Collecte des données et Construction du modèle

Contents

Introduction **135**
6.1 Collecte des données **135**
6.2 Construction du modèle **137**
6.2.1 Étapes menant à la création du modèle 139
6.2.2 Segmentation des étapes de calcul du modèle 142
6.3 Interface de comparaison des modèles **144**
6.4 Implémentation du service web **145**
6.4.1 Rapidité 148
6.4.2 Clarté 148
6.4.3 Reproductibilité 148
6.5 Conclusion et perspectives **148**

Introduction

Dans les chapitres précédents, nous avons décrit l'ensemble de la partie théorique du projet et les résultats observés sur des bases statiques extraites de celle mise à jour en temps réel. Le service, destiné à être mis en ligne, doit répondre à des exigences de mise en production qui influencent le choix de nos paramètres. Les critères pris en compte sont la rapidité de la génération du modèle impliquant l'utilisation d'outils adaptés à l'optimisation logicielle, le partage de ce modèle pour qu'il puisse être interrogeable à chaque requête utilisateur et par n'importe quelle technologie et enfin le contrôle de la qualité des prédictions passant par le stockage et l'analyse régulière de celles-ci. Nous devons gérer une quantité importante de données, les conserver et les exploiter pour enfin les présenter clairement à l'internaute.

Dans ce dernier chapitre, nous décrivons l'implémentation des modules de notre service et la manière dont les différentes étapes s'articulent. L'architecture globale peut se découper en quatre parties qui sont : (i) la création du modèle de prédiction en R, incluant la transformation des données, la segmentation de l'ensemble des vols d'entraînement et l'apprentissage des comportements aboutissant à la construction d'un modèle exportable en PMML. (ii) La création d'un module de comparaison des performances des modèles afin de sélectionner le meilleur ensemble de paramètres. (iii) L'implémentation d'un service web qui charge en mémoire le modèle précédemment créé ainsi que les simulations, reçoit les résultats des recherches utilisateurs et renvoie pour chacun une prédiction. (iv) Le stockage des prédictions et l'évaluation à la volée de la qualité du moteur de prédiction afin d'enclencher le renouvellement du modèle lorsque les performances sont trop faibles.

Nous décrivons les problématiques rencontrées et les solutions que nous avons envisagées afin de créer un service clair, rapide et de qualité. Nous axons notre modélisation sur l'indépendance des composants applicatifs afin de rendre chaque sous-ensemble évolutif sans modifier l'ensemble de l'architecture : nous pouvons ainsi par exemple passer l'historique des prix en *NoSQL*, ou migrer la construction du modèle sous *Apache Mahout*.

Nous rappelons que nos données sont entièrement issues des recherches utilisateurs. Chacune de ces recherches déclenche un enchaînement d'actions impactant l'ensemble des briques applicatives du service de prédiction :

1. A la suite d'une recherche utilisateur, nous stockons les données récoltées dans une base de log que nous avons adaptée afin de conserver l'ensemble des résultats.

2. Ces données sont ensuite formatées pour respecter la structure de la base de données du service de génération des modèles décrite dans le premier chapitre et sur la Figure 6.1.

3. Chaque ligne de la page de résultat possède un bouton "Détails" faisant une demande de prédiction. Cette demande va interroger le service XML du module de prédiction qui va renvoyer l'ensemble des informations nécessaires à la présentation du module d'aide à la décision : la probabilité d'évolution du billet d'avion sélectionné, le taux de confiance et

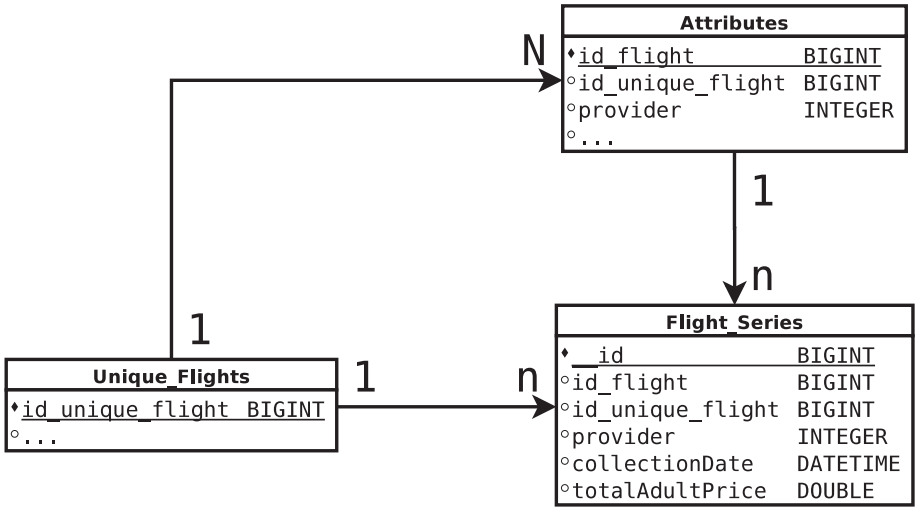

FIGURE 6.1 – Structure de la base de données sur service de génération du modèle

la fourchette de prix du futur tarif. Une base de données des vols encore actifs (dont la date de départ n'est pas encore passée) est interrogée et retourne les données nécessaires à l'affichage des variations de prix passées de la série temporelle. Les prédictions sont enfin stockées en base.

4. Afin d'évaluer les performances de notre service de prédiction, nous vérifions à chaque recherche si une prédiction a été faite 7 jours avant et le cas échéant, nous vérifions et stockons la qualité du conseil. Avec le système d'alerte qui effectue quotidiennement la même recherche et conserve les 5 meilleures offres, nous sommes capable d'évaluer plus de 30% de nos prédictions. En ajoutant les recherches utilisateurs, nous constituons un ensemble représentatif suffisant pour que l'évaluation de nos prédictions reflète la qualité globale de celles-ci.

5. Lorsque le taux de bonnes prédictions baisse en dessous d'un seuil donné, nous lançons la génération d'un nouveau modèle sur la base de données actualisée de l'historique des vols. Celle-ci transmettra alors au service XML du module de prédiction un fichier PMML représentant le modèle de classification et les simulations de tous les centroïdes à tous les temps t.

