

Programmation Machine Learning – K plus proche voisins (Blockly)


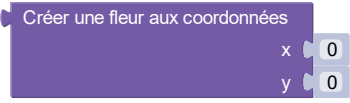
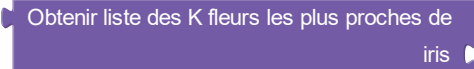
Maintenant que nous avons une bonne compréhension et intuition de l'algorithme des *K plus proche voisins* nous allons le mettre en pratique à l'aide de *Blockly*.



Introduction

Blocs supplémentaires

Beaucoup de concept vus précédemment ont été déjà implémenté pour toi. Nous te donnons déjà implémenté et fonctionnels les blocs suivants :

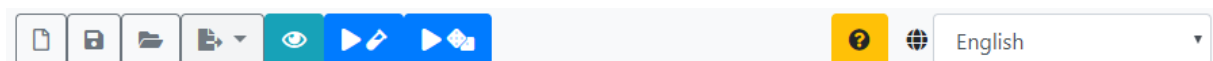
Bloc	Explication	Type de retour
	Donne les noms des différentes fleurs : ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]	Liste de Strings
 x 0 y 0	Permet de créer une fleur qui aura les valeurs données en x et y (dépend de ce que tu as fixé comme paramètres).	Fleur
	Renvoie le nom des K plus proches fleurs de <i>iris</i>	Liste de Strings

Données

Beaucoup de données sont utilisés dans ce projet, tu peux accéder aux données dans le panel de gauche. Tu trouvera entre autre un graphe interactif ainsi qu'un bouton te permettant de consulter les fleurs utilisés dans le modèle d'entraînement. Ce sont ses même fleurs que l'on retrouve dessiné dans le graphe interactif.

Barre d'outil

Dans la partie supérieur de l'écran tu as différents boutons te permettant de réaliser certaines actions que nous allons détailler ici.



Si tu laisse ta souris sur un des boutons, une infobulle t'informera de son action.



: Commencer un nouveau projet, cela effacera tout les blocs.



: Sauvegarder/télécharger ton projet sur ton ordinateur (uniquement les blocs de Blockly)



: Ouvrir un projet précédemment sauvegardé



: Exporte le projet (blocs) sous forme d'images (SVG (vectoriel) ou PNG). Attention que si tu exporte en image tu ne peux pas l'ouvrir comme projet



: Voir le code dans une fenêtre pop-up



: Tester le code. Se contentera d'exécutera le code du projet tel quel.



: Tester le prédicteur. Lance le programme en évaluant si les prédiction sont correctes. Cela te permettra de réussir ton objectif !



: Aide, renvoie vers un fichier qui t'aidera sur le projet (celui-ci)

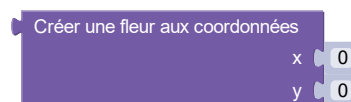


: Te permet de choisir la langue. Seulement le français et l'anglais sont pleinement compatible. Les autres changeront uniquement le texte des blocs de Blockly.

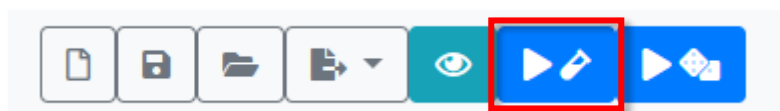
Exemple

Créer une fleur

Les fleurs représentés dans le graphiques font parti du *training set*. Mais tu peux également créer des fleurs « artificielles » (qui ne seront pas pris en compte dans le *training set*) pour tester et expérimenter. Pour cela il te suffit d'utiliser le bloc correspondant (en spécifiant les valeurs x et y) que tu souhaite :



Afin d'ajouter cette fleur dans le graphique, il faut que tu exécute le code (dans le panel *testing*)



