XGBoost, origines et applications

Damien DOUTEAUX

Sommaire

Sommaire

Aspects théoriques

Mise en œuvre

Applications

Conclusion

Outils de veille

- Moteurs de recherche

- Alertes mails

- Réseaux sociaux et autres



Généralités sur XGBoost

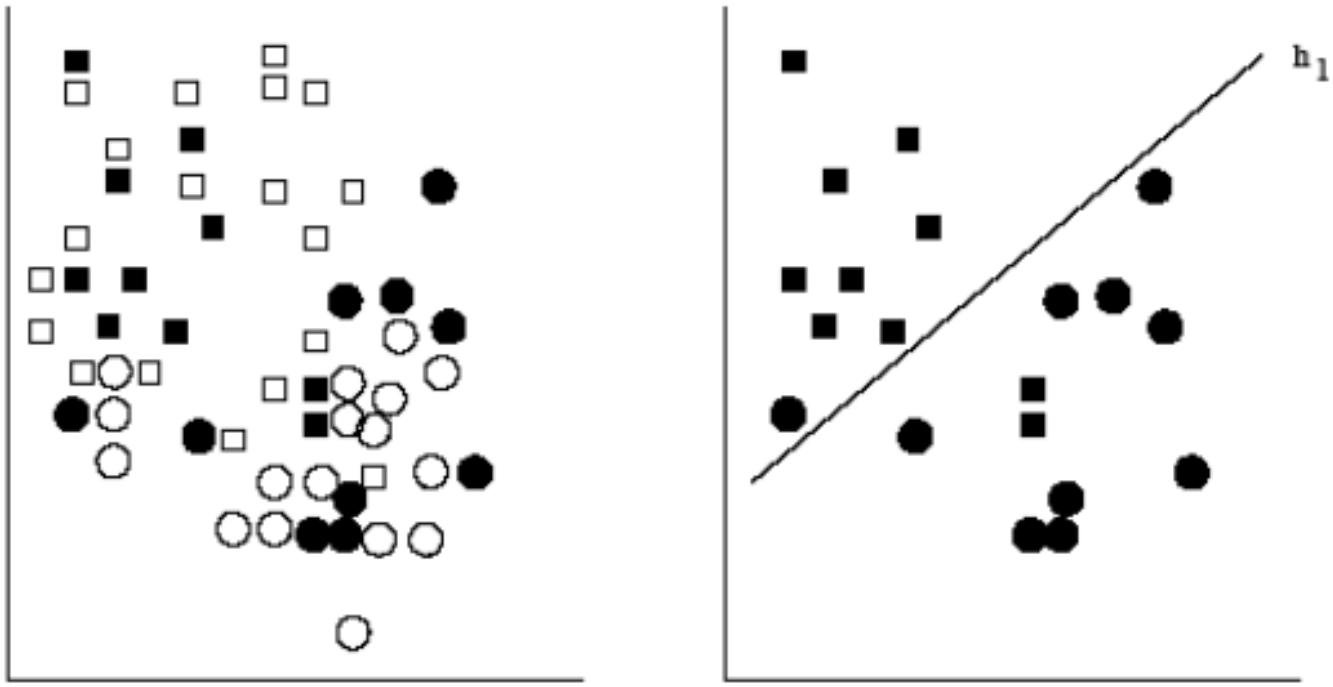
XGBoost : *E**X**treme **G**radient **Boost**ing*

- **Flexibilité** Régression, classification,...
- **Portabilité** Windows, Linux, OS X
- **Multi-langages** Python, R, JAVA, C++, Scala,...
- **Distribué** Yarn, Spark, Flink, AWS, Azure,...
- **Performance** Optimisé et expensif

Le boosting

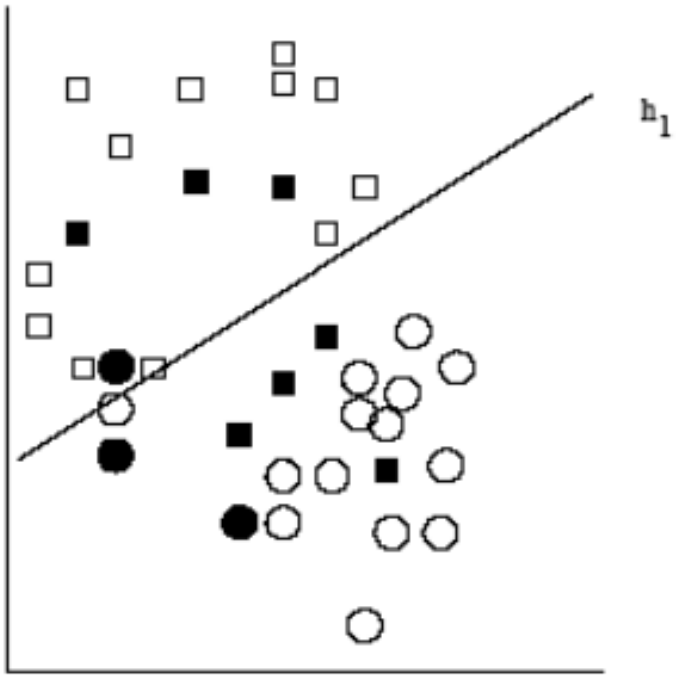
- Une stratégie adaptative.
- Convertir des règles peu performantes en (très) bonne prédiction.
- Réduction variance et biais.
- Convergence rapide.
- Sensible au bruit.