La Figure 6.2 résume l'ensemble du cycle de vie d'un modèle et reprend les quatre composants représentés sur le diagramme de séquence de la Figure 6.3 : le moteur de recherche recevant les appels utilisateurs et les enregistrant en base de données, le service de web effectuant les demandes de prédictions et le générateur de modèle utilisant l'historique des vols. Les zones de couleurs représentent l'ensemble des parties créées ou modifiées pour le service de prédiction.

Un des points clef du service est le renouvellement du modèle et la détection d'une diminution des performances. Cependant le processus de création du modèle étant coûteux en temps et en

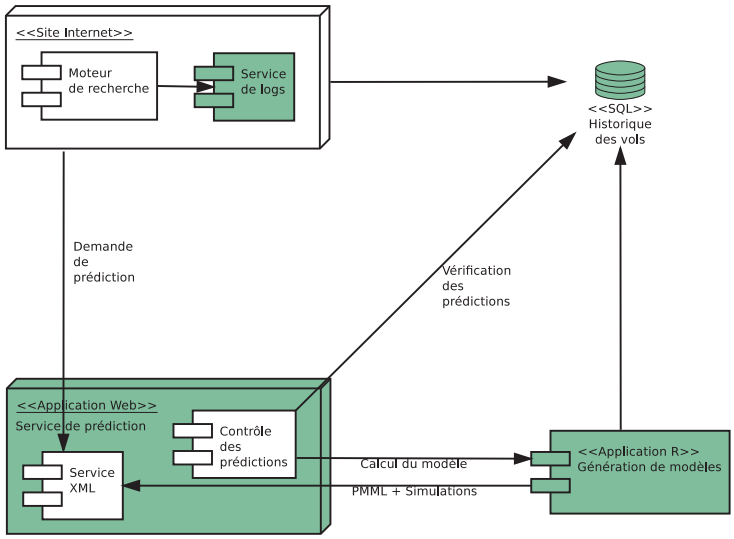

FIGURE 6.2 – Diagramme de déploiement : Composants applicatifs

ressource, il est important de limiter son exécution. Comme expliqué dans le chapitre précédent, il est important de différencier le pourcentage global de bonnes prédictions et l'économie moyenne faite à l'utilisateur. Dans le faits, nous privilégierons la somme économisée par l'utilisateur en observant l'économie faite grâce aux prédictions mais aussi l'écart de cette économie par rapport à celle faite lors de l'achat immédiat systématique. Notre service n'a de sens que s'il fait économiser plus d'argent que la solution "conservatrice". Si le gain réalisé grâce aux bonnes prédictions passe sous la barre du gain à l'achat immédiat, le modèle doit être mis à jour.

6.2 Construction du modèle

La construction du modèle est décrite dans les Chapitres 2, 3 et 4. Les différentes étapes sont synthétisées sur la Figure 6.4 et détaillées ci dessous :

1. Tout d'abord nous stockons à la volée les données de recherche dans les tables précédemment décrites : *Unique_Flights*, *Flight_Series* et *Attributes*.
2. Après avoir extrait les vols consistants, nous créons une base d'apprentissage des niveaux de gris M_{train} grâce à la transformation des séries temporelles. Cette transformation se fait suivant les paramètres b_t et b_r décrivant les dimensions des fenêtres de l'image pixélisée.
3. Nous segmentons ensuite la base des niveaux de gris $X_i(s,t)$ en utilisant les algorithmes décrits dans le Chapitre 3. Les centroïdes, les paramètres des modèles et l'attribution des clusters aux vols de la base d'apprentissage sont enregistrés en base pour pouvoir les utiliser ultérieurement.
4. Nous générons alors les φ_t pour tous les clusters et tous les t possibles grâce aux simulations créées à partir des centroïdes précédemment calculés.

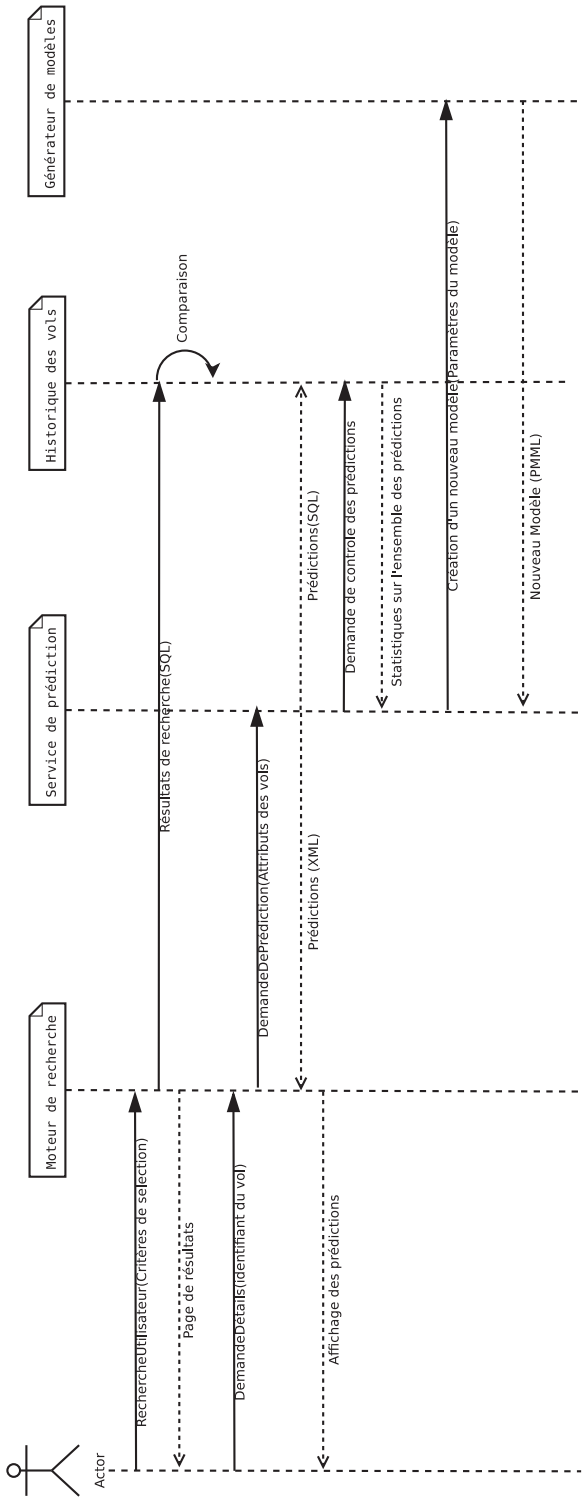

FIGURE 6.3 – Diagramme de séquence

5. Enfin nous créons le modèle de classification en appliquant les algorithmes d'apprentissage des comportements à partir des attributs V_i stockés dans la base N_{train}. Nous exportons ce dernier sous le format PMML pour qu'il puisse être ensuite interrogé par le service de prédiction.

