ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE TOURS
DÉPARTEMENT INFORMATIQUE
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 18
www.polytech.univ-tours.fr

Projet de Fin d'études

SVR avec Boosting
pour la prévision à long terme

Encadrant:

M. Aymen CHERIF

Étudiante :

M^elle^ Imane EL HASSANI

Année Scolaire 2011-2012

Table des matières

REMERCIEMENTS

Je tiens à remercier M Aymen CHERIF pour son encadrement tout au long de ce PFE, pour temps qu'il a pris sur sa thèse afin de suivre l'avancement de ce travail, pour son sens du partage et sa gentillesse. Merci d'avoir été disponible toute les fois qu'il l'a fallu, ni nonchalant ni encombrant: Le Juste parfait.

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

Le présent document est la synthèse de mon projet de fin d'études intitulé : "SVR avec Boosting pour la prévision à long terme", il décrit mon travail de huit mois sur la mise en œuvre d'un outil logiciel, permettant de combiner la méthode SVR et l'algorithme de Boosting afin d'améliorer les résultats des prédictions sur les séries de données temporelles, et ce sur des horizons de prédiction étendus.

Le rapport présente les objectifs du projet à développer, rappelle les spécifications techniques du système et expose des notions théoriques dans le domaine de la prédiction sur séries temporelles. Une deuxième partie pratique présente l'application développée et s'achève par une description des expériences et résultats obtenus.

Ce projet de fin d'études est encadré par M. Aymen CHERIF, doctorant à Polytech Tours dont la thèse traite des séries temporelles.

CHAPITRE 1 : OBJECTIF ET DESCRIPTION DU PROJET

1.1. Contexte:

L'application à réaliser pour le présent PFE s'inscrit dans le cadre de la prédiction à long terme, plus exactement celle relative au temps. La prédiction temporelle consiste à prévoir les valeurs futures d'une série temporelle en se basant sur des valeurs passées supposées connues. C'est un domaine d'étude particulièrement convoité depuis quelques années, mais également un des plus ardus car contraint pas l'accroissement de l'incertitude avec le temps.

On arrive actuellement à faire des prédictions justes et fiables sur les séries temporelles pour le court à moyen terme, toutefois les performances chutent considérablement dès que l'horizon de prédiction dépasse un certain seuil. Le résultat espéré par les chercheurs actuels est donc de diminuer le taux d'erreur sur un horizon de prédiction important.

1.2. Objectifs du projet:

Divers algorithmes d'optimisation tels le Bagging, le Boosting …. ont été combinés à de multiples techniques et classifieurs (Réseaux de neurones, cartes topologiques, Support Vector Machines…) afin de faire face aux problématiques d'accroissement du taux d'erreur dans la prévision temporelle, la méthode du Boosting paraît des plus prometteuses, et fait l'objet de multiples thèses et travaux récents, on peut citer à titre d'exemple l'étude réalisée et soutenue en 2006 au sein de Polytech' Tours par Mohammad Assaad, sous la direction de Hubert Cardot et Romuald Boné, intitulée : "Un nouvel algorithme de Boosting pour les réseaux de neurones récurrents : Application au traitement des données séquentielles" . Et qui a démontré l'effet du Boosting sur les données disposées en réseaux de neurones.

L'application à réaliser n'est pas une suite proprement dite de ce travail, mais s'inscrit dans le même registre: Il s'agit de créer un outil de prédiction pour les séries de données temporelles. Avec pour objectif la mise en évidence de l'effet du Boosting sur les données modélisées non pas en réseaux de neurones cette fois-ci, mais sur des Machines à Vecteurs de Support (SVM), reconnues meilleures en performances. Le détail des outils utilisés et de l'implémentation seront abordés plus amplement dans les chapitres suivants.

1.3. Description du projet

Pour mettre en œuvre le cœur métier de l'application, nous utiliserons la version Java de la librairie LIBSVM, c'est une librairie dédiée aux Machines à Vecteurs de Support, permettant de faire des classifications et des régressions.

Le processus de prédiction classique se déroule principalement en deux étapes: une phase d'apprentissage et une phase de test, Le prédicteur reçoit en entrée des données d'apprentissage qu'il utilise pour générer un modèle, il reçoit par la suite un fichier de test, dont il se servira pour prédire les nouvelles valeurs en se basant sur le modèle préalablement généré. Une fois le processus terminé, les données de test et les résultats des prédictions sont comparés pour traduire le taux d'erreur trouvé (cf. **Figure 1**).

Le fonctionnement de la LIBSVM traduit bien ces deux phases, il consiste à définir des paramètres d'entrée, à utiliser des fichiers d'apprentissage pour générer un fichier modèle, puis exploiter ce dernier pour faire des prédictions sur le fichier de test. A noter que les données à traiter sont contenues dans des fichiers plats, format standard interprété par la quasi-totalité des plateformes logicielles.

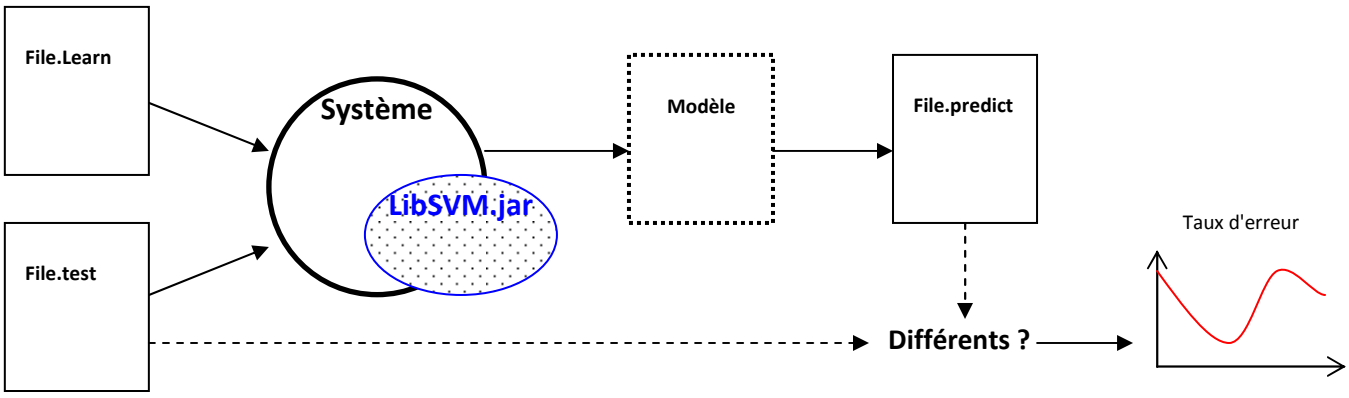


Figure 1: Architecture générale du système

L'application sera développée en langage Java, du Swing/SWT seront utilisés pour la gestion des interfaces graphiques, et le plugin GWT pour faciliter l'édition des interfaces sous L'IDE Eclipse. Les fonctions de la librairie LibSVM sont compressées dans un fichier .jar, qui sera inclus dans le package de l'application.

CHAPITRE 2 : ASSISES THEORIQUES EN PREDICTION TEMPORELLE

2.1. Prédiction sur séries Temporelles

2.1.1. Séries Temporelles

On parle d'une série temporelle pour désigner un ensemble de valeurs numériques observées, relatives à un phénomène qui évolue dans le temps. Depuis plusieurs années les séries temporelles ont été étudiées et analysées dans le but d'interpréter leur comportement dans le temps, que ce soit pour comprendre une évolution passés ou pour prévoir un comportement dans le futur. Elles sont représentables mathématiquement et donnent une idée de de la dynamique des variables représentées, en utilisant le plus souvent des concepts probabilistes et statistiques.

« Définition (Série Temporelle) : *La suite d'observations (**$y_t, t \in T$**) d'une variable **y** a différentes dates **t** est appelée série temporelle. Habituellement, **T** est dénombrable, de sorte que **$t=1,\ldots T$**.*

Une série temporelle est donc toute suite d'observations correspondant à la même variable : il peut s'agir de données macroéconomiques (le PIB d'un pays, l'inflation, les exportations…), microéconomiques (les ventes d'une entreprise donnée, son nombre d'employés, le revenu d'un individu, le nombre d'enfants d'une femme…), financières (le CAC40, le prix d'une option d'achat ou de vente, le cours d'une action), météorologiques (la pluviosité, le nombre de jours de soleil par an…), politiques (le nombre de votants, de voix reçues par un candidat…), démographiques (la taille moyenne des habitants, leur âge…). En pratique, tout ce qui est chiffrable et varie en fonction du temps. La dimension temporelle est ici importante car il s'agit de l'analyse d'une chronique historique: des variations d'une même variable au cours du temps, afin de pouvoir comprendre la dynamique. La périodicité de la série n'importe en revanche pas : il peut s'agir de mesures quotidiennes, mensuelles, trimestrielles, annuelles… voire même sans périodicité.»

Source : Emmanuel Cesar & Bruno Richard, « **Les Séries Temporelles** »,
Université de Versailles Saint-Quentin-en-Yvelines - Mars 2006.

2.1.2. Problématiques spécifiques aux séries temporelles

Depuis les premières études sur les séries temporelles, diverses problématiques ont émergé à ce niveau, et font l'objet de plusieurs recherches à ce jour. Parmi les difficultés majeures dans le traitement des séries temporelles figure la prédiction des valeurs futures d'une suite, la prédiction temporelle est actuellement utilisée dans divers domaines, elle s'avère d'une justesse satisfaisante

sur des horizons moyens, mais elle est toutefois contrainte par l'accroissement du taux d'erreur avec l'élargissement de l'horizon de prédiction.

« Il existe toute une gamme de problèmes spécifiques aux séries chronologiques qui ne sont pas étrangers aux praticiens de statistiques descriptives et qui vont nécessiter la mise au point d'un certain nombre de techniques pour un traitement économétrique (c'est-à-dire à fondements probabilistes). C'est là la première raison du développement de l'économétrie des séries temporelles. Ces problèmes sont les suivants : la prévision, l'identification et le retrait de la tendance, la correction des variations saisonnières, la détection de rupture, la séparation du court terme et du long terme, l'étude des anticipations des agents… »

Source : Sébastien LECHEVALIER, « Une introduction à l'économétrie des séries temporelles », Université Paris-I

2.2. Machines à Vecteurs Support (SVM)

2.2.1. Présentation

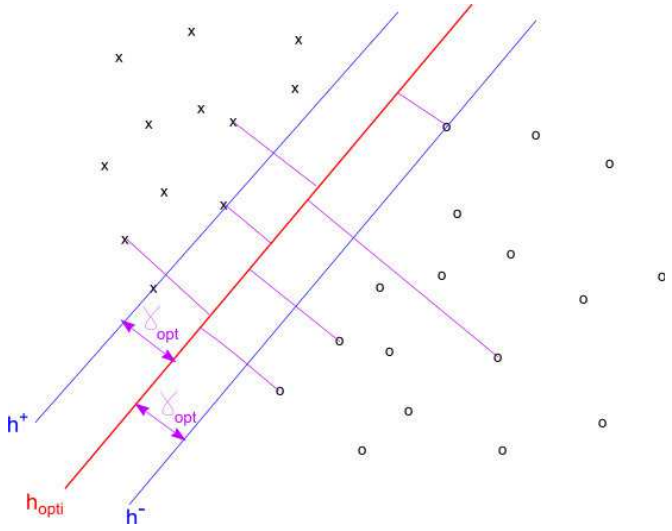
SVM (Support Vector Machines) ou Machines à vecteurs supports sont des classifieurs faisant partie des techniques d'apprentissage supervisé. Introduits par Vapnik (1995) pour résoudre des problèmes de classification, ils ont connu depuis un grand succès car utilisés massivement dans divers domaines : reconnaissance de formes, OCR, bioinformatique… L'usage s'est également répandu vers la résolution des problèmes de régression, aussi bien gérés que les classifications.

Cette technique fait appel à un jeu de données dit « d'apprentissage » dont les instances contiennent une valeur cible (target) également appelée « étiquette de classe » ainsi que plusieurs attributs (attributes) représentant l'ensemble des variables observées, et ce pour produire un modèle permettant de prédire les valeurs cibles d'un autre ensemble dit « de test », en ne fournissant à ce modèle-là que les attributs des données de test. En d'autres termes, si on considère un jeu de données divisé en deux groupes : un groupe pour les exemples connus et un autre pour les exemples non connus, le but des SVM est d'apprendre une fonction qui traduit le comportement des exemples connus pour prédire les cibles des exemples inconnus.

2.2.2. Fonctionnement des SVM

Les SVM sont également appelés « classifieurs à vaste marge » car leur objectif est de trouver l'hyperplan séparateur optimal qui maximise la marge entre les classes dans un espace de grande

dimension. La marge est la distance entre la frontière de séparation et les échantillons les plus proches, ces derniers sont appelés vecteurs supports. Une marge maximale permet d'obtenir une plus petite dimension de VC (Vapnik-Chervonenkis, « Théorie statistique de l'apprentissage »,1990), ce qui assure de bonnes performances en généralisation.



Source : Notes de Clément PUËLL du cours magistral de Reconnaissance des formes, dirigé par Hubert CARDOT

Figure 2: Hyperplan séparateur optimal

En considérant un jeu de données d'apprentissage avec des exemples (xi; yi); i = 1,……l, où xi ϵ R^n et y ϵ {-1,+1} ^{l}, les SVM (Boser et al., 1992; Cortes et Vapnik, 1995) requièrent la résolution du problème d'optimisation suivant :

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned}$$

2.2.3. Temps d'exécutions

Les SVM sont considéré plus faciles à utiliser que les réseaux de neurones. Leur avantage principal est leur performance remarquable en généralisation avec des données de grandes dimensions. On leur reproche cependant d'être lents en convergence et gourmands en mémoire dès qu'il s'agit de traiter des jeux de données volumineux. En fait, entrainer un jeu de données d'une taille n, revient à résoudre un problème de programmation quadratique (QP) de taille n^2, qui prend un temps de l'ordre $O(n^3)$. Cependant plusieurs recherches allant dans ce sens ont permis d'améliorer les temps d'exécution vers $O(n^2)$.

2.2.4. SVR

Lorsque les SVM sont utilisés dans des problèmes de régression pour prédire des valeurs réelles, on parle des SVR (Support Vector Regression).

« *Les SVM peuvent également être mis en oeuvre en situation de régression,c'est-à-dire pour l'approximation de fonctions quand Y est quantitative. Dans le cas non linéaire, le principe consiste à rechercher une estimation de la fonction par sa décomposition sur une base fonctionnelle. La forme générale des fonctions calculées par les SVM se met sous la forme :*

$$\phi(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^{\infty} w_i v_i(\boldsymbol{x}).$$

Le problème se pose toujours comme la minimisation d'une fonction coût, mais plutôt que d'être basée sur un critère d'erreur quadratique (moindres carrés), celle-ci s'inspire des travaux de Huber sur la recherche de modèles robustes et utilise des écarts absolus.

On note $|.|_\epsilon$ la fonction qui est paire, continue, identiquement nulle sur l'intervalle [0,ϵ] et qui croit linéairement sur [ϵ,+oo]. La fonction coût est alors définie par :

$$E(\boldsymbol{w}, \gamma) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \phi(\boldsymbol{x}_i, \boldsymbol{w})|_\epsilon + \gamma \|\boldsymbol{w}\|^2$$

Où γ est, comme en régression rigide, un paramètre de régularisation assurant le compromis entre généralisation et ajustement. De même que précédemment, on peut écrire les solutions du problème d'optimisation. Pour plus de détails, se reporter à Schölkopf et Smola (2002). Les points de la base d'apprentissage associés à un coefficient non nul sont là encore nommés vecteurs support. Dans cette situation, les noyaux k utilisés sont ceux naturellement associés à la définition de bases de fonctions. Noyaux de splines ou encore noyau de Dériclet associé à un développement en série de Fourier sont des grands classiques. Ils expriment les produits scalaires des fonctions de la base. »

Source : http://wikistat.fr/

2.3. Boosting (SVM)

2.3.1. Définition

Le Boosting est une technique servant à améliorer les capacités de généralisation d'un système de classification ou de prédiction en optimisant les performances de son algorithme d'apprentissage. Il fait partie des méthodes de combinaison de modèles, dont l'objectif est de combiner plusieurs hypothèses pour obtenir des estimations meilleures que celles d'un modèle unique. Ces méthodes ont la capacité d'améliorer les performances d'un algorithme d'apprentissage faible (weak learner),

pourvu qu'il fasse mieux que le hasard (se trompe moins d'une fois sur deux en moyenne). L'idée est qu'en divisant l'ensemble d'apprentissage en plusieurs sous-ensembles moins complexes, il est plus facile d'apprendre les données, et par conséquent d'obtenir une meilleure généralisation.

C'est (Schapire 1989) qui a introduit le premier algorithme de boosting à temps d'exécution polynomial. (Freund 1990) a pu développer une année après, un algorithme plus efficace mais qui avait cependant quelques inconvénients pratiques. La véritable émergence du boosting a débuté avec la création de AdaBoost -pour Adaptative Boosting- (Freund & Schapire, 1995), dont le détail est donné plus loin par un pseudo-code de l'algorithme.

