Multiple Lineare Regression

Multiple Lineare Regression ermöglicht die Abbildung eines linearen Einflusses mehrerer Parameter auf einen Zielwert. Damit eignet sie sich hervorragend als Methode für die Erstellung von Energieverbrauchsprognosen. in dieser Aufgabe soll zuerst mit Trainingsdaten ein Regressionsmodell erstellt und anschließend dessen Güte mit Testdaten überpüft werden.

Aufgabe 1: Daten laden

Zuerst wird die Datei `features.csv` mit den Daten mit Hilfe der Funktion `readtable` geladen. Dann werden die Timestamps und die Features separat gespeichert und mit der Funktion `table2array` zu Arrays konvertiert.

```
clearvars;
clc;

data = readtable('features.csv', 'Delimiter', ',', 'TreatAsEmpty', 'true');
timestamps = table2array(data(:, 1));
features = table2array(data(:, 2 : end));
```

Aufgabe 2: Aufteilung Trainings- und Test-Daten

In diesem Schritt werden die Daten in Trainingsdaten zum Erstellen und in Testdaten zum Testen des Regressionsmodells unterteilt. Als Trainingsdaten werden die ersten 729 Zeilen verwendet da diese einen Zeitraum von zwei Jahren abdecken. Außerdem werden die Zielwerte (1. Spalte) von den Features getrennt. Es sollen also die Arrays `training_targets`, `training_features`, `test_targets` und `test_features` vorliegen. Zusätzlich sollen als Vorarbeit für folgende Schritte auch die Timestamps in `training_timestamps` und `test_timestamps` aufgeteilt werden.

```
training_targets = features(1 : 729, 9);
training_features = features(1 : 729, [1 : 8, 10 : end]);
training_timestamps = timestamps(1 : 729, :);
test_targets = features(730 : end, 9);
test_features = features(730 : end, [1 : 8, 10 : end]);
test_timestamps = timestamps(730 : end, :);
```

Aufgabe 3: Regressionsmodell erstellen

Die linearen Einflüsse der Features auf den Zielwert werden durch die folgende lineare Modellfunktion beschrieben:

$$Q(t+1) = x(t) \cdot p = p_0 + p_1 \cdot Q(t) + p_2 \cdot Q(t-1) + \ldots + p_{14} \cdot Q(t-13)$$
$$+ p_{15} \cdot T(t) + p_{16} \cdot T(t-1) + \ldots + p_{28} \cdot T(t-13)$$
$$+ p_{29} \cdot S(t+1)$$

$$+p_{30} \cdot W(t+1)$$

wobei folgende Notationen gelten:

$p = [p_0, \dots, p_30]^T$,

$x(t) = [1, Q(t), \dots, Q(t-13), T(t), \dots, T(t-13), S(t+1), W(t+1)]$,

$Q(t) =$ Wärmeverbrauch von Tag t,

$T(t) =$ Außentemperatur von Tag t,

$S(t) =$ Saisonkomponente von Tag t,

$W(t) =$ Werktagsindikator von Tag t

Da Prognosemodelle im Allgemeinen nicht lineare Einflüsse besitzen und Daten Messfehler behaftet sind, ist es nicht möglich, p so zu wählen, dass in der Modellfunktion für alle t Gleichheit gilt. Folglich handelt es sich um ein Optimierungsproblem bei dem p so gewählt wird, dass $\sum_{t=1}^{m}(Q(t+1) - x(t) \cdot p)^2$ minimal ist.

Es sei $X = \begin{bmatrix} x(1) \\ \vdots \\ x(m) \end{bmatrix} \in \mathbb{R}^{m \times 30}$ und $Q = \begin{bmatrix} Q(1+1) \\ \vdots \\ Q(m+1) \end{bmatrix} \in \mathbb{R}^m$.

Dann kann das optimale p^* durch die Moore-Penrose Pseudoinverse $X^{\#}$, die in Matlab mit dem Befehl `pinv(X)` bestimmt werden kann, wie folgt berechnet werden:

$p^* = X^{\#} \cdot Q$.

