Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML - Supervised Learning
3 Regression
3.2 Klassisch

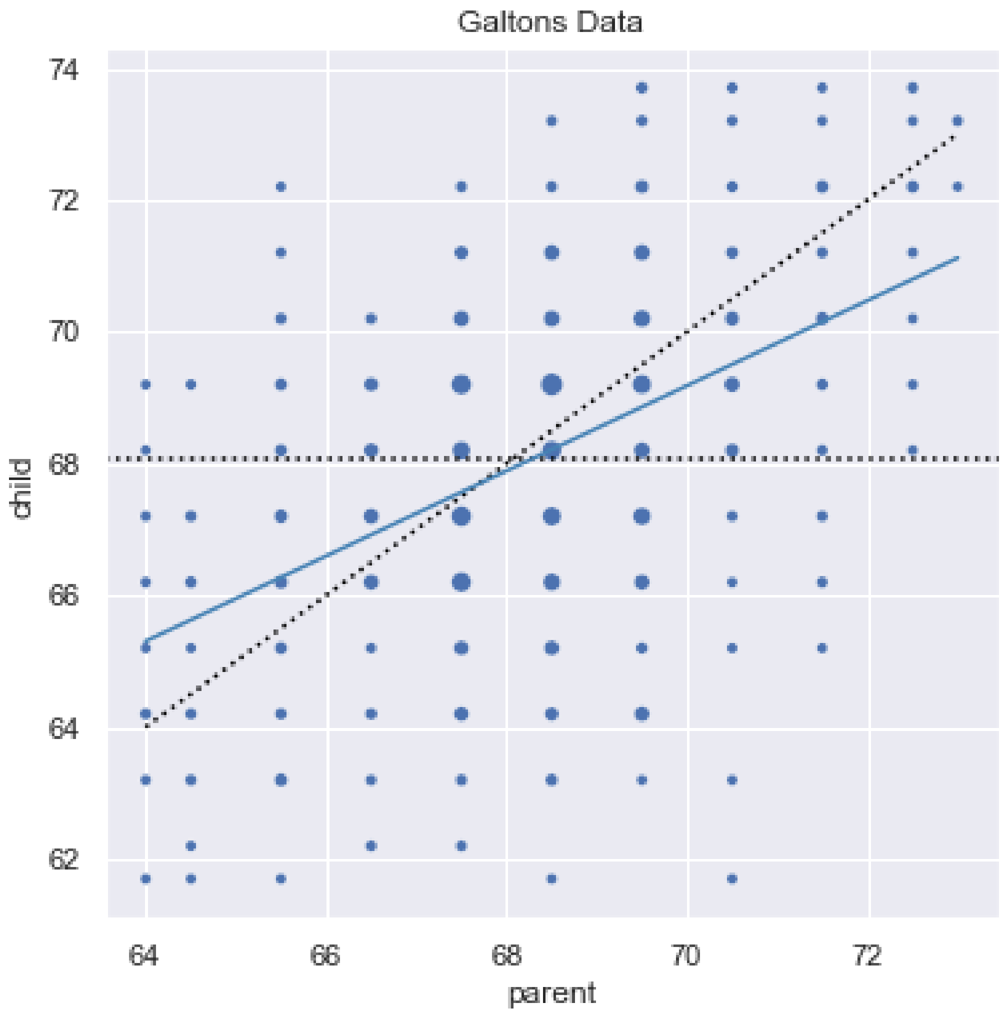
Werner Dähler 2024

3 Regression - AGENDA

3.2 Regression - Regression klassisch

- "klassische" Methode:
  bereits im Einsatz lange vor Machine Learning

- der Begriff *Regression* wurde im 19. Jahrhundert von Francis Galton, geprägt
- beschrieb damit ein biologisches Phänomen, bekannt als *Regression zur Mitte*, wonach Nachfahren grosser Eltern dazu tendieren, nur durchschnittlich gross zu werden
- für Galton hatte der Begriff Regression nur diese biologische Bedeutung [Wikipedia]

