Apprentissage supervisé - Classification - Prédiction

Contents

Introduction

Chaque recherche utilisateur retourne une liste de résultats correspondants aux paramètres d'entrée : jour de départ, jour de retour, ville de départ et ville d'arrivée. Chaque résultat possède une liste d'attributs propre à son vol et au site proposant le vol : horaires de départ et de retour, compagnie aérienne, aéroport de départ et d'arrivée, site marchand etc. L'information que nous souhaitons fournir aux usagers de liligo.com est une prédiction d'évolution de prix par ligne de résultats, c'est-à-dire pour un trajet vendu par un site marchand. Les seules informations que nous possédons sur les vols sont celles que nous venons de décrire, c'est pourquoi nous voulons pouvoir associer à ces vols une prédiction uniquement grâce à leurs attributs.

La base d'apprentissage des niveaux de gris est à présent segmentée en groupes de vols similaires représentés par un comportement type (moyenne des niveaux de gris du groupe) ou par un ensemble de paramètres (α_k, I_k) suivant l'algorithme de segmentation utilisé. Nous voulons maintenant attribuer aux résultats de la recherche le comportement le plus probable parmi ces groupes en se basant sur la répartition des attributs des vols d'apprentissage.

Nous appliquons donc un algorithme d'apprentissage supervisé sur les données segmentées : chaque vol possède une série d'attributs V_i et une étiquette E_i correspondant au groupe dans lequel il a été placé. Nous allons alors créer des règles de classification uniquement basées sur l'ensemble des attributs $V_i(1), \ldots, V_i(p)$ afin d'affecter aux vols de test leur groupe $k \in \{1, \ldots, C\}$ le plus vraisemblable. Pour cela nous avons utilisé plusieurs algorithmes usuels qui sont les arbres de classification CART [5] et C4.5 [41], Adaboost [19] qui utilise CART comme classifieur faible dans son étape de boosting et les Forêts d'arbres décisionnels [9] (Random Forest) qui utilisent aussi CART comme classifieur faible dans son étape bagging. Avec CART et C4.5, nous pourrons observer la segmentation de l'espace des attributs et les règles de prédiction de comportement. Adaboost va exécuter itérativement CART sur la base d'apprentissage pondérée différemment à chaque itération. La pondération donne plus de poids aux vols mal classés afin d'améliorer la prédiction des vols dit "difficiles" à prédire correctement. Les forêts aléatoires, dont le principe est de multiplier les arbres de décision CART et d'agréger leurs résultats par vote, améliorent les résultats tout en fournissant un classement des attributs les plus influents dans la classification. Nous utiliserons ensuite ce classement pour ne conserver que les attributs les plus pertinents, technique nommée *feature selection*. Nous décrivons aussi une extension de l'algorithme Esperance-Maximisation qui prend en compte les attributs et combine l'étape de segmentation à l'étape de classification.

Nous attribuons donc à chaque résultat de la recherche utilisateur son identifiant de cluster le plus probable auquel est associé un centroïde représentant son comportement global. Dans le Chapitre 2, nous avons montré qu'il était possible de simuler une courbe de prix à partir d'une image pixélisé pour reconstituer un série de prix similaire à l'originale. Il est donc possible de simuler un ensemble de séries appartenant potentiellement à un groupe en utilisant le centroïde du cluster en question. En moyennant un nombre prédéfini de courbes simulées, nous sommes capable d'estimer l'évolution d'un vol assigné à ce groupe. Par exemple, si nous sommes à 20 jours de la date de départ et que le premier résultat de la recherche est associé au groupe numéro 2, nous pouvons simuler des courbes issues du deuxième cluster, se placer à -20 jours pour chacune d'elles et observer l'évolution de celles-ci. Il est possible d'extraire de ces simulations

autant d'informations qu'une série de rendements peut fournir : la hausse ou la baisse du prix à n jours, le rendement de cette variation mais aussi l'existence d'un baisse pendant plus de 24h dans un intervalle de m jours ou la probabilité d'une forte hausse ou d'une faible baisse. Cette flexibilité est très importante car elle se répercute directement sur la flexibilité du service final.

Dans un second temps, nous explicitons ce que nous avons nommé la prédiction "directe" : une prédiction basée sur l'apprentissage direct de l'évolution de prix à l'instant t et non plus d'un comportement global. Cette prédiction peut-être binaire (hausse ou baisse du prix dans 7 jours), par cadran (forte hausse, faible baisse etc.) ou continue (pourcentage d'évolution). L'étiquette E_i que nous tentons d'apprendre n'est donc plus le groupe le plus probable attribué aux vols mais la prédiction à l'instant t, impliquant la création d'un modèle par nombre de jours avant la date de départ. Nous nous passons ainsi de la création de comportements types et de la simulation de courbes. Nous étudierons les avantages et les inconvénients de cette approche.

4.1 WorkFlow

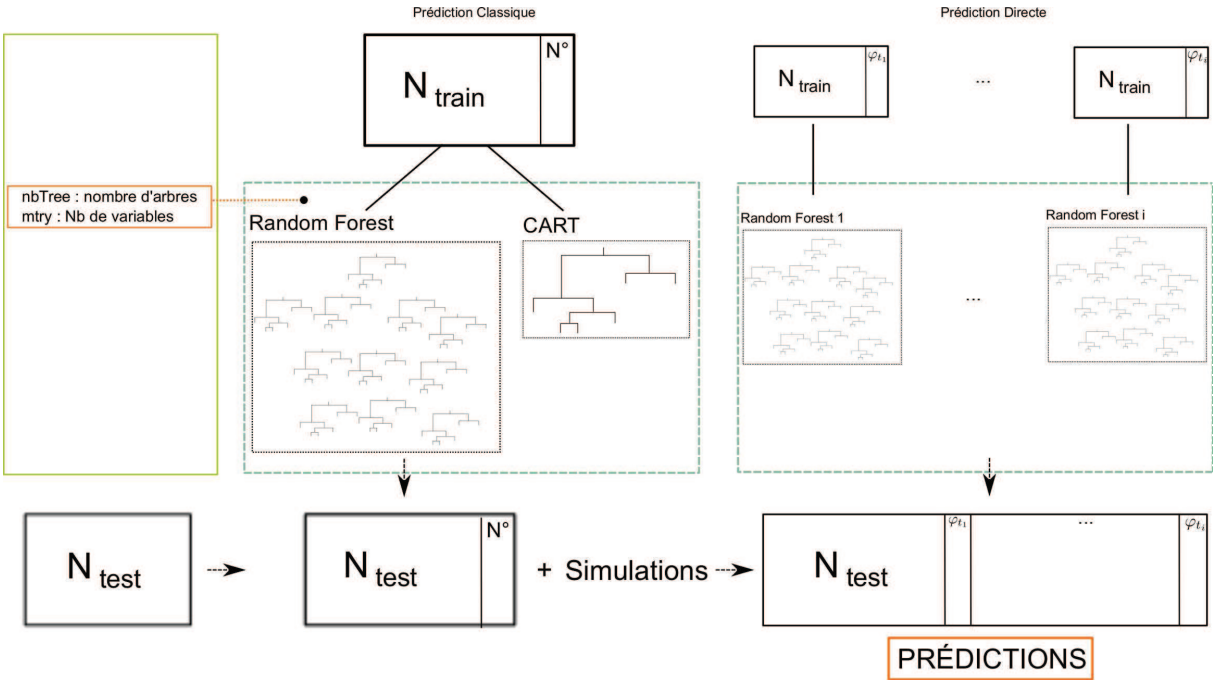

FIGURE 4.1 – Workflow - Étape de classification de la base d'apprentissage et de prédiction de la base de test

4.2 Notations

i	:	Numéro du vol de la base d'apprentissage $i \in 1, ..., n$
n_{test}	:	Nombre de vols de test
p	:	Nombre d'attributs
V_i	:	Vecteur de p attributs du vol i parmi l'ensemble des attributs $\mathcal{V}$
N_{train}	:	Base d'apprentissage des vecteurs d'attributs de taille $n \times p$
N_{test}	:	Base de test des vecteurs d'attributs de taille $n_{test} \times p$

Posons par commodité la date de départ à l'origine, $T_0^{(i)} = 0$. Nous définissons la variable $\varphi_t^{(i)} \in \{0, 1\}$ correspondant respectivement aux conseils "achat" et "attendre" à l'instant t pour le vol i. Nous définissons pour le reste du Chapitre, $\varphi_{t_1}^{(i)} = 1$ si et seulement si le prix $p_i(-t_2)$ à 7 jours est inférieur ou égal à $p_i(-t_1)$. Il est possible de calculer $\varphi_{t_1}^{(i)}$ de plusieurs autres manières mais nous avons décidé de nous concentrer sur une prédiction simple pour ensuite la raffiner.

Nous rappelons q'une fois la classe j obtenue en appliquant le classifieur aux attributs du trajet i considéré, le modèle de la classe j est utilisé pour calculer la probabilité $\mathbb{P}(\varphi_{t_1}^{(i)} = 1)$ grâce aux simulations précédemment décrites. Cette valeur est utilisée pour proposer un conseil d'achat ou d'attente pour un niveau de confiance donné. Dans le cas de la prédiction directe, nous verrons qu'elle servira d'étiquette lors de la phase d'apprentissage.

