
Datenanalyse in der Physik

Spring Semester 2026

Lecture Notes

Contents

1 Einführung in Python	1
1.1 Ein Erstes Python Programm	1
2 Die Messung als Zufallsprozess	2
2.1 Fehlerfortpflanzung	3
2.2 Korrelationen	4
3 Die spektrale Leistungsdichte	5
3.1 Filtern	5
4 Bayesische Statistik	6
4.1 Charakteristiken von Wahrscheinlichkeitsverteilungen	6
4.2 Momente	6
4.3 Bayes'sche Datenanalyse	7
4.4 Wahrscheinlichkeitsverteilungen	8
4.5 Likelihood-Funktion	9
5 Machine Learning	11
5.1 Neuronale Netze	13
5.2 Nicht-numerische Daten	14
5.3 Clustering	14

References

- [1] Vorlesungsskript

Datenanalyse in der Physik

Fynn Krebsler—fkrebsler@student.ethz.ch

Spring Semester 2026

Lec 1

1 Einführung in Python

Wir verwenden Python als Programmiersprache, unter anderem, da Python sehr populär ist. Ein Grund dafür ist, dass Python einfach zu lernen und zu lesen ist. Weiter ist Python interaktiver als zum Beispiel C++. Als Programmierumgebung in diesem Kurs verwenden wir Jupyter Notebooks.

1.1 Ein Erstes Python Programm

Betrachten wir das folgende Python Programm:

```
def compute_mean(measurements):  
    """Compute mean of a list of measurements."""  
  
    if len(measurements) == 0:  
        return 0  
  
    sum = 0  
    for value in measurements:  
        sum += value  
    mean = sum / len(measurements)  
    return mean
```

```
measurements = [1, 0.1, 2.0]  
compute_mean(measurements)
```

Kommentare können auf Zwei Arten geschrieben werden. Einzeilige Kommentare beginnen mit einem # und mehrzeilige Kommentare werden von drei Anführungszeichen umschlossen.

In Python haben Variablen keinen Typ. Dafür müssen Variablen initialisiert werden. Der Typ wird dann automatisch bestimmt.

```
a = 3  
a/2
```

In diesem Beispiel erhalten wir 1.5.

Weiter wichtig in Python das Codeblöcke durch Einrückungen definiert werden.

Das if/else statement in Python sieht wie folgt aus:

```
if condition:  
    # do something  
elif condition2:  
    # do something else  
else:  
    # do something else
```

Ähnlich funktionieren while Schleifen:

```
x = 0  
while x < 5:  
    print(x)  
    x = x+1
```

Eine for Schleife sieht wie folgt aus:

```
for x in range(0,5):  
    print(x)
```

Hierbei erzeugt range(0,5) eine Liste mit den Zahlen 0,1,2,3,4.

In Python können Listen unterschiedliche Typen besitzen und sehen wie folgt aus:

```
l = [3, "Hund", "Katze"]
```

Die Indexierung ist analog zu C++ 0-basiert. Ausserdem hat das letzte Element Index -1.

Eine hilfreiche Funktion kann die help Funktion sein. Diese gibt die Dokumentation eines Objekts zurück. Zum Beispiel:

```
help(l)
```

gibt die Dokumentation der Liste zurück.

In Python ist ALLES eine Referenz. Wenn wir also b=a setzen und a bearbeiten, dann wird auch b bearbeitet.

Auch wichtig sind Tupel. Diese sind ähnlich wie Listen, aber unveränderlich. Sie werden mit runden Klammern definiert:

```
t = (1, 2, 3)
```

Die Klammern sind optional, aber es ist eine gute Praxis sie zu verwenden.

Tupel unpacking ist eine nützliche Funktion, die es ermöglicht, die Werte eines Tupels in einzelne Variablen zu entpacken. Zum Beispiel:

```
number, name = (1, "Hund")
```

Funktionen werden mit dem def Schlüsselwort definiert, wie im Einführungsbeispiel. Um mehrere Werte zurückzugeben, kann ein Tupel verwendet werden.

Sinnvolle Wissenschaftliche Bibliotheken beinhalten unter anderem: NumPy, SciPy, Pandas und Matplotlib.

Ein Array wird in NumPy wie folgt erstellt:

```
array_1d = np.array([1, 2, 3])  
array_2d = np.array([[1, 2, 3], [4, 5, 6]])  
array_range = np.arange(0,4, step=0.5)
```

Mathematische Operationen können direkt auf Arrays angewendet werden:

```
np.square(array_1d)  
np.sqrt(array_1d)
```

Um Matplotlib zu importieren, verwenden wir:

```
import matplotlib.pyplot as plt
```

2 Die Messung als Zufallsprozess

Ein Modell ist eine vereinfachte Vorstellung welches nie vollständig und nicht bewiesen werden kann. Mit einem Experiment können wir es lediglich falsifizieren.

Ein Experiment hingegen ist nie fehlerfrei und auch die Interpretation ist nicht eindeutig.

Betrachten wir als Modell periodische Funktionen. Hierbei ist $\phi(t)$ die Phase und es gilt

$$\tan(\phi) = \frac{\Im(z)}{\Re(z)}.$$

Mit einer linearen zeitabhängigen Phase erhalten wir eine Oszillation der Form

$$x(t) = x_0 \cdot \Re(e^{i\phi(t)}).$$

Der getriebene gedämpfte harmonische Oszillator erfüllt die Gleichung:

$$\ddot{x} + \Gamma \dot{x} + \omega_0^2 x = \frac{F_0 \cos(\omega t)}{m}.$$

Nach einer langen Zeit erreicht die Lösung die Form

$$x_0(\omega) = \left| \frac{F_0/m}{\omega_0^2 - \omega^2 + i\Gamma\omega} \right|.$$

Wenn wir einen Quarzkristall haben, dann sieht die Resonanzkurve wie folgt aus: INSERT FIGURE

Der unterschied zur erwarteten Kurve entsteht dadurch, dass die beiden Platten einen Kondensator bilden, welcher die Resonanzfrequenz verschiebt. somit ist

$$I_{\text{in}}(\omega) \propto |x(\omega) + K(\omega)| \neq x_0.$$

Somit haben wir einen systematischen Fehler begangen. Das Modell war nicht vollständig. Mit einem besseren Modell können wir diesen Fehler korrigieren.

Definition 2.1: Systematischer Fehler

Ein Fehler, welcher durch ein unvollständiges Modell entsteht, wird als systematischer Fehler bezeichnet. Er tritt bei jeder Messung auf und kann durch ein verbessertes Modell korrigiert werden.

Bei einer Messung können auch zufällige Fehler auftreten, was als **RAUSCHEN** bezeichnet wird. Wir simulieren uns einen Analog-Digital-Converter (ADC). Hierbei gibt es eine zeitliche Auflösung, eine maximal Messbare Spannung und eine Anzahl von Bits, welche die Auflösung der Messung bestimmt.

Wir definieren folgende Größen:

- Zeitlicher Abstand zwischen Messungen: Δ_t
- Sampling Rate / Abtastrate: $f_{\text{sample}} = 1/\Delta_t$
- Bandbreite der Messung: $f_{\text{bandwidth}} = \frac{1}{2\Delta_t}$
- Totale Messzeit: t_{tot}
- Spektrale Auflösung: $\Delta f = 1/t_{\text{tot}}$
- Signalauflösung: U_{min}

- Signalbereich: U_{max}
- Rauschen der Messapparatur: U_{noise}

Die Meisten dieser Dinge führen zu statistischen Fehlern, welche durch wiederholte Messungen korrigiert werden können.

Wenn wir eine Grösse messen mit erwartetem Wert \tilde{x} und einen Wert x_1 messen, dann bezeichnen wir den **FEHLER** als $e_1 = \tilde{x} - x_1$.

Das Problem ist, dass wir a priori \tilde{x} nicht kennen.

Die Möglichen Fehler können wie bereits erwähnt in systematische und statistische Fehler unterteilt werden. Systematische Fehler können durch eine Kontrollmessung korrigiert werden. Statistische Fehler können nicht verhindert werden. Wir können sie aber durch wiederholte Messungen quantifizieren.

Um einen gegebenen Datensatz zu analysieren, diskretisieren wir den Messbereich x in gleich grosse **BINS**.

Dies kann man dann in einem Histogramm darstellen.

Diese Daten sind zufällig verteilt um einen Mittelwert mit einer Streuung.

Definition 2.2: Mittelwert

Der Mittelwert eines Datensatzes x_1, x_2, \dots, x_N ist definiert als

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i.$$

Um die Streuung um den Mittelwert zu quantifizieren, verwenden wir die Standardabweichung:

Definition 2.3: Varianz und Standardabweichung

Die Varianz eines Datensatzes x_1, x_2, \dots, x_N ist definiert als

$$\text{var}(x) = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2.$$

Die Standardabweichung ist die Quadratwurzel der Varianz:

$$s = \sqrt{\text{var}(x)}.$$

Achtung: Die Standardfunktionen von Python verwenden die Population Varianz, welche durch N dividiert wird, anstatt durch $N-1$.

Wir definieren weiter die **GENAUIGKEIT** einer Messung als die Abweichung des Mittelwerts von dem wahren Wert \tilde{x} .

Ausserdem ist die **PRÄZISION** einer Messung definiert als das Doppelte der Standardabweichung. Sie ist ein Mass für zufällige Fehler.

