDinter . Un des réflexes est de regarder dans quel régime est inclus l'index inter et de récupérer la volatilité de ce régime pour l'assimiler à la volatilité que je recherche. **CEPENDANT le calcul de volatilité de ce régime se sert d'informations situées chronologiquement après inter index, car le régime ne s'arrête pas à T inter.**

Cela n'est pas un problème si l'utilisateur utilise la partition à des fins autres que de construire des modèles. (C'est pour cela que l'action de partition est un module séparé du logiciel globalement).



Le modèle est ensuite utilisé pour réaliser des prédictions à partir d'informations journalières dans le présent. Il n'est donc pas possible de construire des modèles à partir de calcul de volatilité qui se servent de données futures. Le modèle serait construit certes mais aurait des prédictions erronées.

Il faut impérativement créer la base de construction de la même manière que pour réaliser une prédiction à l'instant t . A l'instant t l'utilisateur peut éventuellement réaliser une partition mais donc forcément sur du passé.



Partition totale :

On considère que l'on souhaite traiter un signal sur une période $T=[T_{deb}, T_{fin}]$ d'une durée de 200 jours (fixé pour l'exemple) ;

On va réaliser 195 partitions :

Partition 1 ($[T_{deb}, t_5]$) (on commence sur une période de 5 jours il n'est pas très judicieux de partitionner un signal de 4 valeurs)

Partition 2 ($[T_{deb}, t_6]$)

Partition 3 ($[T_{deb}, t_7]$)

Partition 18 ($[T_{deb}, t_{22}]$)

.....

Partition 100

```

def runpartitiontotale(name,index1,index2):
    listepartition=[]
    for i in range (5,100):
        print(i)
        listepartition.append(runpartition(name,index1,index1+i))
    i=100
    while index1+i<=index2:
        print(i)
        listepartition.append(runpartition(name,index1+i-100,index1+i))
        i=i+1
    return listepartition

```

Figure 40 on voit bien ici le processus de sliding windows

Il s'agit d'un processus de sliding Windows. Ici la date de début est Tdeb +i et la date de fin de Tdeb +i +100. J'ai jugé que réaliser des partitions de 100 jours glissants était plus judicieux en vue de la quantité de données qui seront extraites à partir de chaque partition.

De T deb à Tdeb +100, les partitions ne seront pas évidemment de 100 jours, la période est moindre. Une fois le cap des 100 jours atteint à Tdeb+100, on se fixe cette taille de fenêtre.

On se retrouve ainsi avec un fichier comportant une série de partitions où chaque partition comporte une série de régimes avec période, volatilité journalière et return journaliers.

Soumettre un signal de 200 jours à une partition globale, revient ainsi à effectuer 195 partitions.

L'opération globale de partition peut donc être effectuée selon de nombreuses valeurs (index, currency, interest rate, eco indicators).

2.3 Fabrication des éléments d'entrée des algorithmes

Les fonctions de modélisations possèdent des entrées qui ont des paramètres rigides, en terme de taille et de contenu.

Deux entrées doivent être renseignées : une matrice appelée matrice prédicteurs et un vecteur Target.

(la dimension des colonnes et du vecteur Target doivent être exactement les mêmes).

Le nombre de colonnes de la matrice représentent le nombre de prédicteurs pour créer un modèle à partir du vecteur Target. Le vecteur Target est la vectrice observation.

En fonction des valeurs prises par le vecteur Target, il faut établir des relations statistiques avec les éléments de la matrice prédicteurs.

Une des étapes est de construire cette matrice prédicteurs et ce vecteur Target. **C'est la nature des vecteurs colonnes qui est essentielle. A un vecteur Target donné, il faut anticiper pour savoir**

quelles informations stockées dans la matrice des prédicteurs est susceptible de contribuer à un apprentissage statistique cohérent.

Un des principes fondamentaux lors de la construction de la matrice de prédicteurs est que chaque ligne est composée d'information de la même date. C'est fondamental pour la construction du modèle. **Par exemple une ligne i de la matrice de prédicteurs comportera le return du titre à la date Ti sa volatilité à la date Ti son monthly return à la date Ti qui représente la valeur VTi/VTi-22.**

On est confronté à un problème: certaines données existent à la date t d'autres non, à cause de la nature de certains titres ou tout simplement à cause de problèmes de données manquantes. Dans un premier temps certains endroits de la matrice comporteront des absences de valeurs. Ce cas sera traité par la suite.

Une de mes préoccupations fut la gestion de la matrice de prédicteurs pour chaque titre : quelle donnée doit on incorporé dans cette matrice sous forme de vecteurs colonnes. Il faut savoir incorporer les return nécessaires et les volatilités calculées de manière judicieuse.

Les algorithmes de machine Learning n'ont pas cette notion temporelle il faut donc s'assurer que chaque ligne de la matrice corresponde à l'élément du vecteur colonne Target. Un décalage fausse totalement l'apprentissage statistique.

Un des problèmes majeurs est la cohérence entre les dates : le signal de base est associé à une série de date. Ma démarche fut la suivante et je pense que c'est la meilleure démarche intellectuelle pour s'y retrouver, **on s'occupe des Stocks choisi un par un. Une fois le titre sélectionné, on choisit le nombre et la nature des options à partir desquelles on crée des vecteurs prédicteurs de ce titre.**

2.3.1 CONSTRUCTION DE LA MATRICE DE PREDICTEURS

2.3.1.1) A l'échelle d'un stock

2.3.1.1.1)option qui n'utilise pas de segmentation

Option last Price

Les séries brutes de chaque valeur datées sont stockées.

On extrait les prix last Price à la période temporelle désirée, on stocke le vecteur des valeurs ainsi créé et le vecteur date associé à ces valeurs

$$\begin{bmatrix} P(1) \\ P(2) \\ \dots \\ P(t) \\ \dots \\ P(end) \end{bmatrix}$$

Option Daily change

On extrait les prix last Price à la période temporelle désirée, on calcule la différence de cours entre chaque jour. On obtient donc un vecteur de valeur et un vecteur date avec une unité en moins en dimension que le vecteur de prix extrait. On stocke le vecteur des valeurs ainsi créé et le vecteur date associé à ces valeurs.

$$\begin{bmatrix} P(2) - P(1) \\ P(3) - P(2) \\ \dots \\ P(t) - P(t-1) \\ \dots \\ P(end) - P(end-1) \end{bmatrix}$$

Option lag change :

On extrait les prix last Price à la période temporelle désirée, on calcule la différence de cours entre chaque période déterminée par le lag. On stocke le vecteur des valeurs ainsi créé et le vecteur date associé à ces valeurs (qui est plus court).

Option monthly return

Il s'agit ici de calculer le return sur 22 jours soit un mois boursier, on extrait les prix last Price à la période temporelle désirée. On ne se sert pas de fichier partition, seule la série pure de prix est traitée. Pour les 22 premiers jours de la série il est compliquer de calculer un return mensuel ne disposant pas de 22 jours.

On utilise l'approximation qui consiste à calculer le return de chacun de ces jours depuis le jour zéro puis de l'élever à la puissance 22/n où n est la distance entre le jour considéré et le jour zéro. Apres cette période de 22 jours on calcule le rapport mensuel de manière classique en utilisant le rapport V_{i+22}/V_i . On obtient donc un vecteur de valeur et un vecteur date avec une unité en dimension que le vecteur de prix extrait.

$$\begin{bmatrix} P(22)/P(1) \\ P(23)/P(2) \\ \dots \\ P(t)/P(t - 21) \\ \dots \\ P(end)/P(end - 21) \end{bmatrix}$$

RETURN SANS PARTITIONS

J'ai exploité plusieurs types de données qui peuvent être intégrées en tant que prédicteur. Il y a les returns de prix. Un des paramètres important est la durée du return. On peut effectivement l'établir sur plusieurs jours ou un seul jour. J'ai donc mis en option la possibilité de rentrer le return jusqu'à 25 jours d'intervalle.

Ainsi dans une ligne de matrice des prédicteurs correspondant à l'instant t. L'utilisateur peut y incorporer les return suivants.

$\left(\frac{s_t}{s_{t-25}} - 1\right) * 100 \quad \dots \left(\frac{s_t}{s_{t-20}} - 1\right) * 100 \quad \dots \left(\frac{s_t}{s_{t-15}} - 1\right) * 100 \quad \dots \left(\frac{s_t}{s_{t-10}} - 1\right) * 100 \quad \dots$
 $\left(\frac{s_t}{s_{t-5}} - 1\right) * 100 \quad$ jusqu'au daily return. Il faut absolument que la date du numérateur soit la même pour tous les return, pour ainsi figurer dans la même ligne de la matrice des prédicteurs.

Une fois la série extraite en fonction de l'intervalle de temps désiré. On rencontre un problème puisqu'on ne peut pas calculer des return sur plusieurs jours dès le premier jour, cela donne lieu à une matrice qui commence par des valeurs qui ne comportent pas de valeurs réelles à certains endroits.

Une question évidente est de se demander pourquoi par exemple si je mets le return sur trois jours et un jour quel est l'intérêt d'incorporer celui sur deux jours. Cela est dû au fait que chaque prédicteur traduit une évolution quelle qu'elle soit.

Une des notions clés est de savoir quel prédicteur peut s'avérer utile en termes d'apprentissage statistique. Il ne faut pas surcharger la matrice et s'assurer que chaque vecteur colonne apporte réellement une information. Les options de calcul de prédicteurs se comportent comme un espace vectoriel. Il faut seulement incorporer les prédicteurs qui agissent comme des vecteurs bases. Une option est inefficace lorsqu'elle apporte des informations au point de vue de l'apprentissage statistique redondantes avec les options déjà présentes .Nous verrons après comment classer les prédicteurs en termes d'efficacité et en terme de contribution au modèle.

