Analyse de Survie
Professeur Abdellatif El Afia

Arbres de Survie

Processus

1) Choisir une caractéristique telle que lorsque le nœud parent est divisé, il en résulte un gain d'informations maximal.
2) Arrêtez si les nœuds enfants sont purs ou si aucune amélioration de la pureté de la classe ne peut être apportée.
3) Reprendre l'étape 1 pour chacun des deux nœuds enfants.

Algorithme Arbre de classification

- Si $y = 1 \forall < x; y > \in D$ ou $y = 0 \forall < x; y > \in D$:
- Return Tree
- Sinon :
- Choisir la meilleure caractéristique x_j :

❖ D_0 à $arbre\_fils_0$: $x_j = 0 \forall < x; y > \in D$

❖ D_1 à $arbre\_fils_1$: $x_j = 1 \forall < x; y > \in D$

Return Nœud(x_j, GénérerArbre(D_0), GénérerArbre(D_1))

Variantes

La plupart des algorithmes d'arbre de décision diffèrent des manières suivantes :

- Critères de fractionnement : information gain (Entropie de Shanon, Gini impurity, misclassification..)
- Division binaire vs divisions multivoies
- Variables discrètes ou continues
- Pré vs post élagage

1- ID3 – Iterative Dichotomizer 3

- caractéristiques discrètes, ne peut pas gérer les caractéristiques numériques
- Divisions multi-catégories
- Pas d'élagage, sujet au surajustement (overfitting)
- Arbres courts et larges (par rapport à CART)
- Maximise le gain d'information/minimise l'entropie
- Caractéristiques discrètes, caractéristiques binaires et multi-catégories

Algorithme

```
fonction ID3(exemples, attributCible, attributsNonCibles)
  si exemples est vide alors /* Nœud terminal */
     retourner un nœud Erreur
  sinon si attributsNonCibles est vide alors /* Nœud terminal */
     retourner un nœud ayant la valeur la plus représentée pour attributCible
  sinon si tous les exemples ont la même valeur pour attributCible alors /* Nœud terminal */
     retourner un nœud ayant cette valeur
  sinon /* Nœud intermédiaire */
     attributSélectionné = attribut maximisant le gain d'information parmi attributsNonCibles
     attributsNonCiblesRestants = suppressionListe(attributsNonCibles, attributSélectionné)
     nouveauNœud = nœud étiqueté avec attributSélectionné

     pour chaque valeur de attributSélectionné faire
        exemplesFiltrés = filtreExemplesAyantValeurPourAttribut(exemples, attributSélectionné, valeur)
        nouveauNœud->fils(valeur) = ID3(exemplesFiltrés, attributCible, attributsNonCiblesRestants)
     finpour

     retourner nouveauNœud
```

2- C4.5

- Fonctionnalités continues et discrètes (le fractionnement continu des fonctionnalités est très coûteux car il doit prendre en compte tous les seuils possibles)
- Le critère de fractionnement est calculé via le rapport de gain
- Gère les attributs manquants (les ignore dans le calcul du gain d'informations)
- Effectue la post-élagage (élagage ascendante)

C4.5 quelques cas de base :

- Tous les échantillons de la liste appartiennent à la même classe. Lorsque cela se produit, il crée simplement un nœud feuille pour l'arbre de décision en disant de choisir cette classe.
- Aucune des fonctionnalités ne fournit de gain d'informations. Dans ce cas, C4.5 crée un nœud de décision plus haut dans l'arbre en utilisant la valeur attendue de la classe.
- Instance de classe inédite rencontrée. Encore une fois, C4.5 crée un nœud de décision plus haut dans l'arbre en utilisant la valeur attendue.

Pseudocode

1) Vérifier les cas de base précédents.
2) Pour chaque attribut a , trouver le rapport de gain d'informations normalisé provenant de la division sur a .
3) Soit a_{best} l'attribut avec le gain d'informations normalisé le plus élevé.
4) Créer un nœud de décision qui se divise sur a_{best}.
5) Récurer sur les sous-listes obtenues en divisant sur a_{best}, et ajouter ces nœuds en tant qu'enfants de nœuds.

