Régression Linéaire

Principe : Modèle paramétrique établissant une relation linéaire entre les variables indépendantes (X) et la cible (y). Formule : y=w1x1+w2x2+…+wnxn+b.

Fonction de Coût : La fonction de coût pour la régression linéaire est l'erreur quadratique moyenne (MSE) : J(w,b)=1/2n*∑i=1n(yi−y^i)exp(2), , où y^i=w1x1+w2x2+…+b.

Exemple : Problème : Prédire les notes d'un étudiant (y) en fonction du nombre d'heures d'étude (x1) et de la note moyenne de l'élève (x2).

Modèle : y=0.5x1+0.7x2+2. Pour un élève ayant étudié 5 heures (x1=5) avec une note moyenne de 15 (x2=15) : y=0.5·5+0.7·15+2=15.

Scikit-learn :

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Régression Logistique

Principe : Modèle pour la classification binaire. Transforme une relation linéaire en une probabilité à l'aide de la fonction sigmoïde. Formule : P(y=1∣x)=σ(w1x1+w2x2+…+b), avec σ(z)=1/(1+eexp(−z)).

Fonction de Coût : La régression logistique utilise la fonction de perte logistique (log-loss) : J(w,b)=−1/m*∑i=1m[yilog(y^i)+(1−yi)log(1−y^i)], , où y^i=σ(w1x1+…+b).

Exemple : Problème : Classer si une personne est à risque de maladie cardiaque (y=1) ou non (y=0) selon son âge (x1) et son IMC (x2). Modèle : P(y=1∣x)=1/(1+eexp−(0.3x1+0.7x2−5)). Pour une personne de 50 ans (x1=50) avec un IMC de 30 (x2=30) : P(y=1∣x)=1/(1+eexp−(0.3·50+0.7·30−5))≈0.99. Cette personne est fortement à risque.

Scikit-learn :

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Régression Polynomiale

Principe : Étend la régression linéaire en ajoutant des termes polynomiaux (x2,x3,) pour modéliser des relations non linéaires. Formule : y=β0+β1X+β2Xexp(2)+…+βdXexp(d)+ϵ.

Fonction de Coût : Même que pour la régression linéaire (MSE), mais avec des termes

polynomiaux supplémentaires : J(w,b)=1/2m*∑i=1m(yi−y^i)exp(2), , où y^i=β0+β1X+β2Xexp(2)+…

Exemple : Problème : Modéliser la trajectoire d'un ballon (y) en fonction du temps (x) : Modèle : y=−4.9xexp(2)+20x+50 (courbe parabolique). Pour x=3 secondes : y=−4.9·9+20·3+50=80.1 m.

Scikit-learn :

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)
model = LinearRegression()
model.fit(X_poly, y_train)
predictions = model.predict(poly.transform(X_test))
```

Méthode Normale (Résolution Analytique)

Lorsque vous avez un petit ensemble de données, une méthode analytique (aussi appelée méthode normale) permet de calculer directement les coefficients ββ sans recourir à des approches itératives comme la descente de gradient. La méthode normale est une solution exacte pour le problème de régression linéaire, et elle s'écrit de la manière suivante :
β=(XTX)−1XTy

XT : La transposée de la matrice des caractéristiques X.
XTX : Le produit de la transposée de X avec X lui-même. C'est une matrice carrée de dimension p×p, où p est le nombre de caractéristiques.
(XTX)−1 : L'inverse de la matrice XTX, qui n'existe que si XTX est inversible (c'est-à-dire si X a des colonnes linéairement indépendantes).
XTy : Le produit de la transposée de X avec le vecteur des cibles y.
y : Le vecteur des valeurs cibles (dimension n×1).

Interprétation de β

Les coefficients β peuvent être interprétés comme suit :

Chaque coefficient βj (avec j=1,2,…,p) représente l'effet ou l'influenc de la j-ème caractéristique sur la variable cible y, en supposant que les autres caractéristiques sont maintenues constantes.

β0 (le premier coefficient ou l'ordonnée à l'origine) représente le terme constant ou l'intercept, c'est-à-dire la valeur de y lorsque toutes les caractéristiques sont égales à zéro.

Les signes des coefficients (positifs ou négatifs) indiquent si la relation entre la caractéristique et la cible est positive (augmentation de y avec xj) ou négative (diminution de y avec xj).

β représente les coefficients (ou poids) dans un modèle de régression linéaire. Chaque coefficient βj quantifie l'impact de la caractéristique correspondante sur la variable cible.
La méthode normale permet de résoudre directement les coefficients β en utilisant la formule β=(XTX)−1XTy lorsque l'ensemble de données est petit.

Ridge et Lasso (variante de régression linéaire)

Ridge (Régularisation L2)

Principe : Modifie la fonction de coût en ajoutant une pénalité basée sur l somme des carrés des coefficients : Lridge=1/
n*∑i=1n(yi−y^i)exp2+λ∑j=1pθjexp2. *Réduit la magnitude des coefficients sans les rendre nuls*.

Exemple : Si deux caractéristiques (taille et poids) sont fortement corrélées pour prédire la consommation calorique, Ridge répartit l'importance entre les deux.

Applications : Régression avec des variables corrélées ou nombreuses. Réduction des risques d'overfitting.

Lasso (Régularisation L1)

Principe : Ajoute une pénalité basée sur la somme des valeurs absolues des coefficients : Llasso=1/n*∑i=1n(yi−y^i)exp2+λ∑j=1p|βj|. *Peut rendre certains coefficients exactement nuls, permettant une sélection automatique des caractéristiques.*

Exemple : Dans un dataset avec 100 caractéristiques, Lasso peut identifier automatiquement les 10 plus pertinentes en annulant les coefficients des 90 autres.

Applications : Modèles où seule une sous-ensemble des caractéristiques est pertinent. Besoin de simplifier un modèle pour l'interprétabilité.

Arbre de Décision

Principe : Divise les données en sous-groupes en appliquant des règles conditionnelles successives. Exemple de règle : « Si âge ≤30, alors prédire classe 1 ; sinon, prédire classe 0. »

Fonction de Coût : En classification : Entropie ou indice de Gini pour mesurer l'homogénéité. En régression : Réduction de la variance.

Exemple : Problème : Classer si un client achètera (y=1) ou non (y=0) selon : Âge (x1) et Revenu (x2). Règles conditionnelles : Si x1≤30, prédire 1. Sinon, si x2≤50,000, prédire 0. Sinon, prédire 1.

Scikit-learn :

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

L'algorithme d'un arbre de décision construit ses règles conditionnelles en divisant récursivement les données en sous-ensembles basés sur les caractéristiques (features), afin de maximiser une certaine mesure de séparation. Voici une explication détaillée du processus : Étapes pour Trouver les Règles