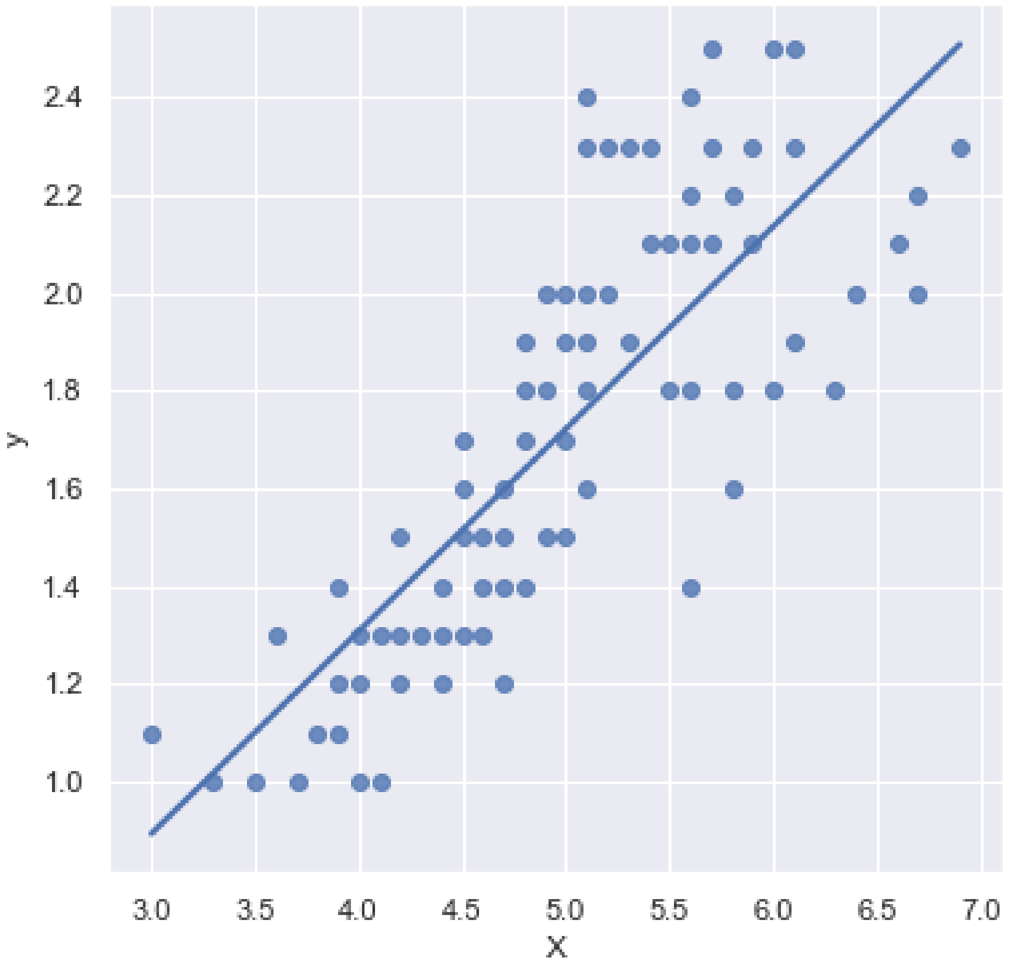


3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

- nebenstehende Darstellung zeigt als Scatterplot den nummerischen Zusammenhang zwischen den eingangs genannten Merkmalen X und y des Demo Datasets
- **Konvention** bei derartigen Darstellungen:
  - X-Achse: Feature (unabhängige Variable)
  - Y-Achse: Target (abhängige Variable) quantitative Beschreibung des Zusammenhangs liefert der Korrelationskoeffizient, der aber in diesem Kontext von untergeordneter Bedeutung ist

  (dies impliziert, bei der statistischen Anwendung der Methode, dass X die Ursache ist, y die Wirkung, welche erzielt wird: geklärte Kausalität)



3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

- die eingezeichnete Linie wird als Regressionsgerade bezeichnet
- sie bezeichnet die beste mögliche (lineare) Anpassung (fit) von X und y
- anhand dieser Geraden kann für jeden beliebigen Wert des Features X der wahrscheinlichste Wert des Targets y vorausgesagt werden

- die Regressionsgerade ist in diesem Beispiel damit ein **prädiktives Modell**, d.h. ein Modell zur Vorhersage der Werte von y aufgrund jener von X

3.2.1.1 Theorie

Vorgehen

- eine Gerade kann in der Geometrie mit der folgenden linearen Beziehung beschrieben werden:

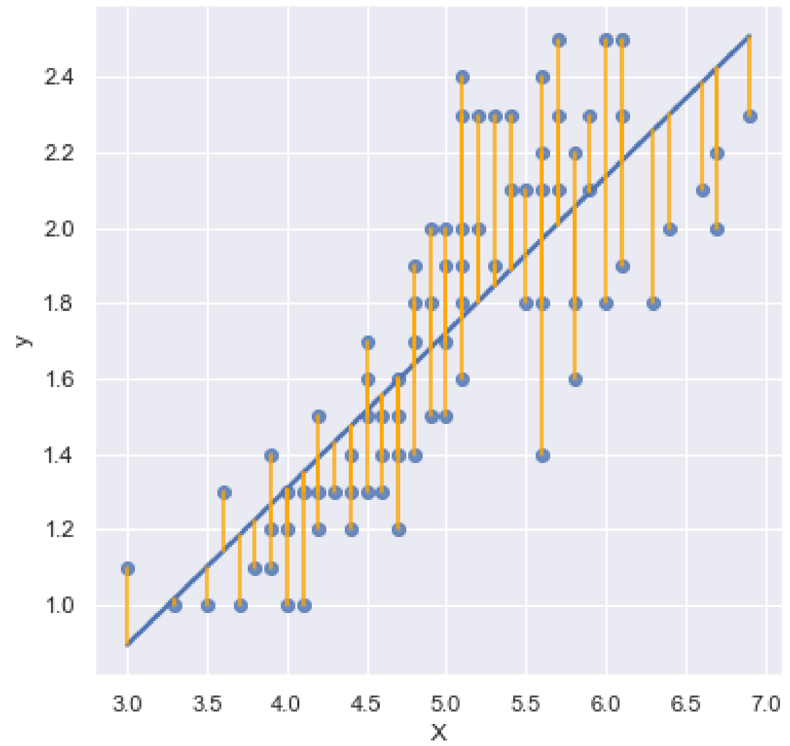
  $y = ax + b$

  übertragen auf das vorliegende Beispiel sowie auf die Schreibweise in der Statistik:

  $y = \beta_0 + \beta_1 \cdot X$

  dabei bedeuten

  - β_0 (b): der Nullstellendurchgang (intercept), d.h. der Wert von y an der Stelle X=0
  - β_1 (a): die Steigung der gesuchten Geraden: $\frac{\Delta y}{\Delta X}$



