Avant-propos

Je suis diplômée d’un master 2 Données, Apprentissage et Connaissances (DAC) à Sorbonne Université Sciences. Lors de ma derniere année de master, j’ai pris l’initiative de collecter mes connaissances et de les partager. Cet ouvrage, est un résumé des cours que j’ai suivis à Sorbonne Université Science (ex. UPMC), des livres « le Machine Learning avec Python », « Machine Learning avec Scikit Learn », Deep Learning avec tensorflow » et de vidéos sur Youtube. Je l’ai rédigé pendant les vacances.

Knowledge is power

Le temps est d’or

Le savoir se partage

**Celia KHERFALLAH**

# **Chapitre 1 : Introduction au Machine Learning (ML)**

## Qu’est-ce que le ML ?

L’apprentissage automatique concerne l’extraction de connaissance à partir de données. C’est un champ de recherche qui se situe entre les statistiques, l’informatique et l’intelligence artificielle. On parle aussi d’analyse prédictive ou apprentissage statistique.

« *Etant donné une tache T est une mesure de performance P, on dit qu’un programme informatique apprend à partir de l’expérience E si les résultats obtenus sur T, mesurés par P, s’améliorent avec l’expérience E.* » TOM MITCHELL, 1997

Exemple : Email spam

|  |
| --- |
| Tâche T consiste à identifier parmi les nouveaux courriels ceux qui sont frauduleux. |
| L’expérience E est constituée par les données d’entrainement. |
| La mesure P doit être définie, fonction de coût / hypothèse. |

## **Pour quoi le ML?**

De nombreux systèmes utilisent des règles en dur (figées dans le code), basées sur des décisions de type Si/Sinon ~ Règles décisionnelles ciblées à la main par un expert.

* Système de règles expert

Points négatifs :

* La modification peut impliquer la réécriture de l’ensemble du système ;
* La présence d’un expert humain ;
* Nécessite énormément de règles.

La collecte de données :

* Processus différents à chaque fois ;
* Peu onéreuse ;
* Nécessite des matériaux couteux (image médicale) ;
* Expertise rare et chère ;
* Problème d’éthique et protection de la vie privée.

Remarque : Features => caractéristiques (variables)

## **Comprendre le but et les données**

1. Comprendre les données sur lesquelles on travaille.
2. Comprendre comment elles se relient au problème qu’on veut résoudre afin de bâtir le modèle (choix de l’algorithme).

Exemples de régression : Prédire le salaire annuel en fonction de l’éducation, de l’âge, du sexe.

Différence entre la classification et la régression

Une régression Prédire une valeur

La performance P MSE

Classification Prédire une classe

La performance P Accuracy

La différence réside dans la sortie.

## Sur ou sous APPRENTISSAGE ?

|  |  |
| --- | --- |
| Contexte | |
| Si le modèle est **trop simple** alors on **ne capture pas** tous les aspects de nos données ainsi que leurs variabilités | Construire un modèle qui soit capable d’effectuer une généralisation avec l’exactitude la plus élevée possible, revient à construire un modèle parfois **très complexe** pour une quantité de données.  Il suit trop précisément les particularités du jeu d’apprentissage.  Il est incapable de généraliser |
| Evaluation | |
| Train score <= 50 % et test score <= 50 % | training score – test score| < ɛ | 99 % |
| Solution | |
| Choisir un modèle plus puissant ou de diminuer le degré de régularisation | Choisir un modèle plus simple, contraindre le modèle « Régularisation » ou ajouter plus de données. |

## **COMPLEXITÉ ?**

La complexité d’un modèle est infiniment liée à la variance des éléments contenus dans votre jeu de données d’apprentissage. Plus les éléments sont variés, plus le modèle qu’on utilisera sera complexe pour éviter le sur-apprentissage.

# **CHAPITRE 2 : Modèles d’apprentissages automatiques supervisés**

### *KNN*

|  |  |
| --- | --- |
| Nombre de voisin | |
| 1 voisin | N voisins |
| Utiliser un seul voisin produit une frontière de décision (voir dessin) qui suit de près les données d’apprentissage.  La prédiction portant sur le jeu de données Train est parfaite => modèle complexe. | Lorsque que le nombre de voisin augmente, le modèle devient plus simple, la frontière tend à se lisser.  Utiliser plus de voisin correspond à une moindre complexité du modèle.  L’exactitude de la prédiction chute progressivement, signifiant que le modèle se simplifie. |
| Avantages et inconvénients | |
| Forces | Faiblesses |
| * Donne des performances raisonnables sans à avoir à effectuer un tas d’ajustement. * Facile à comprendre, la construction du modèle est rapide. * Il existe une variante de l’algorithme de knn pour la régression :   Knn\_reg = KNeighborsRegressor(n\_neighbors =N) | * Le calcul de la prédiction peut être très l**ent** si le jeu de données est très étendu (que ça soit le nombre d’échantillons ou le nombre de caractéristiques). * Marche mal avec des exemples **comportant plus d’une centaine de caractéristiques** et même particulièrement mauvaise dans le cas où la plupart **des caractéristiques sont nulles** (*sparse*). * Possède **deux paramètres** :   1. n\_neighbors.   2. Le choix de la mesure de distance entre les points de données. |
| Conclusion | |
| * A utiliser *avant* d’envisager des techniques plus avancées. * Il est important **d’effectuer un prétraitement sur les données.** * **Il n’est pas très utilisé en pratique, du fait de sa lenteur et de son incapacité à gérer de nombreuses caractéristiques.** | |

# Modèles linéaires/ régression Linéaire :

Largement utilisés en pratique, ils effectuent une prédiction en utilisant une fonction linéaire des caractéristiques présentées en entrée.

La prédiction est une somme pondérée des paramètres, plus un terme constant (biais ou intercept).

Le biais est un paramètre, il varie tout au long de l’entrainement, une fois l’apprentissage terminé il devient bel et bien constant.

Ils ont des scores de 95% en Test.

## Avantages et inconvénients :

-Ils gagent aussi en complexité ce qui augmente le risque de sur-apprentissage.

- Interpréter les coefficients si les caractéristiques sont corrélées.

+ Rapide.

+ Puissant si beaucoup d'échantillons et beaucoup de dimension contrairement à une unique caractéristique (perspective plus au moins faussée).

+ Facile à comprendre le processus de prédiction en partant des formules.

+ Il existe de nombreux modèles linéaires différents pour la régression. La différence est liée à la manière dont les paramètres w et b sont entrainés.

## Descente de gradient

|  |  |
| --- | --- |
| Trouver valeur de qui minimise la fonction de cout ? | |
| **Equation Normale** | **Descente de gradient** (Algorithme de généralisation) |
| Forme analytique (Formule mathématique)  **Inv(X.T.dot(X)).dot(X.T).dot(y)**  Est une matrice (n+1) . (n+1)  n le nombre de variable.  La complexité algorithmique est entre O () et O ().  Elle prend beaucoup de temps lorsque n >>>  Elle est linéaire par rapport au nombre d’observation du jeu d’entrainement ce qui lui permet de traiter efficacement des jeux de données >> (tenir en mémoire). | **L’idée :** corriger petit à petit les paramètres dans le but de minimiser une fonction de cout.  Si gradient = 0 Alors on atteint le minimum.  Meilleur quand n >>>  **Initialisation aléatoire.**    On fait décroitre la fonction de cout pas à pas jusqu’à convergence vers un minimum.  Taux d’apprentissage : Un hyperparamètre de l’algorithme d’apprentissage et non du modèle.  Ce n’est pas un paramètre qu’on cherche à optimiser pendant l’apprentissage.  Si <<< l’algorithme devra effectuer un grand nombre d’itération pour converger et prendra beaucoup de temps.  Si >>> risque de dépasser le point le plus bas et on se retrouver de l’autre côté ou plus haut qu’avant.  ==> diverger l’algorithme avec des valeurs de plus en plus grandes. |

|  |  |  |
| --- | --- | --- |
| Descente de gradient | | |
| DG Ordinaire (Batch) | DG Stochastique | DG par mini lot |
| Calcule le gradient de la fonction de cout par rapport à chaque paramètre du modèle (la modification de la fonction de cout lorsqu’on modifie un petit peu dérivé partielle))  = = (X.  A chaque époque le calcule se fait sur toute la base  -Lent sur des jeux de données volumineux.  Si données normalisées, taux d’apprentissage = 0.01 | Le calcul du gradient se base uniquement sur un exemple tiré aléatoirement  Moins régulier qu’une descente de gradient batch, la fonction cout effectue des sauts vers le haut et vers le bas et ne décroitra qu’en moyenne.  Les valeurs finales des paramètres sont bonnes mais pas optimales (=> solution réduire progressivement le taux d’apprentissage (grand au début pour échapper des minimas locaux puis petit à la fin pour rester dans le minima globale - **Learning Schedule**)  + Arrive à s’échapper des minimums locaux  + Converge vite  + Bon pour des gros jeux de données  + Certains exemples peuvent être vus plusieurs fois (🡺 mélange le jeu à chaque époque) | Calcule le gradient d’un sous ensemble de données sélectionnées aléatoirement.  Idem que la DS la solution est au tour de la solution optimale. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithme | M (exemples) >>> | N (variables) >>> | Normalisation requise ? | Scikit - Learn |
| Equation normale | Rapide | Lent | Non | linearRegression |
| DG Batch | Lent | Rapide | Oui | - |
| DG Stochastique | Rapide | Rapide | Oui | SGDRegressor |
| DG Mini Batch | Rapide | Rapide | Oui | SGDRegressor |

## Modèle linéaire pour la régression

Dans le cas de la régression, la formule de prédiction générale pour le modèle linéaire se présente comme suit :

­y­ = w [0]\*x [0] + w [1]\*x [1] + … + w [p]\*x [p] +b, p+1 caractéristiques, b Є R.

Produit scalaire

Sous forme vectorielle.

Transposé, car en ML un vecteur colonne est : [[1], [2], [3]] [1, 2, 3]

T

X = x1, x2 … (x0 = 1)

Si size[w] = 1 caractéristique, alors la prédiction est une ligne droite.

Si size[w] = 2 caractéristiques, alors la prédiction est un plan.

Si size[w] = n caractéristiques, alors la prédiction est un hyperplan.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *MSE*(LinearRegression) | | *LASSO* (Least Absolute Shrinkage and Selection Operator, *Alternative du Ridge*) | *RIDGE* | | |
| La plus simple et la plus classique.  Elle trouve W et b, qui minimisent la valeur de l’erreur quadratique moyenne. | | Même formule que MSE + une contrainte. | Même formule que MSE + une contrainte supplémentaire pour le choix de W.  Il faut qu’il soit le plus petit possible ~ 0. Cela signifie que chaque caractéristique devrait avoir aussi peu d’impact que possible sur la sortie *(donc présenter une pente faible)* tout en fournissant une bonne prédiction. | | |
| **Régulation : Une quantité est ajouté à la fonction cout que durant l’entrainement, la fonction de performance est non régularisée.**   * Quantité de régularisation est réglable à l’aide du paramètre alpha. * Alpha >>> revient à force davantage les coefficients à se rapprocher de 0 => ce qui fait décroitre les performances sur le jeu d’apprentissage mais peut aider à obtenir une meilleure généralisation. * Alpha <<< le modèle ressemble de plus en plus à une régression linéaire (les deux finissent par se confondre). | | | | | |
| Aucune régulation | | Régulation L1 (Manhattan) | | Régulation L2 (Euclidienne) | |
|  | | | | | |
| Aucun paramètre alpha | Paramètres : Alpha 1.0 par défaut   * Si alpha = 0 alors Ridge/Lasso = LinearRegression * Si Alpha >> alors :   + On autorise plus de caractéristique à être non nul (conseil : augmenter max\_iter : 100000).   + Permet d’avoir un modèle plus complexe. * Si Alpha << alors :   + Tous les coefficients de pondération finiront par avoir des valeurs très proches de 0 et le résultat sera une ligne horizontale passant par la moyenne des données. De plus, on perd l’effet de régularisation et on finit par retourner au problème de base *🡺 sur-apprentissage*, même résultat que celui de LR. | | | |
| Aucune contrainte. | | Contraindre les coefficients à se rapprocher de zéro (plus que Ridge), mais avec une régularisation L1, effectue la somme des valeurs absolues des coefficients au lieu de la moitié du carré de la norme L2 | | Contrainte de régularisation : restreindre explicitement un modèle pour éviter le sur-apprentissage.  Avec la norme L2, d’un point de vue mathématique, la méthode pénalise le carré de la norme L2, ou encore la longueur euclidienne de W. | |
| - Sur-apprentissage est probable.  + Bonne performance avec des dataset de grande taille.  +/- N’a pas d’autres paramètres ce qui est un avantage mais implique aussi qu’elle ne permet pas de contrôler la complexité du modèle. | | + Idem que Ridge.  + A choisir si on a beaucoup de caractéristique et qu’on pense que certains sont inutile.  + Facile à interpréter du fait qu’elle sélectionne un sous ensemble de caractéristique. | | + Modèle restrictif, donc le sur-apprentissage est moins probable.  - En pratique, elle est la plus utilisée. | |
| LR vs Ridge | | LASSO vs Ridge | | Ridge | |
| Pour des dataset **<<** 400 exemples :  La **régression linéaire** **n’apprend rien** mais le Ridge a de bon score dans le test du fait de la régularisation.  Au fur et à mesure que la taille de du dataset augmente (0-400), le Ridge s’améliore et le score de la RL sur le test décroit (1-.96).  A 400 les deux modèles ont les mêmes performances.  - La régularisation devient moins importante.  - Il devient plus difficile pour un modèle linéaire de sur-apprendre. | | Avec une régularisation L1, certains coefficients valent exactement 0, cela signifie que certaines caractéristiques sont totalement ignorées par le modèle, cela peut être vu comme une sélection automatique des caractéristiques, avoir des coefficients nuls peut le rendre plus facile à interpréter et met en évidence les caractéristiques les plus importantes. | | Ridge établit un compromis entre la simplicité du modèle (coefficient proche de 0) et les performances sur le jeu de données. | |
| Ex :  From sklearn.linear\_model import LinearRegression  % jeu de donnée simple, une caractéristique  X, y =gleran.datestes.make\_wave(n\_samples = 60)  X\_train, X\_test, y\_train y\_test =  train\_test\_split(X,y, randome\_state = 42)  Lr = LinearRegression.fit( X\_train, y\_train)  Print(« w : {} ».format(lr.coef\_)) %.67  Print(« b : {} ».format(lr.intercept\_)) %.66  % pas tres bon, mais pas de grande difference  % jeu de donnée complexe, 104 caractéristiques  X, y =gleran.datestes.load\_extended\_boston()  X\_train, X\_test, y\_train y\_test =  train\_test\_split(X,y, randome\_state = 0)  Lr = LinearRegression.fit( X\_train, y\_train)  Print(« w : {} ».format(lr.coef\_)) %.95  Print(« b : {} ».format(lr.intercept\_)) %.61  % clairement un signe de sur-apprentissage, donc mieux vaux trouver un autre modèle qui contrôle sa complexité | | From sklearn.linear\_model import Lasso  Lass = Lasso().fit(X\_tain, y\_train)  Print(« Training set score {} ».format(lasso.score(X\_tain, y\_train) ))  Print(« Test set score {} ».format(lasso.score(X\_test, y\_test) ))  % .29 vs .21  Print(“Number of features used {}”.format(np.sum(lasso.coef\_ != 0)))  % 4 parmi 104 | | EX :  From sklearn.linear\_model import Ridge  ridge = ridge().fit(X\_train,y\_train)  Print(« training set score : {} ».format(ride.score(X\_train,y\_train))  Print(« test set score : {} ».format(ride.score(X\_train,y\_train))  % .89 vs .75 | |

ELASTIC NET

Mélanges des termes de régularisation de ces deux régressions, contrôlé par le ratio de mélange R.

