# ERKLÄRBARE KÜNSTLICHE INTELLIGENZ

Vorlesung 6

# Inhalte der Vorlesung

- Rückblick XAI-Verfahren
  - LIME
  - SHAP
  - Grad Cam
  - Counterfactual Explanations
  - Saliency Maps
  - Integrated Gradients
  - Attention Scores
  - Partial Dependence Plots
- Übungen
  - Shallow Decision Trees
  - Anchors

# LIME

- LIME ist ein modellagnostisches Verfahren, das für die lokale Erklärung einzelner Vorhersagen eines komplexen Modells entwickelt wurde.
- Es funktioniert, indem künstliche Datenpunkte in der Nähe eines Zielpunktes generiert werden.
- Diese Datenpunkte werden durch das ursprüngliche Modell klassifiziert, und anschließend wird ein einfaches, interpretiertes Modell (z. B. ein lineares Modell) erstellt, das die Vorhersagen in der lokalen Nachbarschaft erklären kann.
- Ziel ist es, die Entscheidungslogik für diesen speziellen Fall verständlich zu machen, ohne das gesamte Modell analysieren zu müssen.

# LIME

- **LIME (Local Interpretable Model-agnostic Explanations)**
- **Wann gut:**
  - Liefert lokale Erklärungen für einzelne Vorhersagen.
  - Funktioniert unabhängig vom Modelltyp.
  - Gut für Black-Box-Modelle wie Deep Learning oder Random Forest.
- **Wann nicht gut:**
  - Kann bei hochdimensionalen Daten unzuverlässig sein.
  - Ergebnisse können je nach Sampling variieren.

# SHAP

- SHAP ist ein Framework für die Erklärung von Modellvorhersagen, das auf der Theorie der Shapley-Werte aus der kooperativen Spieltheorie basiert.
- Es berechnet den Beitrag jedes Eingabefeatures zur Vorhersage, indem es die Unterschiede zwischen den Modellvorhersagen mit und ohne die entsprechenden Features analysiert.
- SHAP ist sowohl lokal als auch global einsetzbar, indem es Erklärungen für einzelne Vorhersagen oder für die generelle Logik eines Modells liefert.
- Die additive Eigenschaft von SHAP sorgt dafür, dass die Beiträge aller Features zur Gesamtvorhersage aufaddiert werden können.

# SHAP

- **SHAP (SHapley Additive exPlanations)**
- **Wann gut:**
  - Liefert konsistente, globale und lokale Erklärungen.
  - Basierend auf kooperativer Spieltheorie (Shapley-Werte).
  - Gut bei der Identifikation von Feature-Wichtigkeit.
- **Wann nicht gut:**
  - Sehr rechenintensiv für komplexe Modelle.
  - Kann bei großen Datensätzen ineffizient sein.

# GradCam

- Grad-CAM ist ein Visualisierungsverfahren, das insbesondere in Convolutional Neural Networks (CNNs) verwendet wird.
- Es analysiert die Gradienten der Ausgabe der Modellklasse bezüglich der Aktivierungen in den letzten Convolutionsschichten.
- Diese Informationen werden genutzt, um Heatmaps zu erstellen, die aufzeigen, welche Bereiche der Eingabedaten (z. B. eines Bildes) am meisten zur Vorhersage beigetragen haben.
- Grad-CAM wird häufig zur Erklärung von Bildmodellen verwendet und eignet sich gut, um wichtige visuelle Muster hervorzuheben.

# GradCam

- **Grad-CAM (Gradient-weighted Class Activation Mapping)**
- **Wann gut:**
  - Funktioniert gut für neuronale Netze, insbesondere für Bildverarbeitung.
  - Liefert visuelle Erklärungen (Heatmaps).
- **Wann nicht gut:**
  - Nicht geeignet für tabellarische oder textbasierte Daten.
  - Funktioniert nur mit Modellen, die Gradienten unterstützen.

