TP n° 2 : k-plus proches voisins, analyse discriminante linéaire et régression logistique

Vous devez télécharger votre fichier de solution sur eCampus sous le format `ipynb` avant le vendredi 22/12/2023 23h59. Ce travail peut s'effectuer en **binome**, si vous choisissez cette option merci d'indiquer clairement les deux noms dans le notebook (au début).

En plus des question du TP, les aspects suivants seront évalué :

- aspect global de présentation : qualité de rédaction, orthographe, présentation, graphes, titres, etc...,
- aspect global du code : indentation, style, lisibilité du code, commentaires adaptés,
- absence de bugs.

Le fichier `tp_knn_source.py` est également disponible sur eCampus. Ils contient 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 :

— http://www.python.org
— http://scipy.org
— http://www.numpy.org
— 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 $\mathcal{Y} = \{1, \ldots, L\}$ pour représenter les L étiquettes possibles (le cas de la classification binaire est le cas où $L = 2$).

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

— $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots n\}$ 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 : $\forall i \in \{1, \ldots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.

— 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 des observations décrites en deux dimensions (afin de pouvoir les visualiser facilement) à savoir $p = 2$ dans le formalisme ci-dessus. On reprend en partie les jeux de données artificiels du TP sur le perceptron, avec certaines modifications.

1) Étudiez les fonctions `rand_bi_gauss`, `rand_tri_gauss`, `rand_clown` et `rand_checkers`. Que renvoient ces fonctions ? À quoi correspond la dernière colonne ? Générez les données en utilisant chaque de ces 4 fonctions avec les valeurs des paramètres suivantes :
   - pour `rand_bi_gauss` prenez 20 observations dans chaque classe centrées en $(1, 1)^\top$ et $(-1, -1)^\top$, tous les écart-types égaux à 0.9 ;
   - pour `rand_tri_gauss` générez 50 observations dans chaque classe, centrées en $(1, 1)^\top$, $(-1, -1)^\top$ et $(1, -1)^\top$, tous les écart-types égaux à 0.9 ;
   - pour `rand_clown` prenez 50 pour les deux premiers arguments et 1 et 5 pour les deux suivants ;
   - pour `rand_checkers` prenez 150 pour les deux premiers arguments et laissez le troisième par défaut.

   On va nommer ces jeux de données #1, #2, #3 et #4, respectivement.
2) Utilisez la fonction `plot_2d` afin d'afficher les jeux de données générés avec chacune des fonctions.

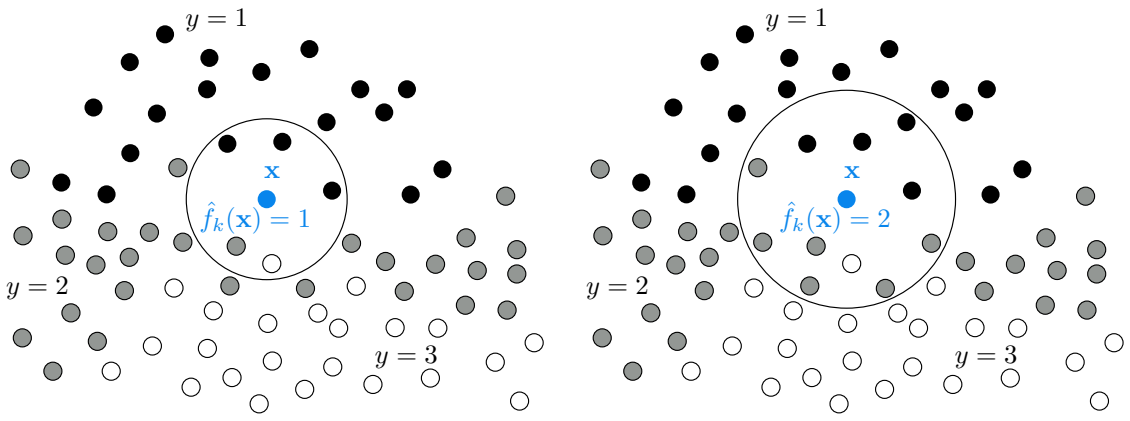
- La méthode des k-plus proches voisins -

Approche intuitive

L'algorithme des k-plus proches voisins (k-nn : pour *k-nearest neighbors* en anglais) est un algorithme intuitif, aisément paramétrisable 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.

