Guide pour la réalisation de projets en Machine Learning

Amine Hadj-Youcef

Version 0.1, 2023-02-05

Table des matières

Introduction

Ce document est un guide des étapes impliquées dans un projet d'apprentissage automatique. Il couvre différentes étapes telles que la préparation des données, l'analyse des données, la modélisation et la validation. L'étape de préparation des données implique le nettoyage et la préparation des données pour la modélisation, y compris le nettoyage des données, la sélection des caractéristiques, la transformation des variables et la division des données.

L'étape de modélisation comprend l'apprentissage supervisé et non supervisé, avec des algorithmes tels que la classification, la régression ou le regroupement, et nécessite une attention minutieuse pour éviter les fuites de données. L'étape de modélisation comprend la sélection des caractéristiques et des variables cibles, la division des données, la transformation des variables, la validation croisée et l'entraînement du modèle d'apprentissage automatique. La dernière étape consiste à choisir le bon algorithme de modélisation et à évaluer sa performance.

Préparation des données pour l'analyse exploratoire

- Données non structurées: image/vidéo ...
- Données structurées:

Importation des données

Charger les données depuis différentes sources telles que des fichiers CSV, JSON, Parquet, etc.

Vous pouvez charger un fichier CSV avec `pandas` en utilisant la méthode `read_csv` comme ceci :

```
import pandas as pd

df = pd.read_csv("file.csv")
```

Vous pouvez charger un fichier JSON avec `pandas` en utilisant la méthode `read_json` comme ceci :

```
df = pd.read_json("file.json")
```

Vous pouvez charger un fichier Parquet avec `pandas` en utilisant la méthode `read_parquet` comme ceci :

```
df = pd.read_parquet("file.parquet")
```

Il est important de noter que pour utiliser les méthodes read_parquet, vous devrez avoir installé la bibliothèque `fastparquet` en plus de `pandas`.

Nettoyage des données

Supprimer les valeurs manquantes, les valeurs aberrantes et les lignes dupliquées, afin d'obtenir des données propres pour l'analyse.

Voici un exemple de code

```
import pandas as pd

# charger le fichier csv
df = pd.read_csv('data.csv')

# supprimer les valeurs manquantes
df = df.dropna()
```

```
# supprimer les valeurs aberrantes
df = df[(df - df.mean()).abs() <= 3 * df.std()]

# supprimer les lignes dupliquées
df = df.drop_duplicates()
```

Choix des features pertinents

Sélectionner les features (colonnes) qui seront les plus utiles pour l'analyse exploratoire.

Voici un exemple de code Python pour choisir les features pertinents avec Pandas :

```
import pandas as pd

# Charger les données dans un DataFrame
df = pd.read_csv("data.csv")

# Sélectionner un sous-ensemble de features (colonnes)
selected_features = ["feature1", "feature2", "feature3"]
df = df[selected_features]

# Vérifier les données pour vous assurer que les colonnes sont sélectionnées
print(df.head())
```

Ajout / Transformation des features

Créer de nouvelles features ou transformer les features existantes pour les rendre plus adaptées à l'analyse. L'ajout ou la transformation des features peut être fait avec le code suivant:

```
# Ajout de nouvelles features
df['new_feature'] = df['existing_feature_1'] + df['existing_feature_2']

# Transformation des features existantes
df['existing_feature'] = df['existing_feature'].apply(lambda x: np.log(x + 1))
```

(Optionnel) Réduction de dimension

Utiliser des techniques telles que l'analyse en composantes principales (ACP) pour réduire le nombre de features.

Voici un exemple de code Python pour effectuer la réduction de dimension par analyse en composantes principales (ACP) à l'aide de la bibliothèque scikit-learn :

```
from sklearn.decomposition import PCA
```

```
# Initialiser le modèle de réduction de dimension
pca = PCA(n_components=2)

# Ajuster le modèle sur les données d'entraînement
pca.fit(X_train)

# Appliquer la réduction de dimension aux données d'entraînement et de validation
X_train_pca = pca.transform(X_train)
X_valid_pca = pca.transform(X_valid)
```

Ici, nous initialisons le modèle PCA avec `n_components=2` pour réduire les données à deux dimensions. Nous ajustons (entraînons) ensuite le modèle sur les données d'entraînement `X_train` et appliquons la réduction de dimension aux données d'entraînement et de validation `X_train_pca` et `X_valid_pca`.

