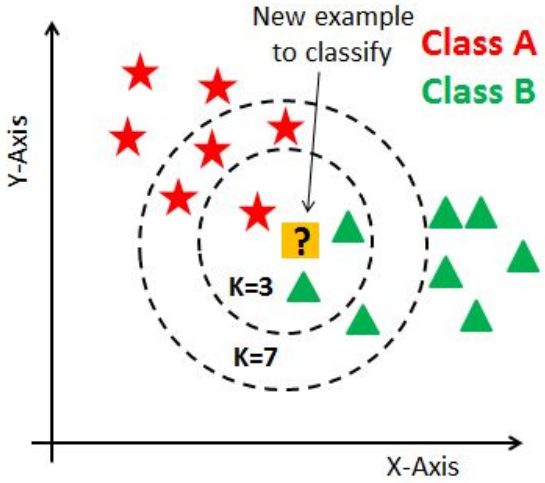
Calcul des performances de l'algorithme des K-plus proches voisins

Preparé par ELbouziani Youssef

Présentation de l'algorithme

La méthode des "k plus proches voisins" fait partie des méthodes les plus simples d'apprentissage supervisé pouvant être utilisée pour les cas de régression et de classification

Les 'k plus proches voisins' ou k-nearest neighbors en anglais (d'où l'appellation knn) est une méthode non paramétrique dans laquelle le modèle mémorise les observations de l'ensemble d'apprentissage pour la classification des données de l'ensemble de test.



1. les étapes

➔ **étape 1 :**
On fixe le nombre de voisins k

➔ **étape 2 :**
On détecte les k-voisins les plus proches des nouvelles données d'entrée que l'on veut classer.

➔ **étape 3 :**
On attribue les classes correspondantes par vote majoritaire

➔ **étape 4 :**
On fait varier la Valeur de K et on calcule le taux d'erreur de l'ensemble de test

Implementation

pour notre implémentation , on choisi ce dataset avec les donnés suivants

testL	testX	testY	testZ	trainL	trainX	trainY	trainZ
0	-0.624704356	-4.688029297	2.102693217	0	-4.170290684	4.870222019	7.677576867
0	-8.281681544	7.029983978	2.323949035	0	4.296635386	-3.750514992	0.389868009
0	-0.654612039	7.969329366	3.905546654	1	-1.249713893	-3.280842298	0.524910353
0	-1.757534142	-3.281757839	1.452658885	0	5.553368429	-7.082169833	2.915999084
0	0.295719844	-0.031586175	0.781261921	1	3.997558557	5.7628748	0.584420539
0	5.937895781	-7.664454108	0.036621653	1	-0.937819486	-2.108949416	0.537117571
0	-2.344090944	7.382467384	3.124284733	0	2.773785	-9.209887846	2.526894026
1	-0.312809949	3.515220874	1.390096895	0	0.13946746	-0.043488212	0.19531548
0	-2.265964752	7.621423667	6.250095369	1	-8.07660029	-4.215609979	0.975051499

ce dataset contient train-data et test-data avec 3 paramètres (x,y,z)

Implementation

La première étape est de récupérer les valeurs de chaque colonne a partir du fichier **dataa.txt** et initialiser les variables ,accuracy ,precision et recall (vectors)

```
filename = 'dataa.txt';
file_data = dlmread(filename);
yval = file_data(:,5);
xval = file_data(:,6);
zval = file_data(:,7);
tval = file_data(:,8);

train_data = [xval zval tval yval];

Yval = file_data(:,1);
Xval = file_data(:,2);
Zval = file_data(:,3);
Tval = file_data(:,4);

test_data = [Xval Zval Tval Yval]
      %AXTRACT JUST 100  ROW from the data
     rows=100;

     neighbor_distances_and_indices = zeros(rows,2);
     Accuracies=zeros(10,1);
     Precisions=zeros(10,1);
     Recalls=zeros(10,1);

     # 3. For each example in the
```

Implementation

La 2éme étape et pour chaque valeur de K , la récupération de chaque entrée pour calculer la distance euclidienne avec toutes les autres points de train-set en utilisant la fonction suivante

cette fonction retourne un vecteur (D) de toutes les distances

```
function D = distance_fn(train_data,test_data)


    trainX = train_data(:,1);
    trainY = train_data(:,2);
    trainZ = train_data(:,3);
    X = zeros(1, size(train_data,1));
    Y = zeros(1, size(train_data,1));
    Z = zeros(1, size(train_data,1));
    for j = 1:size(train_data,1)
        X(j) = power (trainX(j) - test_data(1),2);
        Y(j) = power (trainY(j) - test_data(2),2);
        Z(j) = power (trainZ(j) - test_data(3),2);
        D(j) = sqrt (X(j) + Y(j) + Z(j));
    end
    end
```