4.3 Apprentissage

Avant de pouvoir prédire l'évolution des prix d'un vol à un instant t, c'est-à-dire $\mathbb{P}(\varphi_t^{(i)} = 1)$, il nous faut prédire le groupe auquel il appartient. Les seuls informations dont nous disposons sont la liste de ses attributs $V_i = V_i(1), \ldots, V_i(p)$ dans l'espace des attributs $\mathcal{V}$, et éventuellement les premières évolutions de prix. Nous tentons dans un premier temps d'utiliser uniquement les attributs du vol pour prédire son comportement le plus vraisemblable. Pour cela nous utilisons des algorithmes de la famille des arbres de classification, qui vont segmenter l'espace des attributs de la base d'apprentissage selon l'étiquette E_i correspondant au groupe associé à chaque vol, afin de créer des régles de classification :

1. **CART** (Breiman, 1984 [5]) est un algorithme de classification basé sur les arbres de décision. Il va découper l'espace des attributs récursivement en maximisant à chaque étape le gain d'information. A chaque nœud, il choisit l'attribut le plus discriminant et sépare l'ensemble des vols en deux groupes qui, à leur tour, seront divisés en deux sous-ensembles. L'arbre final est un arbre binaire très facilement interprétable puisqu'il suffit de remonter l'arbre depuis chaque feuille pour obtenir les règles de classification. Nous décrivons par la même occasion un autre algorithme basé sur les arbres de décision et très largement utilisé, nommé C4.5 (introduit par Quilan [41]) qui est une amélioration de l'algorithme ID3 [40], et dont les récentes améliorations (il existe maintenant l'algorithme C5.0) le font se rapprocher de CART [23].

2. **Adaboost** (Freund, 1995 [19]) est une méthode de boosting dont le principe est de sélectionner des classifieurs faibles afin de minimiser l'erreur en classification selon des exemples d'apprentissage pondérés itérativement en fonction de leur difficulté à être correctement classés. Nous utiliseront les arbres CART comme classifieurs faibles.

3. **Les forêts aléatoires** (Breiman, 2001 [9]) sont une variante du bagging (également introduit par Breiman [6]) dont le principe est de construire un grand nombre d'arbres décorrélés pour ensuite les moyenner. Dans la plupart des cas d'utilisation, les forêts aléatoires ont des performances similaires au boosting (Adaboost), tout en étant plus facile à entraîner et à paramétrer. C'est pourquoi les forêts aléatoires sont populaires, et sont implémentées dans de nombreux langages. Genuer([21], 2010), fait une description complète et détaillée de la théorie et de la pratique des forêts aléatoires, détaillant notamment la possibilité de sélectionner les variables les plus pertinentes afin d'améliorer la classification.

4.3.1 Arbres de décision : CART & C4.5

L'algorithme de CART peut être divisé en 4 étapes. La première étape consiste à construire un arbre de manière récursive en séparant l'ensemble des vols d'un nœud en deux sous-ensembles selon un des attributs : il est important de noter qu'à chaque nœud, terminal ou non, est associée une classe prédite en fonction de la distribution des vols du groupe correspondant et de la matrice de coût. Cette construction doit ensuite être stoppée suivant différents critères d'arrêt. A l'issue de la construction, l'arbre est de grande taille et surapprend les données d'apprentissage. La troisième étape consiste donc à élaguer l'arbre pour le rendre plus simple et donc plus généralisable. Enfin il s'agira de retenir l'arbre élagué qui minimise le taux d'"erreur estimé" mesuré sur un échantillon test.

1. **La construction de l'arbre** commence à la racine où tous les vols sont regroupés. L'algorithme va alors trouver le meilleur attribut avec lequel séparer l'ensemble des vols en deux : il faut alors tester toutes les valeurs possibles de chaque attribut afin d'optimiser un critère prédéfini. Pour éviter un temps de calcul trop important il est possible de préciser le nombre de niveaux à tester pour les attributs qualitatifs.

   A chaque itération, nous cherchons à maximiser la pureté moyenne des deux sous-nœuds. Pour un nœud m représentant une sous-partition N_m de l'ensembles des attributs de la base d'apprentissage N_{train} et possédant n_m vols posons :

   $$\hat{p}_{mk} = \frac{1}{n_m} \sum_{i \in N_m} \mathbb{1}_{y_i = k},$$

   la proportion des vols de la classe k au nœud m. A chaque nœud m, nous classons l'ensemble des vols dans la classe $k(m) = \arg\max_k \hat{p}_{mk}$. Nous définissons alors différentes métriques de l'impureté d'un nœud :

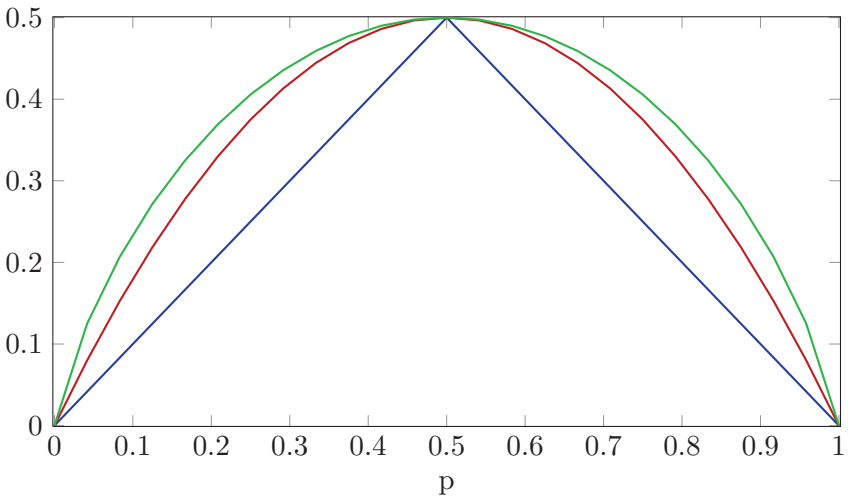

FIGURE 4.2 – Mesures d'impureté d'un nœud pour une classification binaire en fonction de la proportion p de la classe 2.

Erreur en classification :	$\frac{1}{n_m} \sum_{i \in N_m} \mathbb{1}(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$.
Indice de Gini :	$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{C} \hat{p}_{mk}(1 - \hat{p}_{mk})$.
Entropie croisée ou déviance :	$-\sum_{k=1}^{C} \hat{p}_{mk} \log(\hat{p}_{mk})$.

Dans le cas d'une classification à deux classes, si p est la proportion dans la deuxième classe, ces trois mesures s'écrivent respectivement $1-(max(p, 1-p)$, $2p(1-p)$ et $-p \log p - (1-p) \log(1-p)$. La Figure 4.2 nous montre graphiquement les différences entre ces trois mesures d'impureté.

Nous n'utiliserons que l'indice de "Gini", introduit par Breiman dans son étude de CART, et qui servira aussi de mesure d'impureté dans l'algorithme des forêt aléatoires. L'impureté d'une population pour une classification multiclasse est mesurée par la formule suivante :

$$I(T) = 1 - \sum_{j=1}^{c} (\frac{n_{j.}}{n_{..}})^2$$

où n_{ji} représente l'effectif de la classe j au nœud i ; le point symbolise alors la sommation selon l'indice correspondant. Cette mesure peut être vue comme l'erreur de classification attendue si la classe prédite était choisie aléatoirement en suivant la distribution des probabilités de classes, approximée par les effectifs du nœud T courant.

Breiman suggère que, dès lors que les mesures utilisées possèdent de bonnes propriétés de spécialisation, c'est-à-dire tendent à mettre en avant les partitions avec des feuilles pures,

elles ne jouent pas un rôle majeur dans la qualité de la prédiction.

Un des inconvénients est que l'algorithme de partitionnement favorise les variables qualitatives avec le plus de niveaux : plus ce nombre augmente, plus le nombre de partitions augmente rapidement ainsi que le choix de segmentation. Il est alors plus probable de trouver une séparation du nœud en utilisant les variables avec le plus de niveaux.

2. **L'arrêt de la construction** intervient lorsque :

   (a) Il n'y a plus qu'une seule observation dans les feuilles
   (b) La distribution des attributs dans toutes les feuilles est identique rendant la séparation impossible
   (c) La profondeur limite définie a été atteinte

   L'arbre final est souvent trop fidèle aux données et n'est donc pas capable de généraliser correctement sur la base de test. C'est pourquoi il est nécessaire de passer par l'étape d'élagage.

3. **L'élagage** consiste à supprimer a posteriori des branches de l'arbre jugées inutiles. Si aucun élagage n'est réalisé l'arbre sur-apprend inévitablement les données d'apprentissage et devient par conséquent très mauvais en généralisation. Le principe de la démarche consiste à construire une suite emboîtée de sous-arbres de l'arbre maximum par élagage successif puis à choisir, parmi cette suite, l'arbre optimal au sens d'un critère. La solution ainsi obtenue par un algorithme pas à pas n'est pas nécessairement globalement optimale mais l'efficacité et la fiabilité sont préférées à l'optimalité.