3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

- β_0 und β_1 werden so gesucht, dass die Summe der **quadrierten vertikalen** Abstände der einzelnen Punkte zur Geraden minimal wird
- die Abstände selber werden als "Residuen" (auch Fehler) bezeichnet (vgl. orange Linien)
- das oben genannte Verfahren wird in der Literatur auch als Methode der kleinsten Quadrate bezeichnet (**Ordinary least squares, OLS**)
- die Lineare Regression ist ein geschlossenes Verfahren, d.h. es existieren Formeln, mit denen β_0 und β_1 direkt aus den Daten berechnet werden können
- Formeln (1 Feature)

$$\beta_1 = \frac{\sum(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$$

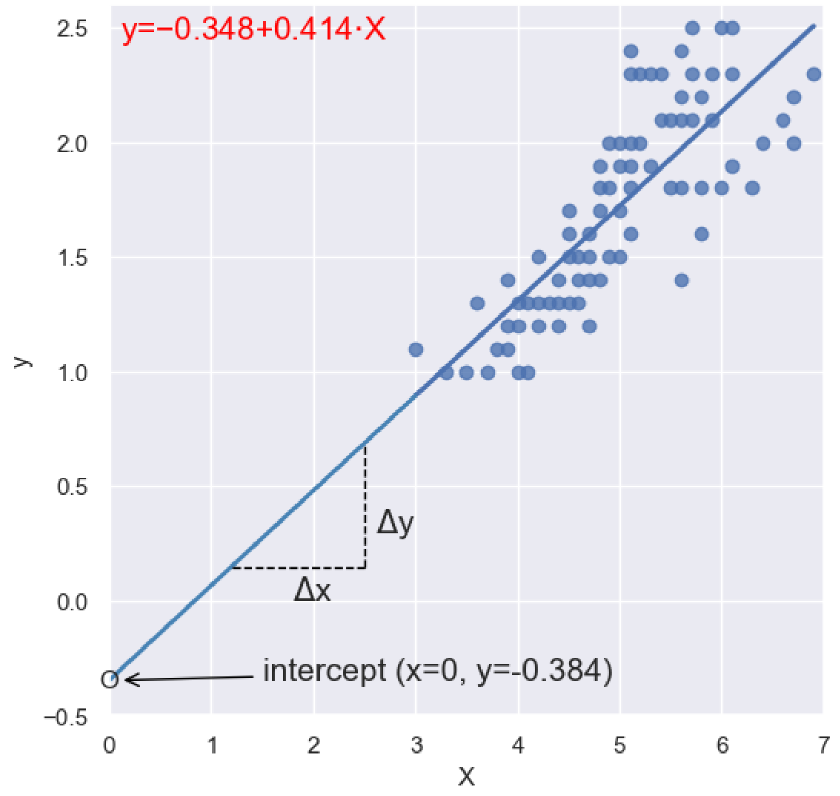
(Details dazu allenfalls hier)

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

- die Herleitung der Formeln ersparen wir uns hier, wir werden im Folgenden mit scikit-learn Methoden arbeiten, welche die gesuchten Koeffizienten direkt ermitteln (und noch einiges mehr)

- in der nebenstehenden Abbildung sind die gesuchten Parameter bereits eingetragen: β_0 =-0.348, β_1 =0.414
- (die Grafik wurde so parametrisiert, dass die X-Achse bei 0 beginnt, damit kann der Wert für β_0 dort direkt abgelesen werden)



3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie (Extra)

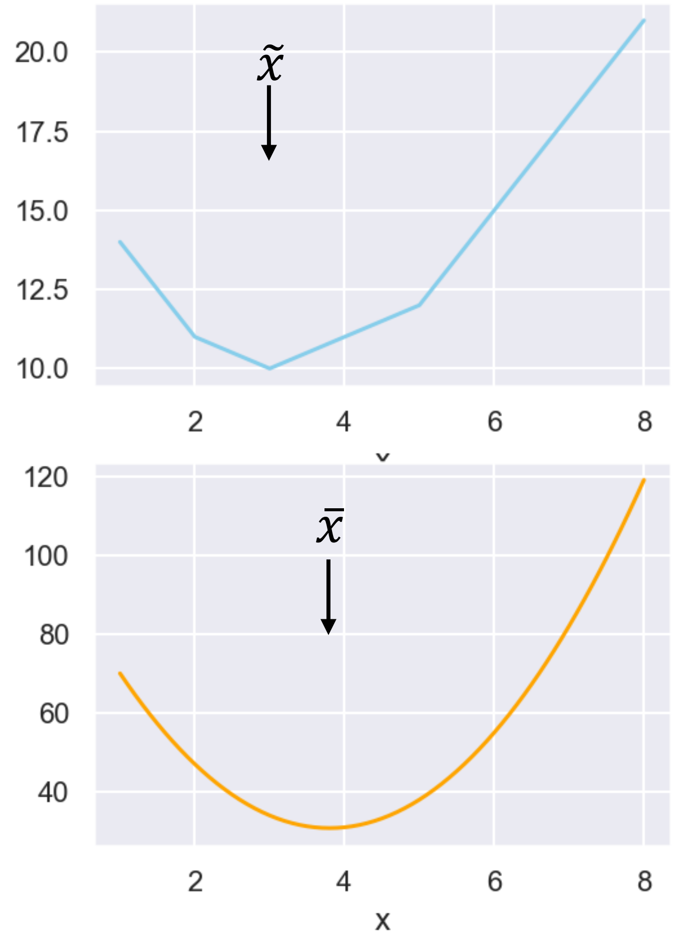
wieso eigentlich den quadratischen Fehler minimieren und nicht den absoluten?

- dazu ein kleines konstruiertes Zahlenbeispiel zur Illustration:
- gegeben ist ein Vektor mit den Werten 1, 2, 3, 5, 8
- gesucht der Wert x so, dass die Summe der **absoluten Abstände** minimal wird (z.B. optimaler Standort eines Marktfahrers an einer Strasse, mit potentiellen Kunden an den genannten Adressen)
- durch Ausprobieren verschiedener Werte von x findet man den Wert 3 als besten Wert
  - zwar gerade der Median der Ausgangswerte, ist aber nicht trivial, mathematisch zu "beweisen"
- alternativer Vorschlag: minimieren der **quadrierten Abstände** führt zu einer quadratischen Funktion

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie (Extra)