	SHARP	SHARP
734688	nan	nan
734689	inf	inf
734690	0.138	-2.864
734691	9.854	4.593
734692	3.774	6.847
734695	7.312	-1.225
734696	9.466	-3.152
734697	9.909	-1.923
734698	13.24	-5.401
734699	10.871	-5.113
734702	8.145	-8.451
734703	3.227	-4.237
734704	4.445	-3.721
734705	4.17	-2.832
734706	3.612	-1.493
734709	3.384	-1.691
734710	2.049	-2.834
734711	1.517	-3.906
734712	1.762	-3.757
734713	1.033	-5.145
734716	0.175	-6.688
734717	0.077	0.46

Figure 41: exemple de matrice de deux éléments avec ratio de sharp daily

RESULT	
Germany Sell 2Y & Buy 10Y Bond STXE 600 Retail € Pr	
VOL	VOL
nan	nan
0.0	0.0
1.124	0.448
1.793	0.504
1.918	0.44
1.937	0.613
1.769	0.571
1.835	0.533
1.737	0.617
1.829	0.593
1.926	0.661
1.925	0.641
1.888	0.636
1.848	0.643
1.828	0.669
1.784	0.647
1.835	0.647
1.804	0.642
1.753	0.628
1.737	0.637
---	---

Figure 42 volatilité dans ce cas

VOLATILITE SANS PARTITIONS :

Une seconde partie des prédicteurs sont ceux de la gamme des volatilités. Tous mes calculs de volatilité ont été effectués sur 25 données. Je pense que c'est suffisant pour calculer réellement une volatilité. Il faut maintenant statuer sur quel type de Return, il faut effectuer les calculs de volatilité. On calcule dans un premier temps les return avec le lag de temps souhaité, on stocke ces valeurs et on calcule la volatilité sur 25 jours. Pour ne pas raccourcir trop la série des volatilités au début, pour les premiers return des séries de return stocké, j'ai calculé la volatilité sur moins de jours. Les valeurs des volatilités dépendent du type de return sur lesquels elles sont calculées, un bon réflexe est d'annualiser la volatilité pour avoir une juste appréciation de la valeur de celle –ci.

Une troisième partie des prédicteurs est la catégorie des ratios de Sharp. On fixe aussi le lag de jours utilisé pour calculer les return. On calcule par la suite les volatilités associées et on réalise le rapport entre les returns et les volatilités.

On peut se demander quel est l'intérêt d'incorporer des Sharp à partir du moment où on a déjà incorporé des données de volatilité et de return. Mais il semble que le calcul des Sharp apporte de

l'information statistique. En effet étudier les volatilités et les returns individuellement ne permet pas de rendre compte de la corrélation entre les deux.

Une option envisagée était d'incorporer des prédicteurs à partir du volume d'échanges des titres. En effet, les volumes d'échanges sont des données essentielles pour traduire le comportement d'un titre.

Ainsi à partir d'un seul titre on peut déjà créer 75 vectrices colonnes :

-25 vecteurs colonnes représentant chacun les returns d'un lag allant de 1 à 25 jours pour toute la durée de la série

-25 vecteurs colonnes représentant chacun les volatilités des returns de lag allant de 1 à 25 jours pour toute la durée de la série

-25 vecteurs colonnes représentant chacun les sharps des returns et volatilité des lag allant de 1 à 25 jours pour toute la durée de la série

A chaque ligne de la matrice qui symbolise un jour dans le temps est associée toutes ses caractéristiques dont le calcul de certaines peut remonter à des périodes de 50 jours précédents cette date (cas du calcul de volatilité des returns 25 jours sur 25 jours).

On réalise ce protocole pour tous les titres que l'on souhaite incorporer dans la construction du modèle.

debut period	2011-07-24
fin period	2013-06-23
HORIZON	25
PREDICTOR ALL RETURN TYPE	
LAG MAX	10
BUILD VOL LAG MAX	SAVE VOL LAGn MAX

Figure 43: tableau permettant de contrôler le lag maximal des returns et volatilités

result																						
0.0	nan	0.0	nan	nan	nan	nan																
1.124	0.0	nan	0.448	0.0	nan	nan	nan	nan														
1.793	1.059	0.0	nan	0.504	0.143	0.0	nan	nan	nan	nan												
1.918	0.919	1.347	0.0	nan	nan	nan	nan	nan	nan	0.44	0.473	0.036	0.0	nan	nan	nan	nan	nan	nan	nan	nan	nan
1.937	0.798	1.623	0.786	0.0	nan	nan	nan	nan	nan	0.613	0.592	0.233	0.675	0.0	nan	nan	nan	nan	nan	nan	nan	nan
1.769	1.135	1.456	1.436	0.203	0.0	nan	nan	nan	nan	0.571	0.763	0.605	0.597	0.353	0.0	nan	nan	nan	nan	nan	nan	nan
1.835	1.346	2.139	1.462	2.069	1.094	0.0	nan	nan	nan	0.533	0.697	0.717	0.641	0.316	0.132	0.0	nan	nan	nan	nan	nan	nan
1.737	1.682	2.193	2.417	1.918	2.296	0.47	0.0	nan	nan	0.617	0.709	0.778	1.016	0.752	0.376	0.759	0.0	nan	nan	nan	nan	nan
1.829	1.687	2.031	2.207	1.808	2.152	0.406	1.324	0.0	nan	0.593	0.698	0.737	0.949	0.86	0.538	0.62	0.099	0.0	nan	nan	nan	nan
1.926	2.312	2.509	2.257	2.113	2.043	2.262	1.448	1.572	0.0	0.661	0.696	0.905	1.01	0.943	1.043	1.012	0.42	0.825	0.0	0.0	0.0	0.0
1.925	2.612	3.34	3.084	2.623	2.756	2.486	2.918	1.52	1.273	0.641	0.691	0.858	1.023	0.908	0.995	1.118	0.595	0.679	0.114	0.0	0.0	0.0
1.888	2.494	3.333	3.451	3.043	2.79	2.628	2.633	1.683	1.429	0.636	0.728	0.814	0.976	0.857	0.921	1.021	0.61	0.613	0.37	0.0	0.0	0.0
1.848	2.388	3.227	3.555	3.683	3.408	2.876	2.858	1.547	1.709	0.643	0.799	0.941	0.958	0.923	0.878	0.996	0.654	0.548	0.37	0.0	0.0	0.0
1.828	2.42	3.12	3.53	3.971	4.228	3.77	3.226	2.209	1.649	0.669	0.902	1.121	1.242	1.051	1.13	1.056	0.91	0.828	0.476	0.0	0.0	0.0
1.784	2.423	3.178	3.443	3.995	4.535	4.628	4.061	2.713	2.263	0.647	0.906	1.173	1.367	1.323	1.218	1.249	0.982	0.978	0.729	0.0	0.0	0.0
1.835	2.498	3.337	3.709	4.06	4.737	5.191	5.195	4.071	3.143	0.647	0.891	1.133	1.347	1.365	1.34	1.241	1.072	0.939	0.748	0.0	0.0	0.0
1.804	2.58	3.434	3.909	4.374	4.815	5.406	5.717	5.214	4.37	0.642	0.911	1.137	1.299	1.323	1.332	1.286	1.038	0.946	0.699	0.0	0.0	0.0
1.753	2.514	3.436	3.932	4.483	4.987	5.381	5.779	5.56	5.201	0.628	0.886	1.123	1.274	1.275	1.306	1.31	1.16	0.94	0.781	0.0	0.0	0.0
1.737	2.459	3.384	3.981	4.56	5.137	5.595	5.782	5.684	5.619	0.637	0.868	1.117	1.297	1.294	1.267	1.26	1.135	0.963	0.741	0.0	0.0	0.0
1.727	2.482	3.35	3.966	4.666	5.258	5.79	6.038	5.747	5.806	0.657	0.931	1.135	1.339	1.392	1.373	1.275	1.107	0.923	0.711	0.0	0.0	0.0
1.688	2.457	3.348	3.918	4.633	5.322	5.864	6.172	5.961	5.825	0.733	1.069	1.336	1.472	1.577	1.652	1.591	1.324	1.084	0.833	0.0	0.0	0.0
1.651	2.397	3.283	3.868	4.543	5.232	5.847	6.157	6.01	5.929	0.746	1.184	1.514	1.719	1.77	1.901	1.948	1.765	1.458	1.171	0.0	0.0	0.0
2.221	2.863	3.57	3.982	4.533	5.207	5.747	5.978	5.821	5.717	0.851	1.324	1.781	2.062	2.205	2.277	2.397	2.371	2.201	1.879	0.0	0.0	0.0
2.731	4.367	4.942	5.228	5.455	5.87	6.414	6.548	6.215	6.111	0.834	1.378	1.868	2.255	2.468	2.616	2.689	2.734	2.699	2.481	0.0	0.0	0.0
2.757	4.375	5.465	5.781	5.954	6.198	6.598	6.781	6.434	6.182	0.954	1.451	2.064	2.489	2.83	3.061	3.22	3.228	3.285	3.231	0.0	0.0	0.0
2.803	4.289	5.551	6.492	6.71	6.908	7.122	7.193	6.988	6.712	1.002	1.429	2.028	2.506	2.873	3.187	3.399	3.469	3.496	3.498	0.0	0.0	0.0
2.79	4.35	5.439	6.597	7.437	7.688	7.865	7.81	7.546	7.411	1.051	1.418	2.038	2.501	2.947	3.295	3.603	3.739	3.836	3.811	0.0	0.0	0.0
2.76	4.382	5.542	6.475	7.581	8.473	8.71	8.673	8.336	8.123	1.176	1.42	2.085	2.453	2.887	3.238	3.555	3.738	3.87	3.885	0.0	0.0	0.0
2.74	4.382	5.538	6.518	7.428	8.484	9.232	9.276	8.991	8.692	1.398	1.932	2.29	2.759	2.959	3.284	3.506	3.663	3.777	3.792	0.0	0.0	0.0
2.756	4.455	5.543	6.523	7.429	8.326	9.119	9.549	9.349	9.074	1.393	2.09	2.667	2.94	3.248	3.339	3.544	3.615	3.703	3.7	0.0	0.0	0.0
2.788	4.528	5.676	6.572	7.463	8.325	8.98	9.38	9.484	9.272	1.402	2.095	2.834	3.315	3.456	3.665	3.629	3.691	3.679	3.647	0.0	0.0	0.0
nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	1.428	2.14	2.872	3.541	3.904	3.945	4.047	3.857	3.841	3.687	0.0	0.0	0.0
2.978	4.764	5.957	6.967	7.679	8.513	9.056	9.424	9.200	9.133	1.603	2.10	2.888	3.546	3.921	4.076	4.105	3.987	3.805	3.652	0.0	0.0	0.0

Figure 44: matrice de prédicteurs on voit ici le décalage au début de certains vecteurs colonne dû au fait que j'ai incorporé des return avec un lag supérieur que 1 jour



Figure 45: interface pour gérer la création de la matrice

2.3.1.1.2) options avec partitions

Option Daily return partition

Il faut impérativement ici réaliser une partition totale du signal au préalable.

