BILDUNGSCAMPUS
DIETER SCHWARZ STIFTUNG

HOCHSCHULE HEILBRONN

Hochschule Heilbronn
Fakultät Wirtschaft und Verkehr

LASSO Regression

Methodensteckbrief[a] | Data Science - Grundlagen

Autoren
Daniel Messner[b]
Sebastian Kahlert[c]
Sebastian Strassburg[d]

[a]Vorliegendes Dokument basiert auf Template von François Rozet.
[b]Matrikel-Nr. 193842 | damessner@stud.hs-heilbronn.de
[c]Matrikel-Nr. 196089 | skahlert@stud.hs-heilbronn.de
[d]Matrikel-Nr. 208383 | strassburg@stud.hs-heilbronn.de

Inhaltsverzeichnis

1 Einleitung

In diesem Methodensteckbrief wird die Funktionsweise der LASSO Regression erklärt. Neben der Formula werden ebenfalls die Modellprämissen und -gütemaßen als auch die Stärken und Schwächen der Regressionsmethode erläutert. Die interaktive Demonstration mit einem Beispieldatensatz schließt den Methodensteckbrief.

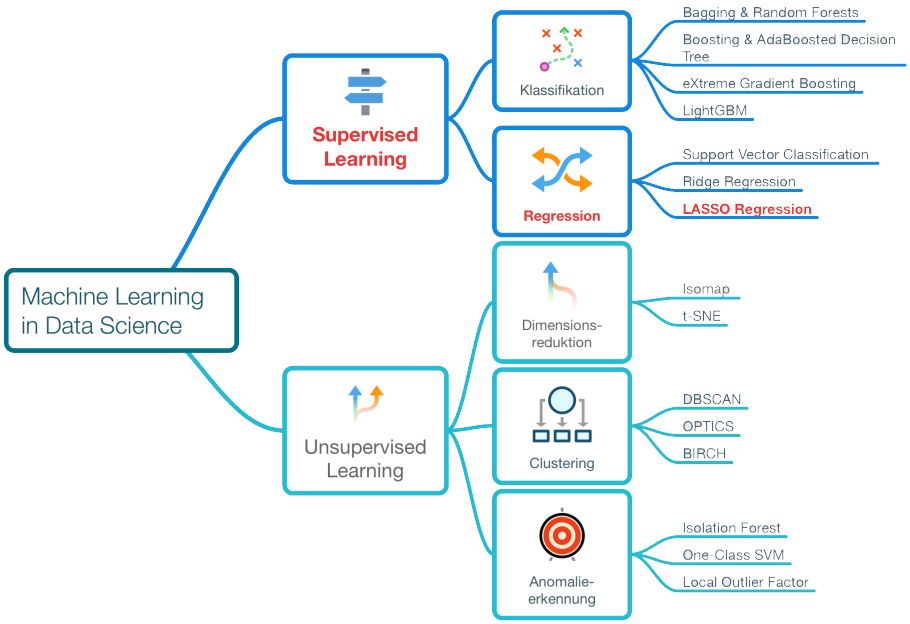


Abbildung 1: Einordnung in Data Science - Grundlagen

2 Geschichte

Der Operator für die geringste absolute Schrumpfung und Selektion (LASSO) wurde von Tibshirani 1996 für die Parameterschätzung und gleichzeitig für die Variablen Modellselektion in der Regressionsanalyse eingesetzt (Vgl. Muthukrishnan und Rohini 2016). Unabhängig davon, wurde die LASSO Regression bereits in der geophysikalischen Literatur von Santosa und Symes 1986 angewandt, zur Popularität hat jedoch Tibshirani 1996 maßgeblich beigetragen. Heute kommt die LASSO Regression vor allem in der Statistik im Allgemeinen und beim maschinellen Lernen zum Einsatz.

3 Kernidee und Funktionsweise

Lineare Regressionsmodelle dienen zur Vorhersage einer Zielvariablen. Dabei wird die Kausalität zwischen der Ziel- und zahlreicher Einflussvariablen überprüft. Die Einflussvariablen können einen schwachen, starken oder gar keinen Einfluss auf die Zielvariable besitzen. Mithilfe von zahlreichen Regressionsmethoden können die Kausalitäten untersucht werden. Bei der LASSO Regression handelt es sich viel mehr um ein Verfahren, die die Anzahl der Einflussvariablen eines bestehenden linearen Regressionsmodells reduziert. Die Reduzierung der Einflussvariablen sowie die Funktionsweise und mathematische Herleitung wird im nachfolgenden Kapitel erklärt.

Key Info

Jedes lineare Regressionsmodell hat folgenden Aufbau: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \times x_1 + \hat{\beta}_2 \times x_2 + \cdots + \hat{\beta}_n \times x_n + \epsilon$. Dabei ist $\hat{y}$ die Zielvariable, $\hat{\beta}_0$ der konstante Parameter (*Intercept*), $\hat{\beta}_n$ der Steigungsparameter (*Slope*), x_n die Einflussvariable sowie ϵ der Residualfehler (*Residuals*) bzw. die nicht erklärte Modellvarianz. Kennzeichen eines linearen Regressionsmodells ist, dass alle Einflussvariablen im 1. Grad betrachtet werden.

3.1 Hintergrund

Die Problematik aller Modelle, die im Bereich des maschinellen Lernens generiert werden, ist die Wahl der optimalen Modellkomplexität. In der Abbildung 2 (in Anlehnung nach Deng u. a. 2015, S. 35) wird die Problematik im *Bias-Variance-Tradeoff* ersichtlich.

- Beim **Bias** handelt es sich um die statistische Verzerrung, also um die Abweichung der Modellwerte von den Realwerten.
- Bei der **Variance** handelt es sich um die Streuung der Daten um den Mittelwert. Je stärker die Daten um den Mittelwert streuen, desto höher ist die Varianz.
- Der **Total Error** setzt sich aus der Summe des **Bias** und der **Variance** zusammen. (Vgl. Singh 2018)

Je höher die Modellkomplexität (bzw. je höher die Anzahl der Einflussvariablen), desto niedriger ist der Bias und höher die Variance. Der Total Error nimmt ebenfalls zu. Ein Modell mit einer hohen Komplexität (*Overfitting*) erklärt zwar sehr gut die Trainingsdaten, jedoch kommt es zu häufigen Fehlerraten hinsichtlich der Testdaten. (Vgl. Singh 2018) Außerdem ist die Interpretierbarkeit solcher Modelle erheblich schwierig und schlecht anwendbar für weitere Berechnungen. Bei einem Modell mit niedriger Komplexität (*Underfitting*) erklärt es jedoch unzureichend die Grundgesamtheit. Das Modell sagt die Werte ungenau vorher. In der Animation in `Jupyter Notebook` (in Anlehnung

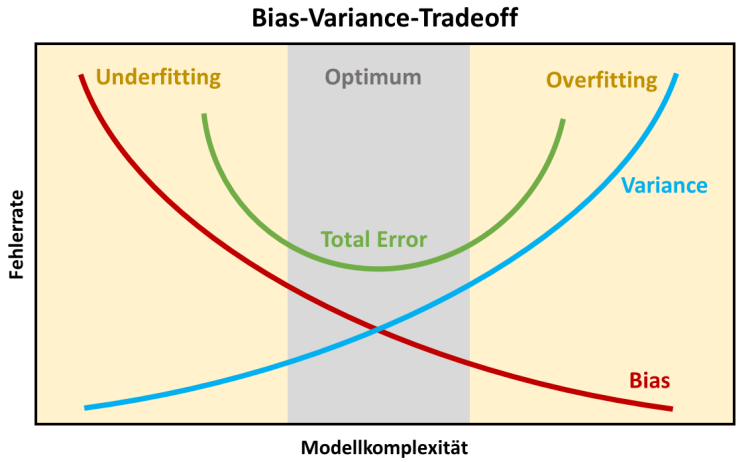


Abbildung 2: Bias-Variance-Tradeoff

nach Rashidi u. a. 2019, S. 11) ist die Problematik dargestellt.

Exemplarisches Beispiel

Die Elektroautos von dem Unternehmen Tesla sind dafür bekannt, mithilfe von KI-Algorithmen autonomes Fahren zu ermöglichen. Eine Autonomie erfordert neben den Kameras, die rund um das Auto verbaut sind, auch entsprechende Algorithmen wie Mustererkennung und neuronales Denken. Mithilfe dieser Algorithmen kann ein Tesla Fahrzeug zum Beispiel die Verkehrsschilder klassifizieren. Angenommen wir haben ein Modell (unter der Annahme, dass Tesla LASSO Regressionen verwendet), welches sich im optimalen Bereich befindet. Das Fahrzeug kann mit diesem Modell effizient autonom agieren. Es erkennt Verkehrsschilder und andere Umwelteinflüsse (Menschen, Wildtiere etc.). Würde das Modell nun erkennen wollen, welchen sozialen und kultruellen Hintergrund die Menschen haben, so wäre das Modell überangepasst (*Overfitting*), da es für das Auto irrelvant ist, ob jemand angloamerikanisch oder südasiatisch geprägt ist. Würde das Modell nur Verkehrsschilder erkennen, so würde er alle Menschen und Umweltfaktoren in seiner Umgebung nicht berücksichtigen, was fatale Folgen hätte. Insofern wäre das Modell nicht genug angepasst (*Underfitting*).