# Counterfactual Explanations

- Counterfactual Explanations konzentrieren sich darauf, aufzuzeigen, wie eine Vorhersage geändert werden könnte, indem die Eingabewerte minimal angepasst werden.
- Zum Beispiel könnte eine Kreditentscheidung mit einem Gegenbeispiel erklärt werden: „Hätte das Einkommen um 5 % höher gelegen, wäre der Kredit bewilligt worden.“
- Dieses Verfahren hilft, kausale Zusammenhänge und Schwellenwerte zu verstehen.
- Es bietet klare und intuitive Erklärungen, die sich auf spezifische Eingaben beziehen und alternative Ergebnisse beleuchten.

# Counterfactual Explanations

- **Counterfactual Explanations**
- **Wann gut:**
  - Zeigt, wie sich Eingabedaten ändern müssten, um eine andere Entscheidung zu erreichen.
  - Gut, um Entscheidungslogik in sensiblen Kontexten zu verstehen (z.B. Kreditbewilligung).
- **Wann nicht gut:**
  - Kann bei komplexen Modellen schwer zu berechnen sein.
  - Möglicherweise keine eindeutige Lösung in hochdimensionalen Räumen.

# Saliency Maps

- Modellvorhersage besonders wichtig sind. Sie analysieren die Gradienten der Vorhersage bezüglich der Eingabe, um die Sensitivität des Modells gegenüber Veränderungen bestimmter Eingabewerte zu messen.
- Das Ergebnis ist oft eine Heatmap, die die wichtigsten Bereiche der Eingabe hervorhebt. Dieses Verfahren wird häufig in Bildverarbeitungsanwendungen eingesetzt, um zu zeigen, welche Pixel die Vorhersage beeinflusst haben.

# Saliency Maps

- **Saliency Maps**
- **Wann gut:**
  - Gut für Bilddaten, um wichtige Regionen zu identifizieren.
  - Basiert auf Gradienteninformationen.
- **Wann nicht gut:**
  - Funktioniert nicht für tabellarische Daten.
  - Kann verrauschte oder unklare Ergebnisse liefern.

# Integrated Gradients

- Integrated Gradients ist ein Gradienten-basiertes Verfahren, das eine Basislinie (z. B. einen Nullwert für jedes Eingabefeature) als Ausgangspunkt verwendet.
- Es berechnet die Veränderung der Modellvorhersage entlang eines geraden Pfades von dieser Basislinie zur Eingabe.
- Die Gradienten entlang dieses Pfades werden integriert, um die Bedeutung jedes Features zu bestimmen.
- Dieses Verfahren liefert eine theoretisch fundierte Methode, um den Einfluss von Features auf Vorhersagen zu quantifizieren, ohne dass zusätzliche Daten benötigt werden.

# Integrated Gradients

- **Integrated Gradients**
- **Wann gut:**
  - Liefert genaue und konsistente Feature-Wichtigkeiten.
  - Funktioniert für neuronale Netze.
- **Wann nicht gut:**
  - Kann rechenintensiv sein.
  - Benötigt Zugriff auf interne Gradienten.

# Attention Scores

- Attention Scores stammen aus Modellen mit Attention-Mechanismen, wie sie in Transformer-Architekturen verwendet werden (z. B. BERT, GPT).
- Diese Scores zeigen, welche Teile der Eingabe (z. B. Wörter in einem Satz) am meisten zur Modellvorhersage beigetragen haben.
- Sie werden als Gewichte dargestellt, die den Fokus des Modells auf spezifische Eingabebereiche reflektieren.
- Attention Scores bieten eine visuelle und intuitive Möglichkeit, die Entscheidungslogik des Modells zu verstehen, insbesondere in der Verarbeitung natürlicher Sprache.

# Attention Scores

- **Attention Scores**
- **Wann gut:**
  - Besonders gut bei NLP-Modellen, um relevante Token zu identifizieren.
  - Liefert intuitive Erklärungen bei Transformer-Modellen (z.B. BERT).
- **Wann nicht gut:**
  - Kann in die Irre führen, da Aufmerksamkeit nicht direkt mit Wichtigkeit gleichzusetzen ist.
  - Nicht geeignet für Modelle ohne Attention-Mechanismus.