Le fichier partition totale comprend un ensemble de partitions où chaque partition est référencée par une date (la date du dernier jour de partition). Pour chaque jour de la période on accède à la partition de cette date, au dernier régime de celle-ci. Au sein de ce dernier régime on récupère la valeur du return de ce dernier régime. On réalise ce procédé pour chaque jour de la période, le vecteur temps et le vecteur des valeurs sont ensuite stockés.

$$\left[\begin{array}{l} \text{partition}(1).\text{lastreg.return} \\ \text{partition}(2).\text{lastreg.return} \\ \dots \\ \text{partition}(t).\text{lastreg.return} \\ \dots \\ \text{partition}(end).\text{lastreg.return} \end{array} \right]$$

Option daily volatility partition

Il faut aussi ici impérativement réaliser une partition totale du signal avant. Sinon on ne possède aucun moyen de calculer la volatilité journalière selon ce procédé. Une fois le fichier partition totale créé, pour chaque jour de la période, on accède à la partition de cette date, au dernier régime de celle-ci. Au sein de ce dernier régime on récupère la valeur de la volatilité journalière de ce dernier régime. On réalise ce processus pour tous les fichiers de partition présents dans la partition totale en les associant à leur date d'extraction. Dans aucun cas il ne faut dissocier la partition avec sa date, c'est le principal fil conducteur de la programmation. Un décalage est problématique.

$$\begin{bmatrix} \text{partition}(1).\text{lastreg.volatility} \\ \text{partition}(2).\text{lastreg.volatility} \\ \dots \\ \text{partition}(t).\text{lastreg.volatility} \\ \dots \\ \text{partition}(\text{end}).\text{lastreg.volatility} \end{bmatrix}$$

Option daily sharp :

A partir du fichier de partition globale. On extrait à chaque date la volatilité et le return du dernier régime. On calcule le rapport entre les deux qui constitue le ratio de Sharp. On stocke ainsi le vecteur des valeurs et le vecteur des dates correspondants.

$$\begin{bmatrix} \text{partition}(1).\text{lastreg.return}/\text{partition}(1).\text{lastreg.volatility} \\ \text{partition}(1).\text{lastreg.return}/\text{partition}(2).\text{lastreg.volatility} \\ \dots \\ \text{partition}(1).\text{lastreg.return}/\text{partition}(t).\text{lastreg.volatility} \\ \dots \\ \text{partition}(1).\text{lastreg.return}/\text{partition}(\text{end}).\text{lastreg.volatility} \end{bmatrix}$$

Option volatility avec lag :**Cas numero 1 :**

On fixe un lag de T. A partir du fichier de partition globale on se situe à la date qui correspond à une partition d'au moins T jours d'historiques. On extrait le fichier partition de la date t considérée.

Soit un lag L. On isole cette partition.

On extrait le dernier régime de cette partition. Il y a deux cas :

-ce régime comporte au moins L dates, dans ce cas on récupère la volatilité de ce régime. On multiplie celle-ci par \sqrt{L} , afin de passer d'une volatilité journalière à une volatilité sur T jours.

- ce régime comporte moins de date que L, à ce moment-là on s'intéresse à l'avant dernier régime et ainsi de suite jusqu'à avoir le nombre de dates souhaitées. On a donc relevé plusieurs volatilités. On calcule la moyenne de ces volatilités en pondérant chaque volatilité par son occurrence. On obtient une volatilité moyenne créée à partir d'une seule partition. On multiplie celle-ci par \sqrt{L} , afin de passer d'une volatilité journalière à une volatilité sur T jours.

Exemple : on considère une partition associée à la date t dans le fichier de partition globale.

régime	Date début régime	Date fin régime	Return	volatilité	
end	40	48	X	V	
End-1	33	39	X1	V1	
End -2	26	32	X2	V2	
End -3					
End-4					

Imaginons qu'on souhaite calculer une volatilité sur 20 jours.

$$\text{ON aura } V_{moy} = \frac{V*9+V1*7+V2*4}{20} \quad \text{on aura } V_{20j} = \sqrt{20} * V_{moy}$$

Cas numéro 2 : A l' instant on extrait les L partitions de partition(t) à partition (t-L).

Pour chaque partition on extrait la volatilité du dernier régime. Une fois toutes ces volatilités extraites on calcule la moyenne.

Exemple : on considère qu'on souhaite calculer la volatilité sur 20 jours à l'instant t.

$$\text{On a } V_{20J} = \sum_{i=0}^{20} \frac{\text{partition}(t-i).reg(end).volatility}{20}$$

Option return avec lag :

Cas numero 1 :

On fixe un lag de T. A partir du fichier de partition globale on se situe à la date qui correspond à une partition d'au moins T jours historiques. On extrait le fichier partition de la date t considérée. Soit un lag L. On isole cette partition.

On extrait le dernier régime de cette partition. Il y a deux cas :

- ce régime comporte au moins L dates, dans ce cas on récupère le return de ce régime.

$$\text{On a donc } R_{20J} = (1 + R_{reg})^{20} - 1$$

- ce régime comporte moins de date que L, à ce moment-là on s'intéresse à l'avant dernier régime et ainsi de suite jusqu'à avoir le nombre de dates souhaitées. On a donc relevé plusieurs returns. On calcule similairement le produit de tous les returns pondérés par leur occurrence :

Exemple : on considère une partition associée à la date t dans le fichier de partition globale.

régime	Date début régime	Date fin régime	Return	volatilité	
end	40	48	X	V	
End-1	33	39	X1	V1	
End -2	26	32	X2	V2	
End -3					
End-4					

On suppose qu'on veut calculer le return de 20 jours associés à cette date t

$$\text{On a } R_{20j} = (1 + X)^9 * (1 + X1)^7 * (1 + X2)^4$$

Cas numéro 2 :

A l'instant on extrait les L partitions de partition(t) à partition (t-L).

Pour chaque partition on extrait le return du dernier régime. Une fois toutes ces returns extraits, on calcule le produit cumulé.

Exemple : on considère qu'on souhaite calculer la volatilité sur 20 jours à l'instant t.

$$\text{On a } R_{20j} = \prod_{i=0}^{20} (1 + \text{partition}(t - i).reg(end).return)$$

Option daily volatility moving average:

Il n'est pas ici nécessaire de réaliser une partition totale

Option sharp avec lag :

On calcule le ratio de Sharp en calculant au préalable, la volatilité et le return avec le lag désiré. On réalise ensuite le rapport entre les deux.

Option variance causé par la partition :

On crée un vecteur de valeurs de la façon suivante :

A l'instant t considéré : on extrait les 20 partitions simples situées entre t-20 et t

A l'instant t-20 on distingue de 20 valeurs différentes de returns en se servant des 20 partitions qui couvrent cette date.

A l'instant t-19 on dispose de 19 valeurs différentes de returns en se servant des 19 partitions qui couvrent cette date.

....

A l'instant t-5 on dispose de 5 valeurs différentes de returns en se servant des 5 partitions qui couvrent cette date.

A l'instant t-1 on dispose de 2 valeurs différentes de returns en se servant des 2 partitions qui couvrent cette date.

A l'instant t on dispose d'une valeur différente de return en se servant.

Pour chacun de ces groupes de returns prélevés correspondant à chaque date t. On calcule la variance de chaque groupe. Une fois les 20 variances calculées on calcule une moyenne de ces 20 variances, en les pondérant proportionnellement aux nombres d'éléments présents dans le groupe.

Cela traduit l'évolution de l'action de partitionnement sur le signal.

$$\sum_{i=0}^{20} \frac{\text{var}(R_{part\ i\ t} - i)}{\sum_{i=0}^{20} i}$$

Une fois qu'on a calculé les vecteurs de valeur des différentes options qu'on a choisies, il faut à partir de ces vecteurs créer une unique matrice qui représente tous les prédicteurs choisis pour ce titre. Lors de cette concaténation. Certaines valeurs non réelles se formeront au début de la matrice dû à des lags différents entre prédicteurs.

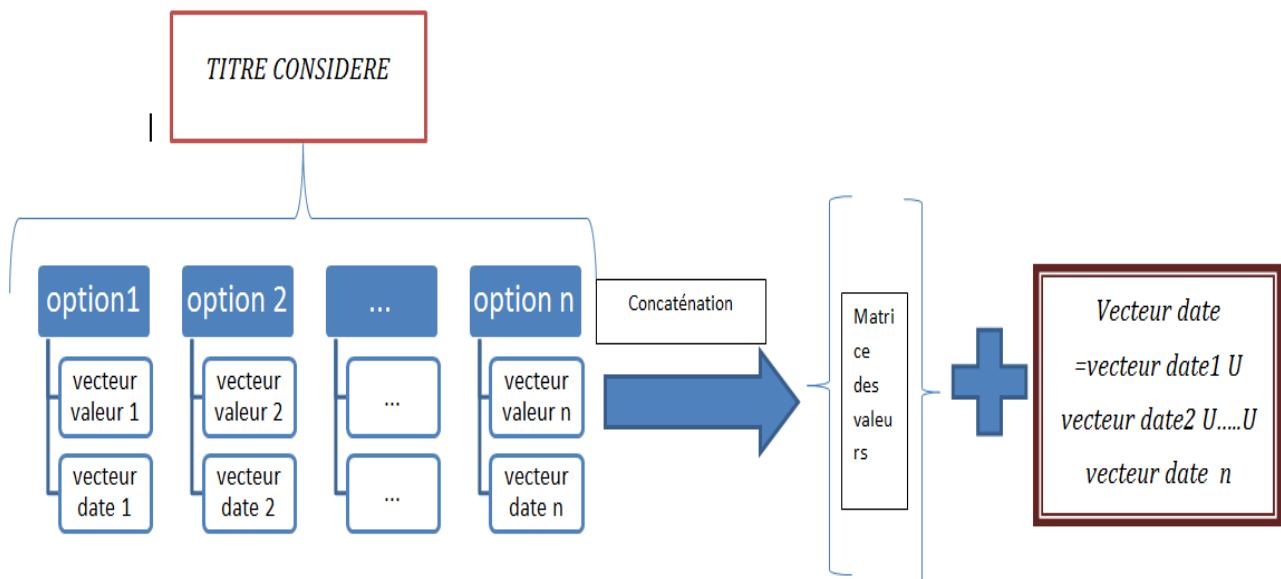


Figure 46 processus à échelle d'un titre avec les options choisies

2.3.1.2)concaténation des matrices formations matrice prédicteur

Il faut maintenant effectuer la concaténation de ces matrices.

