
KNN
Algorithm
FROM SCRATCH IN PYTHON

Entraînez votre premier k-NN : Application _ Test De Personnalité

Partie 1 : Base de données, Analyse, Prétraitement et Préparation

Partie 2 : Développement et entraînement d'un modèle KNN

KNN From Scratch

KNN Sklearn

Partie 3 : Mettre en place la solution dans l'application de test de personnalité

Partie 1 : Base de données, Analyse, Prétraitement et Préparation

1.0 Compréhension du sujet

Score :

- 1 ou a ou A vaut 1 point
- 2 ou b ou B vaut 0 point
- 3 ou c ou C vaut 2 points

Interpretation :

- score < 10 => C
- score < 20 => B
- score < 30 => A

Notre jeu de données comporte plusieurs erreurs.

Premièrement, il comporte des NaN, c'est-à-dire des valeurs manquantes.

Deuxièmement, il comporte des valeurs erronées, qui ne correspondent pas aux valeurs acceptées en entrée par le questionnaire. Il faut donc les remplacer.

1- Traitement des valeurs manquantes NaN

```
dataset.isnull().sum()
```

On observe plusieurs valeurs manquantes.

On peut soit les remplacer par des valeurs nulles (b où 2), où utiliser la fonction "mode" de pandas.

Celle-ci va rendre compte des valeurs les plus redondantes dans chaque colonne du dataset.

```
dataset.mode().iloc[0]
```

On remplace les NaN par les données explicitées

```
dataset.fillna(dataset.mode().iloc[0], inplace=True)
```

On vérifie qu'il ne nous reste plus de valeurs manquantes NaN.

```
dataset.isnull().sum()
```

2- Traitement des valeurs erronées

Notre hypothèse de travail sera la suivante :

1) On va devoir séparer les valeurs de notre jeu de données en deux : True et False.
2) Ensuite, on remplace l'une des deux valeurs (qui représente les valeurs erronées) par un NaN.
3) Enfin, on applique la solution effectuée à la question précédente, pour remplacer les NaN par les valeurs de la fonction.mode()

- *Remplacement des valeurs par True et False*

On va devoir remplacer toutes les valeurs qui ne correspondent pas à a, b, c ou 1,2 et 3.

```
dataset.iloc[:, 0:10].isin(["a", "b", "c", "1", "2", "3"])
```

- *Remplacement des valeurs erronées par des NaN*

On écrase notre dataset originel par le nouveau dataset dans lequel les valeurs erronées sont remplacées par des NaN.

```
dataset.iloc[:, 0:10] = dataset.iloc[:, 0:10].where(dataset.iloc[:, 0:10].isin(["a", "b", "c", "1", "2", "3"]))
```

- *Remplacement des NaN par la fonction .mode()*

```
dataset.fillna(dataset.mode().iloc[0], inplace=True)
```

- *Remplacement des valeurs et conversion du dtype*

Une fois que les NaN et les valeurs erronées ont été nettoyées de la base de données, il va falloir changer les lettres (A,B,C) par des chiffres (1,2,3), puis convertir les dtypes "object" en "int32".

```
dataset.info()
```

On remplace les lettres par des chiffres.

```
dataset= dataset.replace(['a', 'b', 'c'], [1, 2, 3])

dataset= dataset.replace(['A', 'B', 'C'], [1, 2, 3])

dataset
```

Puis on convertit les "dtypes"

```
dataset.astype("int")
```

1 - Modèle from scratch

Trois fonctions de calcul de distances sont initialisées, la distance euclidienne, la distance Manhattan, la distance Minkowski.

Pour ces trois fonctions, le module Numpy est utilisé afin d'utiliser le calcul matriciel, pour alléger l'écriture.

La fonction distance retourne les distances sur le Dataframe.

La fonction predict calcule la distance minimum et prédit l'interprétation.

Calculer la précision de notre modèle : `0.8333333333333334`

GridSearch

```
from sklearn.model_selection import GridSearchCV
```

Réglage des hyperparamètres

A partir d'ici, nous allons régler les paramètres pour l'application du GridSearch. Il va donc falloir régler les hyperparamètres que nous voulons, c'est-à-dire le nombre de "K" ainsi que la distance. On va avoir en sortie les meilleurs paramètres.

- *Matrice de confusion*

```
from sklearn.metrics import confusion_matrix

from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(knn, X_test, y_test)
```

3 KNN Sklearn

1- Séparer le Target et les features :

```
X = data.drop(['Interpretation', 'Score'], axis=1)
y = data['Interpretation']
```

2- Entrainer les données :

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2)
```

3- Normaliser les données :

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
Précision du modèle :  0.6976744186046512
```

Pour nos données, on remarque que le modèle from scratch (0.83) nous donne une meilleure précision que le modèle KNN (0.69)