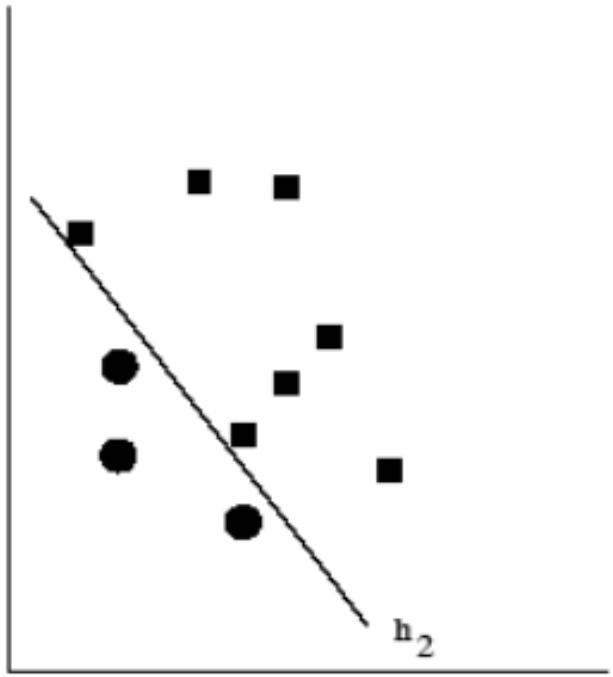
Premier modèle



Source : Cours Machine Learning, Haytham Elghazel

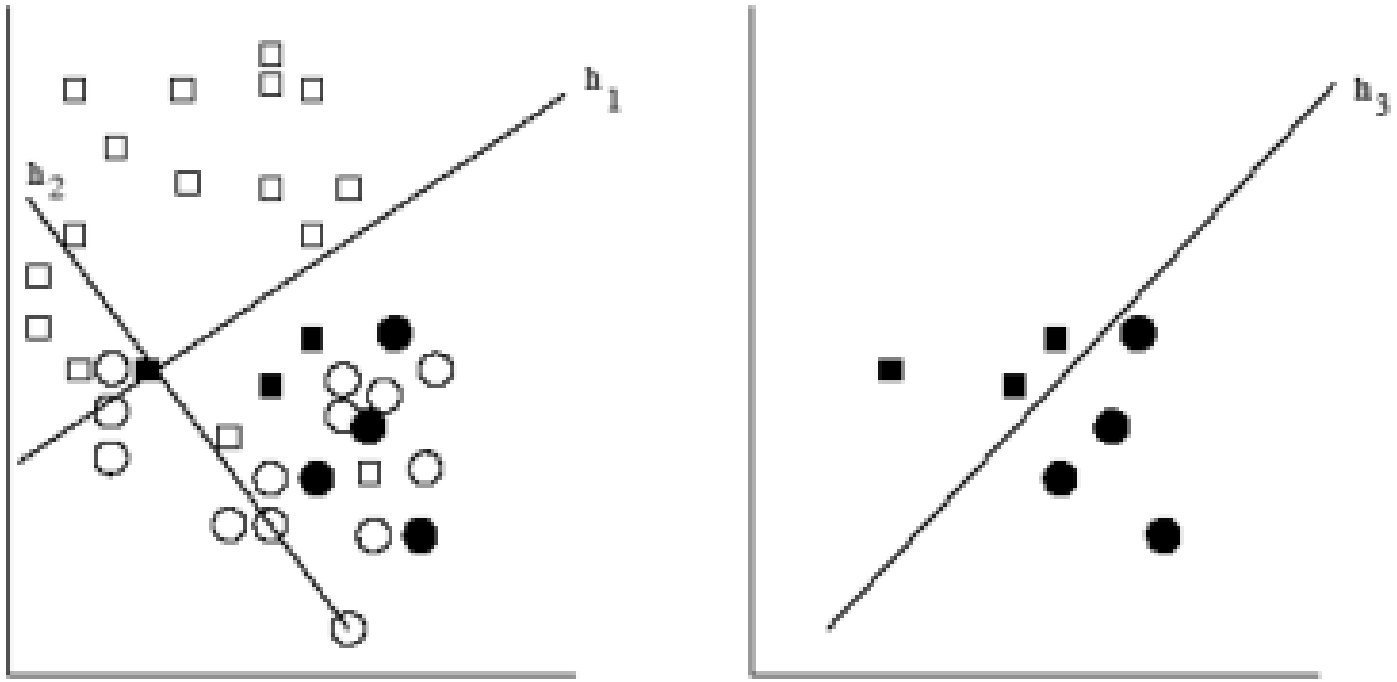
Deuxième modèle





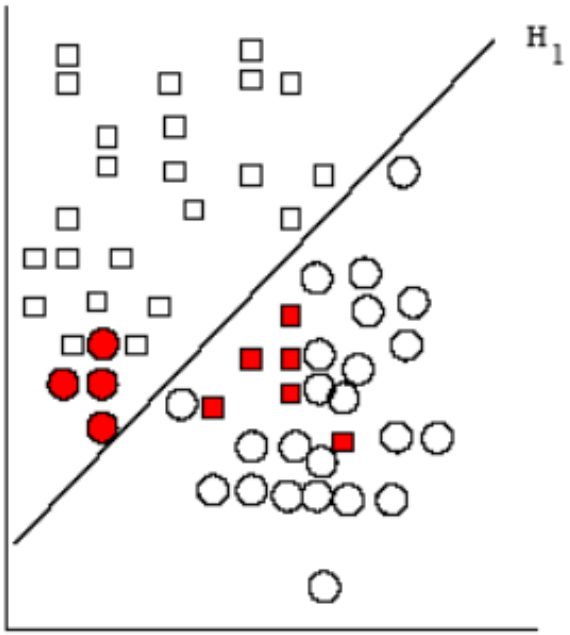
Source : Cours Machine Learning, Haytham Elghazel