Es existieren verschiedene Wahrscheinlichkeitsverteilungen, wobei die **NORMALVERTEILUNG** die wichtigste ist. Dies ist aufgrund des zentralen Grenzwertsatzes der Fall, welcher besagt, dass die Summe von vielen unabhängigen Zufallsvariablen, welche eine endliche Varianz haben, normalverteilt ist.

Die Normalverteilung ist eine kontinuierliche Wahrscheinlichkeitsdichtefunktion, welche durch die folgenden Parameter definiert ist:

$$PDF(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Bei einer Normalverteilung ist die Wahrscheinlichkeit, dass eine Messung innerhalb von einem Standardabweichung vom Mittelwert liegt, etwa 68%, innerhalb von zwei Standardabweichungen etwa 95% und innerhalb von drei Standardabweichungen etwa 99.7%.

Mit diesen Werten kann effizient geprüft werden, ob die Daten normalverteilt sind.

Unsere Konvention ist es für die empirischen Grössen, \bar{x} und Δx zu verwenden, während die Werte für die Normalverteilung mit $\mu = E(x) = \langle x \rangle$ und σ bezeichnet werden.

Zusammenfassend können wir einen Messwert angeben durch

$$x_m = \bar{x} \pm \Delta x.$$

Diese Aussage ist nur eine Angabe für die Messung eines Datenpunkts. Es ist keine Aussage über die Genauigkeit von \bar{x} .

Lec 4

Wir wollen nun den Mittelwert abschätzen. Wenn wir eine Messung wiederholt durchführen, so wird die Abschätzung des Mittelwerts immer genauer. Die Standardabweichung bleibt aber erhalten. Der Fehler in der Standardabweichung wird jedoch auch genauer.

Der Mittelwert ist definiert als $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$. Für $N \rightarrow \infty$ konvergiert \bar{x} gegen den wahren Wert μ . Der Wert von \bar{x} ändert sich nicht systematisch mit N , sondern schwankt um μ herum. Wenn wir die Varianz des Mittelwerts berechnen, so erhalten wir

$$\text{var}(\bar{x}) = \frac{1}{N^2} \sum_{i=1}^N \text{var}(x_i) = \frac{\sigma^2}{N}.$$

Wobei wir verwendet haben, dass $\text{var}(\sum x_i) = \sum \text{var}(x_i)$, da die x_i unkorreliert sind. Somit ist die Standardabweichung des Mittelwerts

$$\Delta \bar{x} = \frac{\sigma}{\sqrt{N}}.$$

Die Standardabweichung des Mittelwerts kann somit kleiner als die Standardabweichung der Daten sein.

Zusammenfassend, μ und σ ändern sich nicht systematisch mit N , aber deren Abschätzungen werden genauer, mit N .

2.1 Fehlerfortpflanzung

Häufig können wir nicht direkt die Grösse messen, welche wir interessiert, sondern müssen sie aus anderen Messgrössen berechnen.

Zum Beispiel hängt die Amplitude eines Pendels ab von

$$A = f(m, g, T, \dots).$$

Welchen Einfluss haben die Fehler in m , g und T auf den Fehler in A ?

Um das Problem zu vereinfachen, betrachten wir $f(x)$ mit einem Fehler. Wir können $f(x)$ um den Mittelwert \bar{x} entwickeln:

$$f(x) = f(\bar{x} + \delta x) = f(\bar{x}) + \delta x f'(\bar{x}) = \bar{f} + \delta f(x).$$

Für den Fall der Standardabweichung von f erhalten wir

$$f(x) \approx f(\bar{x}) + \left. \frac{\partial f}{\partial x} \right|_{\bar{x}} (\Delta x) = \bar{f} + \Delta f(x).$$

Für zwei Variablen x und y erhalten wir

$$\begin{aligned} \Delta f^2(x, y) &= \frac{1}{N-1} \sum_{n=1}^N (f_n - \bar{f})^2 \\ &= \frac{1}{N-1} \sum_{n=1}^N \left(\frac{\partial f}{\partial x} (x_n - \bar{x}) + \frac{\partial f}{\partial y} (y_n - \bar{y}) \right)^2 \\ &= \left(\frac{\partial \bar{f}}{\partial x} \right)^2 + \left(\frac{\partial \bar{f}}{\partial y} \right)^2 \\ &\quad + 2 \frac{\partial \bar{f}}{\partial x} \frac{\partial \bar{f}}{\partial y} \frac{1}{N-1} \sum_{n=1}^N (x_n - \bar{x})(y_n - \bar{y}). \end{aligned}$$

Der Erste Term ist das Resultat für unkorrelierte Variablen. Der Zweite Term ist die Korrelation zwischen x und y . Wenn x und y unkorreliert sind, dann verschwindet der zweite Term.

Dies kann geschrieben werden als

$$\sigma_{\bar{f}}^2 = (\partial_x, \partial_y) f \begin{pmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{pmatrix} \begin{pmatrix} \partial_x f \\ \partial_y f \end{pmatrix}.$$

Für unkorrelierte Variablen, also $\sigma_{xy}^2 = 0$, erhalten wir

$$\sigma_{\bar{f}}^2 = \left(\frac{\partial f}{\partial x} \right)^2 \sigma_x^2 + \left(\frac{\partial f}{\partial y} \right)^2 \sigma_y^2.$$

Die Gauss-Methode setzt voraus, dass die Fehler klein sind, so dass die lineare Näherung gültig ist.

Nehmen wir als Beispiel ein Federpendel mit $T = 2.5s \pm 0.08s$ und $m = 0.3kg \pm 0.02kg$.

Die Federkonstante ist gegeben durch

$$k = m \left(\frac{2\pi}{T} \right)^2 \Rightarrow \bar{k} = 1.89 \frac{N}{m}.$$

Wir bilden die partiellen Ableitungen:

$$\frac{\partial k}{\partial m} = \left(\frac{2\pi}{T} \right)^2 = 6.32 \frac{N}{m \cdot kg}.$$

$$\frac{\partial k}{\partial T} = -4\pi m \left(\frac{2\pi}{T^3} \right) = -0.96 \frac{N \cdot s}{m}.$$

$$\Delta k = \sqrt{\left(\frac{\partial k}{\partial m} \right)^2 \Delta m^2 + \left(\frac{\partial k}{\partial T} \right)^2 \Delta T^2} = 0.175 \frac{N}{m}.$$

Es macht keinen Sinn mehr Stellen anzugeben, als die Fehler erlauben.

$$k = (1.9 \pm 0.2) \frac{N}{m}.$$

Tip 2.4:

Man sollte die Fehlerrechnung sowie korrektes Runden an der Prüfung können.

2.2 Korrelationen

Wir erinnern uns, dass die Fehlerfortpflanzung ein Korrelations-Term enthalten war. Damit wollen wir uns nun beschäftigen.

Stellen wir uns ein Pendel vor, mit einer Resonanzfrequenz f und der Temperatur T . Wir können nun die Resonanzfrequenz über einen Tag messen. Wenn wir die Temperatur gleichzeitig messen, so können wir erkennen, dass beide Kurven sehr einen ähnlichen Verlauf haben. Es gibt eine sogenannte **KORRELATION** zwischen f und T .

Wie kann nun Korrelation anschaulich erklärt, und mathematisch formuliert werden?

Eine Korrelation bedeutet, dass eine Änderung in T mit einer Änderung in f einhergeht. Mathematisch beschreiben wir die Variable T als $T = \bar{T} + \delta T$ und die Variable f als $f = \bar{f} + \delta f$. Der Ausdruck

$$(x_n - \bar{x})(y_n - \bar{y}),$$

sollte entweder immer positiv oder immer negativ sein.

Formell definieren wir

Definition 2.5: Kovarianz

Die Kovarianz zwischen zwei Variablen x und y ist definiert als

$$\sigma_{xy}^2 = \frac{1}{N-1} \sum_{n=1}^N (x_n - \bar{x})(y_n - \bar{y}).$$

Visuell möchten wir sagen, dass eine Korrelation stark ist, wenn sie grösser als die Standardabweichungen der Variablen ist. Daher definieren wir den

Definition 2.6: Korrelationskoeffizient

Der Korrelationskoeffizient zwischen zwei Variablen x und y ist definiert als

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \in [-1, 1].$$

Diese Grösse ist nun Normiert, wenn die Korrelation stark ist, dann ist $\rho_{xy} \approx 1$ oder $\rho_{xy} \approx -1$. Wenn die Korrelation schwach ist, dann ist $\rho_{xy} \approx 0$.

Im ersten Fall sprechen wir von maximaler (anti-) Korrelation, im zweiten Fall von keiner Korrelation.

Betrachten wir nun die nichtlineare Funktion $y = x^2$, wobei x eine Zufallsvariable mit symmetrischer Verteilung um 0 ist. In diesem Fall erhalten wir für die Kovarianz $\sigma_{xy}^2 = 0$, obwohl y von x abhängt.

Mit dem Korrelationskoeffizienten können wir somit nur auf lineare Korrelationen testen. Nichtlineare Fälle müssen generell anders behandelt werden.

Korrelation kann auch implizieren dass wir ein sehr grosses Rauschen haben, welches aber gleichzeitig beide Variablen beeinflusst. Wichtig, von Korrelation können wir nicht auf Kausalität schliessen. Zum Beispiel korrelieren die Anzahl der Piraten mit der globalen Durchschnittstemperatur.

Wir betrachten die Funktion $x = x^{(1)} + x^{(2)}$, wobei $x^{(1)}$ und $x^{(2)}$ jeweils N mal gemessen wurden.

