**DIPLOMARBEIT**

DermaAI: Intelligente Hautanalyse

Gesamtprojekt

Entwicklung von KI-Modellen zur Detektion und Klassifikation pigmentierter Hautläsionen und deren Integration in eine Mobile App

Diplomarbeitsnummer  
5AHINF-24/25-DA11

**Datenbank, Verwaltung der medizinischen Daten**

Jonas Maier 5AHINF Betreuer: Dipl.-Ing. Dr.

Gerhard Gaube

**Mobile Integration, Frontend**

Jonas Bogensberger 5AHINF Betreuer: MSC

Michael Prader

**KI-Modelle in Python, API und Appanbindung**

Daniel Jessner 5AHINF Betreuer: Dipl.-Ing. Dr.

Gerhard Gaube

Schuljahr 2024/25

Abgabevermerk:

Datum: TT.MM.JJJJ übernommen von:

(VORHER THEORIE DANN UMSETZUNG)

(Theorie in Projektplanung)

# Allgemeines & Projektübersicht

## Projektbeschreibung

## Projektteam und Schnittstellen

## Funktionale Anforderungen

## Nicht Funktionale Anforderungen

## Projektplanung

### Recherche / Vorarbeit

#### Daniel Jessner (KI & Python)

##### Einführung

Bevor mit dem eigentlichen Training von KI-Modellen begonnen werden kann, ist eine gründliche Recherche und Vorarbeit essenziell. Eine fundierte theoretische Grundlage ermöglicht es, die geeigneten Algorithmen und Frameworks auszuwählen sowie potenzielle Herausforderungen frühzeitig zu erkennen. Zudem hilft die Recherche dabei, bewährte Methoden zur Datenaufbereitung, Modelloptimierung und Fehlervermeidung zu identifizieren. Da sich das Projektteam mit dieser Arbeit in ein ihnen unbekanntes Terrain wagt, ist dieser Schritt unerlässlich für den Erfolg des Projektes.

Die Vorarbeit umfasst mehrere wichtige Schritte, die die Qualität und Effizienz des Modelltrainings maßgeblich beeinflussen. Dazu gehören die Auswahl geeigneter Technologien, die Beschaffung und Bereinigung von Trainingsdaten (eigene Projekt-Komponente) sowie erste Tests mit Basisalgorithmen. Durch eine systematische Vorgehensweise wird sichergestellt, dass das spätere Modelltraining auf einer stabilen und gut vorbereiteten Grundlage aufbaut.

##### Theoretische Grundlagen – KI

Diese Projekt-Komponente beschäftigt sich zum größten Teil mit der Verwaltung und auch Entwicklung von KI-Modellen, um Eingabedaten anhand zuvor definierter Trainingsdaten zu klassifizieren. Dabei bedient sie sich sogenannten „**Classifiers**“, also KI-Modellen, welche ein solches Konzept realisieren. Diese Classifier sind definierte, oftmals komplexe Algorithmen und basieren auf zwei Ansätzen:

1. **Maschinelles Lernen** ist ein Bereich der Künstlichen Intelligenz, der es Computern ermöglicht, Muster in Daten zu erkennen und Vorhersagen zu treffen, ohne explizit programmiert zu sein. Dabei unterscheidet man zwischen verschiedenen Lernarten:

* **Überwachtes Lernen**: Das Modell wird mit gekennzeichneten Daten (Labels) trainiert, um eine bestimmte Ausgabe vorherzusagen (relevant für dieses Projekt).
* **Unüberwachtes Lernen**: Das Modell erkennt Muster in unmarkierten Daten, z. B. durch Clustering.
* **Bestärkendes Lernen**: Das Modell lernt durch Belohnungen aus Interaktionen mit seiner Umgebung.