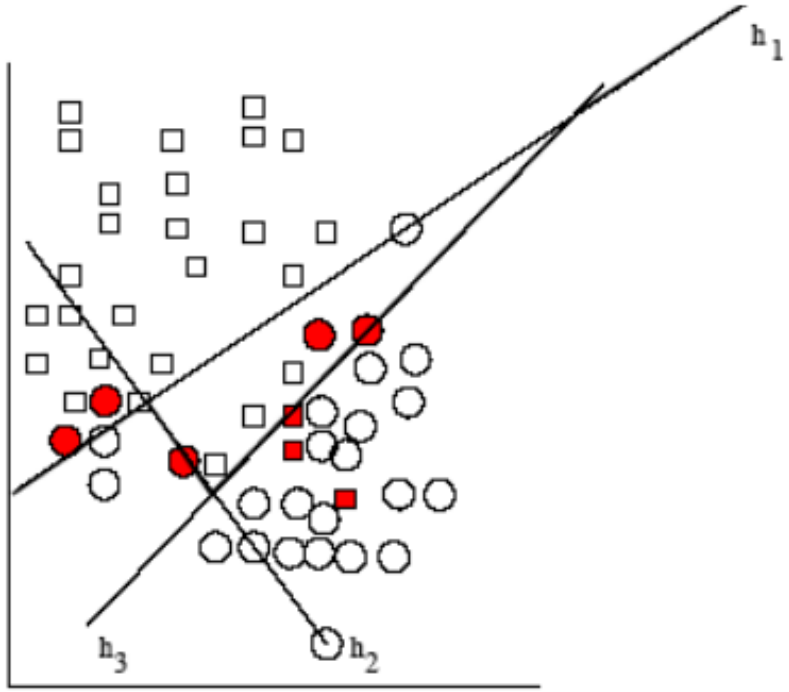
Troisième modèle



Source : Cours Machine Learning, Haytham Elghazel

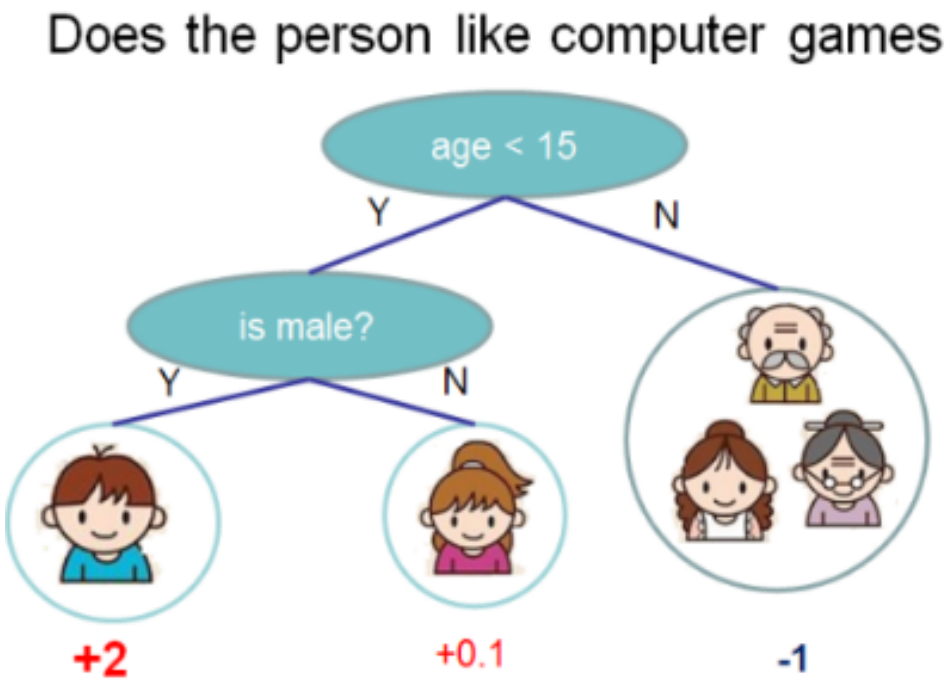
Vote majoritaire





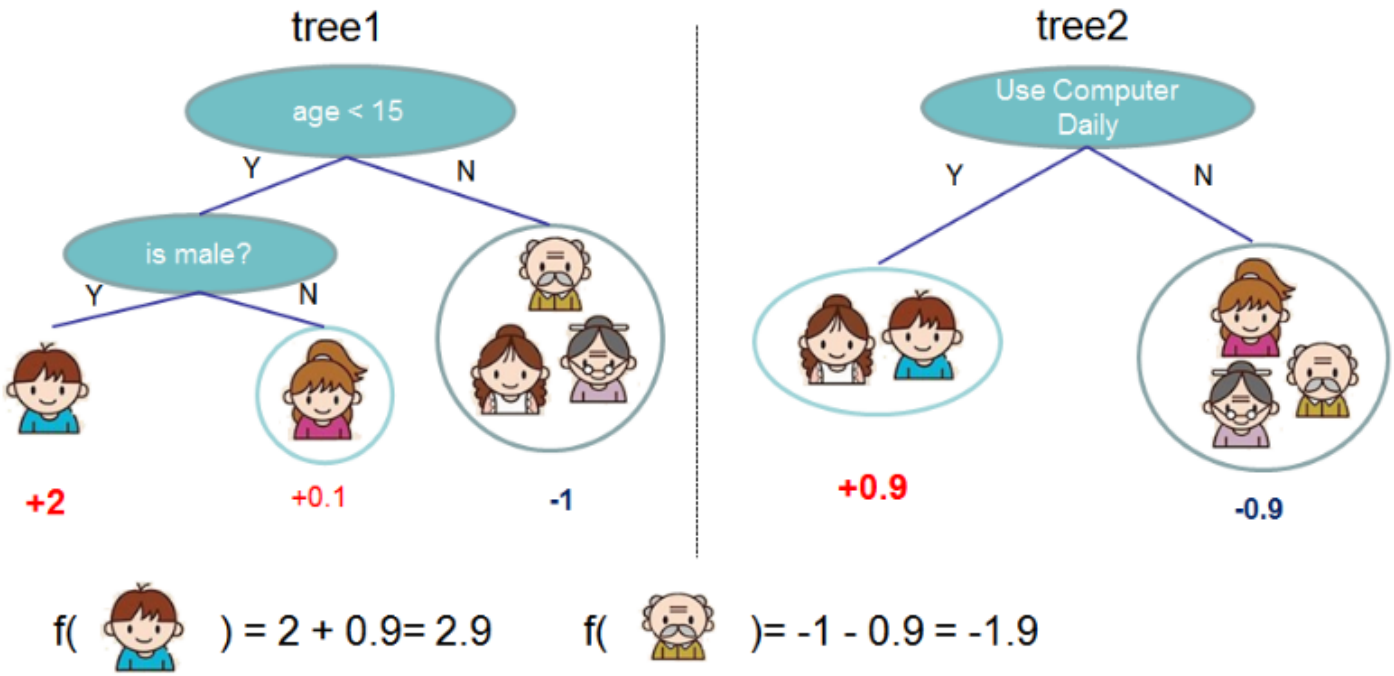