Implementation

La première 3éme étape est d'extraire pour chaque entrée le predict_label (y)

```
distance = distance_fn(train_data(:,1:3),test_data(i,1:3));
distance = distance';

# 3.2 Add the distance and the index of the example to an ordered collection

%Stockage des indices et distances
for j=1:rows
  neighbor_distances_and_indices(j,1)=distance(j,1);
  neighbor_distances_and_indices(j,2)=j;
end
# 4. Tri de tableau par distance
# Tri Croissant
sorted_neighbor_distances_and_indices = sortrows(neighbor_distances_and_indices,1);

# 5. Recuperation de  K entreés a partir de la collection trié
k_nearest_distances_and_indices = sorted_neighbor_distances_and_indices(1:k,1:2);

# 6. Get the labels of the selected K entries
k_nearest_labels = zeros(k,1);
for t=1:k
 k_nearest_labels(t,1)=train_data(k_nearest_distances_and_indices(t,2),4);
end
k_nearest_labels;


label(i,1)=mode(k_nearest_labels);
k_nearest_distances_and_indices_matrix=k_nearest_distances_and_indices;
```

< - - récupération des voisins et ses indices

- tri des voisins selon les distances

recuperation des labels les plus fréquents

Implementation

Calcul des performances (Accuracy)

```
  pred_val = label;
  accuracy = mean(double(pred_val == Yval));
acc_all0 = mean(double(0 == Yval));
```

```
printf("|--> accuracy == %f vs accuracy_all0 == %f \n",accuracy,acc_all0);
```

Pour Calculer l accuracy , on utilise la formule $accuracy = (TP + TN) / TP + TN + FP + FN$

ou tout simplement la moyenne de (predict_values == Real_values) (voir la fonction

Implementation

Calcul des performances (Précision)

Pour Calculer la précision , on utilise la formule :

Précision = TP / (TP + FP)

Calcul des performances (Recall)

Pour Calculer Recall , on utilise la formule :

Recall= TP / (TP + FN)

```
actual_positives = sum(Yval == 1);
actual_negatives = sum(Yval == 0);
true_positives = sum((pred_val == 1) & (Yval == 1));
false_positives = sum((pred_val == 1) & (Yval == 0));
false_negatives = sum((pred_val == 0) & (Yval == 1));
precision = 0;
if ( (true_positives + false_positives) > 0)
  precision = true_positives / (true_positives + false_positives);
endif

recall = 0;
if ( (true_positives + false_negatives) > 0 )
  recall = true_positives / (true_positives + false_negatives);
endif

F1 = 0;
if ( (precision + recall) > 0)
  F1 = 2 * precision * recall / (precision + recall);
endif
```

Implementation

et la dernière étape la récupération de la vecteur des performances pour chaque valeur de K , avant de dessiner un graphe pour chacune

```
    %stock data
    Accuracies(k)=accuracy;
    Precisions(k)=precision;
    Recalls(k)=recall;

end
    Accuracies
    Precisions
    Recalls
    k_values=1:10;                    % make up some sample data*
subplot(3 ,1 ,1);
plot(k_values,Accuracies)                    % plot it
[~,imn]=min(Accuracies);             % get min/max locations
[~,imx]=max(Accuracies);
hold on
plot(k_values([imn;imx]),Accuracies([imn;imx]),'or')  % option one v
ix=[imn;imx];                         % second option to build index
title('Accuracy values per K-neignbours')
xlabel('K-Values')
ylabel('Accuracy')
plot(k_values(ix),Accuracies(ix),'xk')                % plot that wa

%PRECISION
```

Implementation

Le programme affiche les valeurs des ces performances pour chaque valeur de K

```
Pour k = 1
|--> accuracy == 0.730000 vs accuracy_all0 == 0.590000
|-->  true_positives == 20   (actual positive =41)
|-->  false_positives == 6
|-->  false_negatives == 21
|-->  precision == 0.769231
|-->  recall == 0.487805
|-->  F1 == 0.597015
Pour k = 2
|--> accuracy == 0.680000 vs accuracy_all0 == 0.590000
|-->  true_positives == 11   (actual positive =41)
|-->  false_positives == 2
|-->  false_negatives == 30
|-->  precision == 0.846154
|-->  recall == 0.268293
|-->  F1 == 0.407407
Pour k = 3
|--> accuracy == 0.710000 vs accuracy_all0 == 0.590000
|-->  true_positives == 17   (actual positive =41)
|-->  false_positives == 5
|-->  false_negatives == 24
|-->  precision == 0.772727
|-->  recall == 0.414634
|-->  F1 == 0.539683
Pour k = 4
|--> accuracy == 0.700000 vs accuracy_all0 == 0.590000
|-->  true_positives == 12   (actual positive =41)
|-->  false_positives == 1
|-->  false_negatives == 29
|-->  precision == 0.923077
|-->  recall == 0.292683
|-->  F1 == 0.444444
Pour k = 5
|--> accuracy == 0.700000 vs accuracy_all0 == 0.590000
|-->  true_positives == 17   (actual positive =41)
|-->  false_positives == 6
|-->  false_negatives == 24
|-->  precision == 0.739130
|-->  recall == 0.414634
|-->  F1 == 0.531250
```

Implementation

et enfin donne le résultat sous forme de graphe qui affiche les performances en fonction de K , et la valeur maximale et minimale pour chacune