2.3.2. Fonctionnement

Diverses méthode s'inscrivent dans le même registre que le boosting, tels le bagging,le cascading, le stacking, qui agissent toutes par combinaison de modèles, mais qui diffèrent dans leur manière de générer ces modèles-là. Le bagging par exemple divise l'ensemble d'apprentissage en plusieurs sous-ensembles et applique par la suite son algorithme d'apprentissage sur chacun d'entre eux.

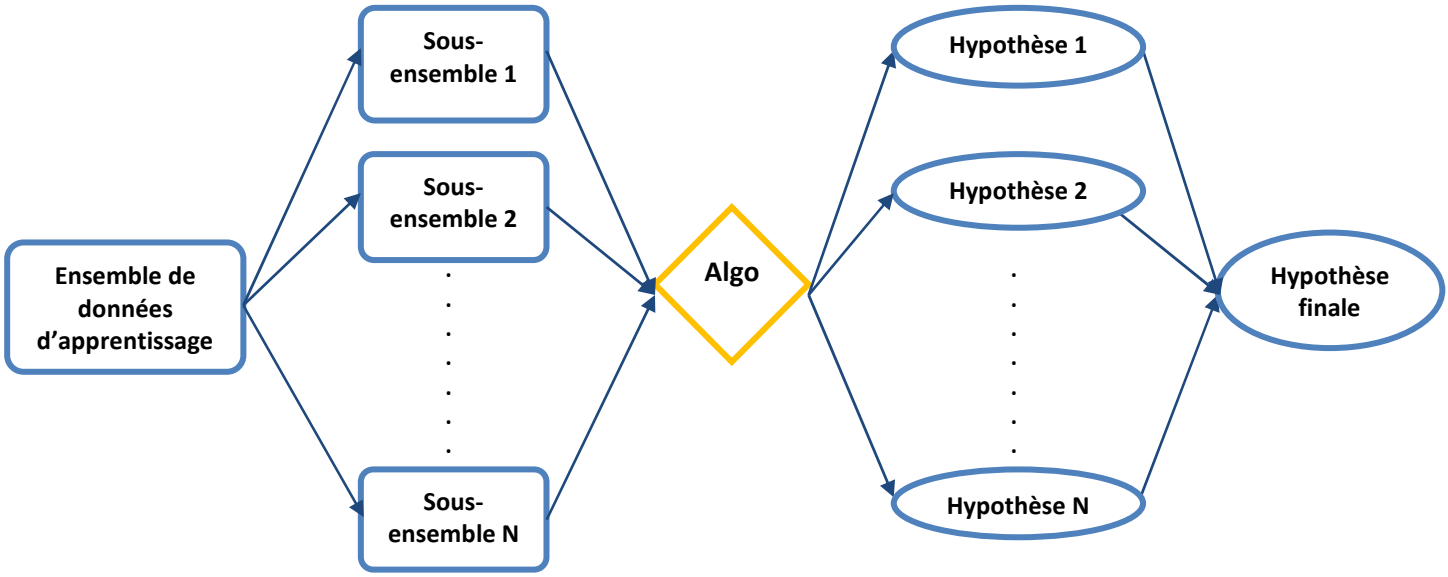


Figure 3: Fonctionnement du bagging

Le boosting quant à lui procède de façon séquentielle, à chaque itération la sélection d'un élément du nouveau sous-ensemble dépend des résultats obtenus lors de l'apprentissage précédent sur cet élément-là. Une façon de réduire rapidement l'erreur d'apprentissage en se concentrant sur les exemples les plus difficiles.

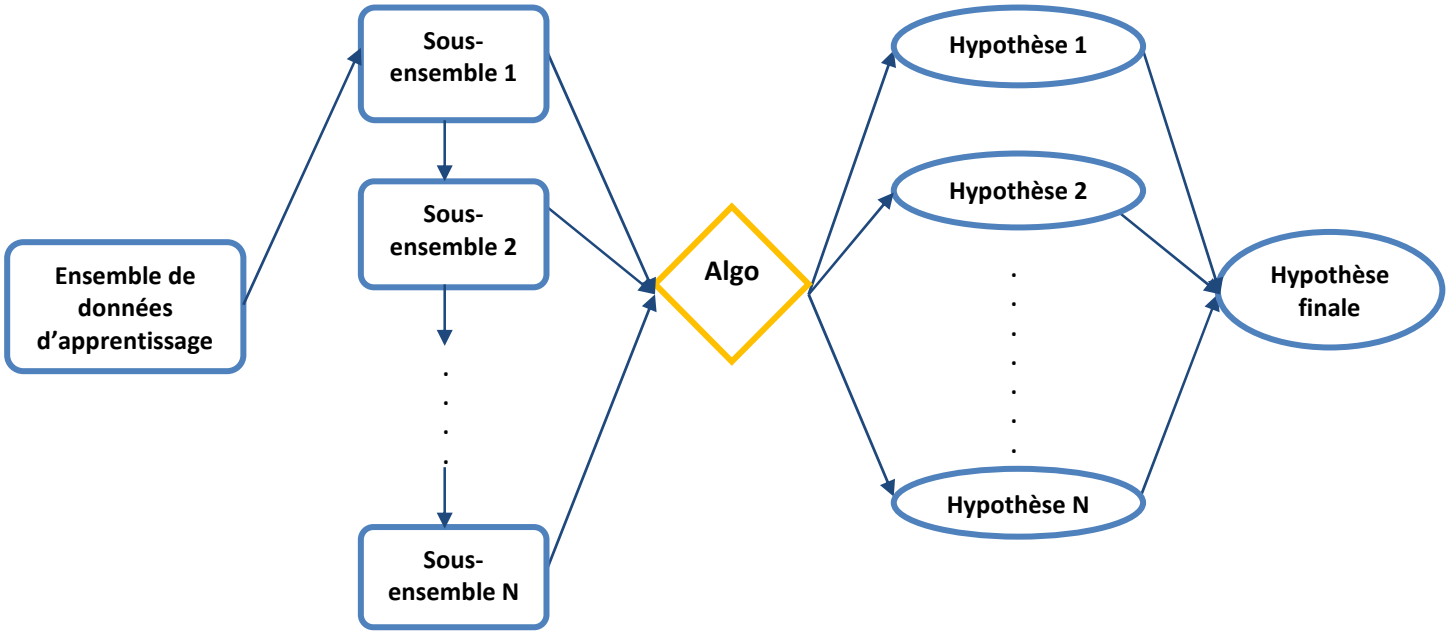


Figure 4: Fonctionnement du boosting

Le boosting a connu un grand succès sur divers problèmes d'optimisation, en montrant des performances très satisfaisante par rapport à d'autres algorithmes, entre autre à celles du bagging (dans le cas de bases de données non bruitées), ce qui explique le fait que son étude soit si prisée par les chercheurs actuellement.

2.3.3. Etat de l'art du boosting (Historique)

Les premiers intérêts en boosting dans le domaine de l'apprentissage automatique datent du début des années 80, avec la création du modèle mathématique « PAC » pour Probably Approximately Correct learning (Leslie Valiant, 1984). Ce modèle a été le premier à étudier la possibilité de transformer un algorithme d'apprentissage faible (légèrement meilleur que le hasard) en un algorithme fort performant. Ci-après un historique des principales recherches et expériences faites dans ce domaine :

- [Valiant'84] : Création du modèle théorique PAC pour l'étude de l'apprentissage automatique
- [Kearns & Valiant'88] : Introduction de la problématique de recherche d'un algorithme de boosting

- [Schapire'89], [Freund'90] : Premier algorithme de boosting avec un temps d'exécution polynomial

- [Drucker, Schapire & Simard '92] : Première expérience manipulant le boosting

- [Freund & Schapire '95] : Introduction de l'algorithme AdaBoost : une grande performance pratique par rapport aux algorithms de boosting antérieurs

- Expériences manipulant l'algorithme AdaBoost:
  - → [Drucker & Cortes '95]
  - → [Freund & Schapire '96]
  - → [Quinlan '96] [Breiman '96]
  - → [Jackson & Cravon '96]
  - → [Maclin & Opitz '97]
  - → [Bauer & Kohavi '97]
  - → [Schapire & Singer '98]
  - → [Schwenk & Bengio '98]
  - → [Dietterich'98]

- Enchainement du développement de nouvelles théories et algorithmes:
  - → [Schapire,Freund,Bartlett & Lee '97]
  - → [Breiman '97]
  - → [Mason, Bartlett & Baxter '98]
  - → [Grive and Schuurmans'98]
  - → [Friedman, Hastie & Tibshirani '98]
  - → [Schapire & Singer '98]

Source : Cours A. Coruéjols « Combiner des apprenants »

2.3.4. Adaboost (Adaptative Boosting)

Comme cité précédemment, c'est la création de l'algorithme Adaboost (Freund & Schapire, 1995) qui a révolutionné le domaine du boosting, Adaboost a pu résoudre plusieurs problèmes d'optimisation qui posaient des difficultés aux algorithmes de boosting le précédant. Il reprend le principe général du boosting en construisant une combinaison linéaire d'hypothèses, conçues en exécutant itérativement un algorithme d'apprentissage faible (weak leaner), sur des distributions de probabilité calculée en fonction des résultats des itérations précédentes. Ceci afin de permettre à l'apprenant de focaliser son attention sur les exemples difficiles à apprendre.

Depuis sa création, plusieurs versions d'Adaboost ont vu le jour, entre autre des versions spécifiques à la régression. Adaboost.R (Freund & Schapire, 1995) est un algorithme qui traite le

problème de la régression, en le ramenant à un problème de classification à deux classes sur lequel est appliqué l'algorithme Adaboost standard. Ci-dessous un pseudo code de Adaboost.R :

Données : $\{(\mathbf{x_q}, y_q); q = 1,\ldots,Q\}$ où $\mathbf{x_q} \in X$ est une entrée et $y_q \in Y = [0,1]$ est la sortie associée à $\mathbf{x_q}$.

Initialisation du vecteur des poids $d_1(q,y) = \dfrac{\tilde{d}(q)|y - y_q|}{z}$ pour $q = 1,\ldots,Q$ et $y \in Y$ où

$z = \sum_{q=1}^{Q} \tilde{d}(q) \int_0^1 |y - y_q| dy$. Notons $\tilde{d}(q)$ la distribution sur les exemples d'apprentissage.

Répéter

1. Etablir la densité $\mathbf{p}_n = \dfrac{\mathbf{d}_n}{\sum_{q=1}^{Q} \int_0^1 d_n(q,y).dy}$
2. Générer un nouvel ensemble d'apprentissage selon la densité $\mathbf{p}_n$ et construire une hypothèse $h_n : X \to Y$.
3. Calculer $\varepsilon_n = \sum_{q=1}^{Q} \left| \int_{y_q}^{h_n(\mathbf{x_q})} p_n(q,y).dy \right|$, l'erreur d'apprentissage de h_n.
4. Calculer $\alpha_n = \dfrac{\varepsilon_n}{(1-\varepsilon_n)}$.
5. Etablir le nouveau vecteur des poids :

$$d_{n+1}(q,y) = \begin{cases} d_n(q,y) & \text{si } y_q \le y \le h_n(\mathbf{x_q}) \text{ or } h_n(\mathbf{x_q}) \le y \le y_q \\ d_n(q,y).\alpha_n & \text{sinon} \end{cases}$$

Jusqu'à $\varepsilon_n < 1/2$. Notons N la dernière valeur de n.

Sortie : Calculer l'hypothèse finale

$$H(x) = h_f(\mathbf{x}) = \inf\left\{ y \in Y : \sum_{n:h_n(\mathbf{x}) \le y} \log\left(1/\alpha_n\right) \ge \frac{1}{2} \sum_{n=1}^{N} \log\left(1/\alpha_n\right) \right\}.$$

Source : Mohammad Assaad (2006), Thèse : « Un nouvel algorithme de Boosting pour les réseaux de neurones récurrents », Polytech' Tours

Figure 5: Algorithme Adaboost.R

2.3.5. Adaboost.R : Version Drucker 97

(Drucker 1997) a proposé une amélioration d'Adaboost.R en introduisant trois fonctions de cout spécifiques aux problèmes de régression. C'est la version de Drucker qui a été utilisé dans ce projet.

Données : $\{(\mathbf{x_q}, y_q); q = 1,\ldots,Q\}$ où $\mathbf{x_q} \in X$ est une entrée et $y_q \in \mathbf{R}$ est la sortie associée à $\mathbf{x_q}$.

Initialisation : $d_1(q) = 1/Q$ pour chaque exemple d'apprentissage q.

Répéter

1. La probabilité $p_n(q)$ que l'exemple d'apprentissage q soit dans l'ensemble d'apprentissage est : $p_n(q) = d_n(q) \Big/ \sum d_n(q)$.
2. Générer un nouvel ensemble d'apprentissage selon les probabilités $p_n(q)$ et construire un régresseur h_n sur l'ensemble d'apprentissage $h_n : X \to \mathbf{R}$.
3. Présenter chaque élément de l'ensemble d'apprentissage à ce régresseur pour obtenir l'estimation $y_q^{(n)}(\mathbf{x_q}); q = 1,\ldots,Q$.
4. Calculer l'erreur $l_n(q)$ pour chaque exemple d'apprentissage avec une des trois fonctions de coût :

   Linéaire : $l_n(q) = \left| y_q^{(n)}(\mathbf{x_q}) - y_q \right| / s_n$

   Quadratique : $l_n(q) = \left| y_q^{(n)}(\mathbf{x_q}) - y_q \right|^2 / s_n^2$

   Exponentielle : $l_n(q) = 1 - \exp\left[- \left| y_q^{(n)}(\mathbf{x_q}) - y_q \right| / s_n \right]$

   où $s_n = \sup_q \left| y_q^{(n)}(\mathbf{x_q}) - y_q \right|; q = 1,\ldots,Q$.

5. Calculer l'erreur d'apprentissage : $\varepsilon_n = \sum_{q=1}^{Q} l_n(q)\, p_n(q)$.
6. $\alpha_n = (1 - \varepsilon_n)/\varepsilon_n$ est une mesure de la confiance dans le régresseur.
7. Mettre à jour les poids : $d_n(q) \to d_n(q) \alpha_n^{(1 - l_n(q))}$.

Jusqu'à $\varepsilon_n < 1/2$. Notons N la dernière valeur de n.

Sortie : Calculer l'hypothèse finale en utilisant la médiane pondérée :

$$H(\mathbf{x}) = \inf\left\{ y \in Y : \sum_{n:h_n(\mathbf{x}) \le y} \log \alpha_n \ge \frac{1}{2} \sum_{n=1}^{N} \log \alpha_n \right\}$$

Source : Mohammad Assaad (2006), Thèse : « Un nouvel algorithme de Boosting pour les réseaux de neurones récurrents », Polytech' Tours

Figure 6: Version Drucker de l'algorithme Adaboost Pour la régression.

« Dans l'étape 4 de cet algorithme, le choix de la fonction de coût parmi les 3 proposées est laissé à l'utilisateur. Les deux fonctions quadratique et exponentielle croissent de façon plus rapide que la fonction linéaire en fonction de x, donc les fortes erreurs de prévision seront d'avantage pris en compte.

Pour le calcul de l'estimation finale par médiane pondérée, chaque régresseur h_n a une estimation $y^{(n)}_q$ sur le q ^{ème} exemple et une mesure de confiance associée α_n dans le régresseur. Pour un exemple q, les estimations sont renumérotées de manière à avoir $y^{(1)}_q < y^{(2)}_q <\ldots< y^{(N)}_q$ en gardant l'association entre α_n et $y^{(n)}_q$. On somme ensuite les $\log\alpha_n$ jusqu'au plus petit n pour obtenir la médiane pondérée :

$$\sum_n \log\alpha_n \geq \frac{1}{2}\sum_{n=1}^{N} \log\alpha_n$$

»

Source : Mohammad Assaad (2006), Thèse : « Un nouvel algorithme de Boosting pour les réseaux de neurones récurrents », Polytech' Tours

Pour la génération des nouveaux sous-ensembles (étape2 de l'algorithme de Drucker), c'est la technique de la roulette qui a été utilisé. Le principe de cet algorithme de sélection est simple : considérons un tableau de pourcentages (dans notre cas ce sont des probabilités), par exemple un tableau nommé pourcentage[1..n] , dont la somme des éléments est égale à N, on génère ensuite un nombre aléatoire entre 1 et N, appelons-le random, le traitement à effectuer par la suite est comme suit : (pseudo-code)

```
int  cumul  =  0

   for i = 1,n do
       cumul = cumul + pourcentage[i]
       if random <= cumul then
           return i
       end
   end
```

CHAPITRE 3 : RAPPEL DES SPECIFICATIONS DU PROJET

Ce chapitre rappelle brièvement les fonctionnalités à implémenter, l'environnement du projet et les spécifications techniques du système à créer.

3.1. Environnement du projet :

Le projet sera entièrement codé en langage Java, par conséquent le minimum logiciel requis pour pouvoir utiliser l'application est une machine virtuelle installée sur le poste où se déroulera l'exécution. Les fonctions de la LibSVM sont compressées dans un fichier .jar, qui sera inclus dans le package de l'application.