A chaque mini matrice des N prédicteurs est associé un vecteur temps qui lui est unique. Le vecteur temps est de la taille de la taille des colonnes de la matrice et comprend l'ensemble des dates journalières qui pour lesquels sont calculés des valeurs des données de prédicteurs.

Le principe que j'ai mis est le suivant on implémente une opération qui concatène deux matrice selon les colonnes.

Soit X1 et T1 la première matrice et son vecteur temps associé soit X2 et T2 la deuxième matrice et son vecteur temps associé :

On réalise l'union des vecteurs temps T1 et T2 en termes de contenu on se retrouve avec un vecteur temps Tmax qui est forcément plus grand que T 1 ou T2.

Une fois ce vecteur T max créée. On crée une nouvelle matrice vide de longueur de la taille de T max et dont la largeur est la somme des largeurs des matrices X1 et X2

On parcourt les données une par une. A chaque date t de ce vecteur Tmax, on regarde si la matrice X1 possède des valeurs associées à ce temps t :

Si c'est le cas, on remplit la partie gauche (jusqu'à la colonne de numéro la largeur de X1) de la nouvelle matrice avec les valeurs de X1 associées à ce temps t.

Si ce n'est pas le cas on remplit la partie gauche de la nouvelle matrice avec des NaN (not a number en python)

On réalise la même chose avec X2 mais en mettant les éventuelles valeurs dans la partie gauche de la nouvelle matrice.

On obtient donc une matrice finale de longueur de la taille de T max et dont la largeur est la somme des largeurs des matrices X1 et X2 comportant quelque valeur de type Nan dû à la fusion des matrice X1 et X2.

Une fois cette matrice créée on réalise ce procédé avec celle-ci et X3. Ainsi à chaque la matrice finale gagne en longueur et en largeur. On réalise ce processus autant de fois que l'on dispose de matrice au départ, On obtient aussi un vecteur temps final qui est l'union de tous les vecteurs temps du début.

Une question essentielle est de se demander pourquoi à chaque étape avoir réalisé l'union des vecteurs temps plutôt que l'intersection des vecteurs temps pour à la fin avoir une matrice pleine uniquement de valeur sans NaN.

Cela est dû au fait que les cases disposant de NaN seront traitées après et la valeur manquante sera remplacée par une valeur exploitable et traduisant le comportement moyen de ce prédicteur dans la zone temporelle considérée.

On a donc ainsi créé une matrice de prédicteur.

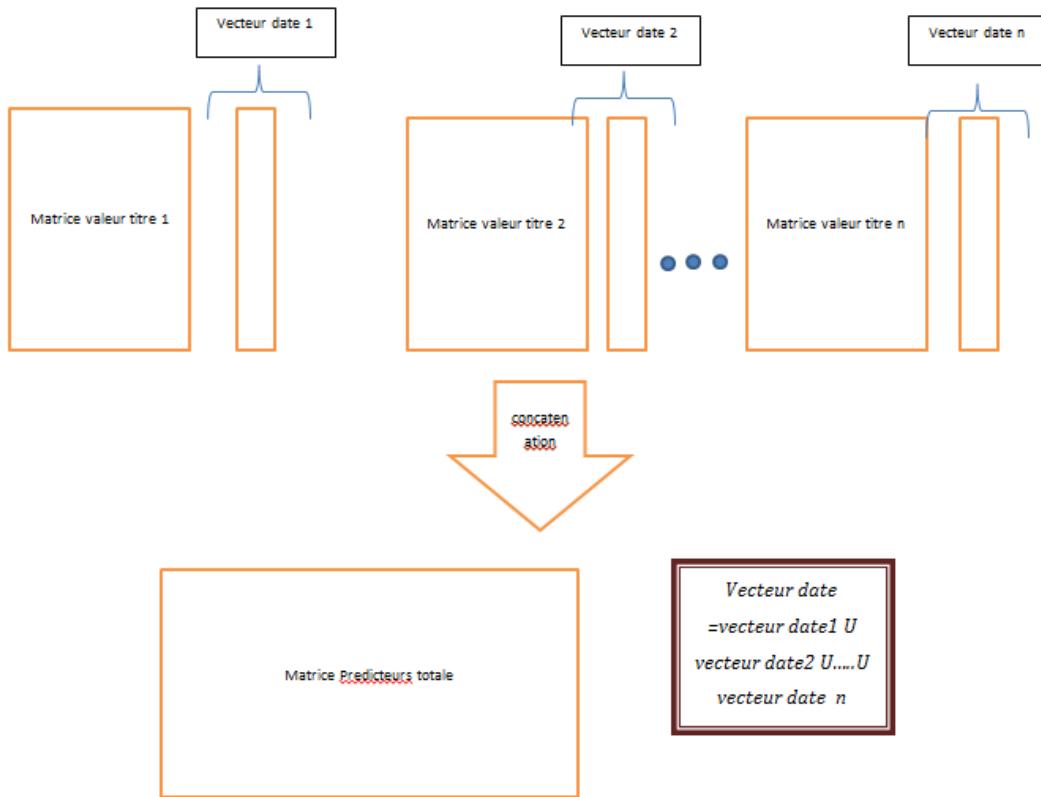


Figure 47: processus de création de la matrice finale de prédicteurs

2.3.2)vecteur Target

Il faut savoir que la majorité des algorithmes de machine Learning utilise des données Target binaires associées à des numéros de classe, il faut donc choisir des données qui peuvent être binarisées, les returns et les sharps sont donc idéals.

L'horizon du return choisi correspond à la période de prédiction. On peut certes servir un lag de return différent que le vecteur horizon, mais cela est étrange de réaliser des prédictions à une date T pour des returns de lag inférieur à T. Le return n'est pas encré dans le présent, cependant certaines options le proposent.

Le vecteur Target est lui associé à un vecteur colonne et à un vecteur temps, comportant les dates ou sont calculée la forme de sortie désirée (return avec un lag correspondant à l'horizon de la prédiction, ou sharp).

Un des paramètres caractéristique de la construction est la donnée horizon : elle représente le nombre de jours de décalage pour une même ligne entre la date de la valeur du vecteur Target et la date des valeurs de la ligne dans la matrice prédicteurs ;

L'unité du paramètre horizon est le jour.

L'utilisateur doit choisir une seule option pour la construction du vecteur Target. Voici les différentes options :

Option performance :

On calcule ici le return horizon. Il faut donc faire une extraction des valeurs du titre pour la période donnée et calculer ces rapports. On ne réalise ici pas de simplification il faut attendre une durée de horizons jours pour avoir la première donnée. Le vecteur de sortie est donc plus court que celui d'origine de horizon valeurs.

Option spread :

On calcule la performance horizon du titre comme on la fait précédemment. Chaque valeur de ce nouveau vecteur est comparée à la performance horizon moyenne d'un ensemble de titres qui constituent un portefeuille choisi par l'utilisateur.

Option segment :

Cette option requiert une partition totale du titre au préalable. A partir de la période choisie pour chaque date de celle-ci. On va s'intéresser à la date $t+horizon$. On récupère la partition de la date $t+horizon$, le dernier régime de celle-ci. On récupère le return de celui-ci.

On réitère le procédé pour toute la période. On obtient donc un vecteur plus court d'horizon valeurs, avec un décalage temporel.

Option sharp ratio :

A partir de la période choisie pour chaque date de celle-ci, on va s'intéresser à la date $t+horizon$. On récupère la trend et la volatilité de la même manière que précédemment et on effectue la division entre les deux. On obtient la aussi un vecteur plus court avec un décalage temporel de horizon.

Option Sharp ratio avec fenêtre contrainte :

Il s'agit ici de calculer un ratio de Sharp pour une durée de horizon jours. Il faut donc calculer le return sur horizon jours et la volatilité sur horizon jours.

Pour calculer le return on extrait les valeurs brutes et on calcule le rapport sur une durée de horizons jours. On a donc réduit le vecteur de horizon données.

Il faut maintenant calculer la volatilité sur horizons jours. On va calculer cela à partir des volatilités journalières pendant horizons jours.

On se considère à une date $t+horizon$. On va donc récupérer les volatilités des jours suivants

$\{t+hor ; t+hor-1 ; t+hor-2 ; t+hor-3 ; \dots ; t+1\}$. Deux possibilités viennent à l'esprit :

On utilise seulement la partition du jour $t+hor$, on prélève les volatilités à partir des régimes qui couvrent les dates précédentes (cette période peut s'étendre sur un ou deux régimes). On voit ici lorsqu'on stocke les partitions de garder tous les régimes même si le dernier régime est souvent le seul utilisé. On calcule ensuite la moyenne des volatilités. On multiplie cette valeur par $(horizon)^{1/2}$.

Pour passer sur une volatilité d'horizon jours.

On utilise pour chaque jour évoqué précédemment sa partition. On va donc utiliser horizons partitions. Pour chaque partition on prélève la volatilité du dernier régime. Une fois toutes les volatilités collectées, on calcule la moyenne. On multiplie cette valeur par $(horizon)^{1/2}$. Pour passer sur une volatilité d'horizon jours.