Für die Varianz von x erhalten wir

$$\text{var}(x) = \text{var}(x^{(1)}) + \text{var}(x^{(2)}) + 2 \text{cov}(x^{(1)}, x^{(2)}).$$

Generell gilt

$$\begin{aligned} \text{var} \left(\sum_{i=1}^N a^{(k)} x^{(k)} \right) \\ = \sum_{i=1}^N a^{(k)2} \text{var}(x^{(k)}) + 2 \sum_{i < j} a^{(i)} a^{(j)} \text{cov}(x^{(i)}, x^{(j)}), \end{aligned}$$

wobei $a^{(k)}$ Konstanten sind.

Wenn wir in die Kovarianz für $y = x$ einsetzen, so erhalten wir

$$\text{cov}(x, x) = \frac{1}{N-1} \sum_{n=1}^N (x_n - \bar{x})(x_n - \bar{x}) = \text{var}(x).$$

Wenn wir nun aber für $y = x + \Delta$ einsetzen, also zum Beispiel eine zeitliche Verschiebung, so erhalten wir

$$R_{xx}(\Delta) = \frac{1}{N-1-\Delta} \sum_{n=1}^{N-\Delta} (x_n - \bar{x})(x_{n+\Delta} - \bar{x}).$$

Dies wird die **DISKRETE AUTO-KOVARIANZ** genannt. Eine Andere Schreibweise wäre

$$R_{xx}(\tau) = \frac{1}{N-1-\Delta} \sum_{t=t_1}^{t_N-\tau} (x(t) - \bar{x})(x(t+\tau) - \bar{x}).$$

Wir können analog zur Kovarianz auch den Autokorrelationskoeffizienten definieren:

$$\rho_{xx}(\tau) = \frac{R_{xx}(\tau)}{R_{xx}(0)} = \frac{R_{xx}(\tau)}{\text{var}(x)}.$$

Weniger relevant für echte Daten, aber dennoch interessant, ist die kontinuierliche Autokovarianz:

$$R_{xx}(\tau) = \frac{1}{T} \int_0^T (x(t) - \bar{x})(x(t+\tau) - \bar{x}) dt.$$

Wenn ein Signal in der Autovarianz-basis Periodisch ist, so ist es auch in der Zeitbasis periodisch.

Die Auto-Kovarianz ist symmetrisch um $\tau = 0$. Die Samplingzeit setzt ein unteres Limit für die zeitliche Auflösung der Autokovarianz.

3 Die spektrale Leistungsdichte

Die heutige Hauptaussage ist das folgende Theorem:

Theorem 3.1: Fouriers Theorem

Wir können jedes Signal $x(t)$ als Summe von trigonometrischen Funktionen darstellen.

Wir definieren die diskrete inverse Fourier-Transformation als

$$x(t_k) = \sum_{n=-N/2}^{N/2} X(f_n) e^{i2\pi f_n t_k}.$$

Dabei ist $\Delta_f = \frac{1}{t_{\text{tot}}}$ die kleinste Frequenzdifferenz und $f_n = n\Delta_f$ die n -te Frequenz, wenn wir als Funktion der diskreten Zeit $t_k = k\Delta_t$ betrachten.

Die Diskrete Fourier-Transformation ist definiert als

$$X(f_n) = \frac{1}{N} \sum_{k=0}^{N-1} x(t_k) e^{-i2\pi f_n t_k}.$$

Mit ihr können wir die (komplexen) Frequenzkomponenten eines Signals bestimmen.

Der Frequenzbereich einer Fouriertransformation ist nach oben limitiert durch die **NYQUIST-FREQUENZ** $f_{Ny} = \frac{1}{2\Delta_t}$, da dann mindestens 2 Messpunkte pro Periode vorhanden sind.

Die kleinste Frequenz, welche wir messen können ist gegeben durch **GARBORS LIMIT** $\sigma_t \sigma_f \geq \frac{1}{2}$, Die maximale Messzeit t_{tot} limitiert σ_t . Die maximale Frequenzauflösung ist somit $\sigma_f \approx \Delta_f = \frac{1}{t_{\text{tot}}} = \frac{2f_{\text{max}}}{N}$.

Die Fourier-Transformation kann ausgewertet werden bei $f_n = n \cdot \Delta_f$. Hierbei haben wir dann N punkte von $-f_{Ny}$ bis f_{Ny} .

Zusammenfassend ist der maximale Informationsgehalt eines Signals in der Frequenz und der Zeitbasis gleich.

In vielen Anwendungen wird das normierte Quadrat der Fourier-Transformation, also $S_{xx}(f_n) = \frac{\Delta_t}{N} |X(f_n)|^2$, verwendet. Dies wird die **POWER SPECTRAL DENSITY** genannt. Sie gibt die Leistung eines Signals in der Frequenzbasis an. Dies ist nützlich, da uns Sinus und Kosinus in der Praxis selten interessieren. Weiter wird die Normierung die Frequenzauflösung unabhängig machen. Dies ist ähnlich zu einer Dichte einer Masse.

Die Normierung mit Δ_t mach aus einem Power Spectrum eine PSD mit der Einheit $\frac{x^2}{Hz}$.

Ein interessantes Resultat ist, dass

Theorem 3.2: Parseval's Theorem

Die Varianz eines Signals in der Zeitbasis ist gleich der Varianz in der Frequenzbasis, also

$$\sigma_x^2 = \sum_{n=-N/2}^{N/2} S_{xx}(f_n) \Delta_f.$$

Wenn wir zwei unkorrelierte Variablen x und y haben, so erhalten wir für die PSD von $z = x + y$:

$$\sum S_{zz}(f) = \sum S_{xx}(f) + \sum S_{yy}(f).$$

Bisher haben wir die double sided PSD betrachtet, welche von $-f_{Ny}$ bis f_{Ny} definiert ist. In der Praxis verwenden wir aber die single sided PSD, welche von 0 bis f_{Ny} definiert ist. Hierbei ist

$$\sigma_x^2 = \Delta_f \sum_{f_n=1}^{f_{max}} S_{xx}(f_n).$$

Hierbei ist dann jedoch für Konsistenz ein Faktor von 2 notwendig.

$$S_{xx}(f_n) = 2 \frac{\Delta_t}{N} \left| \sum_{k=1}^N x(t_k) e^{-i2\pi f_n t_k} \right|^2.$$

Wir wollen nun unsere Theorie ein wenig anwenden. Angenommen wir haben ein Rauschen mit durchschnittlicher PSD $\bar{S}_{xx} = 5 \cdot 10^{-6} \frac{V^2}{Hz}$, welche wir über $500Hz$ messen. Dann haben wir eine Standardabweichung des Rauschens von

$$\sigma_x = \sqrt{\Delta_f \sum_{f_n=1}^{f_{Ny}} S_{xx}(f_n)} = \sqrt{500 \cdot 5 \cdot 10^{-6}} = 0.05V.$$

3.1 Filtern

Wir wollen weisses Rauschen aus einer Messung entfernen. Wir können dies tun, indem wir die Fourier-Transformation der Messung berechnen, die Frequenzkomponenten mit $f > f_{\text{cutoff}}$ entfernen und dann die inverse Fourier-Transformation berechnen.

Eine einfacher Filter wäre auch zu Mitteln. Hierbei nehmen wir jeweils N Messungen vor und nach der eigentlichen Messung und bilden den Mittelwert. Hierbei sind dann aber die Messungen nicht mehr unkorreliert! Wir müssen aufpassen bei der Mittelungszeit. Sie sollte möglichst gross sein, aber auch deutlich kleiner als die Zeitkonstante des Signals.

Ein Tiefpassfilter entfernt schnelle Änderungen im Signal, beispielsweise durch Mitteln. Dabei werden die Datenpunkte über die Filterzeit korreliert, und die Informationsmenge reduziert. Man kann die Information durch eine reduzierte Anzahl unkorrelierter Messpunkt darstellen. An vielen Geräten lässt sich die Messrate reduzieren, um dafür eine bessere Präzision zu erhalten.

Um weisses Rauschen zu quantifizieren, reicht es den Mittelwert der PSD zu bestimmen. Wenn das Spektrum nicht flach ist, so kann jeweils über einen Frequenzbereich von N Punkten gemittelt werden, analog zum Tief-Pass Filter.

Wenn gewisse Details nicht verwaschen werden sollen, so kann man auch die Messung mehrmals durchführen und die Ergebnisse mitteln.

4 Bayesische Statistik

Wir wollen uns Gedanken über das Ergebnis eines Experiments machen. Wir wollen die Wahrscheinlichkeit für ein bestimmtes Ergebnis E angeben. Diese wird bestimmt durch $P(A) = \frac{k}{n}$, wobei k die Anzahl der Fälle ist, in welchen A eintritt, und n die Gesamtzahl der Fälle.

Example 4.1: Würfel

Die Wahrscheinlichkeit, dass eine Zahl grösser als 4 gewürfelt wird, ist $P(x > 4) = \frac{2}{6} = \frac{1}{3}$.

Dieser Sachverhalt kann elegant in einem Venn-Diagramm dargestellt werden. Es gilt im Allgemeinen, dass $A \subset S$ und somit $0 \leq P(A) \leq 1$. Ausserdem gilt

$$1 = P(S) = P(A \cup \bar{A}) = P(A) + P(\bar{A}).$$

Die Wahrscheinlichkeit von A oder B ist gegeben durch

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

Die Wahrscheinlichkeit dass A eintritt, wenn B bereits eingetreten ist, ist gegeben durch

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Umgeformt erhalten wir

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A).$$

Achtung: Es muss kein kausaler Zusammenhang zwischen A und B bestehen, damit $P(A|B)$ und $P(B|A)$ definiert sind. wenn A und B unabhängig sind, so gilt $P(A|B) = P(A)$ und $P(B|A) = P(B)$.