Lors du processus de prédiction, les phases d'apprentissage peuvent prendre des temps de calcul importants lorsque le volume de données traitées est grand, par conséquent plus la machine utilisée pour l'exécution de l'application est performante (mémoire/CPU) plus les temps d'exécution seront minimaux et vice versa. Les performances matérielles se verront réellement solliciter lors de l'exécution du processus de prédiction sur des bases de données réelles, généralement très volumineuses.

3.2. Langages et technologies utilisés :

Comme cité précédemment, l'application sera développée en langage Java, le code sera édité avec l'IDE Eclipse. Nous utiliserons du Swing/SWT pour générer les interfaces graphiques, en s'aidant de WindowBuilder plugin de Google Web Tools (GWT) permettant de faciliter l'édition des interfaces. Pour standardiser le développement, séparer les couches logiques et mieux structurer l'application, on s'appuiera sur la modélisation MVC, chose qui s'avère peu évidente vu le contexte de l'application, mais qui permettra de concevoir une architecture lisible, extensible, et surtout facile à maintenir.

La version Java de la LibSVM est architecturée sous forme de classes dépendantes les unes des autres. Ces classes décrivent la structure des objets qui contiendront à chaque instanciation les données des fichiers texte. Les classes contiennent également un ensemble de méthodes qui implémentent les algorithmes SVM de classification et de régressions, ce sont ces méthodes-là qui seront appelées par les fonctions métiers de l'application afin de mettre en œuvre le processus de prédiction.

3.3. Fonctionnalités du système

Le système doit permettre à l'utilisateur d'effectuer un certain nombre de tâches, qui ne sont d'autre que les étapes du processus de prédiction antérieurement décrits, à rappeler : définir des paramètres d'entrée, charger des données d'apprentissage pour générer un modèle, puis utiliser ce modèle pour faire des prédictions. Les fonctionnalités utilisateurs sont rassemblées dans le diagramme use case ci-dessous.

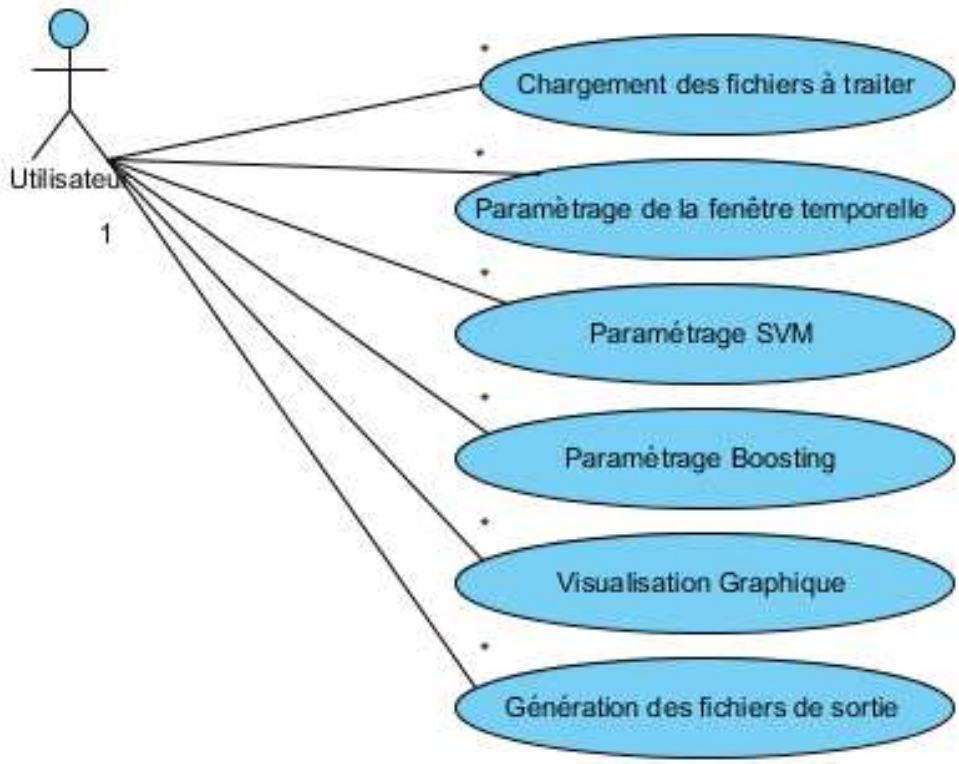


Figure 7: Use case global du système

3.4. Caractéristiques des utilisateurs

L'utilisateur principal de l'application, pour le moment, est l'encadrant du projet : M. Aymen Cherif, sa thèse traite des séries temporelles, c'est donc un praticien du domaine capable de manipuler l'application et d'interpréter les résultats des prédictions.

Aucun droit d'accès n'a été exigé pour l'utilisation de l'application, l'intérêt étant plus centré sur le développement de l'outil proprement dit et l'obtention de résultats probants.

3.5. Contraintes de développement:

La contrainte principale au niveau de ce projet est l'implémentation de l'algorithme de Boosting, d'une part cet algorithme nécessite un temps important à son étude et à sa compréhension avant de pour pouvoir le mettre en place, et d'une autre il en existe plusieurs versions dans la littérature, il s'agira de choisir la version la mieux adaptée au besoin de l'application.

Deuxième contrainte est que l'on ne dispose d'aucune référence pour valider les résultats obtenus, chose qui rend difficile de conclure catégoriquement sur les prédictions réalisées, il faudra par conséquent tester le code plusieurs fois, et de façon rigoureuse afin d'être sûrs qu'aucune erreur ne s'est glissée dans l'algorithme implémenté, par manque d'incompréhension ou suite à une faute d'inattention. Un codage modulaire s'impose afin d'éviter l'accumulation des erreurs et de pouvoir détecter les problèmes éventuels.

CHAPITRE 4 : DESCRIPTION DES OUTILS UTILISES

Ce chapitre décrit les principaux outils utilisés dans la mise en œuvre de l'application de prédiction temporelle, nous verrons une introduction à l'usage de la librairie LIBSVM et de ses principales fonctionnalités, puis une présentation du plugin « WindowBuilder » et de la librairie graphique « JFreeChart ». La description des outils est tirée en majeure partie de la documentation officielle de ces derniers.

4.1. LIBSVM

4.1.1. Introduction

La LIBSVM est une librairie dédiée aux Machines à Support de Vecteurs (Support Vector Machines). Élaborée par Chih-Chung Chang et Chih-Jen Lin, deux chercheurs du département informatique de l'université de Taipei à Taiwan « National Taiwan University ». La librairie englobe un ensemble de fonctions implémentant des algorithmes de fouille de données, particulièrement utilisée pour mettre en œuvre des classifications et des régressions. Elle est disponible en langage C++ et en JAVA, et compatible avec diverses plateformes logicielles (Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, LabVIEW, interfaces PHP, C# .NET, extensions CUDA).

La LIBSVM est vastement exploitée dans le domaine de l'apprentissage automatique (learning machines), principalement en supervisé. Elle a gagné en popularité depuis le début des années 2000 et jusqu'à maintenant vu qu'elle traite divers problèmes d'optimisation (minimisation) en intégrant des solveurs pour la classification de type C-support vector (C-SVC) et ν-support vector (ν-SVC), pour l'estimation de la distribution (one-class SVM), pour la régression de type ϵ-support vector (ϵ-SVR), et ν-support vector (ν-SVR), ainsi que pour la classification multi-classes. Les temps de traitement sont particulièrement satisfaisant tout autant que les résultats obtenus, raison pour laquelle la librairie est considérée comme référence en la matière.

Un site web dédié à la LIBSVM a été dressé par les auteurs pour fournir la documentation et les guides d'utilisation de la librairie, les exécutables et les codes sources en plusieurs langages (un point fort de l'outil), des jeux de données pour les tests, ainsi que les dernières mises à jours : http://www.csie.ntu.edu.tw/~cjlin/libsvm . De même une foire aux questions est présente sur le même site et est significativement constructive.Il est possible de contacter les auteurs pour toute remarque, report de bug ou suggestion d'amélioration de la librairie.

4.1.2. Fonctionnalités de la LIBSVM

La LIBSVM peut être exploitée en l'intégrant à un code applicatif donné, ou directement en faisant appel aux fichiers exécutables. Que l'utilisateur tourne sous un environnement Windows ou Unix, il peut facilement récupérer les makefile ou les DLL disponibles en ligne, puis de les compiler pour générer leurs .exe respectifs. Après compilation, l'utilisateur peut faire appel aux fonctionnalités de la librairie en ligne de commande, ci-dessous un descriptif des commandes les plus basiques :

- **Normalisation des données : svm-scale**

La normalisation des données est préconisée dans la plupart des traitements d'analyse, la LIBSVM appuie cette pratique et propose une fonctionnalité pour la normalisation : svm-scale, elle recommande une normalisation linéaire utilisant les intervalles [-1; +1] ou [0; 1], en utilisant biensur la même intervalle pour le jeu de données d'apprentissage et de test. Pour faire appel à cette commande il faut taper en ligne de commande :

svm-scale -y lower upper

Où **lower** et **upper** constituent les bornes de l'intervalle à utiliser pour la normalisation, généralement [0,1] ou [-1,1].

- **Apprentissage : svm-train**

La commande svm-train permet de lancer le processus d'apprentissage sur les données, elle prend en entrée un format de données spécifique, décrit plus loin, et génère un fichier de modèle. La commande pour utiliser cette fonctionnalité est :

svmtrain [options] training_set_file [model_file]

Où **training_set_file** constitue le fichier d'apprentissage et **model_file** le nom du modèle à générer.

- **Prédiction : svm-predict**

Génère les classes prédites ou les valeurs prédites (en cas de régression), en se basant sur un modèle entrainé, la commande à utiliser est la suivante :

svmpredict test_file model_file output_file

Où **test_file** est fichier de test, ou les données que l'on souhaite prédire, à noter qu'il doit également être sous le format LIBSVM et **model_file** le nom du modèle à générer.

Après prédiction, la fonction svm-predict compare les résultats de la prédiction avec les labels du fichier de test, pour en déduire la justesse de la prédiction et par la même occasion le taux d'erreur commis.

4.1.3. Paramètres SVM pris en charge par la librairie :

Afin d'entamer le processus d'apprentissage avec la LIBSVM, certains paramètres sont à renseigner selon qu'on souhaite faire une classification, une régression ou autre, le choix des bons paramètres est déterminant pour obtenir des résultats satisfaisants. Ci-dessous la liste de tous les paramètres LIBSVM possibles. À noter que dans le cas où ces paramètres ne sont pas renseignés, la librairie utilise les valeurs par défaut, précisées ci-après également:

-s svm_type : C'est le type de l'algorithme SVM à utiliser, peut être l'une des fonctions: C_SVC, NU_SVC, ONE_CLASS, EPSILON_SVR, NU_SVR, et peut prendre respectivement les valeurs suivantes :

0: Pour une classification de type C-SVM

1: Pour une classification de type nu-SVM

2: Pour une classification de type one-class-SVM

3: Régression de type epsilon-SVM

4: Régression de type nu-SVM

Par défaut le type SVM utilisé est C_SVC.

-t kernel_type : C'est le type de la fonction noyau à utiliser, peut être défini à : LINEAR, POLY, RBF ou SIGMOID, et par conséquent prendre les valeurs :

0: Pour utiliser une fonction linéaire de formule =u'*v

1: Pour une fonction polynomiale de formule = (gamma*u'*v + coef0)^degré

2: Pour une fonction Radiale de formule = exp(-gamma*|u-v|^2)

3: Pour une fonction sigmoïde de formule = tanh(gamma*u'*v + coef0)

4: Pour créer un kernel dans le fichier de test (cf. documentation LIBSVM pour le détail).

Tels que : **u'** représente la transposé du vecteur contenant les valeurs des attributs de l'ensemble d'apprentissage, et **v** le vecteur des labels (étiquettes). Le **gamma**, **degré** et **coef0** sont des paramètres (rentrés pas l'utilisateur). Par défaut le type de la fonction noyau utilisée est RBF, la documentation LIBSVM recommande l'utilisation de la fonction RBF pour plusieurs raisons, entre autre parce que RBF gère le cas où la relation entre les labels et les attributs est non linéaire.

Paramètres des fonctions noyau :

-d degree : Paramètre degré de la fonction noyau , par défaut 3

-g gamma : Paramètre gamma de la fonction noyau, par défaut 1

-r coef0 : Paramètre coef0 de la fonction noyau, par défaut 0.

Paramètres dépendants du type SVM choisi :

-c cost : C'est le paramètre C (coût), qui représente la pénalité de l'erreur, à renseigner lors de l'utilisation du type SVM C-SVC, epsilon-SVR et nu-SVR, par défaut le coût est égal à 1

-wi weight : pour changer le paramètre C à **weight*C**, s'il n'est pas renseigné weight est égale à 1 sa valeur par défaut, et par conséquent neutre.

-n nu : Paramètre nu du type nu-SVC, One-class-SVM et nu-SVR, par défaut 0.5

-p epsilon : Paramètre epsilon de la fonction de perte (Loss Function) pour le type epsilon-SVR, par défaut égal à 0.1

Paramètres SVM généraux :

-m cachesize : Pour paramétrer la taille mémoire (Mbits) allouée au noyau, par défaut 100

-e epsilon : C'est le critère d'arrêt de l'apprenant, par défaut 0.001. Il est recommandé d'utiliser une valeur de 0.00001 pour le nu-SVC, et 0.001 pour les autres types.

-h shrinking : Pour activer/désactiver l'heuristique de shrinking, par défaut mis à 1

-b probability-estimates : Pour activer/désactiver l'apprentissage avec probabilité, par défaut à 0

-v n : paramètre spécifique au mode validation-croisée (cross-validation), le n est le nombre de sous-divisions de l'ensemble d'apprentissage (n-fold), par défaut désactivé.

-q: Mode silencieux, pour omettre l'affichage des messages, par défaut mis à 0

À noter que la librairie propose un utilitaire pour vérifier la validité des paramètres avant de faire appel à la fonction d'apprentissage, vie la commande svm_check_parameter().

4.1.4. Fichiers de données

- **Entrées : Fichier au format LIBSVM**

Les fonctions de la LIBSVM reçoivent en paramètre des fichiers textes, qui doivent suivre un format bien particulier, il s'agit en fait d'un tableau où les données doivent commencer par le libellé de la classe (habituellement 1 ou -1), suivi d'une série de données sous la forme de paires (index : valeur). Dans le cadre d'une régression, comme c'est le cas pour ce projet, le principe est le même, sauf que les données du tableau commencent par une valeur réelle qui constitue la valeur à prédire appelée cible (target), suivie d'un ensemble de données toujours sous la forme (index : valeur) qui constituent les antérieurs de la cible.

En ouvrant un fichier d'apprentissage ou un fichier de test on retrouve l'ensemble du jeu de données sous la forme:

[label] [index1]:[value1] [index2]:[value2] ...

[label] [index1]:[value1] [index2]:[value2] …

[label] [index1]:[value1] [index2]:[value2] ...

Label : C'est le libellé ou la classe de l'enregistrement. C'est un entier lors d'une classification, et un réel lors d'une régression.

Index : Indexes dans l'ordre croissant, ils sont généralement contiguës.

Value : Ce sont les valeurs à apprendre, généralement des nombres réels.

Chaque ligne du fichier représente un enregistrement (ou un individu) appartenant un une classe du jeu de données, comme sur l'exemple ci-dessous :

+1 1:0.708 2:1 3:1 4:-0.320 5:-0.105 6:-1

- **Sorties :**

-Fichier modèle :

Le fichier du modèle contient les paramètres qui ont été utilisés lors de la phase d'apprentissage, suivis d'un tableau dont les lignes représentent les supports vecteurs. Les supports vecteurs sont listés dans l'ordre des labels, et regroupés par classes. Sachant que pour un jeu de données de k classes, on aura k-1 coefficients pour les supports vecteurs de chacune des classes.

Si l'on a par exemple un jeu de données de 4 classes, les supports vecteurs seront regroupés dans le modèle comme dans la représentation ci-après :

```
+-+-+-+--------------------+
|1|1|1|                    |
|v|v|v|  SVs from class 1  |
|2|3|4|                    |
+-+-+-+--------------------+
|1|2|2|                    |
|v|v|v|  SVs from class 2  |
|2|3|4|                    |
+-+-+-+--------------------+
|1|2|3|                    |
|v|v|v|  SVs from class 3  |
|3|3|4|                    |
+-+-+-+--------------------+
|1|2|3|                    |
|v|v|v|  SVs from class 4  |
|4|4|4|                    |
+-+-+-+--------------------+
```

Figure 8: Illustration du format des supports vecteurs

-Fichier de prédiction

Le format du fichier de prédiction résultant après l'appel de la fonction svm-predict est simple, chaque ligne de ce fichier contient le label relatif à la classe prédite pour la ligne équivalente du fichier de test. Dans le cas d'une régression, le fichier contient des valeurs réelles illustrant les prédictions trouvées pour les antérieurs contenus dans le fichier de test. La figure ci-dessous représente un fichier de prédiction (régression) ouvert sous Bloc note.