Chacune de ces étapes est indépendante, c'est-à-dire qu'elle possède son paramétrage propre et peut être exécutée dans différentes configurations sans avoir à ré-exécuter les étapes précédentes. L'avantage est de pouvoir tester facilement différents paramètres et ainsi identifier plus facilement les meilleures configurations.

Nous allons maintenant détailler les difficultés rencontrées lors de la création d'un modèle et les solutions que nous avons mises en œuvre.

6.2.1 Étapes menant à la création du modèle

La création d'un modèle est la génération d'un objet interrogeable par le service de prédiction du site liligo.com. Cet objet est capable de fournir pour chaque ligne de la page des résultats, une prédiction du groupe le plus probable, au vue des paramètres du vol. Les simulations de chaque cluster, préalablement effectuées, nous fournissent les φ_t pour tous les t et tous les clusters. Le tableau des φ_t est alors mis en mémoire et associé à la prédiction de groupe pour afficher au final une prédiction de comportement.

Si le taux de bonnes prédictions vient à diminuer, il est important de pouvoir re-générer ce modèle dans des délais raisonnables en gardant la même qualité de prédiction. Nous avons donc utilisé le logiciel *R*, équivalent libre de *Matlab*, dont l'importante communauté permet d'avoir accès à de nombreux algorithmes d'intelligence artificiel et à des optimisations de code permettant la manipulation d'importants volumes de données en des temps raisonnables.

Optimisation du code

La problématique de la création rapide du modèle de prédiction provient du volume de données à traiter : la manipulation de grande matrice utilise une grande quantité de ressources, à la fois en mémoire et en nombre de cœurs, interdisant ainsi certaines opérations tout en demandant un temps de calcul considérable. Par exemple, lors de la segmentation, les niveaux de gris sont "aplatis" pour avoir un vecteur pour chaque vol dont chaque valeur représente l'intensité d'une case. La matrice M_{train} dont chaque ligne correspond à un vol et chaque colonne à une case de l'image pixélisée, peut alors obtenir une taille de 10000×198.

Nous avons donc utilisé différents packages de *R* pour pouvoir manipuler ces matrices et appliquer des algorithmes de segmentation dans des temps raisonnables :

1. La librairie *bigmemory* permet la création, le stockage et l'accès à de très grandes matrices, celles-ci étant allouées à la mémoire partagée. Le partage de mémoire est un moyen de partager des données entre différents processus en permettant à plusieurs processus d'accéder à une même zone de la mémoire vive.

2. Une librairie connexe (*biganalytics*), implémente une version de l'algorithme de clustering K-Means nommée Big-K-Means, utilisant cette implémentation des matrices, évitant ainsi