   Pour un arbre A donné, nous notons K le nombre de feuilles ou nœuds terminaux de A ; la valeur de K exprime la complexité de A. La mesure de qualité de discrimination d'un arbre A s'exprime par un critère

   $$D(A) = \sum_{k=1}^{K} D_k(A)$$

   où $D_k(A)$ est le nombre de vols mal classés de la $k$ème feuille de l'arbre A.

   La construction de la séquence d'arbres emboîtés repose sur une pénalisation de la complexité de l'arbre :

   $$C(A) = D(A) + \gamma K$$

   Pour $\gamma = 0$, $A_{max} = A_K$ minimise $C(A)$. En faisant croître γ, l'une des divisions de A_K, celle pour laquelle l'amélioration de D est la plus faible (inférieure à γ), apparaît comme superflue et les deux feuilles obtenues sont regroupées (élaguées) dans le nœud père qui devient terminal ; A_K devient A_{K-1}. Le procédé est itéré pour la construction de la séquence emboîtée :

   $$A_{max} = A_K \supset A_{K-1} \supset ...A_1$$

où A_1, le nœud racine, regroupe l'ensemble de l'échantillon. L'arbre optimal correspond donc au k qui minimise $D(A_k)$.

Sur la Figure 4.3, nous observons un exemple d'arbre de décision créé à partir de notre base de données. Nous remarquons que des variables peuvent être utilisées plusieurs fois sur différents nœuds, ici *initialPrice*, et que tous les groupes peuvent ne pas apparaître dans les nœuds terminaux (le groupe 4 dans notre cas). Comme expliqué précédemment, il est facile d'interpréter ces arbres et de créer des règles de classification en remontant les feuilles jusqu'à la racine. Les deux règles de la branche de gauche peuvent alors s'écrire :

(a) Si le prix initial est inférieur à 148€ et que la distance entre les aéroports est **inférieure** à 913km alors le vol fait partie du group 1
(b) Si le prix initial est inférieur à 148€ et que la distance entre les aéroports est **supérieure** à 913km alors le vol fait partie du group 2

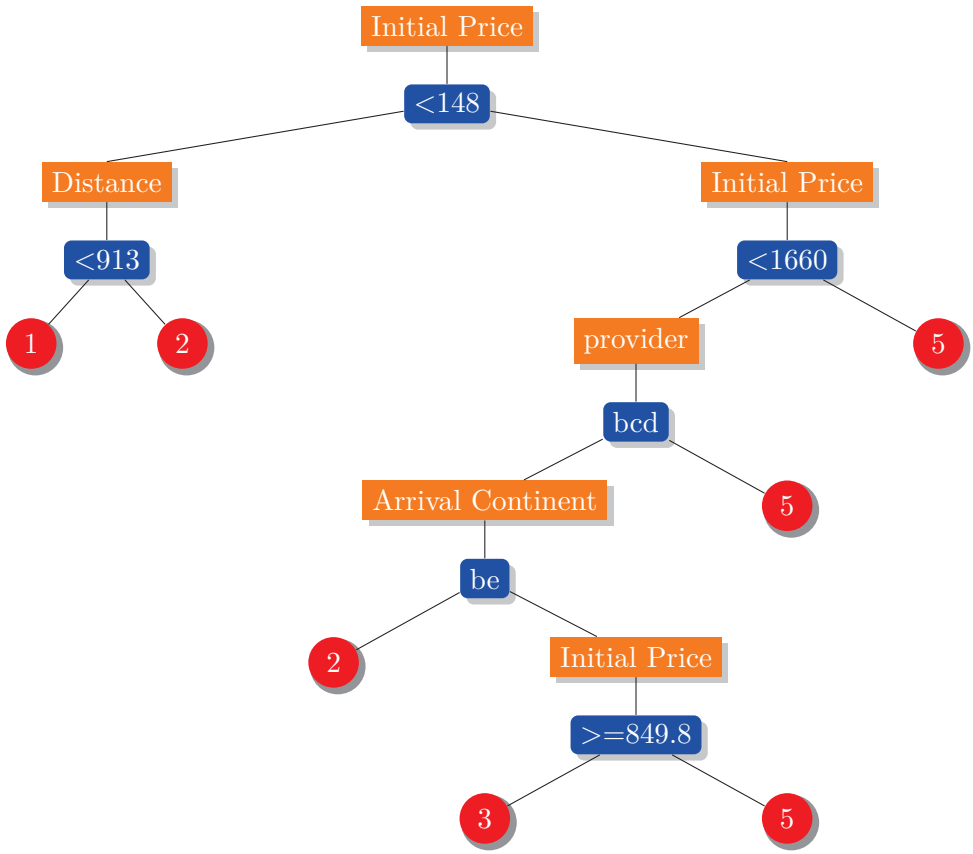

FIGURE 4.3 – Test de représentation d'un arbre binaire : Modèle 348 (CART)

Ce ne sont pas réellement les critères d'évaluation des règles de partitionnement qui conditionnent la structure des arbres, mais la topologie de la base d'apprentissage. En effet CART

est un algorithme qui sur-apprend les données d'apprentissage, et dépend par conséquent inévitablement de la représentativité de ces données vis à vis du domaine de définition du problème. L'avantage des arbres de décision est qu'ils sont simples à comprendre et à interpréter, qu'ils supportent à la fois les données quantitatives et qualitatives et que le temps de calcul sur un grand volume de données reste faible sans dégrader les performances.
Cependant les arbres peuvent devenir trop complexes, ne généralisant pas correctement les données et l'utilisation de variables qualitatives favorisant celles avec le plus de niveaux.

L'algorihtme C4.5 et par extension C5.0, qui est une amélioration des performance de l'ancien algorithme, reprend les principes de construction d'un arbre de décision de CART. Toutefois certains points diffèrent :

1. Alors que traditionnellement CART utilise l'indice de GINI comme critère de sélection de l'attribut suivant lequel séparer le nœud courant, C4.5 utilise l'entropie croisée.

2. L'arbre généré n'est pas binaire : C4.5 créé plusieurs branches à chaque séparation suivant le nombre de niveaux pour une variable qualitative. Pour une variable quantitative, une branche est créée pour chaque intervalle de la version discrétisée de la variable.

Les algorithmes de CART et C4.5, comme tout autre algorithme unique, possède des limitation décrites dans [16] et résumées dans [4] qui expliquent les faibles performances de celui-ci :

La limite statistique concerne les situations pour lesquelles l'espace de recherche du classifieur est particulièrement grand proportionnellement au nombre de données disponibles en apprentissage. Il est en revanche possible de réduire l'erreur d'un classifieur peu performant en généralisation, en combinant les prédictions de plusieurs classifieurs du même type, entraînés sur les mêmes données, et qui se sont montrés performants en apprentissage.

La limite de représentation de la "véritable" fonction de prédiction $y = f(x)$ décrit l'impossibilité de représenter celle-ci par un classifieur unique. Nous pouvons étendre l'espace de recherche en ajoutant des classifieurs combinant plusieurs classifieurs individuels. Il est par exemple possible de résoudre des problèmes de classification multi-classes à l'aide d'algorithmes d'apprentissage binaires.

La limite computationnelle représente le coût pour un algorithme d'apprentissage d'effectuer une recherche plus "large" afin d'éviter les optima locaux. Il est alors également possible de réduire le risque de choisir un mauvais classifieur en combinant plusieurs classifieurs sous-optimaux (localement optimaux).

En résumé, la combinaison de plusieurs classifieurs permettrait (i) de fiabiliser les décisions grâce à l'avis de plusieurs experts face à un seul, (ii) d'élargir l'ensemble des solutions possibles, (iii) d'éviter les optima locaux, et (iv) de traiter des problèmes trop complexes pour un simple classifieur.

Dans la domaine des système multi-classifieurs (MCS), les Ensemble de Classifieurs est une des approches les plus populaires et les plus efficaces, et consiste à combiner un ensemble de classifieurs de même type. L'objectif est de créer de la diversité au sein d 'un ensemble de classifieurs performants et d'établir le meilleur consensus possible entre ces classifieurs. La diversité est la propriété importante pour un MCS.

4.3.2 Adaboost

Adaboost [19] est une technique de boosting basée sur la sélection itérative de classifieurs faibles (en l'occurence CART). L'idée est d'améliorer les compétences d'un faible classifieur c'est-à-dire celle d'un modèle de discrimination dont la probabilité de succès sur la prévision d'une variable qualitative est légèrement supérieure à celle d'un choix aléatoire.