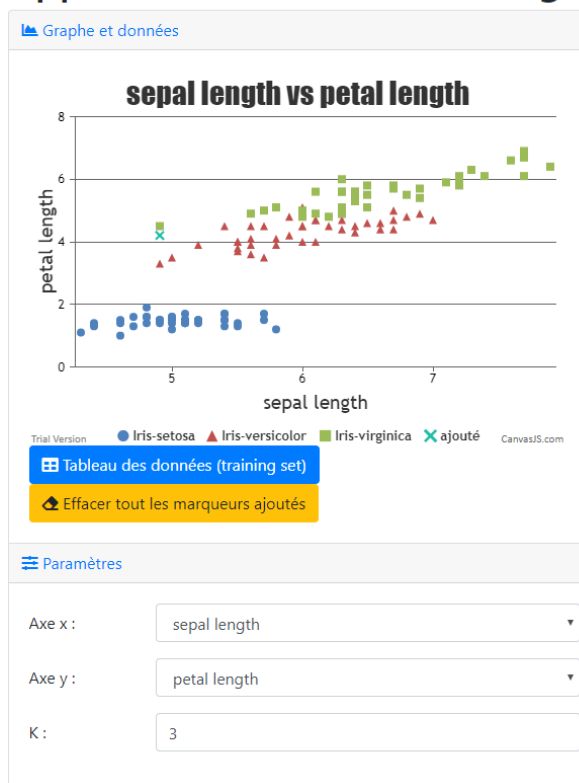
Attention que le graphique se mettra à jour selon les paramètres que tu as choisi pour l'axe x et y . Une croix cyan apparaîtra comme marqueur de ta/tes fleur(s) ajouté.



A partir de là tu peux par exemple tester d'autres blocs comme

Obtenir liste des K fleurs les plus proches de iris

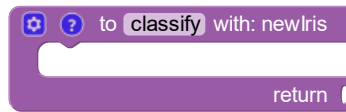
Application de K Nearest Neighbours



Dans cet exemple, on ajoute une fleur artificielle en (4.9, 4.2) avec comme axe x le sepal length et y le petal length. On demande les K (ici 3) plus proche voisins qu'on print à l'écran dans une boîte de dialogue. On voit graphiquement que les 3 plus proche de la croix sont 1 carré vert (virginica) et 2 triangles rouge (versicolor), ce qui est bien le résultat annoncé dans la boîte de dialogue

Objectif

Ton but sera de définir la fonction `classify(newIris)`, cette fonction prends une fleur en argument `newIris` et renvoie le nom prédit de cette fleur.



Ton objectif final est d'atteindre la meilleur prédiction possible (100%).

Prédiction courante :

100.00%

Pour cela, il faut que tu modifie cette fonction de manière qu'elle implémente le concept vu de l'algorithme des *K plus proches voisins*. En plus de cela tu as différent paramètres que tu peux faire varier, tu devra trouver la combinaison qui te donnera la meilleur prédiction. Ses paramètres sont : *K* et le choix des métriques (*x* et *y*).

Paramètres

Axe x :

sepal length

Axe y :

sepal width

K :

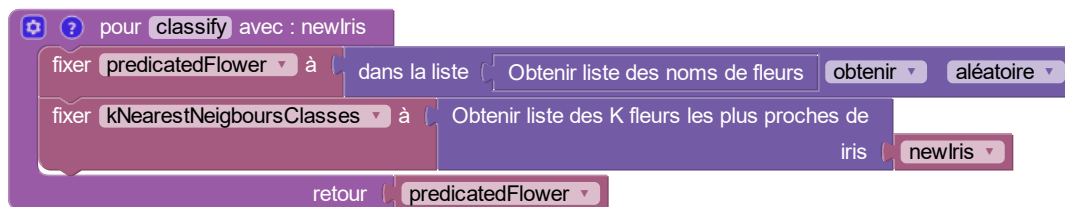
3

Implémentation

L'algorithme peut être séparé en plusieurs étapes distinctes qui petit à petit te rapprocheront de la solution. Je vais t'aider dans cette partie à atteindre la bonne solution en te donnant les intuitions. Libre à toi de faire autrement si tu le souhaite tant que la prédiction finale est bonne

Etape 0

Dans la section TODO, une partie de la fonction à définir est donné (et requise pour tester le prédicteur). Commence par la prendre et teste le résultat de ton prédicteur avec.