- im Gegensatz zum ersten Ansatz kann der zweite direkt (d.h. ohne Ausprobieren) gelöst werden
  - geschlossene Lösung
  - mittels Methoden der Linearen Algebra
- die nebenstehende Darstellung visualisiert diese beiden Ansätze
  - oben: minimieren der Summe der absoluten Abstände
    - der Funktionsgraph weist Ecken auf (Singularitäten) und kann daher nicht differenziert werden, d.h.: auf den Ecken kann die Steigung nicht eindeutig bestimmt werden
  - unten: minimieren der Summe der quadrierten Abstände
    - die Funktion ist differenzierbar
    - die erste Ableitung einer quadratischen Funktion ist eine Gerade
    - Nullsetzen dieser Geraden markiert gerade das Maximum Ausgangsfunktion (klassisches Optimierungsverfahren)



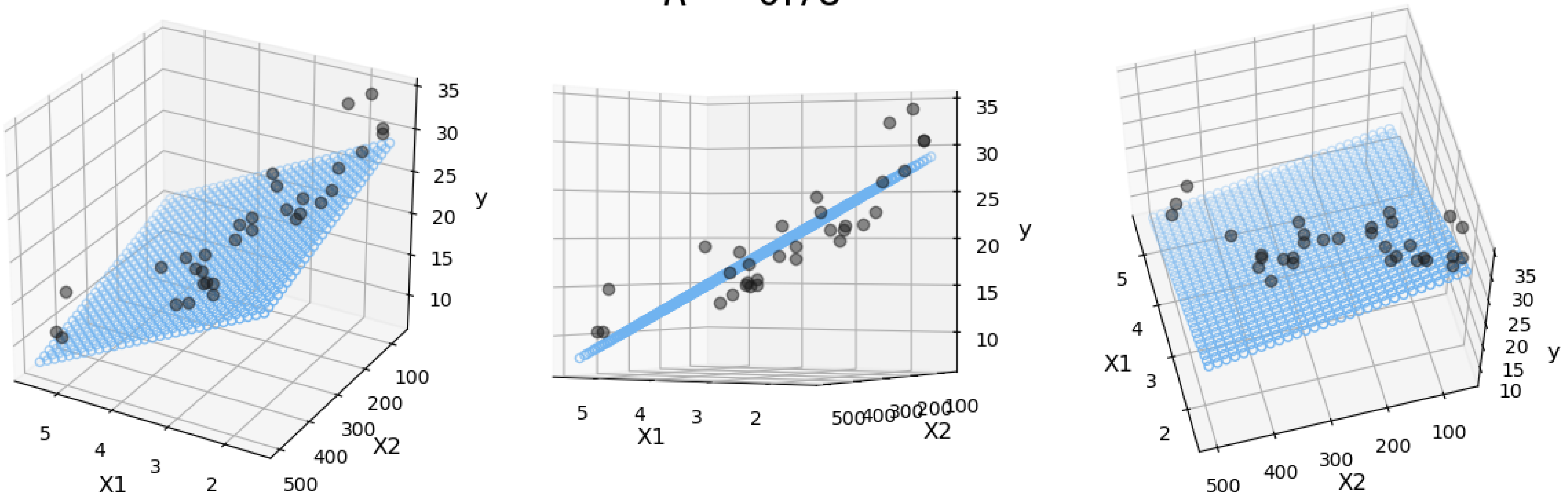
3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

Multiple Lineare Regression

- Lineare Regression wird im Machine Learning dann besonders interessant, wenn mehrere Features gleichzeitig berücksichtigt werden
- bei beispielsweise zwei Features, kann man sich das Regressionsmodell als Ebene im Raum vorstellen, so dass alle Datenpunkte der Trainingsdaten minimale (vertikale!) Abstände zu dieser Ebene aufweisen

$R^2 = 0.78$



3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

Multiple Lineare Regression

- bei mehr als zwei Features wäre es dann eine n-1-dimensionale Hyperebene im n-dimensionalen Raum
- dabei wird folgendes berechnet
  - die Koeffizienten für jedes Feature (β_j)
  - intercept (β_0)

- Modellbeschreibung $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_p x_p$

$$= \beta_0 + \sum_{j=1}^{p} (\beta_j x_j)$$

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

Multiple Lineare Regression

- dabei werden die Koeffizienten β_0 bis β_n derart gesucht, dass die Summe der quadrierten Residuen minimal wird

$$\boldsymbol{RSS} = argmin \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = argmin \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j \cdot x_{i,j} \right)^2$$

- dabei bedeuten
  - $\boldsymbol{RSS}$: Summe der Residuenquadrate (Residual Sum of Squares, hier farbig, da später wieder verwendet)
  - y_i: wahrer Targetwert der i-ten Beobachtung
  - $\hat{y}_i$: geschätzter (vom Modell vorhergesagter) Targetwert der i-ten Beobachtung
  - n: Anzahl Beobachtungen (Index: i)
  - p: Anzahl Features (Index: j)

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

Multiple Lineare Regression

- die zu minimierende Funktion wird auch Straffunktion (loss function) genannt
- die eigentliche Minimierung erfolgt unter anderem durch Auflösen eines Gleichungssystems mit p Unbekannten (unter Zuhilfenahme des Matrix-Kalkül der Linearen Algebra, mehr Details: CAS DA)

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.1 Theorie

Performance Metriken (ein Ausblick auf Kap. 4.4.2)

- wie bei Klassifikation stehen verschiedene Performance Metriken zur Verfügung
- dabei werden jeweils zwei Vektoren einander gegenübergestellt:
  - y_true: wahrer Wert des Targets der Testdaten (auch als y_test bezeichnet)
  - y_pred: vom Modell aufgrund der Features der Testdaten vorhergesagter Wert des Targets
- die zwei populärsten Metriken werden hier (schon einmal) kurz vorgestellt
  - R2 (sklearn.metrics.r2_score): Bestimmtheitsmass (oder Determinationskoeffizient), zugleich interne Scorer Methode bei allen Regressionsklassen (default Scorer) Wertebereich 0-1, wobei 1 für perfekte Anpassung (<0 bei falschen Voraussetzungen)
  - MSE (sklearn.metrics.mean_squared_error): mittlere Quadratische Abweichung der wahren Werte vom Modell und somit der Mittelwert der quadrierten Residuen Wertebereich 0-∞, wobei 0 für perfekte Anpassung
- eine detaillierte Zusammenstellung weiterer Metriken erfolgt im Kapitel 4.4.2 (Performance Metriken Regression)