Si R=0, régression Ridge (Ridge mieux mais si variables inutiles alors Lasso ou Elastique Net).

Si R=1, régression Lasso (Mieux que Lasso si variables >>> exemples ou variables corrélés).

### Régression polynomiale

Si sur-ajuster, alors :

* 1. Augmente la taille des données.
  2. Appliquer une régularisation l1 ou l2 (=> réduira aussi les degrés de liberté du module).
  3. Diminuer le nombre de degré de liberté.

Est-ce que tous les algorithmes de DG aboutissent-ils au meme modèle si vous les laissez s’exécuter suffisamment longtemps ?

Si la function à optimiser est convexe et que le taux d’apprentissage n’est pas trop élevé 🡺 tous les algorithms de DG s’approcheront du minimum global ~ modèles similaires.

Si on ne réduit pas graduellement le taux d’apprentissage, les DG stochastique et par mini-bach ne convergeront jamais (autour de la solution meme si iteration >>>)

Remarques :

* ***Variable\_*** : scikit-learn enregistre tout ce qui est dérivé des données d’apprentissage dans des attributs qui se terminent par un trait.
* **ELASTICNET** : Scikit-learn propose une classe ELASTICNET qui est un mélange du Ridge et Lasso. En pratique, elle fonctionne bien mais au prix d’un ajustement à effectuer entre deux paramètres (L1 et L2).

Modèle linéaire pour la classification

Y­ = w [0]\*x [0] + w [1]\*x [1] + … + w [p]\*x [p] +b, p+1 caractéristiques, b Є > 0.

Au lieu de retourner simplement la somme pondérée des caractéristiques, nous effectuons un seuillage de la valeur prédite par rapport à zéro. Si la fonction est négative, nous prédisons la classe en tant que -1 sinon +1. Cette règle est utilisée pour tous les modèles de classification.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

---------------------------------------------------------------------------------------------------------------------

## Model pour la classification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classification pour les modèles linéaires | | | | |
| Régression logistique | | | | SVM |
| From sklearn.learn\_model import LogisticRegression | | | |  |
| Estime la probabilité qu’une observation appartienne à une classe particulière. | | | |
| Classe positive | | Classe négatif | |
| * Les deux modèles ont des frontières de décision très semblable * Elles appliquent une régularisation L2 (Par défaut), comme le fait Ridge | | | | |
| Paramètres | | | | |
| C >>>  Moins y'a de régularisation, moins on met l'accent sur la bonne classification des points. | C <<<  Forte régularisation, conduit à de nombreux coefficients nuls. | | PENALTY   * - "l1" * - "l2" | |

|  |
| --- |
| Estimation de la probabilité :  Fonction sigmoide [0 :1]  Retourne la logistique (la somme pondérée + intercept) |
| Entrainement et fonction de cout :  Perte logistique (Log loss) :  N’existe pas de solution analytique pour la regression logistique.  La fonction de cout est convexe => DG est assuré de trouver un minima global |
| Prédiction :  0 si < 0.5, 1 sinon |

## Modèle pour la classification

Slope-intercept form:

La prédiction est effectuée en utilisant la formule :

Seuillage de la valeur prédite par rapport à 0

Si la fonction est négative 🡪 classe -1

Si la fonction est positive 🡪 classe +1

Il existe plusieurs façons de trouver les coefficients 𝑊et l’intercept 𝑏, ainsi que différents types de régularisation.

La frontière de décision est une fonction linéaire de l’entrée.

Un classifier linéaire binaire est un classifier qui sépare deux classes en utilisant une ligne, un plan ou un hyperplan.

Algorithme :

* Régression logistique (≠ régression linéaire) `sklearn.linear\_model.LogisticRegression` : Estime la probabilité qu’une observation appartienne à une classe particulière.
* Support Vector Machine ou séparateur à vaste marge (SVM) `sklearn.svm.LinearSVC`
* LinearSVC et LogisticRegression appliquent une régularisation (par défaut). Mais, peuvent aussi appliquer une régularisation , si on souhaite réduire le nombre de caractéristiques ou que le modèle soit facilement interprétable.
* LinearSVC et LogisticRegression possède un paramètre d’équilibrage ou de compromis 𝐶.
  + C >> :
    - Ajuster le jeu d’apprentissage le mieux possible
    - Le modèle met l’accent sur l’importance pour chaque point de donnée individuel d’être classifié correctement
    - Moins y'a de régularisation, moins on met l'accent sur la bonne classification des points
  + C << :
    - Forte régularisation, conduit à de nombreux coefficients nuls
    - Le modèle met l’accent sur le fait de trouver des coefficients 𝑊, b proches de 0
    - Le modèle essaye d’ajuster la majorité des points de données
* Comme pour la régression, les modèles linéaires peuvent sembler restrictifs dans les espaces à petites dimensions, en autorisant des frontières de décision : droite, plan, hyperplan.
* Les modèles linéaires peuvent devenir très puissants avec des espaces à grandes de dimensions.

Régression SoftMax (exponentielle normalisée) : LogisticRegression (multi\_class = ‘multimonial’, solver = ‘lbfgs’, C=10)

Prend en compte plusieurs classes, sans avoir à entrainer plusieurs classifieurs binaires puis à les combiner (comme la stratégie OvR). Elle ne prédit qu’une seule classe (=> utilisé avec des classes mutuellement exclusives), est multi-classe mais pas multi-sortie.

Objectif : est d’avoir un modèle qui estime une forte probabilité pour la classe ciblée => minimiser la fonction de cout « entropie croisée » (pour mesurer l’écart entre deux distributions de probabilités).

### **Classifieurs bayésiens**

|  |  |  |
| --- | --- | --- |
| Classifieurs bayésiens | | |
| - Une famille de classifieurs.  - Ils sont assez semblables aux modèles linéaires.  - Ils sont plus rapides mais la généralisation est moins bonne.  - Ils sont efficaces car les paramètres sont déterminés en analysant individuellement les caractéristiques. | | |
| GuaussianNB | BernouilliNB | MultinomialNB |
| Pour la classification textuelle | |
| Appliquée à des données continues quelconques. | Suppose que les données sont binaires. | Appliquée a des données qui sont dénombrements de quelque chose. |

### Arbre de décision

Des modèles largement employés pour des taches de classification et de régression. Ils se construisent sous la forme d’une hiérarchie de question de type si/sinon qui amène à une décision.

Entrainer un arbre de décision consiste à former la séquence de questions si/sinon (*test*) qui nous conduise le plus rapidement possible ç la bonne répondre.

Les données sont représentées sous forme de caractéristiques continues.

Estimator\_ = le nombre d’arbre pour construire la foret (une valeur élevée est préférable).

N\_jobs = -1 pour utiliser tous les cœurs disponibles dans l’ordinateur.

Samples =le nombre d’observation d’entrainement sont passées par ce nœud.

Value = tableau de n classes ; indique combien d’observation d’entrainement de chaque classe sont passées par là.

Gini = mesure sont impureté, un nœud est dit pur si toutes les observations qui y aboutissent appartienne à la meme classe (pur= gini =0).

Scikit-learn : utilise l’algorithme CAR, qui produit des arbres binaires. L’algothime sépare les données en deux sous-ensembles les plus purs en utilisant une seule caractéristique k et un seuil (exemple : pétal <= 20). Il minimiise la fonction de cout J(k, et ainsi de suite jusqu’à ce qu’il n’est plus de donnée a partagé ou atteindre max\_depth.

CART est un algorithme glouton, approche la meilleure solution mais pas forcément la plus optimale, c’est un problème Np-Complet O(exp(n))

Citerion = geni ou entropy (mesure l’impureté)

Modifier max\_depth va contraindre le modèle et le régulariser en cas de sur-ajustement.

|  |  |
| --- | --- |
| Gini | Entropie |
| Gini rapide à calculer, lorsqu’elle diffère. Gini à tendance à isoler la classe la plus fréquente dans une branche de l’arbre, | entropy d’un jeu de donnée est nulle lorsque les observations appartiennent à une seule classe tandis que l’entropie produit en general des arbres légèrement plus équilibré |

|  |  |
| --- | --- |
| **Prédiction sur un nouveau point de donnée** | |
| Classification From sklearn.tree import DecsionTreeClassifier | Regression From sklearn.tree import DecsionTreeRegressor |
| Est réalisée en vérifiant dans quelle région de partition se trouve ce point. | Est la cible moyenne des points d’entrainement pour la feuille. |
|  | N’est pas capable d’extrapolation (de faire des prédictions en d’hors du champ des données d’apprentissage) |
| **Contrôler la complexité des arbres de décision**  Deux stratégies servent à éviter le sur-apprentissage | |
| Pré-élagage (Pré-pruning) | Post-élagage (Post-pruning) |
| Stopper la création de l’arbre, par :  - La limitation de la profondeur maximale de l’arbre *‘max\_depth’*.  - La limitation du nombre maximal de feuilles ‘*max\_leaf\_nodes’*  - Imposer un nombre minimum de points dans un nœud pour qu’il soit partageable ‘*min\_samples\_leaf’*. | Apres construction de l’arbre, on supprime ou regroupe les nœuds qui contiennent eu d’information |

Les arbres de décisions présentent des avantages sur les modèles précédents :

+ Facile à visualiser et compris par des non-experts (arbre de taille raisonnable).

+ Ils sont invariants quant au recalibrage des données, du fait que chaque caractéristique est traitée séparément.

+ Aucun prétraitement (normalisation ou standardisation des caractéristiques à n’est nécessaire.

+ Fonctionnent bien si les caractéristiques ont des échelles totalement différentes, ou encore un mélange de caractéristique binaires et continues.

- Malgré le pré-élagage ils tendent au sur-apprentissage, c’est pour quoi on applique des méthodes d’ensemble à la place d’arbre de décision uniques.

- Sensible à l’instabilité, c’est pourquoi on utilise des FA.

Ensemble d’arbre de décision

Sont des méthodes qui combinent de multiples modèles d’apprentissage automatique pour créer des modèles plus puissants, aussi bien pour la classification que pour la régression, qui se basent sur les arbres de décision.

Bagging (avec remise) pasting (sans remise)

Boosting

ADABOOST

|  |  |  |  |
| --- | --- | --- | --- |
| **Forets aléatoires** | | **Gradient boosting** | |
| * Une solution au problème de sur-apprentissage. * Une collection d’arbre de décision, chaque arbre est différent des autres | | * Idem | |
| RandomForestRegressor | RandomForestClassifier |  | GadientBoostingClassifier |
| La prédiction on fait la moyenne des résultats. | La prédiction se fait par un soft voting (vote souple, prédire la classe ayant la plus grande moyenne des probabilités sur l’ensemble des classifieurs)  Chaque arbre fournit une probabilité pour chaque étiquette de sortie possible.  La prédiction est celle de la plus forte probabilité. |  | |
| * Choisir le nombre d’arbre **’n\_estimators’** * Les arbres sont indépendants les uns des autres * Pour construire un arbre, on prend un **échantillon** Bootstrap (Bootrapping, Bootstrap aggregating**), tirage aléatoire** avec remise parmi n\_samples => **création d’un jeu de donnée de même taille** (certains seront **répétés** d’autres **absents**) que le jeu d’origine. * Chaque nœud **sélectionne aléatoirement** un sous-**ensemble de caractéristiques**, puis il recherche le meilleur test possible ‘*max\_features’*.   + Si **‘*max\_features’ ~ n\_features****, chaque partage peut prendre en compte* ***toutes les caractéristiques*** *du jeu de donnée =>* ***Les arbres sont semblables.***   + *Si* ***‘max\_features’ =1****,* ***les partages n’ont pas*** *le* ***choix*** *quant à* ***la caractéristique à tester****, et que la seule possibilité est* ***d’affecter des recherches sur certains seuils de la caractéristique*** *sélectionnée de manière aléatoire.*   + *Si* ***‘max\_features’ <<,*** *les* ***arbres sont différents****, qu’ils* ***auraient besoin d’être très profond******pour*** *bien ajuster les données,* ***réduit le sur-apprentissage.***   *Max\_features = sqrt(n\_features) si classification*  *Max\_features = n\_features si régression*   * Chacun d’eux commet plusieurs erreurs car certains des points ne se trouvent pas dans le jeu de d’apprentissage à cause de la méthode de tirage avec remise. | | **Fonctionne en construisant des arbres** de manière **sérielle**, chaque arbre **essaye de corriger les erreurs faites par le précédant**.   * Pas d’aléatoire * **Fort pré-élagage** (réglage avec max\_depth) * **Arbre très peu profond** (1-5 niveaux) * + **Moins de mémoire donc prédiction plus rapide**.   Paramètre :  Learning\_rate : Degré d’intensité de correcteur des erreurs.   * Si Learning\_rate >> Alors chaque arbre corrige davantage => modèle complexe.   Pour diminuer l’exactitude du training score 100% :   * Pré-élagage (max\_depth). * Learning\_rate. | |
| Moins sujette au sur-apprentissage que chacun pris individuellement. | | Idem. | |
| Une application du monde réel. | | Une application du monde réel. | |
| Donne une **exactitude** de **97**% ce qui est **meilleur** qu’avec des M linéaires ou avec un arbre de décision seul. | | Idem | |
| **- Définir une valeur pour le paramètre random\_state** (cela peut **changer considérablement le modèle** qui est construit)  **- Mauvais comportement** avec les **données éparses et/ou multiples dimensions** (ex : données textuelles) dans ce **cas les modèles linéaires sont mieux appropriés**.  **+/-** Parallélisable mais demandent beaucoup de mémoire.  **-Lente à entrainer et à prédire que des modèles linéaires**. | | - **Ajustement précis des paramètres.**  - **Entrainement long**  + **Pas besoin de rééquilibrer les données.**  + **Type des caractéristiques binaires** ou **continues.**  - **Mauvais comportement** avec les **données éparses et/ou multiples dimensions**.  - **N\_estimators et learning\_rate sont interconnectés.**   * Si **learning\_rate <<** Alors **n\_estimtors >>.** | |
| **On utilise beaucoup d’arbres (des centaines ou des milliers)** ce qui **produit des frontières plus lisses**. | | **n\_estimators** est **de préférable petit** **sinon modèle complexe et donc sur-apprentissage.** | |
| **L’importance des caractéristiques** n’est **pas nulle**. | | **L’importance des caractéristiques** peut être **nulle (ignorées)**. | |

Bootstrap : mécanisme d’échantillonnage pour construire des arbres diversifiés afin d’améliorer les performances, variantes :

Bagging : bootsrap aggregating

Après la création de base aléatoires (avec/sans remise) …

Agrégation des prédictions.