Introduit par Freund et Schapire, le boosting est une des méthodes d'ensemble les plus performantes à ce jour. Étant donné un échantillon d'apprentissage $\mathcal{V}_n$ et une méthode de prédiction faible (ici CART) qui construit un prédicteur $\hat{h}(\mathcal{V}_n)$. Un premier échantillon $\mathcal{V}_n^1$ est tiré aléatoirement (bootstrap), où chaque observation a une probabilité $1/n$ d'être tirée. Nous appliquons alors le classifieur de base à cet échantillon pour obtenir un premier prédicteur $h(\mathcal{V}_n^1)$. Ensuite, l'erreur de $\hat{h}(\mathcal{V}_n^1)$ sur l'échantillon $\mathcal{V}_n$ est calculé. Un deuxième échantillon bootstrap $\mathcal{V}_n^2$ est alors tiré en se basant sur les prédictions de $\hat{h}(\mathcal{V}_n^1)$. Le principe est d'augmenter la probabilité de tirer une observation mal prédite, et de diminuer celle de tirer une observation bien prédite. Nous réitérons alors le processus pour obtenir un ensemble de classifieur que nous agrégeons ensuite en faisant une moyenne pondérée. Le Boosting est donc une méthode séquentielle, chaque échantillon étant tiré en fonction des performances de la règle de base sur l'échantillon précédent. L'idée du Boosting est de se concentrer de plus en plus sur les observations mal prédites par la règle de base, pour essayer d'apprendre au mieux cette partie difficile de l'échantillon, en vue d'améliorer les performances globales.

Avantages : ce type d'algorithme permet de réduire sensiblement la variance mais aussi le biais de prévision comparativement à d'autres approches. Cet algorithme est considéré comme la meilleure méthode "off-the-shelf" [8], c'est-à-dire ne nécessitant pas un long pré-traitement des données ni un réglage fin de paramètres lors de la procédure d'apprentissage. Le boosting adopte le même principe général que le bagging : construction d'une famille de modèles qui sont ensuite agrégés par une moyenne pondéré des estimations ou un vote. Il diffère nettement sur la façon de construire la famille qui est dans ce cas récurrente : chaque modèle est une version adaptative du précédent en donnant plus de poids, lors de l'estimation suivante, aux observations mal ajustées ou mal prédites. Intuitivement, cet algorithme concentre donc ses efforts sur les observations les plus difficiles à ajuster tandis que l'agrégation de l'ensemble des modèles permet d'échapper au sur-ajustement. Les algorithmes de boosting existants diffèrent par différentes caractéristiques :

1. la façon de pondérer c'est-à-dire de renforcer l'importance des observations mal estimées lors de l'itération précédente
2. leur objectif selon le type de la variable à prédire Y : binaire, qualitative à k classes, réelles
3. la fonction perte, qui peut être choisie plus ou moins robuste aux valeurs atypiques, pour mesurer l'erreur d'ajustement

4. la façon d'agréger, ou plutôt pondérer, les modèles de base successifs.

La littérature sur le sujet présente donc de très nombreuses versions de cet algorithme et il est encore difficile de dire lesquelles sont les plus efficaces.

Il est important de noter tout de même que dans le cas de forêts boostées, les arbres de décision doivent impérativement être élagués. En effet le boosting est réalisé sur la base des erreurs de prédiction en apprentissage des classifieurs faibles. Or, sans élagage, les arbres de décision sur-apprennent les données d'apprentissage jusqu'à avoir une erreur en apprentissage nulle dans la plupart des cas.

Dans notre cas, chaque vol d'apprentissage est pondéré à chaque itération afin de minimiser l'erreur de classification. Il n'y a donc aucune part d'aléatoire dans cette approche contrairement aux Random Forest.

Algorithm 3 Adaboost

Entrée : $\mathcal{S} = (V_i, E_i)_{i=1,...,n}$, avec E_i la classe obtenue en appliquant le classifieur
Pour $i = 1...n$ **faire**
  $p_1(V_i) \leftarrow \frac{1}{n}$
Fin Pour
Pour $j = 1...nbClassifieur$ **faire**
  Tirer un échantillon d'apprentissage S_j dans $\mathcal{S}$ selon les probabilités p_j
  Entraîner un classifieur C_j sur ce sous-échantillon.
  Soit q_j l'erreur apparente de C_j sur $\mathcal{S}$
  Calculer $\alpha_j = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
  Pour $i = 1...n$ **faire**
    si $h_j(V_i) = y_i$ (bien classé par h_t) **alors**
      $p_{j+1}(V_i) \leftarrow \frac{p_j(V_i)}{Z_t} e^{-\alpha_t}$
    sinon
      $p_{j+1}(V_i) \leftarrow \frac{p_j(V_i)}{Z_t} e^{+\alpha_t}$
    Fin si
    Avec Z_t normalisé tel que $\sum_{i=1}^{m} p_j(V_i) = 1$
  Fin Pour
Fin Pour
return L'ensemble des arbres $\{T_b\}_1^B$

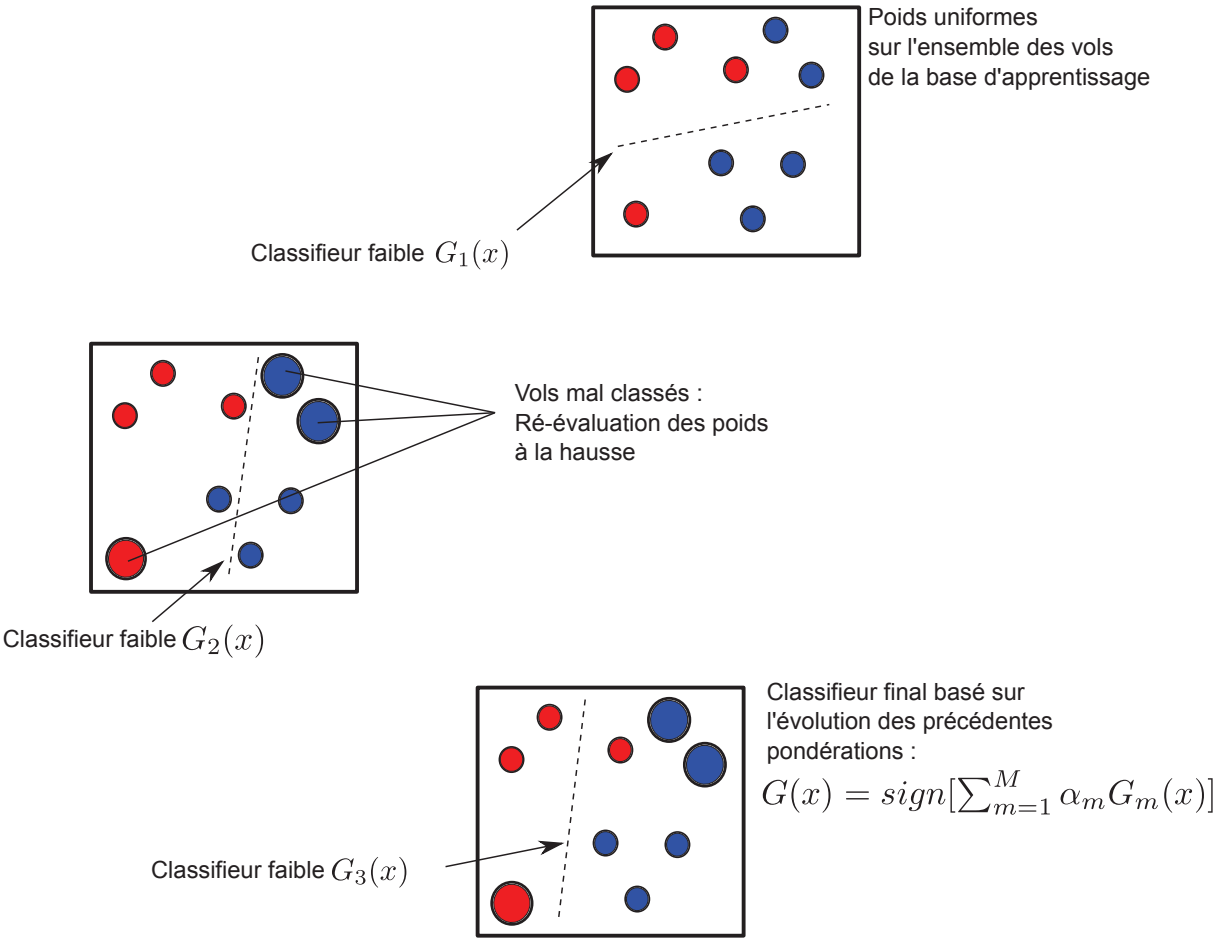

FIGURE 4.4 – Adaboost

Les méthodes de Boosting sont en fait très fréquemment citées à titre de comparaison dans les travaux qui étudient les forêts aléatoires, puisqu'en plus d'être particulièrement efficaces pour l'induction de forêts de décision, elles s'opposent aux forêts aléatoires dans l'idée principale : le Boosting s'appuie sur un principe déterministe pour la création de la diversité dans les ensembles alors que les forêts aléatoires par définition le font via les principes de randomisation.

4.3.3 Forêts aléatoires (Random Forest)

Les méthodes de forêts aléatoires sont basées sur la combinaison de classifieurs élémentaires de types arbres de décision. Individuellement, ces classifieurs ne sont pas réputés pour être particulièrement efficaces, mais ils possèdent une propriété intéressante à exploiter : leur instabilité. Un petit changement dans la base d'apprentissage provoque un changement important dans la structure de l'arbre et donc dans ses performances en généralisation. La spécificité des arbres utilisés dans les forêts aléatoires est que leur induction est perturbée d'un facteur aléatoire, et

ce dans le but de générer de la diversité dans l'ensemble. C'est sur la base de ces deux éléments — utiliser des arbres de décision comme classifieurs élémentaires et faire intervenir l'aléatoire dans leur induction — qu'a été introduit le formalisme des forêts aléatoires.