3) Proposez une version adaptée de cette méthode pour la régression, *i.e.,* quand les observations y 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 ($y = 3$).

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-œquos de manière arbitraire (généralement aléatoirement).*

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

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

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.

4) Écrivez votre propre classe `KNNClassifier` avec les méthodes d'apprentissage `fit` et de classification `predict`. Choisissez une stratégie de traitement des ex aequo, c'est-à-dire des points avec la même distance ou le même nombre de points de chaque classe dans $V_k(\mathbf{x})$. Vérifier la validité des résultats en les comparant à ceux de la classe `KNeighborsClassifier` de scikit-learn en utilisant le jeu de données #2. Vous proposerez votre propre méthode de comparaison (par exemple, en utilisant les observations d'indice pair pour le set d'apprentissage et celles d'indice impair pour le set de test).
Vous pouvez utilisez le bloc de code si-dessous en complétant le méthodes proposées. Pour plus d'information sur les classes on peut consulter par exemple http://docs.python.org/3/tutorial/classes.html.

```
from sklearn.base import BaseEstimator, ClassifierMixin

class KNNClassifier(BaseEstimator, ClassifierMixin):
    """ Homemade kNN classifier class """
    def __init__(self, n_neighbors=1):
        # Complete the mehtod

    def fit(self, X, y):
        # Complete the mehtod
        return self

    def predict(self, X):
        # Complete the mehtod
```

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

5) Faites tourner sur les quatre exemples de jeu de données cet algorithme de classification, en utilisant la distance euclidienne classique $d(\mathbf{x}, \mathbf{v}) = \|\mathbf{x} - \mathbf{v}\|_2$ et $k = 5$. Visualisez les règles de classification obtenues en utilisant la fonction `frontiere_new`. (Souvent, les autres choix de distance peuvent être utiles, par exemple la distance de Mahalanobis.)

6) Pour les observations d'indice pair du jeu de données #2, faites varier le nombre k de voisins pris en compte : $k = 1, 2, \ldots, n$. Que devient la méthode dans le cas extrême où $k = 1$? $k = n$? Afficher ces cas sur les données étudiées en utilisant la fonction `frontiere_new` et présentez les dans une forme facilement lisible. Dans quels cas la frontière est-elle complexe ? simple ?

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

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

Implémentez cette variante dans scikit-learn en passant le paramètre `weights` au constructeur de `KNeighborsClassifier`. (Une autre possibilité consiste à pondérer les variable et non seulement les observations, on le regarde pas ici.) 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 Pour tester l'impact du choix de h sur les frontières de classification, visualisez les règles de classification pour $k = 7$ et $h = 10^j$ avec $j = -2, -1, 0, 1, 2$; utilisez les observations d'indice pair du jeux de données #2 pour l'entrainement du classifieur (Vous pouvez utiliser la fonction `frontiere_new`.)

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

9) Pour le jeu de données #4, en utilisant les observations d'indice pair pour l'apprentissage et les observations d'indice impair pour le test, tracez le taux d'erreur en fonction de k pour $k = 1, 2, \ldots, 50$. Vous pourrez utiliser la classe fournie `ErrorCurve`.

10) Tracez les différentes courbes d'erreur en fonction du paramètre $k = (1, 2, \ldots, 50)$ sur le jeu de données #4, pour des nombres d'échantillons d'entrainement n prenant les valeurs 100, 200, 500 à 1000. Cette fois, tirez l'ensemble d'apprentissage et l'ensemble de test indépendamment et de même taille. Quelle est la meilleure valeur de k ? Est-ce la même pour les différents datasets ? Vous pourrez utiliser la classe fournie `ErrorCurve`. Pour $n = 1000$ visualisez les données et la règle de décision sur le même graphique. (Vous pouvez utiliser la fonction `frontiere_new`.)