2. **Deep Learning** ist eine spezielle Form des maschinellen Lernens, die auf künstlichen neuronalen Netzen basiert. Diese Netzwerke bestehen aus mehreren Schichten (Deep Neural Networks) und sind besonders gut für komplexe Mustererkennung in großen Datenmengen geeignet. Im Vergleich zu klassischen ML-Algorithmen können DL-Modelle automatisch relevante Merkmale aus den Daten extrahieren, was sie für Aufgaben wie Bild- und Spracherkennung besonders leistungsfähig macht. Dieser Ansatz des maschinellen Lernens wird von CNNs, also **Convolutional Neural Networks**, implementiert, worauf später noch genauer eingegangen wird.

##### Auswahl geeigneter Technologien

Grundsätzlich war von Anfang an klar, dass sich im Bezug auf verwendete Technologien sowie das Basiskonstrukt des Projektes an die Vorgaben des Tech-Stacks des Projektgebers (siehe Kapitel XY) gehalten wird. Dadurch fiel die Entscheidung auf relativ einfach auf folgende Punkte:

* **KI-Modelle**
  + Pytorch
  + Tensorflow Keras
  + Scikit-Learn
* **API**
  + FastAPI

Abgesehen von den Vorgaben sprechen noch einige weitere Aspekte für die Verwendung oben genannter Technologien:

**1. Python**

Python ist die bevorzugte Programmiersprache für Künstliche Intelligenz und maschinelles Lernen aufgrund seiner einfachen Syntax, umfangreichen Bibliotheken und starken Community-Unterstützung.

Die wichtigsten Vorteile sind:

* **Lesbarkeit und Benutzerfreundlichkeit**: Ermöglicht eine schnelle Entwicklung und erleichtert die Zusammenarbeit.
* **Breites Ökosystem an ML- und DL-Bibliotheken**: Bibliotheken wie **Scikit-Learn, TensorFlow und PyTorch** bieten leistungsfähige Tools für maschinelles Lernen und Deep Learning.
* **Effiziente Datenverarbeitung**: Python unterstützt leistungsfähige Bibliotheken wie **NumPy, Pandas und Matplotlib**, die für Datenanalyse und Visualisierung essenziell sind.
* **Gute Integration mit anderen Technologien**: Python kann leicht mit **C++, Java und cloudbasierten ML-Plattformen** integriert werden, was die Skalierbarkeit von KI-Modellen verbessert.

**2. KI-Bibliotheken**

Je nach Anwendungsfall eignen sich Frameworks unterschiedlich gut für zu bewältigende Aufgaben. Die KI-Komponente des DermaAI-Systems implementiert einen vielfältigen Mix von Modellen aus den drei verwendeten Bibliotheken.

|  |  |  |  |
| --- | --- | --- | --- |
| **Framework** | **Einsatzgebiet** | **Vorteile** | **Nachteile** |
| |  | | --- | | **Scikit-Learn** | | |  | | --- | | Klassische ML-Modelle (z. B. Entscheidungsbäume, SVMs, logistische Regression) | | Einfache Implementierung, gute Dokumentation, ideal für kleine bis mittelgroße Datensätze | Nicht für neuronale Netze oder große Datenmengen optimiert |
| |  | | --- | | **PyTorch** | | |  | | --- | | Deep Learning, flexible neuronale Netze (z. B. CNNs, RNNs, Transformer) | | |  | | --- | | Dynamische Berechnungsgrafen, einfaches Debugging, intuitive API | | Etwas weniger für großskalige Produktlösungen optimiert als TensorFlow |
| |  | | --- | | **TensorFlow** | | Skalierbares Deep Learning für Produktion, Cloud und Mobilgeräte | Hohe Effizienz, TensorFlow Serving für Deployment, GPU-Unterstützung | Komplexere API als PyTorch, steilere Lernkurve |

**-> Scikit-Learn** eignet sich besonders für klassische ML-Modelle wie Klassifikation, Regression oder Clustering.  
**-> PyTorch** ist ideal für forschungsorientierte Deep-Learning-Experimente, da es eine flexible und intuitive API bietet.  
**-> TensorFlow** ist besser für großskalige neuronale Netze in Produktionsumgebungen, insbesondere für Cloud- und Mobile-Deployments.