Vérification de la qualité des données

Vérifier que les données sont fiables et complètes et corriger les erreurs ou les valeurs manquantes.

Voici un exemple de code Python pour vérifier la qualité des données et corriger les erreurs ou les valeurs manquantes :

```
import pandas as pd

# Charger le dataset
df = pd.read_csv("dataset.csv")

# Vérifier les valeurs manquantes
print("Valeurs manquantes avant nettoyage :")
print(df.isnull().sum())

# Supprimer les lignes avec des valeurs manquantes
df = df.dropna()

# Vérifier les valeurs manquantes après suppression
print("Valeurs manquantes après nettoyage :")
print(df.isnull().sum())

# Supprimer les doublons
df = df.drop_duplicates()

# Vérifier les valeurs aberrantes
print("Valeurs aberrantes avant nettoyage :")
print(df.describe())

# Supprimer les valeurs aberrantes
df = df[(df > df.mean() - 3*df.std()) & (df < df.mean() + 3*df.std())]

# Vérifier les valeurs aberrantes après suppression
```

```
print("Valeurs aberrantes après nettoyage :")
print(df.describe())
```

Exploration des données

Utiliser des méthodes telles que des graphiques, des statistiques descriptives, des corrélations, etc. pour comprendre les relations entre les features et les tendances dans les données.

Voici un code pour explorer les données avec des graphiques et des statistiques descriptives en utilisant le package pandas en Python:

```
import pandas as pd
import matplotlib.pyplot as plt

# Charger le jeu de données dans un dataframe
df = pd.read_csv("data.csv")

# Afficher les informations générales sur les données
print(df.info())

# Afficher les statistiques descriptives pour les features numériques
print(df.describe())

# Dessiner des histogrammes pour les features numériques
df.hist(bins=50, figsize=(20,15))
plt.show()

# Calculer la corrélation entre les features numériques
print(df.corr())

# Dessiner des graphiques de dispersion pour les paires de features numériques
pd.plotting.scatter_matrix(df, figsize=(12, 8))
plt.show()
```

Notez que les étapes décrites ci-dessus ne sont qu'un exemple et peuvent varier en fonction des données et des besoins de l'analyse. Il est important de s'assurer que les graphiques et les statistiques choisis sont pertinents pour le problème à résoudre et les données à explorer.

Résumé et visualisation des données

Visualiser les données pour comprendre les relations entre les variables et pour identifier des tendances ou des modèles cachés.

Voici un code simple en Python utilisant Pandas et Matplotlib pour effectuer un résumé et une visualisation des données:

```
import pandas as pd
```

```
import matplotlib.pyplot as plt

# Charger les données dans un DataFrame pandas
df = pd.read_csv('data.csv')

# Aperçu des données
print(df.head())

# Statistiques descriptives
print(df.describe())

# Visualisation de la distribution de la variable cible
plt.hist(df['target'])
plt.show()

# Visualisation de la relation entre 2 features numériques
plt.scatter(df['feature_1'], df['feature_2'])
plt.xlabel('feature_1')
plt.ylabel('feature_2')
plt.show()
```

Ce code lit un fichier CSV et utilise les méthodes `head()` et `describe()` de `Pandas` pour obtenir un aperçu des données et des statistiques descriptives. Il utilise également `Matplotlib` pour visualiser la distribution de la variable cible et la relation entre deux features numériques.

Modélisation par Machine Learning

La modélisation par Machine Learning est une technique utilisée pour apprendre les relations entre les variables d'une donnée, sans être explicitement programmées. Il existe deux types principaux de modélisation par Machine Learning : supervisée et non supervisée.

La modélisation supervisée implique que les algorithmes de Machine Learning sont formés à partir de données d'entraînement qui ont été étiquetées. Les algorithmes apprennent à identifier les relations entre les variables d'entrée et les variables de sortie. Cette technique est utilisée pour la classification des données, où les données sont classées en différentes catégories en fonction de leurs caractéristiques. Elle est également utilisée pour la prédiction / régression, où les algorithmes apprennent à prédire une variable de sortie en fonction de variables d'entrée.

La modélisation non supervisée implique que les algorithmes sont formés à partir de données non étiquetées. Les algorithmes apprennent à identifier les relations entre les variables d'entrée, mais sans être guidés par les étiquettes. Cette technique est utilisée pour le clustering, où les données sont groupées en clusters en fonction de leurs similitudes.

