

# Projet python : Classifieur Knn

sbeniamine@linguist.univ-paris-diderot.fr  
benoit.crabbe@linguist.univ-paris-diderot.fr

Vous rendrez ce projet par email (aux deux adresses ci-dessus) le jour de l'examen. Vous écrirez toutes les fonctions dans un unique fichier `classifieur.py`. Le projet est individuel. Vous déclarerez la variable suivante en haut du programme, pour indiquer votre nom et votre prénom :

```
nom = "Nom Prénom"
```

## 1 Énoncé du problème

Ce projet traite d'un problème de classification. On vous propose de réaliser un classificateur de fruits (ou de mots pour la langue naturelle si vous souhaitez le généraliser). Pour un fruit inconnu le classificateur devra déterminer si ce fruit est plutôt une orange ou un citron. Ce projet repose sur un algorithme appelé **algorithme des K plus proches voisins (K nearest neighbors, ou Knn)**. L'idée est qu'un objet situé dans un espace géométrique reçoit la catégorie la plus fréquente parmi les catégories des  $k$  objets les plus proches de lui dans cet espace géométrique.

**Exemple** On dispose dans un espace à deux dimensions des oranges et des citrons en fonction de leurs poids et de leur taille. On veut déterminer pour un nouveau fruit si il s'agit d'une orange ou d'un citron. Pour le catégoriser, on lui attribue la catégorie majoritaire chez ses  $k$  plus proches voisins dans cet espace.

## 2 Résolution du problème

On vous propose d'implémenter l'algorithme des  $k$  plus proches voisins. Pour ce faire, on vous suggère de résoudre le problème en utilisant la décomposition en sous-problèmes suggérée dans ce qui suit.

Les éléments donnés dans l'énoncé (noms de fonctions, arguments, classes) doivent être utilisés à l'identique. Une partie de la note sera obtenue en soumettant votre travail à des tests automatiques qui dépendent de ces informations.

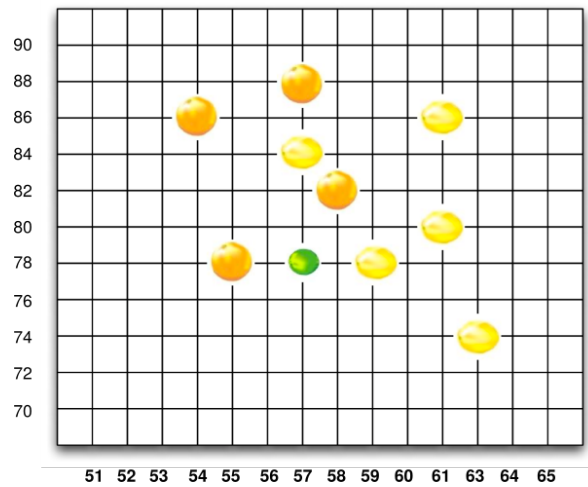


FIGURE 1 – Pour  $k = 3$ , on voit que parmi les trois voisins les plus proches du fruit inconnu, on a deux oranges et un citron. On décide que le fruit inconnu est plutôt une orange.

## 2.1 Représentation d'un objet dans l'espace géométrique

Un objet situé dans l'espace géométrique possède deux propriétés essentielles : une catégorie et des coordonnées  $(x_1, \dots, x_n)$  dans cet espace. On vous propose de représenter un objet par la classe suivante :

```
class Instance(object):
    def __init__(self, categorie, coords):
        """
        @param cat: a string
        @param coords: a tuple of floats
        """
        self.cat = categorie
        self.coords = coords
```

**Exercice 1 :** Écrire dans la classe `Instance` une méthode `__str__(self)` qui renvoie une représentation de cette instance sous forme de chaîne de caractères.

**Exercice 2 :** Écrire dans la classe `Instance` une méthode `distance(self, other)` qui calcule et renvoie la distance euclidienne entre cette instance et une autre instance quel que soit le nombre de dimensions de cet espace. Pour rappel la distance euclidienne  $d(x, y)$  entre deux points  $x = (x_1, x_2, \dots, x_n)$  et  $y = (y_1, y_2, \dots, y_n)$  se mesure comme suit :

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

C'est la racine carrée de la somme sur chaque dimension du carré de la coordonnée de  $x$  moins la coordonnée de  $y$  pour cette dimension.

**Exercice 3 :** Écrire dans la classe `Instance` une méthode `knn(self, k, listeInstances)` qui trouve dans la liste les  $k$  plus proches instances de cette instance-ci. Cette fonction renvoie une liste d'instances de taille  $k$ .

## 2.2 Fonctions de catégorisation

Dans cette section, on vous propose d'implémenter le coeur de l'algorithme Knn. Utilisez les fonctions déjà écrites quand cela est possible.

**Exercice 4 :** Écrire une fonction `most_common(listInstances)` qui renvoie la catégorie la plus fréquente parmi une liste d'instances.

**Exercice 5 :** Écrire une fonction `classify_instance(k, instance, all_instances)` qui renvoie la catégorie la plus fréquente parmi les  $k$  plus proches voisins de `instance`. Le paramètre `all_instances` dénote l'ensemble des instances déjà catégorisées dont `instance` ne fait pas partie.

