Regularisierung

Regularisierung ist eine Technik, die in maschinellem Lernen verwendet wird, um **Overfitting** zu verhindern. Overfitting tritt auf, wenn ein Modell so stark an die Trainingsdaten angepasst wird, dass es Muster lernt, die nicht verallgemeinerbar sind (z. B. Rauschen). Regularisierung fügt eine Strafe (Penalty) zur Kostenfunktion des Modells hinzu, die komplexe Modelle bestraft und einfachere Modelle bevorzugt.

Regularisierung und ihre Rolle im maschinellen Lernen

Regularisierung ist eine Technik, die dazu beiträgt, Modelle robuster zu machen und die Balance zwischen **Overfitting** und **Underfitting** zu finden. Im Kern zielt sie darauf ab, die **Komplexität eines Modells** zu kontrollieren, indem sie eine zusätzliche "Strafkomponente" zur Kostenfunktion des Modells hinzufügt.

Wie löst Regularisierung das Problem?

Regularisierung greift ein, indem sie die Komplexität eines Modells reduziert und einfachere Modelle bevorzugt. Das geschieht durch Hinzufügen eines **Strafterms** zur Kostenfunktion des Modells. Dieser Strafterm beeinflusst die Parameter des Modells so, dass:

1. Große Werte der Modellparameter (die typischerweise zu komplexeren Modellen führen) reduziert werden.
2. Einfachere Modelle bevorzugt werden, da sie in der Regel besser generalisieren.

Warum wird die Kostenfunktion angepasst?

Ohne Regularisierung versucht das Modell, den Fehlerterm Fehlerterm zu minimieren. Dies führt oft dazu, dass das Modell übermäßig komplex wird, um jeden Fehler in den Trainingsdaten auszugleichen. Durch die Hinzufügung des Regularisierungsterms wird das Modell gezwungen, nicht nur den Fehler zu minimieren, sondern gleichzeitig die Größe oder Anzahl der Parameter klein zu halten.

Das Ergebnis:

- Modelle werden **einfacher**, was ihre Fähigkeit zur Generalisierung verbessert.

- **Overfitting** wird reduziert, da das Modell nicht in der Lage ist, sich an unwichtige Details oder Rauschen in den Daten anzupassen.

Arten der Regularisierung

1. L1-Regularisierung (Lasso-Regression)

- Fügt die absolute Summe der Koeffizienten ($|w|$) als Strafe zur Kostenfunktion hinzu.
- **Kostenfunktion:**

$$J(w) = \text{Fehler} + \lambda \sum |w_i|$$

- **Effekt:**
  - Setzt viele Koeffizienten auf genau **0**.
  - Liefert ein **sparses (dt. „spärlich“ oder „dünn“) Modell**, das nur die wichtigsten Merkmale verwendet.

2. L2-Regularisierung (Ridge-Regression)

- Fügt die quadrierte Summe der Koeffizienten (w^2) als Strafe hinzu.
- **Kostenfunktion:**

$$J(w) = \text{Fehler} + \lambda \sum w_i^2$$

- **Effekt:**
  - Verkleinert die Koeffizienten, ohne sie auf 0 zu setzen.
  - Gut geeignet für Modelle mit vielen Merkmalen, die alle relevant sind.

3. Elastic Net

- Kombination aus L1- und L2-Regularisierung.
- **Kostenfunktion:**

$$J(w) = \text{Fehler} + \lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2$$

- **Effekt:**
  - Liefert einen Kompromiss zwischen Lasso (Sparsity) und Ridge (Stabilität).

Warum Regularisierung wichtig ist

- Verhindert, dass das Modell zu stark auf die Trainingsdaten angepasst wird.
- Macht das Modell robuster und verbessert die Generalisierbarkeit auf neue Daten.
- Reduziert die Varianz des Modells, indem es extrem große Koeffizienten vermeidet.

Parameter in der Regularisierung

- λ (oft auch als `alpha` bezeichnet):
  Der Regularisierungsparameter steuert die Stärke der Strafe.
  - Ein **großer Wert** von λ führt zu starker Regularisierung (einfaches Modell).
  - Ein **kleiner Wert** von λ bedeutet schwache Regularisierung (komplexeres Modell).

Die Wahl des richtigen λ-Werts ist entscheidend und wird oft durch Techniken wie Kreuzvalidierung ermittelt.