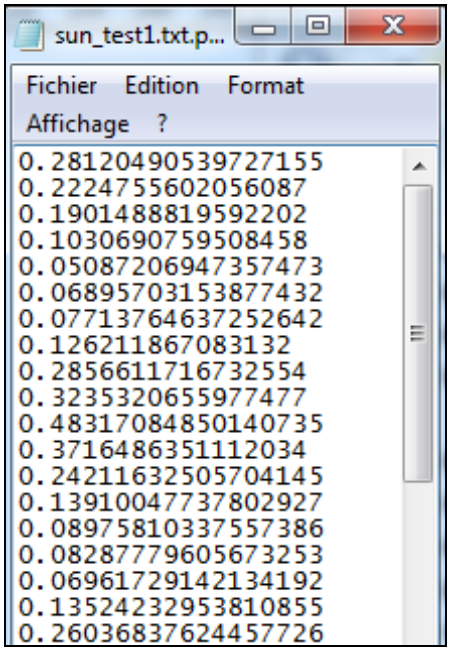


Figure 9: Exemple de fichier de prédiction dans le cas d'une régression

4.2. Edition d'interfaces : WindowBuilder (GWT)

WindowBuilder (http://www.eclipse.org/windowbuilder), est un plugin pour l'édition des interfaces graphiques sous JAVA. Conçu par Google et faisant initialement partie de la suite Google Web Tools (GWT), le plugin a été offert à la fondation Eclipse sous licence open-source. Il est compatible avec Eclipse ainsi que tous les IDEs qui sont basés dessus (RAD, RSA, MyEclipse, JBuilder…).

WindowBuilder est considéré l'un des meilleurs outils pour la construction WYSIWYG d'IHMs multiplateformes, et s'avère extrêmement stable. Il est constitué par Les deux Java Designer Swing et SWT et supporte diverses API : JFace, RCP, XWT et GWT. Son utilisation est simple, il offre un espace de travail assez intuitif, rappelant celui de Visual Basic, moins convivial que l'éditeur graphique de

Netbeans, et nettement plus lent, mais le rendu est tout aussi satisfaisant. Avec de simples Drag-and-drop, l'outil permet la création de formulaires basiques tout comme des fenêtres complexes, générant derrière le code java des composants utilisés.

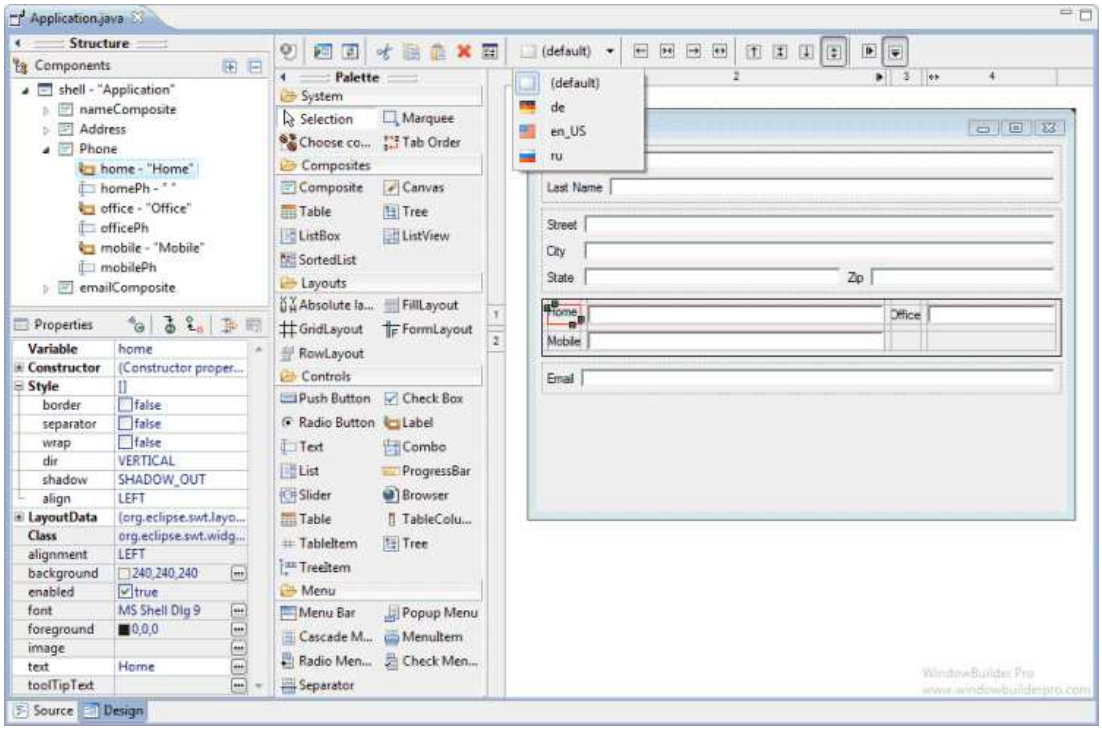


Figure 10: Interface WindowBuilder

Bien évidemment l'usage de ce type d'éditeurs facilite considérablement le développement sauf qu'il ne suit aucune logique d'organisation pour la génération du code, ce qui impose la restructuration manuelle du code par la suite, pour des raisons de fiabilité, de maintenabilité ou tout simplement pour les aficionados du code propre.

4.3. Rendus Graphiques : JFreeChart

JFreeChart est une librairie Open-source gratuite (http://www.jfree.org/jfreechart/), qui permet d'afficher des données sous la forme de graphiques. Compatible pour une utilisation dans des applications de type standalones (comme c'est le cas dans ce projet) ou des applications orientées web. La librairie propose plusieurs formats pour la représentation des données (camembert, barres, courbes, lignes …), l'un des points forts de celle-ci.

Elle offre également diverses options de configuration pour personnaliser à sa guise le rendu des graphiques, et permet d'exporter ces graphiques sous divers formats, incluant le format JAVA des

composants Swing, les fichiers images standards (PNG, JPEG), ainsi que les fichiers graphiques vectoriels (PDF, EPS, SVG) …

Pour pouvoir utiliser la librairie il suffit de télécharger l'une des versions disponibles sur le site web de l'éditeur, et d'ajouter dans son classpath le chemin vers les le dossier lib de la librairie décompressée.

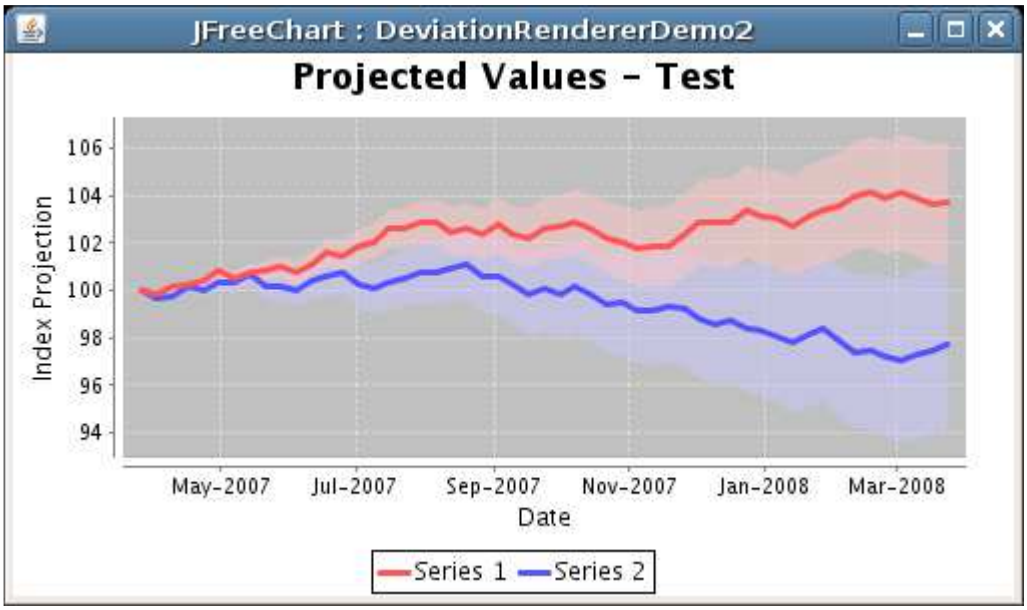


Figure 11: Exemple de graphique JFreeChart

Si la librairie est gratuite et open-source fournie sous licence GNU Lesser General Public Licence (LGPL), la documentation quant à elle est payante, elle est cependant bien faite et complète. À noter qu'une distribution de type LGPL permet l'utilisation de la librairie dans des applications propriétaires.

CHAPITRE 5 : PLANNING

Pour la planification du projet un découpage "SCRUM-like" a été utilisé, chaque itération (Sprint) contient un ensemble de tâches, liées ou indépendantes les unes des autre. À la fin de chaque itération on aboutit à un produit finalisé et prêt à être exploité par l'utilisateur.

5.1. Description des Sprints

- **Sprint 0 : Initialisation**

Ce Sprint est une phase de démarrage pour le projet, c'est une étape de découverte et de prise de conscience du sujet, c'est aussi un premier contact avec la documentation de la librairie LIBSVM.

- **Sprint 1 : Analyse et Spécifications**

Il s'agit à ce niveau d'étudier le projet de façon à dégager le besoin des utilisateurs, ce Sprint comprend l'analyse de la problématique, la rédaction des fonctionnalités de l'application, puis la prise en main du code source de la librairie LIBSVM.

- **Sprint 2 : Application Java : Implémentation SVR**

C'est à ce niveau que commence le développement en dur de l'application, il s'agit dans un premier temps de mettre en place la structure du système, puis de développer la fonctionnalité chargement de fichiers, avant d'entamer l'implémentation des fonctions SVR.

- **Sprint 3 : Application Java : Implémentation Boosting**

La deuxième phase du développement est l'implémentation de l'algorithme de Boosting, une étude préalable de l'algorithme est faite avant d'entamer le codage.

- **Sprint 4 : Application Java : Visualisations graphiques**

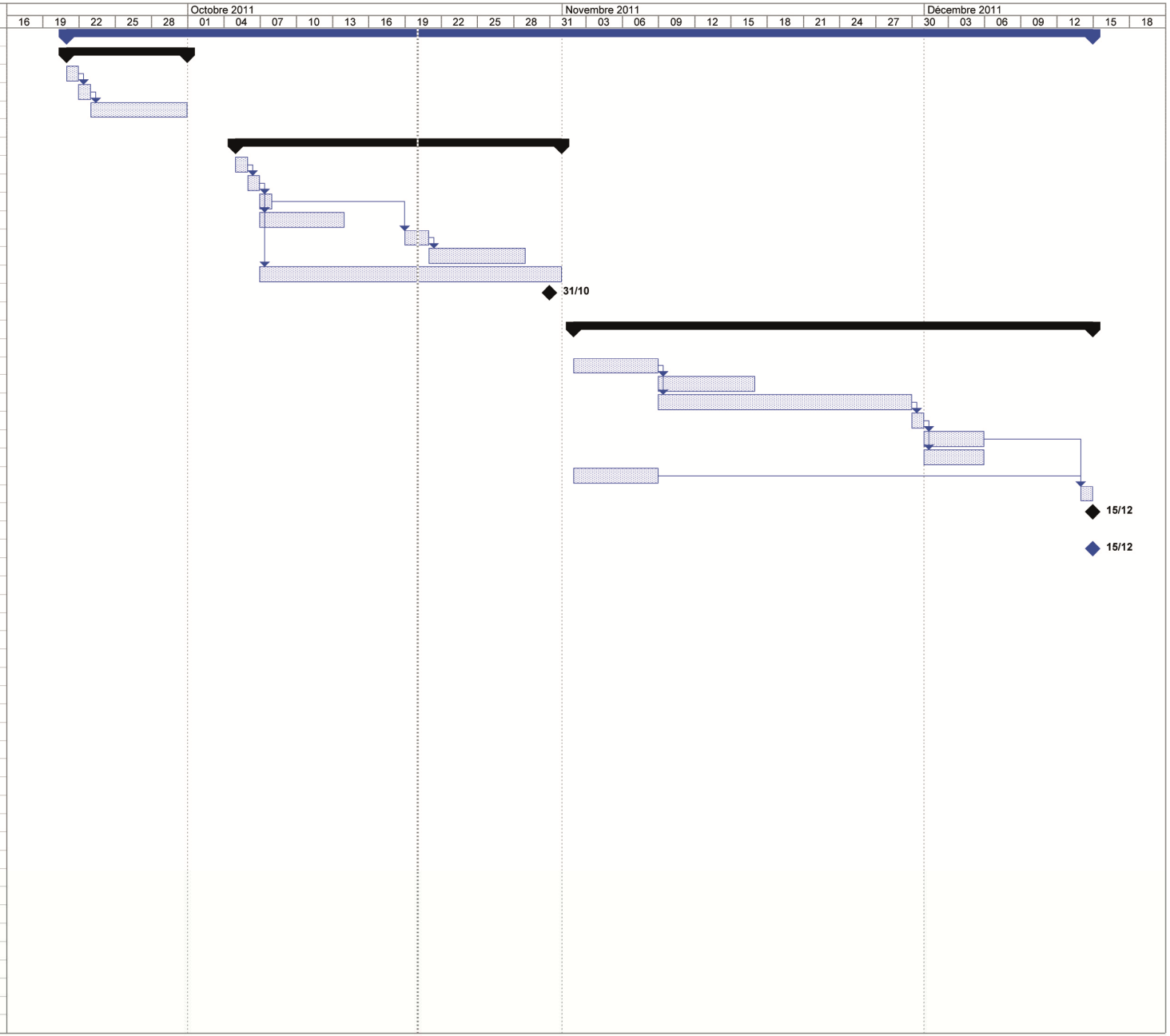
Il s'agit ici de mettre en place la fonctionnalité d'affichage des résultats sous forme de graphes.

- **Sprint 5 : Modules complémentaires**

Ce dernier Sprint se focalise sur le développement des modules complémentaire de l'application, tel l'ajout d'une fonctionnalité pour convertir les fichiers plats sous le format LibSVM standard.

5.2. Diagramme de Gant du projet

N°	Nom de la tâche
1	**RELEASE 1 : REALISATION DU MODULE SVR**
2	**Sprint 0 : Initialisation**
3	Réunion début PFE (Encadrants)
4	Prise de conscience de la problématique du PFE
5	Documentation sur l'utilisation LIBSVM + Tests sur librairie
6	
7	**Sprint 1 : Analyse et Spécifications**
8	Réunion de planification - Fonctionnalités de l'application (Encadrants)
9	Analyse de la problématique
10	Récupération des fichiers de données tests
11	Rédaction des fonctionnalités de l'application
12	Analyse LibSVM sur les fichiers de données fournis
13	Etude des codes source LibSVM
14	Rédaction cahier des spécifications
15	Livraison Cahier des spécifications
16	
17	**Sprint 2 : Application Java : Implementation SVR**
18	Réunion de planification - SVR (M. Cherif)
19	Mise en place de la structure de l'application
20	Développement du module Chargement de fichiers
21	Implémentation des fonctions SVR
22	Tests fonctionnels des modules développés
23	Test module SVR sur les fichiers de données fournis
24	Recherche sur la bibliothèque des rendus graphiques
25	Création du poster du PFE
26	Soutenance Mi-temps
27	Fin du développement du module SVR
28	
29	**Mise à disposition du Release 1**
30	
31	**RELEASE 2 : REALISATION DU MODULE BOOSTING + GRAPHES**
32	**Sprint 3 : Application Java : Implementation Boosting**
33	Réunion de planification - Boosting (M. Hallam, M.Bone)
34	Etude de l'algorithme de Boosting
35	Implémentation de l'algorithme de Boosting
36	Tests fonctionnels des modules développés
37	Test algo Boosting sur les fichiers de données fournis
38	Fin du développement du module Boosting
39	
40	**Sprint 4 : Application Java : Visualisations graphiques**
41	Réunion de planification - Visualisations (M. Cherif)
42	Développement du module Visualisations graphiques
43	Tests fonctionnels des modules développés
44	Fin du développement du module Visualisations
45	
46	**Sprint 5 : Modules complémentaires**
47	Réunion de planification - Modules complémentaires (M. Cherif)
48	Convertisseur de fichiers au format LibSVM
49	Tests de fonctionnels des modules développés
50	Soutenance finale
51	Fin du développement des modules Complémentaires Visualisations
52	
53	Rédaction du rapport
54	
55	**Mise à disposition du Release 2**



Projet : GantPFE
Date : Jeu 20/10/11

Tâche
Fractionnement
Avancement
Jalon
Récapitulative
Récapitulatif du projet
Tâches externes
Jalons externes
Échéance

