Cours de Data Science

Marc Tommasi

17 novembre 2020

Outline

Outline

Principes

- vote (pondéré) d'un ensemble de classeurs : le vote majoritaire emporte la décision
- résultats théoriques donnés par le boosting
- Intuition : Si des classeurs de base sont suffisamment indépendants se trompent moins qu'un classeur aléatoire, alors leur vote forme une méthode plus précise que les classeurs de base.
- De nombreuses méthodes permettent de combiner ces classeurs : Bagging, Pasting, Boosting, Stacking, ...

Bagging

- On peut combiner des classeurs de nature différente ou les mêmes classeurs obtenus sur des données différentes (mais issues de la même distribution)
- L'objectif est de réduire la variance et limiter l'overfitting
- bagging : tirage uniforme avec remise d'un sous ensemble des données (boostrap aggregating) ;
- pasting : idem mais tirage sans remise.
- random subspace : similaire au boostrap mais tirage sur les attributs et non sur les exemples
- random patches : quand on tire à la fois selon les exemples et les attributs
- L'agrégation : par le mode (plus grande fréquence des prédictions), par un vote majoritaire, par la moyenne des fonctions de décision
- Le passage à l'échelle est facilité si on peut distribuer les calculs

Random Forests

- C'est fondamentalement du bagging avec des arbres de décision
- Introduire de la variété dans la construction d'un ensemble d'arbres de décision
- L'aléa est obtenu par
  - un tirage d'un sous-échantillon des exemples ou
  - dans la recherche du meilleur test : attributs mélangés aléatoirement, choisis dans un sous-ensemble des attributs
- On perd la lisibilité des arbres de décision
- Mais on peut calculer l'importance des attributs comme la moyenne des gains sur tous les arbres de la forêt et visualiser cette importance.

Boosting

- modifier la distribution des exemples en utilisant un poids sur chaque exemple qui varie en fonction des du taux de bonne classification
- le poids des exemples mal classés augmente graduellement
- la décision est un vote pondéré. Le poids est fonction de la précision du classeur

Principes du boosting

- Rappel : l'erreur en généralisation pour ERM = erreur d'approximation + erreur d'estimation
- On dispose d'une classe de fonctions pauvres avec une grande erreur d'approximation : des apprenants faibles dont la performance est juste un peu supérieure à l'aléatoire.
- Comment la rendre plus riche de façon itérative ?
- Booster on agrège des apprenants faibles pour former des apprenants plus forts, dont la performance peut être aussi grande qu'on veut.
- Objectifs :
  - une méthode qui casse la complexité d'apprendre un apprenant fort directement
  - Répondre à la question théorique de transformer un apprenant faible en un apprenant fort.
- Adaboost est une implantation qui résout ce problème théorique

Apprenant faible

Definition (Apprenant faible)

A est γ -faible pour une classe $\mathcal{H}$ si il existe $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ tel que pour tout $\delta \in (0,1)$, pour toute distribution $\mathcal{D}$ sur $\mathcal{X}$ et pour toute fonction $f : \mathcal{X} \to \{-1,1\}$, si l'hypothèse de réalisation est vraie, alors la sortie de A avec $m \geq m_{\mathcal{H}}(\delta)$ exemples tirés iid selon $\mathcal{D}$ en entrée étiquetés par f, est une fonction h telle que avec probabilité supérieure à $1-\delta$, $L_{\mathcal{D},f}(h) \leq 1/2 - \gamma$.

- on remplace ϵ (apprentissage fort) par $1/2 - \gamma$.
- la définition n'a d'intérêt que pour la complexité algorithmique : (pas vraiment d'incidence sur la PAC-apprenabilité)

Adaboost

- on a un apprenant faible
- procédure itérative qui modifie la distribution $\boldsymbol{D}^{(t)}$ sur S à chaque étape t
- l'erreur de l'apprenant faible quand il prédit l'hypothèse h_t est $\epsilon_t = \sum_i^m D_i^{(t)} \mathbb{1}_{[h_t(\boldsymbol{x}_i) \neq y_i]}$ avec proba $1/2 - \gamma$.
- Le poids associé à h_t est inversement proportionnel à l'erreur $w_t = \frac{1}{2} \log(\frac{1}{\epsilon} - 1)$: les exemples mal classés obtiennent un poids plus élevé.
- la sortie de Adaboost est un vote pondéré de toutes les hypothèses

Algorithme

$\mathcal{D}^{(1)} = (1/m, \ldots, 1/m)$

for $t = 1, \ldots, T$

$h_t = WL(\mathcal{D}^{(t)}, S)$

$\epsilon_t = \sum_i^m D_i^{(t)} \mathbb{1}_{[h_t(\boldsymbol{x}_i) \neq y_i]}$

$w_t = \frac{1}{2} \log(\frac{1}{\epsilon_t} - 1)$

$D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\boldsymbol{x}_i))}{\sum_j^m D_j^{(t)} \exp(-w_t y_j h_t(\boldsymbol{x}_j))}$

return $h_s(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^T w_t h_t(\boldsymbol{x})\right)$

- La classe des hypothèses d'Adaboost avec un apprenant faible dans B est $\{\text{sign}\left(\sum_{t=1}^T w_t h_t(x)\right) \mid \boldsymbol{w} \in \mathbb{R}^T, h_t \in B\}$
- Théorème : L'erreur d'entraînement de Adaboost est bornée par $exp(-2\gamma^2 T)$

Gradient boosting

- autre façon de se concentrer sur les erreurs : le classeur suivant tente de prédire correctement les erreurs résiduelles.
- Exemple avec `DecisionTreeRegressor`.
  - Une instance `dtr` de ce classeur produit un résiduel `y2 =y - dtr.predict(X)`.
  - On construit un `dtr2` pour prédire `y2`
  - On combine par la somme des prédictions des 2 classeurs
- La classe `GradientBoostingRegressor` fait cela.

Stacking

- Réduction du biais par des compositions de classeurs,
- on apprend à combiner les prédictions de classeurs de base
- l'apprentissage est découpé en deux parties :
  - partie A entraîne n classeurs de base
  - partie B est utilisée pour faire un ensemble P de vecteurs de prédictions de dimension n
- L'ensemble P est utilisé pour entraîner une combinaison (*blender*)

En sklearn

- une classe `VotingClassifier` permet de faire un simple vote pondéré ou pas de classeurs
- la classe `AdaBoostClassifier`
- la classe `BaggingClassifier`
  - Réglage bagging/pasting par le paramètre `boostrap`
  - Réglage des `max_features` et `max_samples`
- la classe `RandomForestClassifier`
- la classe `StackingClassifier`
- Dans ces implantations l'agrégation c'est la moyenne des fonctions de décision.
- Les mêmes classes existent pour la régression.