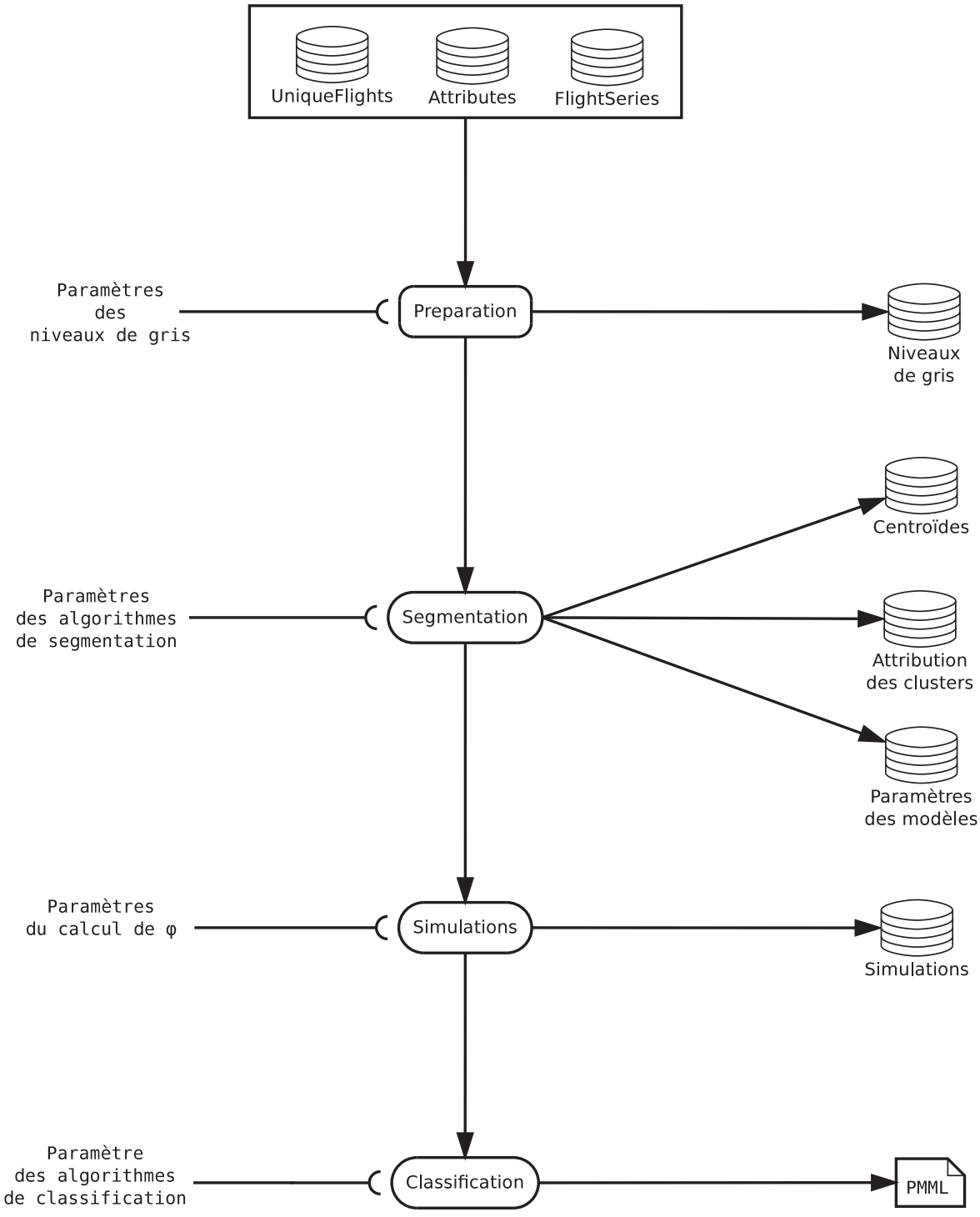

FIGURE 6.4 – Création d'un modèle

la surcharge de données en mémoire et autorisant la distribution des différentes initialisations sur l'ensemble des cœurs disponibles. Cet algorithme ne nécessite pas de mémoire supplémentaire autre que les données tandis que l'algorithme usuel fait au moins deux copies supplémentaires de la matrice de données. Un choix de plusieurs départs aléatoires s'il n'est pas exécuté en parallèle utilise aussi beaucoup de ressources et de temps et nécessité de stocker autant de vecteurs d'adhésions aux clusters que de départs.

3. Cette parallélisation des processus se fait grâce au paquet *foreach* capable d'exploiter en parallèle l'ensemble des cœurs d'un ordinateur. Il est possible de coupler *foreach* avec de nombreux autres paquets de *R* comme celui des forêt aléatoires, permettant de paralléliser la création des arbres et de combiner par la suite les différentes forêts entre elles.

Export des arbres de décisions en un format standard

R n'offrant pas les garanties de stabilité requises à une mise en production, la possibilité pour une application grand public de déployer son service sur une plateforme de Cloud Computing est restreinte. C'est pourquoi nous avons dû trouver un format de fichier vers lequel exporter notre modèle de classification afin d'être utilisé par nos serveurs lors d'une demande de prédiction. Ce format devait être neutre et standard.

La création d'un fichier au format neutre, indépendant de tout langage de programmation, a plusieurs autres avantages. Tout d'abord elle permet de conserver une indépendance entre les composants applicatifs : il n'y a pas besoin de faire dialoguer *R* et l'interface web pour effectuer les prédictions. Ensuite, si l'on souhaite changer la technologie derrière la génération du modèle (par exemple, utiliser l'API *JAVA Weka* ou le framework *Apache Mahout* qui implémente des algorithmes de fouille de données distribués sur la plateforme *Apache Hadoop*), la partie cliente du service n'a pas a être modifiée. Enfin la création de fichiers interrogeables, nous permet de multiplier le nombre de modèles pour affiner la prédiction en fonction du nombre jours avant la date de départ. En effet, nous avons constaté des différences de performance entre le modèle créé sur les vols de 90 jours et celui sur les vols de 28 jours. Deux modèles sont alors créés et interrogés en fonction de la date de demande de prédiction. Par ailleurs, chaque modèle pourra avoir ses propres paramètres d'interprétation de la probabilité $\mathbb{P}(\varphi = 1)$, ajoutant encore de la flexibilité à notre service.

Predictive Model Markup Language ou *PMML* est un langage de marquage basé sur *XML* conçu pour définir des modèles de données et visant à rendre inter-opérables les systèmes de data-mining. Nous utilisons ce langage pour exporter l'arbre de décision généré par l'algorithme CART ou les forêts d'arbres décisionnels. Nous importons par la suite cet arbre de décision dans un module web qui sera interrogé à chaque recherche utilisateur.

L'inconvénient d'un format issu de l'XML, est la taille du fichier dû à la verbosité du langage, et donc au temps de création du fichier. La parallélisation n'est pas possible et le temps de calcul dépasse souvent 3 heures. Ce temps est par ailleurs indexé sur le nombre d'arbres créés si l'on a choisi les forêts aléatoire, rendant le choix de ce paramètre délicat lors de la création du modèle.

Enfin tous les algorithmes ne sont pas encore pris en charge par *PMML*, comme Adaboost, ou par l'implémentation *PMML* d'*R* comme l'algorithme C4.5.

6.2.2 Segmentation des étapes de calcul du modèle

Notre approche a nécessité de nombreux tests et de multiples combinaisons de paramètres, nécessitant d'effectuer les mêmes étapes de calcul pour toutes les combinaisons. Pour éviter d'effectuer toutes les étapes coûteuses en temps (création des niveaux de gris, clustering etc.) lors du changement d'un paramètre sur le nombre d'arbre dans les random forest par exemple, nous avons décorrélé ces étapes et stocké tous les résultats intermédiaires dans des tables, que l'on interroge par la suite. Cela nous permet également de conserver des statistiques sur tous les algorithmes testés.

Si l'on reprend la Figure 6.4, nous constatons clairement cette séparation ainsi que les différentes données conservées en base afin d'être réutilisées à l'étape suivante :

1. **Création des niveaux de gris** : Pour chaque vol nous stockons en base une version compressée du niveau de gris dans différentes configurations : $b_r = \{1, 0.1, 0.05, 0.01\}$ et $b_t = \{1 \times 24, 3 \times 24, 5 \times 24\}$.

2. **Clustering** : Selon la configuration souhaitée, nous chargeons les niveaux de gris appropriés sous la forme d'une matrice. Une fois la segmentation effectuée, nous stockons les statistiques de cette dernière, les centroïdes et l'étiquette E_i associée à chaque vol.