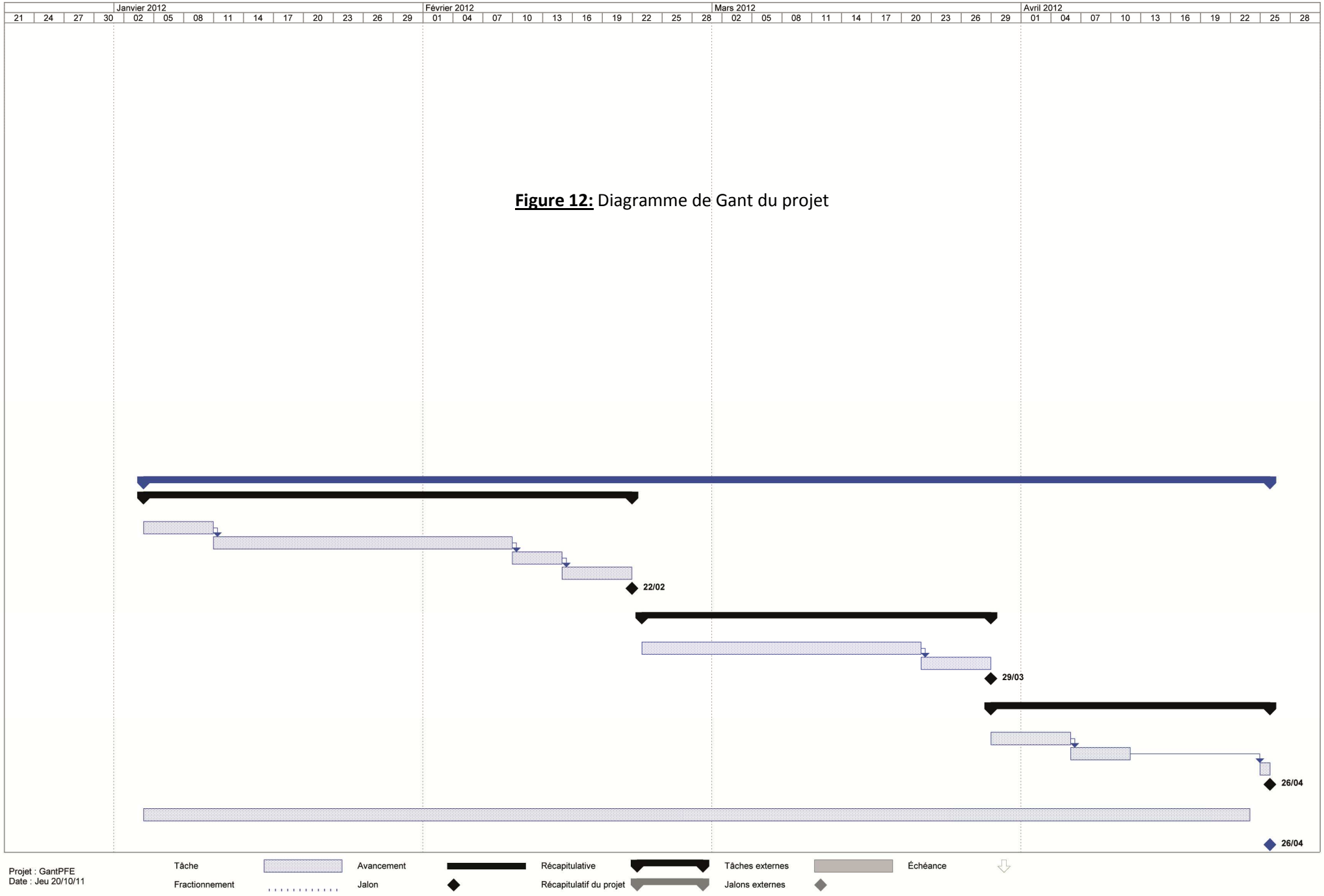


Figure 12: Diagramme de Gant du projet

6.1. Structure interne des données dans la LIBSVM :

Comme il a été mentionné précédemment, Les fonctionnalités de la LIBSVM peuvent être utilisées en ligne de commande via ses exécutables, ou en les implémentant dans le code d'une application. Dans le cadre de ce projet c'est plutôt la deuxième option qui sera retenue, les sources de la librairie seront incluses dans le package de l'application JAVA sous forme de fichier .jar. Ces sources JAVA (ou C++) sont les mêmes qui ont été utilisées pour créer les exécutables de la librairie.

L'ensemble de ces classes constitue la structure des objets qui vont englober à chaque instanciation les données de nos fichiers texte. Ces classes contiennent aussi les méthodes qui seront appelées par les fonctions métiers de l'application afin de mettre en œuvre le processus de prédiction. Ci-dessous une description de la structure de base des classes les plus importantes :

- **SVM-parameter**

```
package libsvm;
public class svm_parameter implements Cloneable,java.io.Serializable
{
    /* svm_type */
    public static final int C_SVC = 0;
    public static final int NU_SVC = 1;
    public static final int ONE_CLASS = 2;
    public static final int EPSILON_SVR = 3;
    public static final int NU_SVR = 4;

    /* kernel_type */
    public static final int LINEAR = 0;
    public static final int POLY = 1;
    public static final int RBF = 2;
    public static final int SIGMOID = 3;
    public static final int PRECOMPUTED = 4;

    public int svm_type;
    public int kernel_type;
    public int degree;  // for poly
    public double gamma;    // for poly/rbf/sigmoid
    public double coef0;    // for poly/sigmoid

    // these are for training only
    public double cache_size; // in MB
    public double eps;  // stopping criteria
    public double C;    // for C_SVC, EPSILON_SVR and NU_SVR
    public int nr_weight;       // for C_SVC
    public int[] weight_label;  // for C_SVC
    public double[] weight;     // for C_SVC
    public double nu;   // for NU_SVC, ONE_CLASS, and NU_SVR
    public double p;    // for EPSILON_SVR
    public int shrinking;   // use the shrinking heuristics
    public int probability; // do probability estimates
}
```

- **SVM-node**

```
package libsvm;
public class svm_node implements java.io.Serializable
{
    public int index;
    public double value;
}
```

- **SVM-problem**

```
package libsvm;
public class svm_problem implements java.io.Serializable
{
    public int l;
    public double[] y;
    public svm_node[][] x;
}
```

- **SVM-model**

```
package libsvm;
public class svm_model implements java.io.Serializable
{
    public svm_parameter param; // parameter
    public int nr_class;        // number of classes, = 2 in regression/one class svm
    public int l;           // total #SV
    public svm_node[][] SV; // SVs (SV[l])
    public double[][] sv_coef;  // coefficients for SVs in decision functions
(sv_coef[k-1][l])
    public double[] rho;        // constants in decision functions (rho[k*(k-1)/2])
    public double[] probA;          // pariwise probability information
    public double[] probB;

    // for classification only

    public int[] label;     // label of each class (label[k])
    public int[] nSV;       // number of SVs for each class (nSV[k])
                // nSV[0] + nSV[1] + ... + nSV[k-1] = l
};
```

6.2. Architecture Technique : Séparation des couches logiques : MVC

L'application à réaliser est simple d'un point de vue complexité des composants, il est toutefois important de la coder de façon modulaire, et de réaliser une séparation entre les couches logiques qui la composent, et ce pour une meilleure lisibilité du code et une meilleure standardisation de l'architecture logicielle. La séparation des entités logiques permet de concevoir des applications standards, extensibles et surtout faciles à maintenir et à faire évoluer. On peut par exemple aisément changer la manière d'accéder aux données sans se soucier de la partie présentation, ou de la manière avec laquelle ces données seront manipulées. L'application sera donc découpée en 3 couches.

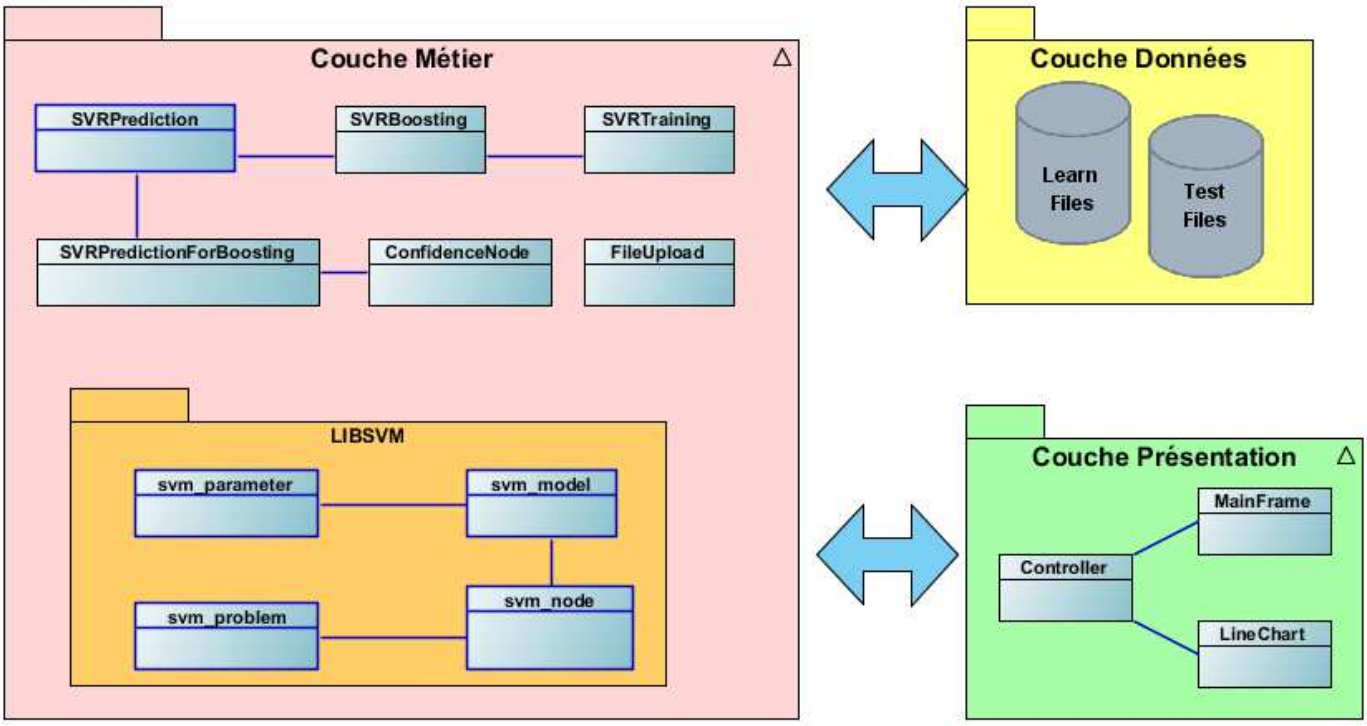


Figure 13: Découpage de l'application en 3 couches logiques

- **Couche Métier :**

La couche métier regroupe toutes les classes qui décrivent la logique du processus de prédiction (chargement des fichiers, apprentissage des données, prédiction des nouvelles valeurs, boosting…)

- **Couche Données :**

Représente les données de l'application, sachant qu'elles sont stockées dans des fichiers plats.

- **Couche Présentation :**

Rassemble le Contrôleur de l'application ainsi que l'IHM, c'est l'interface vue par l'utilisateur.

6.3. Diagramme de classes:

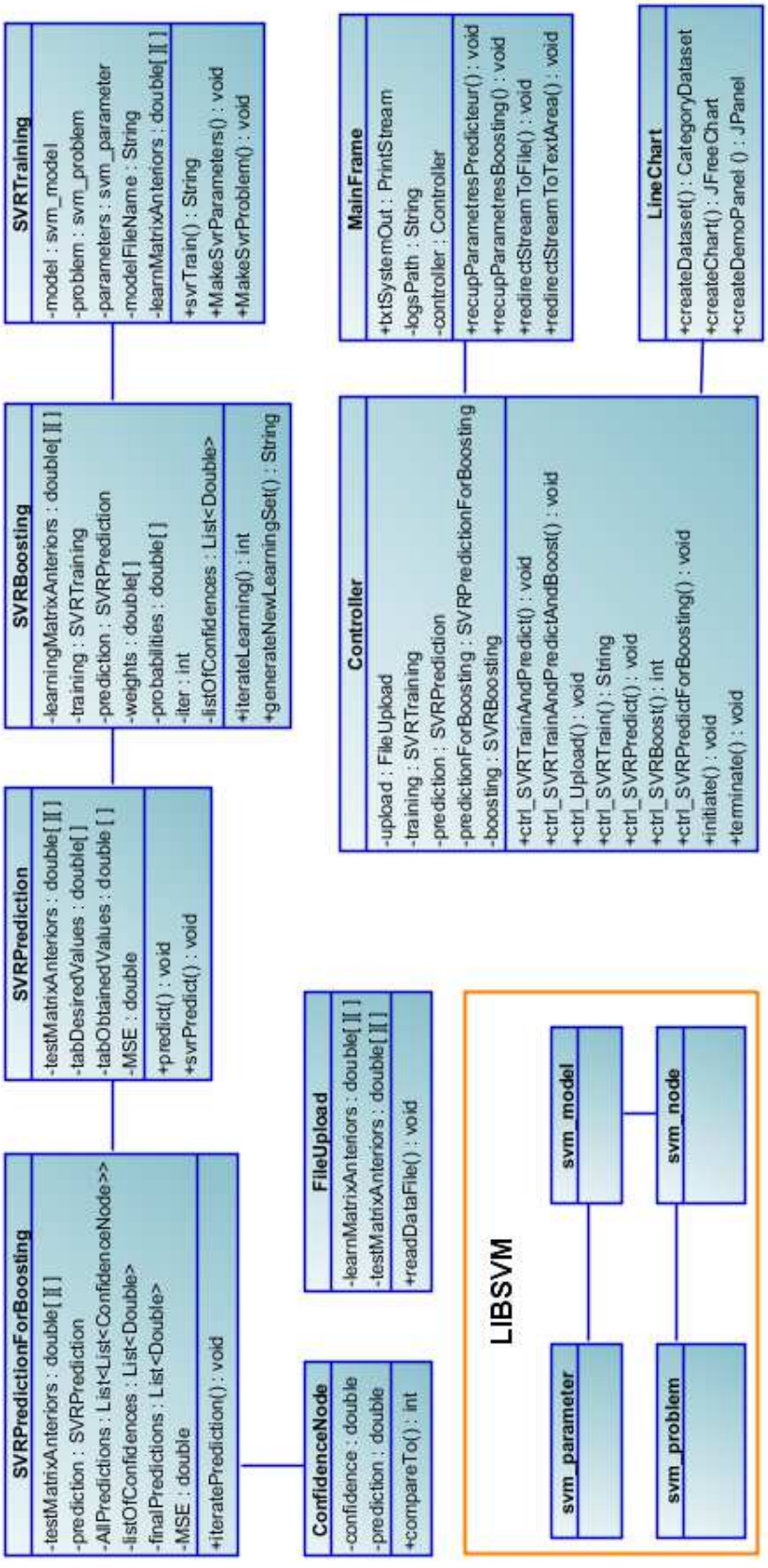


Figure 14: Diagramme de classes de l'application

6.4. Aperçu de l'application développée:

L'ergonomie de l'interface graphique n'a pas été particulièrement étudiée, l'intérêt étant plus centré sur le développement de l'outil proprement dit et l'obtention de résultats probants.

C'est une interface simple avec une organisation des plus basiques, elle contient des commandes standards communes : boutons, boutons radio, listes déroulantes, zones de texte… À améliorer par la suite.

6.4.1. Présentation de l'interface:

Au démarrage de l'application, La fenêtre de l'outil s'affiche, elle est composée de 5 parties : chargement des données **(1)**, choix de la fenêtre temporelle **(2)**, paramètres des SVM **(3)**, et paramètres de Boosting **(4)**, en plus d'une zone pour l'affichage des logs **(5)**.