OUTPUT: siehe `Jupyter Notebook`

Um die Modellkomplexität zu verringern, also die Anzahl der Einflussvariablen, gibt es 3 Möglichkeiten:

- **Dimensionsreduktion** → *Verschmelzung* von ähnlich strukturierten Variablen
- **Variablenselektion** → Selektion einzelner Variablen unter Berücksichtigung bestimmter Kriterien

- **Regression Shrinkage** → Skalierung der Regressionsparameter

Die LASSO-Regression ermöglicht eine Verringerung der Modellkomplexität mittels *Regression Shrinkage* (siehe in `Jupyter Notebook`).

Key Info

Das Ziel der LASSO-Regression im allgemeinen Kontext und für die nachfolgende Demonstration im Methodensteckbrief ist die Optimierung der Modellkomplexität (Reduzierung der **Variance**, Erhöhung des **Bias**), um die Fehlerrate der Testdaten zu reduzieren.

3.2 Formula

Die LASSO Regression versucht, wie auch bei jeder Regression, die Summe der quadratischen Abweichungen zu minimieren. Durch diese Minimierung entsteht die bestmögliche Regression zwischen der Einfluss- und Zielvariablen. Im Zusammenhang mit dem maschinellen Lernen wird jedoch ein Schrumpfungsterm (*Shrinkage Penalty*) benötigt: $\lambda \sum_{j=1}^{p} |\beta_j|$. Der Term bewirkt eine Skalierung aller Parameter des Regressionsmodells mit Ausnahme des konstanten Parameters (*Intercept*). (Vgl. James u. a. 2013, S. 219)

Im Nachfolgenden ist die Forumla dargestellt (in Anlehnung nach Tibshirani 1996, S. 268, modifiziert nach James u. a. 2013, S. 219 - 220), die die LASSO Regression versucht zu minimieren:

$$\underset{\beta_0,\beta_j}{\arg\min} \Big\{ \sum_{i=1}^{n} \Big(y_i - \beta_0 \sum_{j=1}^{p} \beta_j x_{ij} \Big)^2 \Big\} \text{ subject to } \lambda \sum_{j=1}^{p} |\beta_j| \leq s \tag{1}$$

Hierbei ist n die Gesamtanzahl der Variablen, p Gesamtanzahl der Werte einer Variable, β_0 der konstante Parameter, β_j die Steigungsparameter der jeweiligen Variablen, y_i die Zielvariable, x_i die Einflussvariablen und λ der Sensibilitätsparameter für den Schrumpfungsterm. (Vgl. James u. a. 2013, S. 219) *argmin* bedeutet, dass an der Stelle des Minimums (im Sinne der kleinsten Summe der quadratischen Abweichung) die Funktion berechnet werden soll. Hierbei muss auch der Schrumpfungsterm gleichermaßen mit berücksichtigt und erfüllt werden (*subject to*).

Zur vereinfachten Darstellung wird folgende Formula (James u. a. 2013, S. 219) betrachtet (hierbei versucht die LASSO Regression die Formula zu minimieren):

$$\sum_{i=1}^{n} \Big(y_i - \beta_0 \sum_{j=1}^{p} \beta_j x_{ij} \Big)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{2}$$

Die Skalierung mithilfe des Schrumpfungsterm bewirkt, dass Parameter mit minimalen Steigungen absolut auf 0, während Parameter mit größeren Steigungen nahezu auf 0 skaliert werden. Duch diese Schrumpfung der Parameter

werden einzelne Variablen ausgeschlossen. Somit führt die LASSO Regression auch eine *Variablenselektion* durch. Wie sensitiv die Skalierung durchgeführt wird, hängt von der Größe des λ ab. Es gilt: je höher λ, desto stärker ist die Skalierung der Steigungsparameter des Regressionsmodells. (Vgl. Tibshirani 1996, S. 270)

Key Info

Mithilfe einer Cross-Validation kann der beste λ-Wert identifiziert werden. Hierbei ist der beste λ-Wert der Wert, wo für das Modell die höchste Modellgüte und das beste Informationskriterium vorliegt. Im nachfolgenden Kapitel werden die Modellgüte und Kriterien erläutert.

3.3 Modellgüte und Modellauswahl

Für die Bestimmung der Güte eines Modells sowie der Modellauswahl werden verschiedene Maßen herangezogen. Nachfolgend werden 3 Maßen vorgestellt, die in der Demonstration verwendet werden:

1. Das Bestimmtheitsmaß (Modellgüte) R^2 erklärt die Modellvarianz an der Gesamtvarianz. Je höher das Bestimmtsheitsmaß, desto besser erklärt das Modell die Regression multivariater Dimensionen. Je höher die Streuung der Werte, desto ungenauer wird die Regression erklärt und folglich sinkt R^2. Die Formula für R^2 ist (in Anlehnung nach Backhaus u. a. 2016, S. 84):

$$R^2 = \frac{SSE}{SST} = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3}$$

   Hierbei ist SSE die Modellvarianz und SST die Gesamtvarianz. Für SSE werden alle Variationen der Zielvariablen vom Regressionsmodell berechnet (daher $\hat{y}$ als Schätzer); für SST alle Variationen der Zielvariablen von den realen Werten (daher y als Beobachtung).

2. Das Aikaike-Informationskriterium AIC ist als Maß dafür geeignet, verschiedene Modelle miteinander zu vergleichen. Dabei besitzt das AIC einen Strafterm - je höher die Anzahl der Parameter, desto sensitiver agiert der Strafterm. Das AIC hat folgende Formula (in Anlehnung nach Backhaus u. a. 2016, S. 333):

$$AIC = 2 \times k - 2 \ln \mathcal{L} \tag{4}$$

   Dabei ist k die Anzahl der Parameter und $\ln \mathcal{L}$ die logarithmierte Likelihood-Funktion. Dabei werden die wahrscheinlichsten Werte aus der Stichprobe für die Schätzung des Parameters betrachtet.

3. Das Bayessche Informationskriterium (BIC) baut auf das AIC auf. Der Unterschied ist der zugrundeliegende Strafterm. Statt der doppelten Multiplikation mit der Anzahl der Parameter erfolgt eine Multiplikation zwischen den logarithmierten Umfang der Stichprobe und der Anzahl der Parameter.

Die Formula ist (in Anlehnung nach Backhaus u. a. 2016, S. 333):

$$BIC = \ln U \times k - 2 \ln \mathcal{L} \tag{5}$$

Hierbei ist $\ln U$ der logarithmierte Stichprobenumfang.

Key Info

Das BIC bestraft den Term insgesamt stärker als das AIC bei einer hohen Anzahl der Parameter. (Vgl. Backhaus u. a. 2016, S. 334) Es gibt jedoch kein optimales Informationskriterium, welches das beste Modell selektiert. Stattdessen werden beide Informationskriterien für die Modellauswahl angewandt.

4 Anwendungsvoraussetzungen

Für die LASSO Regression gelten die gleichen Modellprämissen wie für jede multivariate lineare Regression: (Vgl. Backhaus u. a. 2016, S. 98, 111)

1. **Linearität und korrekte Spezifizierung**: alle Variablen dürfen nur als 1. Grad vorliegen. Es darf keine Potenzierung der Werte stattfinden. Weiterhin muss ersichtlich sein, welche Variable(n) die Ziel- und Einflussvariable(n) darstellt. So gilt: β_0 und β_n sind linear sowie für X sind alle relevanten Einflussvariablen enthalten.
2. **Homoskedastizität**: Die Variation der Residuen ist gleichmäßig ausgeprägt. Bei einer Heteroskedastizität folgt es zu einer ungleichmäßigen Variation der Residuen um das Regressionsmodell und folglich zu einer Verzerrung des Modells (*Underfitting*). Ein Rückschluss auf die Grundgesamtheit ist somit nicht mehr geeignet.
3. **keine Multikollinearität**: Die Korrelationen innerhalb der Einflussvariablen beträgt: $Cor(X) < 1$. Bei einer perfekten Korrelation $Cor(X)$ von $|1|$ würde eine vollkommende lineare Abhängigkeit vorliegen. Somit würde eine Redundanz innerhalb des Modells vorliegen und führt folglich zu einer Überanpassung.
4. **Normalverteilung der Residuen**: Die Residuen müssen normalverteilt sein. Die Verteilung der Residuen um den Wert 0 muss gleichmäßig ausgeprägt sein. Liegt keine Normalverteilung vor, folgt es wie bei der 3. Annahme zu einer Verzerrung des Modells. Für die Überprüfung der Normalverteilung wird neben der grafischen Darstellung auch der Anderson-Darling-Test (Vgl. Anderson und Darling 1952, S. 193) verwendet.
5. **keine Autokorrelation**: Die Korrelation der Residuen beträgt: $Cor(\mathcal{E}) = 0$. Bei einer Autokorrelation würden die Residuen bei einer Veränderung der Einflussvariable(n) linear gleichermaßen miterklärt werden. Dies führt zu einer Überanpassung (*Overfitting*) des Modells. Für die Überprüfung einer Autokorrelation wird der Durbin-Watson-Test (Vgl. Gujarati und Porter 2009, S. 320 - 324) verwendet.