+ réduit la variance du modèle.

FA

Un sous ensemble des attributs pris aléatoirement, le partitionnement sur les valeurs des attributs.

Extrem tree (variante des FA)

Attributs sont aléatoires et la partition sur ces valeurs est prise aléatoirement (rien à apprendre).

Boosting : combine des arbres très simples (faibles), chaque arbre essaye de corriger son prédécesseur. 2 variantes :

AdatBoost et Gradient Boosting

1. adatBoost : pondérer les exemples mal classé et … (modifie un peu les poids)
2. Gradient Boosting

Adaboost : technique d’apprentissage séquentielle ~ DG, sauf qu’ajuster les paramètres d’un seul prédicteur pour minimiser les poids d’une fonction de cout, Ada ajuste des prédicteurs à l’ensemble en les améliorant progressivement.

Corrige son prédécesseur en considérant les points qu’il a sur-ajuster, les suivant se concentrent de plus en plus sur les cas difficiles.

Arbre 1 = un arbre de décision, entrainé puis effectue des prédictions, les poids relatifs aux observations mal classées sont alors accrus.

Le second classifieur utilise mes poids modifiés, entrainé puis effectue des prédictions, poids modifiés ….

-Non parallélisable

-Non adapté aux données de grande taille.

### SVM à noyau (séparateur à vaste marge) :

Nous avons vu les SVM pour la classification dans les modèles linéaires. Les SVM à noyau en constituent une extension qui permet de former des modèles plus complexes qui ne sont pas simplement définis par des hyperplans dans l’espace d’entrée. Ils s’appliquent à la classification comme à la régression.

Comme vu précédemment les modèles linéaires sont assez limitatifs pour les espaces de faibles dimensions, car les lignes et les hyperplans ont une flexibilité limitée. Une façon de s’y prendre pour rendre un modèle linéaire plus souple consiste étendre l’ensemble des caractéristiques. Autrement dit, à y ajouter des caractéristiques supplémentaires/axes supplémentaires, par exemple des intersections ou des variables polynomiales.

Par rapport aux caractéristiques d’origines, le modèle linéaire n’est en fait plus linéaire. Ce n’est en effet pas une ligne, mais plutôt une ellipse (voir exemple page 99).

# **L’astuce noyau :**

Ajouter des caractéristiques non linéaires à la représentation de nos données peut rendre les modèles linéaires bien plus puissants.

Quelle caractéristique ajouter ?

Trop de possibilité…

Trop de cout de traitement...

* D’où l’astuce noyau, qui consiste à calculer directement la distance (plus précisément, les produits scalaires) des points de données pour la représentation étendue des caractéristiques, sans jamais calculer réellement cette extension.

Il existe deux manières de s’y prendre

Il considère tous les polynômes possibles à tous les degrés mais avec une décroissance de l’importance des caractéristiques pour les degrés les plus élevés.

Calcule tous les polynômes possibles jusqu'à un certain degré de caractéristiques d’origine (features1\*\*2 features2\*\*5)

SVM apprend quelle importance chaque point de donnée du jeu d’apprentissage a pour la représentation de la frontière de décision entre les deux classes. Seul un sous-ensemble de ces points compte vraiment pour définir la frontière de décision : ce sont des échantillons qui se trouve prés de la frontière entre les classes => on les appelle des vecteurs supports.

La prédiction sur un nouveau point, on mesure la distance jusqu’à chacun des vecteurs supports, et de l’importance de ces vecteurs (déterminé au cours de l’entrainement sous dual\_coef\_ de SVC).

*Ajuster les paramètres du SVM :*

|  |  |
| --- | --- |
| Détermine la portée de l’influence d’un certain échantillon de donnée. | C est un paramètre de régularisation. Il limite l’importance de chaque point (coefficient dual\_coef\_). |
| **ꝩ** Є [0.1, 10] | C Є [0.1, 1000] |
| **ꝩ <<** correspond à une grande portée.  Plus le rayon du noyau gaussien est large, et plus l’influence de chaque échantillon du jeu d’apprentissage est grande.   * Grand rayon pour le noyau gaussien. * Nombreux points considérés comme proche. * Frontière de décision lisse.   Une valeur basse, indique que la frontière va varier lentement =>   * Modèle moins complexe. | **C >>** On renforce l’influence, de ce fait :   * La frontière de décision prend une forme courbe pour classifier correctement tous les points. |
| **ꝩ >>** correspond à une portée limitée.   * Modèle plus complexe. * Une frontière qui se concentre davantage sur des points individuels. | **C<<** implique que chaque point de donnée ne peut avoir qu’une influence très limitée sur le modèle.  Les points mal classés ont une influence négligeable sur la ligne. |

Avantages/inconvénients :

+ Les SVM donnent généralement de bon résultat.

- Ils sont très sensibles au paramétrage et au réglage des données.

- En pratique**, ils exigent** que toutes les **caractéristiques** **varient selon une échelle similaire**.

**Solution**

Recalibrer chacune des caractéristiques afin qu’elles se trouvent approximativement toutes sur une même échelle. Ils existent deux méthodes courantes pour les SVM à noyau.

Prétraitement MinMaxScaler

Recalibrer les données pour qu’elles soient comprise entre 0 et 1

Forces, faiblesses et paramètres.

|  |  |
| --- | --- |
| Forces | Faiblesses |
| * Puissant. * Efficaces avec une large variété de donnée. * Permettent d’obtenir des frontières complexes. * Fonctionnent bien avec des données ayant beaucoup de dimension peu de dimension contrairement aux Modèles linéaires. * Intéressant à tester avec des pixels (même échelle). | * Ne s’adaptent pas trop bien aux grandes quantités de données (problématique en termes de temps et de consommation de mémoire si >> 100 000). * Prétraitement des données et ajustement précis des paramètres (c’est pourquoi on préfère en pratique les arbres). * Difficile d’expliquer une prédiction a des non-experts. * Corrélation forte entre les paramètres ꝩ et C. |

# **Réseau de neurones :**

1. **Fenêtre temporelle**

Une famille d’algorithme connu sous le nom de réseaux de neurones/ perceptrons multicouches/ MLP/ réseaux de neurones feed-forward (non bouclés ou acycliques, ou encore propagation directe) voient récemment une nouvelle jeunesse sous l’application d’apprentissage profond ou deep Learning applicable a la régression et la classification.

Ils sont vus comme une généralisation des modèles linéaires effectuant de multiples étapes de traitement pour arriver à prendre une décision.

Après calcule de la somme pondérée pour chaque unité caché on applique au résultat une fonction non linéaire :

* Une unité de rectification linéaire - RELU (Rectified Linear Unit).

Coupe les valeurs en dessous de zéro.

* Tangente hyperbolique - TANH.

Sature à -1 les valeurs d’entrées basses et à +1 les valeurs d’entrées élevées.

1. Ajuster les réseaux de neurones :

Il y’a plusieurs manières de contrôler la complexité d’un RN :

* Le nombre de couches cachées.
* Le nombre d’unités dans chaque couche cachée.
* Le paramètre de régularisation (alpha).

Attention :

Les **coefficients de pondération** sont **définis** de manière **aléatoire** **avant que l’entrainement ne débute**, et cela aléatoire **affecte** évidemment le **modèle** qui est entrainé. Ce signifie que, même **si on utilise exactement les mêmes paramètres**, nous **obtiendrons des modèles très différents** selon la manière dont le générateur de nombre aléatoires est initialisé. **Si le jeu de données est >>** et **complexité bien choisie** alors **pas** de **problème**. Si le jeu de données << alors faut garder cette notion en tête.

Les RN **s’attendent aussi que toutes les caractéristiques d’entrée** **varient de manière similaire**, et qu’elles aient dans l’idéal une **moyenne** **de 0 et une variance de 1** (StandarScalar).

Rmq : nombre de bons modèles atteignent la mémé exactitude 0.972, tous les modèles font exactement le même nombre d’erreurs, soit 4. Cela peut provenir du fait que le jeu de donnée a une taille réduite, ou que les points concernés sont réellement différents des autres.

Avec Python : MPLRegressor – MPLClassifier (ce n’est pas les plus complet).

1. Avantages et inconvénients

|  |  |
| --- | --- |
| Force | Faiblesse |
| **+** Peuvent capturer des informations contenues dans de vastes quantités de données.  **+** Bonnes données + bon réglage + temps de calcule suffisant => battent tous les autres algorithmes (Classification et régression).  **+** Construire des modèles très complexes. | * Prétraitement soigneux des données, travaillent avec des données homogène (caractéristique ont des significations similaires, comme SVM).   Si les caractéristiques sont de natures différentes, les arbres sont mieux.   * Réglage des paramètres. * Temps long pour l’entrainement. |

Estimer la complexité des réseaux de neurones

Une mesure utile pour juger de la complexité d’un RN est le nombre de poids ou de coefficients qui ont été entrainés.

Si **classification binaire** & # [**caractéristique**] = 100 & 100 **unités cachées**

=>

100 \* 100 = 10000 poids entre l’entrée et la première couche cachée + 100 \* 1 = 100 Poids entre la dernière couche cachée et la couche de sortie

…

**Une manière courante d’ajuster les paramètres** dans un RN consiste à commencer par créer un RN suffisamment **grand pour engendre un sur-apprentissage**, en s’assurer sue la tache peut être entrainé par le réseau. Puis

* Soit on **réduit** le RN,
* Soit on **augmente la valeur d’alpha** pour ajouter de la régularisation ce qui améliore les performances de généralisation.

1. Algorithme d’entrainement :

|  |  |  |
| --- | --- | --- |
| Adam | Lbfgs | Sgd |
| + Bien pour la plupart des situations.  - Sensible aux données (entre 0 et 1). | +Robuste.  -Lent avec les modèles complexes et de grande taille. | +Pour les chercheurs en depp learning.  +/-Paramètre supplémentaires à ajuster. |

1. Estimer l’incertitude pour les classifieurs binaires :

Nous ne sommes pas seulement intéressés de savoir si l’exemple appartient ou pas un une classe, mais on souhaite aussi savoir le degré de certitude du classifieur a quant à la justesse de cette prédiction. Il existe deux fonctions sous scikit-learn :

|  |  |
| --- | --- |
| Decision\_function | predict\_proba |
| Le modèle nous dit avec quelle force il croit qu’un point appartient à une classe ‘positive =1‘ ou à la classe ‘négative =0’ | Le modèle donne une probabilité pour chaque classe.  Plus facile à interpréter. |
|  | En sur-apprentissage le modèle aura tendance à être plus certain pour la prédiction sur tous les points (même si il se trompe). Un modèle mois complexe aura tendance à être incertain. |

Rmq :

La classe négative est toujours la première de l’attribut classes\_.

Un modèle est dit calibré si ses prédictions sont en accords avec la réalité.

## Grande famille de RN :

1. INSERTION IMAGE

**Points négatifs d’un RN FEEDFORWRD** :

* Nombre d’entrée/sortie fixé.
* Complexité du réseau dépend du nombre de couche et du nombre de lien entre les neurones (fixé).
* Difficile pour modéliser les séries temporelles (vidéo, musique et traduction de phrase, le réseau se souvient mal de ce qui se passe avant, ex : Je vais en France, pour apprendre le … ? français).

##### Type de RN :

* One To One
* One To Many (ex: Entrée = Image, Sorties = Phrases pour décrire l’image)
* Many To One (ex: Entrées = Phrases, Sortie = Une émotion)
* Many To Many

##### RNN :

RNN => Très profond. Une couche cachée dépend de la couche précédente, utile pour les séries temporelles.

Avec une E on une S avec une E et une S, on a une autre S

C.C

1. **Point positif/ négatif d’un RNN :**

**+** Possède une mémoire.

**-**En pratique, on a un problème d’entrainement, le gradient explose.

**- - Vanishing problem** gradient le réseau s’arrête, on pense qu’il a fini d’apprendre car la valeur de la différence du gradient est tres petite => solution **clipping** réduit les valeurs du gradient.

##### **LSTM :**

LSTM => Pour palier du problème VPGradient, avec des portes dans les neurones cachées.Rmq :

Entrées : 2 vidéos, si elles sont de tailles différentes (n’ont pas le meme nombre de pixel, Le vecteur d’Input n’ont pas la même taille) = > Problème => Solution ROGNER l’image

Si le nombre de couche caché est élevé => problème de puissance des machines. En pratique 2-3 couches caché.

Plus on a de paramètre plus on a besoin de données sinon on Over fit.

Yann Lecan, premier CNN « LENET » une combinaison entre la convolution et les RN

Une image => 16 filtres (convolution), 16 images de même taille => Relu 0 si x < 0 sinon x => Pooling, intervient sur la taille de l’image, on la réduit en gardant un max d’information.

##### Auto-encodeur :

Reproduire en sortie ce qu’il a eu en entrée en passant par un espace plus restreint (comme une PC (qui n’est pas linéaire) on peut ajouter plusieurs layer) cela le force à détecter les combinaisons de dimension les plus intéressantes et capturer le maximum d’information.

Il est composé en deux parties :

1. Encodeur : Image en entrée puis la plongé dans 🡺 🡺 🡺 🡺

🡺 Embedding : (bottleneck) dimension en espace latent réduit la dimension et capture les plus importantes caractéristiques.

1. Décodeur : Qui prend la dimension plus restreinte puis reproduit la sortie. Non supervisé.

Exemple :

Entrée : image 🡺 A l’intérieur : bruité l’image 🡺 Sortie : reconstitué l’image.