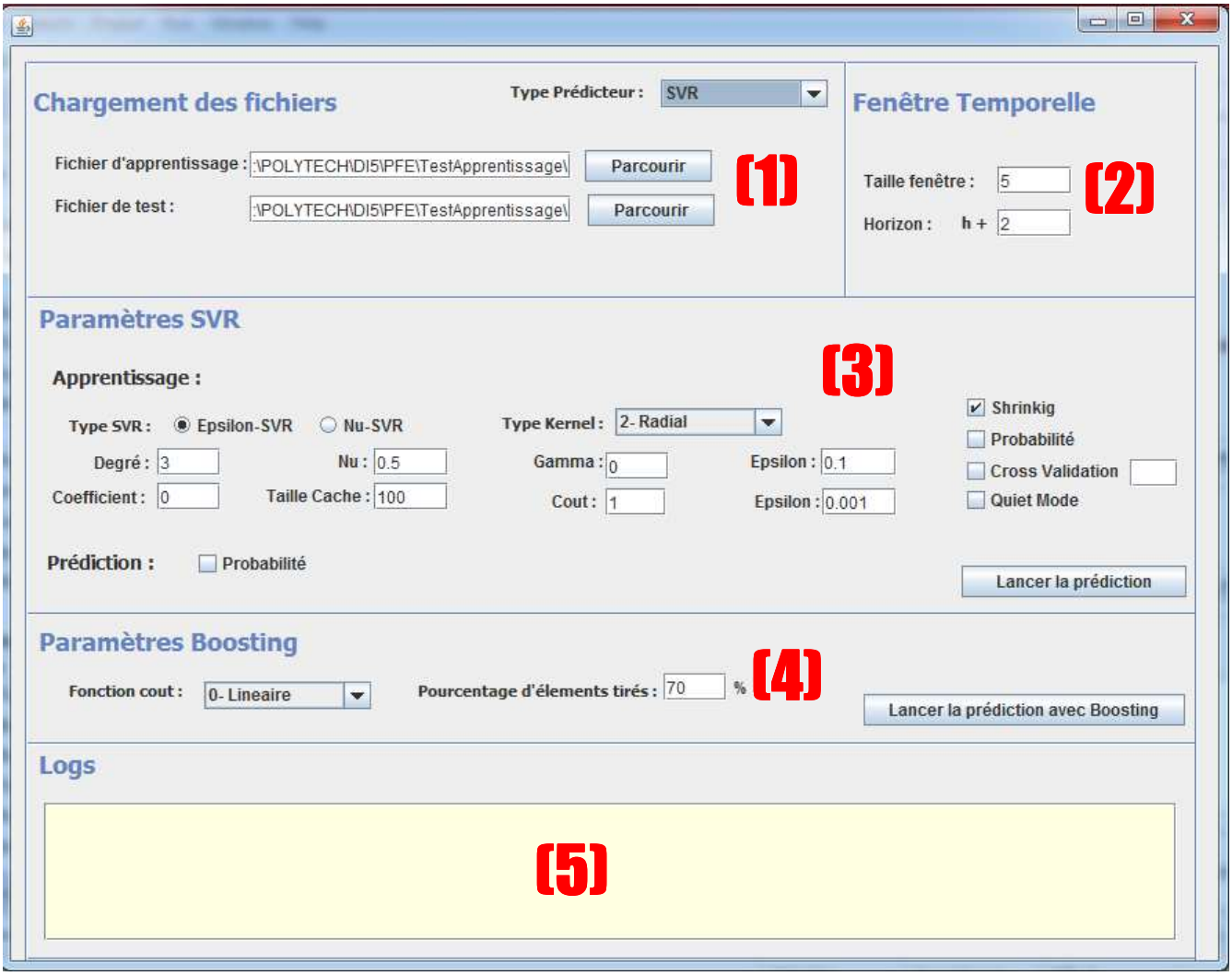


Figure 15: Interface de l'application

6.4.2. Chargement des fichiers

L'utilisateur commence par renseigner le chemin des fichiers de données, à rappeler que deux types de fichiers sont nécessaires à l'exécution du processus : un fichier pour les données d'apprentissage et un fichier pour les données de test. Les fichiers de validation n'ont pas été utilisés pour cette application.

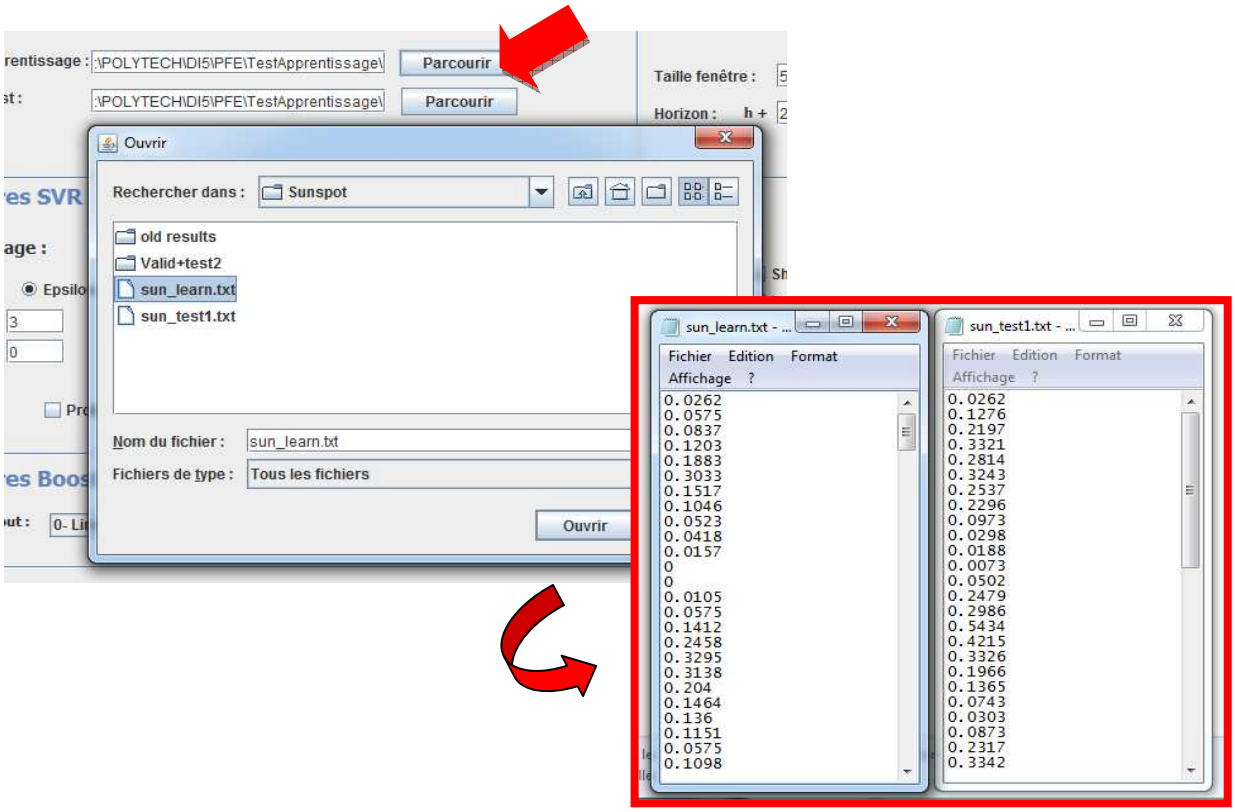


Figure 16: Chargement de fichiers

6.4.3. Choix de la fenêtre temporelle

L'utilisateur renseigne par la suite la taille de la fenêtre temporelle qu'il souhaite utiliser, ainsi que l'horizon sur lequel les prédictions seront faites. Ces deux paramètres seront envoyés à la fonction de chargement de fichiers pour mettre en forme les données.

Figure 17: Choix de la fenêtre temporelle

6.4.4. Paramètres SVM

L'utilisateur renseigne à ce niveau les paramètres des deux phases d'apprentissage et de prédiction : Type du SVR, Type du noyau, coût, Epsilon, Nu, Gamma, Degré, Taille du cache utilisée … (cf. section 1.3 du chapitre 4 pour la signification des paramètres SVM et le rôle de la sélection de paramètres dans le processus d'apprentissage).

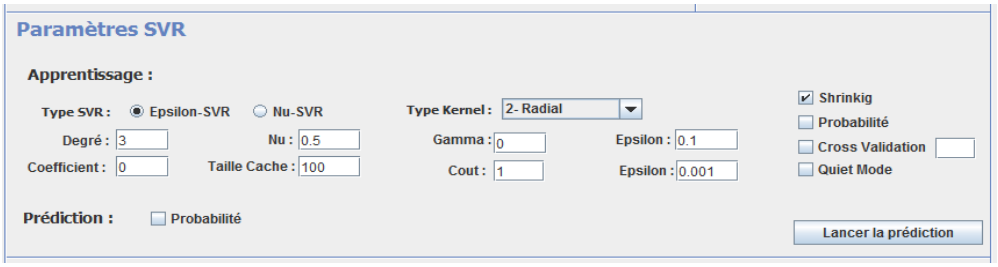


Figure 18: Choix des paramètres SVM

Les options par défaut utilisées par la LIBSVM sont déjà cochées. Une fois les paramètres renseignés, l'utilisateur peut lancer le processus de prédiction SVM standard (i.e. sans Boosting).

6.4.5. Paramètres Boosting

Dans la partie Boosting l'utilisateur renseigne deux paramètres :

- le type de la fonction de cout : à rappeler qu'il y en a 3 utilisés dans l'algorithme de Drucker: Linéaire, quadratique et exponentielle.
- et le pourcentage des éléments à tirer lors de l'exécution de l'algorithme de roulette (utilisé pour la sélection pondérée qui génère les nouveaux sous-ensembles d'apprentissage).

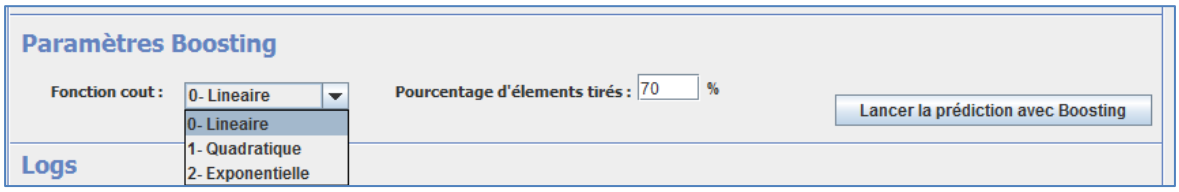


Figure 19: Choix des paramètres Boosting

6.4.6. Outputs de l'application :

Après l'exécution du processus d'apprentissage, la zone de log affiche un message pour en signaler la fin, un dossier nommé à la l'horodatage du jour est créé à l'emplacement où sont enregistrés les fichiers de données d'apprentissage, il contiendra l'ensemble des outputs de l'application. Le chemin du dossier créé est également affiché dans la zone de log pour rappeler à l'utilisateur l'emplacement exact.

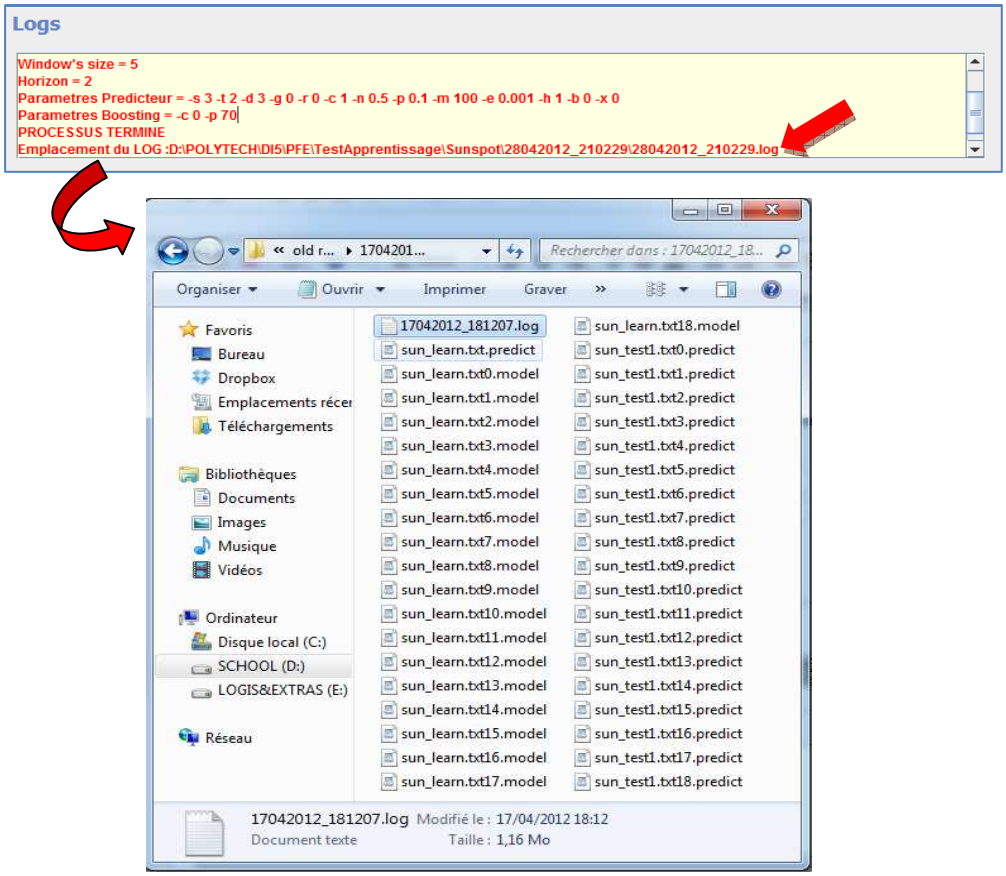


Figure 20: Contenu du dossier résultats de prédiction

À chaque exécution ce dossier contient:

- **Fichier Modèle :** Un seul modèle est généré si la prédiction a été faite sans boosting, ou plusieurs si elle a été réalisée avec boosting.

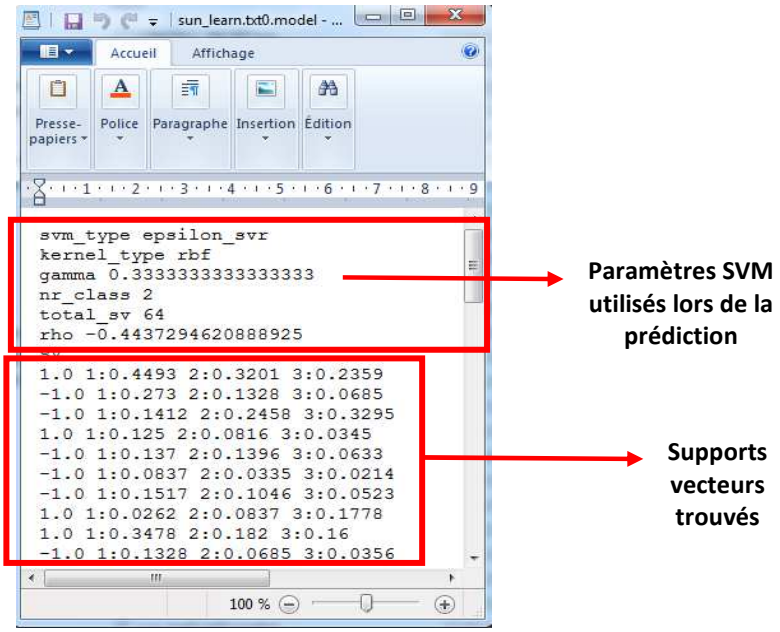


Figure 21: Contenu du Fichier modèle

- **Fichier des prédictions :**

Les prédictions faite sur la base du fichier test et du modèle (ou des modèles) sont contenues dans un fichier texte, à rappeler qu'il s'agit dans notre cas de régressions, les résultats générés sont donc des valeurs réelles.

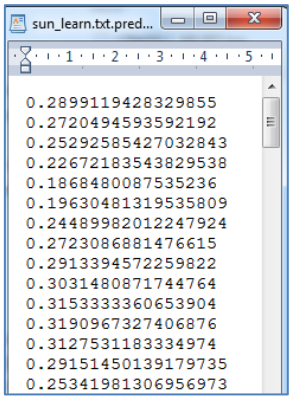


Figure 22: Contenu du Fichier des prédictions

- **Fichier log :**

Au fur et à mesure de l'exécution de l'application, plusieurs logs sont écris dans un fichier texte, ci-dessous un aperçu non exhaustif des informations enregistrées :

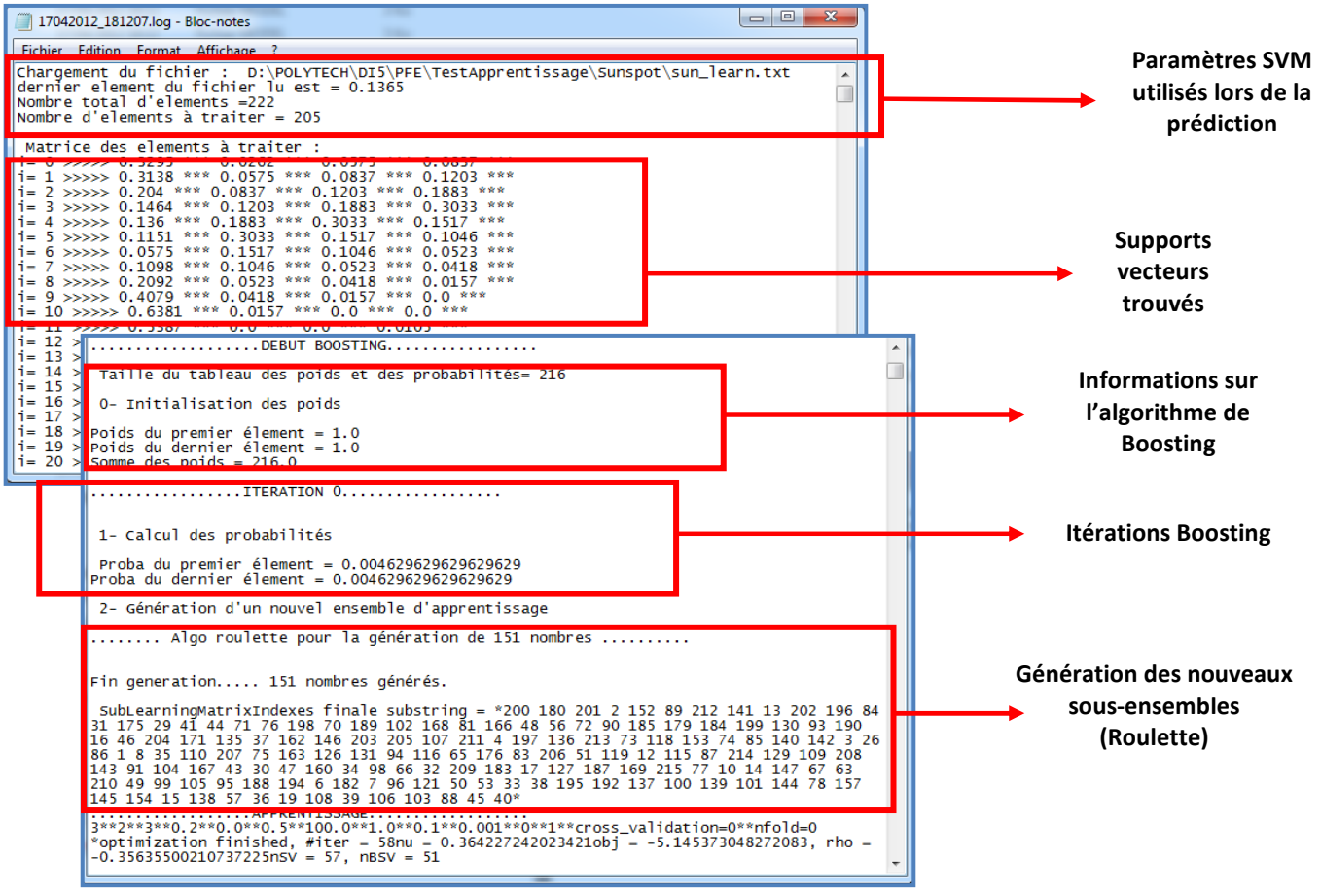


Figure 23: Contenu du Fichier de logs

6.4.7. Affichage des graphiques :

À l'issu de l'exécution de l'application un graphique s'affiche, il contient une représentation des valeurs désirées et obtenues du jeu de données de test : deux courbes en deux couleurs différentes, une légende pour expliquer les courbes, le taux d'erreur en prédiction (représenté par la MSE), et le chemin du fichier représenté.