3. **Simulations** : Le choix du critère de prédiction n'intervient qu'au dernier moment, car l'apprentissage se fait sur le comportement global et non sur φ. Comme nous effectuons les simulations à l'avance pour pouvoir les interroger à la volée, nous devons statuer de la prédiction à faire en amont. Nous effectuons alors pour chaque t possible 1000 simulations, et calculons un version simple et une version par cadran de φ_t que nous stockons ensuite en base. Celle-ci sera alors chargée en mémoire sur les machines de production en vue d'être associées à la prédiction d'un identifiant de cluster et de fournir une prédiction d'évolution de prix. Cette méthode évite de devoir effectuer les simulations à la volée et garantit une reproductibilité des prédictions.

4. **Classification** : Nous chargeons la matrice N_{train} contenant l'ensemble des attributs des vols, sous la forme d'une matrice à laquelle nous ajoutons une colonne représentant l'étiquette du vol créée à l'étape précédente. Une modèle de classification est alors créé et exporté au format R (pour une utilisation locale) et PMML (pour être utilisée en production).

Identification unique des configurations

La multiplication des combinaisons possibles d'algorithmes et de leurs paramètres, a nécessité la création d'un système d'identifiants uniques imbriqués pour chaque étape, correspondant à un jeu de paramètres. La Figure 6.5 nous montre les différents identifiants existants et la Figure 6.6 est une visualisation simplifiée de la hiérarchie ainsi créée.

Nous avons dans pour la première étape de transformation l'identifiant correspondant à la configuration des niveaux de gris. Puis l'identifiant de la longueur des séries qui va définir avec

