IA - TP3 : Classifieur bayésien naif

Vincent Lagogué - Thomas Peugnet - David Tejeda

Explications

Le jeu de données n’était pas au format CSV standard, car il utilise des points-virgules (;) comme délimiteurs. Non l’avons donc converti en remplacçant les points-virgules par des virgules.

Le classificateur Naive Bayes est un type de modèle qui utilise les probabilités pour faire des prédictions. Il fonctionne mieux avec des données où les caractéristiques sont clairement séparées en catégories ou groupes distincts, comme "oui" ou "non", "rouge" ou "bleu". Ce modèle compte combien de fois chaque catégorie apparaît pour chaque caractéristique afin de faire ses calculs. Ces caractéristiques catégorielles sont idéales pour ce modèle, car elles n'ont qu'un nombre limité d'options différentes, ce qui facilite le comptage et le calcul des probabilités.

Nous commençons par importer les bibliothèques nécessaires telles que pandas pour la manipulation des données, sklearn pour la division du jeu de données et l'encodage des étiquettes.

Ensuite nous chargeons le jeu de données depuis un fichier CSV.

Puis, nous sélectionnons uniquement les colonnes catégorielles et utilisons LabelEncoder pour convertir les valeurs catégorielles en nombres entiers, ce qui est nécessaire pour le calcul des probabilités.

Pour pouvoir entrainer et tester l'IA, nous divisons le jeu de données en deux ensembles.

calculate_probabilities permet decalculer la probabilité de chaque classe (a priori) et la probabilité de chaque valeur de caractéristique dans chaque classe (conditionnelle).

predict utilise les probabilités calculées pour prédire la classe de nouvelles observations. Pour chaque observation, elle calcule le score pour chaque classe en multipliant les probabilités conditionnelles correspondantes, puis choisit la classe avec le score le plus élevé.

Finalement, nous évaluons la précision du modèle en comparant les prédictions avec les vraies étiquettes de l'ensemble de test.

.
.

Code Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Chargement du jeu de données
file_path = 'processed_cardiovascular_data.csv'
data = pd.read_csv(file_path)

# Sélectionner uniquement les colonnes catégorielles
categorical_columns = data.select_dtypes(include=['object']).columns
# Créer une copie pour éviter l'avertissement de modification d'une vue d'un autre
DataFrame
categorical_data = data[categorical_columns].copy()

# Encodage des caractéristiques catégorielles
label_encoders = {}
for column in categorical_data.columns:
    label_encoders[column] = LabelEncoder()
    # Utilisation de .loc pour une modification sûre
    categorical_data.loc[:, column] =
label_encoders[column].fit_transform(categorical_data[column])

# Division du jeu de données en ensembles d'entraînement et de test (80 % pour
l'entraînement, 20 % pour les tests)
X = categorical_data.drop('GOAL-Heart Disease', axis=1)
y = categorical_data['GOAL-Heart Disease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Fonction pour calculer les probabilités a priori et les probabilités
conditionnelles
def calculate_probabilities(X, y):
    class_probabilities = y.value_counts(normalize=True)
    conditional_probabilities = {}

    for column in X.columns:
        conditional_probabilities[column] = {}
        for value in X[column].unique():
```

```
            # Calcul des probabilités pour chaque valeur de caractéristique dans
chaque classe
            probabilities = X[column][y == value].value_counts(normalize=True)
            conditional_probabilities[column][value] = probabilities

    return class_probabilities, conditional_probabilities

# Fonction pour prédire les classes
def predict(X, class_probabilities, conditional_probabilities):
    predictions = []
    for _, row in X.iterrows():
        class_scores = {}
        for class_value in class_probabilities.keys():
            class_scores[class_value] = class_probabilities[class_value]
            for column in X.columns:
                # Multiplier les scores de classe par les probabilités
conditionnelles
                class_scores[class_value] *= conditional_probabilities[column]
[class_value].get(row[column], 0)

        # Choisir la classe avec le score le plus élevé
        predictions.append(max(class_scores, key=class_scores.get))

    return predictions

# Calcul des probabilités
class_probabilities, conditional_probabilities = calculate_probabilities(X_train,
y_train)

# Prédiction sur l'ensemble de test
predictions = predict(X_test, class_probabilities, conditional_probabilities)

# Calcul de la précision
accuracy = sum(predictions == y_test) / len(y_test)
print(f'Précision du modèle : {accuracy}')
```

.

.

Résultat de l'execution du code

Nous avons obtenu une précision de 84% suite au test du modèle après entrainement.