11) 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 ?

12) Étudiez la base DIGITS de scikit-learn. On pourra se référer à https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html pour le chargement et la manipulation de la base de données. Décrivez la nature et le format des données (précisément), affichez un exemple. Tracez l'histogramme pour des classes. Coupez l'échantillon en deux parties de même taille et utilisez la première partie pour l'apprentissage et la deuxième pour le test. Appliquez la méthode aux données issues de la base DIGITS pour un choix de $k \geq 1$ (*e.g.,* $k = 30$) et indiquez le taux d'erreur.

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

14) Proposez une méthode pour choisir k et mettez-la en œuvre. Vous pourrez utiliser la classe fournie `LOOCurve`. En utilisant toutes les données, tracez la courbe du taux d'erreur *leave-one-out* pour $k = 1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 100, 200$. Pour plus d'information sur la validation croisée (*cross-validation*) on peut consulter [**?**, Chapitre 7.10].

- Pour aller plus loin -

Des détails généraux sur la méthode des k-plus proches voisins se trouvent dans [**?**, Chapitre 13]. Pour améliorer la compréhension théorique de la méthode on peut se reporter au livre [**?**, 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.

Le nom anglais est *Linear Discirmiant Analysis* (LDA). Il est préférable de se reporter à cette dénomination en vue de trouver de l'aide en ligne pour la partie numérique. Attention toutefois à ne pas confondre avec Latent Dirichlet Allocation qui est un modèle statistique hiérarchique pour données catégorielles et qui n'a pas de lien avec l'Analyse Discriminante Linéaire.

Aspect théorique

On considère deux populations gaussiennes dans $\mathbb{R}^p$ ayant la **même** structure de covariance. On observe des points dans le mélange de ces deux populations.

Les lois conditionnelles de X sachant $Y=+1$ (respectivement $Y=-1$) sont des gaussiennes multivariées $\mathcal{N}_p(\mu_+,\Sigma)$ (respectivement $\mathcal{N}_p(\mu_-,\Sigma)$). On notera leurs densités respectives f_+ et f_-. Les vecteurs μ_+ et μ_- sont dans $\mathbb{R}^p$ et la matrice Σ est (symétrique) de taille $p\times p$. On note également $\pi_+=\mathbb{P}\{Y=+1\}$.

On rappelle que la densité p-dimensionnelle de la loi $\mathcal{N}_p(\mu,\Sigma)$ est donnée par :

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{\det(\Sigma)}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right\} .$$

et que la matrice de covariance d'un vecteur aléatoire X est définie par $\Sigma = \mathbb{E}\big[(X-\mathbb{E}(X))(X-\mathbb{E}(X))^\top\big]$.

15) En utilisant la formule de Bayes donner la formule des probabilités a posteriori : $\mathbb{P}\{Y=+1 \mid X=\mathbf{x}\}$, $\mathbb{P}\{Y=-1 \mid X=\mathbf{x}\}$, comme fonctions de f_+, f_- et π_+.

16) Exprimer le log-ratio des deux classes :

$$\log\left(\frac{\mathbb{P}\{Y=+1 \mid X=\mathbf{x}\}}{\mathbb{P}\{Y=-1 \mid X=\mathbf{x}\}}\right)$$

en fonction de π_+, μ_+, μ_- et Σ.