On privilégie la solution une car étant à l'instant $t+hor$ au moment du calcul, il ne serait pas judicieux de ne pas utiliser des informations du futur pour calculer des volatilités à l'instant $t+hor-5$ par exemple. Cette méthode est donc plus précise et plus révélatrice. Cependant la solution 2 peut être utilisée. Il faut cependant spécifié à l'utilisateur, une fois le modèle construit qu'il faudra calculer ces volatilités de la même manière. (en réalisant une cascade de partitions)

2.3.3) réunification

Une fois le vecteur Target et la matrice construite, il faut s'assurer de la cohérence temporelle entre les deux.

On crée ensuite un vecteur temps qui correspond à l'intersection des dates du vecteur de la matrice des prédicteurs et de celui du vecteur temps.

On supprime chaque ligne de la matrice des prédicteurs qui correspondent à une date qui n'est pas présente dans ce vecteur temps intersection. On obtient ainsi une matrice de prédicteur dont la dimension en longueur est la même que celle du vecteur Target.

Une fois cette étape il va falloir redimensionner la matrice et le vecteur target afin de réellement réaliser un apprentissage ayant pour but de prédire dans le futur.

On se réfère au paramètre horizon qui correspond au nombre de jour ou on veut réaliser la prédiction. Ainsi on enlève les horizons premières valeurs du vecteur Target et les horizons dernières ligne de la matrice des prédicteurs. Cela est nécessaire pour réaliser l'apprentissage dans le futur. Si on ne réalise pas ce décalage on construit un modèle qui servira à prédire une valeur journalière à partir d'autres valeurs journalières du même jour, ce qui a seulement un intérêt pour calculer la corrélation entre certaines valeurs mais n'a aucune valeur prédictive. Il faut savoir que les algorithmes de machine Learning sont rarement associés à des notions temporelles. Ils servent à statuer sur l'importance ou non d'une chose à un groupe donné et sont utilisés en biologie ou dans les domaines de la santé et de la publicité.

Il a fallu gérer la gestion des valeurs NaN. Un module a été mis en place qui consiste à remplacer la valeur NaN, par la moyenne des trois valeurs voisines de la colonne. Cependant il ne faut pas que les valeurs voisines soient issues de ce processus. On s'intéresse aux valeurs existantes, initialement.

Les modèles de machines learning pour leur construction ne tolère que des vecteurs Target comprenant des valeurs entières (classe). Dans la plupart du temps le vecteur Target comprenant des valeurs négatives et positives dû à la nature des composants (volatilités , return) sera mis sous forme (1,-1)

2.4 LEARNING

Algorithme utilisé :

2.4.1 Algorithme Radom Forest classifier :

Cet algorithme consiste à créer des arbres de décisions statistiques

Paramètres d'entrée de l'algorithme :

Number of estimators : nombre d'arbre à créer constituant la foret

Critère : critère utilisé pour statuer sur l'utilité d'un nœud d'un arbre désigné

Les deux critères principaux sont le critère gini et le critère d'entropy

Max feature split : nombre maximum de catégorie de prédicteur à utiliser pour créer un nœud

(on entend par nœud un ensemble de conditions statistiques vérifié ou non par le vecteur horizontale considéré)

Max depth tree : nombre maximum d'étage d'un arbre

Régression logistique :

Dans le cas d'un modèle de sortie binaire (cas dans prisms), on a :

$$\ln \frac{p(X|1)}{p(X|0)} = a_0 + a_1 * x_1 + a_2 * x_2 + \dots + a_j * x_j$$

2.4.2 Kneight classifier :

Le principe est le suivant :

Pour chaque nouveau point x on commence par déterminer l'ensemble de ses k plus proches voisins, la classe que l'on affecte à ce nouveau point x est alors la classe majoritaire de l'ensemble.

Soit x_i les points de l'ensemble, soit x le point que l'on souhaite classifier on a :

$$r_1(x) = i^* \text{ si et seulement si } d_{i^*} = \min_{1 \leq i \leq n} d_i(x)$$

2.4.3 support vector machine:

C'est un algorithme d'apprentissage construit pour la classification binaire. Il consiste à rechercher une règle de décision basée sur une séparation par hyperplan optimale. Les hyperplans sont solutions de problème d'optimisation sous contraintes (les contraintes sont la qualité de la prédiction et la complexité du modèle). L'objectif est de tracer un hyperplan qui classifie tous les vecteurs de training en 2 classes.

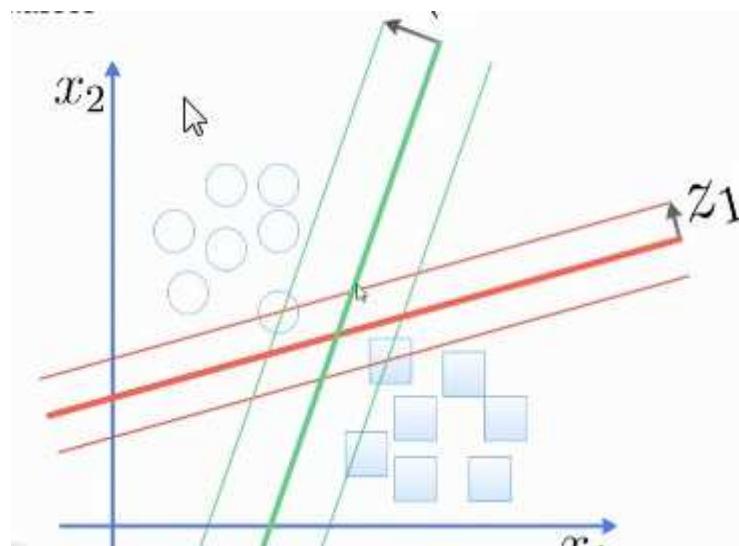


Figure 48 l'hyperplan vert possède une distance séparatrice plus importante il sera donc choisi. Cet hyperplan est le noyau d'une forme linéaire, la partie gauche par exemple représente l'ensemble des points du plan où la valeur de la forme linéaire est inférieure à zéro.

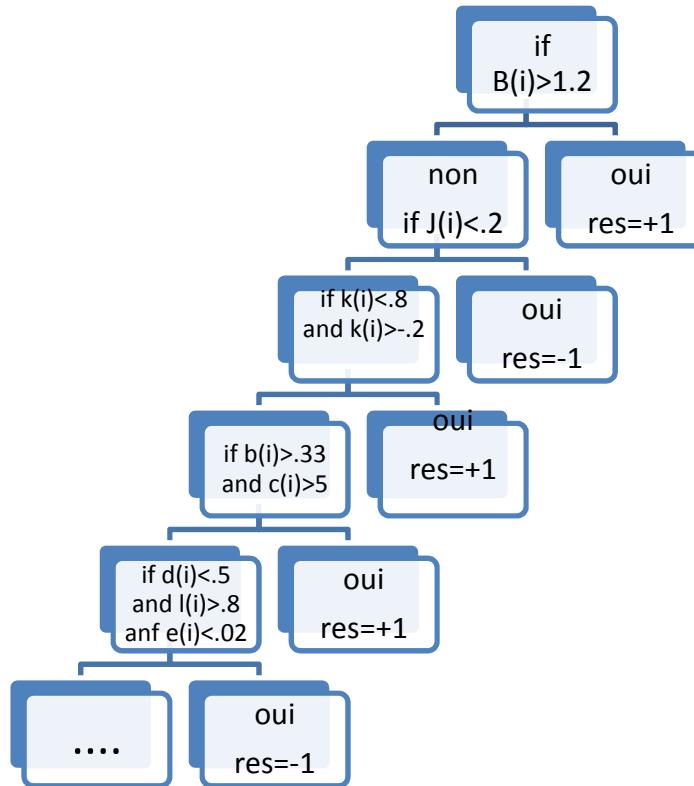
Avantage :

- efficace dans un espace de grandes dimensions
- toujours fonctionnel lorsque le nombre de prédicteurs est supérieur au nombre de dates
- on peut sélectionner la fonction de noyau

Inconvénients :

- quand le nombre de prédicteurs est fortement supérieur au nombre de date ses performances baissent rapidement.

2.4.4 ALGORITHME EXPERIMENTALE



Un arbre statistique peut être visualisé comme une succession chronologique de conditions sur la valeur des coordonnées de la vectrice ligne à partir duquel on veut réaliser une prédiction. **Le respect d'une condition entraîne immédiatement le résultat de la prédiction, un non-respect de la condition entraîne une mise en situation suivante. On soumet le vecteur à d'autres conditions statistiques**

Construction d'arbre :

Level1

On considère une colonne de la matrice. Pour chaque valeur de la matrice $c(i)$;

1) On crée un groupe de vecteur ligne ou pour tout vecteur ligne de ce groupe on a $c(j)<c(i)$ (ou respectivement $c(j)>c(i)$). On s'intéresse ensuite à la valeur du vecteur résultat associé à chacun de ces vecteurs lignes. Si toutes ces valeurs sont identiques (par exemple 1) On a donc déterminer une condition si $c(j)<c(i)$ alors $res = 1$. On retient cette zone et le nombre de vecteur ligne qu'elle possède, si ce n'est pas le cas on ne la retient pas. Une fois ce procédé effectué pour toutes les valeurs de la colonne. On ne retient que la zone la plus grande on a donc trouvé notre première condition.