3.2.1.2 Lineare Regression mit Matrix-Operationen (Extra)

- Lineare Regression lässt sich als System von mehreren linearen Gleichungen mit ebenso vielen Unbekannten formulieren
- die Lineare Algebra, ein Teilgebiet der Mathematik, entstand aus zwei konkreten Anforderungen heraus, wie
  - dem Lösen von linearen Gleichungssystemen
  - der rechnerischen Beschreibung geometrischer Objekte
- die heute noch gültigen Ansätze dazu gehen auf Arbeiten namhafter Mathematiker des 19. Jahrhunderts zurück (u.a. Legendre und Gauss)
- um den Schreibaufwand zu reduzieren, werden die Gleichungssysteme im Folgenden in der Matrixform dargestellt
- numpy und numpy.linalg enthalten zahlreiche Funktionen (Operationen) zur Behandlung von Matrizen (und Vektoren)

3.2.1.2 Lineare Regression mit Matrix-Operationen (Extra)

- die für Lineare Regression relevanten Matrix-Operationen, sowie die entsprechenden Funktionen aus numpy sind dabei:
  - transponieren: `numpy.matrix.transpose(M)`
  - multiplizieren: `numpy.matmul(M1, M2)`
  - invertieren: `numpy.linalg.inv(M)`

- diese sollen im Folgenden anhand eines einfachen Zahlenbeispiels kurz vorgestellt werden

- folgende Bezeichnungen
  - X: Feature Matrix
  - y: Target Vektor

3.2.1.2 Lineare Regression mit Matrix-Operationen (Extra)

- die Matrix-Darstellung der Linearen Regression:

$$\boldsymbol{y} = \boldsymbol{X}\beta + \epsilon$$

- wobei

$$\underset{(n \times 1)}{\boldsymbol{y}} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad \underset{(n \times K)}{\boldsymbol{X}} = \begin{bmatrix} x_{1,1} & \dots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,k} \end{bmatrix} \qquad \underset{(n \times 1)}{\boldsymbol{\beta}} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} \qquad \underset{(n \times 1)}{\boldsymbol{\varepsilon}} = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

- Schätzer:

$$\boldsymbol{b} = (\boldsymbol{X}^T \cdot \boldsymbol{X})^{-1} \cdot \boldsymbol{X}^T \cdot \boldsymbol{y}$$

$$\hat{\boldsymbol{y}} = \boldsymbol{X} \cdot \boldsymbol{b}$$

$$\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$$

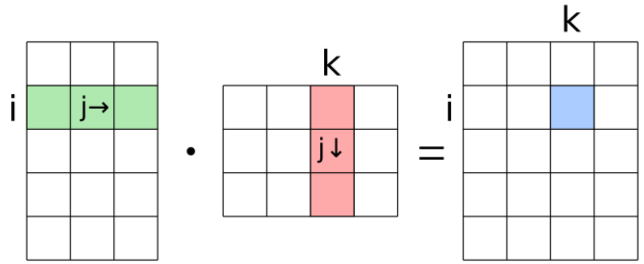
$$s^2 = \frac{\boldsymbol{e}^T \cdot e}{n - K} = \frac{\sum_{i=1}^{n} \left(y_i - \hat{y}^i\right)^2}{n - K}$$

3.2.1.2 Lineare Regression mit Matrix-Operationen (Extra)

- dabei bedeuten
  - $\boldsymbol{M}^T$: transponierte Matrix von $\boldsymbol{M}$, d.h. Vertauschen von Zeilen und Spalten (in der Literatur auch gelegentlich $\boldsymbol{M}'$) (numpy.matrix.transpose())

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

  - $\boldsymbol{M}_1 \cdot \boldsymbol{M}_2$: Matrix-Multiplikation, Details dazu unter [Matrizenmultiplikation] (numpy.matmul())



  - $\boldsymbol{M}^{-1}$: invertiere Matrix von $\boldsymbol{M}$, d.h. multiplizieren von $\boldsymbol{M}^{-1}$ und $\boldsymbol{M}$ führt zu Einheitsmatrix $\boldsymbol{E}$ (vgl. rechts) Details dazu unter [Gauss-Jordan-Algorithmus] (numpy.linalg.inv())

$$\boldsymbol{E} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

3.2.1.2 Lineare Regression mit Matrix-Operationen (Extra)

- damit sind die Operationen vorgestellt, um die β Werte berechnen zu können

$$\beta = (\boldsymbol{X}^{\boldsymbol{T}} \cdot \boldsymbol{X})^{-1} \cdot \boldsymbol{X}^{\boldsymbol{T}} \cdot y$$

- ausgeführt mit Operationen von numpy, hier als Funktion codiert (vgl. extra_3.2.1.2_linear_regression_with_matrix_operations.ipynb)

```
import numpy as np
def MyLinReg(X, y):
    XT = np.matrix.transpose(X)            ## transpose
    XT_X = np.matmul(XT, X)                ## matrix product 1
    XT_X_INV = np.linalg.inv(XT_X)         ## invert
    XT_X_INV_XT = np.matmul(XT_X_INV, XT) ## matrix product 2
    betas = np.matmul(XT_X_INV_XT, y)      ## matrix product 3
    return (betas)
```