Hinweis: Als Faktor für p_0 wird der Features-Matrix eine 1-Spalte hinzugefügt

```
X = [ones(size(training_features, 1), 1) training_features];
p = pinv(X) * training_targets;
```

Aufgabe 4: Regressionsmodell testen

Zum Testen des soeben errechneten Regressionsmodells werden die zu Anfang von den Trainingsdaten getrennten Testdaten verwendet.

```
X = [ones(size(test_features, 1), 1) test_features];
output = X * p;
```

Um die Güte der Prognose zu bestimmen werden die durch die Regression prognostizierten Zielwerte mit den tatsächlichen Zielwerten verglichen. Hierfür bieten sich verschiedene Metriken an. An dieser Stelle soll der RMSE (Rooted Mean Square Error) verwendet werden: $RMSE = \sqrt{\frac{\sum_{t=1}^{n}(y(t) - \hat{y}(t))^2}{n}}$

Hinweise:

- zur Subtraktion zweier Matrizen kann die Funktion `gsubtract` verwendet werden
- für das Quadrat zweier Matrizen kann der Operator `.^` verwendet werden

```
RMSE = sqrt(mean(gsubtract(output, test_targets).^2))
```

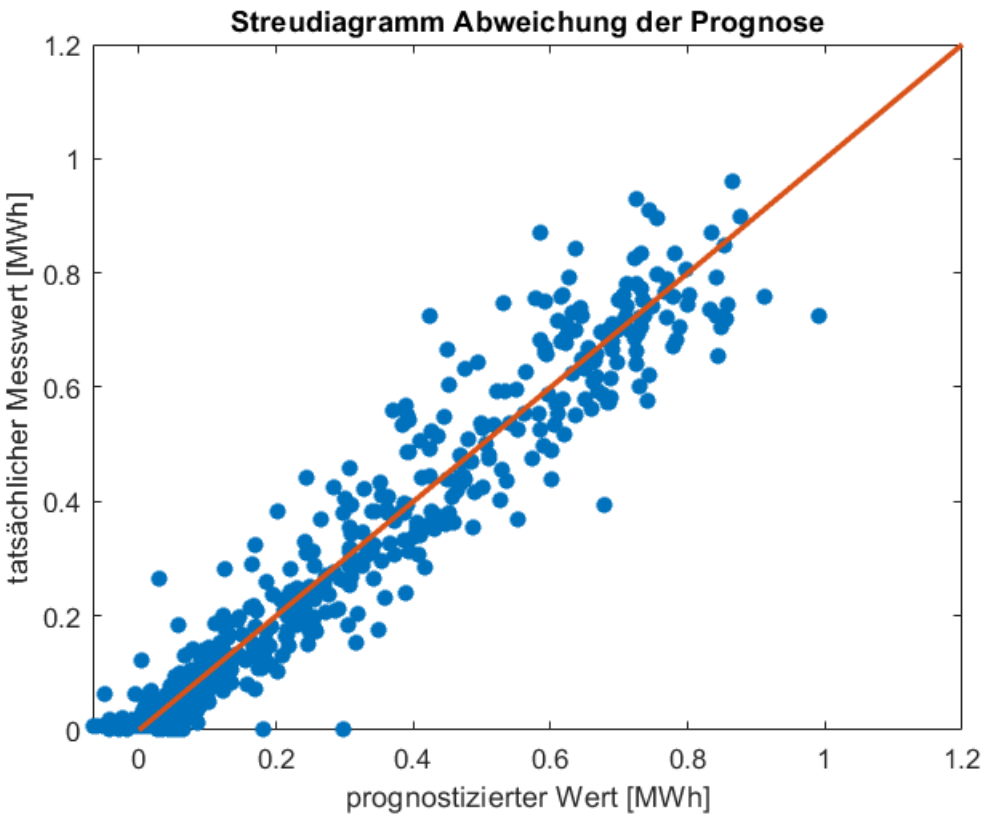
```
RMSE = 0.0704
```

Aufgabe 5: Ergebnisse visualisieren

Für einen visuellen Vergleich der Testergebnisse können verschiedene Plots erstellt werden.