2) On réduit la matrice prédicteur et le vecteur Target en supprimant les vecteurs lignes qui respectent la condition trouvée en 1). On passe à une colonne suivante....

```

import numpy as np
import random
from sklearn import datasets

def resultuniformplus(matrix,value,result,colon,params):
    bucket,res=[],0
    for i in range(np.size(matrix,0)):
        if (params==0 and matrix[i,colon]<=value) or (params==1 and matrix[i,colon]>=value):
            bucket.append(result[i])
    a=list(set(bucket))
    if len(a)==1 and len(bucket)>=np.floor(np.size(matrix,0)/10) and len(bucket)!=np.size(matrix,0):
        res=1
    return res,len(bucket)

def trouverplusgrand(listoflist):
    bucket=[]
    for i in range(len(listoflist)):
        bucket.append(listoflist[i][0])
    g=sorted(bucket,reverse=True)
    return listoflist[bucket.index(g[0])][1]

iris = datasets.load_iris()
X = iris.data
Y = iris.target

def level1column(X,Y,column,params):
    liste,joe,[],[]
    for i in range(np.size(X,0)):
        gui=resultuniformplus(X,X[i,column],Y,column,params)
        if gui[0]==1:
            liste.append([gui[1],i])
    if liste != []:
        a,bucket=trouverplusgrand(liste),[]
        joe.extend((X[a,column],column,params,Y[a]))
        for i in range(np.size(X,0)):
            if (params==1 and X[i,column]>=X[a,column]) or (params==0 and X[i,column]<=X[a,column]):
                bucket.append(i)
        X=np.delete(X,(bucket),axis=0)
        bucket.sort(reverse=True)
        Y=list(Y)
        for i in range(len(bucket)):
            Y.remove(Y[bucket[i]])
    return X,Y,joe

def level1(X,Y,params):
    bucket,k=[],np.size(X,1)
    l=range(k)
    for i in range(k):
        j=random.choice(l)
        l.remove(j)
        gui=level1column(X,Y,j,params)
        bucket.append(gui[2])
        X,Y=gui[0],gui[1]
        print(np.size(X,0))
    return X,Y,bucket

def laverlistlist(listlist):
    bucket,new,[],[]
    for i in range(len(listlist)):
        if listlist[i]!=[]:
            new.append(listlist[i])
    return new

def realiserlevel1(X,Y):

```

```

b,j=[0,1],0
i=random.choice(b)
if i==0:
    j=1
gui=level1(X,Y,i)
a,X,Y=gui[2],gui[0],gui[1]
gui=level1(X,Y,j)
a.extend(gui[2])
X,Y=gui[0],gui[1]
return X,Y,a

def imputparser2(params1,params2,value1,value2,value2):
res=0
if(params1==1 and params2==0 and value1>=value1 and value2<=value2) or \
   (params1==1 and params2==1 and value1>=value1 and value2>=value2) or \
   (params1==0 and params2==1 and value1<=value1 and value2>=value2) or \
   (params1==0 and params2==0 and value1<=value1 and value2<=value2):
    res=1
return res

def result2uniformplus(matrix,value1,value2,result,colon1,colon2,params1,params2):
bucket,res=[],0
for i in range(np.size(matrix,0)):
    if params1==1 and params2==0 and matrix[i,colon1]>=value1 and
matrix[i,colon2]<=value2:
        bucket.append(result[i])
    if params1==1 and params2==1 and matrix[i,colon1]>=value1 and
matrix[i,colon2]>=value2:
        bucket.append(result[i])
    if params1==0 and params2==1 and matrix[i,colon1]<=value1 and
matrix[i,colon2]>=value2:
        bucket.append(result[i])
    if params1==0 and params2==0 and matrix[i,colon1]<=value1 and
matrix[i,colon2]<=value2:
        bucket.append(result[i])
a=list(set(bucket))
if len(a)==1 and len(bucket)>=np.floor(np.size(matrix,0)/10) and
len(bucket)!=np.size(matrix,0):
    res=1
return res,len(bucket)

def level2column(X,Y,column1,column2,params1,params2):
liste,joe,[],[]
for i in range(np.size(X,0)):

gui=result2uniformplus(X,X[i,column1],X[i,column2],Y,column1,column2,params1,params2)
if gui[0]==1:
    liste.append([gui[1],i])
if liste != []:
    a,bucket=trouverplusgrand(liste),[]
    joe.extend((X[a,column1],column1,X[a,column2],column2,params1,params2,Y[a]))
    for i in range(np.size(X,0)):
        if params1==1 and params2==0 and X[i,column1]>=X[a,column1] and
X[i,column2]<=X[a,column2]:
            bucket.append(i)
        if params1==1 and params2==1 and X[i,column1]>=X[a,column1] and
X[i,column2]>=X[a,column2]:
            bucket.append(i)
        if params1==0 and params2==1 and X[i,column1]<=X[a,column1] and
X[i,column2]>=X[a,column2]:
            bucket.append(i)
        if params1==0 and params2==0 and X[i,column1]<=X[a,column1] and
X[i,column2]<=X[a,column2]:
            bucket.append(i)
    X,Y=np.delete(X,(bucket),axis=0),list(Y)

```

```

        bucket.sort(reverse=True)
        for i in range(len(bucket)):
            Y.remove(Y[bucket[i]])
        return X,Y,joe

def level2(X,Y,params1,params2):
    k,bucket=np.size(X,1),[]
    cont1=range(k)
    cont2=range(k)
    for i in range(k):
        m=random.choice(cont1)
        cont1.remove(m)
        for j in range(k):
            if m!=j:
                if np.size(X,0)<=2:
                    break
                a=level2column(X,Y,m,j,params1,params2)
                bucket.append(a[2])
                X,Y=a[0],a[1]
                print(np.size(X,0))
    return X,Y,bucket

def realiserlevel2(X,Y):
    a,b=[],[(0,0),(1,0),(0,1),(1,1)]
    while len(b)>0:
        i,j=random.choice(b)
        b.remove((i,j))
        gui=level2(X,Y,i,j)
        a.extend(gui[2])
        X,Y=gui[0],gui[1]
    return X,Y,a

def predictuniquelevel1(liste):
    end,res=0,None
    if liste[2]==1:
        if X[liste[1]]>=liste[0]:
            pred,end=liste[3],1
    if liste[2]==0:
        if X[liste[1]]<=liste[0]:
            pred,end=liste[3],1
    return end,res

def predictlevell(listlist):
    a=None
    for i in range(len(listlist)):
        gui=predictuniquelevel1(listlist[i])
        if gui[0]==1:
            a=gui[1]
            break
    return a

def predictarbre(listarbre,X):
    for i in range(len(listarbre)):
        if predictunique(listarbre[i][0],listarbre[i][1],listarbre[i][2],X)[0]==1:
            break
    return predictunique(listarbre[i][0],listarbre[i][1],listarbre[i][2],X)[1]

z=realiserlevel1(X,Y)
X=z[0]

```

```

Y=z[1]
a=laverlistlist(z[2])
print(a)
g=realiserlevel2(X,Y)

print((g[2]))
k=laverlistlist(g[2])
print(k)

def
result3uniformplus(matrix,value1,value2,value3,result,colon1,colon2,colon3,params1,
params2,params3):
    bucket=[]
    for i in range(np.size(matrix,0)):
        if params1==1 and params2==0 and params3==0 and matrix[i,colon1]>=value1
and matrix[i,colon2]<=value2 \
            and matrix[i,colon3]<=value3:
            bucket.append(result[i])
        if params1==1 and params2==1 and params3==0 and matrix[i,colon1]>=value1
and matrix[i,colon2]>=value2 \
            and matrix[i,colon3]<=value3:
            bucket.append(result[i])
        if params1==0 and params2==1 and params3==0 and matrix[i,colon1]<=value1
and matrix[i,colon2]>=value2\
            and matrix[i,colon3]<=value3:
            bucket.append(result[i])
        if params1==0 and params2==0 and params3==0 and matrix[i,colon1]<=value1
and matrix[i,colon2]<=value2\
            and matrix[i,colon3]<=value3:
            bucket.append(result[i])
        if params1==1 and params2==0 and params3==1 and matrix[i,colon1]>=value1
and matrix[i,colon2]<=value2 \
            and matrix[i,colon3]>=value3:
            bucket.append(result[i])
        if params1==1 and params2==1 and params3==1 and matrix[i,colon1]>=value1
and matrix[i,colon2]>=value2 \
            and matrix[i,colon3]>=value3:
            bucket.append(result[i])
        if params1==0 and params2==1 and params3==1 and matrix[i,colon1]<=value1
and matrix[i,colon2]>=value2 \
            and matrix[i,colon3]>=value3:
            bucket.append(result[i])
        if params1==0 and params2==0 and params3==1 and matrix[i,colon1]<=value1
and matrix[i,colon2]<=value2 \
            and matrix[i,colon3]>=value3:
            bucket.append(result[i])
        a=list(set(bucket))
        if len(a)==1 and len(bucket)>=30:
            res=1
        else:
            res=0
        return res,len(bucket)

def level3column(X,Y,column1,column2,column3,params1,params2,params3):
    liste=[]
    for i in range(np.size(X,0)):

gui=result2uniformplus(X,X[i,column1],X[i,column2],Y,column1,column2,params1,params2)
        if gui[0]==1:
            liste.append([gui[1],i])
    if liste != []:
        a,bucket=trouverplusgrand(liste,[])
        for i in range(np.size(X,0)):
            if params1==1 and params2==0 and params3==0 and X[i,column1]>=X[a,column1]

```

```

and X[i,column2]<=X[a,column2]\
    and X[i,column3]<=X[a,column3]:
    bucket.append(i)
if params1==1 and params2==1 and params3==0 and X[i,column1]>=X[a,column1]
and X[i,column2]>=X[a,column2]\
    and X[i,column3]<=X[a,column3]:
    bucket.append(i)
if params1==0 and params2==1 and params3==0 and X[i,column1]<=X[a,column1]
and X[i,column2]>=X[a,column2]\
    and X[i,column3]<=X[a,column3]:
    bucket.append(i)
if params1==0 and params2==0 and params3==0 and X[i,column1]<=X[a,column1]
and X[i,column2]<=X[a,column2]\
    and X[i,column3]<=X[a,column3]:
    bucket.append(i)
if params1==1 and params2==0 and params3==1 and X[i,column1]>=X[a,column1]
and X[i,column2]<=X[a,column2]\
    and X[i,column3]>=X[a,column3]:
    bucket.append(i)
if params1==1 and params2==1 and params3==1 and X[i,column1]>=X[a,column1]
and X[i,column2]>=X[a,column2]\
    and X[i,column3]>=X[a,column3]:
    bucket.append(i)
if params1==0 and params2==1 and params3==1 and X[i,column1]<=X[a,column1]
and X[i,column2]>=X[a,column2]\
    and X[i,column3]>=X[a,column3]:
    bucket.append(i)
if params1==0 and params2==0 and params3==1 and X[i,column1]<=X[a,column1]
and X[i,column2]<=X[a,column2]\
    and X[i,column3]>=X[a,column3]:
    bucket.append(i)
X=np.delete(X,(bucket),axis=0)
bucket.sort(reverse=True)
Y=list(Y)
for i in range(len(bucket)):
    Y.remove(Y[bucket[i]])
return X,Y

def level3(X,Y,params1,params2,params3):
    k=np.size(X,1)
    for i in range(k):
        for j in range(k):
            for l in range(k):
                if j!=i and l!=i and l!=j:
                    gui=level3column(X,Y,i,j,l,params1,params2,params3)
                    X,Y=gui[0],gui[1]
    return X,Y

```

Une fois toutes les colonnes étudiées, On passe aux conditions qui reposent sur deux coordonnées puis 3, puis ...10... en jouant sur les conditions < et >.