3.2.1.2 Lineare Regression mit Matrix-Operationen (Extra)

- das oben dargestellte Verfahren mit numpy Funktionen berechnet alle β Werte (Koeffizienten) der Features, nicht aber den Intercept β_0
- um diesen zu ermitteln ist ein kleiner Kunstgriff notwendig: auf der ersten Spalte wird ein Vektor der Länge k hinzugefügt, welcher überall den Wert 1 aufweist

$$\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{k,1} & \cdots & x_{k,n} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{k,1} & \cdots & x_{k,n} \end{bmatrix}$$

- z.B. mit untenstehendem Code:

```
X = np.c_[np.repeat(1, X.shape[0]), X]
```

- danach ermitteln der β nach demselben Verfahren, der Koeffizient für die erste Spalte ist dann gerade β_0, oder eben der Intercept

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.3 Praxis

- Schritt für Schritt mit dem Melbourne Housing Dataset (beachten Sie das analoge Vorgehen wie bei Klassifikation)
- Voraussetzungen:
  - Daten sind geladen
  - Features - Target - Split ausgeführt
  - Train - Test - Split ausgeführt

  (vgl. prep_data() in Modul bfh_cas_pml.py,
  in Kap 3.1.4 Vorbereiten der Umgebung)

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.3 Praxis

- laden der Klasse, instanziieren, parametrisieren und trainieren des Modells

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
print(model.get_params())
```

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}
```

- `.get_params()` zeigt die per Default eingestellten Parameterwerte
- hier einzig von Bedeutung: fit_intercept, veranlasst, dass Intercept berechnet wird, andernfalls wird dieser Wert auf 0 festgelegt

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.3 Praxis

- die Attribute des trainierten Modells:

```
print(model.intercept_)
print(model.coef_)
print(X_train.columns) ## no model attribute
```

```
-105513873.23401378
[ 2.45383606e+05 -1.41356398e+05 -4.03836664e+04  1.61336039e+05
  4.03911483e+04  8.33032709e+04  2.73783998e+05 -2.48422914e+03
:
Index(['Rooms', 'Type', 'Distance', 'Bathroom', 'Car', 'logLandsize',
       'logBuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
:
```

  - `X_train.columns` gibt die Feature Namen aus, damit die ermittelten Koeffizienten zugeordnet werden können

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.3 Praxis

- Modell score

```
print(model.score(X_test, y_test))
```

```
0.5601419746121148
```

  - gemäss Dokumentation wird hier `r2_score` ermittelt

- Kontrolle mit predict und expliziter Nutzung von `r2_score` aus `sklearn.metrics`:

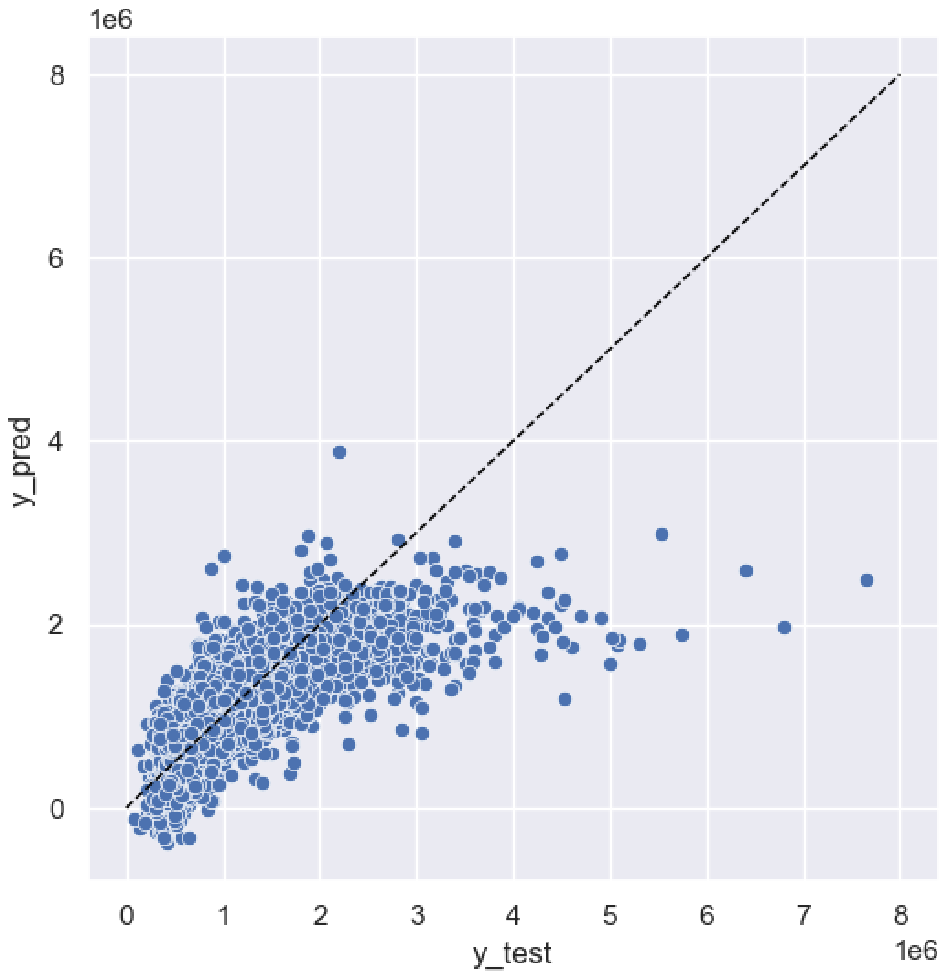
```
from sklearn.metrics import r2_score
y_pred = model.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.5601419746121148
```

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.3 Praxis

- ein visueller Vergleich der vorhergesagten (y_pred) und der wahren Targetwerte des Testsets (y_test)
- die eingezeichnete Diagonale wurde manuell hinzugefügt und markiert Identität



- Fazit:
  - dieses Modell erzeugt tatsächlich Vorhersagen mit negativen Werten, was doch recht unglaubwürdig erscheint
  - der Zusammenhang zwischen Voraussagen und wahren Werten erscheint nicht linear, ev. aufgrund von Korrelationen in den Features
  - welche Rolle spielen Extremwerte in den wahren Werten?