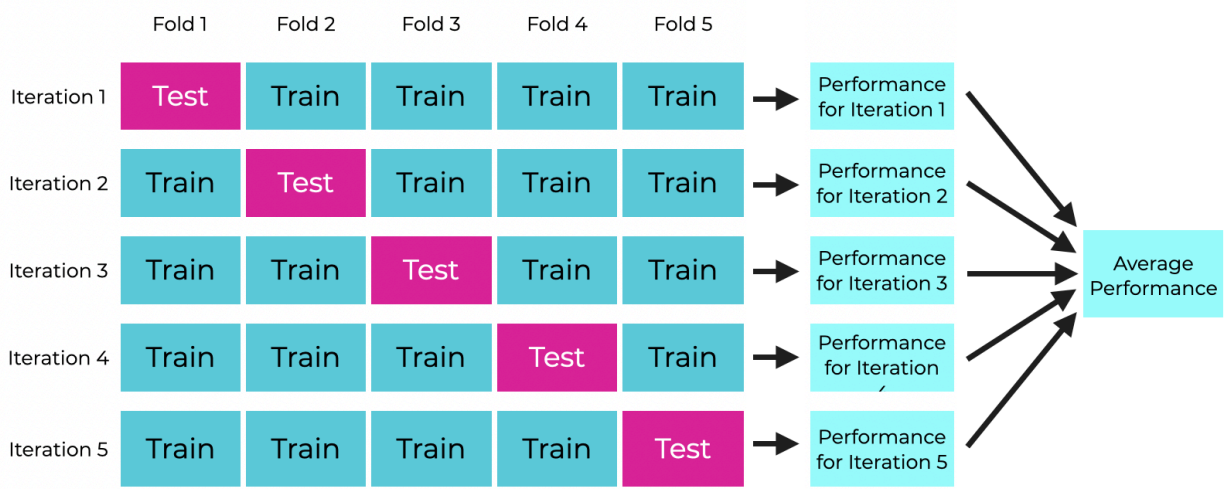
Bezug zu maschinellem Lernen

- In **Linearen Modellen** wie Ridge oder Lasso wird die Regularisierung direkt auf die Gewichte der Merkmale angewendet.
- In **nicht-linearen Modellen** (z. B. Entscheidungsbäumen) wird Regularisierung oft durch Begrenzung der Baumtiefe oder der Anzahl von Blättern implementiert.

Regularisierung hilft dabei, ein Gleichgewicht zwischen **Bias** (zu einfache Modelle) und **Varianz** (zu komplexe Modelle) zu finden und somit die **Generalisierung** zu verbessern.

Cross-Validation

CROSS VALIDATION, EXPLAINED



1. Einführung in Cross-Validation

Cross-Validation ist eine Technik im maschinellen Lernen, um die **Generalisierungsfähigkeit eines Modells** zu bewerten. Sie wird verwendet, um zu überprüfen, wie gut ein Modell auf unbekannten Daten funktioniert, ohne ein separates Testset zu benötigen.

Warum ist Cross-Validation wichtig?

- Ein einfaches Training-Test-Splitting kann dazu führen, dass die Ergebnisse stark vom spezifischen Testset abhängen.
- Cross-Validation nutzt den gesamten Datensatz effizienter, da jedes Datenstück mehrfach zum Training und Testen verwendet wird.
- Es verhindert Overfitting, indem es Modelle bestraft, die gut auf einem Teil der Daten funktionieren, aber schlecht auf anderen.

2. Wie funktioniert Cross-Validation?

Beispiel: k-Fold Cross-Validation:

- Der Datensatz wird in **kkgleich große Teile** (Folds) aufgeteilt.
- Das Modell wird **k-mal trainiert und getestet**, wobei jedes Mal ein anderer Fold als Testset verwendet wird und die übrigen k−1 Folds als Trainingsset dienen.
- Am Ende wird die **durchschnittliche Leistung** über alle Folds berechnet.

Beispiel:

siehe Bild oben

3. Arten von Cross-Validation

1. **k-Fold Cross-Validation:**
    - Standardmethode, wie oben beschrieben.
2. **Stratified k-Fold Cross-Validation:**
    - Sicherstellt, dass die Klassenverteilung in jedem Fold gleich bleibt (wichtig bei unbalancierten Daten).
3. **Leave-One-Out Cross-Validation (LOOCV):**
    - Spezialfall von k-Fold, bei dem jeder Datenpunkt einmal als Testset verwendet wird (k=Anzahl der Datenpunkte).
    - Sehr rechenaufwändig, aber nützlich bei kleinen Datensätzen.