# Partial Dependence Plots

- Partial Dependence Plots visualisieren den Einfluss eines oder mehrerer Eingabefeatures auf die Modellvorhersage, während alle anderen Features konstant gehalten werden.
- Sie zeigen den Durchschnittseffekt eines Features, indem für verschiedene Werte dieses Features die Modellvorhersage berechnet wird.
- PDPs helfen dabei, die globale Beziehung zwischen einem Feature und dem Zielwert zu verstehen, indem sie Trends wie lineare, nichtlineare oder monotone Zusammenhänge verdeutlichen.
- Dieses Verfahren ist vor allem bei tabellarischen Daten verbreitet.

# Partial Dependence Plots

- **Partial Dependence Plots (PDPs)**
- **Wann gut:**
  - Liefert globale Erklärungen und zeigt den Einfluss eines Features auf das Modell.
  - Gut für tabellarische Daten und Interpretationen auf Datensatzebene.
- **Wann nicht gut:**
  - Unzuverlässig bei stark korrelierten Features.
  - Kann nicht lokale Entscheidungen erklären.

# Übersicht und Vergleich der Verfahren

| Methode | Datenarten | Lokale Erklärung | Globale Erklärung | Modelltyp | Herausforderungen |
|---|---|---|---|---|---|
| **LIME** | Alle | ✅ | ❌ | Agnostisch | Variabilität in Erklärungen |
| **SHAP** | Alle | ✅ | ✅ | Agnostisch | Rechenaufwändig |
| **Grad-CAM** | Bilder | ✅ | ❌ | DL | Nur für Bildmodelle |
| **Counterfactuals** | Alle | ✅ | ❌ | Agnostisch | Rechenintensiv, keine eindeutige Lösung |
| **Saliency Maps** | Bilder | ✅ | ❌ | DL | Verrauschte Ergebnisse |
| **Integrated Gradients** | Alle (hauptsächlich DL) | ✅ | ✅ | DL | Gradienten notwendig |
| **Attention Scores** | Text (NLP) | ✅ | ❌ | DL | Missverständnisse bei Aufmerksamkeit |
| **PDPs** | Tabellarisch | ❌ | ✅ | Agnostisch | Starke Feature-Korrelationen problematisch |

# Anchors

- **Anchors** ist ein Verfahren aus dem Bereich der erklärbaren künstlichen Intelligenz (XAI), das lokale Erklärungen für Modellvorhersagen liefert.
- Im Gegensatz zu anderen Methoden (z. B. LIME) legt Anchors besonderen Wert darauf, Regeln zu finden, die hochpräzise und intuitiv sind, sogenannte **"Anker-Regeln"**.
- Diese Regeln definieren Bedingungen, unter denen eine Modellvorhersage gültig bleibt, unabhängig von kleinen Änderungen der Eingabedaten.

# Anchors Allgemein

- Anchors ist XAI-Verfahren und wurde von den LIME Entwicklern Marco Tulio Ribeiro, Sameer Singh und Carlos Guestrin entworfen.
- Es folgt den Algorithmen LIME und aLIME, welche ebenfalls von diesen Autoren entwickelt wurden.
- Anchors kann als Erweiterung dieser gesehen werden, da seine Funktionsweise ähnlich ist, jedoch im Detail diverse Verbesserungen aufweist.
- Anchors ist ein Post-hoc-Erklärer,
  - liefert lokale, modell agnostische Erklärungen, genannt anchors.
  - Diese besitzen ein IF-THEN Format und haben eine klar definierte Coverage, was bedeutet, dass ihre Gültigkeit bzw. Bedeutung im Inputbereich deutlich bekannt ist. Dies macht Erklärungen intuitiv und leicht verständlich.
  - Im Bild-Bereich sind es Super-Pixel