**3. FastAPI**

**FastAPI** ist ein modernes und leistungsstarkes Web-Framework für den Bau von APIs mit Python. Es hat einige herausragende Merkmale, die es besonders attraktiv für Entwickler machen, die APIs bauen möchten:

**Hohe Leistung**

* **FastAPI** ist eines der schnellsten Web-Frameworks in Python und bietet eine außergewöhnliche Leistung, die mit **Node.js** und **Go** konkurrieren kann. Die hohe Geschwindigkeit ist ein wichtiger Faktor, wenn es um die Entwicklung von APIs geht, da sie eine geringe Latenz und schnelle Antworten gewährleisten kann.
* Der Grund für die Geschwindigkeit ist, dass **FastAPI** auf **Starlette** basiert, einem schnellen und asynchronen Web-Framework. FastAPI verwendet die **ASGI**-Spezifikation, die eine hohe Performance bei gleichzeitigen Verbindungen ermöglicht.

**Automatische Dokumentation**

* FastAPI erstellt automatisch interaktive API-Dokumentationen für deine Endpunkte, und zwar direkt während der Entwicklung.
* Es verwendet **Swagger UI** und **ReDoc**, um eine benutzerfreundliche, visuelle API-Dokumentation bereitzustellen. Diese wird unter der URL /docs (Swagger UI) und /redoc (ReDoc) bereitgestellt.

**Vorteile**:

* + Entwicklern und API-Nutzern wird eine leicht verständliche, sofort zugängliche Dokumentation zur Verfügung gestellt.
  + Die Dokumentation ist immer aktuell, da sie auf der tatsächlichen Implementierung der API basiert.

**Einfach zu bedienen und flexibel**

* FastAPI nutzt Python-Typannotationen (zum Beispiel int, str, float, etc.), um Eingaben zu validieren, was die Entwicklung vereinfacht und weniger fehleranfällig macht. Das bedeutet, dass du keine separate Validierungslogik schreiben musst, sondern die Typen automatisch validiert werden.
* Durch die Verwendung von **Pydantic** für die Datenvalidierung können Modelle sehr einfach und effizient erstellt werden. Pydantic sorgt für automatische Datentypprüfung und Fehlerbehandlung.

**Asynchronität (Async / Await)**

* FastAPI ermöglicht es, asynchrone Endpunkte zu erstellen, die bei Bedarf sehr viele gleichzeitige Anfragen effizient verarbeiten können. Das bedeutet, dass du **async**/**await** verwenden kannst, um **asynchrone Programmierung** zu integrieren und z. B. auf **Datenbanken** oder **Web-APIs** effizient zuzugreifen.
* Diese asynchronen Endpunkte bieten eine deutlich bessere Leistung bei hoher Last und stellen sicher, dass deine API schnell bleibt, selbst wenn viele gleichzeitige Anfragen verarbeitet werden müssen.

**Sicherheit und Authentifizierung**

* FastAPI bietet Unterstützung für gängige **Authentifizierungsmethoden** wie **OAuth2**, **JWT (JSON Web Tokens)**, und **API-Schlüssel**.
* Es gibt eingebaute Mechanismen, um sicherheitsrelevante Operationen wie **CORS (Cross-Origin Resource Sharing)**, **CSRF (Cross-Site Request Forgery)** und **Request Validation** effizient zu implementieren.

**Einfacher Einstieg und gute Dokumentation**

* FastAPI hat eine **exzellente Dokumentation**, die für Einsteiger gut verständlich ist. Die Dokumentation führt Schritt für Schritt durch den Prozess der API-Entwicklung und hilft dabei, Best Practices zu etablieren.
* Die Nutzung von Python-Typen für **Datenvalidierung** und **Autovervollständigung** in modernen IDEs wie **VS Code** erleichtert das Schreiben von Code.