Key Info

In der Praxis ist es geläufig, dass nicht immer alle Anwendungsvoraussetzungen erfüllt sind. So kann ein gutes lineares Modell vorliegen mit heteroskedastischen Residuen.

5 Stärken und Schwächen

Nachfolgend werden die Stärken und Schwächen der LASSO Regression tabellarisch zusammengefasst: (Vgl. Pereira, Basto und Silva 2016; Vgl. Fonti und Belitser 2017)

Merkmal	**Stärken**	**Schwächen**
Variablen-selektion	• Schrumpfung von nutzlosen Variablen auf 0 • Ausschluss aus dem Modell	• willkürliche Auswahl von Merkmalen bei Gruppe von korrelierenden Merkmalen • möglicher Informationsverlust und geringere Genauigkeit des Modells
Einfachheit	• Einfache Interpretation des Modells durch Variablenselektion	• starke Automatisierung des Modells vernachlässigt Modellanpassungen
Informations-gehalt	• Verringerung des Varianz • Anpassung der Modellkomplexität zwischen Verzerrung und Varianz	• wichtige bzw. relevante Variablen können auch ausgeschlossen werden
Rechenzeit	• niedrigdimensionelle Datensätze erfordern weniger Rechenzeiten	• hochdimensionelle Datensätze erfordern mehr Rechenzeiten

6 Demonstration

In diesem Kapitel wird die Demonstration in Python und dazu die Interpretation der Ergebnisse sowie die Analogie zu der Formula bzw. zu den vorher erklärten Ergebnissen durchgeführt.

6.1 Packages

Neben den `numpy`-, `matplotlib`-, `pandas`- und `seaborn`-Packages wird überwiegend auf die *Scikit-Learn* Umgebung (Pedregosa u. a. 2011) zurückgegriffen. Das Package `warnings` unterdrückt die Warnmeldungen innerhalb der Prozedur. Für alle Plots wird ein spezifisches Design von `matplotlib.style` verwendet.

> Bei der standardmäßigen Installation sind alle Packages bereits enthalten. Das Package `pywaffle` ist die Ausnahme und muss nachträglich installiert werden. Die Installationsschritte sind unter folgendem Link zu finden: `https://pypi.org/project/pywaffle/`

```
# Ausblenden von stoerenden Warnmeldungen
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import statsmodels.stats.stattools as stm
import statsmodels.api as sm
import scipy.stats as sps

from ipywidgets import interact, interactive, fixed,
    interact_manual
import ipywidgets as widgets

from sklearn import linear_model, model_selection
from sklearn.decomposition._nmf import EPSILON
from sklearn.model_selection import train_test_split,
    cross_val_score
from pywaffle import Waffle

# einheitliche Plot-Groesse
plt.rcParams['figure.figsize'] = [15, 12]
plt.subplots_adjust(left=0.1, bottom=0.7, right=0.6, top=1.5)

# einheitlicher Plot-Stil
import matplotlib.style as style
style.use('bmh')
```

6.2 Dataset und Splitting

Für die Demonstration wird der Rotweindatensatz betrachtet. Dieser wurde von Cortez u. a. 2009, heruntergeladen von Dua und Graff 2017, erhoben.

> Dabei stammt der Rotwein aus der nördlichen Region von Portugal (*Vinho Verde*). Neben dem Rotwein wurden auch Daten für Weißweine erhoben. Beide Weinsorten zusammen bilden 15% der Weinproduktion in Portugal sowie 10% des Weinexportes.

Der Datensatz besteht aus 12 Attributen, wovon 11 Attribute quantitativ (metrisch kontinuierlich) und 1 Attribut qualitativ (ordinal) ausgeprägt sind. Folgende Attribute sind enthalten: (Vgl. Cortez u. a. 2009, S. 548 - 549; Vgl. Rhein-Ahr-Wein 2020; UCI Machine Learning 2017)

- `fixed acidity` - feste Säure in $\frac{g}{dm^3}$. Feste Säure ist ein Grundbestandteil jeden Weines, der aus den Trauben ursprünglich herkommt. Feste Säuren sind zum Beispiel Weinsäuren.
- `volatile acidity` - flüchtige Säure in $\frac{g}{dm^3}$. Die Säure verflüchtet sich schnell, zum Beispiel die Essigsäure.
- `critic acidity` - Zitronensäure in $\frac{g}{dm^3}$. Die Säure wird verwendet, um den Wein *frischer* zu machen.
- `residual sugar` - Restzucker in $\frac{g}{dm^3}$. Je eher die Gärung eines Weins gestoppt wird, umso höher ist der Zuckeranteil im Wein, da der Zucker nicht vollständig zum Alkohol abgebaut werden konnte.
- `chlorides` - Chloride in $\frac{g}{dm^3}$. Die Chloride sind Salzbausteine, die im Wein enthalten sind.
- `free sulfur dioxide` - ungebundener Schwefeldioxid in $\frac{mg}{dm^3}$. Ungebunden bedeutet, dass er keine Verbindungen mit anderen chemischen Elementen eingeht.
- `total sulfur dioxide` - ungebundener und gebundener Schwefeldioxid in $\frac{mg}{dm^3}$. Der gebundene Schwefeldioxid exisitiert in Verbindung mit anderen chemischen Elementen.
- `density` - Dichte in $\frac{g}{cm^3}$. Da der Wein zum größten Teil aus Wasser besteht, liegt die Dichte sehr nah an 1.
- `pH` - pH-Wert. Der pH-Wert gibt an, wie hoch der saure oder basische Charakter ist.
- `sulphates` - Sulfate in $\frac{g}{dm^3}$. Genau wie beim Schwefeldioxid schützen die Sulfate vor dem Zerfall des Weins.
- `alcohol` - Alkoholgehalt in $vol.\%$.
- `quality` - Qualität des Weins in einer Skala von 0 bis 10. Dabei wurde jeder Wein von mindestens 3 Gutachtern überprüft. Es wurde anschließend

der Median für alle 3 Bewertungen des jeweiligen Weins berechnet und im Datensatz aufgenommen.

Exemplarisches Beispiel

Die Weinsorten lieblich, halbtrocken und trocken stehen für den Grad der Gärung. Ein lieblicher Wein ist zum Beispiel nicht vollkommen gegärt und schmeckt dadurch *süßer*. Ein trockener Wein dagegen enthält kaum Zucker und schmeckt dadurch *schwerer*. Schwefel ist zwar ein unnatürlicher Bestandteil, jedoch in allen Weinsorten zu finden. Der Schwefel verhindert den bakteriellen und oxydativen Zerfall des Weins und dient somit als *Konservierungsstoff*. Der pH-Wert schwankt in einer Skale von 0 bis 14. Je niedriger der Wert, desto stärker ist der saure Charakter.

Im nächsten Schritt werden die Daten eingelesen und die ersten Zeilen des Datensatzes angezeigt.

```
# Daten einlesen
data = pd.read_csv("data/winequality-red.csv", sep=";",
    decimal=".")

data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.7	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.2	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.997	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.998	3.16	0.58	9.8	6
4	7.4	0.7	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

6.3 Datenbereinigung und Datenlage

Im weiteren Verlauf wird die Datenlage beschrieben mithilfe geeigneter Visualisierungen. In dem Datensatz sind keine fehlenden Werte vorhanden. Des Weiteren besteht der Datensatz aus 11 Attribute, die quantitativ (kontinuierlich), und 1 Attribut, welches qualitativ (ordinal) geprägt ist. Nachfolgend werden die Datentype der einzelnen Attribute dargestellt.

```
data.info()
```

```
    <class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

6.3.1 Boxplot

Mithilfe von `ipywidgets` kann die Datenlage für jedes Attribut erklärt werden. Boxplots eignen sich hierfür besonders gut.

Resultat

Für die Attribute `resiudal sugar`, `chlorides`, `total sulfur dioxide`, `density` und `sulphates` sind besonders viele Outliers zu erkennen. Für die Zielvariable `quality` sind die Qualitätsstufen 3 und 8 Ausreißer. Bislang gibt es keinen Rotwein, der die Qualitätsstufe 10 besitzt.

```
# @source: (Vgl. Holscher u. a. 2020)
def f(Attribut):
    sns.set_context("talk")
    plt.figure(figsize=(15,7))
    sns.boxplot(x=Attribut, data=data).set(title="Boxplot: " +
        Attribut, xlabel="")

interact(f, Attribut=list(data.columns));
```

OUTPUT: siehe `Jupyter Notebook`

6.3.2 Histogramm

Neben Boxplots können die Verteilung der Daten nach ihrer Anzahl in Histogrammen dargestellt werden. Mittels `ipywidgets` werden die Histogramme für jedes Attribut dargestellt.