Dans [7] (1996), Breiman propose une méthode pour construire de multiples classifieurs indépendants sur des sous-ensembles aléatoires de la base d'apprentissage. Chaque classifieur est un arbre de décision CART précédemment décrit mais qui n'a pas été élagué. Les prédictions de chaque arbre sont regroupées et la prédiction finale est choisie par un système de vote à la pluralité : chaque arbre participe au vote de la classe la plus populaire pour une donnée d'entrée x. Cette méthode est basé sur un principe déjà connue nommé bagging.

Le Bagging (**B**ootstrap **Agg**regat**ing**) applique le principe de Bootstrap à l'agrégation de classifieurs. Le Bootstrap est un principe de ré-échantillonnage statistique [17] traditionnellement utilisé pour l'estimation de grandeurs ou de propriétés statistiques. L'idée du bootstrap est d'utiliser non plus une unique distribution empirique, mais plusieurs ensembles de données ré-échantillonnées à partir de l'ensemble des données observées à l'aide d'un tirage aléatoire avec remise. Le bagging est alors l'agrégation de classifieurs appliqués chacun sur des sous-ensembles de données ré-échantillonnées.

Breiman [9] (2001) décide avec les random forest d'ajouter une couche d'aléatoire dans la construction de ces classifieurs en plus de la sélection aléatoire du sous-ensemble sur lequel apprendre l'arbre. Dans le cas du bagging, l'algorithme de construction des arbres est le même que CART et divisent les nœuds en choisissant la meilleure variable parmi toutes les variables possibles. Dans le cas des forets aléatoires, la sélection de la meilleure variable se fait sur un sous-ensemble aléatoire de l'ensemble des variables. Puisque les classifieurs sont de même type, il est évidemment important de générer des "différences" entre chacun d'eux, le critère essentiel dans les système multi-classifieurs étant, encore une fois, de créer de la diversité.

Breiman démontre la propriété importante de convergence des forêts aléatoires. Plus précisément il démontre que le taux d'erreur en généralisation, d'une forêt aléatoire converge nécessairement vers une valeur limite, quand le nombre d'arbres qui la composent augmente. Concrètement, cela signifie une chose importante : avec un nombre suffisamment élevé d'arbres, une forêt aléatoire évite le sur-apprentissage.

Cette approche donne de meilleurs résultats que la majorité des autres classifieurs, est robuste face au sur-apprentissage et ne nécessite que deux paramètres : la taille du sous-ensemble de variables et le nombre d'arbres à construire.

L'algorithme 4 [21] décrit les différentes étape de la construction de la forêt.

La principale force du bagging est donc de réduire l'instabilité pour augmenter les performances en généralisation. Mais il y a un autre point qui fait la force du bagging, ce sont les mesures out-of-bag :

Les forets aléatoires ne nécessitent pas de validation croisée, ou même d'une base de test indépendante pour estimer l'erreur en test. Cette erreur est calculée tout au long du processus comme suit : Pour chaque arbre construit sur le sous-ensemble Z^*, une prédiction est faite sur tous les vols de l'ensemble $Z^{*\complement}$. En agrégeant les prédictions faites par tous les arbres (en moyenne un vol sera prédit une fois sur trois), nous pouvons calculer l'erreur en prédiction que nous nommons alors l'estimation de l'erreur OOB.

Algorithm 4 Forêt d'arbres décisionnels pour la classification

Entrée : $\mathcal{S} = (V_i, E_i)_{i=1,...,n}$, avec E_i la classe obtenue en appliquant le classifieur
Entrée : $ntree$ le nombre d'arbres dans la forêt
Entrée : $mtry$ le nombre de caractéristiques à sélectionner aléatoirement à chaque nœud
Pour $j = 1...ntree$ **faire**
  $\mathcal{S}_j \leftarrow$ ensemble boostrap, dont les données sont tirées aléatoirement (avec remise) de $\mathcal{S}$
  $arbre \leftarrow$ un arbre vide composé de sa racine uniquement
  $arbre.racine \leftarrow RndTree(arbre.racine, \mathcal{S}_j, mtry)$
  $foret \leftarrow foret \cup arbre$
Fin Pour
return $forest$

Algorithm 5 RndTree

Entrée : n le nœud courant
Entrée : $\mathcal{S}$ l'ensemble des données associées au nœud n
Entrée : $mtry$ le nombre de caractéristiques à sélectionner aléatoirement à chaque nœud
si n n'est pas une feuille **alors**
  $C \leftarrow mtry$ caractéristiques choisies aléatoirement
  pour tout $A \in C$ **faire**
    Procédure CART pour la création et l'évaluation (critère de Gini) du partitionnement produit par A, en fonction de $\mathcal{S}$
  Fin Pour
  $partition \leftarrow$ partition qui optimise le critère Gini
  $n.ajouterFils(partition)$
  pour tout $fils \in n.noeudFils$ **faire**
    $RndTree(fils, fils.donnes, mtry)$
  Fin Pour
Fin si
return

Importance des variables Il est très utile, en pratique, d'avoir des informations sur les variables des données que l'on étudie. Quelles sont les variables vraiment nécessaires pour expliquer notre classification ? De quelles variables peut-on se passer ? Ces informations peuvent être d'une grande aide pour l'interprétation des données. Elles peuvent également servir à construire de meilleurs prédicteurs : un prédicteur construit en utilisant uniquement les variables utiles pourra être plus performant et moins complexe qu'un prédicteur construit avec des variables susceptibles de générer du bruit.

Deux méthodes de calcul de l'importance des variables existent dans l'algorithme des forêts aléatoire qui nous permettent de classer les variables par ordre d'importance dans l'apprentissage :

1. La "moyenne des baisses en précision" d'une variable se calcule grâce à l'erreur OOB. Pour mesurer l'importance du j-ième attribut après l'apprentissage, les valeurs du j-ième attribut sont permutées dans la base de données d'apprentissage et l'erreur OOB est à nouveau calculée sur cet ensemble de données "perturbée". L'importance du j-ième attribut est alors calculée en faisant la moyenne de la différence entre l'erreur OOB avant et après la perturbation sur tous les arbres. Le score est normalisé par l'écart type de ces différences. Plus la précision de la classification de la forêt décroît par l'ajout d'une variable supplémentaire, plus celle-ci est importante.

2. La "moyenne des baisses du coefficient de Gini" mesure l'influence d'une variable dans l'homogénéité des nœuds et des feuilles des arbres de la forêt. Dès qu'une variable est sélectionnée pour diviser un nœud, le coefficient de Gini est calculé pour les deux enfants et comparé au nœud initial. La différence des coefficient est sommée pour chaque variable et normalisée à l'issue du processus : plus la baisse est importante, plus l'influence de la variable sur la pureté des nœuds l'est aussi.

Ces méthodes présentent certains inconvénients : elles favorisent les attributs quantitatifs et les attributs qualitatifs possédant le plus de niveaux [47] (comme pour CART). De plus, elles se sont révélées être fortement corrélées. D'autres articles ajoutent que si certains attributs sont corrélées ou ont la même pertinence dans la prédiction, alors les petits groupes sont favorisés par rapport aux grands groupes [49]. Dans le papier [47], Strobl et al. proposent d'utiliser une autre implémentation des forêts aléatoires où les classifieurs faibles sont des arbres de classification non-biaisés (une estimation non-biaisée du taux d'erreur en classement lors de la phase d'élagage) basés sur une approche conditionnelle (package R [25]). De plus, Strobl et al. préconisent l'utilisation d'un boostrap sans remise afin d'obtenir une mesure d'importance des variables sans biais.

Cardinalité des attributs Un autre inconvénient des forêts aléatoires est sa limitation dans le nombre maximum de niveau pour les attributs qualitatifs. L'étape de permutation interdit un trop grand nombre de niveaux en raison de sa nature combinatoire. L'implémentation R, par exemple, nous limite à 32 niveaux par attributs. De plus, la multiplication des niveaux perturbe la classification aussi bien qualitativement que computationnellement [5]. Une solution proposée dans [35] lors de la compétition KDD 2009, est de regrouper les niveaux peu représentés entre

eux et garder les plus fréquents. Nous avons donc décidé d'utiliser pour les variables qualitatives les niveaux apparaissant plus de 2% du temps dans une limite de 30 niveaux. Un nouveau niveau "-1" est alors créé afin de regrouper l'ensemble des autres valeurs.

4.4 Prédiction d'un comportement

Dans cette section nous décrivons le passage de la prédiction d'un groupe à partir d'un vecteur d'attributs à la prédiction d'une évolution de prix. Grâce aux algorithmes précédemment décrits, nous sommes capable d'associer à un vol un numéro de groupe E_i à partir du vecteur d'attributs V_i en suivant des régles de classification. Une fois ce groupe assigné au vol, nous utilisons les simulations du centroïde associé (décrites dans le Chapitre 2) pour prédire le comportement futur de la série de prix à l'instant t de la demande de prédiction.