**Unterstützung für moderne Python-Funktionen**

* FastAPI nutzt moderne Python-Funktionen wie **Type Hints** und **Pydantic-Modelle**. Dadurch kannst du sicherstellen, dass deine API effizient und fehlerfrei bleibt.
* Es unterstützt **Python 3.6+**, einschließlich der neuesten Versionen, die die neuesten Funktionen und Verbesserungen bieten.

**Leichtgewichtig und skalierbar**

* FastAPI ist leichtgewichtig, sodass es keine unnötige Komplexität oder Overhead mit sich bringt, aber gleichzeitig auch in der Lage ist, komplexe Anwendungen und Skalierung zu bewältigen.
* FastAPI kann man für kleine Projekte bis hin zu großen, skalierbaren APIs verwenden.

**Open-Source und Community**

* FastAPI ist **Open-Source**, was bedeutet, dass du jederzeit den Quellcode einsehen und zur Weiterentwicklung beitragen kannst.
* Es gibt eine aktive Community, die regelmäßig neue Funktionen, Plugins und Erweiterungen entwickelt.

**Sehr gute Integration mit anderen Technologien**

* FastAPI lässt sich sehr gut mit anderen Tools und Frameworks kombinieren, wie zum Beispiel **SQLAlchemy** (für die Datenbankinteraktion), **Celery** (für Hintergrundaufgaben), **Redis** (für Caching und Messaging) und vielen weiteren.
* FastAPI kann man problemlos mit anderen modernen Python-Bibliotheken und Frameworks verwenden.

##### Installation KI-Modelle

Um im Anschluss die KI-Bibliotheken verwenden zu können, müssen diese zuvor installiert werden. Dazu wird ganz einfach der für Python entwickelte Package Manager „**PIP, packager installer python**“ verwendet, die Bibliotheken werden automatisch aus dem Internet geladen:

* **Scikit-Learn**

****

* **Pytorch**
  + **Ohne GPU-Untersützung (CPU-Only)**

****

* + **Mit GPU-Unterstützung**

****

* **TensorFlow**
  + **Ohne GPU-Untersützung (CPU-Only)**

****

* + **Mit GPU-Unterstützung**

****

**1. Scikit-Learn (scikit-learn)**

* **Was wird installiert?**
  + **Scikit-Learn** ist eine weit verbreitete Bibliothek für maschinelles Lernen in Python. Sie stellt eine Sammlung von Werkzeugen für viele gängige maschinelle Lernverfahren zur Verfügung, wie Klassifikation, Regression, Clustering und Dimensionalitätsreduktion.
* **Was wird durch die Installation bereitgestellt?**
  + Funktionen zur Modellbildung, wie **lineare Regression, Support Vector Machines (SVM), K-nearest Neighbors (KNN), Random Forest** und viele andere klassische ML-Algorithmen.
  + Funktionen zur **Datenvorverarbeitung** (z. B. Skalierung von Merkmalen, Umgang mit fehlenden Werten).
  + **Kreuzvalidierung** und **Hyperparameter-Tuning**.

**2. PyTorch (torch)**

* **Was wird installiert?**
  + **PyTorch** ist ein Deep-Learning-Framework, das es Entwicklern ermöglicht, neuronale Netze zu bauen und zu trainieren. Die Bibliothek enthält Funktionen zum Arbeiten mit **Tensors** (mehrdimensionalen Arrays) und zur **Automatischen Differenzierung** (für Backpropagation und Gradientenberechnung).
* **Was wird durch die Installation bereitgestellt?**
  + Grundlegende Funktionen für Deep Learning, einschließlich **Tensor-Operationen** (z. B. Addition, Multiplikation).
  + **Autograd**: Automatische Differenzierung, um Gradienten für das Training von neuronalen Netzen zu berechnen.
  + Unterstützung für **GPU-Computing** (durch CUDA) und **Datenparallele Verarbeitung**.
  + Flexibilität, um benutzerdefinierte **neuronale Netzwerke** zu erstellen und zu trainieren.