Source : Cours Machine Learning, Haytham Elghazel

Un arbre simple (CART)



Source : https://xgboost.readthedocs.io/en/latest/model.html

PLUSIEURS ARBRES VALLENT MIEUX QU'UN

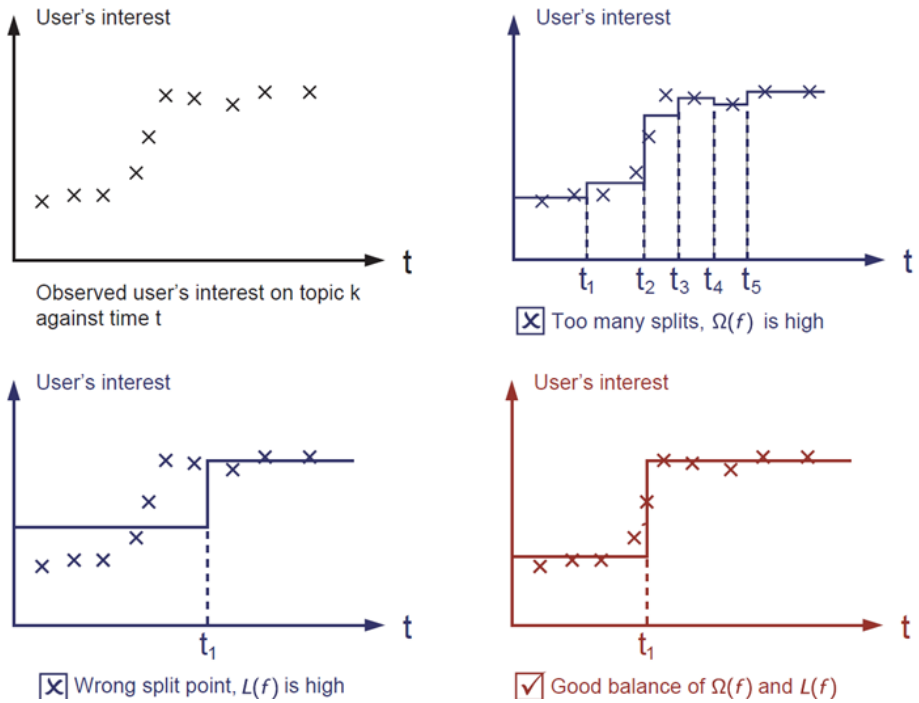


Additive training (Boosting)