En conclusion, la modélisation par Machine Learning est un outil puissant pour découvrir des relations cachées dans les données et pour améliorer les prévisions en utilisant les données historiques. Les deux types de modélisation ont leurs avantages et leurs limites et le choix dépend du type de problème à résoudre et des données disponibles.

Les étapes importantes

Voici les étapes importantes que vous pouvez inclure dans votre projet de Machine Learning :

1. Étape de compréhension des données : analyse exploratoire des données, visualisation, gestion des valeurs manquantes et de la distribution des données.
2. Prétraitement des données : transformation des features numériques (scaling), encodage des variables catégorielles, sélection des features.
3. Sélection du modèle : comparaison de différents algorithmes de Machine Learning, choix du modèle le plus adapté pour votre cas d'utilisation.
4. Entraînement et évaluation du modèle : entraînement du modèle sur les données d'entraînement, évaluation du modèle sur les données de test, ajustement des hyperparamètres avec la validation croisée et la grille de recherche.
5. Interprétation des résultats et présentation de la solution : interprétation des métriques d'évaluation, visualisation des résultats, présentation de la solution et des recommandations.
6. Déploiement : déploiement du modèle dans un environnement de production pour l'utilisation réelle.

Il est important de souligner que ces étapes ne sont pas strictes et peuvent varier en fonction de la complexité et de la nature de votre cas d'utilisation. Il est également important de documenter soigneusement chaque étape pour une meilleure compréhension et reproduction des résultats.

Choix de données X et de la variable cible y

Sélectionner les features (X) qui seront utilisées pour entraîner le modèle et définir la variable cible (y) que le modèle cherche à prédire.

Voici un exemple de code python pour la préparation des données pour la modélisation en utilisant pandas:

```
import pandas as pd

# Chargement des données dans un DataFrame
df = pd.read_csv('nom_du_fichier.csv')

# Sélection des features (colonnes) pour X
X = df[['feature1', 'feature2', 'feature3']]

# Définition de la variable cible y
y = df['cible']
```

Dans cet exemple, nous utilisons la fonction `train_test_split` de `scikit-learn` pour séparer les données en ensembles d'entraînement et de validation, avec `80%` des données utilisées pour l'entraînement et `20%` pour la validation.

Split / partition des données

Diviser les données en ensemble d'entraînement, de validation et de test. Cette étape est importante pour évaluer la performance du modèle et éviter la fuite de données. Fuite des données)

Voici un exemple de code Python pour spliter les données en ensemble d'entraînement, de validation et de test en utilisant la librairie `scikit-learn`:

```
from sklearn.model_selection import train_test_split

# Séparer les données en entrées et sorties
X = df.drop("target", axis=1)
y = df["target"]

# Diviser les données en ensemble d'entraînement, de validation et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
random_state=42)
```

Dans ce code, nous utilisons la fonction `train_test_split` pour diviser d'abord les données en ensemble d'**entraînement** et de **test**, puis nous divisons l'ensemble d'entraînement en un ensemble d'**entraînement** et de **validation**.

La variable `test_size` définit la taille de l'ensemble de test par rapport au nombre total d'exemples, et la variable `random_state` définit le générateur de nombres aléatoires utilisé pour sélectionner les exemples pour les ensembles d'entraînement et de test.

Transformation des variables

Effectuer des transformations telles que l'encodage des variables catégorielles et la normalisation des variables numériques.

Voici un exemple de code python pour la transformation des variables dans le contexte d'un programme de machine learning :

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

# Chargement des données
data = pd.read_csv("data.csv")

# Sélectionner les colonnes pour les variables X et y
X = data.drop("target_column", axis=1)
y = data["target_column"]

# Transformation des variables numériques
numerical_transformer = StandardScaler()

# Transformation des variables catégorielles
categorical_transformer = OneHotEncoder()

# Définir les colonnes pour les transformations
numerical_cols = X.select_dtypes(include=["float64"]).columns
categorical_cols = X.select_dtypes(include=["object"]).columns

# Application de la transformation de colonnes
preprocessor = ColumnTransformer(
    transformers=[
        ("num", numerical_transformer, numerical_cols),
        ("cat", categorical_transformer, categorical_cols)
    ])