**Résumé**

Conseil :

# Chapitre 2 : Apprentissage non supervisé

|  |  |
| --- | --- |
| Transformation du jeu de données | Partitionnement des données (Clustering) |
| 1. Réduction de dimension.    * Nouvelle représentation plus compréhensible.    * Part d’une représentation des données avec de nombreuses dimensions, pour trouver une nouvelle manière de représenter ces données afin d’en déduire ce qui est essentiel.   Souvent utilisé à des fins de visualisation en 2D.   1. Trouver les parties ou composants qui structurent les données, exemples :    * Extraire des thèmes dans des collections de documents textuels. | 1. Partitionnent les données en groupes distincts d’éléments similaires, exemples :  * Réseau pouvant regrouper les photos d’un utilisateur. |

Défis majeurs de l’apprentissage non supervisé est de pouvoir évaluer si l’algorithme a appris quelque chose d’utile. Car ils ne contiennent aucune information d’étiquette, la seule manière est de l’inspecter manuellement. Ces algorithmes sont souvent employés :

* dans une démarche exploratoire, lorsqu’on veut mieux comprendre les données.
* Phase de prétraitement des données pour la construction de modèles supervisés.
* Former une nouvelle représentation.

# Chapitre 3 : Pré traitement et recalibrage

**Les données transformées** ont le **même format** que les données d’origine**. Les caractéristiques sont simplement décalées et recalibre**r. **Il faut** également **transformer le jeu de test**. **Il est important d’appliquer exactement la même transformation au jeu d’apprentissage et au jeu de test** pour le modèle (unique fit sur X\_train). Pour mieux le comprendre on représente le jeu de test comme un point unique. Il n’y a bien sur aucun moyen de recalibrer correctement un seul point pour qu’il remplisse les conditions (exemple MinMaxScaler) de minima et de maxima.

|  |  |  |  |
| --- | --- | --- | --- |
| StandardScaler | RobustScaler | MinMaxScaler –  Normalization | Normalizer |
| S’assure que pour chaque caractéristique, la moyenne est nulle et **la variance égale à 1**. Ont toutes la même amplitude.  Elle ne permet pas de former des valeurs minimale et maximale pour les caractéristiques.  Il est moins affecté par les valeurs aberrantes | Fonctionne de la même manière que StandardScaler, **mais utilise la médiane et les quartiles** au lieu de la moyenne et la variance. De ce fait, **elle ignore les points** **de donnée** qui sont différents du reste (exemple des erreurs de mesure) ils sont dit points aberrants ou **outliers** et peuvent occasionner des problèmes pour d’autres techniques de calibrage. | Décale et recalibre les données afin que **toutes les caractéristiques soient comprises exactement entre 0 et 1.** Toutes les données sont contenues dans un carré créé sur l’axe x entre 0 et 1, et également entre 0 et 1 sur l’axe y.  Calcule-le min/max de chaque caractéristique. Les valeurs sont décalées et recalibrées afin qu’elles se situent toutes entre 0 et 1 ‘sympa pour un RN’ | Il calibre chaque **point de donnée de manière à ce que tous les vecteurs de caractéristiques aient une longueur euclidienne de 1**. Il **projette un point de donnée sur le cercle** (sphère si multi dimension) **de rayon** 1. Elle est appliquée lorsque seule la direction/angle compte et non la longueur des vecteurs de caractéristiques. |

Exemple :

|  |  |
| --- | --- |
| From sklearn.processing import MinMaxScaler  Scaler = MinMaxScalet()  Scaler.fit(X\_train) // que sur le X\_train  X\_train\_scaled = Scaler.transform(X\_train)  X\_test\_scaled = Scaler.transform(X\_test) | From sklearn.processing import MinMaxScaler  Scaler = MinMaxScalet()  Scaler.fit\_tranform(X\_train)) |

# Blanchiment des images => utile

I make the distinction of an object recognition convolutional neural network because this process of PCA whitening can transform your data in such a way that makes an image unsuitable for certain tasks. Indeed, PCA whitening can be used for many more types of applications via a CNN. The idea of Principle Component Analysis (PCA) whitening is similar, but much less visually interesting. You are using PCA to reduce the dimensionality of your data to only the most important information. This makes it much easier for an **object recognition** convolutional neural network (CNN) to get meaningful results from your data, because you have often eliminated most of the **noise** in your image.

ZCA stretches the dataset to make it spherical, but *tries not to rotate it* (whereas PCA does rotate it quite a lot).

PCA – moins bon.

ZCA – garder l’orientation des données, composants ne sont pas orthogonales

ZCA étire l'ensemble de données pour le rendre sphérique, mais essaie de ne pas le faire tourner (alors que PCA le fait tourner beaucoup). ZCA pourrait être préférable, c'est le prétraitement pour les réseaux de neurones convolutifs.

# Réduction de la dimension, extraction de caricaturesques et apprentissage de variétés.

Motivation :

* La visualisation.
* La compression des données.
* La recherche d’une présentation qui soit plus informative pour des traitements ultérieurs.
* Etape préparatoire à une tache d’apprentissage supervisé.

Fléau de la dimensionnalité => : beaucoup de dimension, les vecteurs échantillonnés aléatoirement dans un espace de grande dimension sont souvent éparpillés => accroit le risque de sur-apprentissage.

MNIST (les pixels du bord sont blancs on peut les éliminer du jeu de d’entrainement sans perdre de l’information. Ces pixels ont une importance quasi nulle dans une tache de classification, deux pixels voisins sont souvent fortement corrélés)

## Analyse en composante principale (PCA) :

Il identifie l’hyperplan le plus proche des données, et le projette sur celui-ci.

Il **choisit l’axe qui conserve la plus grande part possible** **de variance**, car il est très vraisemblable qu’il perdra moins d’informations que les autres projections. Pour justifier ce choix, on peut également dire qu’il s’agit de l’axe qui minimise la moyenne du carré des distances entre les données d’origine et leurs projections.

Axe1 : a le plus de variance.

Axe2 : axe orthogonal au premier, qui contribue le plus à la variance résiduelle.

**Le sens des flèches des composantes principales n’est pas stable,** **une petite perturbation du jeu de donnée peut les inversées.**

Attention **: PCA s’occupe de centrer les données pour nous**, si on utilise autre que celle de Sklearn, ne pas oublier de centrer les données avant d’appliquer une PCA.

**Application 1 : visualisation des données, Dim = 2 || 3.**

La méthode transforme des caractéristiques dites corrélées en caractéristiques statistiquement non corrélées. Puis sélectionne un sous-ensemble des nouvelles caractéristiques, selon l’importance qu’elles présentent pour la compréhension des données. L’algorithme commence par trouver la direction du maximum de la variance (component 1). Il s’agit de la direction (ou du vecteur) dans les données qui contient la plus grande part d’informations ou, en d’autres termes e la direction dans laquelle les caractéristiques sont le plus corrélées les unes avec les autres. L’algorithme trouve ensuite la direction qui contient le plus d’informations tout en restant orthogonale à la première. L’orientation n’a aucune importance. Les directions déterminées par ce processus sont appelées **composantes principales**, **car elles déterminent les principales directions de la variance dans les données**. Il existe autant de composantes principales que de caractéristiques originales. **La PCA se charge de la rotation et du décalage des données**, mais en conservant toutes les composantes principales.

Exemple :

Scaler = StandardScaler ()

Scaler.fit(X\_train)

X\_train\_Scaled = Scaler.tranform(X\_train)

From sklearn.decomposition import PCA

Pca = PCA(n\_components= 2)

Pca.fit(X\_train\_Scaled)

Pca.transform(X\_train\_Scaled)

La PCA est une méthode non supervisée, elle n’utilise aucune information provenant des classes pour trouver la rotation. Elle cherche simplement la corrélation dans les données.

*En page 149 axe-x = component principale 1, axes-y = component principale 2. On constate depuis la figure que les deux classes sont assez bien séparées, cela nous conduit à croire que même un classifier linéaire pourrait faire un travail raisonnable.*

**Application 2 : extraction de caractéristiques.**

Utile pour le classement d’image, la sécurité …

Exemple : on a une base avec des photos de personne.

Idée 1 : On construit un classifier par personne (classes distinctes). Cependant, il y’a peu d’image par classe (de la même personne) donc très peu d’échantillon d’apprentissage par classe, l’ajout d’une nouvelle personne (classe) nous oblige a ré-entrainer tout le modèle (qui devient de plus en plus volumineux).

Idée 2 : on construit un classifier KNN, mais on obtient un score 0.23

La PCA rentre en scène, car calculer la distance entre les pixels n’est pas une bonne idée pour mesurer la similitude entre les visages.

Lorsqu’on fait appel à une représentation sous forme de pixel pour comparer des visages, nous comparons en fait la valeur de niveau de gris pour chaque pixel à la valeur correspondante pour le pixel qui occupe la même position dans l’autre image. Si un pixel se voit décaler se traduit par une représentation différente. Ce qu’on espère, c’est qu’utiliser des distances en même temps que les composantes principales puissent améliorer notre exactitude.

La PCA fait une transformation dans laquelle les données subissent une rotation, les composants ayant une faible variance étant ensuite abandonnés. Il est aussi utile de voir la reconstruction des données en se servant de certaines composantes. Si component principale = nombre de pixel de l’image => on ne perd aucune information à la suite de la rotation.

**Réduire la dimension perd toujours un peu d’information et par conséquent, meme si cela accélère l’entrainement, cela peut aussi dégrader légèrement les performances de votre système et rend les pipelines plus difficiles à maintenir.**

* + Conseil **: On doit d’abord entrainer le système avec les données d’origine, puis envisager une réduction de dimension des données d’entrainement peut filtrer certains aléas et détails inutiles, occasionnant ainsi une amélioration des performances ; mais en général, cela permet uniquement de réduire la durée d’entrainement.**

De visualiser les données **; représenter les données en 2D ou 3D permet parfois d’en tirer des renseignements importants en détectant visuellement une structuration telle que des partitions.**

1. Si la dimension est petite, alors la probabilité de choisir au hasard un point dans un carré unité 1X1, il n’aura que 0.4% de chances d’etre situé au moins de 0.001 d’une frontière (improbable qu’un point pris au hasard soit extrême sur une quelconque des dimensions). Mais un hypercube de dimension 10000, la probabilité est de 99.99%, la plupart des points sont très proches du bord.
2. Carré unité 1X1, la distance moyenne entre deux points sera en moyenne de 0.52, **dans un cube unité 0.66 et dans une très grande dimension la distance est >>>>> car : Les jeux de données de grande dimension risquent d’etre très épars (la plupart des observations risquent d’etre très éloignées les unes des autres)** 🡺 **ce qui rend les prédictions moins fiables, car elles seront basées sur des extrapolations plus larges** 🡺 **sur-apprentissage.**

En pratique, on **peut combattre le fléau de la dimension en accroissant la taille des données de façon à atteindre une densité suffisante des observations**. Il faudrait plus d’observation d’entrainement que d’atomes dans l’univers observables pour que ces observations soient à une distance moyenne de 0.1 l’une de l’autre, en supposant qu’elles soient réparties uniformément sur toutes les dimensions.

**Avantage et inconvénients d’une PCA Ordinaire :**

* Entrainement plus rapide.
* Elimine le bruit et les variables redondantes.
* Visualiser les données pour avoi une idée des variables les plus importantes.
* Compression (économiser l’espace).
* Perte de l’information => dégrade les performances.
* Ne garantit pas qu’on aura une meilleure ou plus simple solution « tout dépend du jeu de donnée ».
* Ajoute de la complexité au pipeline.
* Variable transformer sont souvent difficile à interpréter.

Apres utilisation d’une réduction de dimensionnalité, il est impossible de faire marche arrière, car une partie de l’information est perdue lors de la réduction. Certain algorithme comme **PCA peuvent possèdent une transformation inverse qui peut reconstruire un jeu assez sembles**, d’autres ne peuvent pas comme t-SNE.

Evaluer une Réduction de Dimensionnalité :

* Appliquer une transformation inverse, et mesurer l’erreur de de reconstruction.
* Verifier le score d’un algorithme d’apprentissage automatique pour les données d’origine et sur les données transformées. Si, peut écart (à la suite de la perte d’informations) alors bon, mauvais sinon.

SVD méthode de composition en valeur singulière qui permet de décomposer la matrice X en un produit de 3 matrices : U sigma V.T, V.T contient toutes les composantes principales.

U, sigma, v.t = Np.linalg.svd(X) % X = X – X.means(axis=0)

v.t[ :,0], v.t[ :,1]

PCA s’occupe que les données soient centrées autour de l’origine

pca.explain\_variance\_ratio

Choisir le nombre de dimensions représentant une continuation suffisamment importante à la variance (au moins 95%).

**Application 3 : Compression**

On peut réduire la taille du jeu de donnée en prenant l’essentiel de la variance. Ce qui permet d’accélérer considérablement un algorithme de classification (tel que SVM).

Il est aussi possible de décompresser le jeu de données réduit pour le ramener à sa taille d’origine en appliquant la transformation inverse. Il faut savoir qu’on ne trouvera pas exactement les données d’origine puisque la projection à perdu un peu d’information, mais les nouvelles données seront vraisemblablement très proches.

Erreur de reconstruction = distance quadratique moyenne entre les données d’origine et les données reconstruites.

« inverse\_transform()

### PCA incrémentale :

Si on utilise la PCA ordinaire le jeu de données doit tenir en mémoire pour pouvoir effectuer la recherche des valeurs singulières.

🡺PCAI effectue une analyse en composante principale en ligne (à la volé, au fure et à mesure de l’arrivée de nouvelle observation).

### PCA randomisée : PCA (n\_components=n, svd\_solver= ‘randomized’)

Algorithme stochastique trouvant rapidement une approximation des d premières composantes principales. Elle donc considérablement plus rapide que les algorithmes précédents.

### Projection à noyau :

Une technique mathématique qui projette implicitement les observations dans un espace de tès grande dimension appelé espace de rescription

Projection plus complexe, elle donne de bon résultat lorsqu’il s’agit de préserver des regroupements d’observation après projection. Ou parfois meme pour dérouler des jeux de données se situant à proximité d’une variété enroulée.

Est un algorithme non supervisé, il n’y a pas de mesure de performance pour nous aider à sélection le meilleur noyau et les valeurs des paramètres. Cependant, la réduction de dimensionnalité est souvent une étape préparatoire à une tache d’apprentissage supervisé.

Pipe = Pipeline ([

(‘pca’, KernelPCA(n\_component =2)), (‘log\_reg’ , LogisticRegression())

])

Params = [{‘pca\_\_gamma’ : np.linspace(0.03, 0.05,10) ,

‘pca\_\_kernel’ : [’rbf’, s=’sigmoid’}]

Grid = GridSearchCV(pipe, param\_grid = params, cv =3)

Grid.fit(X\_train,y\_train)

## Factorisation en matrice non négative (NMF) :

**Pré condition : données >= 0**

Est un algorithme **non supervisé visant à extraire des caractéristiques utiles**. Il fonctionne d’une manière semblable à la PCA, et peut aussi servir à la réduction de dimension.