C4.5 amélioration de ID3

- Gestion des attributs continus et discrets - Afin de gérer les attributs continus, C4.5 crée un seuil, puis divise la liste en ceux dont la valeur d'attribut est supérieure au seuil et ceux qui lui sont inférieurs ou égaux.
- Gestion des données d'entraînement avec des valeurs d'attribut manquantes - C4.5 permet aux valeurs d'attribut d'être marquées comme ? pour disparu. Les valeurs d'attribut manquantes ne sont tout simplement pas utilisées dans les calculs de gain et d'entropie. Ceci pourrait servir notre sujet sur survival)
- Gestion des attributs avec des coûts différents.
- Élagage des arbres après la création - C4.5 parcourt l'arbre une fois qu'il a été créé et tente de supprimer les branches qui n'aident pas en les remplaçant par des nœuds feuilles.

3- CART

• Caractéristiques continues et discrètes
• Que des questions-tests binaires (les arbres résultants sont plus grands par rapport à ID3 et C4.5)
• Les fractionnements binaires peuvent générer de meilleurs arbres que C4.5, mais ont tendance à être plus grands et plus difficiles à interpréter. C'est-à-dire que pour k attributs, nous avons $(2^{k-1}-1)$ façons de créer un partitionnement binaire
• Réduction de la variance dans les arbres de régression
• Utilise l'impureté de Gini dans les arbres de classification
• Effectue un élagage de la complexité des coûts

CART

- Pour chaque caractéristique, seuil a_j.
- Sous-nœud gauche S_g: $v_j < a_j$. $(p_g \approx \frac{|S_g|}{|S|})$.
- Sous-nœud droit S_d : $v_j \geq a_j$. $(p_d \approx \frac{|S_d|}{|S|})$.
- $E[I(S_{gd})] = p_g I(S_g) + p_d I(S_d)$
- $\Delta I(S) = I(S) - E[I(S_{gd})] = I(S) - p_g I(S_g) - p_d I(S_d)$.
- Le problème d'optimisation est le suivant :

$$\underset{j,a_j}{argmax} \ (\Delta I(S))$$

l'index de Gini $I_G(S)$

- Partitionner S sur les valeurs de la cible en m groupes : C_1,...,C_m,
- Calculer p_i : probabilité estimée qu'un élément de S se retrouve dans C_i ($p_i \approx \frac{|C_i|}{|S|}$),
- $I_G(S) = \sum_{i=1}^{m} p_i(1-p_i) = \sum_{i=1}^{m}(p_i - {p_i}^2) = 1 - \sum_{i=1}^{m} {p_i}^2$
- $I_G(S) = \sum_{i \neq j} p_i p_j$ index de Gini,
- $I_G(S) = 0$ si S est homogène (tous les éléments sont dans la même classe, donc impureté du groupe nulle).

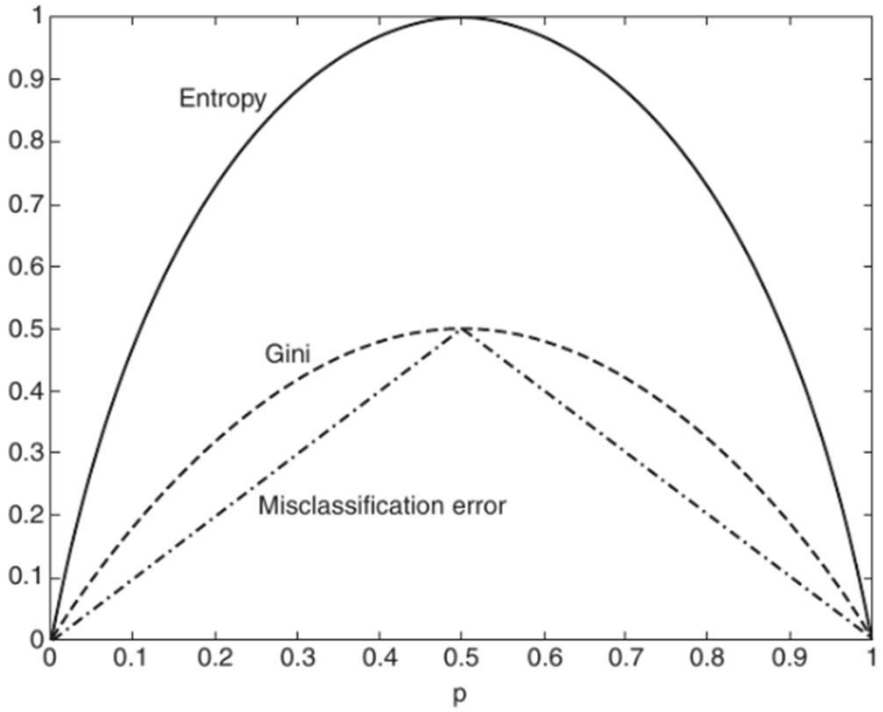
Autres mesures d'impureté

Toujours en classification on peut utiliser d'autres types de mesures d'impureté :