Theorem 4.2: Satz von Bayes

Es gilt

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}.$$

Wir Schreiben nun A_i für die möglichen Ergebnisse eines Experiments, also A_1, A_2, \dots, A_n . Es gilt dann

$$P(A_i) \geq 0 \quad \text{und} \quad \sum_{i=1}^n P(A_i) = 1.$$

Für einen diskreten Satz von Ereignissen A_i nennen wir das eine Wahrscheinlichkeitsmassefunktion, **PROBABILITY MASS FUNCTION** (PMF).

Falls das Ergebnis des Experiments kontinuierlich ist, so können wir die Wahrscheinlichkeitsdichtefunktion, **PROBABILITY DENSITY FUNCTION** (PDF) verwenden. Es gilt

$$f(x) \geq 0 \quad \text{und} \quad \int_{-\infty}^{\infty} f(x)dx = 1.$$

Die Wahrscheinlichkeit, dass das Ergebnis in einem Intervall $[a, b]$ liegt, ist dann gegeben durch

$$P(a \leq x \leq b) = \int_a^b f(x)dx.$$

Achtung: $f(x)$ ist eine Dichte, keine Wahrscheinlichkeit.

Example 4.3: Dirac Delta

Die Dirac Delta Funktion $f(x) = \delta(x)$ erfüllt alle Eigenschaften einer PDF.

Im folgenden besprechen wir Konzepte anhand einer kontinuierlichen Wahrscheinlichkeitsfunktion, sie können aber auch für diskrete Fälle angewendet werden.

4.1 Charakteristiken von Wahrscheinlichkeitsverteilungen

Folgende Grössen sind für alle PDFs definiert:

Definition 4.4: Mode

Der Wert von x bei dem die PDF ihr Maximum erreicht, wird als **MODE** bezeichnet.

$$x_{\text{mode}} = \left. \frac{df(x)}{dx} \right|_{x=x_{\text{mode}}} = 0.$$

Definition 4.5: Median

Die Mitte der PDF

$$x_{\text{median}} = \int_{-\infty}^{x_{\text{median}}} f(x)dx = \frac{1}{2}.$$

Definition 4.6: FWHM

Die Halbwertsbreite **FULL WIDTH AT HALF MAXIMUM** (FWHM) ist definiert als die Breite der PDF an der Stelle, an der sie halb so gross ist wie ihr Maximum.

$$f(a) = f(b) = \frac{1}{2}f(x_{\text{mode}}) \Rightarrow \text{FWHM} = b - a.$$

Für eine Normalverteilung gilt, dass die FWHM

$$2\sqrt{2 \ln 2} \sigma \approx 2.355\sigma$$

ist.

4.2 Momente

Für viele PDFs sind ihre Momente wie folgt definiert

Definition 4.7: Momente

Das m -te Moment einer PDF $f(x)$ ist definiert als

$$M_m = \int_{-\infty}^{\infty} x^m f(x)dx.$$

Das erste Moment M_1 ist der Mittelwert. Wichtig ist, dass jedoch nicht jede PDF einen Mittelwert hat.

Wenn wir weitere Momente berechnen, so wollen wir diese Häufig um den Mittelwert zentrieren, um die **ZENTRAL-MOMENTE** zu erhalten:

Definition 4.8: Zentralmomente

Das m -te zentrale Moment einer PDF $f(x)$ ist definiert als

$$\tilde{M}_m = \langle (x - M_1)^m \rangle.$$

Hierbei ist $\langle \cdot \rangle$ die Erwartungswert-Notation, welche das Integral über die PDF impliziert.

Somit ist das zweite zentrale Moment die Varianz, das dritte zentrale Moment quantifiziert die Schiefe und das vierte die Wölbung der PDF.

Das folgende ist nicht klausurrelevant, aber dennoch interessant:

Es kann hilfreich sein, die Momenterzeugende Funktion zu definieren:

$$M(\zeta) = \int_{-\infty}^{\infty} \exp(\zeta x) f(x) dx.$$

Wenn wir davon die Taylor-Entwicklung um $\zeta = 0$ bilden, so erhalten wir

$$M(\zeta) = M_0 + \zeta M_1 + \frac{\zeta^2}{2} M_2 + \dots$$

Da $M_0 = 1$ sein muss, können wir das m -te Moment durch die m -te Ableitung der Momenterzeugenden Funktion an $\zeta = 0$ berechnen:

Theorem 4.9: Momenterzeugende Funktion

Es gilt

$$M_m = \left. \frac{\partial^m M(\zeta)}{\partial \zeta^m} \right|_{\zeta=0}.$$

Erneut ist es wichtig zu beachten, dass nicht jede PDF eine Momenterzeugende Funktion hat.

Weiter ist es Hilfreich, die charakteristische Funktion zu definieren:

$$\phi(\nu) = \int_{-\infty}^{\infty} \exp(i\nu x) f(x) dx.$$

Dies ist die inverse Fourier-Transformation der PDF. Somit können wir die PDF durch die Fourier-Transformation der charakteristischen Funktion berechnen:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(-i\nu x) \phi(\nu) d\nu.$$

Entscheidend ist, dass die Charakteristische Funktion für jede Wahrscheinlichkeitsverteilung existiert, auch wenn die Momenterzeugende Funktion nicht existiert.

Theorem 4.10:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(-i\nu x) (1 + i\nu M_1 + \dots) d\nu.$$

4.3 Bayes'sche Datenanalyse

Die Bayessche Datenanalyse beruht auf bedingten Wahrscheinlichkeiten. Um dies auf ein Experiment anzuwenden definieren wir

$A \rightarrow D =$ Gemessene Daten

$B \rightarrow B_1, B_2, \dots =$ Ein endlicher Satz von Ergebnissen

$I =$ Vorwissen.

Somit stellt sich der Satz von Bayes für die Datenanalyse wie folgt dar:

$$P(B_i|D, I) = \frac{P(D|B_i, I)P(B_i|I)}{P(D|I)}.$$

Wenn D impliziert, dass eines der B_i eingetreten ist, so können wir den Nenner umschreiben als

$$P(D|I) = \sum_{i=1}^n P(D|B_i, I)P(B_i|I).$$

Hierbei ist $P(B_i)$ die Wahrscheinlichkeit, dass B_i eintritt basierend auf a-priori Wissen. Es ist die Sogenannte **PRIOR**. Das Ergebnis ist die A-posteriori Wahrscheinlichkeitsverteilung.

Example 4.11:

Nehmen wir an, es gibt eine Infektionskrankheit wobei 100 von 100000 Menschen infiziert sind. Es gibt einen Test, welcher in 99% der Fälle korrekt ist, also

$$P(\text{positiv}|\text{infected}) = 0.99$$

$$P(\text{negativ}|\text{infected}) = 0.01$$

$$P(\text{negativ}|\text{not infected}) = 0.99$$

$$P(\text{positiv}|\text{not infected}) = 0.01.$$

Die Wahrscheinlichkeit vor dem Test dass eine Person infiziert ist, ist $P(\text{infected}) = 0.001$. Nun nehmen wir an, dass eine Person positiv getestet wird. Wie hoch ist die Wahrscheinlichkeit, dass sie tatsächlich infiziert ist?

Nach dem Satz von Bayes erhalten wir

$$P(I|+) = \frac{P(+|I)P(I)}{P(+|I)P(I) + P(+|\bar{I})P(\bar{I})} = 0.1.$$

Wenn wir einen zweiten unabhängigen Test durchführen, so ändern sich die a-priori Wahrscheinlichkeiten für den zweiten Test. Nun ist $P(I) = 0.1$ und $P(\bar{I}) = 0.9$. Somit erhalten wir

$$P(I|+) = \frac{P(+|I)P(I)}{P(+|I)P(I) + P(+|\bar{I})P(\bar{I})} = 0.92.$$

Somit können wir den Wert eines Parameters a schätzen mit dem Ausdruck $f_1(a|D) = \frac{L(D|a)f_0(a)}{\int L(D|a)f_0(a)da}$, wobei $L(D|a)$ die Likelihood-Funktion ist, welche die Wahrscheinlichkeit der Daten D gegeben den Parameter a angibt, und $f_0(a)$ die Prior-Verteilung für den Parameter a ist.

4.4 Wahrscheinlichkeitsverteilungen

Wir haben bereits die Normalverteilung kennengelernt. Sie hat folgende PDF

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Die Normalverteilung ist die wichtigste Verteilung, da sie aufgrund des zentralen Grenzwertsatzes in vielen Fällen auftritt.