On essaye d’écrire chaque point de donnée sous **la forme d’une somme non négative pondérée de certaines composantes** (les valeurs >= 0) contrairement la PCA qui elle cherche des composantes orthogonales et une variance maximale. La NMF n’est pas appliquée aux données pour lesquelles les caractéristiques sont négatives.

La NMF est utile pour des données qui ont été créés sous forme d’addition ou de recouvrement de plusieurs sources indépendantes (exemple : enregistrement audio sur lequel plusieurs personnes parlent, ou orchestre avec plusieurs instrument). Dans de telles situations NMF est capable d’identifier les composantes originales constituant les données combinées. Elle produit des composantes et des coefficients mieux interprétables que la PCA.

LDA

Réduire le nombre de composant ne supprime pas certaines directions, mais crée en fait un ensemble totalement différent de composantes. Dans NMF les composantes jouent toutes à égalité, car il n’existe pas d’ordre particulier. NMF utilise une initialisation aléatoire, ce qui pourrait conduire à des résultats différents.

Le nombre de composantes qu’on souhaite extraire << le nombre de caractéristiques en entrée (sinon on considère que chaque pixel est une composante distincte.

Faible pour la reconstruction ou l’encodage des données.

Fort pour trouver des motifs intéressants dans ces données.

## LLE

Une technique non linéaire très efficace de réduction de dimension. C’est une technique d’apprentissage de variété ne reposant pas sur des projections comme les algorithmes précédents. En bref, LLE constitue à mesurer tout d’abord les relations linéaires entre chacune des observations du jeu d’entrainement et ses plus proches voisines, puis à rechercher une représentation du jeu d’entrainement dans un espace de faible dimension qui préserve au mieux les relations locales. Ceci rend cette technique particulièrement efficace pour aplanir des variétés enroulées, en particulier lorsqu’il n’y pas trop de bruit.

-les distances ne sont pas préservées à plus grande échelle (une partie devient rétrécie/élargie alors qu’elle ne l’été pas)

+effectue en général une bonne modélisation de la variété.

Fonctionnement :

Problème d’optimisation sous contrainte

* + - 1. Identifie les k plus proches voisines de chacune des observations d’entrainement
      2. Tente de reconstruire comme fonction linéaire de ses voisins. Il recherche les poids tels que le carré de la distance entre et soit la plus petite que possible
      3. Projeter les observations d’entrainement dans un espace à d dimension tout en préservant autant que possible ces relations locales.

Complexité algorithmique :

O(m log(m) n log(k)),pour la recherche des plus proches voisins

O(m n k^^3) pour l’optimisation des poids

O(d m^^2) pour la construction des représentations en faible dimension.

LLE n’est pas adapté aux données de très grande taille.

## Apprentissage de variétés avec t-SNE

Ce sont des algorithmes destinés **à la visualisation et non à la génération de nouvelles caractéristiques.**

t-SNE calcule une nouvelle représentation des données d’apprentissage, mais ils ne permettent pas d’effectuer des transformations de nouvelles données. Cela implique que ces algorithmes ne peuvent pas être appliqués aux données de test.

+Utile pour l’analyse exploratoire.

+/-Rarement utilisé lorsque le but final est un apprentissage supervisé.

Le but : **Trouver une représentation bidimensionnelle des données qui préserve le mieux possibles les distances entre les points**. Il débute par une représentation aléatoire en 2D pour chaque point de donnée, puis il essaie de rapprocher les points qui sont éloignés l’un de l’autre dans l’espace des caractéristiques d’origine, et les points qui sont éloignés. t-SNE met l’accent sur les points qui sont proches, plutôt que de préserver les distances entre des points éloignés.

**Effectue une réduction de dimension tout en essayant de conserver les observations semblables proches et les observations dissemblances éloignées ? cette méthode est utilisée surtout pour la visualisation, et en particulier pour visualiser des groupes d’observations dans un espace de grande dimension.**

MDS : réduit la dimension tout en s’efforçant de préserver les distances entre observations

Isomap : créer un graphe en connectant chaque observation à ses plus proches voisins, puis effectue une réduction de dimension en s’efforçant de préserver les distances géodésiques entre observations.

LDA : algorithme de classification détermine durant **l’entrainement les axes les plus discriminants** entre les classes, axes qui peuvent alors etre utilisés pour définir un hyperplan sur lequel projeter les données. L’intérêt est que la projection conservera les classes aussi éloignées que possibles, c’est pourquoi la LDA est une bonne technique de réduction de dimension avant d’exécuter un algorithme de classification tel qu’un classifieur SVM.

# Clustering

Le clustering consiste à partitionner le jeu de donnée en groupe (cluster) très différents.

### Partitionnement en k-moyennes :

Un algorithme de clustering très simple et utilisés. Il essaie de trouver des centres de clusters qui sont représentatifs de certaines régions des données.

Etape 1 : affecte chaque point de donnée au centre du cluster le plus proche.

Etape 2 : redéfinit le centre du cluster comme étant la moyenne des points de données qui lui ont été affectés.

L’algorithme se termine quand l’affectation des instances aux clusters ne varie plus.

* Echoue à identifier des clusters ayant une forme complexe.

|  |  |  |
| --- | --- | --- |
| PCA | NMF | K-moyenne |
| Trouver les directions ayant la variance maximale dans les données. | Trouver les composantes additives, qui correspondent souvent à des extrémités ou à un axe dans les données | Représente chaque point de donnée en utilisant un centre de cluster. |
| Algorithme de décomposition  Exprimer les données sous a forme d’une somme pondérée d’un certain nombre de composantes | | Algorithme de partitionnement  (Chaque point était une seule composante)  Quantification vectorielle |

Quantification vectorielle :

+ On peut utiliser plus de cluster que de dimensions de l’entrée pour encoder les données.

Si cluster >> Représentation plus expressive.

Si modèle complexe, utiliser plus de cluster pour couvrir les variations => On aura de nouvelles caractéristiques donc plus de dimension ce qui peut nous amener à utiliser un modèle linéaire.

|  |  |
| --- | --- |
| Force | Faiblesse |
| * Facile à comprendre. * Facile à implémenter. * Rapide. * S’adapte à de gros jeux de données. | * Initialisation aléatoire (exécuter l’algorithme 10 fois et choisir le meilleur dans le sens ou la somme des variances des clusters est minimal). * Relativement restreint quant aux suppositions de la forme des clusters. * Spécifier le nombre de clusters (nombre qu’on ne connait pas dans le monde réel). |

# Clustering agglomératif

L’algorithme commence par déclarer chaque point étant son propre cluster, puis il fusionne les deux clusters les plus similaires ou les plus proches jusqu’à satisfaire un critère d’arrêt (nombre d’itération/clusters). Il y’a plusieurs critères de lien (linkage) qui spécifient la façon exacte de mesurer ce que représente les *clusters les plus similaires*, on cite sous scikit-learn :

|  |  |  |
| --- | --- | --- |
| ward | Average | Complete - Maximim |
| * Le choix par défaut. * Il sélectionne les deux clusters à fusionner de manière à ce que la variance augmente le moins à l’intérieur de tous les clusters –l’inertie intra classe. Cela conduit à des clusters qui sont de taille relativement égale. | * Fusionne les deux clusters qui ont la plus petite distance moyenne entre tous les points. | * Fusionne les deux clusters qui ont la plus petite distance maximale entre leurs points. |
| * Fonctionne bien avec la plupart des jeux de données. | * Si les clusters ont des nombres très dissemblables de membres, les liens Average ou complète pourraient être meilleurs | |

**Clustering hiérarchique ascendant**

Procède de manière itérative

Init : chaque point est un cluster lui-même …

* Toutes ces étapes produisent un nouveau partitionnement des données (donne toutes les combinaisons de clusters possibles.

+ Donne une vue très détaillées.

-Mais s’appuie sur des données bidimensionnelles.

Correction de la lacune 2D

## DBSCAN

Pré condition : Données recalibrées StandardScaler ou MinmaxScaler.

+ L’utilisation ne définit pas a priori le nombre de clusters.

+ Capture des clusters de forme complexe.

+ Identifie des points qui n’appartiennent à aucun cluster.

+ S’adapte aux gros jeux de données.

-Lent comparé aux autres méthodes.

-Pas adapté aux petits jeux de données (changer le paramétrage ).

Fonctionnement :

Idée :

* Les clusters forment les régions les plus denses séparés par des régions relativement vides.
* Identifie les points qui se trouvent dans les régions encombrées - denses dans l’espace de caractéristiques (la ou les points sont proches les uns les autres). Ces points sont dits core points (points intérieurs).
* L’algorithme possède deux paramètres :

|  |  |
| --- | --- |
| Min\_samples  **Détermine** le statut des points (aberrants/outliers/anormaux/atypiques) qui se trouvent dans les régions moins denses => **taille du cluster**.  Si min\_samples >> on diminue le nombre de core point et on augmente la quantité de bruit. | Esp (epsilon)  Important car il **détermine le sens** de **proche**.  Si Esp >> on a plus d’exemple dans un cluster.  Si Esp >>> fusionne les clusters (fini avec 1).  Si Eps << il n’y a que du bruit. |

* S’il y’a au moins min\_samples points se trouvat à une distance < esp d’un certain point de donnée, alors celui-ci est classifié comme étant un core point. Les points intérieurs qui sont plus proches les uns des autres que la distance eps sont placés dans le même cluster.

1. L’algorithme choisit un point aléatoirement pour commencer.
2. Identifie les points qui sont < eps, si le nombre de point trouvés < n\_samples alors le point de départ est caractérisé comme un bruit (il n’appartient à aucun cluster) sinon il est un core point.
3. Tous les points proches < esp du cluster sont visités.
4. On boucle.

A la fin on aura 3 types de points :

|  |  |  |
| --- | --- | --- |
| Core point | Boundary point/points frontières (Au voisinage de core point) | Nose/bruit (-1) |

-Si on exécute l’algorithme plusieurs fois les nose et core points restent inchangés. Par contre, l’appartenance à tel ou tel cluster dépendent de l’ordre dans lequel ils ont été visités (mais pas trop important).

Représentation des données et ingénierie des caractéristiques :

|  |  |
| --- | --- |
| Caractéristique catégorielle/discrète | Caractéristique continue/numérique |

Dans le monde réel, les données ne sont pas collectées sous forme d’un tableau. On les retrouve sous deux types de données :

La manière de **représenter les données** peut avoir une **influence plus grande** sur les performances des **modèles d’apprentissage automatique** qu’un *choix approprié* des **paramètres.**

Représenter le mieux possible les données **l’ingénierie des caractéristique / feature engineering**

## Du catégorielle au numérique :

Il existe plusieurs manières de représenter des variables catégorielles en numériques, on citera :

Encodage one-hot (variables indicatrices) :

La manière de loin la plus courante consiste à utiliser l’encodage one out of N ou encore dummy variables.

L’idée : **Remplacer une variable catégorielle par une ou plusieurs nouvelles caractéristiques (1 si vrai 0 sinon).**

*Exemple :*

*Print (data.gender.value\_counts()) % mieux connaitre la valeur des caractéristiques.*

*Print («Original features», list(data.columns)*

*Data\_dummies = pd.get\_dummies(data)*

*Print («Features after get\_dummies», list (data\_dummies.columns)*

*Rmq : les variables catégorielles ont été encodées sous forme de chaine de caractères donc possibilité de rencontrer des fautes d’orthographe ou de saisie. Mais cette présentation montre clairement que ce sont des caractéristiques catégorielles.*

*Attention :*

* *Vérifier le nombre et le type des colonnes après dummy pour les données d’apprentissage et les données de test.*

*Si la variable catégorielle n’est pas encodée en numérique (1-8), il n’est pas évident de se rendre compte que c’est une variable catégorielle et la fonction get\_dummies de panda ne traite ce genre de représentation => Méthode OneHotEncoder.*

*On encode les variables catégorielles en utilisant des entiers.*

*Il ne faut pas les traiter comme étant des variables continues.*

Binning, discrétisation, modèle linaires et ARBRES :

**La meilleure façon de les représenter dépend :**

**-De leur sémantique. -Du modèle qu’on utilise.**

|  |  |
| --- | --- |
| **Modèle** | |
| Modèles arborescents (AD, Boosting ou FA) | Modèle linéaire |

**Propriétés différentes !!**

|  |  |
| --- | --- |
| * La discrétisation n’apporte pas de bénéfice aux modèles arborescents, car ils n’en ont pas besoin pour apprendre à diviser les données (capables de traiter de multiples caractéristiques en même temps). | Ils deviennent plus puissants/flexibles en utilisant :   * **La discrétisation (binning)**, car il maintient une valeur constante pour chaque bin |

Intersection et polynomes:

Une autre manière consiste à ajouter des intersections et des polynômes aux caractéristiques d’origine. Ce type d’ingénierie est souvent employé en modélisation statistique mais aussi en ML.

|  |  |
| --- | --- |
| * Les FA battent la performance de Ridge, et le fait d’ajouter des intersections et des fonctions polynomiales fait même légèrement baisser les performances. | * Etant donnée que les ML être visualisés sous forme de droites possédant une centaine pente. On peut faire une **discrétisation** + **réintroduire la caractéristique d’origine**. * **Discrétisation** + une nouvelle caractéristique (**produit de l’indicateur bin** **et de la caractéristique d’origine**). |

Exemple :

From sklearn.preprocessing import PolynomialFeatures

Poly = PolynomialFeatures(degree = 10), include\_biais =False)

Poly.fit(X)

X\_poly = poly.transform(X) % transformer les données

Print (poly.get\_feature\_names()) % x0, x0\*\*2, x0\*\*3 ...

Linear\_poly = LinearRegression ().fit(X\_poly, y)

Cependant les polynômes élevés ont tendance à se comporter de manière extrême aux limites ou dans des régions ne comportant que peu de donnée.

Les SVM à noyau donnent une prédiction aussi bonne dans le cas d’une régression polynomiale, mais sans transformation explicite des caractéristiques.

Transformation non linéaires uni-variées:

L’ajout des caractéristiques carrées ou cubiques peut aider à **améliorer les performances des modèles linéaires pour la régression**. Il existe aussi des applications mathématiques telles que log, exp, cos et sin.

Log et exp : Capture les échelles relatives des données ou pour dénombrer des entiers (jamais négatifs). Exemple : Combien de fois un utilisateur à … np.bincount(X) de zéro.

