# L’utilisation de la technique ‘Gradient Boosting’ pour modéliser les prix d’un échantillon de maisons à Ames, Iowa

Introduction

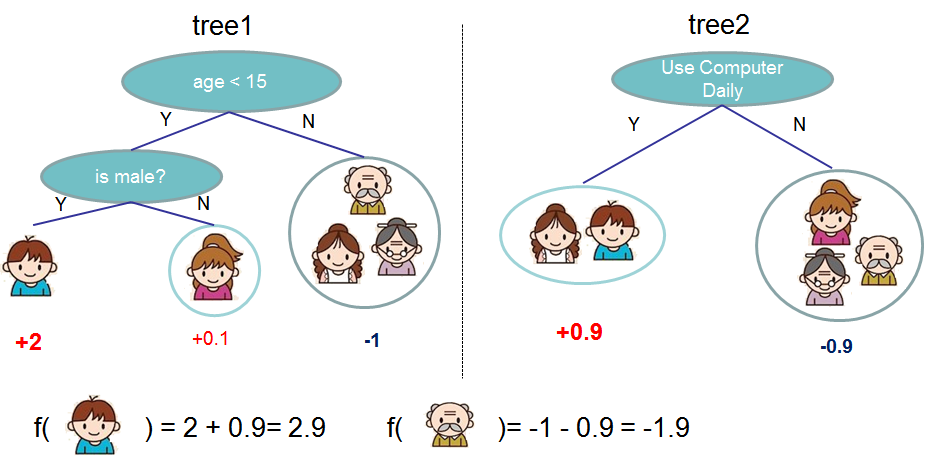
**Gradient Boosting** est une technique dans le domaine du Machine Learning qui permet de résoudre des problèmes de régression et de classification en produisant un modèle prédictif à partir d’un ensemble de modèles prédictifs plus naïves, typiquement des arbres de décision. Le modèle est construit en plusieurs étapes tout comme d’autres méthodes de boosting.

Forêt d’arbres décisionnels : notre modèle prédictif

Nous avons opté pour une forêt d’arbres décisionnels comme modèle prédictif pour une multitude de raisons énumérées si dessous :

1. Ne nécessite pas un prétraitement exhaustif des données pour de bon résultats
2. Invariabilité par translation, par changement d’échelle, par transformation monotone des coordonnées
3. Très interprétable
4. Peut-être entraîner efficacement

Malgré tous ces avantages, les arbres de décision sont généralement instables rendant la calibration du modèle une affaire délicate.



Voilà un exemple d’une forêt de deux arbres décisionnels qui prédit si la personne en question aime les jeux vidéo. La prédiction finale est la somme des prédictions de chaque arbre. On peut constater que pour les meilleurs résultats, il est essentiel que les arbres soient complémentaires. Mathématiquement, notre modèle peut être écrit sous la forme suivante :

Ou est le nombre d’arbres décisionnels, est la fonction appartenant à l’espace qui contient tous les arbres.

L’entraînement

Pour entraîner les arbres de décision, on fait comme pour tous les modèles d’apprentissage supervisé : on définit une fonction objective et on l’optimise.

On choisit l’objectif régularisé suivant :

Vu que ce défi est plus complexe que les problèmes d’optimisation traditionnels, une approche additive sera nécessaire. Il faut améliorer ce qui a déjà été appris en ajoutant un arbre à la fois. On va noter la prédiction de l’étape par , donc on aura :

Un arbre f(x) est défini par :

Ou est le vecteur des scores des feuilles et est la fonction qui associe une feuille à chaque exemple et le nombre de feuilles.

Idéalement, pour choisir un arbre, il faudra énumérer toutes les possibilités et choisir la meilleure ce qui n’est pas chose possible en pratique. Plutôt, on optimise l’arbre un niveau à la fois et on essaye de séparer une feuille en deux (Splitter) tout en améliorant la fonction objective. Cette séparation est validée que si le gain de score est plus grand qu’un certain seuil.

