KNN

Objectifs

Comprendre et implémenter l'algorithme des `k` plus proches voisins (« nearest neighbors »).

Introduction

L'algorithme des k plus proches voisins est l'un des algorithmes utilisés dans le domaine de l'intelligence artificielle. C'est un algorithme d'apprentissage automatique supervisé qui attribue une classe à un élément en fonction de la classe majoritaire de ses plus proches voisins dans l'échantillon d'entraînement.

Son principe peut être résumé par : « Dis-moi qui sont tes amis et je te dirai qui tu es. »

1 Chat ou lapin ?

1.1 Prédiction KNN

Des chats et des lapins ont été recensé en mesurant leur poids ainsi que la taille de leurs oreilles. L'échantillon d'entrainement est constitué de 40 individus regroupé dans une liste `table`. Nous cherchons à déterminer la probabilité pour que la `classe` d'un nouvel animal `cible` soit un chat ou un lapin.

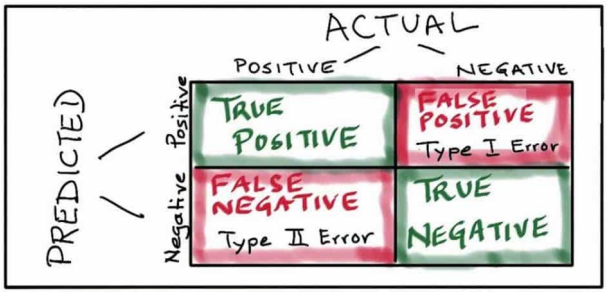
```
>>> table[0] # premier individu de la table. poids : 2kg, oreilles : 3cm
>>> ['chat', 2, 3]
```

1. Écrire une fonction `listes(table, classe)` qui renvoie la liste des poids et la liste des taille d'oreilles des individus d'une `table` s'ils sont de la classe `classe`. On supposera par la suite que les poids sont représentés en abscisses et les tailles d'oreille en ordonnées.
2. Écrire une fonction `distance_cible(donnee)` qui renvoie la distance euclidienne entre une donnée et la cible.
3. Écrire une fonction `k_plus_proches_voisins(table, cible, k)` qui renvoie la liste des `k` plus proches voisins d'une `cible` après avoir trié la `table` selon le critère `distance_cible`.
4. Écrire une fonction `rayon(voisins)` qui renvoie la distance du plus éloigné des plus proches voisin par rapport à la `cible`.
5. Écrire une fonction `population(table, classe)` qui renvoie le nombre d'individus d'une `classe` dans une `classe`.

6. Écrire une fonction `max_voisins(table)` qui renvoie le nom de la classe la plus représentée dans une `table`.
7. Écrire les instructions permettant d'évaluer la `classe` probable de la `cible = [4,8]`.
8. Écrire les instructions permettant d'afficher les individus sur un graphe avec leur poids en abscisses et la taille de leur oreilles en ordonnées. On regroupera les individus par `classe` et on utilisera le nuage de points `plt.scatter`.
9. Écrire les instructions permettant d'ajouter la `cible` sur le graphe dans une autre couleur.
10. Écrire les instructions permettant de tracer un cercle en pointillés autour de la `cible` dont le rayon correspond à la distance euclidienne du voisin le plus éloigné des `k` plus proches voisins.
11. Tester cet algorithme en utilisant la distance de Manhattan puis la distance de Tchebychev définies en annexes après avoir créé les fonctions `distance_M_cible(donnee)` et `distance_T_cible(donnee)` correspondantes. Le résultat de la prédiction est-il stable ?

1.2 Matrice de confusion

On l'algorithme avec la distance euclidienne sur 20 cibles définies dans la `liste_cibles`.



On considère sur cette illustration que `positive=chat` et `negative=lapin`.

12. Que représente un « vrai chat » ? Un « vrai lapin » ? Un « faux chat » ? Un « faux lapin » ?
13. Écrire les instructions permettant de déterminer le type de chaque cible de la liste par l'algorithme pour $k = 7$. On stockera ce résultat dans une `liste_prediction` contenant des chaines de caractères.
14. Écrire une fonction `matrice` qui prends en argument la `liste_reel` et la `liste_prediction` et retourne un tableau de 2 lignes et 2 colonnes (sous la forme de liste de listes par exemple) correspondant à la matrice de confusion de ce système de classification.
15. Calculer la métrique « sensibilité » (ou « rappel ») de vrais positifs définie comme :

$$\text{Rappel} = \frac{TP}{TP + FN}$$

16. Calculer la métrique « précision » de prédictions correctes définie comme :

$$\text{Precision} = \frac{TP}{TP + FP}$$

17. Calculer la métrique « spécificité » de vrais négatifs définie comme :

$$\text{Specificite} = \frac{TN}{FP + TN}$$

18. Calculer la métrique « F-mesure », un score défini comme :

$$\text{Precision} = \frac{2TP}{2TP + FP + FN}$$

19. Conclure.

2 Pokemon

Dans l'univers des Pokémon, les animaux du monde réel n'existent pas (ou très peu). Le monde est peuplé de Pokémon, des créatures qui vivent en harmonie avec les humains, mais possèdent des aptitudes quasiment impossibles pour des animaux du monde réel, telles que cracher du feu ou encore générer de grandes quantités d'électricité.

Chaque espèce de Pokémon se voit attribuer un type parmi les dix-huit existants, qui définissent ses forces et faiblesses aux attaques qu'il peut subir.