3.2.1.4 Lineare Regression in der Datenanalyse (Extra)

- R, SAS, Statista, etc. bieten im Output viel mehr Informationen zu Linearer Regression
- für Python gibt es dazu auch eine entsprechende Library: [statsmodels]

```
import statsmodels.api as sm
model = sm.OLS(y_train, X_train, hasconst=True)
results = model.fit()
print(results.summary())
```

- der Aufruf liefert den folgenden umfangreichen Output
  vgl. `extra_3.2.1.4_linear_regression_in_data_analytics.ipynb`

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.4 Lineare Regression in der Datenanalyse (Extra)

- dieser Output ist natürlich viel ausführlicher als für ML tatsächlich notwendig
- Details: CAS DA

```
 OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.586
Model:                            OLS   Adj. R-squared:                  0.585
Method:                 Least Squares   F-statistic:                     752.8
Date:                Fri, 26 May 2023   Prob (F-statistic):               0.00
Time:                        11:35:55   Log-Likelihood:            -1.7592e+05
No. Observations:               12262   AIC:                         3.519e+05
Df Residuals:                   12238   BIC:                         3.521e+05
Df Model:                          23
Covariance Type:            nonrobust
=========================================================================================================
                                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
const                         -1.055e+08   1.99e+07     -5.300      0.000   -1.45e+08   -6.65e+07
Rooms                          2.454e+05   5402.990     45.416      0.000    2.35e+05    2.56e+05
Type                          -1.414e+05   6342.919    -22.286      0.000   -1.54e+05   -1.29e+05
Distance                      -4.038e+04    928.303    -43.503      0.000   -4.22e+04   -3.86e+04
Bathroom                       1.613e+05   7004.805     23.032      0.000    1.48e+05    1.75e+05
Car                            4.039e+04   4723.191      8.552      0.000    3.11e+04    4.96e+04
logLandsize                     8.33e+04   5149.484     16.177      0.000    7.32e+04    9.34e+04
logBuildingArea                2.738e+05   2.27e+04     12.064      0.000    2.29e+05    3.18e+05
YearBuilt                     -2484.2291    162.873    -15.253      0.000   -2803.486   -2164.972
CouncilArea                   -4977.2245    674.748     -7.376      0.000   -6299.838   -3654.611
Lattitude                      -5.15e+05   7.24e+04     -7.114      0.000   -6.57e+05   -3.73e+05
Longtitude                     1.926e+05   5.94e+04      3.241      0.001    7.61e+04    3.09e+05
.
==============================================================================
Omnibus:                     6530.672   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            95692.398
Skew:                           2.225   Prob(JB):                         0.00
Kurtosis:                      15.942   Cond. No.                     4.86e+07
==============================================================================
```

3.2.1.4 Lineare Regression in der Datenanalyse (Extra)

- sehr schönes Beispiel zum Aufzeigen der Unterschiede zwischen Datenanalyse und Machine Learning
- **Datenanalyse**
  - erkennen und herausarbeiten von Einflüssen ausgewählter Komponenten (unabhängige Variablen) auf eine Zielkomponente (abhängige Variable)
  - modellieren von Wirkungsgefügen, wenn Kausalitäten geklärt sind
  - testen, ob die geforderten Voraussetzungen in den Daten überhaupt gegeben sind, um gültige Aussagen machen zu können
  - Akteure: Datenanalytiker, oft Fachspezialistinnen mit Datenanalyse-Skills
  - Tools: Analyse Software wir R, SPSS, SAS, Statista etc.
  - Ergebnisse: Modellkoeffizienten sowie ausführliche Diagnosemetriken zum Beurteilen der Gültigkeit von Modellen (aus Sicht der Datenanalyse)

3.2.1 Regression - Regression klassisch - LinearRegression (OLS)

3.2.1.4 Lineare Regression in der Datenanalyse (Extra)

- **Machine Learning**
  - erstellen, optimieren, testen und implementieren von Vorhersagemodellen
  - Akteure: ML Spezialistinnen, Data Scientists, oft etwas weiter vom Fach weg als Datenanalytiker
  - Tool(s): Python, Spark, R, etc.
  - Ergebnisse: ausschliesslich Modellkoeffizienten und ausgewählte Scorer

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.1 Theorie

- Schwächen der Linearen Regression:
  - untereinander korrelierte Features können einen ungewollten Einfluss auf Ergebnisse haben (Multikollinearität)
  - bei Vorliegen von mehr Features als Beobachtungen ist keine Lösung errechenbar

- Wiederholung von 3.2.1.1:

$$\boldsymbol{RSS} = argmin \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- während bei der Linearen Regression die Summe der Quadrate der Residuen (loss) minimiert wird, kommt hier noch ein "Strafterm" dazu, welcher grosse Werte von β_i einzuschränken versucht (shrinking), idealerweise sogar auf 0

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.1 Theorie

- durch Hinzufügen eines Strafterms - multipliziert mit einer Konstanten λ - zur Verlustfunktion, werden grosse Werte für die jeweiligen β zurückgebunden

$$\boldsymbol{RSS} + \lambda \cdot penalty$$

- dadurch kann folgendes erreicht werden:
  - Störungen der Regression durch Multikollinearität wird reduziert
  - es können mehr Features als Beobachtungen eingesetzt werden
  - je nach Tuningparameter können Kandidaten zur Feature Selection identifiziert werden
- was hier schon sichtbar ist:
  - $\lambda = 0$ führt zum selben Resultat wie LinearRegression (der Strafterm wird ausgeschaltet)
  - $\lambda \to \infty$ schafft ein Übergewicht der Strafe und drängt die β Werte gegen 0
- λ ist somit ein Tuning Parameter