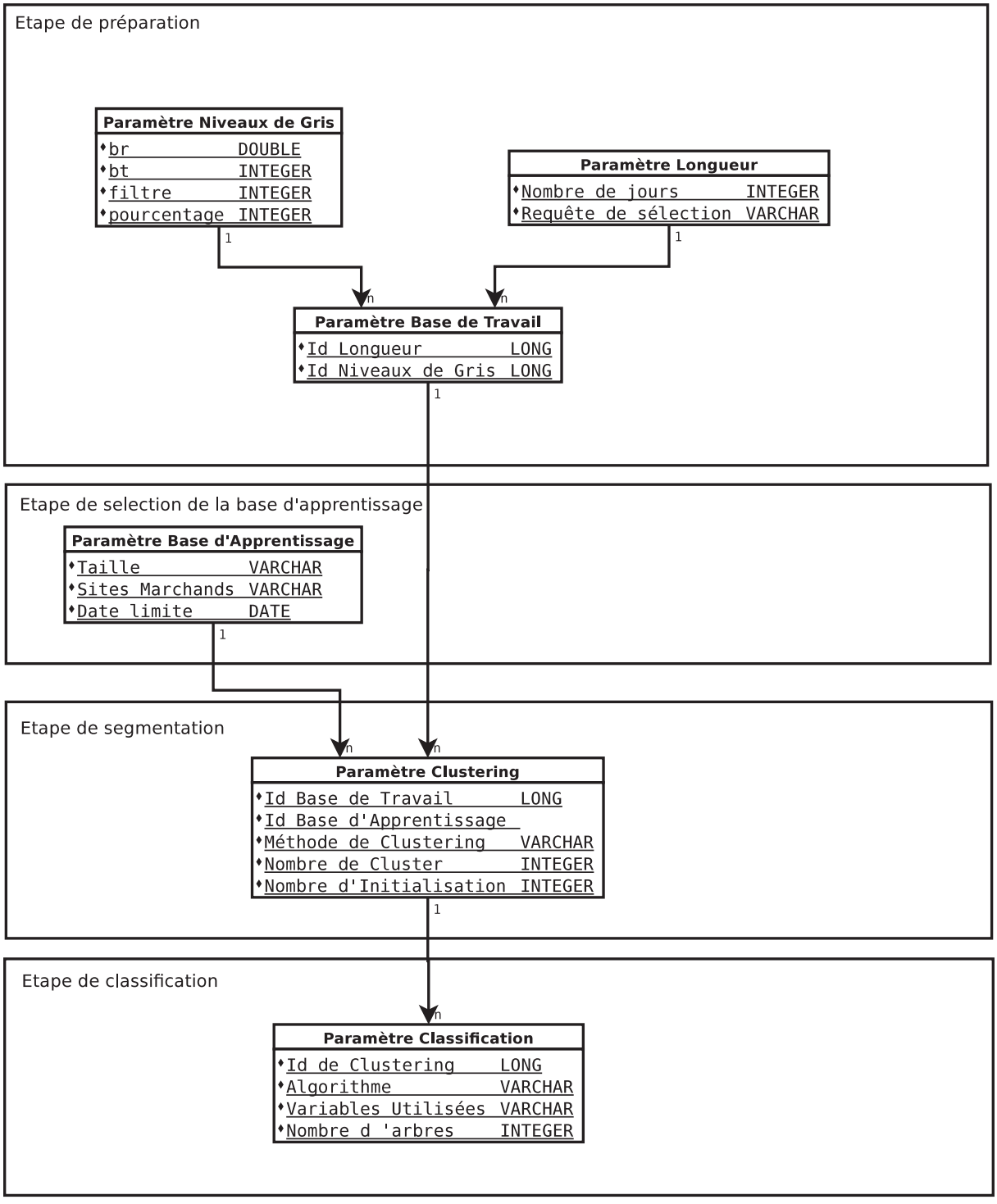

FIGURE 6.5 – SchemaDB

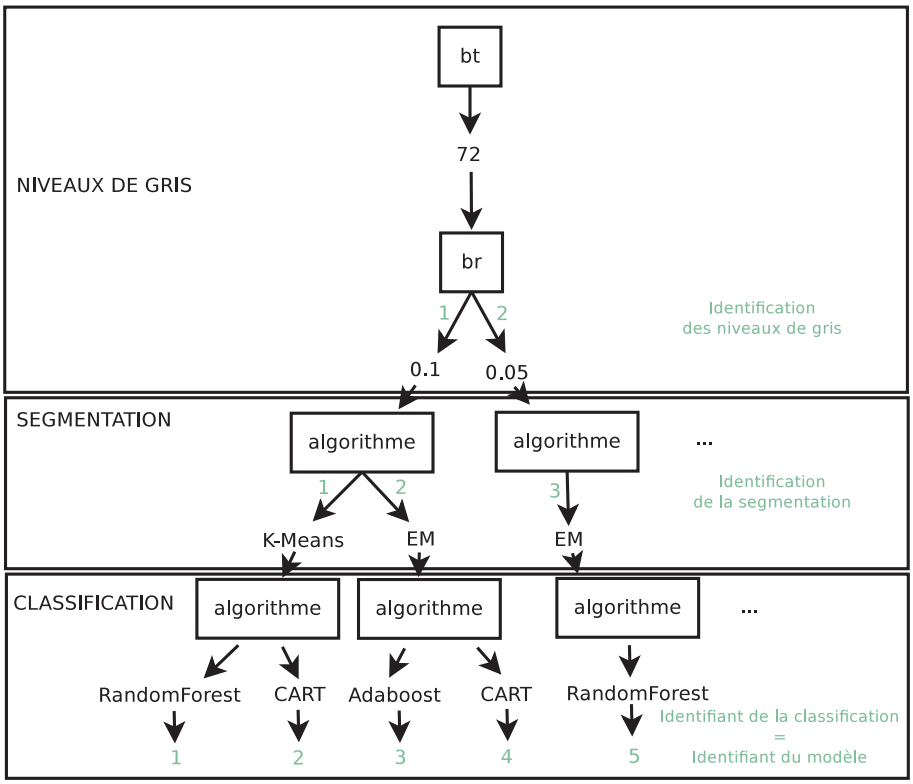

FIGURE 6.6 – Identifiants

la "requête de sélection" la liste des vols consistants. Enfin un table regroupant ces deux identifiant définit l'ensemble des niveaux de gris de la base d'apprentissage M_{train} et une dernière sélectionne dans cette base le nombre de vols étudié et filtre sur les sites marchands si besoin.

Vient ensuite la configuration de la segmentation qui comprend l'algorithme utilisé et ses paramètres et enfin la configuration de la classification de manière identique.

6.3 Interface de comparaison des modèles

Pour permettre la comparaison rapide de nos modèles, nous avons créé une interface permettant d'afficher simultanément les performances des différents modèles selon plusieurs critères (Figure 6.7). Il est possible d'afficher les performances d'autant de modèles que l'on souhaite ainsi que de modifier à la volée certains critères, comme le seuil de séparation des vols à la hausse et à la baisse ou bien la taille et la topologie de la base de test (Figure 6.8).

1. La première étape consiste à sélectionner les paramètres du modèle que l'on souhaite afficher : paramètres des niveaux de gris (b_t, b_r, etc.), les paramètres de la segmentation

(type d'algorithme de segmentation, nombre de groupes, etc.) et les paramètres de la classification (algorithme de classification, variables utilisées, etc.).

2. Il est ensuite possible de sélectionner une base de test spécifique (base des vols consistants, base des vols des recherches utilisateurs ou autre) et pour chacune d'elle filtrer les prédictions par destination ou par site marchand. Nous avons ajouter cette possibilité pour identifier les destinations qui diminuent les performances du service de prédiction ou les sites marchands qui donnent de moins bons résultats. Nous pouvons enfin définir le seuil à partir duquel $\mathbb{P}(\varphi_t = 1)$ est considéré comme une baisse de prix et lancer l'affichage des résultats

3. Tout d'abord pour chaque attribut qualitatif de V_i nous affichons un camembert représentant la répartition des différents niveaux dans l'ensemble des vols bien prédits et celui des vols mal prédits. Là encore, nous voulons pouvoir détecter les situations où la prédiction est moins bonne.

4. Ensuite nous avons trois volets décrivant l'évolution de la bonne prédiction avec le nombre de jours avant la date de départ. Tout à gauche, nous affichons le taux de bonnes prédictions global, au centre celui des vols à la baisse et a droite celui des vols à la hausse. Diviser la prédiction globale en deux permet de vérifier que le prédicteur ne prédit pas tout à la hausse, conservant ainsi une bonne prédiction globale tout en étant complètement inutile.

5. Puis nous affichons les courbes ROC à l'instant T_{-21}. A chaque inflexion de la courbe, nous affichons la valeur du seuil associé ainsi que la distance à la diagonale

6. Nous présentons les statistiques pécuniaires qui sont : la sommes de prix à t, à $t + 7j$, la somme du plus bas prix des deux, et celle du plus élevé. La somme des prix si l'utilisateur suit la prédiction, la différence avec le premier prix et la différence avec l'optimal.

7. Enfin nous affichons sur trois nouveaux volets l'évolution de la somme des gains et des pertes en fonction du nombre de jours avant la date de départ, la somme des gains et la somme des pertes. Dans le premier cadran nous afficherons le gain total si l'utilisateur choisit systématiquement d'acheter à l'instant t et le gain total si l'utilisateur prenant toujours la meilleure solution. Cela permet de situer les performances de notre service du point de vue de l'économie de l'internaute et d'affirmer qu'une aide à la décision permet d'économiser de l'argent face à l'achat immédiat.

6.4 Implémentation du service web

Le processus de prédiction ne doit pas ralentir l'expérience utilisateur et doit donc être décorrélé du retour des résultats sur la page des offres. Il doit donc être interrogé comme un service web. Les informations présentées doivent être claires et permettre à l'utilisateur de comprendre l'aide que l'on lui fournit. La prédiction brute n'est pas suffisante : il est important de donner un indice de confiance et de présenter si possible l'historique des prix sur ce trajet. Ainsi l'internaute sera à même de prendre une décision avec ces outils supplémentaires.