Wir betrachten nun die **BINOMIALVERTEILUNG**. Dazu brauchen wir eine Variable, welche genau zwei Werte u und d annehmen kann. Da jedes Experiment unabhängig ist, gilt

$$P(u) = p \quad \text{und} \quad P(d) = 1 - p = q.$$

Wenn wir nun ein System aus N spins messen, so haben wir 2^N mögliche Ergebnisse. Die Wahrscheinlichkeit für ein bestimmtes Ergebnis ψ_i mit k Spins in u und $N - k$ Spins in d ist gegeben durch

$$P(\psi_i) = p^k q^{N-k}.$$

In der Praxis können wir schlecht unterscheiden, was genau positiv ist, sondern nur die Anzahl der positiven Ergebnisse k . Es gibt $\binom{N}{k}$ Möglichkeiten, k positive Ergebnisse zu erhalten. Somit ist die Wahrscheinlichkeit für k positive Ergebnisse gegeben durch

$$P(k) = \binom{N}{k} p^k q^{N-k}.$$

Als Wahrscheinlichkeitsverteilung ist diese Verteilung natürlich normiert.

$$\sum_{k=0}^N P(k) = \sum_{k=0}^N \binom{N}{k} p^k q^{N-k} = (p+q)^N = 1.$$

Dies ist zugleich das 0-te Moment. Der Erwartungswert der Binomialverteilung ist das erste Moment,

$$\langle y \rangle = M_1.$$

Hierfür können wir die Momenterzeugende Funktion verwenden. Es gilt

$$M_m = \left. \frac{\partial^m M(\zeta)}{\partial \zeta^m} \right|_{\zeta=0}.$$

Dazu leiten wir die Funktion des M -ten Moments nach p ab.

$$\frac{\partial M_m}{\partial p} = \frac{1}{p} M_{m+1} - \frac{N}{q} M_m + \frac{1}{q} M_{m+1}.$$

Somit gilt für das $m+1$ -te Moment

$$M_{m+1} = NpM_m + pq \frac{\partial M_m}{\partial p}.$$

Da das nullte Moment $M_0 = 1$ ist, erhalten wir für das erste Moment

$$M_1 = NpM_0 + pq \frac{\partial M_0}{\partial p} = Np.$$

Das zweite Moment erhalten wir durch Ableiten des ersten Moments:

$$M_2 = NpM_1 + pq \frac{\partial M_1}{\partial p} = Np(Np + q).$$

Die Varianz erhalten wir damit als

$$\sigma^2 = M_2 - M_1^2 = Npq.$$

Also ist $\sigma \propto \sqrt{N}$.

Eine weitere häufige Verteilung ist die **POISSONVERTEILUNG**. Sie ist ein Grenzfall der Binomialverteilung für seltene Ereignisse. Das klassische Beispiel sind die Photonen, in einem Laserstrahl. Es stellt sich heraus, dass die Anzahl der Photonen, welche in einem bestimmten Zeitintervall ankommen, einer Poissonverteilung folgt.

Um das zu messen, senden wir einen Laserstrahl auf einen Einzelphotonendetektor, welcher die Anzahl Photonen in einem Zeitintervall Δ_t zählt. Dies wiederholen wir nun N mal.

Wenn nun N sehr gross, $N \gg k$ und $p \rightarrow 0$ ist, so können wir die Binomialverteilung durch die Poissonverteilung approximieren. Es gilt

$$P(k) = \lim_{N \gg k} \binom{N}{k} p^k q^{N-k} = \frac{N^k}{k!} p^k (1-p)^{N-k}.$$

Wir setze nun $\mu = Np$ und erhalten

$$P(k) = \frac{\mu^k}{k!} ((1-p)^{1/p})^\mu \approx \frac{\mu^k}{k!} e^{-\mu}.$$

Definition 4.12: Poissonverteilung

Die Poissonverteilung mit Parameter μ ist definiert als

$$P(k) = \frac{\mu^k}{k!} e^{-\mu}.$$

Für eine Poissonverteilung gilt, dass der Erwartungswert M_1 gleich dem Parameter μ ist, und die Varianz ebenfalls gleich μ ist.

Damit die Einheitslosigkeit funktioniert, muss μ eine Zahl sein, welche die Anzahl der Ereignisse in einem bestimmten Zeitintervall angibt.

Theorem 4.13: Zentraler Grenzwertsatz

Wenn N gross genug ist, so folgt die Summe von N unabhängigen und identisch verteilten Zufallsvariablen einer Normalverteilung.

Nehmen wir an, wir haben zwei unkorrelierte Variablen ε_1 und ε_2 , mit Erwartungswerten $\mu_1 = \mu_2 = 0$ und Varianzen σ_1^2 und σ_2^2 normalverteilt. Die Wahrscheinlichkeit, Werte in den Intervallen $[\varepsilon_1, \varepsilon_1 + d\varepsilon_1]$ und $[\varepsilon_2, \varepsilon_2 + d\varepsilon_2]$ zu erhalten, ist gegeben durch

$$f(\varepsilon_1, \varepsilon_2) = 1/(2\pi\sigma_1\sigma_2) \exp\left(-\frac{\varepsilon_1^2}{2\sigma_1^2} - \frac{\varepsilon_2^2}{2\sigma_2^2}\right) d\varepsilon_1 d\varepsilon_2.$$

Wir definieren die Variablen $x_i = \frac{\varepsilon_i}{\sigma_i}$, welche beide einer Standardnormalverteilung folgen. Es gilt

$$f(x_1, x_2) = \frac{1}{2\pi} \exp\left(-\frac{x_1^2}{2} - \frac{x_2^2}{2}\right) dx_1 dx_2.$$

Weiterhin, definieren wir die Summe der Quadrate $S = x_1^2 + x_2^2$. Die Wahrscheinlichkeit, dass S in einem Intervall $[S, S + dS]$ liegt, ist gegeben durch die Fläche eines Rings um den Ursprung. Wir machen nun Variablentransformation

$$\begin{aligned} x_1 &= \chi \cos \theta \\ x_2 &= \chi \sin \theta \\ \chi^2 &= x_1^2 + x_2^2 = S. \end{aligned}$$

Damit wird

$$f(\chi) = \exp\left(-\frac{1}{2}\chi^2\right)\chi d\chi.$$

Umschreiben nach χ^2 ergibt

$$f(\chi^2) = \frac{1}{2} \exp\left(-\frac{1}{2}\chi^2\right).$$

Es gilt:

$$\begin{aligned} \chi_{\text{mode}}^2 &= n - 2 \\ \langle \chi^2 \rangle &= n \\ \text{var}(\chi^2) &= 2n. \end{aligned}$$

Lec 10

4.5 Likelihood-Funktion

Bisher haben wir die Messung als Zufallsprozess betrachtet, welche durch eine PDF beschrieben wird. Die Likelihood-Funktion ist die PDF, aber mit vertauschten Argumenten. Sie ist eine Funktion der Parameter, nicht der Daten. Zum Beispiel messen wir zwei unabhängige Datenpunkte x_1 und x_2 aus einer Gauss-Verteilung.

Da die Messungen unabhängig sind, ist die gemeinsame PDF das Produkt der Einzel-PDFs:

$$f(x_1, x_2 | \mu, \sigma) = f(x_1 | \mu, \sigma) f(x_2 | \mu, \sigma).$$

Für n Datenpunkte x_1, x_2, \dots, x_n ist die Likelihood-Funktion gegeben durch

$$L(\mathbf{x}, a_0, a_1) = \prod_{i=1}^n f(x_i | a_0, a_1).$$

Dies ist KEINE PDF, da sie nicht normiert ist.

Um die Parameter zu schätzen, können wir die Likelihood-Funktion maximieren. Dabei ist wichtig, dass der Wert der Likelihood-Funktion selbst nicht aussagekräftig ist, jedoch Verhältnisse von Likelihood-Funktionen sehr aussagekräftig sein können. Weil wir uns nicht für das Maximum der Likelihood-Funktion interessieren, sondern für die Parameter, welche das Maximum liefern, können wir die Log-Likelihood-Funktion verwenden, da sie das gleiche Maximum hat wie die Likelihood-Funktion selbst.

Wenn wir nun die Log-Likelihood-Funktion für die Normalverteilung berechnen, so erhalten wir

$$\ln L(\mathbf{x}, \mu, \sigma) = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2.$$

Die Parameter μ und σ , die die Log-Likelihood-Funktion maximieren, lassen sich durch Ableiten der Log Likelihood-Funktion nach μ und σ bestimmen. Schliesslich finden wir

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad \text{und} \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2.$$

Der Wert der Varianz hat hierbei einen Bias im Vergleich zur empirischen Varianz.

Wenn wir nun mehrere Messungen machen, je mit bekannter Standardabweichung, wollen wir den Mittelwert bestimmen. Die Likelihood-Funktion ist dann gegeben durch

$$L(\mathbf{x}, \sigma, a) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - a)^2}{2\sigma_i^2}\right).$$

Die Log-Likelihood-Funktion ist dann

$$\ln L(\mathbf{x}, \sigma, a) = -\frac{1}{2} \sum_{i=1}^n \left(\ln(2\pi\sigma_i^2) + \frac{(x_i - a)^2}{\sigma_i^2} \right).$$

Nach ableiten nach a und Nullsetzen erhalten wir

$$\hat{a} = \frac{\sum_{i=1}^n \frac{x_i}{\sigma_i^2}}{\sum_{i=1}^n \frac{1}{\sigma_i^2}}.$$

In der Regel ist Maximum Likelihood nicht ganz einfach um die Parameter zu schätzen. Wir definieren die Summe der Abstandsquadrate als

$$S = \sum_{i=1}^n \frac{(x_i - a)^2}{\sigma_i^2}.$$

Diese Funktion wollen wir nun minimieren. Es gilt

$$\frac{\partial}{\partial a} \sum_{i=1}^n \frac{(x_i - a)^2}{\sigma_i^2} = 0.$$

Wenn die σ_i bekannt sind, dann ist $S = \chi^2$. Bleiben wir zunächst aber bei S .