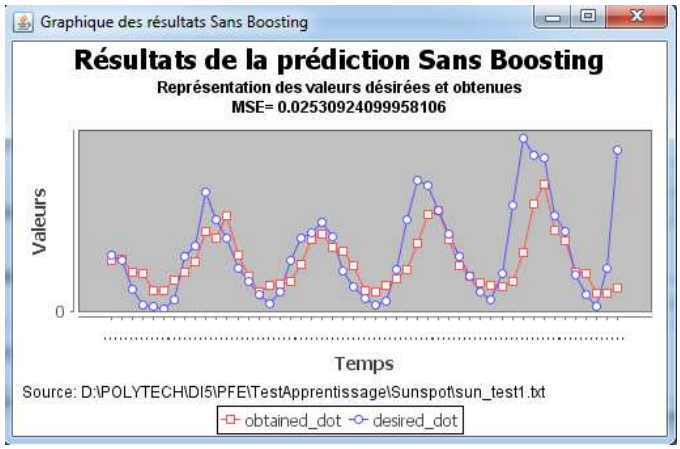


Figure 24: Affichage graphique des résultats

Comme pour l'interface de l'application, le design de la représentation graphique n'a pas été spécialement étudié, il pourra être amélioré par la suite. Comme on peut rajouter d'autres types de graphes représentant d'autres indices de performance.

CHAPITRE 7 : EXPERIENCES ET RESULTATS

7.1. Prétraitement des données

Normalisation des données :

Comme dans la plupart des analyses de données, le processus de prédiction avec des machines SVM nécessite la normalisation des données brutes, afin d'éviter que les valeurs à grandeurs extrêmes n'altèrent la qualité de l'analyse. De plus, les fichiers de données d'apprentissage et de test doivent être normalisés sur le même intervalle et de la même manière. La LIBSVM propose une fonctionnalité pour la normalisation des données, elle peut être utilisée en ligne de commande via la commande svm-scale. Une telle fonctionnalité peut être intégrée dans l'application dans le cadre d'une amélioration future.

Conversion des attributs qualitatifs :

SVM requière que les données soient représentées sous forme de vecteurs de réels, par conséquent les attributs qualitatifs doivent être convertis en données quantitatives. La LIBSVM recommande de convertir un attribut de m modalités en m attributs quantitatifs. Comme précisé précédemment, les tests réalisés ont été faits sur les données Sunspot (Taches solaires), ce sont des données réelles normalisées auparavant. L'application peut bien évidemment recevoir en entrée d'autres fichiers de données, il suffit de les traiter avant de les injecter.

7.2. Sélection des paramètres

La sélection des paramètres SVM est extrêmement importante, dans la mesure où elle influence de façon très significative le processus de prédiction, nous verrons dans les expériences qui suivent qu'une bonne sélection de paramètre rend obsolète le boosting des données dans certains cas.

Les paramètres à choisir ne sont pas identiques à tous les problèmes et ne sont pas forcément connus à l'avance pour un jeu de donné précis, ce qui impose de faire une sélection de paramètres. En d'autres termes il faut rechercher les meilleurs paramètres pour le kernel utilisé de façon à ce que le prédicteur prédise avec une plus grande justesse les valeurs inconnues (données de test).

La LIBSVM propose un utilitaire pour faire de la sélection de paramètres appelé le « Grid search », qui se base sur la division des données en plusieurs sous-ensembles (n-fold) de plus petite taille pour trouver les meilleurs paramètres C et Gamma. Une version spéciale pour la régression a été rajoutée sur le site web de la librairie, elle permet de donner en plus de C et gamma la valeur optimale du Epsilon à utiliser.

7.3. Expériences

Afin d'évaluer la qualité des prédictions obtenues au niveau des tests, l'Erreur Quadratique Moyenne ou Mean Square Error (MSE) a été utilisée comme mesure de l'erreur en prédiction.

7.3.1. Prédiction sans Boosting : Accroissement de la MSE avec l'horizon de prédiction

La première expérience menée est de tester l'accroissement de la MSE avec l'horizon de prédiction. Un tableau a été dressé à cette fin, s'étendant de h+1 à h+10 avec un pas de 1 et de h+10 à h+40 avec un pas de 5, dans lequel ont été recueillies les erreurs de prédiction commises sur les données Sunspots entrainées avec les paramètres LIBSVM par défaut, avec une taille de fenêtre égale à 5 (choix arbitraire, judicieux ?).

Horizon	h+1	h+2	h+3	h+4	h+5	h+6	h+7	h+8
MSE	**0,011683**	0,025309	0,033011	0,029690	0,026880	0,024970	0,025040	0,025744

Horizon	h+9	h+10	h+15	h+20	h+25	h+30	h+35	h+40
MSE	0,025893	0,027060	0,046806	0,054845	0,067781	0,083632	0,087654	**0,129577**

Tableau 1 : Accroissement de la MSE avec l'horizon de prédiction

Pour s'assurer de l'exactitude des valeurs prédites par l'application, les fichiers plats Sunspot ont été convertis sous le format LIBSVM puis utilisés pour la prédiction moyennant les exécutables de la librairie : svm-train et svm-predict. Les exécutions en ligne de commande donnent les mêmes valeurs que celles données par l'application, les valeurs générées par l'application sont donc justes.

En traduisant le tableau de données en graphique nous obtenons :

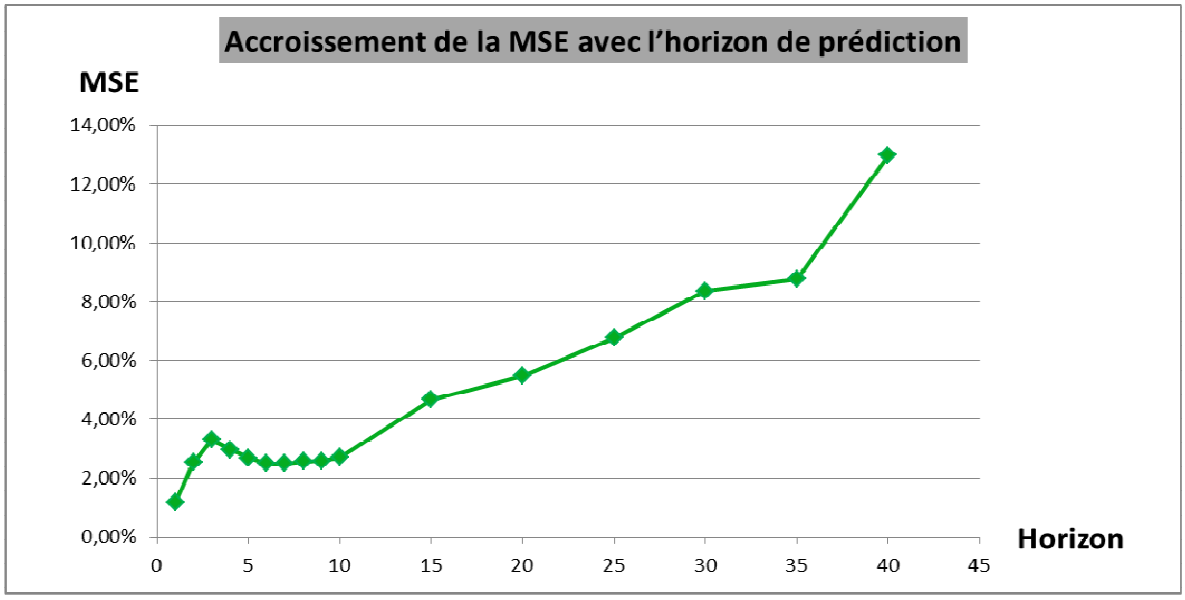


Figure 25 : Accroissement de la MSE avec l'horizon de prédiction

L'erreur est visiblement grandissante avec l'accroissement de l'horizon de prédiction.

7.3.2. Évolution de la MSE avec sélection des paramètres

Une recherche de paramètre a été effectuée à ce niveau afin de récupérer les meilleurs paramètres C, gamma et Epsilon pour chacun des jeux de données testés (de h+1 à h+40). Les paramètres suivants sont sélectionnés :

Horizon	h+1	h+2	h+3	h+4	h+5	h+6	h+7	h+8
Best C	16	32	16	1	0.5	0.5	0.5	0.5
Best Gamma	1	1	0.5	1	0.5	0.25	0.0625	0.25
Best Epsilon	0.0078125	0.03125	0.125	0.0625	0.0078125	0.015625	0.00390625	0.0625

Horizon	h+9	h+10	h+15	h+20	h+25	h+30	h+35	h+40
Best C	32	2	0.5	0.5	1	2	4	32
Best Gamma	0.5	0.25	1	1	0.25	0.5	0.25	0.25
Best Epsilon	0.0625	0.125	0.0625	0.125	0.0625	0.125	0.125	0.125

Tableau 2 : Paramètres d'apprentissage optimaux

En lançant de nouvelles prédictions avec les paramètres sélectionnés nous obtenons les résultats ci-dessous:

Horizon	h+1	h+2	h+3	h+4	h+5	h+6	h+7	h+8
MSE	**0,00634**	0,01743	0,02266	0,02608	0,02298	0,02279	0,02399	0,02663

Horizon	h+9	h+10	h+15	h+20	h+25	h+30	h+35	h+40
MSE	0,02725	0,02489	0,04363	0,04386	0,06944	0,07377	0,08212	**0,11650**

Tableau 3 : Accroissement de la MSE avec sélection des paramètres

Traduits en graphes et comparés aux résultats des prédictions sans sélection de paramètres, les tableaux donnent l'évolution ci-après :

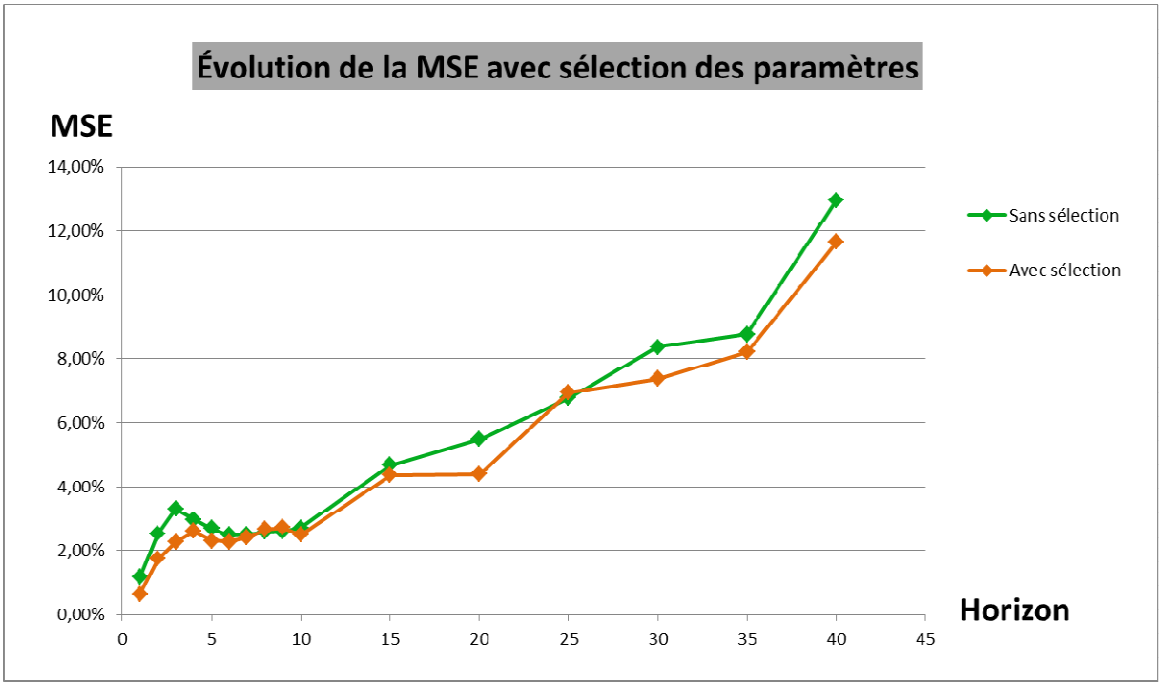


Figure 26 : Accroissement de la MSE avec sélection des paramètres

À quelques exceptions près, toutes les valeurs de la MSE sans sélection de paramètres se sont vues diminuer lorsque le processus de sélection « Grid Search » a été utilisée, particulièrement sur les grands horizons de prédiction.

⇨ Le bon choix des paramètres d'apprentissage influence donc positivement et significativement les résultats des prédictions.

7.3.3. Prédiction avec Boosting : Évolution de la MSE avec de h+1 à h+40:

L'objectif du projet à la base est la mise en évidence de l'algorithme de Boosting dans le processus de prédiction, c'est ce qui a été expérimenté à ce niveau : Ce test a été réalisé sur le même jeu de donné utilisé dans les expériences antérieures, avec un apprentissage itératif par Boosting, toujours pour une taille de fenêtre égale à 5 et en utilisant les paramètres optimaux utilisés lors du test précédent. Sachant que le Boosting est une suite d'apprentissages itératifs, la sélection des paramètres d'apprentissage est toute aussi importante.

Comme expliqué dans les chapitres premiers, la version Drucker de l'algorithme de Boosting inclut des fonctions coût supplémentaires par rapport à l'algorithme classique Adaboost.R, ainsi les tests ont été lancés arbitrairement avec les 3 fonctions coût de l'algorithme, à savoir : linéaire, quadratique et exponentielle. Des tests plus organisés doivent être entrepris à ce niveau, par manque de temps ceci n'a pas pu être réalisé.

Le deuxième paramètre Boosting pris en considération est le pourcentage d'éléments à sélectionner lors de la génération des nouvelles distributions (sous-ensembles d'apprentissage). Des valeurs variant de 10% à 70% ont été appliquée, de même des tests spécifiques au pourcentage doivent être faits ultérieurement.

Pour chaque jeu de données (de h+1 à h+40) une vingtaine d'exécutions ont été réalisées, vu la présence d'un Random dans l'algorithme de Boosting, les valeurs prédites avec Boosting changent d'une exécution à une autre, forçant la MSE à changer par conséquent. Le tableau rassemble les meilleurs résultats de boosting obtenus :

Horizon	**h+1**	**h+2**	**h+3**	**h+4**	**h+5**	**h+6**	**h+7**	**h+8**
MSE sans Boosting	0,00634	0,01743	0,02266	0,02608	0,02298	0,02279	0,02399	0,02663
MSE avec Boosting	0,00649	**0,01698**	**0,02194**	**0,02529**	0,02524	**0,02101**	0,03048	**0,02509**

Horizon	**h+9**	**h+10**	**h+15**	**h+20**	**h+25**	**h+30**	**h+35**	**h+40**
MSE sans Boosting	0,02725	0,02489	0,04363	0,04386	0,06944	0,07377	0,08212	0,11650
MSE avec Boosting	**0,02246**	**0,02469**	**0,03685**	**0,03716**	**0,05017**	**0,05112**	**0,06212**	**0,07120**

Tableau 4 : Évolution de la MSE avec Boosting

Les valeurs en rouge représentent les MSE qui ont été améliorées lors de l'exécution de l'application en mode Boosting, la plupart des valeurs ont pu régresser, de façon très significative pour les grands horizons de prédiction.