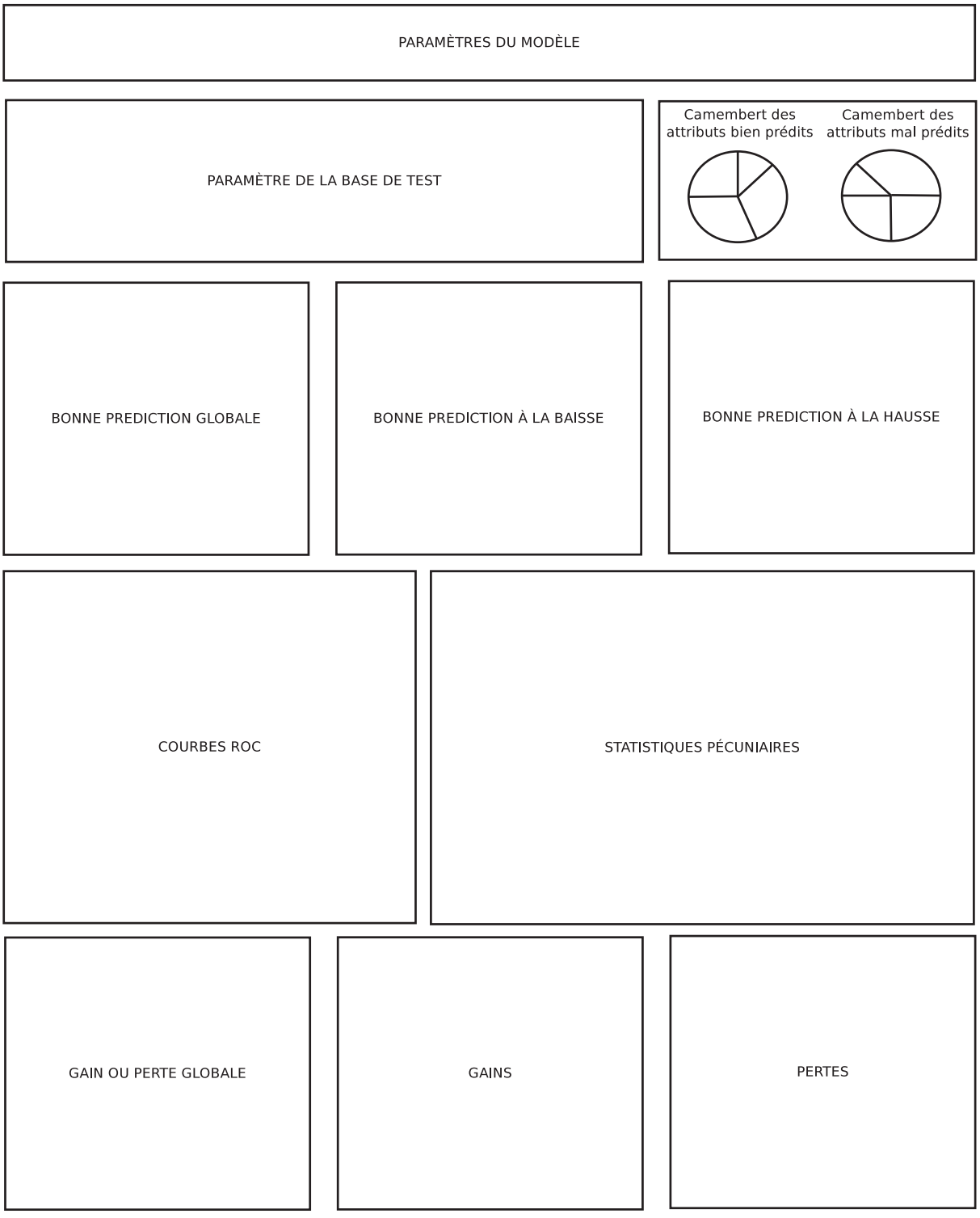

FIGURE 6.7 – Interface de test

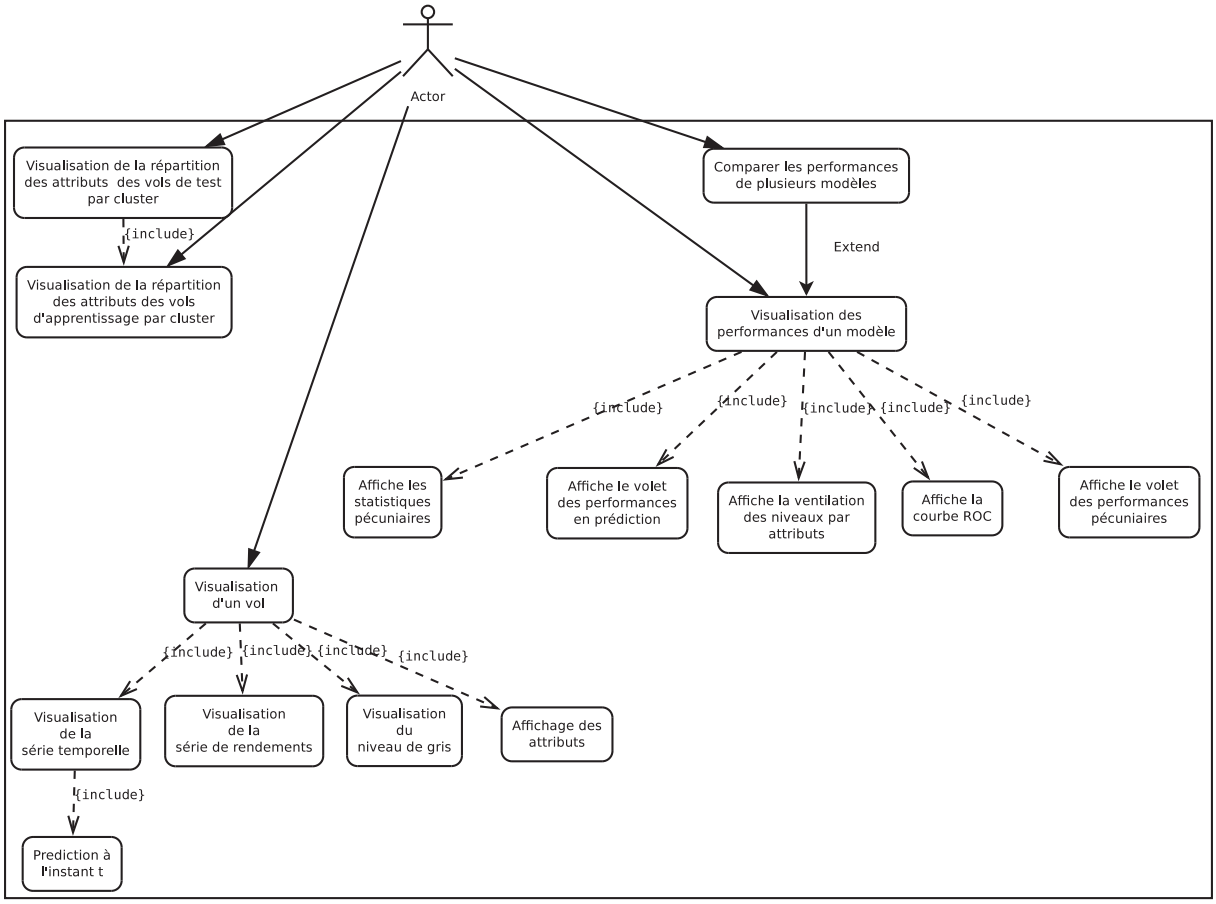

FIGURE 6.8 – Use Case de l'interface de test

6.4.1 Rapidité

Les simulations sont pré-calculées pour toutes les situations possibles : nous calculons φ_t pour les t possible et pour tous les clusters du modèle. Ainsi il est possible de mettre ce tableau $nbCluster \times 90$ (90 étant le nombre de jours maximum que nous gérons actuellement) en mémoire et, une fois la prédiction du groupe faite, extraire le $\mathbb{P}(\varphi_t = 1)$ associé. Les attributs supplémentaires tels que la saison ou le type de site marchand sont calculés aisément à la volée.

6.4.2 Clarté

Pour aider l'utilisateur à prendre sa décision, nous ajoutons au conseil d'achat, la courbe de prix $p_i(t)$ sous la forme d'un graphe et l'évolution présumé de celle-ci. L'évolution prédite correspond à variations du cadran dont la probabilité est la plus importante. Ainsi nous utilisons le pourcentage d'évolution au prix actuel du billet auquel nous ajoutons une variabilité pour enfin afficher une fourchette d'évolution du prix.

6.4.3 Reproductibilité

Enfin il est essentiel de reproduire les prédictions à chaque recherche identique pour ne pas égarer l'utilisateur. Les prédictions étant déterministes, il nous faut maintenant associer une prédiction fixe à chaque cluster et chaque t. Pour cela, ainsi que pour une question de performance, nous avons effectué les simulations de séries à partir des centroïdes du modèle interrogé, et conservé les prédictions issues de ces simulations en mémoire.

6.5 Conclusion et perspectives

Nous avons construit un ensemble de briques applicatives décoréllées permettant la modification des technologie et des algorithmes utilisés sans la refonte complète de l'architecture. Un premier composant, le moteur de recherche, effectue les recherches de vols, log les résultats et lorsque l'utilisateur le souhaite fait une demande de prédiction au second composant. Celui-ci, le service de prédiction, est une application web composé d'un service XML capable de donner des prédictions en fonction d'attributs et de contrôler ses mêmes prédictions 7 jours plus tard. Le fichier XML est généré par le troisième composant, l'application *R*, qui aura utilisé les données historiques stockées dans le dernier composant, la base *SQL*.

L'application *R* a elle aussi était conçu pour permettre une flexibilité et une rapidité d'exécution nécessaire à la génération du fichier PMML. La création de ce fichier se fait par étapes successives, chacune décorélée de la précédente et identifiée par une clef unique dépendante des paramètres de l'étape courante et des étapes précédentes. Ainsi il est possible de combiner plusieurs approches sans réitérer les calculs déjà effectués.