- $H(p) = -\sum_i p_i log_2(p_i)$ (entropie de Shannon)
- $I_\alpha = \frac{1}{1-\alpha}\log(\sum_{i=1}^{n} p_i^\alpha)$ (entropie de Renyi)
- $S_\alpha = \frac{1}{\alpha-1}(1 - \sum_{i=1}^{n} p_i^\alpha)$ (entropie de Tsallis)
- $E(S) = 1 - \max_i (p_i)$ (erreur de classification).
- Gain d'information : $G(\omega, A/D) = H(\omega/D) - \frac{|D_g|}{|D|}H(\omega, D_g) - \frac{|D_d|}{|D|}H(\omega, D_d)$

où H est l'entropie utilisée

Comparaison des mesures d'impureté des nœuds



Arbres de décision optimaux

Les arbres optimaux entraînent de manière flexible et efficace les arbres de décision selon une fonction de perte de la forme :

$$min_T(erreur(T, D) + \alpha.complexité(T))$$

Avec :

- T l'arbre à optimiser,
- D les données d'entrainement
- erreur(T,D) est une fonction mesurant à quel point T s'ajuste aux données D,
- complexité(T) une fonction pénalisant la complexité de l'arbre T.
- α est le paramètre de complexité qui contrôle le compromis entre la qualité de l'ajustement et la taille de l'arbre.

Aperçu

les nœuds de l'arbre sont visités dans un ordre aléatoire et à chaque nœud, nous considérons les modifications suivantes :

- Si le nœud n'est pas une feuille, supprimer la division de ce nœud.
- Si le nœud n'est pas une feuille, trouver la division optimale à utiliser à ce nœud et mettre à jour la division actuelle.
- Si le nœud est une feuille, créez une nouvelle division de ce nœud

Aperçu

- Pour chaque changement, nous calculons la valeur objective de l'arbre modifié par rapport à la minimisation de la fonction de perte.
- Si l'un de ces changements entraîne une amélioration de la valeur de l'objectif, la modification est alors acceptée.
- Lorsqu'une modification est acceptée ou que toutes les modifications potentielles ont été rejetées, l'algorithme procède à la visite d'autres nœuds de l'arbre dans un ordre aléatoire jusqu'à ce qu'aucune autre amélioration ne soit trouvée.
- ceci signifie que cet arbre est localement optimal par rapport à la fonction de perte.
- Ce problème n'est pas convexe, nous répétons donc le processus de descente de coordonnées à partir de divers arbres de décision de départ générés aléatoirement, avant de sélectionner l'arbre final localement optimal avec la valeur d'objectif globale la plus basse comme meilleure solution

Arbres de Survie

- Les arbres et les forêts de survie sont des alternatives non paramétriques populaires aux modèles (semi) paramétriques.
- Ils peuvent détecter automatiquement certains types d'interactions sans avoir besoin de les préciser au préalable.
- Un même arbre peut naturellement regrouper des sujets selon leur comportement de survie en fonction de leurs covariables.
- Ils peuvent se transformer en outils prédictifs très puissants, comme les forêts de survie.

Critères de fractionnement

- L'idée derrière le critère de fractionnement proposé par Gordon et Olshen (1985) était de forcer chaque nœud à être plus homogène en mesurant une métrique de Wasserstein entre la fonction de survie obtenue à partir de l'estimateur de Kaplan-Meier au nœud et une fonction de survie qui a une masse sur au plus un point fini.

- Utiliser la statistique du log-rank ou un rapport de vraisemblance paramétrique statistique pour mesurer la "distance" entre les deux nœuds enfants

- Le fractionnement retenu est celui avec la plus grande valeur statistique de Log-rank test significative.
- L'utilisation du test du log-rank conduit à une séparation qui assure la meilleure séparation des temps de survie médians dans les deux nœuds enfants

OST (Optimal Survival Tree)

Les règles de fractionnement basées sur la dissimilitude ne conviennent pas, car elles ne permettent pas l'évaluation d'un seul nœud isolément.