## Wie funktioniert Anchors?

1. **Datengenerierung in der Nachbarschaft:**
    1. Um eine Anker-Regel zu validieren, werden synthetische Datenpunkte in der Nähe der Eingabedaten generiert. Diese Punkte werden durch das ursprüngliche Modell klassifiziert.
2. **Regelsuche:**
    1. Das Verfahren sucht nach Regeln, die die Vorhersage für möglichst viele Datenpunkte in der Nachbarschaft korrekt erklären. Dabei wird geprüft, ob die Regel für diese Punkte weiterhin zutrifft.
3. **Präzision und Abdeckung:**
    1. Die Präzision misst, wie genau die Regel die Vorhersage erklärt (Anteil der korrekt erklärten Vorhersagen in der Nachbarschaft).
    2. Die Abdeckung misst, wie viele der Nachbarschaftsdatenpunkte durch die Regel abgedeckt werden.
4. **Optimierung der Regeln:**
    1. Das Verfahren optimiert die Anker-Regeln, um eine Balance zwischen hoher Präzision und guter Abdeckung zu erreichen.

# Anchors

- **Wie funktioniert Anchors?**

1. **Datengenerierung in der Nachbarschaft:**
    1. Um eine Anker-Regel zu validieren, werden synthetische Datenpunkte in der Nähe der Eingabedaten generiert. Diese Punkte werden durch das ursprüngliche Modell klassifiziert.
2. **Regelsuche:**
    1. Das Verfahren sucht nach Regeln, die die Vorhersage für möglichst viele Datenpunkte in der Nachbarschaft korrekt erklären. Dabei wird geprüft, ob die Regel für diese Punkte weiterhin zutrifft.
3. **Präzision und Abdeckung:**
    1. Die Präzision misst, wie genau die Regel die Vorhersage erklärt (Anteil der korrekt erklärten Vorhersagen in der Nachbarschaft).
    2. Die Abdeckung misst, wie viele der Nachbarschaftsdatenpunkte durch die Regel abgedeckt werden.

# Anchors

- **4. Optimierung der Regeln:**
- Das Verfahren optimiert die Anker-Regeln, um eine Balance zwischen hoher Präzision und guter Abdeckung zu erreichen.

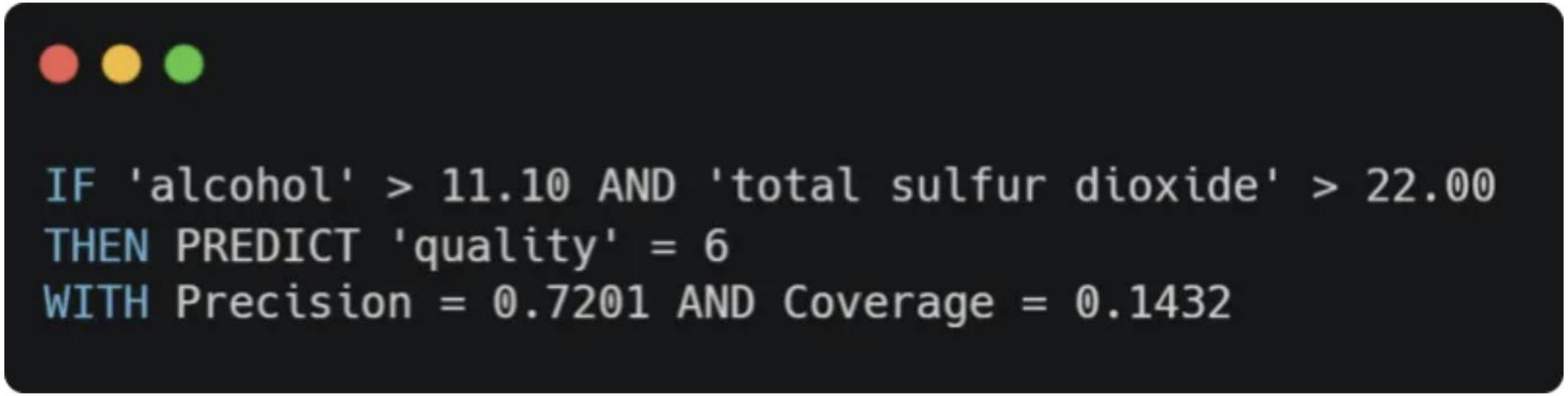
# Anchors Beispiel

- Angenommen, ein Modell entscheidet, ob ein Bewerber für einen Kredit zugelassen wird. Anchors könnte eine Regel wie diese finden:

  **Wenn Einkommen > 50.000 EUR und keine vorherigen Kreditausfälle, dann "Kredit bewilligt".**

- Diese Regel sagt aus, dass das Modell immer „Kredit bewilligt“ vorhersagen wird, solange diese beiden Bedingungen erfüllt sind. Kleinere Änderungen wie das Alter oder der Wohnort haben in diesem Fall keinen Einfluss auf die Entscheidung.