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.1 Theorie

- in den meisten Publikationen wird die oben verwendete Konstante mit λ bezeichnet, in neuerer Zeit auch mit α, letzteres wird auch bei scikit-learn als Parametername verwendet, in den folgenden Grafiken werden die Achsen trotzdem mit λ genennzeichnet
- Gegenüberstellung Lasso und Ridge Regression

	Lasso	**Ridge**
andere Bezeichnung	L1-Regulierung	L2-Regulierung
Strafterm	die Summe der absoluten β-Werte	die Summe der quadrierten β-Werte
zu minimierende Verlustfunktion	$\boldsymbol{RSS} + \lambda \cdot \sum_{j=1}^{p} \lvert\beta_j\rvert$	$\boldsymbol{RSS} + \lambda \cdot \sum_{j=1}^{p} \beta_j^2$

(*Lasso* steht für "least absolute shrinkage and selection operator",
Ridge ist dagegen kein Akronym und verweist auf das hinterlegte Verfahren [Details])

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.1 Theorie

- Gemeinsamkeiten
  - kleines λ : der Strafterm hat kaum Einfluss, die Koeffizienten entsprechen annähernd jenen von LinearRegression
  - (sehr) grosses λ: der Strafterm übernimmt den Lead, die Koeffizienten streben gegen 0
- Unterschiede
  - Lasso: die Annäherung an 0 bei grossen λ erfolgt mehr oder weniger direkt und wird früher oder später vollständig erreicht
  - Ridge: die Annäherung an 0 bei grossem λ erfolgt asymptotisch, wird aber nicht völlig erreicht

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.2 Praxis

- Lasso

```
from sklearn.linear_model import Lasso
model = Lasso()
model.fit(X_train, y_train)
print(model.intercept_)
print(model.coef_)
print(model.score(X_test, y_test))
```

```
-105470886.17680839
[ 2.45382007e+05 -1.41353663e+05 -4.03813210e+04  1.61339539e+05
  4.03901128e+04  8.33016390e+04  2.73752966e+05 -2.48435395e+03
:
0.560142704629316
```

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.2 Praxis

- Ridge

```
from sklearn.linear_model import Ridge
model = Ridge()
model.fit(X_train, y_train)
print(model.intercept_)
print(model.coef_)
print(model.score(X_test, y_test))
```

```
-104848432.75507241
[ 2.45313734e+05 -1.41286745e+05 -4.03551367e+04  1.61451266e+05
  4.03883258e+04  8.32645429e+04  2.73073082e+05 -2.48815619e+03
:
0.5601387631837784
```

- zur Erinnerung, der Parameter alpha entspricht λ in der theoretischen Einführung

3.2.2 Regression - Regression klassisch - Regularisiert

3.2.2.2 Praxis

Fazit zu Lasso und Ridge Regression

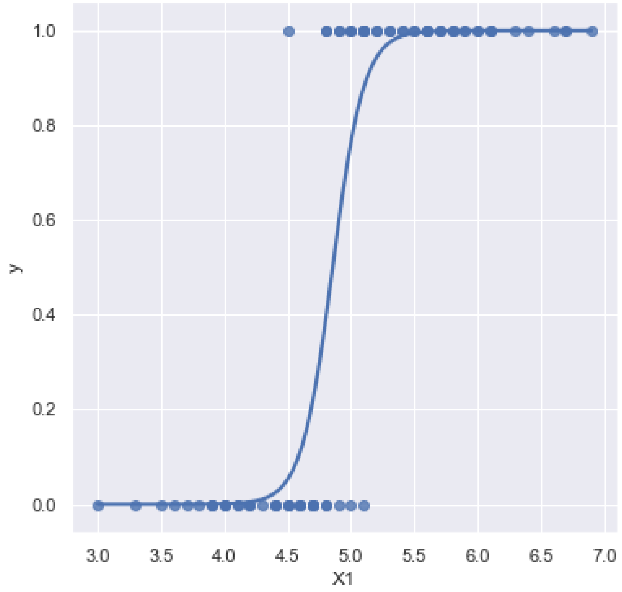
- als Kandidaten im Wettstreit der Regressoren nicht unbedingt Favorit
- **aber**: wichtige Instrumente für Feature Selection, insbesondere Lasso
- eine Darstellung der übrigbleibenden Feature Namen und Koeffizienten (nach trainieren mit dem Parameter `alpha=10000`) zeigt folgende Liste der verbleibenden Features an (vgl. [ipynb])
- von 23 Features bleiben also deren 7 übrig, aus dem Ergebnis könnte damit z.B. eine Filtermaske gebaut werden (vgl. [ipynb])

	cols	coefs
0	Rooms	253568.228957
3	Bathroom	168521.486901
4	Car	34293.196859
5	logLandsize	76942.997592
12	Method_S	18550.768445
17	Regionname_Southern_Metropolitan	286752.054486
22	day_of_week	781.020849

3.2.3 Regression - Regression klassisch - Logistische Regression

- wurde im Zuge der Klassifikationsmethoden eingeführt (Kap. 2.3.4)
- hier nur noch der Vollständigkeit halber erwähnt, da
  - von der Methodik her zwar Regression
  - von der Anwendung her dagegen Klassifikation

- nebenstehend zur Erinnerung eine Visualisierung aus dem erwähnten Kapitel zu Klassifikation mit `sklearn.linear_model.LogisticRegression`



3.2 Regression - Regression klassisch

Workshop 08

Gruppen zu 2 bis 4, Zeit: 30'

- untersuchen Sie den Einfluss des Standardisierens der Features auf folgende Ergebnisse der Linearen Regression:
  - Modellkoeffizienten
  - Predictions
  - Score

- untersuchen Sie den Einfluss des Logarithmierens des Targets auf die Performance der Linearen Regression