Resultat

Interessant zu beobachten ist, dass kein Attribut den Verlauf einer Normalverteilung hat. Entweder sind sie linksschief, rechtsschief oder sind in der Mitte breit gestreut.

```
# @source: (Vgl. Holscher u. a. 2020)
def g(Attribut):
    sns.set_context("talk")
    plt.figure(figsize=(15,7))
    sns.distplot(data[Attribut])

interact(g, Attribut=list(data.columns));
```

OUTPUT: siehe `Jupyter Notebook`

6.3.3 Scatterplot

Mithilfe von Scatterplots können Beziehungen zwischen 2 Attributen dargestellt werden. Nachfolgend können für alle Attribute die Beziehungen beobachtet werden.

Resultat

Eine Korrelation ist in den meisten Attributen nicht zu sehen. Die Korrelationsmatrix im Abschnitt Demonstration zeigt die Korrelationen zwischen einzelnen Attributen auf.

```
# @source: (Vgl. Holscher u. a. 2020)
def g(Attribut_x, Attribut_y):
    sns.set_context("talk")
    plt.figure(figsize=(15,7))
    sns.scatterplot(x=Attribut_x, y=Attribut_y, data=data,
        hue="quality")

interact(g, Attribut_x=list(data.columns),
   Attribut_y=list(data.columns));
```

OUTPUT: siehe `Jupyter Notebook`

6.3.4 Waffle Plot

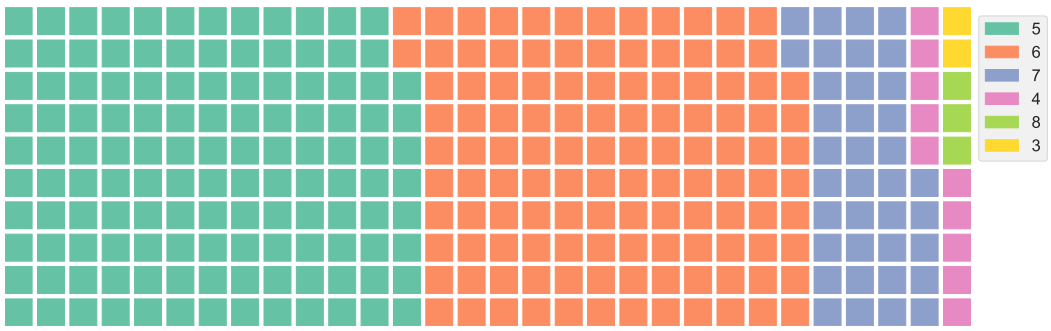
Der Waffleplot zeigt ideal die Ausprägungsmenge einzelner Attributwerte eines Attributs. Hierbei werden die Ausprägungen für die Zielvariable "quality" gezeigt.

Resultat

Die meisten Rotweine haben eine Qualitätsstufe von 5 und 6, nachfolgend mit 7. Nur wenige Weine haben eine Qualitätsstufe von 3, 4 und 8.

```
# @source:
    https://readthedocs.org/projects/pywaffle/downloads/pdf/latest/
waffle_data = pd.DataFrame(data["quality"].value_counts())
waffle_data["label"] = waffle_data.index
waffle_data.head()

fig = plt.figure(FigureClass = Waffle, rows = 5, columns=15,
              values = list(waffle_data["quality"]),
              labels=list(waffle_data["label"]),
              legend={'loc': 'upper left', 'bbox_to_anchor': (1,
                  1)})
```



6.4 Deklarierung der Einfluss- und Zielvariable

Wie bereits erwähnt wird die Zielvariable `quality` sein. Die Frage hierbei lautet: *Welche Einflussvariablen begünstigen eine gute oder schlechte Qualität des Rotweins?* Im Code werden alle Einflussvariablen als `X` gekennzeichnet, die Zielvariable mit `Y`.

Der Datensatz wird zudem in Trainings- und Testdaten gesplittet. Hierbei haben die Testdaten einen Anteil von 25% des Datensatzes; die Trainingsdaten dementsprechend 75%. Mithilfe der Angabe `random_state=1234` sind die Daten und die nachfolgende Analyse für den Zustand mit der Nummer `1234` reproduzierbar.

```
# Einfluss- und Zielvariable
X = data.drop(["quality"], axis=1)
Y = data["quality"]

# Daten splitten in Trainings- und Testdaten
X, x, Y, y = train_test_split(X, Y, test_size=0.25,
    random_state=1234)
```

6.5 Initialisierung des LASSO Modells und Cross-Validation

Um das LASSO Modell aufstellen zu können, muss vorher der beste λ-Wert ermittelt werden. Hierbei werden verschiedene λ-Werte in einer *Dictionary* zusammengefasst, beginnend mit 0 (entspricht einer normalen Regressionsanalyse ohne Schrumpfungsterm) bis hin zu 100. Mithilfe von `GridSearchCV()` lässt sich nun eine Cross-Validation (Unterteilung des Datensatzes in Teildatensätze, die abwechselnd als Test- und Trainingsdatensatz verwendet werden) durchführen (mit 5 Splittings).

```
# @source: Vgl. Peixeiro (2019)
lasso = linear_model.Lasso()

# Lambda Werte
lambda_values = {"alpha": [0, 0.000001, 0.00001, 0.0001, 0.001,
    0.01, 0.1, 1, 2, 5, 10, 25, 50, 100]}

# Ermittlung des besten Lambda
lasso_result = model_selection.GridSearchCV(lasso, lambda_values,
    scoring="neg_mean_squared_error", cv=5)
lasso_result.fit(X, Y)
```

```
GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [0, 1e-06, 1e-05, 0.0001,
                 0.001, 0.01, 0.1, 1,
                                   2, 5, 10, 25, 50, 100]},
             scoring='neg_mean_squared_error')
```

6.5.1 Bester λ-Wert

Der Ergebnis zeigt, dass der beste λ-Wert bei 0.0001 liegt.

```
lasso_result.best_params_
```

```
{'alpha': 0.0001}
```

6.5.2 Beste mittlere quadratische Abweichung (MSE)

Zu dem λ-Wert beträgt die niedrigste Summe der durchschnittlichen quadratischen Abweichungen -0,4416.

```
lasso_result.best_score_
```

```
-0.4416629737902393
```

6.5.3 Anzahl der Cross-Validation Splits

Insgesamt wurden 5 Splits für die Cross-Validation verwendet.

```
lasso_result.n_splits_
```

```
5
```

6.6 Visualisierung der Cross-Validation

Die Cross-Validation kann ebenfalls visualisiert werden. Hierzu wurde der Trainings- und Testdatensatz betrachtet.

6.6.1 für R^2

Der Kontrollparameter ist hierbei das Bestimmtheitsmaß R^2. Der beste λ-Wert zeigt sich dort, wo die höchste Modellgüte (das höchste Bestimmtheitsmaß) ist. Am Anfang liegen die Test- und Trainingsdatensätze dicht beieinander, später weichen sie stärker ab.

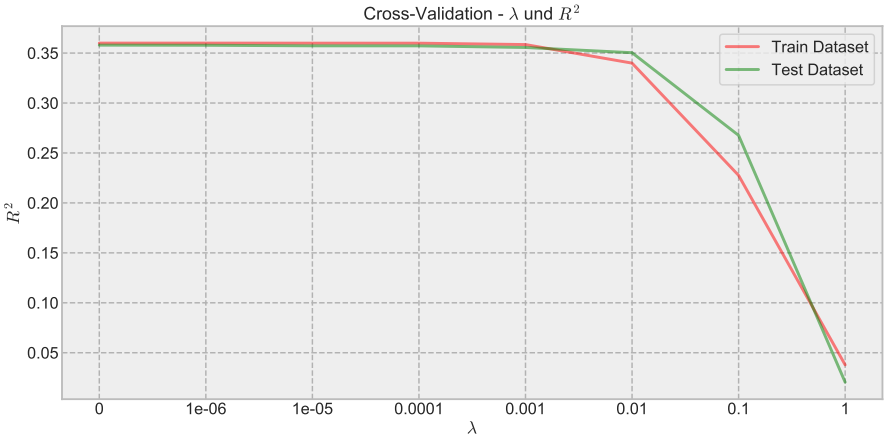
```
# @source: Vgl. Kirenz (2019)
# moegliche L-Werte
lambda_values = (0, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1,
    1)

# Listen fuer Cross-Validation Ergebnisse
lambdas = []
r2_train = []
r2_test = []

# Cross-Validation fuer jeden L-Wert + Speicherung
for i in lambda_values:
    reg = linear_model.Lasso(alpha=i)
    reg.fit(X, Y)
    r2_score = cross_val_score(reg, X, Y, cv=5, scoring="r2")
    lambdas.append(reg.coef_)
    r2_train.append(reg.score(X, Y))
    r2_test.append(reg.score(x, y))

# Erstellung des DataFrame
results = pd.DataFrame(list(zip(r2_train, r2_test)),
    columns=["R2_Train", "R2_Test"])
results["L"] = lambdas

# Erstellung des Plots
plt.figure(figsize=(15,7))
plt.plot(results["R2_Train"], color="red", alpha=0.5,
    label="Train Dataset", linewidth=3)
plt.plot(results["R2_Test"], color="green", alpha=0.5,
    label="Test Dataset", linewidth=3)
plt.xticks(range(0, 8), labels=lambda_values)
plt.title("Cross-Validation - $\lambda$ und $R^2$")
plt.xlabel("$\lambda$")
plt.ylabel("$R^2$")
plt.legend()
```