```
Sélection de la caractéristique et du seuil de division :
    L'algorithme examine toutes les caractéristiques et tous les s
euils possibles pour diviser les données.
    Pour chaque caractéristique et seuil, il évalue une mesure de
qualité (par exemple, entropie, indice de Gini ou réduction de var
iance).
    Il choisit la caractéristique et le seuil qui maximisent la sé
paration des données.

Création des règles conditionnelles :
    Une fois la meilleure caractéristique et le meilleur seuil tro
uvés, l'algorithme crée une règle conditionnelle sous la forme :
        Si feature≤seuilfeature≤seuil, alors envoyer à gauche.
        Sinon, envoyer à droite.

Répétition du processus :
    L'algorithme applique ces étapes récursivement sur les sous-en
sembles créés, jusqu'à atteindre un critère d'arrêt (par exemple,
un nombre minimum d'échantillons par feuille, une profondeur maxim
ale, ou une pureté maximale des données dans un nœud).

Résultat final :
    Chaque feuille de l'arbre contient une prédiction (par exemple
, la classe majoritaire pour les problèmes de classification ou un
e moyenne pour les problèmes de régression).
```

Critères pour Prendre des Décisions

Les arbres de décision utilisent différents critères pour évaluer la qualité d'une division.

1. Classification : Entropie et Indice de Gini

Pour les problèmes de classification, l'objectif est de maximiser l'homogénéité des données dans chaque sous-ensemble.

```
Entropie :
    Mesure le degré de désordre ou d'incertitude.
    La formule de l'entropie pour une partition est :
    H(D)=−∑i=1kpi*log2(pi)
     où pi est la proportion des éléments de classe i dans le sou
s-ensemble.

Réduction d'entropie (Information Gain) :
    Après une division, l'entropie est réduite. La qualité d'une d
ivision est mesurée par la réduction d'entropie :
    Information Gain=H(Parent)−∑(|Dfils|/|Dparent|*H(Dfils))

Indice de Gini :
    Mesure la probabilité qu'un élément choisi au hasard soit mal
classé si on lui attribue une classe aléatoire.
    La formule est :
    Gini(D)=1−∑i=1kpi2

Choix entre Entropie et Gini :
    Les deux critères donnent souvent des résultats similaires, ma
is l'indice de Gini est plus rapide à calculer et est donc souvent
utilisé par défaut.
```

2. Régression : Réduction de la Variance

Pour les problèmes de régression, l'objectif est de minimiser la variance des valeurs dans chaque sous-ensemble.

```
Variance :
    Mesure la dispersion des données dans un nœud.
    Après une division, la qualité de la séparation est mesurée pa
r la réduction de variance :
    Reduction de Variance=VarianceParent−∑(|Dfils|/|Dparent|*Varianc
efils)
```

Exemple d'Arbre de Décision (Classification) Données :

Supposons que nous avons un dataset avec les caractéristiques Age et Salaire, et une classe cible Achete (Oui ou Non) : Age Salaire Achete 25 Faible Non 30 Moyen Non 35 Moyen Oui 40 Élevé Oui 45 Élevé Oui Processus :

```
Sélection de la première caractéristique :
    L'algorithme évalue les divisions possibles sur Age et Salaire
.
    Il calcule l'entropie ou le Gini pour chaque seuil et choisit
la meilleure division.

Création de la règle :
    Supposons que Age <= 30 est choisi comme meilleure règle.
    Le premier nœud devient :
        Si Age <= 30, alors Non.
        Sinon, continuer la division.