En traduisant les tableaux en graphes et en les comparants aux résultats des prédictions avec et sans sélection de paramètres, on obtient l'évolution ci-après :

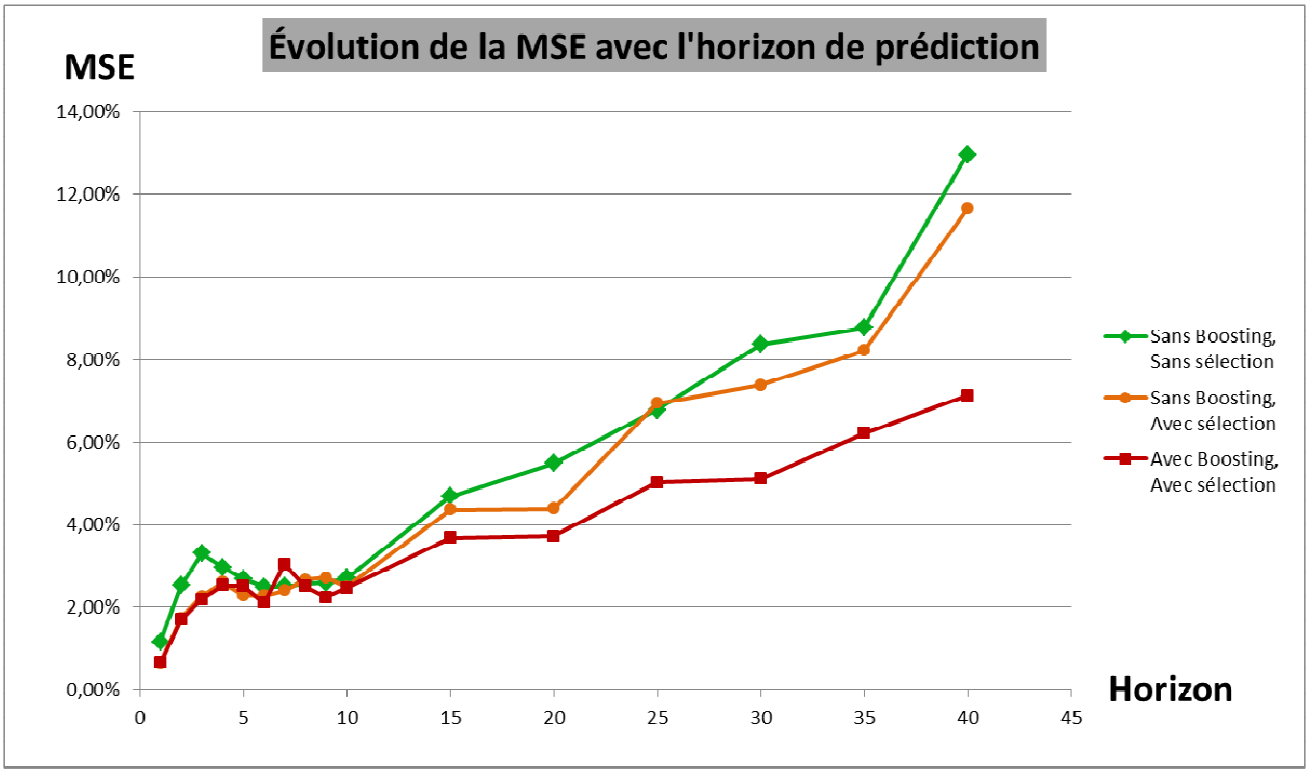


Figure 27 : Évolution de la MSE avec Boosting

Le graphique traduit bien l'amélioration de la qualité des prédictions lors de l'utilisation de la sélection de paramètres, et encore mieux après l'application du Boosting.

Observations:

⇨ Bien que les MSE aient pu changer au cours des 20 tests effectués, elles se sont révélées toujours meilleures que celle commises sans Boosting, et ce pour les horizons de prédiction allant de h+10 à h+40 (≈ 19 fois sur 20 les tests ont donné une valeur MSE meilleure à celle sans Boosting).

⇨ La faible amélioration apportée par le boosting au niveau des petits horizons (h+1 à h+10), peut être expliquée par le fait que les résultats soient déjà optimaux, ne pouvant améliorer le meilleur le boosting réalise des résultats aussi bons voire plus mauvais que ceux faits sans boosting. Étant des apprenants Forts (Strong learner) les SVM réalisent déjà de très bonnes performances en généralisation, ces derniers s'affaiblissent avec l'horizon de prédiction, permettant dans ce cas au boosting d'apporter une amélioration, ce qui expliquerait les bonnes performances Boosting de h+10 à h+40.

⇨ Vraisemblablement, les fonctions de cout quadratiques et exponentielles donnent des résultats meilleurs que ceux donnés par la fonction linéaire. Quant au pourcentage d'éléments sélectionnés, plus il est petit meilleure est la prédiction, une amélioration a été observée en oscillant à chaque fois entre des pourcentages allant de 10% à 70%.

Ces observations doivent néanmoins faire l'objet de plusieurs tests plus poussés pour une meilleure mise en évidence.

7.3.4. Comparaison de l'apprentissage sur fichiers Sunspot test1 et test2

Un dernier test a été réalisé sur un deuxième fichier Sunspot différent du premier, dans la mesure où il rassemble des exemples réputés plus difficiles à apprendre, le tableau rassemble les résultats de Boosting obtenus :

Horizon	h+1	h+2	h+3	h+4	h+5	h+6	h+7	h+8
MSE sans Boosting	0,00634	0,01743	0,06455	0,07878	0,06589	0,05876	0,04602	0,05055
MSE avec Boosting	0,00659	**0,01561**	**0,04696**	**0,05491**	0,06828	**0,05193**	0,05844	0,05622

Horizon	h+9	h+10	h+15	h+20	h+25	h+30	h+35
MSE sans Boosting	0,25981	0,05535	0,13488	0,08085	0,08277	0,08788	0,12057
MSE avec Boosting	**0,10623**	**0,05446**	**0,08999**	**0,04434**	**0,04907**	**0,05776**	**0,09610**

Tableau 5 : Boosting sur exemples difficiles

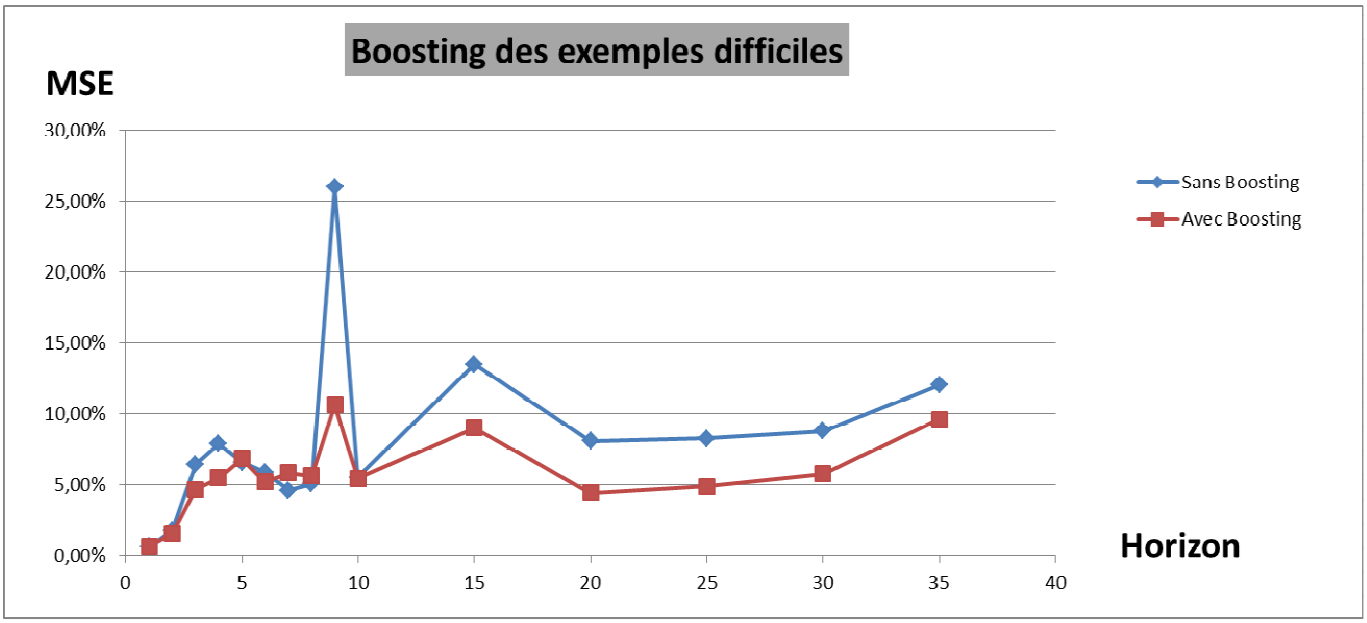


Figure 28 : Boosting sur exemples difficiles

De même une amélioration est faite de h+8 à h+35, une interprétation possible du graphique serait que les performances du boosting ne dépendent pas uniquement de l'horizon de prédiction mais aussi de la difficulté de l'exemple, la prédiction sur h+9 a été très mauvaise elle a pu être améliorée tout de même, par contre celle à h+10 ne l'a pas été malgré qu'elle soit plus avancée dans le temps.

CONCLUSION

L'objectif de ce projet était la mise en œuvre d'un outil logiciel, permettant de combiner la méthode SVR et l'algorithme de Boosting afin d'améliorer les résultats des prédictions des séries temporelles sur des horizons de prédiction étendus.

Le présent document a rappelé des notions théoriques dans le domaine de l'apprentissage automatique, notamment le fonctionnement des SVM et de l'algorithme de Boosting. Il a présenté l'application réalisée dans le cadre de ce PFE, et s'est achevé en exposant les expérimentations faites pour la mise en évidence de l'effet du Boosting sur les prédictions temporelles.

Les expériences menées ont démontré qu'en utilisant l'algorithme de Boosting pour combiner les modèles en apprentissage par SVM, il existe effectivement une amélioration des performances au niveau des prédictions. Cette amélioration est particulièrement observée sur des horizons de prédiction étendus. De même, les tests ont affirmé le rôle de la sélection des paramètres d'apprentissage dans le processus.

Bien que que les expériences effectués semblent converger vers le bon sens, le nombre de tests effectués n'est toutefois pas suffisant pour conclure correctement sur les résultats, d'autres expériences doivent être menées sur d'autres jeux de données, en diversifiant de façon organisée le paramétrage des algorithmes SVM et Boosting.

Les 6 commandements:

Ci-dessous un ensemble de remarque pour les personnes qui reprendraient éventuellement la recherche sur ce sujet :

1- Revoir le codage de l'algorithme de Drucker :

Malgré de nombreuses révisions et débogages, il est possible que le code contienne des erreurs d'inattention ou d'incompréhension de l'algorithme, à revoir.

2- Se pencher d'avantage sur la sélection de paramètres :

Il est important d'étudier plus profondément les paramètres SVM, éventuellement prendre le temps de comprendre le fonctionnement du « Grid-search » de la LIBSVM qui recherche les paramètres kernel optimaux. En plus du « Grid-search » une autre façon de faire est proposée par la LIBSVM dans

le cadre de la sélection de paramètres : c'est d'utiliser le fichier d'apprentissage en argument de la fonction svm-predict, pour pouvoir afficher le taux d'erreur durant la phase d'apprentissage, et savoir ainsi si l'apprentissage en lui-même s'est bien déroulé, en d'autres termes si les paramètres utilisés sont optimaux, puis de les améliorer en conséquence avant de lancer la phase de prédiction.

3- Etudier les articles affirmant l'impossibilité de combiner le Boosting aux SVM :

Selon plusieurs recherches, les SVM étant des classifieurs/régresseurs forts (Strong Learner) ne peuvent pas servir d'apprenant pour l'algorithme de Boosting qui nécessite un (Weak Learner), des articles ont été publiés dans ce sens, entre autre :

- Boosting Support Vector Machines Successfully , Kai Ming Ting and Lian Zhu.
- "Support Vector Machines and Their Application in Chemistry and Biotechnology" , Chapter 4 , Dong-Sheng Cao
- "Boosting Support Vector Machines" , Elkin García et Fernando Lozano.

La version SVM qui a été utilisée dans le cadre de ce projet est une version « Forte » vu que c'est celle de la LIBSVM. Il existe sur le net une version LIBSVM qui a été « affaiblie », à rechercher.

4- Optimisation du code :

Aucune étude particulière n'a été faite au niveau de la complexité du code et des temps d'exécution, l'application est plutôt rapide comparée à des applications sous réseaux de neurones, cependant sur des jeux de données importants cette rapidité peut être compromise.

5- Améliorer l'interface du programme :

L'intérêt du projet étant plus centré sur le développement de l'outil, l'ergonomie de l'interface graphique n'a pas été particulièrement étudiée, et n'est donc pas « User friendly », à remanier.

6- Gestion de projets :

Dans ce PFE les tests ont été insérés à la fin des phases de développement, c'est la séquence logique cependant il serait utile de consacrer aux tests toute une phase dans le projet, s'étalant sur plusieurs mois.

GLOSSAIRE

- Boosting: Méthode permettant, sous certaines conditions, d'améliorer les performances de n'importe quel algorithme d'apprentissage. Il combine linéairement des hypothèses dites faibles ou weak hypotheses (i.e. juste meilleures qu'un tirage aléatoire) en une hypothèse dite forte ou strong hypothesis.
Source : www.lif.univ-mrs.fr/~capponi/epit2008/lib/exe/fetch.ph

- SVM (Support Vector Machines): Machines à Vecteurs de Support ou séparateurs à vaste marge, sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classifieurs linéaires. Elles ont été appliqués à de très nombreux domaines (bioinformatique, recherche d'information, vision par ordinateur, finance...). Selon les données, la performance des machines à vecteurs de support est de même ordre, ou même supérieure, à celle d'un réseau de neurones ou d'un modèle de mixture gaussienne.
Source : http://fr.wikipedia.org/wiki/Machine_à_vecteurs_de_support

- SVR (Support Vector Regression) : Machines à Vecteurs de Support pour la régression, c'est l'application de la méthode SVM pour des tâches de régression, c'est à dire prédiction d'une variable continue en fonction d'autres variables, comme c'est le cas par exemple dans la prédiction de consommation électrique en fonction de la période de l'année, de la température, etc…
Source : http://georges.gardarin.free.fr/Surveys_DM/Survey_SVM.pdf

REFERENCES BIBLIOGRAPHIQUES

Documents:

- Vapnik V. N. , « **The nature of statistical learning theory** », Springer, 2000.
- Drucker Harris, « **Improving Regressors using Boosting Techniques** », Monmouth University - West Long Branch.
- Freund Yoav and Schapire Robert E. (1999), « **A Short Introduction to Boosting** », Journal of Japanese Society for Artificial Intelligence 14(5):771-780.
- Chih-Chung Chang and Chih-Jen Lin (2011), « **LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology** », University of Taiwan.
- Mohammad Assaad (2006), Thèse : « **Un nouvel algorithme de Boosting pour les réseaux de neurones récurrents : Application au traitement des données séquentielles** », Polytech' Tours.
- Clément PUËLL (2009), Notes : «**Cours magistral de Reconnaissance des formes, dirigé par Hubert CARDOT** », Polytech' Tours.
- Emmanuel Cesar & Bruno Richard (2006), Cours : « **Les Séries Temporelles** », Université de Versailles Saint-Quentin-en-Yvelines Module XML et Data Mining.
- Sébastien LECHEVALIER, Cours : « **Une introduction à l'économétrie des séries temporelles** », Université Paris-I.

Sites Web:

SVM :
- http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- http://wikistat.fr/

Boosting :
- http://www.boosting.org

Forum Learning machines :
- http://agbs.kyb.tuebingen.mpg.de

Ressources documentaires sur les logiciels utilisés:
- http://www.jfree.org/jfreechart/
- http://www.eclipse.org/windowbuilder
- http://jmdoudoux.developpez.com

Résumé :

Ce document est un compte rendu du projet de fin d'études intitulé « SVR avec boosting pour la prévision à long terme, il décrit la mise en œuvre d'un outil logiciel, permettant de combiner la méthode SVR et l'algorithme de Boosting afin d'améliorer les résultats des prédictions sur les séries temporelles pour des horizons de prédiction étendus.

Mots-clés : SVR, SVM, Boosting, prédiction, séries temporelles

Abstract:

This document is a review of the graduation project entitled « SVR with boosting for long-term forecasting ». It describes the implementation of a tool that combines SVR method with boosting algorithm to improve time series forecasting for large forecasting horizons.

Keywords: SVR, SVM, Boosting, forcasting, time series

Encadrant :

CHERIF Aymen
Doctorant au DI Polytech' Tours
64 avenue Jean Portalis 37200 Tours
aymen.cherif@etu.univ-tours.fr

Etudiants :

EL HASSANI Imane
imane.elhassani@etu.univ-tours.fr