**Note :**

La méthode de liste `.sort` de python permet de trier une liste : `[4, 5, 3].sort()` retourne `[3, 4, 5]`. Quand la liste contient des éléments complexes, comme des tuples, on peut spécifier une fonction qui prend en argument un élément, et renvoie la valeur sur laquelle trier. Par exemple, si on a écrit une fonction `first_elt` qui prend un tuple et renvoie son premier élément, on peut écrire : `[(10, 5), (1, 6)].sort(key=first_elt)` pour obtenir : `[(1, 6), (10, 5)]`

## 2.3 Lecture et prédiction

**Exercice 6 :** écrire une fonction `read_instances(filename)` qui lit des instances dans un fichier et renvoie une liste d'instances ainsi lues. On suppose que le fichier est structuré selon le schéma suivant : une instance par ligne, les lignes sont structurées en colonnes ; celles-ci sont séparées par des tabulations. La première colonne encode le symbole d'une catégorie, les colonnes suivantes les coordonnées de l'instance.

**Exemple :**

```
orange 77 80
citrons 69 70
oranges 78 89
oranges 88 99
citron 59 71
```

**Exercice 7 :** Écrire une fonction `predict(listInstancesConnues, listInstancesInconnues)` qui catégorise **en place** une liste d'instances dont la catégorie est inconnue en utilisant Knn avec les coordonnées de la liste d'instance connue. Elle assigne à chaque élément de la liste d'instances inconnues son champ `cat`.

## 2.4 Évaluation

**Exercice 8 :** Écrire `eval_classif(ref_instances, pred_instances)`, une fonction qui pour deux listes d'instances de taille identique donne le pourcentage d'instances ayant les mêmes coordonnées et sont catégorisées de manière identique.

**Exercice 9 :** Testez et évaluez votre classificateur en utilisant le protocole suivant :

- Lisez les instances catégorisées du fichier `orangeTest.txt`
- Utilisez les 10 premières instances comme instances connues, et faites la prédiction sur les instances suivantes.
- Comparez ensuite les prédictions réalisées avec les catégories de référence données dans le fichier pour donner une idée de la correction de votre classificateur.

## 2.5 Codage des mots

On peut transposer la classification des oranges et des citrons au cas des mots. Considérons les données suivantes :

```
Entite le chat noir
action ils mangent des
Entite la pomme verte
Action il donne la
Action je viens à
```

où les séquences de mots sont catégorisées en entités (un nom) ou en actions (un verbe). Contrairement au cas de classification des fruits, les mots ne sont pas immédiatement associés à des coordonnées.

Pour obtenir des instances comme dans le cas précédent, il faut encoder les séquences de mots. C'est à dire leur donner des coordonnées. Une manière classique pour ce faire est de définir des fonctions features de la forme :

$$feature_i(w) = \begin{cases} 1 & \text{si } w \text{ a pour premier mot 'le'} \\ 0 & \text{sinon} \end{cases}$$

Dans ce contexte, pour  $n$  fonctions features  $f_i(w)$  ( $0 \leq i < n$ ), chaque instance  $x$  de coordonnées  $(x_0, x_1, \dots, x_n)$  se voit affecter comme coordonnée  $x_i$  la valeur renvoyée par la fonction feature  $f_i(w)$ .

**Exercice 10 :** Écrivez une fonction `compute_coords(wordlist)` qui prend en argument une liste de trois mots et renvoie des coordonnées en utilisant des features que vous écrirez. Écrivez une fonction `read_word_instances` qui lit un fichier semblable à celui décrit ci-dessus, et utilise `compute_coords` pour créer des instances. Utilisez votre classifieur sur ces instances comme précédemment.

### 3 Valider l'interface de votre programme

Une partie de la notation se fondera sur une évaluation automatique de votre programme. Pour cette raison, vous devez respecter scrupuleusement l'interface (nom des fonctions, types des arguments et retours) donnée dans cet énoncé. Pour vous aider, nous mettons à votre disposition un programme de test `test_interface.py` vous permettant de vérifier l'interface de vos fonctions.

Pour tester votre interface, mettez `test_interface.py` dans le même dossier que votre programme `classifieur.py`, puis lancez dans le terminal la commande : `python3 -m unittest -v`. Le résultat devrait être (le temps d'exécution peut varier) :

```
test_InstanceDistance (test_interface.InterfaceTestCase) ... ok
test_Instance_str (test_interface.InterfaceTestCase) ... ok
test_Instanceknn (test_interface.InterfaceTestCase) ... ok
test_classify_instance (test_interface.InterfaceTestCase) ... ok
test_compute_coords (test_interface.InterfaceTestCase) ... ok
test_eval_classif (test_interface.InterfaceTestCase) ... ok
test_mostcommon (test_interface.InterfaceTestCase) ... ok
test_predict (test_interface.InterfaceTestCase) ... ok
test_read_instances (test_interface.InterfaceTestCase) ... ok
test_read_word_instances (test_interface.InterfaceTestCase) ... ok

-----
Ran 10 tests in 0.002s

OK
```