Université de Montpellier **HMM308**
2018/2019 **Alexandre Gramfort, Joseph Salmon, Nicolas Verzelen**

TP n° 1 : k-plus proches voisins

Les fichiers `tp_knn_source.py` et `tp_knn_script.py` sont disponibles sur le site pédagogique du cours. Ils contiennent le code et les fonctions utiles pour la partie sur les k-plus proches voisins.

- Découverte de Python -

Consulter les pages suivantes pour démarrer ou bien trouver quelques rappels de Python :

⋆⋆⋆ https://github.com/agramfort/liesse_telecom_paristech_python/blob/master/1-Intro-Python.ipynb

⋆⋆⋆ https://github.com/agramfort/liesse_telecom_paristech_python/blob/master/2-Numpy.ipynb

⋆⋆⋆ https://github.com/agramfort/liesse_telecom_paristech_python/blob/master/3-Scipy.ipynb

⋆⋆⋆ http://scikit-learn.org/stable/index.html

⋆⋆ http://www.loria.fr/~rougier/teaching/matplotlib/matplotlib.html

⋆⋆ http://jrjohansson.github.io/

- Rappels de classification -

Définitions et notations

On rappelle ici le cadre de la classification supervisée, et l'on présente les notations que l'on utilisera dans la suite. Il est à noter que pour ce TP on considère un cadre plus général qu'au TP précédent : le nombre de classes peut-être plus grand que deux.

— $\mathcal{Y}$ est l'ensemble des étiquettes des données (*labels* en anglais). Ici on raisonne avec un nombre L quelconque de classes, et l'on choisit $\mathcal{Y} = \{1, \dots, L\}$ pour représenter les L étiquettes possibles (le cas de la classification binaire est le cas où $L = 2$).

— $\mathbf{x} = (x_1, \dots, x_p)^\top \in \mathcal{X} \subset \mathbb{R}^p$ est une observation, un exemple, un point (ou un *sample* en anglais). La j^{e} coordonnée de $\mathbf{x}$ est la valeur prise par la j^{e} variable (*feature* en anglais).

— $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots n\}$ est l'ensemble d'apprentissage contenant les n exemples et leurs étiquettes.

— Il existe un modèle probabiliste qui gouverne la génération de nos observations selon des variables aléatoires X et Y : $\forall i \in \{1, \dots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.

— On cherche à construire à partir de l'ensemble d'apprentissage $\mathcal{D}_n$ une fonction appelée classifieur, $\hat{f} : \mathcal{X} \mapsto \mathcal{Y}$ qui à un nouveau point $\mathbf{x}_{\text{new}}$ associe une étiquette $\hat{f}(\mathbf{x}_{\text{new}})$.

Génération artificielle de données

On considère dans cette partie que les observations sont décrites en deux dimensions (afin de pouvoir les visualiser facilement) à savoir $p = 2$ dans le formalisme ci-dessus.

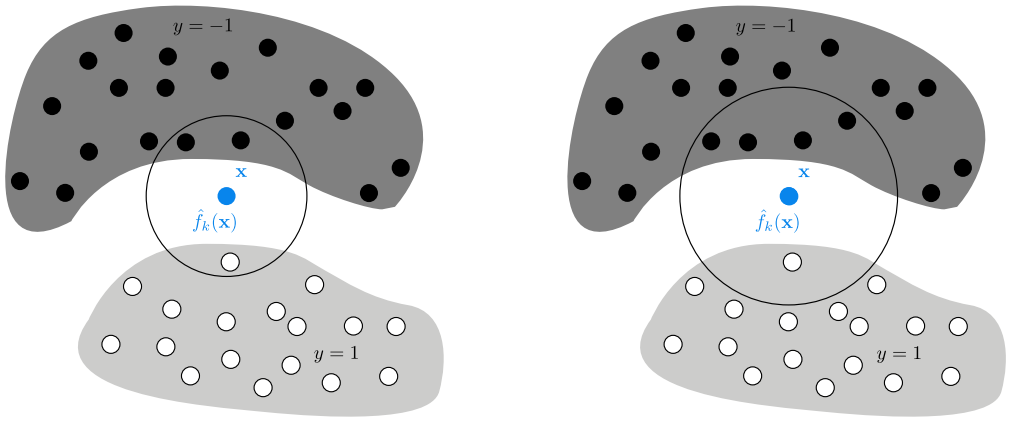
1) Étudiez les fonctions `rand_tri_gauss`, `rand_clown` et `rand_checkers`. Que renvoient ces fonctions ? À quoi correspond la dernière colonne ? Utilisez la fonction `plot_2d` afin d'afficher quelques jeux de données, en jouant si besoin sur les paramètres les générant.