Un theme important est de savoir quel ordre de colonne étudier .L'ordre des colonnes étudiées donne des configurations d'arbre différentes. On fait varier ce paramètre pour créer une foret d'arbres. Il ne s'agit pas de bootstrap mais de variations de l'ordre du processus.

2.4.5 astuces utilisées

Les modèles énoncés précédemment ont de nombreux paramètres j'ai utilisé un module qui permet de optimiser le choix des paramètres en fonction des données d'entrée de l'algorithme. Il s'agit de hyperopt un module de scikit learn.

J'ai utilisé une fonction qui permet de multiplier le nombre de prédicteur dans la matrice, en créant une matrice qui sur la même ligne associe la ligne de la matrice initiale et ses lignes précédentes.

2.5)BACKTESTING

2.5.1principe

Un modèle est créé à partir d'une matrice de dimension $m*n$ et d'un vecteur de taille m . Une fois construit le modèle peut être utilisé pour réaliser des prédictions en entrant une vectrice ligne de dimension n et afin de sortir une valeur entière binaire représentant le caractère ascendant ou descendant du return futur d'un titre.

L'étape de backtesting permet de construire de nombreux modèles d'apprentissage statistiques, de les tester sur une période passée et de calculer leur hit score par comparaison avec la réalité, pour ensuite établir des intervalles de confiance lorsque l'utilisateur réalisera une véritable prédition.

La première chose est de limiter la période historique du backtesting, l'utilisateur rentre la date de début et la date de fin. Le backtesting repose sur une méthode de sliding windows. Une étape backtesting est caractérisée par une construction du modèle sur une période *ptrain*. Le modèle construit en *ptrain* est ensuite testé dans une période *ptest*, la période *ptest* suit immédiatement la période *ptrain*.

L'ensemble {*ptest+ptrain*} glisse d'un jour au cours de chaque étape de backtesting. Ce glissement a lieu le nombre de fois spécifié dans le paramètre sliding windows. Chaque modèle construit dans l'ensemble des périodes *ptrain* et chaque résultat des périodes *ptest* est enregistré.

```

def backtesting(matrix,result,nsw,ratio,d,methodalgo):
    imp=Imputer(missing_values='NaN', strategy='mean', axis=1, verbose=0, copy=True)
    matrix=imp.fit_transform(matrix)
    #matrix=cd.mettre_profondeurdata(matrix,d)
    periodtot=np.size(matrix,0)
    ptrain=int(np.floor((periodtot-nsw)*ratio))

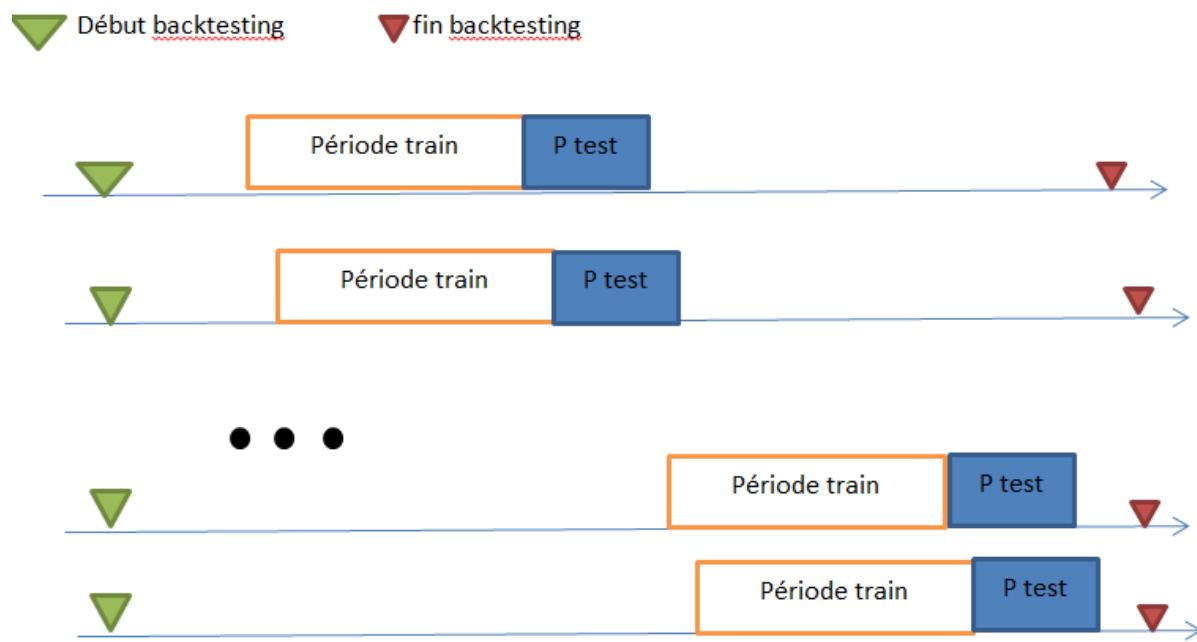
    ptest=periodtot-nsw-ptrain
    trainingbase,trainingtarget,trainingtargetbin,predictedvaluetraining=[[],[],[],[]]
    testbase,testtarget,testtargetbin,predictedvaluetest,learningmodel=[[],[],[],[],[]]
    etalon=matrix[ptrain,:,:]

    for i in range(nsw):
        print("period %s avec methode%s et profondeur%s"%(i,methodalgo,d))
        trainingbase.append(matrix[i:ptrain+i,:])
        trainingtarget.append(result[i:ptrain+i])
        testtarget.append(result[ptrain+i:ptrain+i+ptest])
        testtargetbin.append(bins(testtarget[i]))
        testbase.append(matrix[ptrain+i:ptrain+i+ptest,:])
        Ytrainbin=bins(trainingtarget[i])
        trainingtargetbin.append(Ytrainbin)
        learningmodel.append(methodalgo.fit(trainingbase[i],trainingtargetbin[i]))
        predictedvaluetraining.append(learningmodel[i].predict(trainingbase[i]))
        predictedvaluetest.append(methodalgo.predict(testbase[i]))

    return [ testbase,testtargetbin,predictedvaluetest,learningmodel,nsw,matrix,result,etalon]

```

Le rapport entre la durée de ptrain et ptest est au choix de l'utilisateur. Il est généralement de .66 et .34



2.5.2 hitscore

On dispose puisque nous sommes dans un passé de la véritable valeur du titre que l'on souhaite prédire. Chaque période ptest dans le backtesting donne lieu à une prédition que l'on compare avec les véritables valeurs. **Cela donne lieu au calcul du hitscore qui est le ratio entre le nombre de prédictions bonnes et le nombre de prédictions réalisées.** Celui-ci oscille entre 55 pourcents et 60 pourcents. (L'objectif global du projet étant d'atteindre les 60 pourcents). Une fois tous les hitscores collectés on calcule la distribution des hitscore pour avoir la variance de la prédition.

2.5.3 Intervalle de confiance

A la fin du processus de sliding windows. Pour un même vecteur ligne on a réalisé un nombre de prédition avec un modèle construit différemment à chaque fois en fonction de sa place dans la période et du nombre de sliding windows. (la période de fabrication du modèle a glissé au cours du processus).

Chaque résultat d'une prédition est suivi d'un constat : la prédition est fausse ou bonne. On compte 0 quand elle est fausse et 1 lorsqu'elle est bonne. On réalise cela pour chaque prédition on obtient donc un taux de réussite moyen pour chaque vecteur ligne.

Chaque vecteur ligne de la matrice prédicteurs est donc associé à un taux de réussite. Ce relevé de taux de réussite participera à la construction de l'intervalle de confiance lorsque l'utilisateur voudra faire une prédition.

En effet lorsque l'utilisateur va effectuer une prédition, il va rentrer un vecteur ligne de prédicteur. Après le backtesting on dispose donc d'une base de vecteurs ligne avec un taux de réussite moyen associé.