Sin et cos : Intéressants pour des données qui encodent des signaux périodiques.

|  |  |
| --- | --- |
| Modèles arborescentes | Modèle linéaire et RN |
| Se soucient que **de l’ordre des caractéristiques**. | Très liés à l’**échelle** et à la **distribution** de chaque caractéristique, et par se fait de savoir s’il **existe une relation non linéaire entre la caractéristique et la cible**. |

La plupart des modèles **fonctionnent mieux lorsque chaque caractéristique** (aussi la cible dans le cas de la régression) **est distribuée de manière à peu près gaussienne**. En d’autres termes, un **histogramme de chaque caractéristique** devrait, dans le meilleur des mondes, donner une représentation ayant à peu près la forme d’une **courbe en cloche**.

Histogramme avec beaucoup de **petites valeurs et peu de grandes valeurs** => **distribution en poisson**, assez fondamentale dans le comptage de données. Appliquer une transformation logarithmique (approximation grossière de l’utilisation de la régression de Poisson) pour avoir une courbe en cloche (Si 0 Є X alors **LOG (X+1)**).

*Trouver la transformation qui fonctionne le mieux pour chaque combinaison de jeu de données et de modèle est tout un art.*

*Il peut arriver que chaque caractéristique ait besoin d’être transformer de manière différente*.

*Bonne idée de transformer la variable cible y dans une régression*.

Ces types de **transformations** ne sont pas **adaptés aux modèles arborescents**, ils sont souvent **capables de découvrir par eux-mêmes des interactions importantes**.

Sélection automatique des caractéristiques :

Il existe plusieurs façons de créer de nouvelles caractéristiques, nous sommes tentées *d’augmenter la dimensionnalité* des données => *modèle complexe => augmenter le risque de sur-apprentissage.*

*Dans le cas de grand nombre de dimensions, on réduit le nombre de caractéristiques pour ne conserver que les plus utiles et laisser les autres de côté => modèle simple qui généralise mieux.*

A quel point chaque caractéristique est bonne ?

Trois stratégies se présentent : les *statistiques uni-variées*, *sélection basée sur le modèle* et *la sélection* *itérative,* tous sont des méthodes supervisées (besoin de la cible pour l’ajustement du modèle).

Statistique uni-variées :

Cette méthode calcule s’il existe une relation **statistiquement signifiante** entre chaque **caractéristique et la cible**. Ensuite, les **caractéristiques** qui **sont liées avec le plus haut degré de confiance** sont **sélectionnées**. Dans le cas de la classification, on appelle cela l’analyse de variance (ANOVA-ANalyst Of VAriance). Uni-variée dans le sens qu’elle **considère uniquement chaque caractéristiq**ue de manière individuelle. Par conséquence, une **caractéristique est rejetée si** elle est seulement **informative lorsqu’elle est combinée avec une autre.**

|  |  |
| --- | --- |
| f\_classif (classification) | F\_regressor (régression) |

Via scikit lean => Utiliser la méthode

Caractéristique rejetée << p-values => peu susceptible d’être liée à la cible.

Calculer le seuil from sklearn.feature\_selection import :

* SelectKBest (sélectionne un nombre fixé k de caractéristiques).
* SlectPercentile (sélectionne un pourcentage fixé k de caractéristiques).

+Calcule est rapide.

+Ne nécessite pas la construction d’un modèle.

+Indépendant du modèle.

+/- **Supprimer le bruit améliore les performances, même si certaines des caractéristiques d’origine sont perdues**.

+Utile s’il y’a trop de caractéristiques pour construire un modèle.

+Utile si on suspecte que de nombreuse caractéristique **comporte des informations sont inutiles**.

sélection de caractéristiques basées sur le modèle :

Cette méthode utilise un modèle d’apprentissage automatique supervisé pour juger de l’importance de chaque caractéristique afin de ne conserver que les plus importantes.

Le modèle supervisé servant à cette sélection n’a pas besoin d’être le même que celui qui va servir à la modélisation finale (il doit fournir une certaine mesure de l’importance de chaque caractéristique afin de pouvoir effectuer un classement en fonction de cette mesure).

Les modèles arborescents ont un attribut feature\_importances\_ qui encode l’importance de chaque caractéristique.

Les modèles linéaires disposent de coefficients qui peuvent aussi capturer l’importance des caractéristiques en considérant les valeurs absolues.

* Une forme de sélection pour le modèle lui-même.
* Un prétraitement afin de choisir des caractéristiques pour un autre modèle.

La méthode considère toutes les caractéristiques de manière globale, contrairement à la sélection univariée, peut donc capturer les interactions (si le modèle en est capable).

From sklearn.feature\_selection import SelectFromModel

Si on ne sait pas quelle méthode utilisée, mieux vaut utiliser la sélection automatique de caractéristiques.

+Adaptée pour la réduction de dimensionnalité.

+Accélère les prédictions.

+Rend les modèles plus faciles à interpréter.

Selection iterative de caractéristiques :

La méthode consiste à construire une **série de** **modèles** avec un nombre variable de caractéristiques. Il existe deux façons : Commencer sans caractéristique puis les ajouter une par une jusqu’a ce qu’un certain critère d’arrêt soit atteint ou inversement.

-Couteuse en ressource.

L’algorithme RFE (Recursive Feature Elimination) débute avec toutes les caractéristiques, construit un modèle, et rejette la caractéristique la moins importante pour ce modèle. Un nouveau modèle est ensuite construit à partir des caractéristiques restantes et ainsi de suite …

Attention : Le modèle utilisé doit pouvoir déterminer l’importance des caractéristiques.

Une fois les caractéristiques sélectionnées les MA et ML obtiennent les mêmes scores de prédiction.

Dans le monde réel, ces méthodes apportent peu de gains de performances mais reste un bon outil pour l’ingénierie des caractéristiques.

Savoir utiliser l’expertise :

Une expertise métier est importante dans l’ingénierie des caractéristiques. Bien que le but de l’apprentissage automatique soit d’éviter d’avoir à créer un ensemble de règles conçues par des experts.

Exemple page 246 :

La prédiction effectuée par une foret aléatoire en utilisant uniquement le temps POSIX est une ligne droite (depuis la dernière heure pour laquelle les données ont été observées), car **la valeur temps POSIX pour le jeu de test se trouve en dehors de la plage des valeurs disponibles dans le jeu d’apprentissage.**

Une régression linéaire avec un encodage du jour, de la semaine et de m’heure donne de très mauvais résultats, du fait que l’encodage de ces derniers en entiers, qui sont interprétés comme des variables continues. On peut essayer de capturer ces informations en **interprétant les entiers sous forme de variables catégorielles** (***transformation OneHotEncoder***). => améliore le score !

On peut faire appel à des intersections **pour que le modèle apprenne un coefficient pour chaque combinaison jour/heure du jour** => cela améliore encore mieux ~ modèle FA

+ Modèle FA donne de bon résultat mais ML nous permet de tracer les coefficients.

Les modèles linéaires bénéficient de ka génération de nouvelles caractéristiques via la discrétisation ou l’ajout de polynômes et d’intersections. Mais ce n’est pas le cas pour les modèles plus complexes - - **SVM et FA**.

# Chapitre 5 : Evaluation et amélioration du modèle :

## Cross validation

Il existe plusieurs variantes de la VC dans Sklearn :

1. Cross validation standard

Utiliser pour la régression

Une image contenant texte

Description générée automatiquement

1. Cross validation stratifiée

Utiliser pour la classification

Divise les données de manière que la proportion entre les classes soit la même dans chaque pli que dans le jeu de données tout entier.

Example : si on a 90% de la classe A et 10% de la classe B, alors chaque pli comportera cette distribution.

1. Cross validation Leave-one-out

Un seul point de donnée pour construire le jeu de test.

Bonne pour les données de petite taille.



1. Cross validation non stratifié

Une image contenant texte, intérieur, capture d’écran

Description générée automatiquement

1. Shuffle Split

On peut ne pas couvrir la totalité de l’échantillon => expérimentation sur de grand jeu de donnée.

Classification => StratifiedShuffleSplit

1. Staritified Shuffle Split
2. Validation croisée avec groupes

Les données offrent d’importantes similarités.

Un groupe contient les données qui ne devraient pas être séparées lors de la création des jeux de d’apprentissage et test.

Example :

* + Toutes les expressions faciales d’une même personne
  + Echantillons du même patient

Une image contenant texte

Description générée automatiquement

## Recherche sur grille (à la recherche des meilleurs paramètres):

La méthode grid search fait les combinaisons des paramètres intéressants. On prend l’exemple d’un SVM a noyau gaussien (C : régularisation, Gamma : Détermine la portée de l’influence d’un certain échantillon de donnée). Une combinaison de 6 valeurs pour C X 6 valeurs pour Gamma = 36 combinaisons.

### Recherche simple sur grille => sous forme de boucle for

Cette proclamation pourrait être excessivement optimiste (ou même fausse) car on a utilisé les données de test pour ajuster es paramètres, nous ne pouvons plus nous en servir pour évaluer la qualité de notre modèle. Donc on a besoin d’un nouveau jeu de données indépendant pour réaliser une évaluation correcte, autrement dit un jeu qui n’a pas été utilisé pour crée le modèle.

|  |  |  |
| --- | --- | --- |
| Training set | Validation set | Test set |

Modèle fiting Paramètre sélection Evaluation

Avec cette solution :

* Couramment utilisée.
* On obtient des résultats un peu bas que précédemment probablement par ce qu’on a utilisé moins de données pour l’apprentissage.
* Est assez sensible à la précision avec le partage des données.

### Recherche sur grille avec validation croisée :

On peut faire appel à la validation croisée pour afin d’évaluer les performances de chaque combinaison de paramètres.

-Méthode très lente pour l’entrainement de tous les modèles.

+Couramment utiliser => Scikit-learn fournit une classe GRIDSERACHCV qui l’implémente sous forme d’estimateur (*un estimateur qui est créer en utilisant un autre estimateur est dit méta-estimator*).

L’objet Grid ne se contente pas de recherche les meilleurs paramètres. Il produit de surcroit automatiquement un nouveau modèle basé sur l’ensemble du jeu d’apprentissage pour les paramètres qui ont fourni les meilleures performances de validation croisée.

|  |
| --- |
| Exemple :  From sklearn.model\_selection import GridSearchCV  From sklearn.svm import SVC  **Param\_grid = {‘C’: [0.001, 0.01, 0.1, 1, 10, 100 ], ‘gamma’ : [0.001, 0.01, 0.1, 1, 10, 100]}**  Grid\_search = GridSearchCV( SVC(), param\_grid, cv=5) % 5 plis  Scores = cross\_val\_score(logres,X,y, cv = Shuffle\_split)  X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, random\_state = 0)  Grid\_search.fit( X\_train, y\_train)  Grid\_search.score( X\_test, y\_test)  Print(“Best param : {} “.format(Grid\_search.best\_params\_)) % C: 100, gamma : 0.01 |

* **Analyser le résultat de la validation croisée :**

Du fait que la recherche sur grille avec validation croisée soit couteuse, une idée consiste à tester avec une petite grille, puis de voir s’il est utile d’étendre nos recherches.

L’attribut grid\_serach.cv\_results\_ donne toutes informations sur les ocmbinaisons.

Affiner la grille de paramètre en se basant sur les scores de la validation croisée est une excellente démarche.

* **Effectuer des recherches sur des espaces qui ne sont pas des grilles :**

Dans certains cas, essayer toutes les combinaisons possibles de tpis le sparamètres (CridSearchCv) n’est pas une bonne idée. Par exemple, SVC a un paramètre appelé kernel et sa valeur influe sur les autres paramètres… exemple :

**Param\_grid = [ {‘kernel ‘ : [’rbf’],‘C’: [0.001, 0.01, 0.1, 1, 10, 100 ], ‘gamma’ : [0.001, 0.01, 0.1, 1, 10, 100]}, {‘kernel ‘ : [’linear’],‘C’: [0.001, 0.01, 0.1, 1, 10, 100 ] }]**

* **Utiliser différentes stratégies de validation croisée avec la recherche sur grille :**

GridSearch utilise par défaut :

Une validation croisée stratifiée à k-plis la classification.

Une validation croisée à k-plis la régression.

On peut modifier le paramètre CV …

Avec une seule partition des données en un jeu d’apprentissage et un jeu de test, on peut rendre nos résultats instables et dépendants de cette division simple des données.

Aller plus loin avec **une nouvelle**

**Séparation des données** en utilisant GridSearch

Au sein d’autres validations croisées

Validation Croisée Imbriquée

Exemple :

**Param\_grid = {‘C’: [0.001, 0.01, 0.1, 1, 10, 100], ‘gamma’: [0.001, 0.01, 0.1, 1, 10, 100]}**

Score = cross\_val\_score (GridSearchCV(SVC(), param\_grid, cv=5),X ,y , cv = 5)

Boucle imbriquée

|  |
| --- |
| * Le résultat est une liste de score et non un modèle ou une configuration de paramètre. * Les scores nous informent sur la capacité de généralisation d’un modèle. * Cette méthode n’est pas utilisée pour chercher un modèle prédictif pour être appliqué à de futurs échantillons. * Utile pour évaluer la manière dont un modèle fonctionne sur un jeu de donnée particulier. * Extrement lourde en termes de traitements. |

Parallélisation avec n\_jobs et le nombre de cœurs de CPU

N\_jobs = -1 => on utilise tous les cœurs.

Pour connaitre

Matrice de confusion

Cibler l’importance du Rappel/précision

Changer le seuil

Métrique de l’AUC

Courbe ROC

Courbe précision-rappel

## **Métrique d’évaluation :**

### **Classification binaire:**

Classe positive vs négatif

### Types d’erreurs

***Faux positif/erreur type 1*** *:*

*Un patient en bonne santé, classifié positif => analyses supplémentaires, coût inutile, désordre mentale…*

***Faux négatif/erreur type 2*** *:*

*Un patient malade, classifié incorrecte => pas de traitements, mort...*

Example :

Diagnostique cancéreux, un faux négatif est plus grave qu’un faux positif.

### Jeux de données déséquilibrés :

Mauvais équilibre des deux classes (exemple : les clics des utilisateurs), avec ce problème on peut obtenir une prédiction de 99% (sans construire de modèle d’Apprentissage Automatique) car la prédiction ne concerne une seule classe. On peut s’en rendre compte en appliquant un autre modèle et les scores sont différents.

Il faut une métrique pour évaluer nos modèles, une métrique capable d’éliminer ces prédictions absurdes.

Matrice de confusion

Cette méthode est très utilisée pour la classification binaire, elle consiste à utiliser confusion\_matrix.

### Matrice de confusion

Une image contenant table

Description générée automatiquement

Comment résumer la matrice de confusion ?

#### Exactitude :

EXACTITUDE =

* Le rapport entre le nombre de prédictions correctes et le nombre total d’échantillons traités.
* **Efficace pour la classification de données équilibrées**
* Facile à calculer et à comprendre
* Ne prend **pas** compte la **conséquence des erreurs (type d’erreur)**

#### Précision :

PRECISION =

Mesure **la proportion des échantillons prédits comme étant positifs et qui le sont effectivement**.