Enfin une interface de test a été implémentée afin de comparer les performances de nos différentes approches. Comme évoqué dans le chapitre précédent certaines problématiques se pose quant au choix de la bonne métrique de comparaison. Selon le seuil choisi les résultats peuvent être très différents et l'équilibre entre le taux de bonnes prédictions et le gain moyen par billets est souvent délicat à trouver. Il serait intéressant d'étudier d'autres mesure de performance afin

de réduire le temps passé à comparer les différents algorithmes.

Nous souhaitons à l'avenir nous migrer la génération des modèles sur des framework open-source tel qu'*Apache Hadoop* et son framework d'algorithme de fouille de données *Apache Mahout* tout comme l'adaptation du modèle au fil de l'évolution des données de la base d'apprentissage. L'augmentation constante de la base de données nécessite par ailleurs l'optimisation de la structure de notre application, comme par l'exemple le passage au *NoSQL* afin d'exploiter plus facilement les informations des premiéres évolutions de prix.

Bibliographie

[1] Wesam Barbakh and Colin Fyfe. Online clustering algorithms. International Journal of Neural Systems, 18(03) :185–194, 2008.

[2] Peter P Belobaba. Survey paper—airline yield management an overview of seat inventory control. Transportation Science, 21(2) :63–73, 1987.

[3] Jürgen Beringer and Eyke Hüllermeier. Online clustering of parallel data streams. Data & Knowledge Engineering, 58(2) :180–204, 2006.

[4] Simon Bernard. Forêts Aléatoires : De l'Analyse des Mécanismes de Fonctionnement à la Construction Dynamique. These, Université de Rouen, December 2009.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA, 1984.

[6] Leo Breiman. Bagging predictors. Machine learning, 24(2) :123–140, 1996.

[7] Leo Breiman. Bagging predictors. Mach. Learn., 24(2) :123–140, August 1996.

[8] Leo Breiman. Bias, variance, and arcing classifiers. 1996.

[9] Leo Breiman. Random Forests. Machine Learning, 45(1) :5–32, October 2001.

[10] Jan K. Brueckner. International airfares in the age of alliances : The effects of codesharing and antitrust immunity. The Review of Economics and Statistics, 85(1) :105–118, 2003.

[11] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. Communications in Statistics-theory and Methods, 3(1) :1–27, 1974.

[12] William W. Cohen. Fast effective rule induction. In In Proceedings of the Twelfth International Conference on Machine Learning, pages 115–123. Morgan Kaufmann, 1995.

[13] D.R. Cox and H.D. Miller. The theory of stochastic processes. Wiley publications in statistics. Wiley, 1965.

[14] D. J. Daley and D. Vere-Jones. An introduction to the theory of point processes. Vol. I. Probability and its Applications (New York). Springer-Verlag, New York, second edition, 2003. Elementary theory and methods.