Wir haben dabei Daten y_1, \dots, y_n als Funktion der Kontrollparameter x_1, \dots, x_n gemessen. Die x_i haben keine Unsicherheit, die y_i haben eine Unsicherheit von σ_i . Das bedeutet dass die Wahrscheinlichkeit, dass y gemessen wird ist

$$f(y) \sim \exp\left(-\frac{(y - \mu)^2}{2\sigma^2}\right).$$

Wir nehmen zunächst an, dass $\mu = a_0 + a_1 x$.

Die Likelihood-Funktion ist dann gegeben durch

$$L(x_n, y_n, \sigma_n, a_0, a_1) \sim \exp\left(-\frac{(y_n - a_0 - a_1 x_n)^2}{2\sigma_n^2}\right).$$

Mit der Loglikelihood-Funktion wollen wir somit

$$S = \sum_{i=1}^N w_i (y_i - a_0 - a_1 x_i)^2,$$

minimieren, wobei $w_i = \frac{1}{\sigma_i^2}$ die Gewichte sind. Ausserdem sind die **RESIDUEN** $r_i = y_i - a_0 - a_1 x_i$ die Abstände der

Datenpunkte von der Anpassungsfunktion. Im Idealfall sind die Residuen normalverteilt mit Mittelwert 0.

Ableiten nach a_0 und a_1 und Nullsetzen ergibt das Gleichungssystem

$$\frac{\partial S}{\partial a_0} = -2 \sum_{i=1}^N w_i (y_i - a_0 - a_1 x_i) = 0$$

$$\frac{\partial S}{\partial a_1} = -2 \sum_{i=1}^N w_i x_i (y_i - a_0 - a_1 x_i) = 0.$$

Das Ganze können wir in Matrixform schreiben als

$$\begin{pmatrix} \sum w_n & \sum w_n x_n \\ \sum w_n x_n & \sum w_n x_n^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum w_n y_n \\ \sum w_n x_n y_n \end{pmatrix}.$$

Oder auch

$$N\hat{a} = Y.$$

Diese Darstellung nennen wir die Normalform. Sie hat die Lösung

$$\hat{a} = N^{-1}Y.$$

Dies kann nun für beliebige Polynome erweitert werden, indem wir die Matrix N entsprechend anpassen.

$$\begin{pmatrix} \sum w_n & \dots & \sum w_n x_n^m \\ \vdots & \ddots & \vdots \\ \sum w_n x_n^m & \dots & \sum w_n x_n^{2m} \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} \sum w_n y_n \\ \vdots \\ \sum w_n x_n^m y_n \end{pmatrix}.$$

Wir wollen nun noch die Unsicherheiten unserer Parameter bestimmen. Dazu erinnern wir uns, dass die Residuen ϵ_n normalverteilt sind und $\langle \epsilon_n, \epsilon_m \rangle = \sigma_n^2 \delta_{nm}$ gilt. Es gilt

$$\mathbf{y} = X\mathbf{a} + \boldsymbol{\epsilon}.$$

Aus den neuen Matrizen können wir auch die Normalmatrix und den gewichteten Ergebnisvektor finden

$$N = X^T W X$$

$$Y = X^T W \mathbf{y}.$$

Somit folgt für die Summe der Abstandskquadrate

$$S = (\mathbf{y} - X\hat{\mathbf{a}})^T W (\mathbf{y} - X\hat{\mathbf{a}}).$$

Auflösen nach $\hat{\mathbf{a}}$ ergibt

$$\hat{\mathbf{a}} = (X^T W X)^{-1} X^T W \mathbf{y}.$$

Die Unsicherheiten der Parameter $\hat{\mathbf{a}}$ sind gegeben durch deren Varianz

$$\sigma_n^2 = \langle (\hat{a}_n - a_n)^2 \rangle.$$

Die Kovarianzmatrix ist

$$C = \langle (\hat{\mathbf{a}} - \mathbf{a})(\hat{\mathbf{a}} - \mathbf{a})^T \rangle = (X^T W X)^{-1} = N^{-1}.$$

ACHTUNG! Standardbibliotheken korrigieren die Varianz der Datenpunkte, indem sie

$$C = \frac{S}{N - m - 1} N^{-1}$$

verwenden. Hierbei ist $S = \sum_{n=1}^N w_n (y_n - a_0 - a_1 x_n - \dots)^2$

Ausserdem entspricht S_{min} der Summe der Abstandskquadrate für die optimalen Parameter. Es ist somit eine χ^2 -Verteilung mit $N - m - 1$ Freiheitsgraden, da wir $m + 1$ Parameter geschätzt haben. Somit ist S_{min} mit 67% Wahrscheinlichkeit in einem Intervall

$$\langle \chi_k^2 \rangle \pm \sqrt{\text{var}(\chi_k^2)} = (N - m - 1) \pm \sqrt{2(N - m - 1)}.$$

Wir wollen nun die kleinsten Quadrate nichtlinear schätzen zum Beispiel für exponentielle Abfälle. Also wollen wir

$$S = \sum_{n=1}^N w_n (y_n - f(x_n, \mathbf{a}))^2$$

minimieren. Wir können dies tun, indem wir S als Funktion von allen \mathbf{a} berechnen.

Leider ist es nicht möglich, die Parameter durch Ableiten von S zu bestimmen, da f nicht linear in den Parametern ist. Stattdessen können wir iterativ vorgehen.

1. Gradientenverfahren. Wir starten mit einer initialen Schätzung der Parameter \mathbf{a}_g und berechnen die Taylor-Entwicklung von f um \mathbf{a}_g :

$$S(\mathbf{a}) \approx S(\mathbf{a}_g) + \sum_{m=0}^M \frac{\partial S}{\partial a_m} \Big|_{\mathbf{a}=\mathbf{a}_g} (a_m - a_{g,m}).$$

In der Vektorschreibweise wird dies zu

$$S \approx S(\mathbf{a}_g) + \Delta \mathbf{a}^T \nabla S.$$

Damit ist die Korrektur von \mathbf{a}_g richtung Minimum gegeben durch

$$\Delta a_m = -\alpha \frac{\partial S}{\partial a_m} \Big|_{\mathbf{a}=\mathbf{a}_g}.$$

Hierbei ist α ein Parameter, welche die Stärke der Korrektur bestimmt. Wir wiederholen diesen Schritt, bis die Korrektur klein genug ist (idealerweise $\Delta a \approx 0$).

Um zu sehen ob wir das globale Minimum gefunden haben, können wir zuerst die Daten plotten und anschliessend die Residuen plotten. Wenn die Residuen normalverteilt mit Mittelwert 0 sind, so haben wir wahrscheinlich das globale Minimum gefunden.

Ausserdem verwenden wir idealerweise ein alpha, welches kleiner wird je näher wir zum Minimum kommen.

2. Newtonverfahren. Anstelle der einfachen Linearisierung, nehmen wir nun noch den zweiten Term der Taylor-Entwicklung mit auf:

$$S \approx S(\mathbf{a}_g) + \Delta \mathbf{a}^T \nabla S + \frac{1}{2} \Delta \mathbf{a}^T H \Delta \mathbf{a}.$$

S wird also durch ein Paraboloid approximiert. Der nächste Schritt ist gegeben durch das Minimum dieses Paraboloids, also muss für alle Δa_i gelten

$$\frac{\partial S}{\partial \Delta a_i} = 0.$$

Wenn wir dies ausrechnen, so erhalten wir

$$\Delta \hat{\mathbf{a}} = -H^{-1} \nabla S.$$

Die neuen Parameter sind dann gegeben durch

$$\hat{a} = a_g + \Delta\hat{a}.$$

Auch hier können wir einen Skalierungsfaktor $\alpha < 1$ hinzufügen.

$$\hat{a} = a_g + \alpha\Delta\hat{a}.$$

Zusammenfassend:

Gradientenverfahren: $\Delta\hat{a} = -\alpha\nabla S$

Newtonverfahren: $\Delta\hat{a} = -\alpha H^{-1}\nabla S$

Eine einfache Verbesserung wäre es α dynamisch anzupassen.

$$\Delta\hat{a} = -\alpha \left(\frac{\partial^2 S}{\partial a_m^2} \right)^{-1} \frac{\partial S}{\partial a_m}.$$

Eine Standardverbesserung ist Marquardts-Methode. Dabei verwenden wir wenn wir weit vom Minimum entfernt sind, das Gradientenverfahren, und wenn wir nahe am Minimum sind, das Newtonverfahren. Das bedeutet, dass wir α abhängig von der Entfernung zum Minimum machen:

Dazu verwenden wir eine andere Hesse-Matrix

$$H' = \begin{cases} H_{ij}(1 + \alpha) & \text{für } i = j \\ H_{ij} & \text{für } i \neq j \end{cases}.$$

Nun möchten wir noch die Unsicherheiten der Parameter bestimmen. Wir können durch Linearisierung S abschätzen als

$$S \approx S_{\min} + \Delta a^T N \Delta a.$$

Nach Rechnung findet man schliesslich für die Kovarianzmatrix

$$C = 2H^{-1}.$$

5 Machine Learning

Machine Learning ist sehr ähnlich zu der Grundidee von Datenanalyse, jedoch ohne dem Ziel ein Modell zu bestätigen. Gegeben ist ein Mathematisches Modell und die Daten eines Experiments. Die Aufgabe ist es nun das Modell mit den Daten zu verifizieren. Bei Machine Learning ist der physikalische Zusammenhang nicht so wichtig, jedoch das Vorhersagen von Daten. Wir wollen also ein Modell $f(x, w)$ finden welches die Daten y vorhersagt. Dazu benötigen wir ein **VERGLEICHSMASS** $L(y, \hat{y})$, auch genannt **VERLUSTFUNKTION**. Mit dieser Verlustfunktion sollen dann die Modellparameter w so angepasst werden, dass die Vorhersage \hat{y} möglichst gut mit den Daten y übereinstimmt. Dies nennt sich **ÜBERWACHTES LERNEN**.