Approche intuitive

L'algorithme des k-plus proches voisins (k-nn : pour *k-nearest neighbors* en anglais) est un algorithme intuitif, aisément paramétrable pour traiter un problème de classification avec un nombre quelconque d'étiquettes.

Le principe de l'algorithme est particulièrement simple : pour chaque nouveau point $\mathbf{x}$ on commence par déterminer l'ensemble de ses k-plus proches voisins parmi les points d'apprentissage que l'on note $V_k(\mathbf{x})$ (bien sûr on doit choisir $1 \leq k \leq n$ pour que cela ait un sens). La classe que l'on affecte au nouveau point $\mathbf{x}$ est alors la classe majoritaire dans l'ensemble $V_k(\mathbf{x})$. Une illustration de la méthode est donnée en Figure 1 pour le cas de trois classes.

1) Proposez une version adaptée de cette méthode pour la régression, *i.e.,* quand les observations sont à valeurs réelles : $\mathcal{Y} = \mathbb{R}$.



(a) $k = 5$ (b) $k = 11$

Figure 1 – Exemple de fonctionnement de la méthode des k-plus proches voisins pour des valeurs du paramètres $k = 5$ et $k = 11$. On considère trois classes, $L = 3$, représentées respectivement en noir ($y = 1$), en gris ($y = 2$) et en blanc ($y = 3$). Pour $k = 3$ (gauche) la méthode prédit le label noir en $\mathbf{x}$ alors que pour $k = 5$ (droite) elle prédit gris.

Approche formelle

Pour définir précisément la méthode, il faut commencer par choisir une distance $d : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}$. Pour un nouveau point $\mathbf{x}$, on définit alors l'ensemble de ses k-plus proches voisins $V_k(\mathbf{x})$ au sens de de cette distance. On peut procéder de la manière suivante : pour chaque $\mathbf{x} \in \mathbb{R}^d$ et pour chaque $i = 1, \ldots, n$, on note $d_i(\mathbf{x})$ la distance entre $\mathbf{x}$ et $\mathbf{x}_i$: $d_i(\mathbf{x}) = d(\mathbf{x}_i, \mathbf{x})$. On définit la première statistique de rang $r_1(\mathbf{x})$ comme l'indice du plus proche voisin de $\mathbf{x}$ parmi $\mathbf{x}_1, \ldots, \mathbf{x}_n$, c'est-à-dire

$$r_1(\mathbf{x}) = i^* \qquad \text{si et seulement si} \qquad d_{i^*}(\mathbf{x}) = \min_{1 \leq i \leq n} d_i(\mathbf{x}).$$

Remarque 1. *S'il y a plusieurs candidats pour la minimisation ci-dessus, on ordonne les ex-æquos de manière arbitraire (généralement aléatoirement).*

Par récurrence on peut ainsi définir le rang $r_k(\mathbf{x})$ pour tout entier $1 \leq k \leq n$:

$$r_k(\mathbf{x}) = i^* \quad \text{si et seulement si} \quad d_{i^*}(\mathbf{x}) = \min_{\substack{1 \leq i \leq n \\ i \notin \{r_1, \ldots, r_{k-1}\}}} d_i(\mathbf{x}). \tag{1}$$

L'ensemble de k-plus proches voisins de $\mathbf{x}$ s'exprime alors par $V_k(\mathbf{x}) = \{\mathbf{x}_{r_1}, \ldots, \mathbf{x}_{r_k}\}$. Pour finir, la décision pour classifier le point $\mathbf{x}$ se fait par vote majoritaire, en résolvant le problème suivant :

$$\hat{f}_k(\mathbf{x}) \in \arg\max_{y \in \mathcal{Y}} \left(\sum_{j=1}^{k} \mathbb{1}_{\{y_{r_j} = y\}} \right). \tag{2}$$

Le module `sklearn.neighbors` de `scikit-learn` (*cf.* http://scikit-learn.org/stable/modules/neighbors.html) implémente les méthodes de classification et régression à base de k-plus proches voisins.