On dispose à présent d'un échantillon de ce mélange et on suppose que π_+, μ_+, μ_- et Σ sont des paramètres inconnus. On suppose que l'échantillon considéré contient n observations notées $(\mathbf{x}_1,y_1),\dots,(\mathbf{x}_n,y_n)$ et que $\sum_{i=1}^n \mathbb{1}\{y_i=+1\}=m$. On va utiliser les estimateurs sans biais $\widehat{\pi}_+$, $\widehat{\mu}_+$, $\widehat{\mu}_-$ et $\widehat{\Sigma}$ des paramètres :

$$\widehat{\pi}_+ = m/n\,, \quad \widehat{\mu}_+ = \frac{1}{m}\sum_{i=1}^n \mathbb{1}\{y_i=+1\}\mathbf{x}_i\,, \quad \widehat{\mu}_- = \frac{1}{n-m}\sum_{i=1}^n \mathbb{1}\{y_i=-1\}\mathbf{x}_i$$

et

$$\widehat{\Sigma} = \frac{1}{n-2}\big((m-1)\widehat{\Sigma}_+ + (n-m-1)\widehat{\Sigma}_-\big)$$

avec

$$\widehat{\Sigma}_+ = \frac{1}{m-1}\sum_{i=1}^n \mathbb{1}\{y_i=+1\}(\mathbf{x}_i-\widehat{\mu}_+)(\mathbf{x}_i-\widehat{\mu}_+)^\top\,, \quad \widehat{\Sigma}_- = \frac{1}{n-m-1}\sum_{i=1}^n \mathbb{1}\{y_i=-1\}(\mathbf{x}_i-\widehat{\mu}_-)(\mathbf{x}_i-\widehat{\mu}_-)^\top .$$

17) Justifier le choix du classifieur suivant :

$$\begin{cases} 1 & \text{si } \mathbf{x}^\top\widehat{\Sigma}^{-1}(\widehat{\mu}_+-\widehat{\mu}_-) > \frac{1}{2}\widehat{\mu}_+^\top\widehat{\Sigma}^{-1}\widehat{\mu}_+ - \frac{1}{2}\widehat{\mu}_-^\top\widehat{\Sigma}^{-1}\widehat{\mu}_- + \log(1-m/n) - \log(m/n)\,, \\ -1 & \text{sinon.} \end{cases}$$

Mise en oeuvre

18) Écrivez votre propre classe `LDAClassifier` avec les méthodes d'apprentissage `fit` et de classification `predict`.

19) Importez le module `sklearn.discriminant_analysis` qui contient en particulier la classe `LinearDiscriminantAnalysis` qui nous servira dans la suite.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

Vérifiez la validité des résultats obtenus avec votre méthode en les comparant à ceux de la classe `LinearDiscriminantAnalysis` de scikit-learn en utilisant un jeu de données simulé. Vous proposerez votre propre méthode de comparaison (l'échantillon d'apprentissage doit être petit et l'échantillon de validation doit être assez grand ; on considère par ailleurs que m est différent de $n - m$, *i.e.,* les deux classes ne sont pas représentées par le même nombre d'échantillons observés). Indiquez les taux d'erreur de `LDAClassifier` et de `LinearDiscriminantAnalysis` et le nombre de cas où les prédictions coïncident. En utilisant votre classe `LDAClassifier`, visualisez la règle de classification. (Vous pouvez utiliser la fonction `frontiere_new`.)

20) En utilisant votre classe `LDAClassifier`, visualisez la règle de classification pour les jeux de données #1 et #3. (Vous pouvez utiliser la fonction `frontiere_new`.) Discutez l'efficacité de la méthode dans ces deux cas.

- Régression logistique -

Méthode discriminative avec régression logistique

Importer le module `sklearn.linear_model` qui contient en particulier la classe `LogisticRegression` qui nous servira dans la suite.

```
from sklearn import linear_model
```

21) Appliquez la classification par régression logistique sur les données `rand_bi_gauss`. Comparer les résultats avec la LDA, notamment lorsque une classe est beaucoup plus petite que l'autre (a beaucoup moins d'observations). On parle alors de classes déséquilibrées.

22) À quoi correspond la variable `coef_` du modèle ? `intercept_` ?

23) Utiliser la fonction `frontiere_new` pour visualiser la frontière de décision.

24) Appliquez la classification par régression logistique à des données issues de la base digits. Comme précédemment, coupez l'échantillon en deux parties de même taille et utilisez la première partie pour l'apprentissage et la deuxième pour tester. Indiquez le taux d'erreur.