Source : https://xgboost.readthedocs.io/en/latest/model.html

CHOIX DE L'ARBRE À AJOUTER

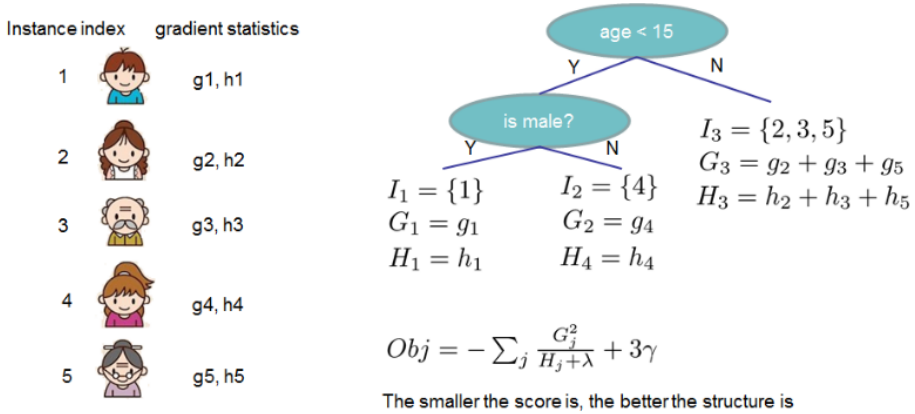
- **Fonction objectif** $\text{obj}(\Theta) = \mathcal{L}(\Theta) + \Omega(\Theta)$



Source : *https://xgboost.readthedocs.io/en/latest/model.html*

CHOIX DE L'ARBRE À AJOUTER

- **Fonction de perte** $\mathcal{L}(t) = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right]$
- **Complexité** $\Omega(t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} \omega_j^2$



Source : *https://xgboost.readthedocs.io/en/latest/model.html*

Choix de l'arbre à ajouter

- Énumérer toutes les structures d'arbres possibles.
- Calculer l'objectif pour chaque structure.
- Trouver l'optimal et optimiser les feuilles.

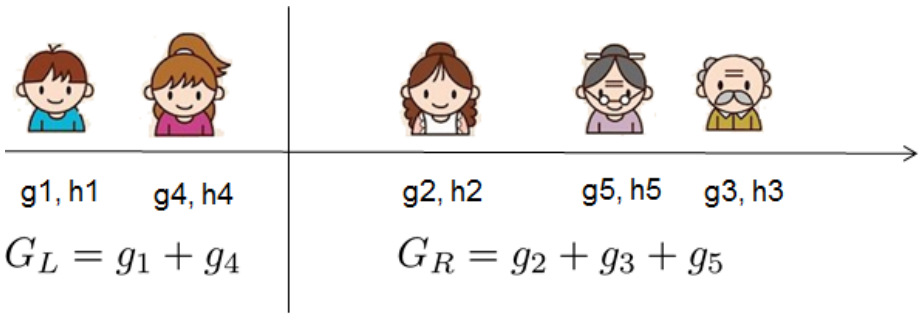
Choix de l'arbre à ajouter

- Énumérer toutes les structures d'arbres possibles.
- Calculer l'objectif pour chaque structure.
- Trouver l'optimal et optimiser les feuilles.

En pratique, construction des arbres au coup par coup.

CHOIX DE L'ARBRE À AJOUTER

- **Le gain** $\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right]-\gamma$



On ne s'arrête pas si Gain<0.

Source : https://xgboost.readthedocs.io/en/latest/model.html

Plus qu'une méthode de Boosting

- Prise en compte de la régularisation.
- Calcul en parallèle.
- Support de Hadoop.
- Possibilité d'adaptation des fonctions objectifs.
- Prise en charge des valeurs manquantes.
- Version améliorée de l'élagage
- Cross-validation native

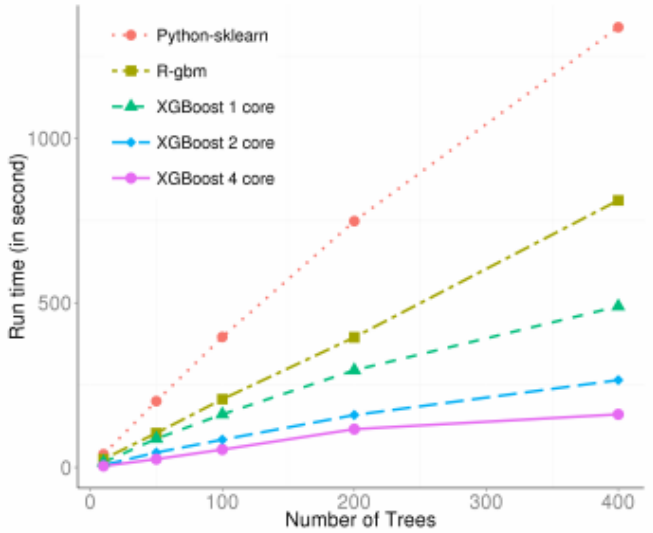
Importance des variables

- Entraîner les arbres.
- Pour chaque chaque variable :
  - Compter le nombre de fois où elle est sélectionnée.
  - Pondérer par la diminution d'erreur engendrée.
  - Moyenner sur les arbres.

Performances

La rapidité est le but initial de XGBoost :

- **Mémoire** Pas de mémoire dynamique.
- **Cache** Éviter de surcharger la mémoire.
- **Amélioration modèle** Voir précédemment.
- **Conception** Parallélisation en arrière plan.
- **Données externalisées** Si mémoire insuffisante.