Normalement tu devrais avoir un résultat d'environ 33% et chaque fois que tu re-test le prédicteur, il donnera un résultat différent. C'est normal car le prédicteur que je te donne ici est totalement aléatoire et se contente de donner un nom parmi ceux disponible de manière aléatoire. Autrement dit, pour chaque fleur il a 1 chance sur 3 de deviner correctement (d'où les 33%).

Prédiction courante : 33.33%

Etape 1 : Compteur pour chaque nom de fleur

Dans l'état actuel des choses la variable kNearestNeighboursClasses contient une liste (de longueur K) avec le nom des K fleurs les plus proche.

Il faut créer un **compteur** qui compte le nombre de fois que chaque nom apparait. Tu peux utiliser la liste renvoyé par `getFlowersName()` pour l'ordre des noms dans le compteur.

Exemples :

Avec `kNearestNeighboursClasses = ["Iris-setosa", "Iris-versicolor", "Iris-virginica", "Iris-setosa", "Iris-versicolor", "Iris-setosa"]` le compteur serait à la fin `[3, 2, 1]` (*setosa* apparait 3 fois, *versicolor* 2 fois et *virginica* 1 fois).

`["Iris-versicolor"] -> [0, 1, 0]`

`["Iris-versicolor", "Iris-versicolor", "Iris-setosa"] -> [1, 2, 0]`

localhost:63342 indique

3,2,1

OK

classify with:
newIris

createFlower
x 0
y 0

to classify with: newIris

set kNearestNeighboursClasses to

create list with
" Iris-setosa "
" Iris-versicolor "
" Iris-virginica "
" Iris-setosa "
" Iris-versicolor "
" Iris-setosa "

set counter to

print counter

return predictedFlower

Etape 2 : Déterminer la majorité

Maintenant qu'on a compté le nombre d'apparence de chaque fleur, il reste à trouver celle qui apparaît le plus. Autrement dit, il faut trouver l'indice de la valeur maximum du compteur.

Pour cela on va d'abord créer une variable `curMaxIndex`. Ensuite on va parcourir notre compteur, chaque fois qu'on rencontre une valeur plus grande que `curMaxIndex` on va la mettre à jour.

Exemples :

Avec `compteur = [3, 2, 1]` l'indice de la valeur maximum est 0 (car à l'indice 0 on a le maximum de la liste (3)).

`[0, 1, 0] -> 1`

`[1, 2, 0] -> 1`

`[3, 2, 5] -> 2`

`[9999, 2, 5] -> 0`

`[1, 2, 2] -> 1` (en cas d'ambiguïté, on prends le premier maximum rencontré)

localhost:63342 indique

0

OK

classify with:
newIris createFlower
x 0
y 0

```
to classify with: newIris
  set predictedFlower to in list getFlowerNames() get # random in
  set kNearestNeighboursClasses to create list with
    "Iris-setosa"
    "Iris-versicolor"
    "Iris-virginica"
    "Iris-setosa"
    "Iris-versicolor"
    "Iris-setosa"
  set counter to
  print counter
  set curMaxIndex to 0
  print curMaxIndex
  return predictedFlower
```