# Créer un pipeline pour l'entraînement
model_pipeline = Pipeline(steps=[("preprocessor", preprocessor),
                                 ("classifier", LogisticRegression())])

model_pipeline.fit(X, y)
```

Dans ce code, nous utilisons StandardScaler pour normaliser les variables numériques et

OneHotEncoder pour encoder les variables catégorielles. Nous utilisons ColumnTransformer pour appliquer les transformations sur les colonnes appropriées. Enfin, nous utilisons Pipeline pour créer un pipeline d'entraînement qui inclut la transformation des données ainsi que l'entraînement du modèle (dans ce cas, une régression logistique).

Validation croisée

Utiliser une validation croisée pour évaluer la performance du modèle et éviter le sur-ajustement / overfitting.

Voici un exemple de code Python utilisant `scikit-learn` pour effectuer la validation croisée sur un classificateur logistique :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Chargement des données
df = pd.read_csv('data.csv')

# Sélection des features et de la variable cible
X = df.drop('target', axis=1)
y = df['target']

# Split des données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Transformation des variables numériques
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialisation et entraînement du modèle de régression logistique
logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)

# Évaluation du modèle en utilisant la validation croisée
scores = cross_val_score(logreg, X_train_scaled, y_train, cv=5)
print("Scores de validation croisée :", scores)
print("Score moyen :", np.mean(scores))
print("Écart-type :", np.std(scores))

# Évaluation du modèle sur l'ensemble de test
print("Score sur l'ensemble de test :", logreg.score(X_test_scaled, y_test))
```

Choix de l'algorithme de modélisation

Déterminer le type de modèle à utiliser, tel qu'un algorithme de régression ou de classification, en fonction des objectifs et des caractéristiques des données.

Pour choisir l'algorithme de modélisation approprié pour un jeu de données donné, vous pouvez suivre les étapes suivantes :

1. Déterminer le type de problème de machine learning: Il peut s'agir d'une tâche de régression ou de classification, selon le but du modèle.
2. Évaluer la quantité et la qualité des données disponibles : la quantité de données et leur qualité peuvent influencer le choix d'un algorithme particulier.
3. Évaluer les caractéristiques des données : les caractéristiques telles que le nombre de variables, le type de variables (numériques, catégorielles), la distribution de la cible, etc. peuvent influencer le choix d'un algorithme.
4. Évaluer les objectifs du modèle : les objectifs tels que la précision, la vitesse d'entraînement, la facilité d'utilisation, etc. peuvent également influencer le choix d'un algorithme.
5. Expérimenter avec différents algorithmes : une fois les facteurs précédents pris en compte, vous pouvez expérimenter avec différents algorithmes pour voir lequel offre les meilleurs résultats pour votre jeu de données.

Après avoir évalué ces facteurs, vous pouvez utiliser des bibliothèques telles que `scikit-learn` pour choisir et entraîner l'algorithme de modélisation approprié pour vos données.

Entraînement du modèle et réglage des Hyperparamètre

Ajuster les hyperparamètres du modèle pour améliorer les résultats et éviter le sur-ajustement.

Pour ajuster les hyperparamètres d'un modèle en utilisant la validation croisée, vous pouvez utiliser la classe `GridSearchCV` ou `RandomizedSearchCV` de scikit-learn.

Ci-dessous est un exemple simple d'utilisation de `GridSearchCV` pour trouver les meilleurs hyperparamètres pour un modèle de régression linéaire en utilisant la validation croisée de 5 plis:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression

# charger les données dans un dataframe
data = pd.read_csv("data.csv")

# sélectionner les features X et la variable cible y
X = data.drop('target', axis=1)
y = data['target']
```

```
# définir le modèle de régression linéaire
model = LinearRegression()

# définir les hyperparamètres à ajuster
param_grid = {'fit_intercept':[True,False], 'normalize':[True,False]}

# définir la validation croisée
grid = GridSearchCV(model, param_grid, cv=5)

# entraîner le modèle en utilisant les hyperparamètres optimisés
grid.fit(X, y)

# afficher les meilleurs hyperparamètres trouvés
print("Meilleurs hyperparamètres: ", grid.best_params_)
```

Choix des métriques d'évaluation

Choisir les métriques d'évaluation appropriées en fonction des objectifs et des caractéristiques des données.

Pour les problèmes de **classification**, les métriques d'évaluation couramment utilisées incluent la `précision`, le `rappel`, le `score F1` et l' `AUC ROC`. Pour les problèmes de **régression**, les métriques d'évaluation couramment utilisées incluent l'`erreur absolue moyenne`, l'`erreur quadratique moyenne`, la `racine carrée de l'erreur quadratique moyenne` et le `score R2`.

En Python, ces métriques d'évaluation peuvent être facilement calculées à l'aide de différentes fonctions du module metrics de scikit-learn.

Évaluation du Modèle Machine Learning supervisé

Pour évaluer un modèle d'apprentissage automatique, il existe plusieurs métriques d'évaluation qui peuvent être utilisées, en fonction du type de problème (par exemple, la classification, la régression) et des exigences spécifiques du problème.

Par exemple, pour calculer la précision d'un modèle de classification, on pourrait utiliser:

```
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Précision:", accuracy)
```

Il est important de choisir les métriques d'évaluation appropriées en fonction du type de problème et des exigences et de prendre en compte plusieurs métriques, pas seulement une, pour avoir une évaluation complète de la performance du modèle.

Amélioration du modèle

En fonction des résultats obtenus, il peut être nécessaire d'apporter des modifications au modèle, telles que l'ajout de nouvelles features, la modification des paramètres du modèle, etc. pour améliorer les résultats.

Déploiement du modèle

Une fois que le modèle est prêt, il peut être déployé pour effectuer des prédictions sur de nouvelles données.

Voici un exemple de code Python pour déployer un modèle de machine learning construit à l'aide de `scikit-learn`:

```
# Importons les modules nécessaires
import pickle
import numpy as np

# Charger le modèle enregistré avec pickle
loaded_model = pickle.load(open("saved_model.pkl", "rb"))

# Préparons les données pour faire des prédictions
new_data = np.array([[6.3, 2.9, 5.6, 1.8]])

# Effectuons les prédictions sur les nouvelles données
prediction = loaded_model.predict(new_data)

print("La prédiction pour les nouvelles données est:", prediction)
```

Dans ce code, nous avons tout d'abord importé les modules nécessaires, `pickle` et `numpy`. Ensuite, nous avons chargé le modèle enregistré avec `pickle` en utilisant la fonction `pickle`.load. Ensuite, nous avons préparé les nouvelles données à prédire en utilisant `numpy`. Finalement, nous avons effectué les prédictions sur les nouvelles données en utilisant la méthode predict du modèle chargé.

Créer une API pour le modèle

Voici un exemple simple de code Python pour déployer un modèle dans une API `Flask` :

```
from flask import Flask, request
import numpy as np
import pickle

app = Flask(__name__)

# Charger le modèle à partir du fichier pickle
model = pickle.load(open('model.pkl', 'rb'))
```

```
# Créer une route pour la prédiction
@app.route('/predict', methods=['POST'])
def predict():
    # Récupérer les données d'entrée
    data = request.get_json()
    # Transformez les données en format numpy
    data = np.array([data['input']])
    # Utiliser le modèle pour faire une prédiction
    prediction = model.predict(data)
    # Renvoyer la prédiction sous forme de réponse JSON
    return {'prediction': prediction.tolist()}

if __name__ == '__main__':
    app.run(port=8000, debug=True)
```

Ce code suppose que vous avez déjà enregistré votre modèle dans un fichier **model.pkl** en utilisant la fonction **pickle.dump**. Il crée une API `Flask` qui attend une entrée sous forme de JSON contenant les données d'entrée, et renvoie une réponse JSON contenant la prédiction du modèle.

Notez que vous pouvez adapter ce code en fonction de vos besoins en termes de validation de données, de gestion d'erreurs, de sécurité, etc. Il s'agit simplement d'un point de départ pour déployer un modèle avec Flask.

Code Python ML supervisé sur des données structurées

Exemple simple

Cet exemple est écrit en Python avec l'utilisation du framework scikit-learn :

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Chargement des données
data = pd.read_csv("data.csv")

# Séparation des features et de la target
X = data.drop("target", axis=1)
y = data["target"]

# Séparation en données d'entrainement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Initialisation du modèle
model = LinearRegression()
```

```
# Entrainement du modèle sur les données d'entrainement
model.fit(X_train, y_train)

# Prédiction sur les données de test
y_pred = model.predict(X_test)

# Calcul de l'erreur quadratique moyenne
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error : ", mse)
```

Cet exemple montre comment charger des données à partir d'un fichier CSV, les séparer en données d'entrainement et de test, initialiser un modèle de régression linéaire, l'entrainer sur les données d'entrainement, faire des prédictions sur les données de test et enfin évaluer les performances du modèle en utilisant l'erreur quadratique moyenne.