Apprentissage d'un comportement Chaque vol de test possède un vecteur d'attributs V_i de même nature que les vols de la base d'apprentissage. Nous avons vu précédemment la limitation de certains algorithmes à traiter des vecteurs d'attributs avec des niveaux absents de la base d'apprentissage. Nous avons donc créé pour chaque attribut qualitatif un niveau supplémentaire nommé "-1". Lors de la construction des V_i de N_{test}, nous prenons soin de vérifier l'existence des niveaux des attributs qualitatifs dans la base d'apprentissage et dans le cas contraire de lui attribuer la valeur "-1". Nous appliquons par la même occasion la selection des valeurs les plus présentes et le regroupement des plus rares dans le même niveau "-1".

Les modèles de classification précédemment créés génèrent pour chaque vol de test, représenté par son vecteur d'attribut V_i, un vecteur de sortie correspondant à la probabilité d'appartenance à chacun des cluster : $\mathbb{P}(V_i \in I_k), \forall k = 1, ..., C$. L'étiquette E_i correspondant au numéro du cluster le plus probable est alors associée aux vols de test. Le cas d'une équiprobabilité d'appartenance à plusieurs groupes pourra être traité de différentes manières que nous décrirons plus tard.

Prédiction d'une évolution Nous avons maintenant un modèle capable d'attribuer un numéro de cluster à un nouveau vol uniquement grâce à ses attributs. Ce numéro de cluster nous permet d'associer au vol de test un centroïde et donc un comportement global au vol représenté par le centroïde du groupe. Nous rappelons qu'à partir de n'importe quel niveau de gris nous sommes capable de reconstituer des vols issues de celui ci. Dans le cas des centroïdes, cela signifie que nous créons des séries temporelles synthétiques qui seraient issues de ce cluster. En d'autres termes, la courbe simulée par le centroïde I_{E_i}, si elle faisait partie de la base d'apprentissage, devrait être classée dans le cluster E_i.

Avant même de simuler les courbes de prix, nous devons définir la nature de la prédiction, *i.e.* choisir le calcul à appliquer sur les séries pour obtenir φ. Il ne faut pas oublier qu'une prédiction se fait à un instant t et qu'elle sera différente à $t+1$: φ_t répond à une interrogation sur l'évolution du prix d'une série de prix à un moment précis de celle-ci. Il y aura donc autant de φ_t que de jours avant la date de départ.

φ_t peut être binaire lorsque l'interrogation sur l'évolution du prix l'est aussi : le prix va-t-il augmenter dans 7 jours ? y aura-t-il une baisse de prix d'au moins 24 heures dans les 10

prochains jours ? Dans ce cas, nous simulons N courbes issues du cluster assigné au vol de test, et nous calculons pour chacune d'elles $\varphi_t \in [0, 1]$. En moyennant les prédictions, nous obtenons $\mathbb{P}(\varphi_t = 1)$ qui sera la prédiction finale du vol.

Mais on peut aussi vouloir prédire l'évolution des prix selon des cadrans tels que "Forte Hausse","Faible Hausse","Stable","Faible Baisse" et "Forte Baisse". Dans ce cas, la procédure reste inchangée à la différence près que φ_t prendra 5 valeurs correspondant aux différents cadrans. Pour toutes les simulations, chaque cadran est incrémenté lorsque l'évolution de prix correspond au critère de celui-ci et les résultats de tous les cadrans sont ensuite normalisés. Nous obtenons ainsi autant de $\mathbb{P}(\varphi_t = 1)$ que de cadran. Nous pouvons alors interpréter comme nous le souhaitons les résultats et même retrouver un φ_t binaire en fusionnant les cadrans.

4.5 Prédiction directe

Une autre méthode de prédiction de φ_t est d'appliquer la phase d'apprentissage à la fonction de calcul de φ_t. De cette manière, l'étape de segmentation n'est plus nécessaire et il suffit seulement de calculer φ_t pour tous les vols de la base d'apprentissage et pour tout t. L'étiquette à apprendre n'est donc plus le numéro du groupe mais directement φ_t. Avec cette méthode, il n'y a plus de perte d'informations due à la transformation en niveau de gris et aux simulations à partir de centroïdes. De plus si l'apprentissage se fait sur une valeur binaire de φ_t, les chances d'erreurs diminuent naturellement avec la diminution de la complexité du problème. L'inconvénient principal de cette approche est l'obligation d'avoir un modèle par nombre de jours avant la date de départ, et surtout de devoir recalculer les modèles à chaque nouvelle définition de φ contrairement à l'approche classique où seules les simulations sont à mettre à jour.

Une manière d'évaluer la pertinence de notre prédiction dite "classique" est donc de la comparer à l'apprentissage dit "direct" de l'évolution du prix. Comme expliqué précédemment, un φ_t correspond à une situation temporelle unique confrontant ainsi un modèle (celui classique) à une multitude d'autres (les modèles directs).

4.6 Approches séquentielles

Dans la section 1.2 nous parlions des techniques de yield management utilisant l'évolution de la demande comme paramètre principal dans la modification des prix. Nous pensons donc que les premières évolutions de prix peuvent nous indiquer la tournure que vont prendre les futurs changements de prix. L'utilisation de cette information est cependant soumise aux recherches des utilisateurs, qui sont notre seule source d'information. Les destinations populaires pendant les périodes scolaires seront les uniques séries pouvant bénéficier de ce complément d'information, car le volume de recherche est suffisamment conséquent pour que les combinaisons de dates, de routes et de durées de séjour se recoupent.

L'utilisation des premiers points peut se faire de différentes manières. Dans la prédiction directe par exemple, l'apprentissage de φ_t par les random forest peut se faire en utilisant les attributs contextuels décrit dans le Chapitre 1 : ces attributs dépendent comme φ_t du nombre de jours avant la date de départ et peuvent représenter la volatilité observée des prix, le prix courant,

le nombre de hausses ou de baisses observées. Ajoutés aux attributs statiques, ils apportent un plus non négligeable comme nous le verrons dans le Chapitre 5.

Il est aussi possible d'utiliser les premiers points comme unique paramètre de prédiction créant ainsi un nouveau type de classifieur. En utilisant le calcul de vraisemblance de l'algorithme Espérance-Maximisation du Chapitre 3, nous attribuons le cluster le plus probable à chacun des vols de test.

Enfin nous utilisons l'information des premiers points dans la phase d'apprentissage des comportements en ajoutant un paramètre à l'algorithme d'Espérance-Maximisation. L'apprentissage est alors couplé à l'étape de segmentation et permet de calculer la vraisemblance d'appartenance à un cluster grâce au V_i et aux niveaux de gris partiels des vols de test.

4.6.1 Classification uniquement par les premiers points

En utilisant le calcul de vraisemblance de l'EM aux premiers points d'une courbe, il est possible d'associer un vol à un cluster sans l'utilisation de ses attributs. Supposons que la courbe du vol i est observée jusqu'à une date T, on peut alors reconstruire le niveau de gris partiel du vol $X_i(s,t)$ pour $(s,t) \in \tilde{\mathcal{R}}$ où $\tilde{\mathcal{R}}$ est l'ensemble des cases dans le plan temps-rendements correspondant aux prix observés jusque là. On note alors $\tilde{X}_i$ l'ensemble des $X_i(s,t)$ pour $(s,t) \in \tilde{\mathcal{R}}$. Connaissant les $\tilde{X}_i$, les centroïdes I_y ainsi que les α_y, nous calculons alors la vraisemblance d'appartenance à chaque cluster des vols de test ainsi :

$$\mathbb{P}_{\alpha,I}(Y_i = y|\tilde{X}_i) = \frac{p_{\alpha,I_y}(\tilde{X}_i, Y_i = y)}{\sum_{j=1}^{C} p_{\alpha,I_j}(\tilde{X}_i, Y_i = j)} \propto \alpha_y \prod_{(s,t)\in\tilde{\mathcal{R}}} \frac{(I_y(s,t))^{X_i(s,t)} e^{-I_y(s,t)}}{X_i(s,t)!},$$

pour $y \in 1, \ldots, C$.

La classification se fait en choisissant le y qui maximise cette vraisemblance. Notons que le calcul des $\tilde{X}_i(s,t)$ est d'autant plus précis que la fréquence d'échantillonnage est élevée (voir section 2.3.1).

4.6.2 EM logit

L'EM logit est une méthode de classification accompagnée d'une étape de segmentation similaire à l'EM décrit dans la section 3.3.3. Contrairement à l'EM de la section 3.3.3, la classification et la segmentation sont indissociables. Les données d'entrée sont l'ensemble des niveaux de gris $\widehat{X}_i$ auquel nous ajoutons l'information des attributs $\widehat{V}_i$.

L'hypothèse principale sur laquelle cette méthode repose consiste à dire que les niveaux de gris X_i du vol i dépendent des attributs, uniquement à travers l'appartenance à un cluster particulier. En termes probabilistes, la loi conditionnelle de X_i sachant Y_i ne dépend pas des attributs. En revanche, la loi de Y_i dépend des attributs à travers une fonction $\psi(\cdot,\cdot) : \mathbb{P}(Y_i = y) = \psi(y|V_i)$.