Nous allons donc nous concentrer sur les règles de séparation de la pureté des nœuds pour développer l'algorithme OST (Optimal Survival Tree).

L'efficacité de calcul est un facteur important dans le choix de la fonction cible, puisqu'elle doit être réévaluée pour chaque changement potentiel de l'arbre pendant les procédures d'optimisation.

La règle de partage implémentée dans l'algorithme OST est basée sur la méthode de vraisemblance.

OST Algorithme

La règle de fractionnement est dérivée d'un modèle à risques proportionnels qui suppose que la distribution de survie sous-jacente pour chaque observation est donnée par

$$P(S_i \leq t) = 1 - e^{-\theta_i \Lambda(t)},$$

Avec $\Lambda(t)$, la fonction de risque cumulé de référence et les coefficients θ_i sont les ajustements du risque cumulé de référence pour chaque observation.

OST Algorithme

• $\Lambda(t)$ est remplacée par une estimation empirique de la probabilité cumulée de décès à chacun des moments d'observation en utilisant l'estimateur de Nelson-Aalen.

$$\widehat{\Lambda}(t) = \sum_{i:t_i \leq t} \frac{\delta_i}{\sum_{j:t_j \geq t} 1}$$

• L'objectif du modèle d'arbre de survie est d'optimiser les coefficients de risque θ_i. Nous imposons que le modèle d'arbre utilise le même coefficient pour toutes les observations contenues dans un nœud feuille donné de l'arbre, c'est-à-dire $\theta_i = \hat{\theta}_{T(i)}$.

OST Algorithme

- Ces coefficients θ_i sont déterminés en maximisant la vraisemblance de l'échantillon intra-feuille

$$L = \prod_{i=1}^{n} (\theta_i \frac{d}{dt} \Lambda(t_i))^{\delta_i} e^{-\theta_i \Lambda(t_i)},$$

- Pour obtenir les coefficients du nœud,

$$\hat{\theta}_i = \frac{\sum_i \delta_i I_{\{T_i = k\}}}{\sum_i \hat{\Lambda}(t_i) I_{\{T_i = k\}}},$$

OST Algorithme

• Pour évaluer dans quelle mesure différents découpages s'adaptent aux données disponibles, nous comparons le modèle d'arbre actuel à un arbre avec un seul coefficient pour chaque observation. Nous appellerons cela un arbre entièrement saturé, car il a un paramètre unique pour chaque observation. Les estimations du maximum de vraisemblance pour ces coefficients de modèle saturés sont :

$$\hat{\theta}_i^{sat} = \frac{\delta_i}{\hat{\Lambda}(t_i)}, \qquad i = 1, \dots, n.$$

OST Algorithme

• Nous calculons l'erreur de prédiction à chaque nœud comme la déférence entre la log-vraisemblance pour le coefficient du nœud ajusté et les coefficients du modèle saturé à ce nœud :

$$erreur_k = \sum_{i:T(i)=k} (\delta_i \log(\frac{\delta_i}{\widehat{\Lambda}(t_i)}) - \delta_i \log(\hat{\theta}_k) - \delta_i + \widehat{\Lambda}(t_i)\hat{\theta}_k).$$

• La fonction d'erreur globale utilisée pour optimiser l'arbre est simplement la somme des erreurs sur les nœuds feuilles de l'arbre T compte tenu des données d'apprentissage D :

$$erreur(T, D) = \sum_{k \in feuilles(T)} erreur_k(D).$$

TP 5 :

1- Construire un arbre de survie optimal OST en utilisant la fonction d'érreure définie ci haut.

2- Utiliser votre OST sur votre data, et comparer les résultats avec les autres modèles.

Références

- 1. Adeline Morisot. Méthodes d'analyse de survie, valeurs manquantes et fractions attribuables temps dépendantes : application aux décès par cancer de la prostate. Médecine humaine et pathologie. Université Montpellier, 2015. Français. ffNNT : 2015MONTT010ff. fftel-01408070ff

- 2. David G. Kleinbaum Mitchel Klein **Survival Analysis A Self-Learning Text** Third Edition