Répétition pour les autres nœuds :
    L'algorithme répète le processus pour les sous-groupes jusqu'à
atteindre un critère d'arrêt.
```

Résumé : Comment l'algorithme prend des décisions

```
Il évalue toutes les caractéristiques et seuils possibles pour div
iser les données.
Il utilise une mesure de qualité comme la réduction d'entropie, l'
indice de Gini, ou la réduction de variance pour choisir la meille
ure division.
Il crée une règle conditionnelle basée sur la caractéristique et l
e seuil sélectionnés.
Il répète ce processus récursivement pour construire l'arbre.
```

Ce processus permet de générer des règles conditionnelles qui segmentent les données de manière optimale pour une prédiction précise.

Dans un arbre de décision, toutes les caractéristiques disponibles sont évaluées individuellement à chaque étape de la construction de l'arbre, et non en sous-ensembles ou combinaisons de caractéristiques. Voici comment cela fonctionne dans un cas où il y a par exemple 15 caractéristiques : Processus : Une caractéristique à la fois

```
Évaluation des caractéristiques :
    L'algorithme examine chaque caractéristique séparément pour tr
ouver celle qui permet la meilleure division des données.
    Pour chaque caractéristique :
        Si c'est une caractéristique numérique (ex. : âge, reven
u), il teste plusieurs seuils possibles (ex. : âge≤30).
        Si c'est une caractéristique catégorielle (ex. : couleur,
ville), il teste différentes combinaisons de catégories.

Choix de la meilleure caractéristique :
    Une mesure de qualité (par exemple, réduction d'entropie ou in
dice de Gini) est calculée pour chaque caractéristique, et celle q
ui maximise la séparation est choisie.
    À ce stade, les autres caractéristiques sont temporairement ig
norées.

Division :
    Les données sont divisées selon la meilleure caractéristique e
t son seuil optimal.

Répétition pour les sous-ensembles :
    Ce processus est répété indépendamment dans chaque sous-ensemb
le résultant, en réévaluant toutes les caractéristiques restantes.
```

L'arbre de décision ne considère jamais directement des combinaisons de caractéristiques pour choisir un seuil. Voici pourquoi :

```
Simplification du processus :
    Évaluer les combinaisons de 15 caractéristiques serait extrême
ment coûteux en calcul.
    Par exemple, pour 15 caractéristiques avec 10 seuils possibles
chacune, il y aurait 10exp(15) combinaisons possibles !

Nature récursive de l'arbre :
    Les arbres de décision divisent les données étape par étape, e
n se concentrant sur une seule caractéristique à chaque division.
La structure de l'arbre capture indirectement des interactions ent
re les caractéristiques, car les décisions conditionnelles sont pr
ises séquentiellement.

Interactions indirectes :
    Bien que les caractéristiques soient évaluées séparément, les
interactions entre elles sont capturées implicitement par la struc
ture de l'arbre. Par exemple, une branche peut utiliser Age comme
première caractéristique, et ensuite Income pour affiner la décisi
on.
```

Exemple Concret

Supposons un dataset avec 3 caractéristiques : Age, Income, et Gender, et une cible Buy (Oui/Non). Étape 1 : Évaluation initiale

```
L'algorithme évalue les 3 caractéristiques séparément :
    Age : Divise les données avec un seuil comme Age≤30.
    Income : Divise les données avec un seuil comme Income≤50,000.
    Gender : Divise les données par catégories Gender=Male/Female.
```

Étape 2 : Choix de la meilleure division

```
Suppose que Age avec Age≤30 donne la meilleure réduction d'entropi
e.
Cette division est appliquée.
```

Étape 3 : Répétition dans les sous-groupes

```
Dans le groupe où Age≤30, toutes les caractéristiques restantes (I
ncome, Gender) sont réévaluées pour trouver la meilleure division.
```

Caractéristiques et sous-ensembles

```
À chaque niveau de l'arbre, toutes les caractéristiques disponible
s sont examinées.
Cependant, dans un sous-ensemble, certaines caractéristiques peuve
nt devenir moins pertinentes ou même redondantes, en fonction des
décisions prises dans les niveaux supérieurs.
```

Forêt Aléatoire

Principe : Ensemble de plusieurs arbres de décision, où chaque arbre est entraîné sur un sous-ensemble aléatoire des données et des caractéristiques. Les prédictions finales sont obtenues par vote majoritaire (classification) ou moyenne (régression).

Fonctionnement : Divise les données en plusieurs sous-ensembles aléatoires. Entraîne un arbre de décision sur chaque sous-ensemble. Agrège les prédictions des arbres pour obtenir une meilleure robustesse.

Exemple : Problème : Prédire si une plante appartient à une espèce donnée (y=1) ou non (y=0) selon la longueur (x1) et la largeur (x2) de ses pétales et sépales. Un arbre de décision peut surapprendre sur des données bruitées. Une forêt aléatoire crée plusieurs arbres et donne une décision robuste.

Avantages : Réduit le risque de surapprentissage par rapport à un arbre unique. Fonctionne bien sur des données bruitées ou complexes.

Scikit-learn :

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```