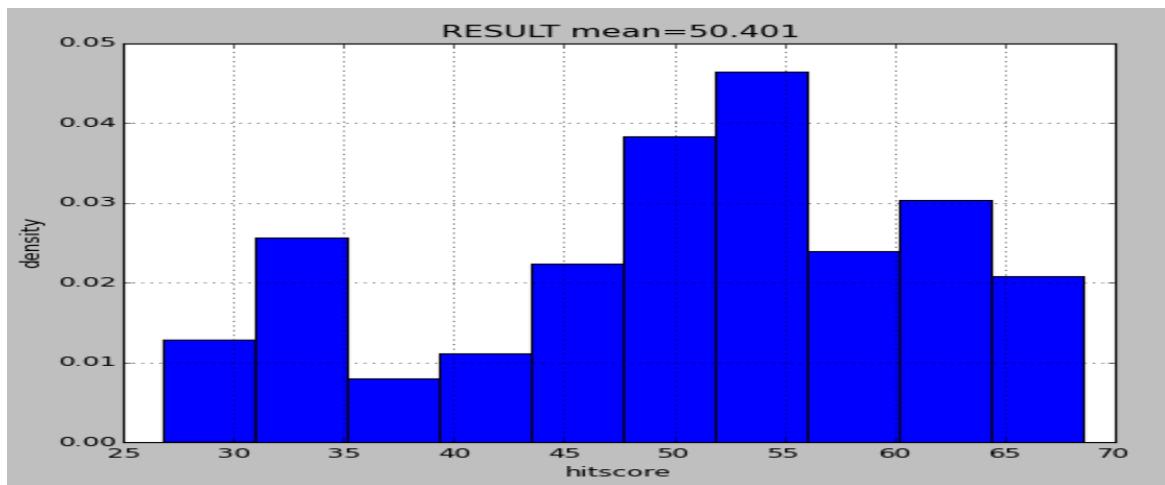


Figure 49 : résultat en termes de hitscore du backtesting

On va donc déterminer quel vecteur de cette base est le plus similaire à celui rentré par l'utilisateur.
On lui communiquera ainsi le taux de réussite moyen associé à ce vecteur ligne. En effet :

$$\begin{pmatrix} X_{1,1} & \cdots & X_{1,n} \\ \vdots & \ddots & \vdots \\ X_{p,1} & \cdots & X_{p,n} \end{pmatrix}$$

$$[[Y_1 \dots \dots \dots \dots Y_n]]$$

Pour chaque ligne i on calcule :

$$\sum_{j=0}^n \frac{(X_{1,j} - Y_j)}{X_{1,j}}$$

A partir de cela on détermine la ligne i tel que :

$$\min_{i \in [1,n]} \sum_{j=0}^n \frac{(X_{1,j} - Y_j)}{X_{1,j}}$$

Il s'agit de la vectrice ligne de la matrice de base donnée qui correspond le plus au vecteur sur lequel on souhaite réaliser la prédiction.

```

def stabmoy(listeliste, listelistel, nsw):
    listelisteint=[]
    for i in range(len(listeliste)):
        listelisteint.append(comparerdeuxlistebis(listeliste[i], listelistel[i]))
    mat=mettreenmatrice(listelisteint,nsw)
    data=[]
    for i in range(np.size(mat,1)):
        data.append(np.round(moyennan(list(mat[:, i])), 3))
    plt.hist(data,10,normed=True)
    plt.xlabel("confidence interval for each value")
    plt.title("moyen=%s" % (np.round(np.mean(data), 3)))
    plt.show()
    return data

def compalistlist(liste, listel):
    count=[]
    somme=0
    for i in range (len (liste)):
        if np.isnan(liste[i])and np.isnan(listel[i]):
            count=count+1
            somme=(liste[i]-listel[i])/liste[i]+somme
    return somme/count

def computenearestpred(listlist, liste):
    data=[]
    for i in range (np.size(listlist)):
        data.append(compalistlist(listlist[i], liste))
    f=sorted(data)
    min=f[0]
    return listlist[data.index(min)], data.index(min)

def stat(listeliste, listelistel):
    data=[]
    for i in range(len(listeliste)):
        data.append(comparedueuxliste(listeliste[i], listelistel[i]))

    plt.hist(data,10,normed=True)

    plt.xlabel("hitscore")
    plt.ylabel("density")

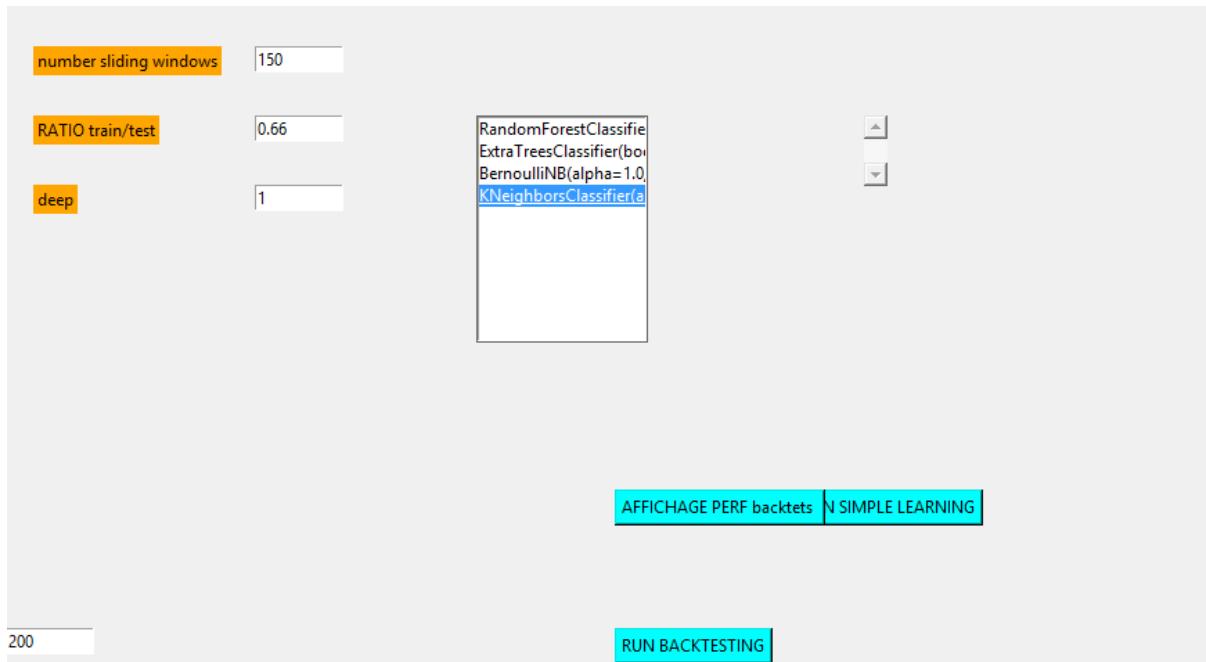
```

Figure 50: calcul des intervalles de confiance

La méthode de bootstrapp a aussi été utilisée elle consiste à construire les bases de construction de modèle et de testing. Elle consiste à créer des bases de testing dont les vecteurs lignes ne sont pas

forcément issues d'une succession chronologiques. Son intérêt est de réaliser plus de tests et d'apprentissage sur une période donnée que le processus d'une sliding windows.

L'étape de backtesting permet réellement de statuer sur l'efficacité des valeurs prises en compte dans la construction du modèle. La distribution des hitscore est importante à construire au fil du temps.



2.5.4 ranking prédicteurs

Il est nécessaire et intéressant de savoir quel prédicteur est utile ou non pour prédire un titre donné. J'ai donc réalisé un module permettant cela. Le principe est le suivant à partir de la matrice des prédicteurs initiale. On enlève à chaque étape une colonne de celle-ci et on effectue le procédé de backtesting, on relève la moyenne de la distribution des hitscore.

Une fois le processus terminé et toutes les moyennes relevées on effectue un classement des prédicteurs afin de déterminer quel prédicteur a vraiment un impact dans la construction du modèle, si sa présence est indispensable pour augmenter le hitscore. L'idée est de réaliser un catalogue en fonction du titre que l'on souhaite prédire sur les predictors susceptibles d'apporter une information statistique pertinente. C'est un outil de recherche infini puisque établir des statistiques demande des tests permanents.

```

def rankingpred(matrix,result,nsw_ratio,d,methodalgo):
    bucket=[]
    ind=[]
    for i in range (np.size(matrix,1)):
        ind.append(i)
        mat=np.delete(matrix,i, axis=1)
        a=backtesting(mat,result,nsw_ratio,d,methodalgo)
        bucket.append(statnoplota(a[1],a[2]))

    buck=sorted(bucket)
    ranking=[]
    for i in range(len(ind)):
        ranking.append(ind[bucket.index(buck[i])])
    return ranking

```

The screenshot shows a PyCharm interface with multiple tabs at the top: segmentation2 (2).py, backtestfinale (1).py, red.py, and result. The 'result' tab is active and contains a very large matrix of binary values (0s and 1s) displayed in a monospaced font. The matrix is so long that it extends beyond the visible area of the terminal window.

Figure 51: résultat d'un backtesting le décalage est du au processus de sliding windows.

2.5.5 Tests des modèles, efficacité.

Une fois le modèle construit, on peut tester l'efficacité de sa construction. Le processus est le suivant :

A partir du vecteur Target sur lequel on a construit le modèle, on crée une multitude de vecteurs Target en inversant l'ordre des coordonnées du vecteur initial. On réalise un apprentissage avec ces nouveaux vecteurs target mais en gardant la même matrice de prédicteurs. On teste ces modèles construits. Les hitscore alors produits doivent être inférieurs au hitscore initial. Si ce n'est pas le cas le modèle initialement construit n'est pas performant.

Exemple :

Modèle	matrice	vecteur	Vecteur détail	hitscore
Modèle officiel	Mat pred	Vecteur Target	[Y1,Y2,...,Yn]	58
	Mat pred	Vecteur Target 1	[Y18,Y27,...,Y3]	32
	Mat pred	Vecteur Target 2	[Yn,Yn-1,...,Y2]	47
	...	Vecteur Target ..	[....]	..
	Mat pred	Vecteur Target n-1	[Y9,Y2,...,Yn]	55
	Mat pred	Vecteur Target n	[Y3,Yn-1,...,Yn]	22

Le modèle est donc bien efficace

2.6 Prédictions

Une fois l'ensemble des modèles construits à partir de l'étape de backtesting. L'utilisateur doit se servir de tout ce process de préparation pour effectuer sa prédiction. Les paramètres de construction des modèles sont sauvegardés. Lorsque l'utilisateur souhaite effectuer sa prédiction il entre en paramètre la valeur qu'il souhaite en horizon. Dans le cas où les modèles sauvegardés correspondent à l'horizon entré, la prédiction est réalisée. Le résultat majoritaire des prédictions engendrées par les modèles créés lors du backtesting est annoncé avec son intervalle de confiance. Le vecteur d'entrée des modèles est généré automatiquement à partir de la date d'utilisation et des caractéristiques utilisées pour construire les modèles.

Conclusion :

Ce stage m'a permis de découvrir de nouvelles techniques mathématiques pour effectuer des prédictions. L'environnement travailleur et sérieux d'Exane m'a fait le plus grand bien pour me préparer à la vie professionnelle. Par ailleurs quelques tâches sans lien particulier avec ce projet principal, m'ont inséré dans une situation de travail en équipe, et permis d'engranger certains réflexes et fondements de l'analyse financière.