Cette une métrique de **performance lorsque le but est de limiter le nombre de faux positifs**.

Exemple : une entreprise veut lancer un produit, les matières premières coûtent cher. Elle veut être que le produit marchera pour débuter la production.

#### Rappel :

RAPPEL =

Identifier tous les échantillons positifs, autrement dit, **éviter les faux négatifs**.

Example :

Diagnostique cancéreux, un faux négatif est plus grave qu’un faux positif.

**Il existe un compromis entre optimiser la Précision et optimiser le Rappel.**

#### F-Mesure :

* La moyenne harmonique de la précision et du rappel
* Meilleur que les deux séparés
* Classification de données déséquilibrées
* Difficile à interpréter et à expliquer que l’exactitude

**From skearn.metrics import classification\_report**

**classifcation\_report(y, modèle,target\_names = [‘1’,’0’])**

Prendre en compte l’incertitude :

Si les données sont trop déséquilibrées, changer le seuil de décision « decision\_function » ou « predict\_proba » pour améliorer les résultats facilement.

Seuil >> taux de certitude >> pour prendre une décision positive => il a besoin moins de certitude pour prendre une décision négative.

Un modèle recalibré est un modèle

Qui fournit une mesure précise de son incertitude

Trouver le seuil, avec toutes les combinaisons possibles

Métrique de l’AUC

Courbe ROC

Courbe précision-rappel

Pour le training,

Sous échantillon, sur augmentation (créer de nouvelles données, dupliquer les points), change la fonction objectif (pondérer la petite classe ⬄ sur augmentation intélligente sans bouffer de mémoire)

## Courbe précision-rappel, courbes ROC et AUC :

1. Courbe précision-rappel :

Poser une exigence sur un classifier, comme un rappel à 90%, est appelé définir le point de fonctionnement (operating point). Exemple : apporter des garanties de performances à des clients … Comme ce point n’est clair, il serait instructif de regarder tous les seuils possibles, ou tous les compromis possibles entre précision et rappel d’un seul coup. Cela est réalisable en faisant appel à ***la courbe précision-rappel.***

***From sklearn.metrics import precision\_recall\_curve***

Cette fonction retourne une liste de valeurs de précision et de rappel pour tous les seuils possibles, selon un ordre de tri permettant de définir les axes de la courbe (page 293). Plus on se rapproche du coin supérieur droit, meilleur est le classifier, ce point possède une haute précision et un haut rappel pour le même seuil.

Point idéal

1. Courbe ROC:

Un outil également utilisé pour analyser le comportement des classifieurs en fonction de différents seuils. **Au lieu de fournit un lien entre Rappel et Précision, elle montre le taux de faux positifs par rapport au taux de vrais positifs (Rappel)**. En utilisant la fonction **roc\_curve de de sklearn.metrics.**

Le point idéal

1. Métrique de l’AUC (l’air sous la courbe) :

Vu qu’on a affaire à des courbes, il serait utile de condenser la courbe ROC en utilisant un seul nombre unique, cette métrique est bien meilleure que l’exactitude dans le problème de classification de données déséquilibrées. Il peut être interpréter comme une évaluation du ranking (ordonnancement) des échantillons positifs. Un AUC parfait à 1 signifie que tous les points positifs ont un score plus élevé que tous les points négatifs. Dans le cas de problème de classification avec des classes déséquilibrées, utiliser AUC pour la sélection du modèle est souvent nettement plus significatif que se baser sur l’exactitude.

+ Vivement utiliser AUC pour la classification de données déséquilibrées.

-Ne possède pas de valeur par défaut.

1. Métrique pour la classification multi-classe :

Toutes les métriques pour la classification multi-classe sont dérivées des métriques pour la classification binaire mais moyennées sur toutes les classes. L’exactitude pour cette classification est également définie en tant que fraction des échantillons correctement classifiés. Et, là encore, lorsque les classes sont déséquilibres, l’exactitude n’est pas une bonne mesure d’évaluation.

On utilisera

* + - * La **matrice de confusion** puis **Classification\_report**
      * F-mesure

L’idée consiste à calculer une valeur f-mesure binaire par classe, celle-ci devenant la classe positive, les autres formant les classes négatives. Ces scores sont ensuite « moyennés » en utilisant les stratégies :

* Macro : effectue-le calcule de la moyenne pour l’ensemble des classes.
  + Utile si on veut que toutes les classes soient considérées à l’identique.
* Pondéré : calcule la moyenne des f-mesures pours/ toutes les classes, pondérée par leur support.
* Micro : : effectue-le calcule de la moyenne pour le total de faux positifs, de faux négatifs et de vrais positifs pour l’ensemble des classes, puis recalcule précision, rappel et f-mesure en utilisant ces résultats.
  + Utile si on veut que chaque échantillon soit traité à l’identique.

1. METRIQUE DE régression :

* Le coefficient de détermination R\*\*2 est une métrique plus intuitive pour évaluer ces modèles.
* Erreur quadratique moyenne.
* Erreur absolue moyenne.
* Utiliser des métriques d’évaluation dans la sélection de modèles :

Sckit-learn permet d’utiliser la métrique AUC pour la sélection de modèle en utilisant **GridSearch ou cross\_val\_score** via l’argument **scoring** AUC.

|  |
| --- |
| Exemple : 9 vs le reste page 304  Cross\_val\_score(modèle, X, y==9, scoring = « roc\_auc »)  Grid = GridSearch(modèle, param\_grid = my\_params, scoring = « roc\_auc »)  Grid.fit(X\_train, y\_train) |

Les valeurs importantes du paramètres scoring pour la classification sont accuracy (par défaut), roc\_auc pour l’aire sous la courbe ROC, average\_precision pour l’air sous la courbe précision-rappel, ainsi que f1, f1\_macro, f1\_micro et f1\_weighted pour les scores f-mesure binaires.

# Chapitre 6 : CHAINAGE D’ALGOTITHME ET PIPLINE

* Erreur Fatale :

|  |
| --- |
| On recalibre **toutes** les **données du jeu d’apprentissage** puis on calcule le minimum et le maximum des données recalibrées  *Grid = GridSearchCV( SVC(), param\_grid= param\_grid, cv = 5)*  *Grid.fit(X\_train, y\_train)*  **On se sert de ces données recalibrées** pour **lancer** notre **recherche sur grille** en utilisant une validation croisée  *Print(Grid.best\_score) // .98*  *Print(Grid.best\_param)// C=1, gamma =1*  *Print(Grid.score(X\_test, y\_test)// .97*  Pour chaque division, une partie servira à un jeu de test (mesure les performances d’un modèle entrainé à partir de la partie apprentissage) **mais nous avons déjà utilisé les informations contenues dans la partie Test** de la division **lors du recalibrage des données**. |

Pour éviter le problème, la division du jeu de données

Pour obtenir ce résultat dans scikit-learn avec les fonctions cross\_val\_score et GridSearchCV

Au cours de la validation croisée devrait être

Effectuée avant tout traitement.

Tout processus **qui extrait des connaissances à partir du jeu de donnée**

Devrait être entrainé uniquement depuis **la partie apprentissage**

De celui-ci, *et donc être contenu dans la boucle de validation croisée.*

La classe pipeline n’est pas restreint au prétraitement

Et à la classification. Elle peut relier un nombre quelconque d’estimateur.

La seule condition, c’est que toutes les étapes sauf a dernière possèdent une méthode transform.

Exemple : T1 – T2 – Classifier

Y

T1.fit(X,y)

X

T1

y

T2

T2.fit(X1, y)

T1.transform(X)

X1 y

Classifier.fit(X2)

T2.transform(X1)

X2

Classifier

1. Construire un pip/Make\_pipeline:

|  |
| --- |
| From sklearn.pipeline import Pipeline  Le fit fait appel à la première étape (Scaler) afin de transformer les données d’apprentissage en utilisant MinMaxScaler  Pipe = Pipeline( [(‘scaler”, MiniMaxScaler() ), (‘svm’, SVC() ) ] )  Pipe.fit(X\_train, y\_train)  Pipe.score(X\_test, y\_test)  Score effectue la transformation des données de test avec scaler, puis passent les données à la méthode score du SVM. |

1. Utilise pipe dans des recherches sur grille:

|  |
| --- |
| param\_grid = {‘**svm\_\_C’**: [0.001, 0.01 ,0.1, 1, 10, 100], ‘**svm\_\_gamma’**: [0.001, 0.01, 0.1, 1, 10, 100]}  Nom De L’Etape suivit d’un double trait puis le nom du paramètre  Grid = GridSearchCV(pipe, param\_grid=param\_grid, cv=5)  MinMaxScaler est réajusté uniquement à partir des données d’apprentissage.Aucune information ne fuite du jeu de test.  grid.fit(X\_train, y\_train)  print(“Best coss\_validation accuracy: {}”.format(grid.best\_score\_))  print(“Test set score: {}”.format(grid. score(X\_test, y\_test)  print(“Best cparameters: {}”.format(grid.best\_params\_)) |

#### **example:**

Rien

param\_grid = [

**{**‘**classifieur’** : [ SVC() ], ‘preprocessing’ : [StandardScaler(), None],

OU

‘**classifieur\_\_C’**: [0.001, 0.01, 0.1, 1, 10, 100], ‘**classifieur\_\_Gamma’**: [0.001, 0.01, 0.1, 1, 10, 100] **}**

**{**‘classifieur’ : [RandomForestClassifieur (n\_estimators=100), ‘preprocessing’ : [StandardScaler(), None], ‘classifieur\_\_max\_features’ :[1, 2, 3] **}**

]

1. *Example page 312:*

|  |
| --- |
| Les données sont crée de manière aléatoire, donc aucune relation entre X et la cible y => impossible d’apprendre quoi que ce soit du jeu de donnée. %100X1000  Select = selectPercentile(score\_func= f\_regression, percentile = 5 ).fit(X,y)  Le score est clairement **faux***, puisque nos données sont aléatoires.* ***Co*mme la sélection a été faite en dehors de la validation croisée**, **elle a pu trouver des caractéristiques qui sont en corrélation à la fois avec les plis d’apprentissage et ceux de test**.  X\_selected = select.transform(X) % 100X500  Np.mean(cross\_val\_score(Ridge(), X\_seleccted, y, cv=5)% .91  **Les éléments qui ont fuités à partir des plis de test**  **se sont donc révélés très informatifs**, conduisant ainsi à des résultats hautement irréalistes. |

* *Correction:*

|  |
| --- |
| Pipe = pipe( [(‘select’, SelectPercentile(  Le pipe place la phase de **sélection à l’intérieur** de la validation croisée=> **seuls les plis d’apprentissage peuvent servir à la sélection**, pas ceux du t est. Impossible de trouver des caractéristiques qui sont corrélées avec la cible sur le jeu d’apprentissage.  score\_func = f\_regression,  percentile = 5 ), (“ridge”,Ridge())])  Np.mean(cross\_val\_score(pipe, X, y,cv=5))  % -.25  **Rectifier le problème de fuite de donnée fait toute la différence entre conclure qu’un modèle fonctionne très bien et conclure qu’il ne fonctionne pas du tout** |

1. Avantage :

+ Encapsulation de toutes les étapes de prétraitement en un seul estimateur.

+ Ajuster les paramètres du prétraitement en utilisant la sortie d’une tache supervisée, comme une régression ou une classification.

+ Outil généraliste servant à chainer plusieurs étapes de traitement dans un flux d’apprentissage automatique.

+ Réduit les erreurs.

## Estimateur bais / variance

Estimateur ponctuelle (statistique) / teta chapeau / approximation (terme non approprié) != paramètres teta réel.

On s'intéresse à son bais et sa variance

1. Le biais:

Il fournit une valeur proche du paramètre, **qui dépend des données, les données sont tirées aléatoirement IID, donc teta une VA**).

Biais(teta chapeau) = E(teta chapeau) - teta réel, défini sur les données et la vraie valeur du paramètre définissant la distribution génératrice des données.

Un estimateur est dit sans biais si Biais (teta chapeau) = 0

Un estimateur est dit asymptotiquement sans biais si limite Biais(teta chapeau) = 0

1. Variance : Voir comment il varie en fonction de l'échantillonnage pour quantifier le degré de variation prévu pour l’estimateur.

Le biais et la variance mesurent deux erreurs différentes d’un estimateur.

**Le bais mesure l’écart prévu par rapport à la valeur réelle de la fonction ou du paramètre.**

**La variance, fournit une mesure de l’écart par rapport à la valeur en espérance possiblement fournie par un quelconque rééchantillonnage des données.**

Compromis ?

Dans un cadre applicatif, la validation croisée est très efficace.

La relation entre biais et variance est liée avec le sur / sous apprentissage.

Généralisation : capacité d’avoir de bonne performance sur des entrées non observées précédemment

# Chapitre 7 Données textuelles

Il existe deux types de caractéristiques :

Corpus : jeu de donnée

Document : chaque point de donnée.

Donnée catégorielle

Texte

Réponse possible obtenues à partir du contenu d’un champ de texte. Pouvant être associée d’un point de vue sémantique à des données catégorielles.

Chaine libre

Attention aux fautes d’orthographe et aux doublant !!

Text\_train = [doc.replace(« … »,  « … ») for doc in text\_train]

Convertir le format en une représentation

Numérique afin d’appliquer un algorithme

1. Représenter les données textuelles sous forme de sacs de mots (bag of word)

Méthode simple et courante, servant à transformer les textes afin de leur donner un format reconnaissable pour l’apprentissage automatique. On **élimine l’essentiel de la structure du texte** présenté en entrée. On **compte le nombre d’apparitions de chaque mot dans chaque texte du corpus.**

*Etape 1 :* ***Tokenization***

Partager un document dans les mots qui le composent => un texte devient une séquence de *tokens*.

*Etape 2 :* ***Construction du vocabulaire***

**Construire un dictionnaire de tous les mots qui apparaissent dans les documents** et les numéroter (exemple dans l’ordre alphabétique).

*Etape 3 :* ***Encodage***

Dénombrer le nombre d’occurrences de chaque mot du vocabulaire.

Exemple :

« This is how you get ants »

Words = [‘The fool doth think he is wise’, ‘but the wise man knows himself to be a fool’]

From sklearn.feature\_extraction.text import CountVectorizer

Vect = CountVectorizer()

Tokenization et construction du vocabulaire

Vect.fit(words)

Vect.vocabulary\_ % {‘the’: 9, ‘fool’: 3... ‘be’: 0}

* On utilise Toarray, car words est de petite taille.
* Le vocabulaire est par ordre alphabétique.