Comme Y_i n'est pas observé, il s'en suit que la loi des X_i dépend des attributs sous la forme : $\mathbb{P}(X_i = x) = \sum_{y=1}^{C} \mathbb{P}(X_i = x|Y_i = y)\psi(y|V_i)$. Comme à la section 3.3.3, la probabilité $\mathbb{P}(X_i = x|Y_i = y)$ dépendra du paramètre $I_y \in (\mathbb{R}_+^*)^{\mathcal{R}}$ et sera donnée par la loi de Poisson.

Nous serons donc encore en présence d'un modèle de mélange de Poisson. La nouveauté est dans le calcul des probabilités d'appartenance aux clusters qui, cette fois, vont être calculées en utilisant les attributs : les paramètres (α_y) sont remplacés par les fonctions $(\psi(y|v))$. Pour pouvoir estimer ces fonctions, nous utiliserons une paramétrisation de type logit (cf équation (4.7)).

Description algorithme

L'algorithme s'apparente à l'algorithme d'Espérance-Maximisation de la section 3.3.3, à cela près qu'il introduit l'information des attributs V_i dans l'étape de segmentation.

Données d'entrée Les données d'entrée sont celles de l'EM décrit dans la section 3.3.3 ainsi que les attributs V_i de l'ensemble des vols $i \in 1, \dots, n$.

Initialisation L'initialisation est quasiment identique à celle de l'EM de la section 3.3.3, mais ajoute un troisième paramètre à optimisé, donné par la fonction ψ définie plus haut. Une première segmentation est donc effectuée par les K-Means, initialisant les centroïdes I_y ainsi que les couples $(X_i, \hat{Y}_i)$. On utilise alors les couples $(V_i, \hat{Y}_i)$ pour optimiser la fonction ψ.

Fonction de vraisemblance La log densité des variables observées X_i, $i = 1, \dots, n$, sachant les attributs observés V_i, $i = 1, \dots, n$ s'écrit :

$$\begin{aligned}
\log p_\theta(x_1, \dots, x_n | v_1, \dots, v_n) &= \sum_{i=1}^{n} \log p_\theta(x_i | V_i) \\
&= \sum_{i=1}^{n} \log \sum_{y_i=1}^{C} p_\theta(x_i | V_i, y_i) p_\theta(y_i | V_i) \\
&= \sum_{i=1}^{n} \log \sum_{y_i=1}^{C} p_\theta(x_i | y_i) p_\theta(y_i | V_i) \ ,
\end{aligned}$$

où, pour $\theta = (I, \psi)$,

$$p_\theta(y_i | V_i) = \psi(y_i | V_i) \tag{4.1}$$

et

$$p_\theta(x_i | y_i) = \prod_{r \in \mathcal{R}} \frac{\mathrm{e}^{x_i(r)(\log I(r|y_i)) - I(r|y_i))}}{x_i(r)!} \ .$$

En factorisant tous les termes factoriels dans le log de la log-vraisemblance, on obtient :

$$\log p_\theta(x_1, \dots, x_n | v_1, \dots, v_n) = (\dots) + \sum_{i=1}^{n} \log \sum_{y_i=1}^{C} \exp(\ell(y_i | V_i, I)) \psi(y_i | V_i) \ , \tag{4.2}$$

où la constante $(\dots)$ ne dépend pas de θ et où

$$\ell(x|y,I) = \sum_{r\in\mathcal{R}} \mathrm{x_i(r)}(\log(\mathrm{I(r|y_i)}) - \mathrm{I(r|y_i)}) \ . \tag{4.3}$$

Etape E L'étape E consiste à calculer, connaissant les attributs $v_1,\dots,v_n$ et les deux paramètres $\theta = (I,\psi)$ et $\theta' = (I',\psi')$, l'espérance conditionnelle de la log-densité jointe au paramètre θ' sachant les variables observées X_i sous le paramètre θ :

$$Q(\theta', x_1,\dots,x_n|\theta, v_1,\dots,v_n) = \sum_{i=1}^{n}\sum_{y_i=1}^{C} \left[\log p_{\theta'}(x_i,y_i|V_i)\right]\ p_\theta(y_i|x_i,V_i) \ .$$

Par commodité, nous allons par la suite omettre $x_1,\dots,x_n$ et $v_1,\dots,v_n$ dans la notation de Q et ecrivons simplement $Q_n(\theta'|\theta)$

Nous obtenons alors :

$$\begin{aligned} Q_n(\theta'|\theta) = (\dots) + \sum_{y=1}^{C}\left(\sum_{r\in\mathcal{R}} B_n(y,r|\theta)\log I'(r|y) - A_n(y|\theta)\sum_{r\in\mathcal{R}} I'(r|y)\right) \\ + \sum_{y=1}^{C}\sum_{i=1}^{n}\left(\log\psi'(y|V_i)\right)\ p_\theta(y|x_i,V_i) \ , \end{aligned}$$

où $(\dots)$ est une constante indépendante de θ', et où l'on a :

$$p_\theta(y|x,v) = \frac{p_\theta(x,y|v)}{\sum_{y'} p_\theta(x,y'|v)} \ , \quad \text{and} \quad p_\theta(x,y|v) = \psi(y|v)\ \exp\ell(x|y,I) \ ,$$

avec ℓ définie dans (4.3) et

$$A_n(y|\theta) = \sum_{i=1}^{n} p_\theta(y|x_i,V_i) \quad \text{and} \quad B_n(y,r|\theta) = \sum_{i=1}^{n} x_i(r)\, p_\theta(y|x_i,V_i) \ . \tag{4.4}$$

Etape M Cette étape consiste à maximiser $Q(\theta'|\theta)$ en θ' pour un θ donné.

Comme pour l'EM de la section 3.3.3, l'optimisation en I' avec le poids ψ peut être traité séparément.

On obtient :

$$I'(r|y) = \frac{B_n(y,r|\theta)}{A_n(y|\theta)}, \quad r\in\mathcal{R}, y = 1,\dots,C \ . \tag{4.5}$$

L'optimisation en ψ dépend de la fonction utilisée dans l'ensemble des fonction Ψ. On obtient :

$$\psi' = \arg\max_{\phi\in\Psi} \sum_{i=1}^{n}\sum_{y=1}^{C}\left(\log\phi(y|V_i)\right)\ p_\theta(y|x_i,V_i) \ . \tag{4.6}$$

La classe Ψ représente un ensemble de fonctions possibles pour ψ. Dans le cas particulier où Ψ est constant en $\mathcal{V}$, c'est-à-dire $\Psi = S_C$, on retrouve

$$\psi'(y) = \frac{A_n(y|\theta)}{\sum_{y'=1}^{C} A_n(y'|\theta)} \ .$$

Ce cas ne nous intéresse pas ici puisque l'on souhaite faire intervenir les attributs. Nous avons choisi la classe des fonctions de type logit. Dans ce cas :

$$\psi(y, v) = logit_\varphi(y|V) = \frac{e^{\varphi_y^T V}}{\sum_{y=1}^{C} e^{\varphi_y^T V}} \tag{4.7}$$

La fonction ψ est entièrement décrite par le paramètre $\varphi \in \mathbb{R}^p$ (espace des attributs). Le calcul de ψ' revient donc à calculer le nouveau paramètre φ'

On obtient alors :

$$\varphi' = \arg\max_{\varphi}(\sum_{i=1}^{n}\sum_{y=1}^{C}(\varphi_y^T V_i)p_\theta(y|x_i, V_i) - \sum_{i=1}^{n}\sum_{y=1}^{C}\log(\sum_{j=1}^{C} e^{\varphi_j^T V_i})p_\theta(y|x_i, V_i))$$

$$= \arg\max_{\varphi}(\sum_{i=1}^{n}\sum_{y=1}^{C}(\varphi_y^T V_i)p_\theta(y|x_i, V_i) - \sum_{i=1}^{n}\log\sum_{y=1}^{C} e^{\varphi_y^T V_i})$$

En posant $\varphi_C = 0$,

$$\varphi' = \arg\max_{\varphi}(\sum_{i=1}^{n}\sum_{y=1}^{C-1}\varphi_y^T V_i p_\theta(y|x_i, V_i) - \sum_{i=1}^{n}\log(1 + \sum_{y=1}^{C-1} e^{\varphi_y^T V_i})$$

En dérivant, on obtient que φ' est solution de l'équation :

$$0 = \sum_{i=1}^{n} V_i p_\theta(y|x_i, V_i) - \sum_{i=1}^{n} \frac{V_i e^{\varphi_y^T V_i}}{(1 + \sum_{j=1}^{C-1} e^{\varphi_y^T V_i})} \quad , \varphi_1, \ldots, \varphi_{C-1} \in \mathbb{R}$$

Données à la sortie

A l'arrêt de l'algorithme, on dispose d'un estimateur $\hat{\theta} = (\hat{I}, \hat{\psi})$ (ou $\hat{\theta} = (\hat{I}, \hat{\varphi})$ dans la paramétrisation logit).