6.6.2 für *AIC* und *BIC*

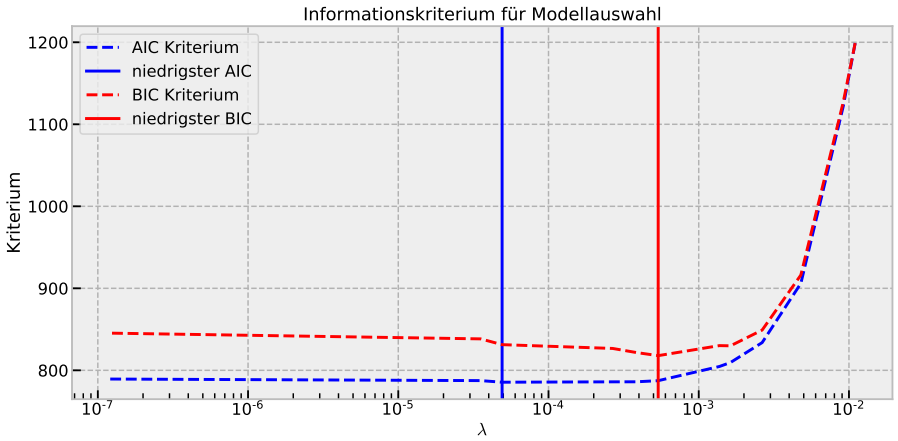
Ein weiterer Kontrollparameter sind die Informationskriterien *BIC* und *AIC*. Hierbei wird der λ-Wert genommen, wo die beiden Informationskriterien den niedrigsten Wert haben. In der Abbildung sind 2 Abweichungen zu erkennen. Beide jedoch grenzen die Wahl des besten λ-Wertes ein zwischen 0.0001 und 0.001 ein.

```
# @source: Pedregosa, F u. a. (2020b)
# BIC
model_bic = linear_model.LassoLarsIC(criterion='bic')
model_bic.fit(X, Y)
alpha_bic_ = model_bic.alpha_

# AIC
model_aic = linear_model.LassoLarsIC(criterion='aic')
model_aic.fit(X, Y)
alpha_aic_ = model_aic.alpha_

# Cross-Validation
def plot_ic_criterion(model, name, color):
    criterion_ = model.criterion_
    plt.semilogx(model.alphas_ + EPSILON, criterion_, '--',
        color=color,
             linewidth=3, label='%s Kriterium' % name)
    plt.axvline(model.alpha_ + EPSILON, color=color, linewidth=3,
            label='niedrigster %s' % name)
    plt.xlabel(r'$\lambda$')
    plt.ylabel('Kriterium')

# Erstellung des Plots
plt.figure(figsize=(15,7))
plt.title("Informationskriterium fuer Modellauswahl")
plot_ic_criterion(model_aic, 'AIC', 'b')
plot_ic_criterion(model_bic, 'BIC', 'r')
plt.legend()
```


Informationskriterium für Modellauswahl
AIC Kriterium
niedrigster AIC
BIC Kriterium
niedrigster BIC
Kriterium
1200
1100
1000
900
800
10^{-7}
10^{-6}
10^{-5}
10^{-4}
10^{-3}
10^{-2}
λ

6.7 Modellerstellung

Nachdem der beste λ-Wert ermittelt wurde, kann nun die LASSO Regression durchgeführt werden.

```
# LASSO Modell mit besten Lambda
reg_best_lambda = linear_model.Lasso(alpha=0.0001)
reg_train = reg_best_lambda.fit(X, Y)
reg_train_score = reg_train.score(X, Y)
reg_test_score = reg_best_lambda.score(x, y)

# LASSO Modell mit Koeffizienten
modell = pd.DataFrame(list(zip(reg_train.coef_, X.columns)),
    columns=["Koeffizient", "Einflussvariable"])
```

6.7.1 R^2 für Trainingsdaten

Das Bestimmtheitsmaß liegt bei $R^2 = 0.3599$. Es werden demnach 35.99% der Daten mit dem Modell erklärt.

```
reg_train_score
```

```
0.35990924714385286
```

6.7.2 R^2 für Testdaten

Das Bestimmtheitsmaß liegt bei $R^2 = 0.3572$. Es werden demnach 35.72% der Daten mit dem Modell erklärt. Beachte hierbei, dass die Modellgüten für die Test- und Trainingsdaten sehr ähnlich sind.

```
reg_test_score
```

```
0.3572333813261682
```

6.7.3 Modell mit Intercept und Koeffizienten

Nun wird das Modell erstellt mit den konstanten Parameter β_0 (Intercept) und den Steigungsparametern β_n der Einflussvariablen X. Basierend auf den Ergebnissen ergibt sich folgendes LASSO Regressionsmodell:

$$\begin{aligned} \hat{y} = 4.725531404346774 - 0 \times x_1 \\ -1.149796 \times x_2 - 0.167023 \times x_3 \\ -0.000660 \times x_4 - 1.688271 \times x_5 \\ +0.005306 \times x_6 - 0.003672 \times x_7 \\ -0 \times x_8 - 0.549538 \times x_9 \\ +0.911532 \times x_{10} + 0.289937 \times x_{11} \end{aligned} \tag{6}$$

Die Einflussvariable `density` hat hierbei ein Koeffizient von 0, wodurch die Variable ausgeschlossen wird. Demnach ergibt sich das folgende LASSO Regressionsmodell:

$$\begin{aligned}\hat{y} = 4.725531404346774 - 1.149796 \times x_2 \\ -0.167023 \times x_3 - 0.000660 \times x_4 \\ -1.688271 \times x_5 + 0.005306 \times x_6 \\ -0.003672 \times x_7 - 0.549538 \times x_9 \\ +0.911532 \times x_{10} + 0.289937 \times x_{11}\end{aligned} \tag{7}$$

```
reg_best_lambda.intercept_
```

```
4.725531404346774
```

```
modell
```

	Koeffizient	**Einflussvariable**
0	-0.0	fixed acidity
1	-1.149795500312533	volatile acidity
2	-0.1670233425735125	citric acid
3	-0.0006597805189011724	residual sugar
4	-1.688271134894735	chlorides
5	0.005306090468115356	free sulfur dioxide
6	-0.003671665715485828	total sulfur dioxide
7	-0.0	density
8	-0.5495380237181456	pH
9	0.9115317854659596	sulphates
10	0.28993738825595905	alcohol

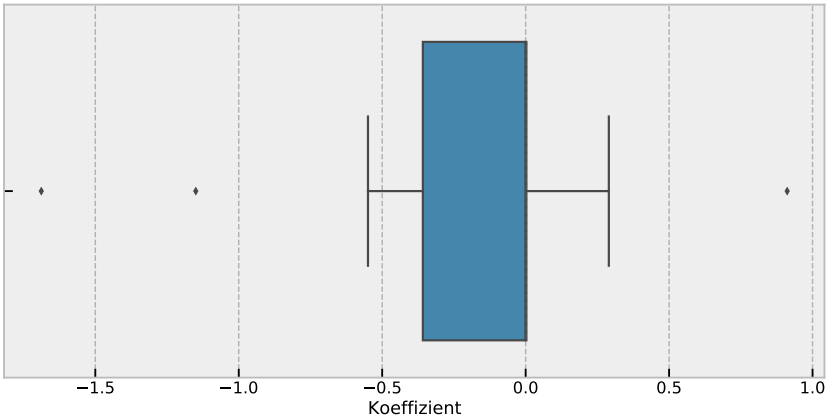
6.7.4 Sortiertes Modell mit Koeffizienten

Um die relvenaten Einflussvariablen nach der LASSO Methode identifizieren zu können, empfiehlt es sich, die Koeffizienten nach absteigender Reihenfolge zu sortieren. Die Einflussvariablen `sulphates`, `chlorides` und `volatile acidity` weichen besonders stark von der 0 ab. Dies kann mithilfe eines Boxplots ebenfalls dargestellt werden.

```
modell.sort_values(["Koeffizient"], ascending=False)
```

```
# Boxplots fuer Koeffizienten mit Ausreisser
plt.figure(figsize=(15,7))
sns.boxplot(modell["Koeffizient"])
```

	Koeffizient	Einflussvariable
9	0.9115317854659596	sulphates
10	0.28993738825595905	alcohol
5	0.005306090468115356	free sulfur dioxide
0	-0.0	fixed acidity
7	-0.0	density
3	-0.0006597805189011724	residual sugar
6	-0.003671665715485828	total sulfur dioxide
2	-0.1670233425735125	citric acid
8	-0.5495380237181456	pH
1	-1.149795500312533	volatile acidity
4	-1.688271134894735	chlorides



Neben den Boxplot für die Koeffizienten können auch die Boxplots für die Ausreißer dargestellt werden. Hierzu werden im Code die Rechenvorschriften für die Berechnung der Ausreißer festgelegt. Anschließend wird in einer Funktion die Darstellung der Inputparameter mithilfe eines Widgets eingestellt.