Grundsätzlich unterscheiden wir zwischen **REGRESSION**, bei welcher die Daten kontinuierlich sind, und **KLASSIFIKATION**, bei welcher die Daten diskret sind. Wir wollen uns zunächst mit Regression beschäftigen.

Das einfachste Problem ist die lineare Regression. Zu finden ist ein Modell

$$\hat{y} = f(x, \mathbf{w}) = w_1 x + w_0.$$

Die Verlustfunktion ist dann gegeben durch

$$L(y, \hat{y}) = (y - f(x, \mathbf{w}))^2.$$

Dieses Optimierungsproblem können wir wie gesehen durch einen Gradient Descent oder durch die Normalform lösen.

Die Algorithmen, welche wir hier sehen sind Gradientenbasiert. Ein Trick um das Lernen zu verbessern ist es die Daten zu standardisieren. Dies funktioniert indem wir die Daten zentrieren und normieren. Es gilt

$$z = \frac{x - \mu}{\sigma}.$$

Somit fallen die Skala und Einheiten weg. Die Rücktransformation ist dann gegeben durch

$$x = z\sigma + \mu.$$

Um so ein Regressionsmodell zu trainieren, verwenden wir die Python-Bibliothek `scikit-learn`. Um eine lineare Regression durchzuführen können wir folgendes tun

```
# Daten Standardisieren
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

# Modell auswählen
model = LinearRegression()

# Trainieren
model.fit(X_scaled, y)

# Vorhersagen
y_pred = model.predict(X_scaled)
```

Wir können dies nun auch in höheren Dimensionen durchführen. Das heisst, jedes x wird ein Vektor \mathbf{x} . Somit wird das Modell zu

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = w_1 x_1 + w_2 x_2 + \dots + w_d x_d + w_0.$$

Wir definieren eine Analoge Verlustfunktion über die Fehlerquadrate und optimieren diese mit Gradientenabstieg oder der Normalform.

Letztendlich können wir auch polynomielle Regression durchführen durch das Modell

$$f(x, \mathbf{w}) = \sum_{i=1}^m x^i w_i + w_0.$$

Prinzipiell transformieren wir hierbei die Daten in einen höherdimensionalen Raum, um dann eine lineare Regression durchzuführen. Wir können analog auch eine Funktion von mehreren Variablen verwenden, also

$$f(\mathbf{x}, \mathbf{w}) = \sum_{k_1 + \dots + k_d \leq m} w_{k_1, \dots, k_d} x_1^{k_1} \dots x_d^{k_d} + w_0.$$

In Scikit-learn können wir dies wie folgt durchführen

```
from sklearn.preprocessing import
↳ PolynomialFeatures

# Transformieren der Daten in einen
↳ höherdimensionalen Raum
poly = PolynomialFeatures(degree=m)
X_poly = poly.fit_transform(X_scaled)

# Modell auswählen
model = LinearRegression()
# Trainieren
model.fit(X_poly, y)
# Vorhersagen
y_poly_pred = model.predict(X_poly)
```

Wenn wir die Anzahl Parameter erhöhen, laufen wir die Gefahr des **OVERFITTING**. Das heisst, dass wir die Trainingsdaten auswendig lernen, aber nicht mehr in der Lage sind, neue Daten vorherzusagen. Man kann dies Bemerkern zum Beispiel durch "grosse" Koeffizienten.

Eine Möglichkeit, dies zu verhindern ist **RIDGE-REGRESSION**. Hierbei fügen wir einen Strafterm zu unserer Verlustfunktion hinzu, welcher grosse Koeffizienten bestraft. Es gilt die neue Verlustfunktion

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i, \mathbf{w}))^2 + \alpha \|\mathbf{w}\|^2.$$

Das Modell dafür in Scikit-learn ist Ridge. Es wird implementiert mit

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=0.1)
model.fit(X_poly, y)
```

Für höhere Dimensionen können wir unsere Daten nicht plotten. Wie können wir also feststellen, ob unser Modell gut ist? Dazu können wir nur einen Teil der Daten verwenden, um das Modell zu trainieren, und den Rest der Daten verwenden, um die Vorhersage zu testen.

Um einen guten Wert für den Polynomgrad zu finden, können wir ausprobieren.

Wir teilen unseren Datensatz nun auf in **TRAININGSDATEN**, **VALIDIERUNGSDATEN** und **TESTDATEN**. Wichtig

ist, dass die Testdaten zu keiner Zeit während des Trainingsprozesses verwendet werden, da sie sonst nicht mehr als Testdaten gelten würden. In Scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.model_selection import
↳ train_test_split
X_train, X_test, y_train, y_test =
↳ train_test_split(X, y, test_size=0.2)
```

Wichtig, anschliessend muss der StandardScaler auf die Trainingsdaten gefittet werden, damit die Testdaten nicht in den Trainingsprozess einbezogen werden. Anschliessend können wir beide Datensätze transformieren mit dem StandardScaler, welcher auf die Trainingsdaten gefittet wurde.

Wir definieren Weiter die Gewichte, welche wir optimieren als **PARAMETER** und die selbst gewählten Eingaben wie α als **HYPERPARAMETER**.

Um die Güte des Modells zu beurteilen, können wir die **ANPASSUNGSGÜTE** R^2 verwenden, welche definiert ist als

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

Wenn R^2 nahe bei 1 liegt, so ist das Modell gut. Wenn R^2 nahe bei 0 liegt, so ist das Modell schlecht. Wenn R^2 negativ ist, so ist das Modell sogar schlechter als die Vorhersage durch den Mittelwert. Dies wäre dann ganz schlecht. In Scikit-learn können wir R^2 wie folgt berechnen

```
from sklearn.metrics import r2_score
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
```

Um die optimalen Hyperparameter zu finden, können wir **KREUZVALIDIERUNG** verwenden. Dazu teilen wir die Trainingsdaten in k Teile auf. Wir trainieren nun auf $k - 1$ **FOLDS** und testen auf dem verbleibenden Fold. Dies wiederholen wir nun k mal, sodass jeder Fold einmal als Testdaten verwendet wird. Wir können nun die durchschnittliche Anpassungsgüte über alle Folds berechnen, um die Güte des Modells zu beurteilen. In Scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.model_selection import
↳ cross_val_score
model = LinearRegression()
cross_val_score(model, X_train, y_train, cv=5,
↳ scoring='r2')
```

Dies können wir nun verwenden um die besten Hyperparameter mit **GRID SEARCH** zu finden. Dabei definieren wir ein Gitter von Hyperparametern und berechnen die Kreuzvalidierung für jedes Hyperparameter-Set. Das Hyperparameter-Set mit der besten Kreuzvalidierung ist dann das beste Hyperparameter-Set. In Scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.model_selection import
↳ GridSearchCV
p_grid = {'alpha': [0, 0.01, 0.1, 1, 10]}
ridge = Ridge()
gs = GridSearchCV(ridge, p_grid, cv=5,
↳ scoring='r2')
gs.fit(X_train, y_train)
```

Zur Erinnerung: Supervised Learning besteht aus einem Supervisor, welcher jedem Input x ein korrektes Label y zuordnet. Wir möchten uns nun auf die binäre Klassifikation konzentrieren. Hierbei gibt es zwei Klassen, welche wir mit $y = 0$ und $y = 1$ bezeichnen. Nun brauchen wir ein Modell, für die binäre Klassifikation.

$$\hat{y} = f(x, w_1, w_0) = \sigma(w_1 \cdot x + w_0).$$

Hierbei ist σ die Logistische Funktion, welche definiert ist als

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Wir können dies Interpretieren als die Wahrscheinlichkeit, dass Merkmal x zur Klasse $y = 1$ gehört.

$$P(Y = 1|X = x) = \sigma(w_1 \cdot x + w_0).$$

Ein grösseres w_1 bedeutet, eine stärkere Schärfe in der Trennung, während w_0 die Position der Trennlinie bestimmt. Um schliesslich die Entscheidung zu fällen, ob x zur Klasse $y = 0$ oder $y = 1$ gehört, können wir einen Schwellenwert definieren, zum Beispiel 0.5. Wenn $\hat{y} > 0.5$, so ordnen wir x der Klasse $y = 1$ zu, andernfalls der Klasse $y = 0$.

In diesem Modell wird die Binary Cross Entropy als Verlustfunktion verwendet, welche definiert ist als

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})].$$

Diese Funktion ist dann hoch, wenn die Vorhersage mit hoher Konfidenz falsch ist.