2) Complétez la classe `KNNClassifier` pour ré-implémenter la méthode écrite ci-dessus. Vérifier la validité des résultats en les comparant à ceux de la classe `KNeighborsClassifier` de `scikit-learn`, sur les exemples de jeux de données précédents.

Pour gagner en temps de calcul, vous utiliserez à partir de maintenant l'implémentation de `scikit-learn`.

3) Faites tourner sur les trois exemples de jeux de données cet algorithme de classification, en utilisant la distance euclidienne classique $d(\mathbf{x}, \mathbf{v}) = \|\mathbf{x} - \mathbf{v}\|_2$.

4) Faites varier le nombre k de voisins pris en compte. Que devient la méthode dans le cas extrême où $k = 1$? $k = n$? Afficher ces cas sur les données étudiées. Dans quels cas la frontière est-elle complexe ? simple ?

5) Quel est le taux d'erreur sur vos données d'apprentissage (*i.e.*, la proportion d'erreur faite par le classifieur) lorsque $k = 1$? et sur des données de test ?

6) Tracez les différentes courbes d'erreurs en fonction du paramètre k sur l'un des jeux de données, pour des nombres d'échantillons n variant de 100, 500 à 1000. Quelle est la meilleure valeur de k ? Est-ce toujours la même pour les différents datasets ? Attention à bien évaluer l'erreur sur des données de test. Vous pourrez utiliser la classe fournie `ErrorCurve`.

7) A votre avis, quels sont les avantages et les inconvénients de la méthode des plus proches voisins : temps de calcul ? passage à l'échelle ? interprétabilité ?

8) Appliquez la méthode aux données issues de la base ZIPCODE avec différents choix de $k \geq 1$. On pourra se référer à http://scikit-learn.org/stable/_downloads/plot_digits_classification.py pour le chargement et la manipulation de la base de données. Pour de plus amples informations sur la nature de la classe `'Bunch'` (une sous-classe de dictionnaire, on se reportera à la documentation sur la classe `'dict'` : http://docs.python.org/2/library/stdtypes.html#mapping-types-dict.

9) Estimez la matrice de confusion $(\mathbb{P}\{Y = i, C_k(X) = j\})_{i,\,j}$ associée au classifieur C_k ainsi obtenu. Pour la manipulation de telles matrices avec `scikit-learn`, on pourra consulter http://scikit-learn.org/stable/auto_examples/plot_confusion_matrix.html.

10) Proposez une méthode pour choisir k et mettez-la en œuvre. Vous pourrez utiliser la classe fournie `LOOCurve`.

11) Une variante possible très utilisée consiste à pondérer les poids du j^{e} voisin selon $e^{-d_j^2(\mathbf{x})/h}$ (pour un paramètre h contrôlant le niveau de pondération) : cela revient à remplacer l'Équation (2) par :

$$\hat{f}_k(\mathbf{x}) \in \arg\max_{y \in \mathcal{Y}} \left(\sum_{j=1}^{k} \exp\left(-\frac{d_j^2(\mathbf{x})}{h} \right) \mathbb{1}_{\{y_{r_j} = y\}} \right). \tag{3}$$

Implémentez cette variante dans votre classe `KNNClassifier` et dans `scikit-learn` en passant le paramètre `weights` au constructeur de `KNeighborsClassifier`. On pourra s'inspirer de `_weight_func` de la partie test de `scikit-learn` : https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/neighbors/tests/test_neighbors.py. Testez l'impact du choix de h sur les frontières de classification.

- POUR ALLER PLUS LOIN -

Des détails généraux sur la méthode des k-plus proches voisins se trouvent dans [HTF09, Chapitre 13]. Pour améliorer la compréhension théorique de la méthode on peut se reporter au livre [DGL96, Chapitre 11] et les limites de la méthode quand $k = 1$ http://certis.enpc.fr/%7Edalalyan/Download/DM1.pdf. Enfin pour les considérations algorithmiques on pourra commencer par lire http://scikit-learn.org/stable/modules/neighbors.html#brute-force et les paragraphes suivants.

Références

[DGL96] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic theory of pattern recognition*, volume 31 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, 1996. 4

[HTF09] T. J. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. 4