Die Formula für die Berechnung der Ausreißer lautet (hierbei ist Q das Quantil, dementsprechend $Q_{25\%}$ das 25%-Quantil): (Vgl. Hubert und Vandervieren 2008, S. 5186)

$$\begin{aligned} \text{oberer Whiskers} &= Q_{75\%} + (1.5 \times Q_{75\%} - Q_{25\%}) \\ \text{unterer Whiskers} &= Q_{25\%} - (1.5 \times Q_{75\%} - Q_{25\%}) \end{aligned} \tag{8}$$

```
# Boxplots fuer Einflussvariablen als Ausreisser
# @source: (Vgl. Holscher u. a. 2020)

outliers = np.array(sorted(modell["Koeffizient"]))
down_Whisker = np.percentile(outliers,25) - (1.5 *
    (np.percentile(outliers,75) - np.percentile(outliers,25)))
top_Whisker = np.percentile(outliers,75) + (1.5 *
    (np.percentile(outliers,75) - np.percentile(outliers,25)))
```

```
outliers = list(outliers[(outliers < down_Whisker) | (outliers >
    top_Whisker)])
modell_outliers = modell[modell["Koeffizient"].isin(outliers)]

def h(Outlier):
    sns.set_context("talk")
    plt.figure(figsize=(15,7))
    plt.title("Boxplot: " + Outlier + " ~ " + Y.name)
    plt.xlabel(Y.name)
    sns.boxplot(x=Y.name, y=Outlier, data=data, whis=np.inf,
        boxprops=dict(alpha=.3))
    sns.stripplot(x=Y.name, y=Outlier, data=data, linewidth=1,
        alpha=1)

interact(h, Outlier=list(modell_outliers["Einflussvariable"]));
```

OUTPUT: siehe `Jupyter Notebook`

6.7.5 Vergleich der Regressionen

Nachfolgend wird die LASSO Regression mit der linearen Regression verglichen.

Lineare Regression

```
X2 = sm.add_constant(X)
model_ols = sm.OLS(Y, X2)
model_ols_fit = model_ols.fit()
model_ols_fit.summary()
```

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	13.0806	24.482	0.534	0.593	-34.951	61.112
fixed acidity	0.0074	0.030	0.247	0.805	-0.052	0.066
volatile acidity	-1.1457	0.139	-8.250	0.000	-1.418	-0.873
citric acid	-0.1730	0.171	-1.010	0.313	-0.509	0.163
residual sugar	0.0029	0.017	0.169	0.866	-0.031	0.037
chlorides	-1.7369	0.482	-3.601	0.000	-2.683	-0.790
free sulfur dioxide	0.0052	0.003	2.044	0.041	0.000	0.010
total sulfur dioxide	-0.0037	0.001	-4.103	0.000	-0.005	-0.002
density	-8.4682	24.971	-0.339	0.735	-57.461	40.524
pH	-0.5224	0.224	-2.334	0.020	-0.961	-0.083
sulphates	0.9312	0.134	6.966	0.000	0.669	1.193
alcohol	0.2819	0.031	9.143	0.000	0.221	0.342

LASSO Regression

```
modell
```

	Koeffizient	**Einflussvariable**
9	0.9115317854659596	sulphates
10	0.28993738825595905	alcohol
5	0.005306090468115356	free sulfur dioxide
0	-0.0	fixed acidity
7	-0.0	density
3	-0.0006597805189011724	residual sugar
6	-0.003671665715485828	total sulfur dioxide
2	-0.1670233425735125	citric acid
8	-0.5495380237181456	pH
1	-1.149795500312533	volatile acidity
4	-1.688271134894735	chlorides

6.8 Anwendungsvoraussetzungen

Bei den Anwendungsvoraussetzungen werden die Modellprämissen des linearen Regressionsmodells überprüft. Da die LASSO Regression auf die lineare Regression aufbaut, wird im Folgenden eine lineare Regression durchgeführt, um zu überprüfen, ob die Daten und das Modell für die lineare und somit LASSO Regression geeignet werden.

Am Anfang wird ein Datensatz erstellt, wo neben den tatsächlichen Werten (*actual values*) auch die vorhergesagten Werte (*predicted values*) und die Residualwerte (*residual values*) zusammengefasst sind.

```
# @source: Vgl. Macaluso (2018)
# Initialisierung des linearen Regressionmodells
assumptions = linear_model.LinearRegression()
assumptions.fit(X, Y)

# Erstellung der tatsaechlichen und vorhergesagten Werte sowie
    Residualwerte
actual_values = Y
predicted_values = assumptions.predict(X)
values = pd.DataFrame({"Tatsaechliche_Werte": actual_values,
    "Vorhergesagte_Werte": predicted_values})
values["Residual_Werte"] = abs(values["Tatsaechliche_Werte"]) -
    abs(values["Vorhergesagte_Werte"])
```

	Tatsaechliche_Werte	**Vorhergesagte_Werte**	**Residual_Werte**
1087	6	6.402564174534436	-0.4025641745344357
1445	6	4.891965860407263	1.1080341395927373
411	5	5.307545561661218	-0.30754556166121816
122	5	4.9868713577835795	0.013128642216420516
652	5	7.362053157536247	-2.3620531575362467
87	5	5.454732621205888	-0.4547326212058884
1076	6	6.375129432511961	-0.37512943251196074
1023	6	6.366313238726995	-0.3663132387269954
477	6	6.609244565202268	-0.6092445652022684
1495	6	6.046137896999466	-0.04613789699946569

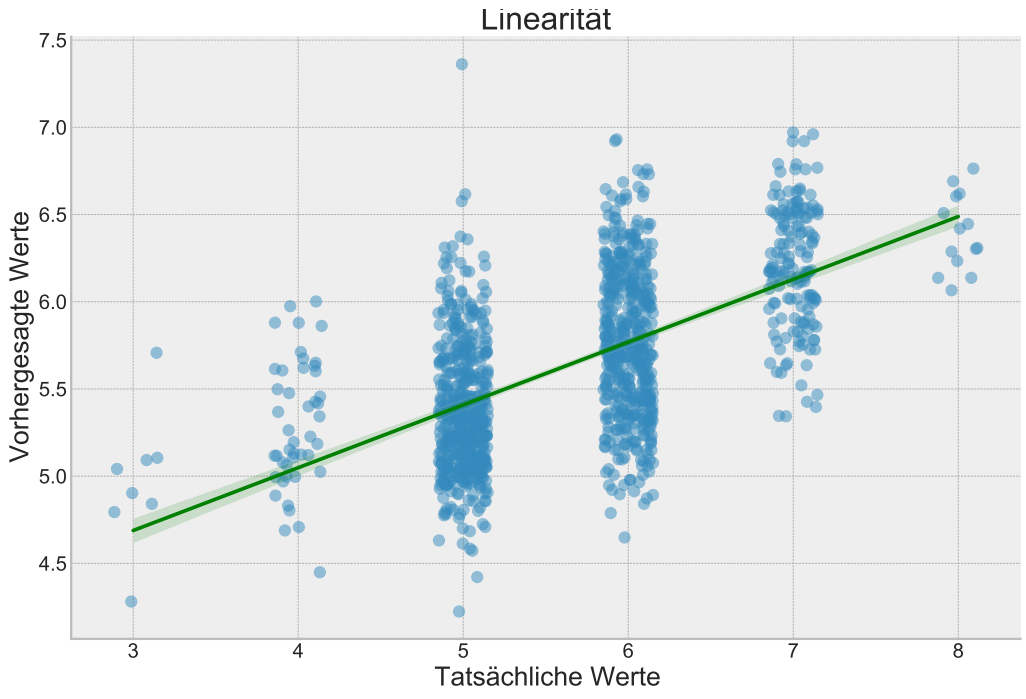
6.8.1 Linearität und korrekte Spezifizierung

Die Linearität kann mithilfe der Visualisierung der tatsächlichen Werte mit den vorhergesagten Werten überprüft werden. Die tatsächlichen Werte sind die vorliegenden Werte der Zielvariablen. Die vorhergesagten Werte sind die Werte, die mithilfe des linearen Modells erzeugt werden. Weichen die tatsächlichen Werte mit den vorhergesagten Werten gleichmäßig von der linearen Regressionsgeraden ab, so liegt eine Linearität vor. (Vgl. L. Li 2018)

Resultat

Im Plot ist zu erkennen, dass die vorhergesagten Werte relativ gleichmäßig von der Regressionsgeraden abweichen. Somit ist die 1. Voraussetzung erfüllt.

```
sns.lmplot(x="Tatsaechliche_Werte", y="Vorhergesagte_Werte",
    data=values, line_kws={"color" : "green"},
        height=8, aspect=15/10)
plt.title("Linearitaet")
plt.xlabel("Tatsaechliche Werte")
plt.ylabel("Vorhergesagte Werte")
```