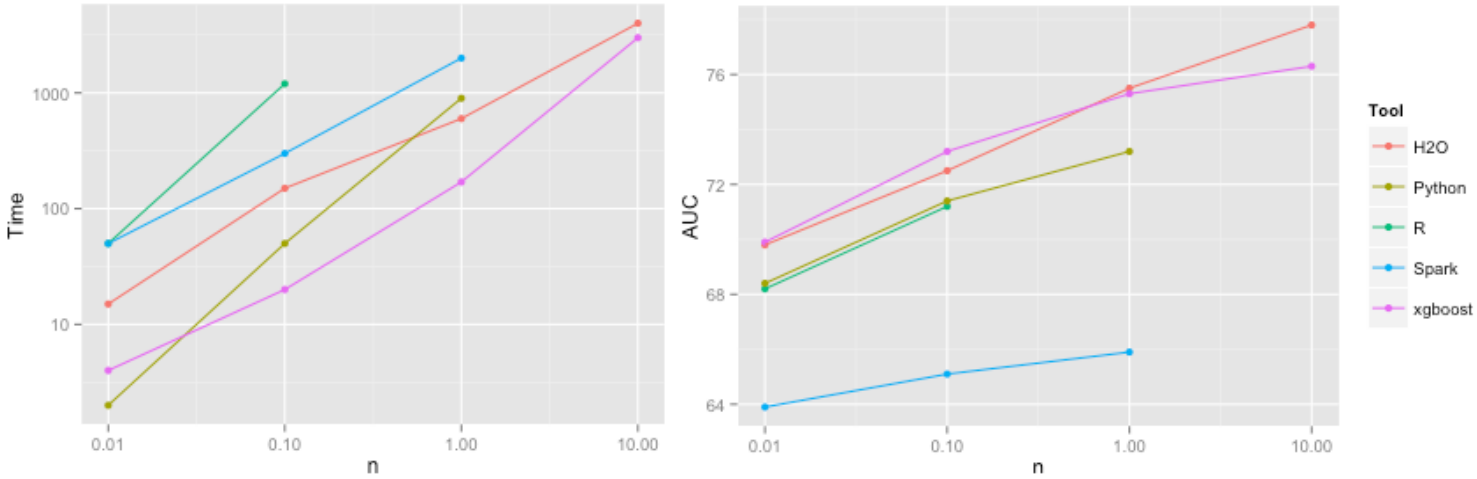


Source : *http://www.jmlr.org/proceedings/papers/v42/chen14.pdf*

Performances

La rapidité est le but initial de XGBoost :

- **Mémoire** Pas de mémoire dynamique.
- **Cache** Éviter de surcharger la mémoire.
- **Amélioration modèle** Voir précédemment.
- **Conception** Parallélisation en arrière plan.
- **Données externalisées** Si mémoire insuffisante.



Source : *http://datascience.la/benchmarking-random-forest-implementations/* [2015]

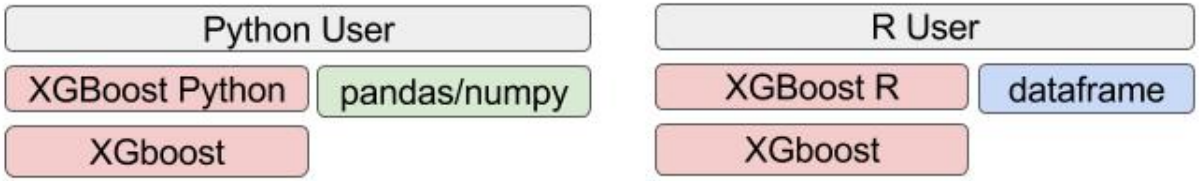
Sur le boosting

- **1989** Boosting (R. Schapire)
- **1996** AdaBoost (Y. Freund et R. Schapire)
- **1999** GBM (L. Breiman puis J. Friedman)
- **2014** XGBoost (T. Chen)

Bref historique

Pour XGBoost

- **Mars 2014** Première release
- **Mai 2014** Python

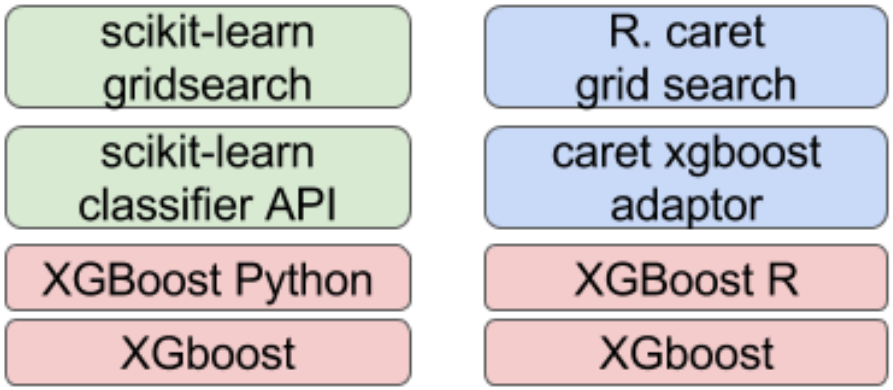


Source : http://homes.cs.washington.edu/~tqchen/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html