Die Populärste Methode für die Güte des Modells ist die **ACCURACY**, welche definiert ist als

$$\text{accuracy} = \frac{\text{Anzahl korrekter Vorhersagen}}{\text{Anzahl aller Vorhersagen}}.$$

So eine logistische Regression können wir auch in höheren Dimensionen durchführen. Das Modell ist dann gegeben durch

$$f(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{i=1}^d w_i x_i + w_0\right).$$

In scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.linear_model import
↳ LogisticRegression
from sklearn.metrics import accuracy_score
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
y_prob = model.predict_proba(X_test)[: , 1]
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
```

Ein nachteil an solch einem linearen Modell ist, dass die Entscheidungsgrenze immer linear ist. Um dies zu umgehen, können wir die Daten in einen höherdimensionalen Raum transformieren, um dann eine lineare Trennung durchzuführen. Wir machen also eine polynomielle Regression analog zur Regression.

Prinzipiell können wir auch andere Funktionen als **BA-SISFUNKTIONEN** verwenden, welche die Features transformieren. Somit ergibt sich dann die **TRANSFORMATIONSFUNKTION** $\phi(x) = [\phi_1(x), \phi_2(x), \dots, \phi_m(x)]$.

In scikit-learn können wir dies wie folgt umsetzen

```
def basisTransform(X):
    return np.hstack((X, np.sin(X), np.cos(X)))
```

```
transformer =
↳ FunctionTransformer(basisTransform)
X_transformed = transformer.fit_transform(X)
```

```
model = LogisticRegression()
model.fit(X_transformed, y)
```

5.1 Neuronale Netze

Wie finden wir nun eine optimale Basisfunktion? In einer idealen Welt können wir alle möglichen Funktionen als Basisfunktionen verwenden. Gradient Decent würde dann die Koeffizienten der Funktionen so anpassen, dass die Verlustfunktion minimiert wird. In der Praxis funktioniert dies jedoch nicht sehr gut.

Die Idee ist nun, die Basisfunktionen selbst mitzulernen. Das heisst, wir lernen eine lineare Transformation, gefolgt von einer nichtlinearen Aktivierungsfunktion.

$$h(\mathbf{x}, \mathbf{w}) = a\left(\sum_{i=1}^d w_i x_i + w_0\right).$$

Dies führt für jedes **NEURON** $d + 1$ neue Parameter ein. Einige Aktivierungsfunktionen sind die **RELU**, welche definiert ist als $a(z) = \max(0, z)$, die **SIGMOID**-Funktion, welche definiert ist als $a(z) = \frac{1}{1 + e^{-z}}$, und die **TANH**-Funktion, welche definiert ist als $a(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

Im Grunde können wir beliebig viele solcher **HIDDEN LAYERS** hintereinander schalten. Ein solches Netzwerk wird als **DENSE** bezeichnet, wenn jedes Neuron mit jedem Neuron des nächsten Layers verbunden ist.

In scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.neural_network import
↳ MLPClassifier
mlp =
↳ MLPClassifier(hidden_layer_sizes=(100,50),
↳ activation='relu', max_iter=500)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
```

Wir wollen noch zwei weitere Klassifikationsalgorithmen anschauen. Dazu verwenden wir den Titanic-Datensatz. Wir wollen vorhersagen, ob ein Passagier überlebt hat oder nicht.

Plotten wir zunächst die Daten Alter gegen Ticketpreis und färben die Punkte nach Überleben oder nicht. Um zu klassifizieren ob ein weiterer Passagier überlebt oder nicht, können wir uns die Umgebung des Passagiers anschauen. Formalisiert wird dies durch **K-NEAREST NEIGHBORS**. Hierbei schauen wir uns die k nächsten Nachbarn an und entscheiden dann durch Mehrheitsentscheid, ob der Passagier überlebt oder nicht.

Die wichtigen Entscheidungen bei KNN sind die Wahl von k und die Wahl der Distanzfunktion. Typischerweise wird hierbei die euklidische Distanz verwendet. Wichtig dabei ist, dass die Daten standardisiert werden müssen. Dieser Algorithmus ist nicht geeignet für hochdimensionale Daten.

In scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.neighbors import
↳ KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

Wenn wir die Entscheidungsgrenzen visualisieren wollen, können wir das wie folgt tun

```
from sklearn.inspection import
↳ DecisionBoundaryDisplay
disp =
↳ DecisionBoundaryDisplay.from_estimator(knn,
↳ X_train, response_method='predict')
disp.plot()
```

Ein weiterer Algorithmus sind Entscheidungsbäume. Hierbei wird die Datenmenge rekursiv in zwei Teile aufgeteilt, basierend auf einem Feature und einem Schwellenwert. Ein Vorteil so eines Entscheidungsbaumes ist es, dass erklärbar ist welche Entscheidungen getroffen wurden.

In scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=5,
↳ criterion='entropy', min_sample_leaf=10)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
```

5.2 Nicht-numerische Daten

Es stellt sich die Frage, wie wir mit nicht-numerischen und fehlenden Daten umgehen. Für fehlende Daten ist die einfachste Variante, die Zeilen oder Daten zu löschen. Dies kann jedoch zu Datenverlust und Bias führen.

Eine Alternative ist es, die Daten durch einen sinnvollen Wert zu ersetzen, zum Beispiel den Mittelwert, den Modus oder den Median. Man kann das auch Modellbasiert machen und mit einem ML Modell die Daten vorhersagen.

Ausserdem kann man eine Kategorie "Missing" hinzufügen, um fehlende Daten zu kennzeichnen.

Um nicht-numerische Daten zu verarbeiten, können wir zum Beispiel Ordinalzahl Kodierung verwenden. Dabei wird jeder Kategorie eine Eindeutige Zahl zugeordnet. Ein Problem dabei ist, dass die ML-Modelle die Ordnung der Zahlen interpretieren können, obwohl es keine Ordnung gibt.

Eine andere Möglichkeit ist die Zielvariablen Codierung. Dabei wird jede Kategorie durch den Durchschnitt der Zielvariablen für diese Kategorie ersetzt. Ein Problem dabei ist, dass wir eine Unterscheidung verlieren könnten, da verschiedene Kategorien den gleichen Durchschnitt haben können.

In Scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.preprocessing import
↳ OrdinalEncoder
encoder = OrdinalEncoder()
X_encoded = encoder.fit_transform(X[['categori_
↳ cal_feature']])
```

Eine Alternative ist die One-Hot-Kodierung, bei welcher für jede Kategorie eine neue Spalte erstellt wird, welche 1 ist, wenn die Kategorie vorhanden ist, und 0 sonst. Dies kann in scikit-learn wie folgt umgesetzt werden

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(drop='first',
↳ sparse_output=False)
X_encoded = encoder.fit_transform(X[['categori_
↳ cal_feature']])
```

Das Problem von One-Hot-Kodierung ist, dass sie die Anzahl der Features erhöhen kann, was zu einem Problem für ML-Modelle führen kann, insbesondere bei vielen Kategorien.

5.3 Clustering

Was können wir machen, wenn der Supervisor nicht vorhanden ist? Können wir überhaupt etwas lernen? Zwei typische Beispiele sind Dimensionalitätsreduktion und Clustering. Wir wollen uns auf das Clustering beschränken. Dazu wollen wir Punkte in Gruppen einteilen, welche sich ähnlich sind. Dabei werden wir mit Distanzen arbeiten, weshalb Standardisierung wichtig ist.

Konzeptionell wollen wir zu einer Menge von Punkten, Mengen C_1, \dots, C_k finden sodass jeder Punkt in genau einer Menge ist. Für ein möglichst gutes Clustering sollte der Verlust klein sein. Eine Mögliche Verlustfunktion ist das Bestrafen von Distanzen zwischen Punkten in demselben Cluster. Die Formel dafür wäre

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,j \in C_k} d(x_i, x_j).$$

Dies ist ein diskretes Optimierungsproblem, weshalb wir Gradientenabstieg nicht verwenden können.

Verbessert wird dies durch den K-Means Algorithmus. Dabei wählen wir die Anzahl Clusters K und wählen zufällig K Punkte als Clusterzentren. Nun wiederholen wir so oft wie nötig die folgenden Schritte: 1. Weisen wir jeden Punkt dem nächsten Clusterzentrum zu. 2. Berechnen wir die neuen Clusterzentren als Mittelwert der Punkte in jedem Cluster.

Das Problem bei K-Means ist, dass es nicht garantiert das globale Minimum findet, sondern nur ein lokales Minimum. Es ist also wichtig, mehrere Initialisierungen durchzuführen und das beste Ergebnis zu wählen. In scikit-learn können wir dies wie folgt umsetzen

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, n_init=10)
kmeans.fit(X)
labels = kmeans.labels_
```

Index

- accuracy, 13
- Anpassungsgüte, 12
- Basisfunktionen, 13
- Binomialverteilung, 8
- Bins, 2
- Dense, 13
- diskrete Auto-Kovarianz, 4
- Fehler, 2
- Folds, 12
- Full Width at Half Maximum, 6
- Garbors Limit, 5
- Genauigkeit, 2
- Grid Search, 12
- Hidden Layers, 13
- Hyperparameter, 12
- K-Nearest Neighbors, 13
- Klassifikation, 11
- Korrelation, 4
- Kreuzvalidierung, 12
- Mode, 6
- Neuron, 13
- Normalverteilung, 2
- Nyquist-Frequenz, 5
- Overfitting, 12
- Parameter, 12
- Poissonverteilung, 8
- Power Spectral Density, 5
- Prior, 7
- Probability Density Function, 6
- Probability Mass Function, 6
- Präzision, 2
- rauschen, 2
- Regression, 11
- ReLU, 13
- Residuen, 9
- Ridge-Regression, 12
- Sigmoid, 13
- tanh, 13
- Testdaten, 12
- Trainingsdaten, 12
- Transformationsfunktion, 13
- Validierungsdaten, 12
- Vergleichsmass, 11
- Verlustfunktion, 11
- Zentralmomente, 6
- Überwachtes Lernen, 11