Le fichier `pokemon.csv` contient une liste de près de 400 pokemons, dont voici un extrait :

nom	pv	att	def	vit	type
Draco	61	84	65	70	Dragon
Carapuce	44	48	65	43	Eau
Pikachu	35	55	40	90	Electrik
Goupix	38	41	40	65	Feu
Rattata	30	56	35	72	Normal

On souhaite déterminer les types des nouveaux Pokemon suivants :

nom	pv	att	def	vit
Hippodocus	108	112	118	47
Profissor	95	50	100	75
Larvax	55	78	65	31
Neuneuch	80	98	25	49
KraKreKri	66	66	66	66

2.1 Préparation

On modélise un Pokemon par un dictionnaire. Exemple pour `'Mucuscule'` :

```
>>> {'nom': 'Mucuscule', 'pv': 45, 'att': 50, 'def': 35, 'vit': 40, 'type':
↪ 'Dragon'}
```

20. Utiliser la fonction de `lecture` appropriée pour récupérer les données des Pokemon du fichier `.csv` afin de les stocker dans une liste de dictionnaires.

21. Écrire une fonction `pokedex(name)` qui prends en argument le nom d'un Pokémon et qui affiche sa modélisation. Donner la modélisation de `'Pikachu'`.

22. Écrire une fonction `distance(pokemon1, pokemon2)` qui prends deux Pokemon et renvoie la distance euclidienne entre eux. Écrire également une fonction `distance_cible(pokemon)` qui prends en argument un Pokemon et renvoie la distance euclidienne par rapport à la cible.

2.2 Trouver le type majoritaire

23. Écrire une fonction `frequence_des_types(table)` qui prend en argument une `table` de Pokemon et qui renvoie le dictionnaire des fréquences des types. Vous devez obtenir :

```
>>> {'Acier': 5, 'Combat': 20, 'Dragon': 11, 'Eau': 59, 'Electrik' : 30,
↪  'Fée' : 15, 'Feu' : 39, 'Glace' : 13, 'Insecte' : 18, 'Normal' : 62,
↪  'Plante' : 35, 'Poison' : 17, 'Psy' : 35, 'Roche' : 9, 'Sol' : 20,
↪  'Spectre' : 10, 'Ténèbres' : 10, 'Vol' : 2}
```

24. Écrire une fonction `type_majoritaire(table)` qui prend en arguiment une `table` de Pokemon et qui renvoie le nom du type le plus représenté.

2.3 Algorithme des k plus proches voisins

On dispose d'une `table` et d'un `nouveau` qui n'a pas encore de type. On cherche les `k` plus proches voisins de ce nouveau.

25. Écrire une fonction `knn_pokemon(table, nouveau, k)` qui renvoie la liste des `k` plus proches voisins d'un `nouveau` après avoir trié la `table` selon le critère `distance_cible_pokemon`.

2.4 Attribuer un type

On dispose d'une `table` et d'un `nouveau` qui n'a pas encore de type. On cherche à lui attribuer un type.

26. Écrire les instructions permettant de déterminer le type des Pokemon dans le tableau d'introduction.

27. Varier les valeurs de `k` et les différents types de distances. Le résultat reste-il stable ?

28. Observer la table `pokemons`, surtout les effectifs des différents types. Pensez-vous que la table soit efficace pour entrainer une intelligence artificielle à la reconnaissance des types Pokemon ?

Annexes

Distances

Soient deux données `donnee1` et `donnee2` de coordonnées respectives (`x1`, `y1`) et (`x2`, `y2`).

Distance euclidienne (dans un repère orthonormé) d_1 :

$$d_1(donnee1, donnee2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Distance de Manhattan d_2 :

$$d_2(donnee1, donnee2) = |x_1 - x_2| + |y_1 - y_2|$$

Distance de Tchebychev d_3 :

$$d_3(donnee1, donnee2) = max(|x_1 - x_2|, |y_1 - y_2|)$$

Test de fonctions

```
cible = [4,8]
voisins = k_plus_proches_voisins(table, cible, k=11)
classe_cible = max_voisins(voisins)
print(classe_cible)
>>> chat
```

```
cible1
k=19
>>> Hippodocus est surement de type Sol
```

Conclusion

L'algorithme des k plus proches voisins a pour but de classifier une nouvelle donnée en lui attribuant une étiquette à partir des étiquettes déjà connues d'un jeu de données similaires (jeu d'entrainement). Il s'agit d'un apprentissage supervisé.
L'algorithme n'apporte pas une réponse exacte mais une prédiction dont la qualité dépend du jeu de données, de la distance utilisée pour mesurer la similarité entre deux données et du nombre k de voisins les plus proches parmi lesquels l'étiquette majoritaire sera choisie.
La dernière décennie a connu un essor spectaculaire d'algorithmes de classification par apprentissage bien plus performants et perfectionnés qui s'appuient sur des jeux de données gigantesques et apportent des réponses à des questions comme :

- Quelle est la classe de cet objet sur cette photo ? une voiture ? une moto ? une poussette ?
- Quel est, au vu de ses données médicales, le diagnostic que l'on peut faire sur ce patient ? malade ou pas ?
- Quel est, au vu de sa consommation précédente, le profil de ce consommateur de streaming video ? quelles séries peut-on lui suggérer ?
- Quel est, au vu de son dossier administratif, le profil de ce contribuable ? fraudeur ou pas ?