Bref historique

Pour XGBoost

- **Mars 2014** Première release
- **Mai 2014** Python
- **Septembre 2014** Parallélisation, R
- **Mai 2015** YARN, gestion HDFS, SKLearn wrapper

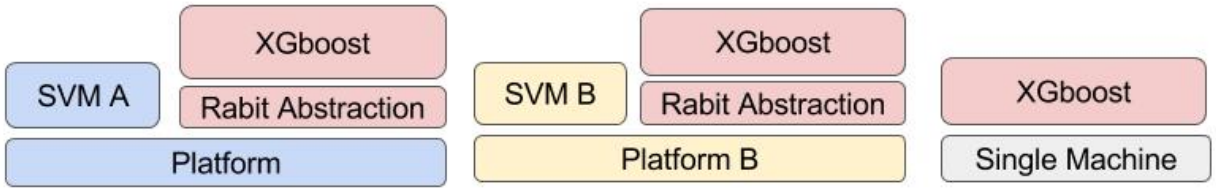


Source : `http://homes.cs.washington.edu/~tqchen/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html`

Bref historique

Pour XGBoost

- **Mars 2014** Première release
- **Mai 2014** Python
- **Septembre 2014** Parallélisation, R
- **Mai 2015** YARN, gestion HDFS, SKLearn wrapper
- **Janvier 2016** API JAVA, Flink, Spark, améliorations
- **Juillet 2016** Totale compatibilité JVM (Spark,...)



Source : *http://homes.cs.washington.edu/~tqchen/2016/03/10/story-and-lessons-behind-the-evolution-of-xgboost.html*

Trois familles de paramètres

Paramètres génériques

Pour définir par exemple quelle méthode Boosting sera utilisée.

Paramètres liés au Boosting

Pour paramétrer le booster choisi.

Paramètres liés à l'apprentissage

Dépend de la tâche d'apprentissage (classification,...).

Paramètres génériques

- `Booster` Linéaire ou arbre.
- `Silent` Affichage de messages.
- `Nthread` Par défaut le maximum possible.

Paramètres liés au Boosting

Pour celui sur les arbres. Douze paramètres utiles...

- `Eta` Contrôle du niveau d'apprentissage.
- `Min_child_weight` Pour contrôler l'over/under-fitting
- `Max_depth` Pour contrôler l'over-fitting.
- `Lambda` Pour de la régularisation.
- `Gamma` Valeur minimale de gain pour diviser.
- ...

Paramètres liés à l'apprentissage

- `Objective` Fonction objectif à minimiser (linéaire, softmax, softprob,...).
- `Eval_metric` Métrique d'évaluation (erreur MSE, MAE, LogLoss, AUC,...).
- `Seed` Pour l'aléatoire.

Utilisation avec Spark et Scala


```scala
val spark = SparkSession.builder().appName("SimpleXGBoost Application").
      config("spark.executor.memory", "2G").config("spark.executor.cores", "4
      ").config("spark.default.parallelism", "4").master("local[*]").
      getOrCreate()

// number of iterations
val numRound = 10
val numWorkers = 4
// training parameters
val paramMap = List(
      "eta" -> 0.023f,
      "max_depth" -> 10,
      "min_child_weight" -> 3.0,
      "subsample" -> 1.0,
      "colsample_bytree" -> 0.82,
      "colsample_bylevel" -> 0.9,
      "base_score" -> 0.005,
      "eval_metric" -> "auc",
      "seed" -> 49,
      "silent" -> 1,
      "objective" -> "binary:logistic").toMap
println("Starting Xgboost ")
val xgBoostModelWithDF = XGBoost.trainWithDataFrame(trainingData, paramMap,
      round = numRound, nWorkers = numWorkers, useExternalMemory = true)

val predictions = xgBoostModelWithDF.setExternalMemory(true).transform(
      testData).select("label", "probabilities")
```

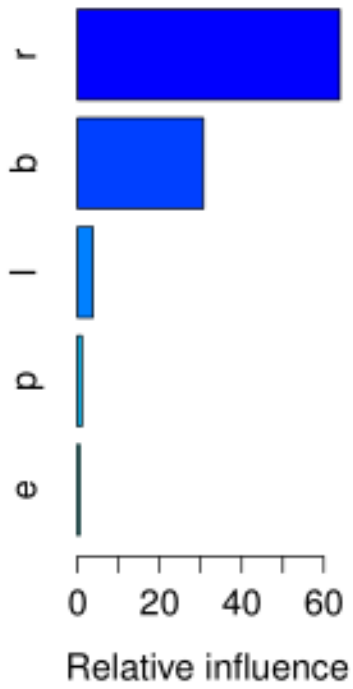
R ET FONCTION DE PERTE PERSONNALISÉE


```r
loglossobj <- function(preds, dtrain) {
  # dtrain is the internal format of the training data
  # We extract the labels from the training data
  labels <- getinfo(dtrain, "label")
  # We compute the 1st and 2nd gradient, as grad and hess
  preds <- 1/(1 + exp(-preds))
  grad <- preds - labels
  hess <- preds * (1 - preds)
  # Return the result as a list
  return(list(grad = grad, hess = hess))
}

model <- xgboost(data = train$data, label = train$label,
                 nrounds = 2, objective = loglossobj, eval_metric = "error")
```