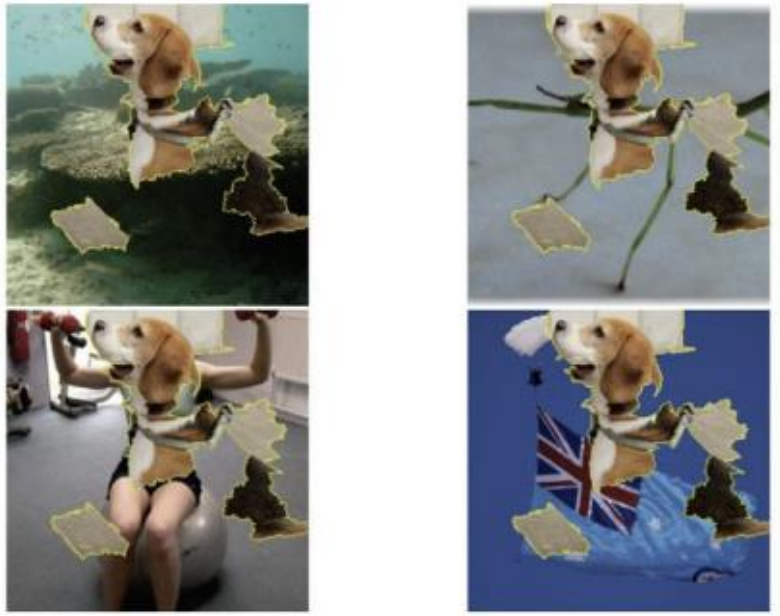
# Anchor Beispiel: If-Then Regel



# Anchor Beispiel: Superpixel

(a) Original image

(b) Anchor for “beagle”



(c) Images where Inception predicts $P(\text{beagle}) > 90\%$

# Shallow Decision Trees

- **Shallow Decision Trees** sind Entscheidungsbäume mit einer begrenzten Tiefe. Sie sind eine vereinfachte Version von klassischen Entscheidungsbäumen und dienen dazu, die Entscheidungslogik eines Modells in einer leicht interpretierbaren Weise darzustellen.
- Ihre begrenzte Tiefe sorgt dafür, dass die Erklärungen für Vorhersagen leicht verständlich bleiben, was sie zu einem wichtigen Werkzeug in der erklärbaren künstlichen Intelligenz (XAI) macht.

# Shallow Decision Trees

- Eigenschaften von Shallow Decision Trees
  - Begrenzte Tiefe:Die maximale Tiefe eines Shallow Decision Trees ist klein (z. B. max_depth=2 oder max_depth=3).Wenige Ebenen bedeuten weniger Regeln, was die Interpretierbarkeit verbessert.
  - Einfache Entscheidungslogik: Die Baumstruktur stellt Entscheidungen durch Wenn-Dann-Regeln dar, die leicht zu interpretieren sind.
  - Lokale und globale Erklärungen: Sie können sowohl einzelne Vorhersagen (lokale Erklärung) als auch die generelle Entscheidungslogik (globale Erklärung) eines Modells verdeutlichen.
  - Schnelles Training: Aufgrund der geringen Tiefe ist der Trainingsprozess sehr schnell.
  - Modellagnostisch: Shallow Decision Trees können als eigenständige Modelle trainiert oder verwendet werden, um komplexe Modelle wie neuronale Netze oder Random Forests zu approximieren.