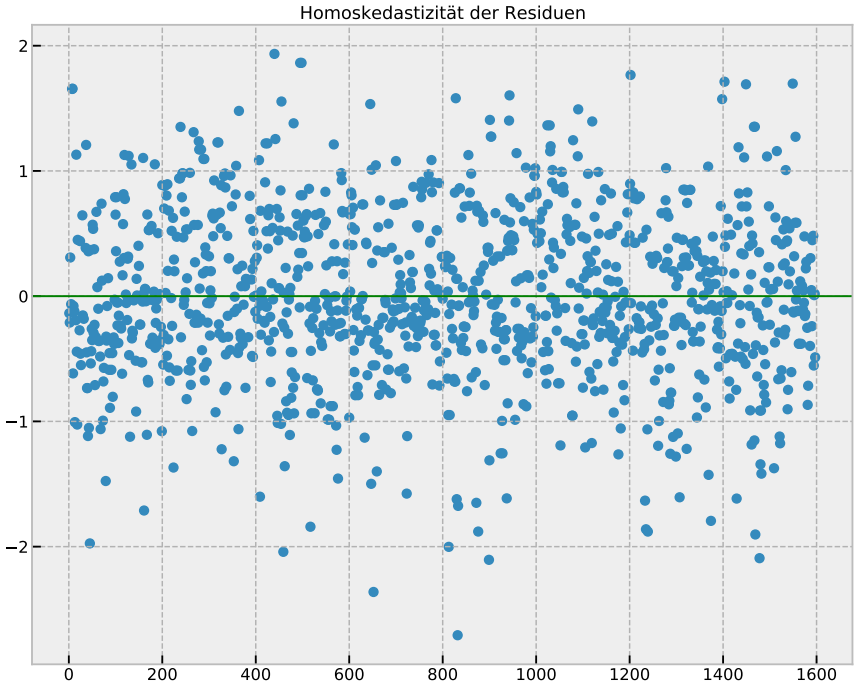
6.8.2 Homoskedastizität

Die Homoskedastizität kann mithilfe einer Visualisierung dargestellt werden. Hierbei wird zu jedem Zeileneintrag des Datensatzes mithilfe des Indexes die Residualwerte um die horizontale Achse bei 0 dargestellt. Sind alle Residualwerte gleichmäßig um die Achse verteilt, so liegt eine Homoskedastizität vor.

Resultat

Die meisten Residuen weichen gleichmäßig über alle Beobachtungen hinweg um die horizontale Achse bei 0 ab. Somit ist die 2. Voraussetzung erfüllt.

```
plt.title("Homoskedastizitaet der Residuen")
plt.scatter(x=values.index, y=values["Residual_Werte"])
plt.axhline(0, xmin=0, xmax=1, linewidth=2, color="green")
```



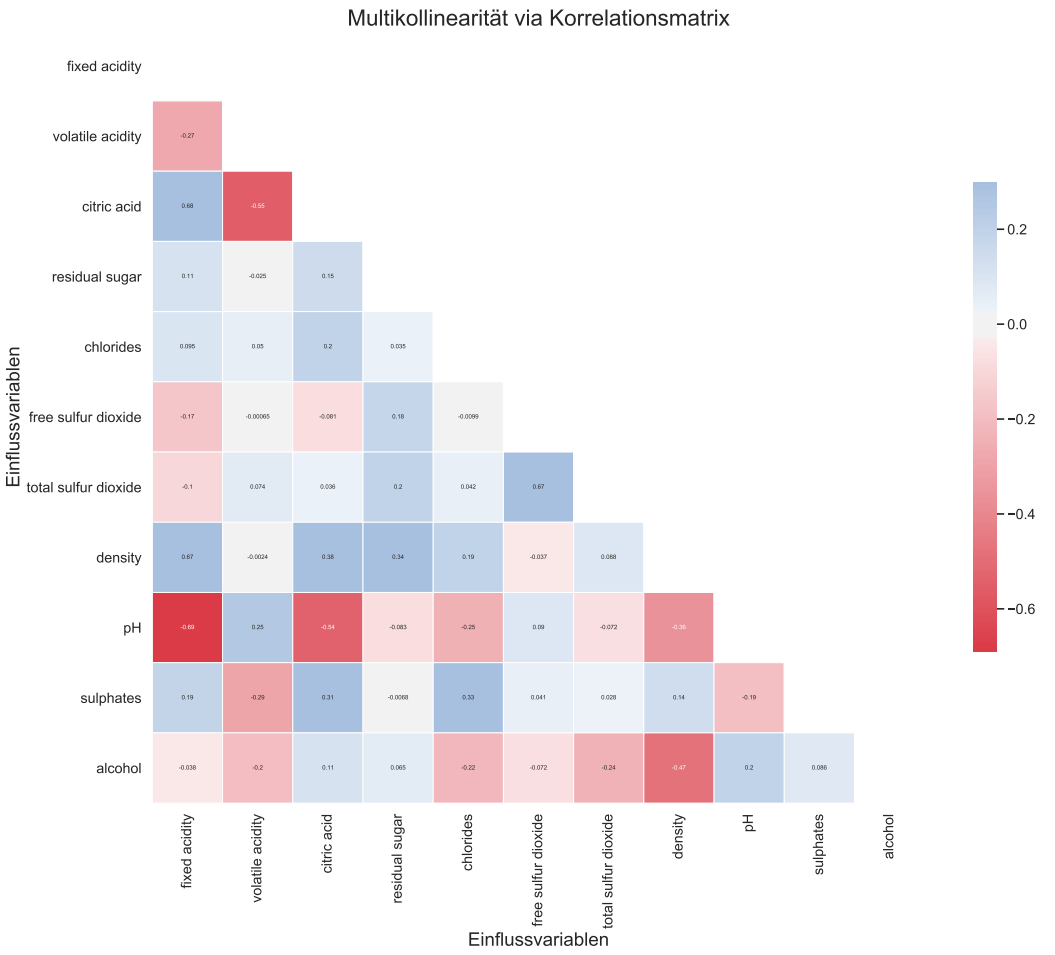
6.8.3 Multikollinearität

Mithilfe einer Korrelationsmatrix kann die Multikollinearität visualisiert werden. Dies erfolgt mit `seaborn.heatmap`. Die Korrelationen liegen hierbei zwischen -1 und 1. Je näher die Korrelationen an der -1 und 1, desto stärker ist die lineare Abhängigkeit zweier Einflussvariablen zueinander.

Resultat

In der Korrelationsmatrix wird ersichtlicht, dass eine relativ starke negative Korrelation zwischen `pH` und `fixed acidity`, `critic acid` und `volatile acidit`, `pH` und `critic acid` sowie `alcohol` und `density` vorliegt. Da die Korrelationen jedoch nicht $Cor(X) = 1$ betragen, liegt keine perfekte Korrelation vor und dementsprechend keine absolute Redundanz von Einflussvariablen innerhalb des Modells. Somit ist die 3. Voraussetzung (trotz einzelner starker Korrelationen) erfüllt.

```
# @source: Vgl. Waskom (2020)
plt.figure(figsize=(15,7))
multi_corr = X.corr()
mask = np.triu(np.ones_like(multi_corr, dtype=bool))
sns.set_style(style="white")
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(10, 250, as_cmap=True)
sns.heatmap(multi_corr, mask=mask, cmap=cmap, vmax=.3, center=0,
        square=True, linewidths=.5, cbar_kws={"shrink": .5},
        annot=True, annot_kws={"size": 7})
plt.title("Multikollinearitaet via Korrelationsmatrix")
plt.xlabel("Einflussvariablen")
plt.ylabel("Einflussvariablen")
```


Multikollinearität via Korrelationsmatrix
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
Einflussvariablen
Einflussvariablen
0.2
0.0
−0.2
−0.4
−0.6

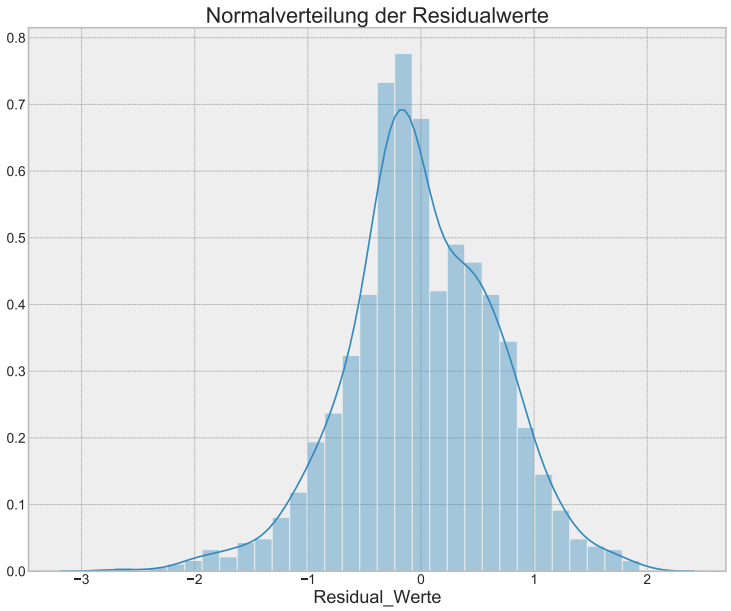
6.8.4 Normalverteilung der Residuen

Mithilfe von `seaborn.distplot` kann ein Histogramm für die Verteilung der Residualwerte erzeugt werden. Sind die Residualwerte gleichermaßen um die 0 verteilt, so liegt eine Normalverteilung vor. Paralell zur Normalverteilung wird der Anderson-Darling-Test verwendet mithilfe von `scipy.stats.anderson`. Der Test berechnet einen Wert zur Aussage, ob eine Normalverteilung vorliegt oder nicht. Ist der errechnete Wert kleiner als die Werte für die Signifkanzniveaus $\alpha = \{1\%, 2.5\%, 5\%, 10\%, 15\%\}$, so liegt eine Normalverteilung mit zunehmender Wahrscheinlichkeit vor. Anderfalls liegt keine Normalverteilung vor.