Comme pour l'EM, l'EM logit calcule le groupe le plus probable pour chaque vol i grâce à la fonction de vraisemblance $p_\theta(x, y|v)$ et aux paramètres précédemment optimisés. Elle est appliquée aux $X_i(s, t)$ afin de déterminer pour chaque vol i l'étiquette $\hat{Y}_i$.

$$\hat{Y}_i = \arg\max_y \mathbb{P}_\theta(Y_i = y|V_i) = \arg\max_y \psi(y|V_i)$$

Il est aussi possible après la collecte de quelques point de rajouter l'information des premières variations dans le calcul de la vraisemblance. On définit alors $X_i = (\tilde{X}_i, \hat{X}_i)$ avec $\tilde{X}_i$ les points

observés, et $\hat{X}_i$ les futures évolutions. On aurait donc pour tout événement $\hat{\varphi}$ dépendant de $\hat{X}_i$:

$$\mathbb{P}_\theta(\hat{\varphi}|V_i, \tilde{X}_i) = \sum_{y=1}^{C} \mathbb{P}_\theta(\hat{\varphi}|V_i, \tilde{X}_i, Y_i = y)\mathbb{P}_\theta(Y_i = y|V_i, \tilde{X}_i)$$

$$= \sum_{y=1}^{C} \mathbb{P}_\theta(\hat{\varphi}|Y_i = y)\mathbb{P}_\theta(Y_i = y|V_i, \tilde{X}_i)$$

avec

$$\mathbb{P}_\theta(Y_i = y|V_i, \tilde{X}_i) = \frac{p_\theta(\tilde{X}_i|y)\psi(y|V_i)}{\sum_{j=1}^{C} \mathbb{P}_\theta(\tilde{X}_i|j)\psi(j|V_i)} \propto p_\theta(\tilde{X}|y)\psi(y|V_i).$$

Dans ce cas, les métriques d'évaluation de la segmentation de l'EM peuvent être utilisées en même temps que les performances en prédiction.

Choix des attributs

Pour limiter le temps de calcul et l'utilisation de ressources, nous devons réduire le nombre d'attributs utilisé. Pour cela nous utilisons la technique de sélection de variables précédemment décrite et choisissons les 10 variables les plus discriminantes. Nous devons ensuite transformer les variables qualitatives en des variables quantitatives. Pour cela nous créons une variable par niveaux existant, créant ainsi un grand nombre de variables binaires : dans le cas de la compagnie aérienne, nous allons créer une variable booléenne nommée "*Air France*", puis une autre nommée "*EasyJet*" etc.

4.7 Conclusion et perspectives

Dans ce chapitre nous avons détaillé les différentes étapes qui nous permettent de fournir une prédiction d'évolution de prix à tous les résultats d'une recherche utilisateur. La première étape consiste à associer à chaque résultat sa classe la plus probable : après avoir segmenté les comportements des vols de la base des niveaux de gris, nous avons appliqué un algorithme d'apprentissage supervisé sur la bases des attributs. Cette base est constituée de l'ensemble de caractéristiques des vols V_i et l'étiquette associée E_i correspondant à la classe attribuée lors de l'étape de segmentation. L'algorithme d'apprentissage supervisé va alors créer des régles de classification nous permettant de classer les vols de test dans leur groupe le plus probable uniquement grâce à leurs attributs.

Pour cela, nous avons donc utilisé des algorithmes bien connus tels que CART et C4.5, des arbres de décision permettant l'extraction de règles faciles à interpréter, Adaboost une méthode de boosting basé sur la sélection itérative de classifieurs faibles et les Random Forests, mélange des sous-espaces aléatoires et du Bagging qui effectue un apprentissage sur de multiples arbres de décision entraînés sur des sous-ensembles de données tirés aléatoirement et tous différents. Nous avons par ailleurs introduit une extension de l'algortithme EM défini dans le chapitre précédent, permettant d'effectuer l'étape de segmentation et de classification simultanément.

Nous obtenons donc pour chaque vol de test, une étiquette E_i à laquelle est associé le centroïde du cluster en question. Grâce aux simulations (Chapitre 2) issues de ce centroïde nous calculons alors les probabilités $\mathbb{P}(\varphi_t^{(i)} = 1), \forall t \in [T_{init}, T_0 - 7]$ (si l'on considère une prédiction à 7 jours). Il faut alors définir un seuil au-delà duquel $\mathbb{P}(\varphi_t^{(i)} = 1)$ est considéré comme une prédiction à la hausse ou à la baisse.

Nous avons par ailleurs introduit la notion de prédiction directe qui consiste à prédire directement l'évolution de la série, c'est-à-dire $\mathbb{P}(\varphi_t = 1)$, sans passer par l'étape de segmentation en comportements similaires. Cette méthode impliquant un modèle par jours avant la date de départ devient coûteuse en temps de calcul, mais impose surtout le renouvellement de l'apprentissage à chaque changement de définition de φ_t.

Enfin nous avons défini les différentes approches possibles quant à l'utilisation des premiers points dans l'affinement de la prédiction. Tout d'abord, nous avons utilisé les attributs contextuels décrit dans le chapitre 1 comme attributs dans la prédiction directe. Les informations sur la demande à l'instant t, sur la volatilité des premières variations de prix et la valeur actuelle du prix permettent d'affiner l'apprentissage supervisé. Ensuite, nous avons montré comment il était possible d'utiliser l'information des premiers points pour créer un niveau de gris partiel et calculer la vraisemblance partielle d'appartenance aux clusters.

Toutes ces méthodes impliquent d'avoir un modèle par jour avant la date de départ et donc sont consommatrices de ressources (temps de calcul et mémoire). En revanche, elles permettront d'estimer l'utilité d'une telle information et de comparer les résultats obtenus à ceux de l'approche par les modèles.

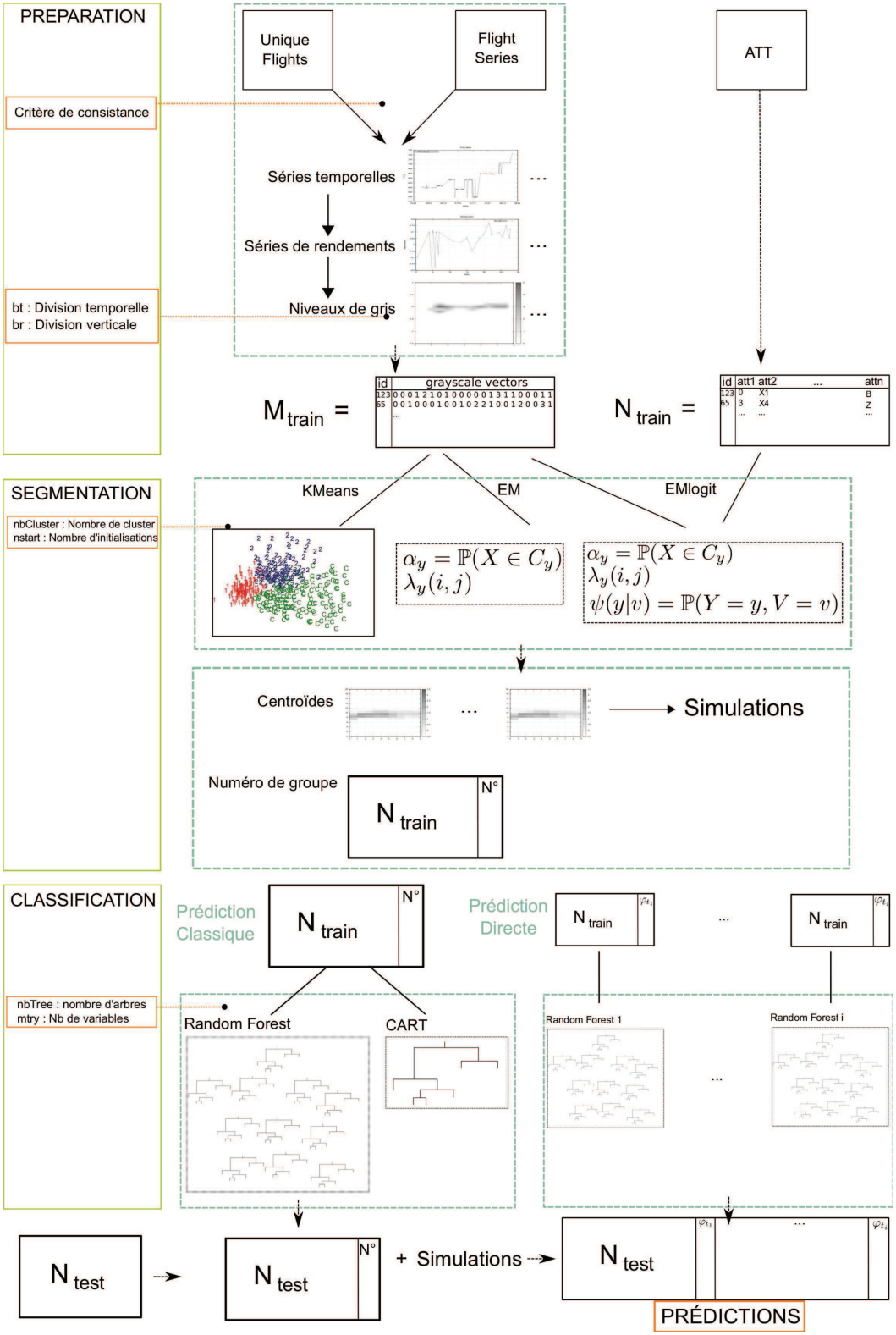

FIGURE 4.5 – Workflow - Vue Détaillée