Words.toarray() % [[0 0 1 1 1 0 1 0 0 1 1 0 1]

[1 1 0 1 0 1 0 1 1 1 0 1 1]]

1. Éliminer les mots sans importances : Pour améliorer les performances du modèle
2. Augmenter le nombre d’apparition :

Vect = CountVectorizer(min\_df /max\_df= 5).fit(text\_train)

X\_train = vect.transform(text\_train)

1. Mots vides (stop words) :

Supprimer les mots très courants qui n’apportent aucune information utile (par exemple les articles définis).

* 1. Utiliser une liste de mots vides dans une langue particulière.

(sklearn.feature\_extraction.text import ENGLISH\_STOP\_WORDS).

* 1. Supprimer les mots très fréquents.

CountVectorizer(min\_df=5, stop\_words =”English”).fit(text\_train)

1. Pondérer les données avec tf-idf ( term frequency inverse document frequency):

Donner un poids élevé aux termes qui apparaissent souvent un document, mais pas dans de nombreux documents du corpus (très représentatif dans un certain document).

* 1. TfidfTransformer

Score Tf-idf (w, d) =tf\*log+1

N le nombre de document

Le nombre de document dans le quel le mot w apparait.

Tf Le nombre d’occurrences du mot w dans le document considéré d

* 1. TfidfVectorize

Cette norme euclidienne implique que la longueur d’un document, c.-à-d. le nombre de mot qui le composent, ne change pas la représentation vectorisée.

### Des sacs avec plusieurs mots (n-grammes) :

L’ordre est totalement laissé de côté. Exemple « c’est mal, pas bien du tout » et « c’est bien, pas mal du tout ».

Il existe plusieurs manières de capturer ce contexte dans une représentation par sac de mots :

Méthode 1 : Extraction des caractéristiques countVectorizer et TfidfVectorizer + à prendre en compte non seulement la fréquence d’un certain *token*, mais aussi celle de paires (bigrammes) ou de triplets (trigrammes) ou n-grammes réglable avec le paramètre

|  |  |
| --- | --- |
| Réglable avec le paramètre ngram\_range(min\_token, max\_token) | |
| countVectorizer | TfidfVectorizer |

Utilise des séquences plus longues fournit généralement bien davantage de caractéristiques, et notamment des caractéristiques plus spécifiques => pas de bi-grammes communes entre les deux phrases (via toarray()) + Conduit une explosion du nombre de caractéristiques et entrainerait un sur-apprentissage, puisqu’il y’a plusieurs caractéristiques spécifiques.

En principe :

Nombre de bigrammes =

Nombre de trigrammes = …

+/-Méthode simple

Méthode2 : Tokenisation avancé, racinisation et lemmatisation

Mot + ing, verbe, pluriel, singulier … le mot a plusieurs formes et les distingué ne ferait qu’accroitre le su-apprentissage

Solution

|  |  |  |
| --- | --- | --- |
| Des formes de normalisation qui essaient d’extraire une certaine forme normale à partir d’un mot. | | |
| Racinisation (stemmer) | Lemmatisation | Correction orthographique |
| Représenter le mot à partir de la racine afin de repérer les éléments d’une même famille (méthode heuristique) | Utiliser un dictionnaire de formes connues (un système explicite et non automatisé) pour déterminer le rôle du mot dans la phrase |  |
| -Limiter à l’ajustement du mot à sa racine.  -Normalise mal les mots.  -Réduit les occurrences (meeting = meet) | +Trouver la forme verbale de base correcte.  +Arriver à normaliser les mots.  +Arriver à faire ma différence entre meeting et meet.  +Complexe.  +Produit de meilleurs résultats.  +Peut être perçue comme une forme de régularisation (elle associe en quelque sorte certaine caractéristique). | |

Méthode 3 : l’allocation de Dirichlet latente (LDA)

Essaie de trouver des groupes de mots (les sujets, ou topics).

Méthode 4 : RNN

# **Chapitre 8 : Pour conclure**

# Aborder un problème d’apprentissage automatique :

1. Se poser les bonnes questions :

Il faut réfléchir aux types de questions auxquelles vous voulez répondre.

* Avez-vous besoin d’une démarche exploratoire ou de trouver quelque chose d’intéressant dans les données ?
* Avez-vous déjà un objectif particulier présent à l’esprit ?

Vous devriez réfléchir en premier lieu non pas à construire un système, mais à comment définir et mesurer votre réussite, et à songe à l’impact qu’une solution efficace et réussie aurait sur votre activité ou vos objectifs de recherche.

1. définir le problème à résoudre
2. avoir les bonnes données et les mettre en forme
3. mesurer la réussite et l’impact de la solution

Mesurer les performances de l’algorithme via une métrique « business », comme l’augmentation du profit ou la réduction des pertes.

1. batir un prototype fonctionnel (choix du modèle)
   1. analyser ces modèles

Analyser les erreurs faites par un modèle fournit de bonne indication sur ce qui manque dans les données, sur quelles données supplémentaires il faudrait collecter, ou encore sur la manière dont la tache devrait être reformulée afin de rendre le processus d’apprentissage automatique plus efficace

Meilleur que d’exécuter à l’infini des recherches sur grille afin d’affiner les paramètres.

Exemple détection de fraude :

* Comment puis-je mesurer la validité de ma prédiction de fraudes ?
* Est-ce que je dispose des bonnes données pour les faire évaluer par l’algorithme ?
* Si je réussis, quel sera l’impact économique de ma solution ?

# **tester des système de production**

Le Test A/B et l’algorithme du bandit manchot.

Utiliser un modèle probabiliste puis utiliser l’apprentissage automatique ou l’inférence probabiliste pour **déterminer le niveau de confiance à accorder à chaque mesure**, et donc raisonner sur la détermination de la meilleure **estimation possible quant à la position de l’utilisateur** (exemple GPS et prédiction de la prochaine positon) Utiliser un **langage de programmation probabiliste** tel que PyMc et STAN

#### **Apprentissage automatique**

Système dans le quel nous voulons retrouver des réponses à une certaine requête en les classant selon leur pertinence.

Exemple :

* Fonctionnement des moteurs de recherche.

Système qui fournit des suggestions aux utilisateurs en fonction de leurs préférences.

Exemple :

* Facebook, Google+ (amis potentiels qui vous sont aimablement suggérés).
* Amazon et sites marchands (Les clients ayant acheté cet article ont également acheté …).
* La prédiction d’évolution dans le temps (cours de la bourse).

Erreur de généralisation

Peut s’exprimer comme la somme de trois erreurs très différentes :

* 1. BIAIS : Mauvaise hypothèses (Exemple :

- supposer que les données sont linéaires alors qu’elles sont quadratiques

Avoir plus de photos d’homme blanc que de d’homme noir (FB).

Si erreur Test/train >> au fur et à mesure + score minable => Biais >>>

* 1. Variance :

Un modèle avec beaucoup de degrés de liberté (nombre de valeurs aléatoires qui ne peuvent etre déterminées ou fixés par une équation, X + Y = 12).

* 1. Erreur irréductible : Due au bruit présent dans les données => nettoyer les données.

|  |  |
| --- | --- |
| La fonction de cout Vs la fonction de performance | |
| Elles sont différentes | |
| La fonction de cout peut etre régularisé | La fonction de performance n’est jamais régularisée |
| La fonction de cout doit avoir des dérivées permettant une bonne optimisation. | La fonction de performance doit vérifier si l’estimation est proche de l’objectif final. |

|  |  |  |
| --- | --- | --- |
| Exactitude (le taux de classification correctes). | Précision (taux d’observation classées comme positives et qui le sont effectivement). | Rappel (le taux d’observations positives qui sont bien détéctées). |

### Arrêt précoce

Quand l’erreur de validation cesse de croitre et commence à augmenter à nouveau (on commence à sur-ajuster). Avec l’arrêt précoce, on interrompt l’entrainement lorsque l’erreur de validation atteint le minimum.

# Méthodes de régularisation

L’objectif principal de la régularisation est de contrôler :

* Le surapprentissage dans l’apprentissage automatique
* L’erreur de type variance

La régularisation s’applique uniquement de l’entraînement du modèle.

## Régularisation L1 et L2 :

|  |  |
| --- | --- |
| L1 | L2 |
| La somme des poids en valeur absolue multiplié par une constante α | La somme des poids au carré multiplié par une constante α |
| Formules | |
| L1 essaie d'estimer la médiane des données. | L2 essaie d'estimer la moyenne des données. |
| Intuition | |
| Si on a beaucoup de features, on peut penser que certaines sont inutiles.  Contraindre les coefficients à se rapprocher de zéro (qu’ils sont totalement ignorés par le modèle). Cela permet aussi de mettre en évidence les caractéristiques les plus importantes.  Permet d’effectuer une feature selection automatique (trouver des sous-ensembles de données pertinents).  Facile à interpréter du fait qu’elle sélectionne un sous ensemble de caractéristique. | La méthode pénalise le carré de la norme L2, ou encore la longueur euclidienne de W.  En pratique, elle est la plus utilisée.  L2 a tendance à réduire les coefficients de façon régulière.  Utile lorsque vous avez des caractéristiques colinéaires/co-dépendantes.  Sensible aux valeurs aberrantes. |
| Commun | |
| La régularisation impose une contrainte sur les poids du modèle. Une sorte de compromis entre la recherche d’un bon ajustement et le maintien d’une valeur basse pour certains poids lorsque les exposants sur les caractéristiques augmentent.  Les petits poids conduisent à des hypothèses plus simples, qui s’avèrent plus généralisables.  A mesure que le jeu d’entrainement augmente, l’effet de la régularisation diminue. | |

## Early stopping :

Interrompre l’entraînement lorsque ses performances sur le jeu de validation commencent à baisser.

Il est recommandé d’associer l’arrêt prématuré avec d’autre technique de régularisation.

## Dropout :

Chaque neurone a une probabilité p (taux d’extinction) d’être temporairement éteint (ignoré au cours de cette étape d’entrainement).

A chaque étape d’entrainement génère un réseau de neurones uniques, soit (où N est le nombre total de neurone), non indépendant car ils partagent leurs poids.

## Régularisation max-norm :

## Batch normalisation :

## Augmentation des données :

Utiliser des instances d’entraînement existantes pour en générer de nouvelles (réalistes), grossissant ainsi artificiellement la taille du jeu de données.

L’ajout du bruit blanc ne contribue pas à l’apprentissage (donc non).

Décaler, pivoter, redimensionner, modifier le contraste, ce qui rend le modèle plus robuste à la position, l’orientation, et la taille des instances, l’éclairage.

Si les instances sont symétriques, on peut retourner horizontalement les images.

# Fonctions d’activations

* Transformer non linéairement la combinaison formée par les entrées
* Imposer un intervalle aux données [0-1] ou [-1, 1]

# Fonctions de perte

Qualifie jusqu’à quel point un modèle est proche de l’idéal.

Une métrique qui calcule l’erreur que fait notre modèle, on peut lui ajouter une régularisation.

La recherche de cet idéal équivaut à trouver les paramètres (poids et biais) qui minimisent l’erreur.

Les paramètres peuvent être déterminés de façon analytique ou par des algorithmes d’optimisation itératifs.

### Fonctions de perte pour la régression

#### Erreur quadratique moyenne (MSE).

Fonction convexe, mais lorsqu’il s’agit de considérer multiples couches cachées, la notion de convexité ne tient plus.

Optimiser la MSE est équivalent à celle de la moyenne.

Sensible aux valeurs aberrantes.

Erreur absolue moyenne

Calcule la moyenne de l’erreur absolue sur l’ensemble du jeu.

#### Mean Squared Log Error (MSLE)

#### Erreur absolue moyenne en pourcentage (MAPE)

Remarque :

Les fonctions MSLE et MAPE méritent d’être prise en compte si l’intervalle des sorties varient fortement. Mais, en règle générale, on normalise les données d’entrées à l’intérieur d’un intervalle approprié puis on utilise MSE ou MAE.

### Fonctions de perte pour la classification

#### Perte hinge :

Classification binaire stricte, classifieur {0, 1} ou classifieur{-1, 1}.

Utilisée chez les SVMs.

Il existe des variantes pour la classification multi-classe : One-vs-All, One-vs-One.

Fonction convexe.

#### Perte logistique – Cross entropie - Log Loss :

N’est pas une classification stricte mais s’intéresse à la probabilité.

La dernière couche doit être une softmax, car contrairement à la sigmoïde, elle modélise les dépendances entre valeurs de sorties.

Optimiser le maximiser de vraisemblance

#### Divergence Kullback-Leibler :

La divergence de Kullback-Leibler est une mesure de dissimilarité entre deux distributions de probabilités

La classification multiclasse avec la fonction Softmax

## le word embedding. Le word embedding (plongement de mots) désigne un ensemble de méthode d’apprentissage visant à représenter les mots d’un texte par des vecteurs de nombres réels.

Problématique :

Un algorithme a été développé pour prédire le prix des maisons en real-time En novembre 2021, Zillow ferme le service real time car l'algorithme a suréstimé le prix de chaque propriété, engendrant de grosses pertes pour l'entreprise Si vous étiez chef de projet / advisor data scientist, qu'elles sont les recommendations que vous auriez faites pour vous assurer de la bonne mise en production du modèle.

Real time, mise à jour du modèle, continuel learning (not real time plus pour des fenetres temporelles)

Interpreter la decision du modèle : lime, shap (modification sur l’entrée pour voir comment impacter le score, plein de test) –> mini base de donnée, regression linéaire, analyser les poids w, pour connaitre les features importantes.

Confiance du modèle,

Pour un exemple donnée

* Feature seclection importante

Cas de l’image: Grad cam, bclasse activation map

Avant le soft, reponderer la feature

Phrase, a titre indicadif,

2019 – 2021: y a une pandémie 🡪 données ont pu change <!> gros drift temporal

Tester le modèle après les périodes 2008, robuste à ça

Feature selection

Se choisir un signal corrélé avec l’évolution du produit vendu

Distribution gaussienne -> tout suit une LG

Mse fonctionne en supponsant que les données suivent une loi gaussienne

Bernouilie pour le cas binaire

Loi poisson:

Quand on utilise la loi de Poisson ?

**Loi de Poisson**. La **loi de Poisson** est une **loi** de probabilité discrète. ... Exemple d'utilisation : Si un événement se produit en moyenne N fois par seconde, pour étudier le nombre d'événements se produisant pendant 60 secondes, on choisit une **loi de Poisson** de paramètre λ = 60xN.

Mais depuis quelques décennies son champ d'application s'est considérablement élargi. Actuellement, on l'utilise beaucoup dans les [télécommunications](https://fr.wikipedia.org/wiki/T%C3%A9l%C3%A9communication) (pour compter le nombre de communications dans un intervalle de temps donné