Resultat

Im Histogramm ist die Verteilung der Residuen dargestellt. Hierbei fällt auf, dass eine exakte Normalverteilung nicht vorliegt. Die Kurve hat im positiven Zahlenspektrum einen größeren Verlauf als im negativen Bereich. Der Anderson-Darling-Test berechnete eine Teststatistik in Höhe von 2.20546 sowie die kritischen Werte für die jeweiligen Signifkanzniveaus. Die Teststatistik ist größer als alle anderen kritischen Werte. Demnach liegt ein Signifkanzniveau vor, was größer als 15% ist und somit eine Normalverteilung aufgrund niedriger Wahrscheinlichkeit auszuschließen ist. Somit ist die 4. Voraussetzung nicht erfüllt.

```
style.use('bmh')
sns.distplot(values["Residual_Werte"])
plt.title("Normalverteilung der Residualwerte")
```



```
sps.anderson(values["Residual_Werte"], dist="norm")
```

```
AndersonResult(statistic=2.205462886105124,
    critical_values=array([0.574, 0.654, 0.784, 0.915,
    1.088]), significance_level=array([15. , 10. ,  5. ,  2.5,
     1. ]))
```

6.8.5 Autokorrelation

Die Autokorrelation ist ein Phänomen, welches bei Zeitreihendaten auftretet. Eine Autokorrelation kann auch bei Querschnittsdaten auftreten, sofern eine Pfadabhängigkeit zwischen einzelnen Variablen besteht.
Mithilfe von `statsmodels.stats.stattools.durbin_watson` kann der Durbin-Watson-Test durchgeführt werden. Hierbei liegt der Wert zwischen 0 und 4.

- Je näher der Wert an der 0, desto wahrscheinlicher ist es, dass eine **positive** Autokorrelation vorliegt.
- Je näher der Wert an der 4, desto wahrscheinlicher ist es, dass eine **negative** Autokorrelation vorliegt.
- Je näher der Wert an der 2, desto wahrscheinlicher ist es, dass **keine** Autokorrelation vorliegt. (Vgl. Perktold, Seabold und Taylor 2020)

Resultat

Beim Durbin-Watson-Test wurde ein Wert in Höhe von 2.0777 berechnet. Da der Wert sich nah an der 2 befindet, liegt mit hoher Wahrscheinlichkeit keine Autokorrelation vor. Somit ist die 5. Voraussetzung erfüllt.

```
stm.durbin_watson(values["Residual_Werte"])
```

```
2.0777657084700154
```

6.8.6 Zusammenfassung der Anwendungsvoraussetzungen

Nachfolgend werden tabellarisch die Ergebnisse der Anwendungsvoraussetzungen sowie deren Folgen zusammengefasst:

Anwendungsvoraussetzung	**Ergebnis**	**Folgen**
Linearität und korrekte Spezifizierung	Parameter sind linear	lineares Modell
Homoskedastizität	Residuen variieren gleichmäßig	keine Verzerrung des Modells
Multikollinearität	vereinzelte starke Korrelation	keine Redundanz im Modell
Normalverteilung der Residuen	keine Normalverteilung	Verzerrung des Modells
Autokorrelation	keine Autokorrelation	keine Verzerrung des Modells

7 Übung

Neben den Daten für Rotweine sind ebenfalls auch Daten für Weißweine vorhanden. Führe mit den oben genannten Schritten ebenfalls eine vollständige Analyse mithilfe der LASSO Regression durch. Du findest den Datensatz unter `data/winequality-white.csv`. Beachte folgende Schritte:

1. Analysiere die Datenlage und führe ggf. eine Datenbereinigung durch.
2. Untersuche die Daten mithilfe einer einfachen linearen Regression. Untersuche dabei die Koeffizienten der Einflussvariablen sowie die p-Werte. (Hinweis: je niedriger die Werte, desto wahrscheinlicher ist die Beziehung der Einflussvariable auf die Zielvariable.)
3. Führe nun eine LASSO Regression durch. Ermittle dabei den besten λ mithilfe einer Cross-Validation.
4. Sind Ausreißer enthalten? Falls ja, um welche Einflussvariablen handeln es sich?
5. Inwiefern können die Ergebnisse der einfachen linearen Regression und LASSO Regression miteinander verglichen werden?

Literatur

Monographien

Backhaus, Klaus u. a. (2016). *Multivariate Analysemethoden*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Gujarati, Damodar N. und Dawn C. Porter (2009). *Essentials of Econometrics*. 4. Aufl. McGraw-Hill Education, S. 554.

James, Gareth u. a. (2013). *An Introduction to Statistical Learning*. Bd. 103. Springer New York.

Zeitschriftenaufsätze

Anderson, T. W. und D. A. Darling (Juni 1952). „Asymptotic Theory of Certain "Goodness of FitCriteria Based on Stochastic Processes“. In: *The Annals of Mathematical Statistics* 23.2, S. 193–212.

Cortez, Paulo u. a. (Nov. 2009). „Modeling wine preferences by data mining from physicochemical properties“. In: *Decision Support Systems* 47.4, S. 547–553.

Deng, Bai-Chuan u. a. (2015). „A new strategy to prevent over-fitting in partial least squares models based on model population analysis“. In: 880, S. 32–41.

Hubert, M. und E. Vandervieren (Aug. 2008). „An adjusted boxplot for skewed distributions“. In: *Computational Statistics & Data Analysis* 52.12, S. 5186–5201.

Muthukrishnan, R und R Rohini (Okt. 2016). „LASSO: A feature selection technique in predictive modeling for machine learning“. In: S. 18–20.

Pedregosa, F u. a. (2011). „Scikit-learn: Machine Learning in Python“. In: 12, S. 2825–2830.

Pereira, Jose Manuel u. a. (2016). „The Logistic Lasso and Ridge Regression in Predicting Corporate Failure“. In: *Procedia Economics and Finance* 39, S. 634–641.

Rashidi, Hooman H u. a. (2019). „Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods“. In: 6.

Santosa, Fadil und William W. Symes (Okt. 1986). „Linear Inversion of Band-Limited Reflection Seismograms“. In: *SIAM Journal on Scientific and Statistical Computing* 7.4, S. 1307–1330.

Tibshirani, Robert (1996). „Regression Shrinkage and Selection via the Lasso“. In: 58.1, S. 267–288.

Software

Holscher, Eric u. a. (2020). *Using Interact*.

Li, Guangyang (2020). *PyWaffle Documentation*.

Matplotlib development team (2017). *pylab_ examples example code: broken_ axis.py*.

Pedregosa, F u. a. (2020a). *Lasso*.

– (2020b). *Lasso model selection: Cross-Validation / AIC / BIC*.

– (2020c). *sklearn.linear_ model.Lasso*.

– (2020d). *sklearn.linear_ model.LassoCV*.

– (2020e). *sklearn.linear_ model.LassoLarsIC*.

– (2020f). *sklearn.model_ selection.cross_ val_ score*.

– (2020g). *sklearn.model_ selection.GridSearchCV*.

– (2020h). *sklearn.model_ selection.train_ test_ split*.

Perktold, Josef u. a. (2020). *statsmodels.stats.stattools.durbin_ watson*.

Waskom, Michael (2020). *Plotting a diagonal correlation matrix*.

Sonstige Schriften

Dua, Dheeru und Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Fonti, Valeria und Eduard Belitser (2017). *Feature Selection using LASSO*.

Hayden, Andy (2014). *Find all columns of dataframe in Pandas whose type is float, or a particular type?* URL: https://stackoverflow.com/questions/21720022/find-all-columns-of-dataframe-in-pandas-whose-type-is-float-or-a-particular-typ (besucht am 23. 10. 2020).

Kirenz, Jan (2019). *Lasso Regression with Python*. Hrsg. von Jan Kirenz. URL: https://www.kirenz.com/post/2019-08-12-python-lasso-regression-auto.

Li, Lorraine (2018). *Introduction to Linear Regression in Python*. URL: https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0 (besucht am 25. 10. 2020).

Macaluso, Jeff (2018). *Testing Linear Regression Assumptions in Python*. URL: https://jeffmacaluso.github.io/post/LinearRegressionAssumptions/.

Peixeiro, Marco (2019). *How to Perform Lasso and Ridge Regression in Python*. URL: https://towardsdatascience.com/how-to-perform-lasso-and-ridge-regression-in-python-3b3b75541ad8 (besucht am 23. 10. 2020).

Rhein-Ahr-Wein (2020). *WEIN INHALTSSTOFFE: WAS IST ALLES DRIN?* URL: https://rhein-ahr-wein.de/blogs/weinwissen/wein-inhaltsstoffe (besucht am 28. 10. 2020).

Singh, Seema (2018). *Understanding the Bias-Variance Tradeoff*. Hrsg. von Towards Data Science. URL: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229.

UCI Machine Learning (2017). *Red Wine Quality*. URL: https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009.