# Shallow Decision Trees

- **Vorteile von Shallow Decision Trees**

1. **Hohe Interpretierbarkeit:**
    1. Die einfache Struktur macht sie ideal für die Erklärbarkeit in sensiblen Anwendungsbereichen wie Medizin oder Finanzwesen.
2. **Effiziente Berechnungen:**
    1. Wenige Ebenen bedeuten weniger Berechnungen, was sie für schnelle Analysen geeignet macht.
3. **Robust gegen Overfitting:**
    1. Eine begrenzte Tiefe reduziert das Risiko, dass das Modell zu stark an die Trainingsdaten angepasst wird.
4. **Visualisierbarkeit:**
    1. Die Baumstruktur kann leicht als Diagramm dargestellt werden, was die Entscheidungslogik verdeutlicht.

# Shallow Decison Trees

- Aufgabe 2: Vorhersage mit einem Shallow Decision Tree
  - Trainiere einen Shallow Decision Tree und teste ihn auf neuen Datenpunkten.
  - Anforderungen: Lade den Iris-Datensatz.Trainiere einen Shallow Decision Tree mit max_depth=3.
  - Wähle einen zufälligen Datenpunkt aus dem Testset.
  - Lasse das Modell Vorhersagen für diesen Datenpunkt machen und vergleiche es mit der tatsächlichen Klasse.

# Lösung

```
# Aufgabe 2: Vorhersage mit einem Shallow Decision Tree

import numpy as np

# Zufälligen Datenpunkt aus dem Testset auswählen
random_index = np.random.randint(0, len(X_test))
random_sample = X_test[random_index].reshape(1, -1)
true_label = y_test[random_index]

# Vorhersage
predicted_label = shallow_tree.predict(random_sample)

print(f"Zufälliger Datenpunkt: {random_sample}")
print(f"Tatsächliche Klasse: {true_label}")
print(f"Vorhersage: {predicted_label[0]}")
```

- Aufgabe 3: Visualisierung eines Shallow Decision Tree
  - Trainiere einen Shallow Decision Tree und visualisiere ihn als Diagramm.
  - Anforderungen:Lade den Titanic-Datensatz (z. B. von Kaggle oder einer anderen Quelle).
  - Verarbeite die Daten so, dass sie für den Entscheidungsbaum geeignet sind (z. B. binäre Kategorien für Überlebensstatus).
  - Trainiere einen Shallow Decision Tree mit max_depth=2.
  - Visualisiere den Baum mit plot_tree aus sklearn.tree.

# Lösung

- # Aufgabe 3: Visualisierung eines Shallow Decision Tree

- from sklearn.tree import plot_tree
- import matplotlib.pyplot as plt

- # Entscheidungsbaum visualisieren
- plt.figure(figsize=(12, 8))
- plot_tree(shallow_tree, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
- plt.title("Shallow Decision Tree")
- plt.show()

- Aufgabe 4: Evaluierung eines Shallow Decision Tree
  - Vergleiche die Performance eines Shallow Decision Tree mit einem tieferen Entscheidungsbaum.
  - Anforderungen: Lade den Iris-Datensatz.
  - Trainiere zwei Entscheidungsbäume:
    - Einen mit max_depth=2.
    - Einen mit max_depth=None (volle Tiefe).
  - Berechne die Genauigkeit für beide Modelle auf dem Testdatensatz und vergleiche die Ergebnisse.

```
# Aufgabe 4: Evaluierung eines Shallow Decision Tree

from sklearn.metrics import accuracy_score

# Zweites Modell (tiefer Entscheidungsbaum)
deep_tree = DecisionTreeClassifier(random_state=42)
deep_tree.fit(X_train, y_train)

# Vorhersagen
shallow_predictions = shallow_tree.predict(X_test)
deep_predictions = deep_tree.predict(X_test)

# Genauigkeit berechnen
shallow_accuracy = accuracy_score(y_test, shallow_predictions)
deep_accuracy = accuracy_score(y_test, deep_predictions)

print(f"Genauigkeit Shallow Tree (max_depth=2): {shallow_accuracy:.2f}")
print(f"Genauigkeit Deep Tree (volle Tiefe): {deep_accuracy:.2f}")
```

# Anchors Aufgabe

- **Aufgabe 1: Präzision und Abdeckung von Anker-Regeln analysieren**

1. Wähle fünf zufällige Testdatenpunkte aus.
2. Erstelle für jeden Punkt eine Anker-Erklärung.
3. Sammle die Präzision und Abdeckung der Regeln in einer Tabelle.