Exemple complet

Voici un exemple complet en Python utilisant scikit-learn pour la prédiction de la régression avec des caractéristiques numériques et catégorielles. Le code inclut également la normalisation des données, le tuning des hyperparamètres, la validation croisée et le pipeline. Cet exemple utilisera le jeu de données titanic pour la prédiction de la survie des passagers.

```
# Importation des bibliothèques
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV

# Chargement des données
data = pd.read_csv("titanic.csv")

# Séparation des variables cibles et prédicteurs
X = data.drop(["Survived"], axis=1)
y = data["Survived"]

# Définition des colonnes numériques et catégorielles
numerical_cols = ["Age", "Fare"]
categorical_cols = ["Pclass", "Sex", "Embarked"]

# Transformation des colonnes
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
```

```
    ])

# Construction du pipeline
pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

# Séparation des données en données d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Optimisation des hyperparamètres
param_grid = {'classifier__C': [0.1, 1, 10], 'classifier__penalty': ['l1', 'l2']}
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)

# Évaluation de la performance du modèle
print("Meilleur score de validation croisée: {:.2f}".format(grid.best_score_))
print("Meilleurs paramètres: ", grid.best_params_)

# Évaluation sur les données de test
print("Score sur les données de test: {:.2f}".format(grid.score(X_test, y_test)))
```

Dans ce code, nous avons utilisé la classe `ColumnTransformer` pour traiter séparément les colonnes numériques et catégorielles. Les colonnes numériques sont normalisées avec `StandardScaler` et les colonnes catégorielles sont codées en variables binaires avec `OneHotEncoder`.

Ensuite, nous avons construit un pipeline en utilisant `Pipeline` qui inclut la transformation des colonnes et l'entraînement du modèle `LogisticRegression`.

En utilisant `GridSearchCV`, nous avons effectué une recherche de grille pour trouver les meilleurs hyperparamètres pour le modèle `LogisticRegression`. Nous avons défini un ensemble de valeurs à tester pour les hyperparamètres et `GridSearchCV` a effectué une validation croisée pour trouver le meilleur ensemble d'hyperparamètres. Enfin, nous avons entraîné le modèle `LogisticRegression` sur l'ensemble de données complet en utilisant les hyperparamètres optimaux trouvés par `GridSearchCV`.

Concepts importants

Difference entre .fit vs .transform

Lors de l'entraînement d'un modèle de machine learning, le modèle doit être d'abord ajusté (fit) aux données d'entraînement (X_train). Ensuite, ce modèle ajusté est utilisé pour transformer (transform) les données d'entraînement et les données de validation (X_valid). Cependant, il est important de ne pas réentrainer (fit) le modèle aux données de validation, car cela pourrait entraîner une fuite de données et altérer les résultats d'évaluation.

Le modèle mentionné dans l'exemple peut être un codage "OneHotEncoding" ou un régressseur logistique, mais il peut être n'importe quel autre modèle de machine learning en fonction des besoins et des données.

Fuite des données

La fuite de données se produit lorsque des informations du jeu de données de test sont accidentellement introduites dans le modèle lors de son entraînement. Cela peut se produire si le jeu de données de test n'est pas correctement séparé du jeu de données d'entraînement et que des informations provenant du jeu de données de test sont utilisées pour entraîner le modèle, ce qui peut donner l'impression d'une performance plus élevée du modèle que ce n'est réellement le cas.

Pour éviter la fuite de données, il est important de séparer correctement les données d'entraînement et de test en utilisant des méthodes telles que la stratification ou la répartition aléatoire, et de ne pas effectuer de transformations sur les données de test qui sont basées sur les informations provenant des données d'entraînement. Il est également important de faire attention aux pipelines de prétraitement et de veiller à ce que les transformations soient effectuées uniquement sur les données d'entraînement et non sur les données de test.

Transformation de variable avec le logarithme

L'application du logarithme peut être utilisée pour transformer une variable qui a une distribution exponentielle ou qui est fortement skewed (asymétrique) dans le but de l'ajuster à une distribution normale.

Lorsqu'une variable est transformée avec un logarithme, le modèle de prédiction peut être entraîné sur la variable transformée, ce qui peut entraîner des résultats de prédiction plus précis en raison de la distribution normalisée des données. Une fois que les prédictions ont été effectuées, il peut être nécessaire de retirer le logarithme pour obtenir des prédictions dans la plage d'origine de la variable. Cela peut être accompli en utilisant l'opération exponentielle (exp).

En résumé, l'application du logarithme, la prédiction avec un modèle de machine learning et le retrait du logarithme sont une technique courante pour ajuster les données pour la modélisation en machine learning.