Sélection de variables


```
bst <- xgboost(data = train$data, label = train$label, max.depth = 2,
           eta = 1, nthread = 2, nround = 2,objective = "binary:logistic")
importance_matrix <- xgb.importance(agaricus.train$data@Dimnames[[2]], model
      = bst)
xgb.plot.importance(importance_matrix)
```



Source : `http://dmlc.ml/rstats/2016/03/10/xgboost.html` et cours d'apprentissage statistique (C. Helbert)

Quelques bonnes pratique

1. Fixer un niveau d'apprentissage élevé.
2. Trouver le nombre optimal d'arbres.
3. Gérer les paramètres des arbres.
4. Gérer les paramètres de régularisation.
5. Réduire le niveau d'apprentissage.
6. Utiliser l'AUC pour estimer les modèles.

Challenges Kaggle

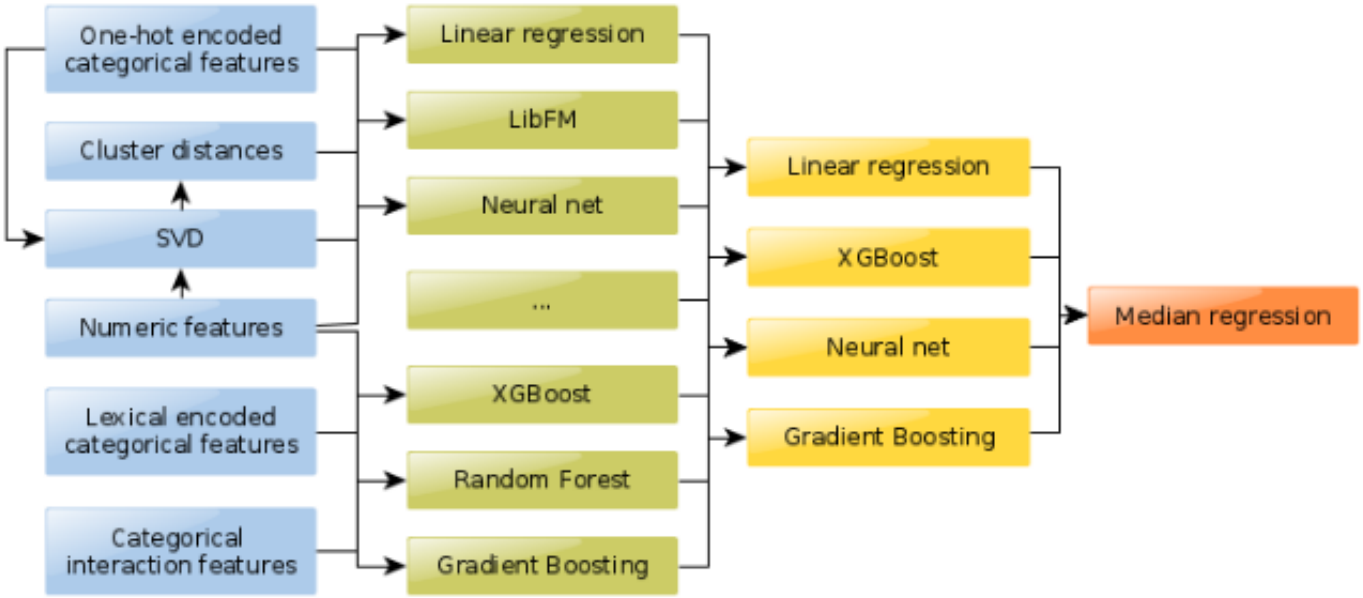
En 2015 sur Kaggle, 17 solutions gagnantes sur 29 utilisaient XGBoost.

Challenges Kaggle

En 2015 sur Kaggle, 17 solutions gagnantes sur 29 utilisaient XGBoost.

- **1er** Knowledge Discovery and Data Mining Cup 2016 (V. Sandulescu).
- **1er et 3ème** CERN LHCb experiment Flavour of Physics competition 2015 (V. Mironov).
- **1er** Caterpillar Tube Pricing competition (M. Filho).
- **2ème** Airbnb New User Bookings (K. Kuroyanagi).
- **2ème** Allstate Claims Severity (A. Noskov).
- **10%** Higgs Boson Competition (T. Chen).
- **...**

Un exemple concret



Source : *http://blog.kaggle.com/2017/02/27/allstate-claims-severity-competition-2nd-place-winners-interview-alexey-noskov/*

En entreprise

Des données difficiles à obtenir...

- ODPS Cloud Service (Alibaba)
- Tencent (QQ)
- AutoHome
- AXA, Expedia, Amazon,...

Conclusion

- Une implémentation récente (3 ans).
- Une forte portabilité et de bonnes performances.
- Une utilisation industrielle qui se développe.
- Savoir-faire nécessaire pour la configuration.

Conclusion

- Une implémentation récente (3 ans).
- Une forte portabilité et de bonnes performances.
- Une utilisation industrielle qui se développe.
- Savoir-faire nécessaire pour la configuration.

Une solution qui semble avoir de l'avenir !

Merci pour votre attention

Et place aux questions !

@DDouteaux

https://ddouteaux.github.io/XGBoost_Veille_Douteaux/index.html