- **Ergebnis:**
  Analysiere, wie sich die Präzision und Abdeckung der Anker-Regeln zwischen den Datenpunkten unterscheiden.

```python
# Aufgabe 2: Präzision und Abdeckung analysieren
results = []

# Fünf zufällige Testdatenpunkte
for idx in np.random.choice(len(X_test), 5, replace=False):
    instance = X_test[idx].reshape(1, -1)
    explanation = explainer_anchor.explain(instance[0])
    results.append({
        "Index": idx,
        "Tatsächliche Klasse": class_names[y_test[idx]],
        "Vorhersage": class_names[model.predict(instance)[0]],
        "Anker-Regeln": " und ".join(explanation.anchor),
        "Präzision": explanation.precision,
        "Abdeckung": explanation.coverage
    })

# Ergebnisse in einer Tabelle anzeigen
df_results = pd.DataFrame(results)
print("Analyse der Präzision und Abdeckung:")
df_results
```

# Anchors Aufgabe

- **Aufgabe 2: Vergleich zwischen LIME und Anchors**

1. Verwende den Iris-Datensatz und trainiere ein Modell.
2. Erstelle Erklärungen für denselben Testdatenpunkt mit **LIME** und **Anchors**.
3. Vergleiche die Ergebnisse beider Methoden.

- **Ergebnis:**
  Welche Methode bietet präzisere und besser verständliche Erklärungen? Notiere die Unterschiede in den Ergebnissen.

```
# Aufgabe 5: Vergleich zwischen LIME und Anchors
idx = np.random.randint(0, len(X_test))  # Zufälliger Testpunkt
instance = X_test[idx].reshape(1, -1)

# Anchor-Erklärung
anchor_exp = explainer_anchor.explain(instance[0])
anchor_rules = " und ".join(anchor_exp.anchor)

# LIME-Erklärung
lime_exp = explainer_lime.explain_instance(instance[0], model.predict_proba, num_features=2)

print(f"Testdatenpunkt: {instance}")
print(f"Tatsächliche Klasse: {class_names[y_test[idx]]}")
print(f"Vorhersage: {class_names[model.predict(instance)[0]]}")

print("\nAnchor-Erklärung:")
print(f"Regeln: {anchor_rules}")
print(f"Präzision: {anchor_exp.precision:.2f}, Abdeckung: {anchor_exp.coverage:.2f}")

print("\nLIME-Erklärung:")
lime_exp.show_in_notebook()
```

- **Aufgabe 3: Anker-Regeln für mehrere Klassen vergleichen**

1. Lade den Iris-Datensatz und trainiere ein Modell.
2. Erstelle Anker-Erklärungen für Testdatenpunkte aus verschiedenen Klassen.
3. Vergleiche die Regeln zwischen den Klassen (z. B. Setosa vs. Versicolor).

- **Ergebnis:**
  Gibt es Unterschiede in den Regeln zwischen den Klassen? Wenn ja, welche Features sind für die Unterscheidung entscheidend?

```python
# Aufgabe 6: Anker-Regeln für mehrere Klassen
class_results = {class_name: [] for class_name in class_names}

# Testdatenpunkte nach Klassen sortieren
for idx in range(len(X_test)):
    instance = X_test[idx].reshape(1, -1)
    prediction = class_names[model.predict(instance)[0]]
    explanation = explainer_anchor.explain(instance[0])
    class_results[prediction].append(" und ".join(explanation.anchor))

# Vergleich der Regeln pro Klasse
for class_name, rules in class_results.items():
    print(f"\nKlasse: {class_name}")
    print(f"Anker-Regeln (erste 3 Beispiele):")
    for rule in rules[:3]:
        print(f"- {rule}")
```