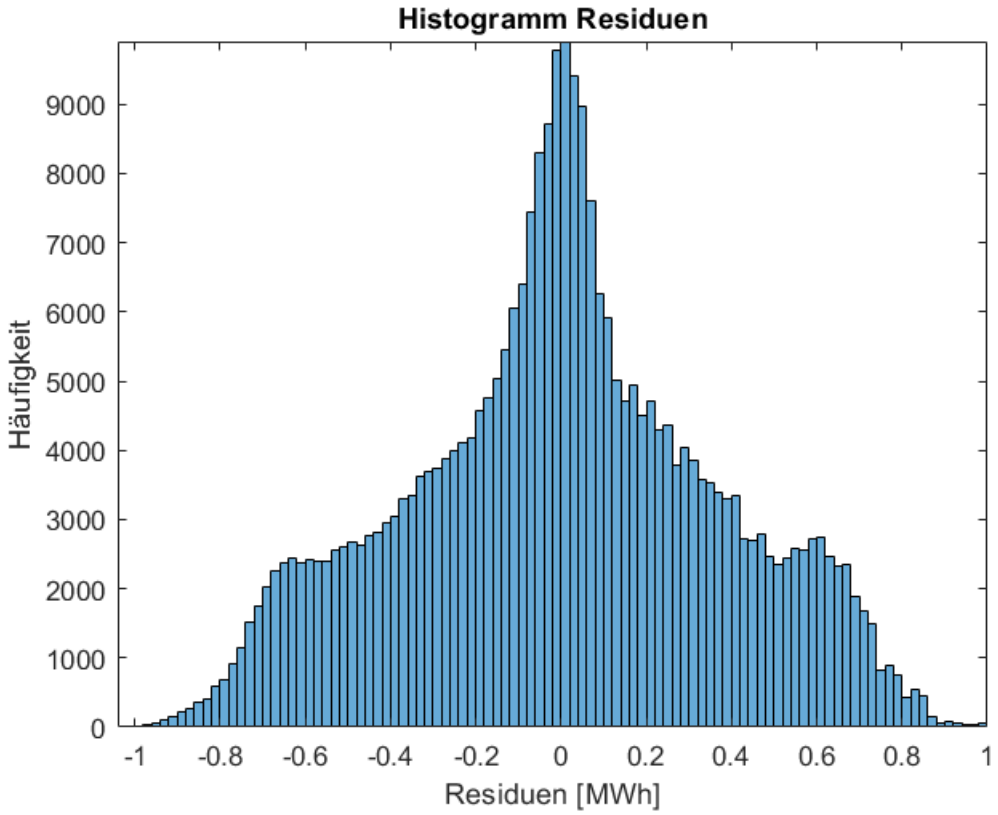
- Ein Streudiagramm mit den prognostizierten Verbräuchen auf der x-Achse und den tatsächlichen Verbräuchen auf der y-Achse:

```
figure
plot(output, test_targets, '*')
hold on
plot(0 : 0.001 : 1.2, 0 : 0.001 : 1.2)
title('Streudiagramm Abweichung der Prognose')
xlabel('prognostizierter Wert [MWh]')%, 'FontSize', 32)
ylabel('tatsächlicher Messwert [MWh]')%, 'FontSize', 32)
%set(gca, 'FontSize', 32)
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
axis tight
hold off
```



- Ein Histogramm der Residuen:

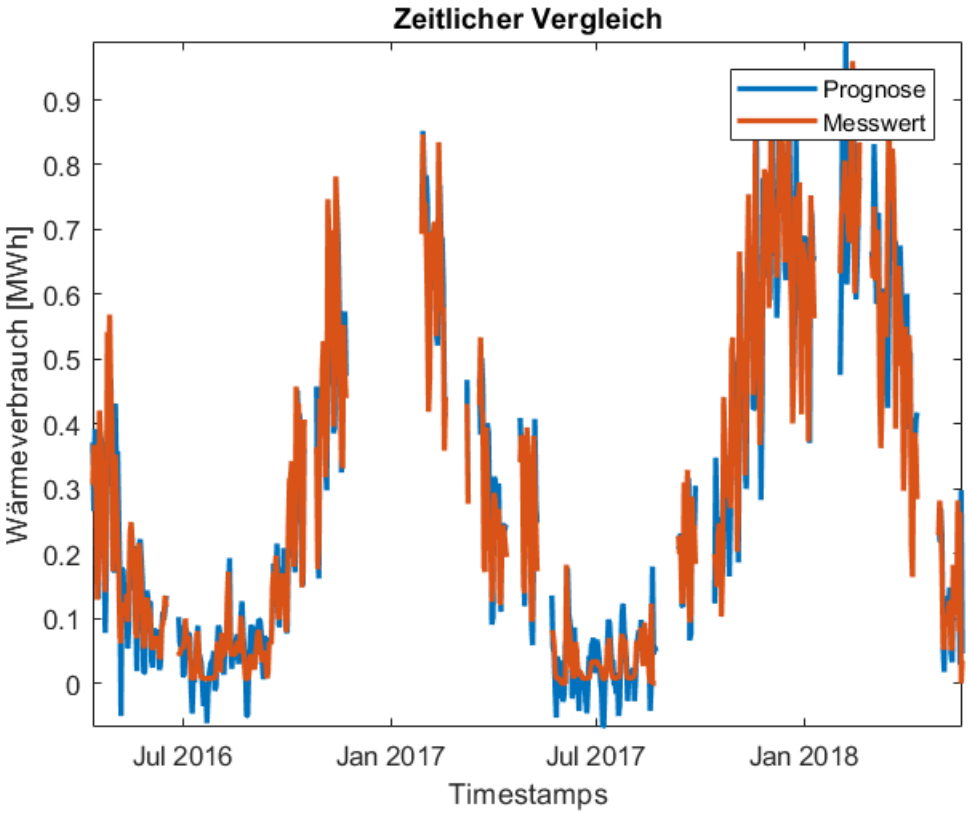
```
figure
histogram(output.' - test_targets, 'BinWidth', 0.02)
hold on
title('Histogramm Residuen')
xlabel('Residuen [MWh]')%, 'FontSize', 32)
ylabel('Häufigkeit')%, 'FontSize', 32)
%set(gca,'FontSize', 32)
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
axis tight
hold off
```



- Ein Liniendiagramm welches den Verlauf der prognostizierten Werte sowie der tatsächlichen Werte über den gesamten Zeitraum zeigt:

Hinweis: um einen schönen Plot zu erhalten müssen die Datenlücken in `output` und in `test_targets` zuvor mit `NaN` gefüllt werden. Dazu müssen diese zuerst zu einer `timetable` konvertiert werden und diese dann an die Funktion `retime(tt, 'daily')` übergeben werden.

```
output_gaps = retime(timetable(test_timestamps, output), 'daily');
test_targets_gaps = retime(timetable(test_timestamps, test_targets), 'daily');
figure
plot(output_gaps.Properties.RowTimes, output_gaps{:, 1})
hold on
plot(test_targets_gaps.Properties.RowTimes, test_targets_gaps{:, 1})
title('Zeitlicher Vergleich')
xlabel('Timestamps')%, 'FontSize', 32)
ylabel('Wärmeverbrauch [MWh]')%, 'FontSize', 32)
lgd = legend('Prognose', 'Messwert');
%lgd.FontSize = 32;
%set(gca, 'FontSize', 32)
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
axis tight
hold off
```



Aufgabe 6: Underfitting

Jetzt sollen Aufgaben 1 bis 5 mit dem Datensatz `features_einzelraeume.csv` wiederholt werden. Als Trainingsdaten sollen die ersten 172 Zeilen verwendet werden.

Was fällt beim Vergleich der Ergebnisse auf und wie lässt sich dieser Effekt erklären?

Antwort: Die Prognose wird viel schlechter, da viel zu viele Dimensionen und gleichzeitig viel zu wenige Samples vorliegen. (Underfitting!)

```
clearvars;
clc;

data = readtable('features_einzelraeume.csv', 'Delimiter', ',', 'TreatAsEmpty', 'true');
timestamps = table2array(data(:, 1));
features = table2array(data(:, 2 : end));

training_targets = features(1 : 172, 9);
training_features = features(1 : 172, [1 : 8, 10 : end]);
training_timestamps = timestamps(1 : 172, :);
test_targets = features(173 : end, 9);
test_features = features(173 : end, [1 : 8, 10 : end]);
test_timestamps = timestamps(173 : end, :);

X = [ones(size(training_features, 1), 1) training_features];
p = pinv(X) * training_targets;
```

```
X = [ones(size(test_features, 1), 1) test_features];
output = X * p;

RMSE_einzelraeume = sqrt(mean(gsubtract(output, test_targets).^2))
```

```
RMSE_einzelraeume = 0.2307
```