Car_ case_study

In dieser Übung wird die lineare Regression als grundlegende Methode des maschinellen Lernens eingeführt. Ziel ist es, anhand des Auto-MPG-Datensatzes die Kraftstoffeffizienz / Beschleunigung etc. von Autos aus den 1970er und 1980er Jahren vorherzusagen. Dabei werden Sie lernen, Daten zu explorieren, vorzubereiten und mit einem Modell zu trainieren sowie die Ergebnisse zu evaluieren. Außerdem werden Cross-Validation und Workflows thematisiert, um die Konsistenz und Wiederholbarkeit der Modellierung zu verbessern.

Bitte lösen Sie die folgenden Aufgaben Schritt für Schritt und orientieren Sie sich dabei an den bereitgestellten Code-Snippets.

Hinweise und Erklärungen sind enthalten, um euch bei jedem Schritt zu unterstützen.

1. Datenexploration und Bereinigung

Frage: Importiert den Datensatz `Sample_car_dataset.csv` und zeigt die ersten 10 Zeilen an. Welche Attribute sind im Datensatz enthalten?
Hinweis:

```
R
KopierenBearbeiten
library(tidyverse)

# Daten einlesen und ansehen
data <- read_csv("Sample_car_dataset.csv")
head(data, 10)
```

Erwartete Ausgabe: Eine Tabelle mit den ersten 10 Zeilen. Prüft Spalten wie `mpg`, `horsepower`, `weight`, `car_name`, etc.

Frage: Untersuchen Sie den Datensatz auf fehlende Werte. Welche Spalten haben fehlende Werte?
Hinweis:

```
R
KopierenBearbeiten
library(DataExplorer)
plot_missing / skim()
```

Zusätzliche Erklärung: Fehlende Werte treten auf, wenn Informationen in bestimmten Feldern fehlen (z. B. PS). Es ist wichtig, diese zu behandeln, um Fehler im Modell zu vermeiden.

Frage: Führen Sie eine Imputation der Spalte `horsepower` durch. Welche Methode würden Sie wählen?

Erklärung: **Imputation** bedeutet, fehlende Werte durch plausible Werte zu ersetzen. Methoden wie "Mean Imputation" (Durchschnitt), "Median Imputation" oder modernere Techniken wie "Predictive Mean Matching" (PMM) können verwendet werden.

Warum PMM?: PMM nutzt ähnliche vorhandene Werte im Datensatz, um realistischere Imputationen durchzuführen, statt den Mittelwert zu nehmen.

Hinweis: Bibliothek `mice` für die Imputation:

```
R
KopierenBearbeiten
library(mice)

# Imputation durchführen
imputed_data <- mice(data, m = 1, method = "pmm", seed = 123)

# Vollständigen Datensatz extrahieren
data_complete <- complete(imputed_data)
```

Zusatzfrage: Warum ist PMM besser als einfache Mittelwertimputation?

---

2. Datenaufteilung und Verwendung von Rezepten

Frage: Warum ist es wichtig, die Daten in Trainings- und Testdaten aufzuteilen? Teilt die Daten im Verhältnis 70:30 auf.

Erklärung: Die Aufteilung trennt den Datensatz in zwei Teile: **Trainingsdaten**, um das Modell zu trainieren, und **Testdaten**, um die Leistung zu evaluieren. Dadurch vermeiden wir Overfitting, d. h., dass das Modell nur auf Trainingsdaten gut funktioniert, aber auf neuen Daten versagt.

Hinweis: Funktion `initial_split()` aus `tidymodels`:

```
R
KopierenBearbeiten
library(tidymodels)

# Daten aufteilen
set.seed(123)
split <- initial_split(data_complete, prop = 0.7)
train_data <- training(split)
test_data <- testing(split)
```

Frage: Was sind Rezepte (`recipes`) und warum werden sie verwendet?

Erklärung: Rezepte in R (aus dem `recipes`-Paket) sind Werkzeuge zur Vorbereitung von Daten. Sie enthalten **alle Schritte**, um Rohdaten in ein Format zu bringen, das für maschinelles Lernen geeignet ist.

Typische Schritte eines Rezepts sind:

- Entfernen irrelevanter Spalten
- Normalisierung numerischer Variablen
- Erstellung von Dummy-Variablen für kategoriale Variablen
- Umgang mit fehlenden Werten

Beispiel für ein Rezept:

```
R
KopierenBearbeiten
recipe_prep <- recipe(AV ~ UV, data = train_data) %>%
  step_rm(car_name) %>%                  # Entfernt irrelevante Spalte
  step_normalize(all_numeric(), -all_outcomes()) %>% # Normalisiert
numerische Variablen
  step_dummy(all_nominal(), -all_outcomes()) # Erstellt Dummy-
Variablen
```

Frage: Warum ist es wichtig, numerische Variablen zu normalisieren und kategoriale Variablen in Dummies umzuwandeln?

Frage: Was ist der Unterschied zwischen `lm(prepped_recipe, ...)` und `train(recipe, ..., method="lm")`?

Erklärung:

- Mit `lm(prepped_recipe, ...)` erstellt ihr ein **lineares Regressionsmodell** direkt aus einem vorbereiteten Datensatz. Das Rezept wird vorher angewandt.
- Mit `train(recipe, ..., method="lm")` wird das Rezept direkt in den Modell-Workflow integriert. Dabei erfolgt das Training und die Datenvorbereitung in einem Schritt, was einfacher ist und weniger Fehleranfälligkeit bietet.

Beispiel:

```
R
KopierenBearbeiten
# Direkt mit lm
prepped <- prep(recipe_prep, training = train_data)
train_processed <- bake(prepped, new_data = NULL)
model <- lm(mpg ~ ., data = train_processed)

# Mit train()
model_with_recipe <- train(recipe_prep, data = train_data, method =
"lm")
```

Frage: Welche Methode würdet ihr bevorzugen und warum?

3. Modelltraining und Evaluation

Frage: Erstellen Sie ein lineares Regressionsmodell und berechnet den RMSE auf den Testdaten.

Hinweis: Nutzen Sie den vorherigen Code, um das Modell zu trainieren. Führen Sie dann Vorhersagen durch:

```
R
KopierenBearbeiten
# Vorhersagen auf Testdaten
predictions <- predict(model, new_data = test_data)

# RMSE berechnen
rmse <- sqrt(mean((test_data$mpg - predictions)^2))
print(rmse)
```

Zusatzfrage: Was sagt der RMSE über die Modellgüte aus?

---

4. Cross-Validation und Modellvergleich

Frage: Führen Sie eine 5-fache Cross-Validation durch. Warum ist diese Methode besser/ schlechter?

Erklärung: Cross-Validation teilt die Trainingsdaten mehrfach auf (z. B. in 5 Teile) und trainiert das Modell auf 4 Teilen, während der fünfte zur Validierung genutzt wird. Dies wiederholt sich für alle Teile und liefert robustere Ergebnisse.

Hinweis: Verwendet:

```
R
KopierenBearbeiten
control <- trainControl(method = "cv", number = 5)
model_cv <- train(recipe_prep, data = train_data, method = "lm",
trControl = control)
print(model_cv)
```

---

5. Modell speichern und wiederverwenden

Frage: Speichern Sie bitte das Modell und laden es neu, um Vorhersagen zu erstellen.

Hinweis:

```
R
KopierenBearbeiten
# Modell speichern
saveRDS(model, "linear_model.rds")

# Modell laden
loaded_model <- readRDS("linear_model.rds")
predict(loaded_model, new_data = test_data)
```

Warum ist es wichtig, Modelle zu speichern?