Application

**Prétraitement des données**

On a énuméré le nombre d’élément nulle de chaque colonne comme si dessous

Alley 2721

BsmtCond 82

BsmtExposure 82

BsmtFinSF1 1

BsmtFinSF2 1

BsmtFinType1 79

BsmtFinType2 80

BsmtFullBath 2

BsmtHalfBath 2

BsmtQual 81

BsmtUnfSF 1

Electrical 1

Exterior1st 1

Exterior2nd 1

Fence 2348

FireplaceQu 1420

Functional 2

GarageArea 1

GarageCars 1

GarageCond 159

GarageFinish 159

GarageQual 159

GarageType 157

GarageYrBlt 159

KitchenQual 1

LotFrontage 486

MSZoning 4

MasVnrArea 23

MasVnrType 24

MiscFeature 2814

PoolQC 2909

SalePrice 1459

SaleType 1

TotalBsmtSF 1

Utilities 2

Nous avons remplacé ces éléments par la valeur moyenne de chaque colonne, pour ne pas influencer la valeur moyenne, ensuite on a fait appliquer le ‘onehot encoding’ sur tous les valeurs non numériques. A la fin on a une matrice à 290 traits.

**Méthode d’entrainement**

On a opté pour la librairie **sklearn**,plus précisément la classe **GradientBoostingRegressor** pour représenter et entraîner le modèle. Les hyper-paramètres sont réparties en trois groupes :

1. **Paramètres spécifiques à l’arbre :** ce qui affecte chaque arbre individuel.
   1. **min\_samples\_split :** représente le nombre minimal d’échantillons dans une node qu’on veut séparer en 2 ce qui permet de contrôler le over-fitting. Une valeur élevée aboutira à un under-fitting.
   2. **min\_samples\_leaf :** est le nombre d’observations qu’il faut dans les feuilles. Tout comme le paramètre d’avant, il permet de contrôler la complexité du modèle.
   3. **max\_depth :** la profondeur maximale de l’arbre.
   4. **max\_features :** le nombre de traits à considérer à chaque niveau de séparation qui sont choisis aléatoirement.
2. **Paramètres pour le boosting :** ce qui affecte l’opération de boosting du modèle
   1. **learning\_rate :** cela détermine l’impact de chaque arbre sur la prédiction finale
   2. **n\_estimators :** indique le nombre d’arbres en série
   3. **subsample :** indique la fraction des observations à choisir pour chaque arbre, une sélection qui se fait aléatoirement
3. **Autres paramètres divers :** autres paramètres pour le fonctionnement général du modèle.
   1. **loss :** correspond à la fonction de perte
   2. **random\_state :** un seed pour contrôler les valeurs aléatoires

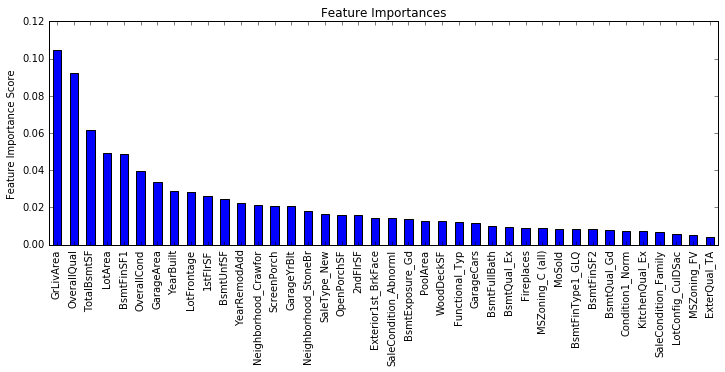
On a utilisé la technique de cross validation via erreur moyenne quadratique pour comparer les modèles, et pour optimiser les paramètres on utilise la fonction **GridSearchCV.**

On a commencé par un modèle de base pour avoir une référence initiale, les paramètres sont choisis aléatoirement. Avec le rapport suivant :

Model Report

Mean squared error: 210362028.60083726

CV Score : Mean - -7.078834e+08 | Std - 1.780542e+08 | Min - -1.001023e+09 | Max - -4.996456e+08



Pour calibrer les paramètres, on a cherché à stabiliser les paramètres du boosting d’abord pour avoir une bonne performance lors de l’étalonnage des paramètres spécifiques aux arbres. Ceci nécessite qu’on fixe le **learning\_rate** à une valeur relativement élevée donc un nombre élevé pour le **n\_estimators.**

Une fois ces paramètres fixés on se concentrera sur la calibration des paramètres.