Etape 3 : Déterminer la fleur correspondante

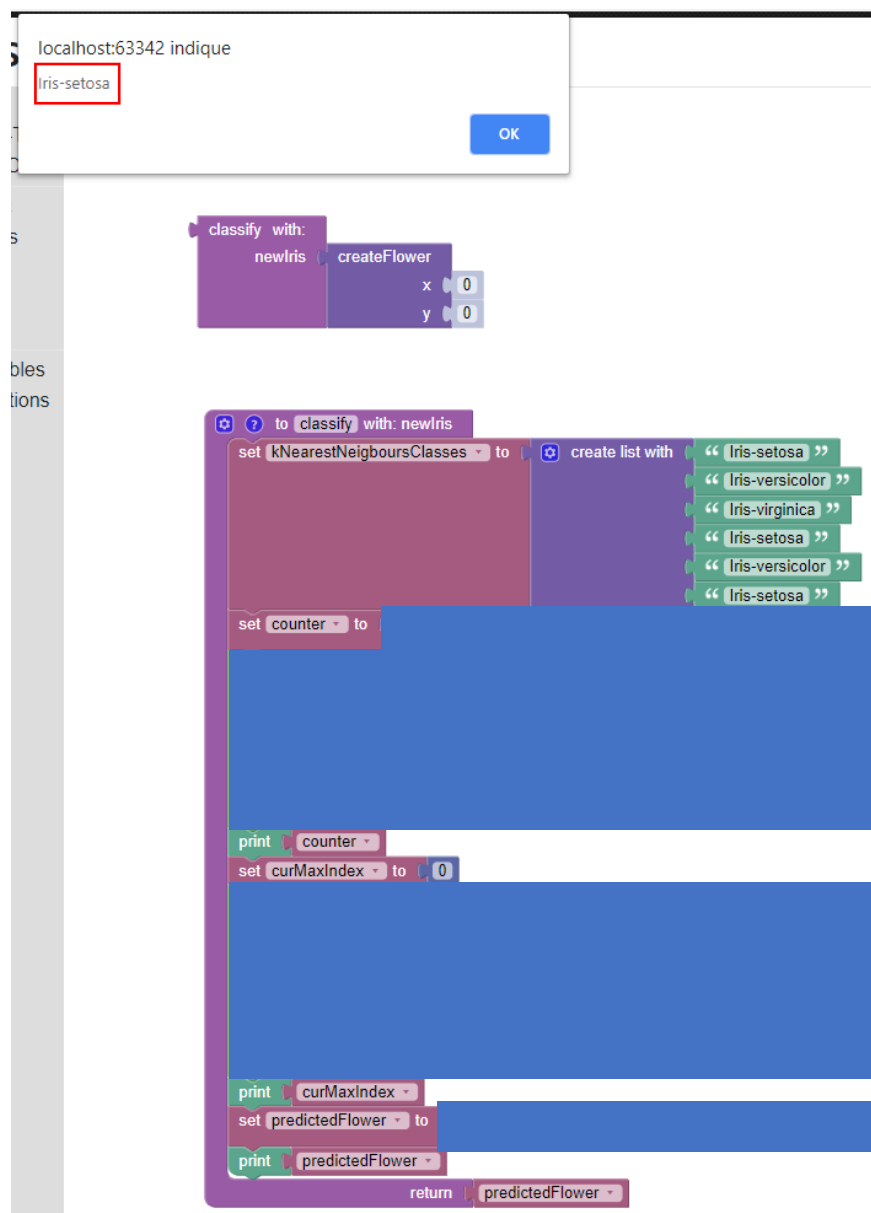
Maintenant qu'on a l'information de l'indice du nom de fleur la plus proche il nous suffit de renvoyer l'élément correspondant dans la liste de référence.

Exemple :

Avec `curMaxIndex` à 0 on aura comme résultat "Iris-setosa".

1 -> "Iris-versicolor"

2 -> "Iris-virginica"



Etape final

N'oublie pas de rétablir la variable `kNearestNeighboursClasses` comme elle l'était initialement.

Test ta méthode avec des fleurs que tu créé pour voir si ton prédicteur à l'air de renvoyer les bonnes valeurs.

Après cela, il ne te reste plus qu'à faire un peu le tri dans ton code ! Enlève tout les `print` qui seront gênant quand tu évaluera ton prédicteur, choisis bien tes paramètres (K , Axe x et Axe y) et croise les doigts pour atteindre les 100% !

Prédiction courante :



100.00%