**3. torchvision**

* **Was wird installiert?**
  + **torchvision** ist eine Erweiterung von **PyTorch**, die speziell auf **Bildverarbeitung** fokussiert ist. Es enthält Tools und vortrainierte Modelle für Computer Vision Aufgaben wie Bildklassifikation, Objekterkennung und Segmentierung.
* **Was wird durch die Installation bereitgestellt?**
  + Vorverarbeitungsfunktionen für Bilder wie **Zuschneiden**, **Normalisierung** und **Skalierung**.
  + Vortrainierte **CNN-Modelle** (Convolutional Neural Networks), wie **ResNet**, **VGG** und **AlexNet**.
  + **Daten-Datasets** wie **CIFAR-10**, **ImageNet**, **COCO**.
  + Methoden zur **Datenaugmentation** (z. B. Bilddrehen, Spiegeln), um die Trainingsdaten zu erweitern.

**4. cudatoolkit**

* **Was wird installiert?**
  + **CUDA-Toolkit** ist eine Sammlung von Tools und Bibliotheken, die von **NVIDIA** entwickelt wurden, um die Leistung von GPU-Computing zu nutzen. Es stellt die Schnittstellen und Bibliotheken zur Verfügung, die erforderlich sind, damit Software wie **TensorFlow** oder **PyTorch** die **GPU** zur Berechnung von rechenintensiven Aufgaben (wie das Training von neuronalen Netzen) verwenden kann.
* **Was wird durch die Installation bereitgestellt?**
  + **CUDA-Bibliotheken** und Tools, die eine schnelle Verarbeitung durch die **NVIDIA-GPU** ermöglichen.
  + Optimierung und Beschleunigung von Deep-Learning-Modellen.
  + Verbindung zwischen der Software und der GPU für **parallelisierte Berechnungen**.
* **Hinweis**: Um GPU-Computing zu nutzen, muss man sicherstellen, dass die verwendete Grafikkarte und Treiber mit CUDA kompatibel sind.

**5. TensorFlow**

* **Was wird installiert?**
  + **TensorFlow** ist ein Open-Source-Deep-Learning-Framework von **Google**, das für maschinelles Lernen und neuronale Netzwerke entwickelt wurde. Es unterstützt sowohl **CPU** als auch **GPU** und eignet sich für **Forschung** und **Produktion**.
* **Was wird durch die Installation bereitgestellt?**
  + **Modelle und Tools** für maschinelles Lernen und Deep Learning, wie **Klassifikation**, **Regressionsmodelle**, **Textanalyse**, **Bildverarbeitung** und **Sprachverarbeitung**.
  + **TensorFlow Serving** und **TensorFlow Lite** für den **Produktionsbetrieb** (z. B. Bereitstellung auf Servern oder mobilen Geräten).
  + Optimierung der **Modell-Performance** (einschließlich GPU-Beschleunigung, **TensorFlow Extended (TFX)** für Produktionspipelines).

**6. tensorflow-gpu**

* **Was wird installiert?**
  + **tensorflow-gpu** ist die Version von **TensorFlow**, die **GPU-Unterstützung** integriert hat. Dies ermöglicht es, die **GPU** für das Training und die Berechnung von Deep-Learning-Modellen zu verwenden, was die Berechnungen deutlich beschleunigt.
* **Was wird durch die Installation bereitgestellt?**
  + **GPU-Unterstützung** für TensorFlow, was besonders bei großen Modellen und großen Datensätzen nützlich ist.
  + Die **CUDA**-Bibliotheken und Treiber werden mitinstalliert, sodass TensorFlow automatisch auf **NVIDIA-GPUs** zugreifen kann.
  + Schnelleres Training von Deep-Learning-Modellen im Vergleich zur **CPU-Version**, da die **GPU** für parallele Berechnungen verwendet wird.

##### Installation FastAPI

Um eine anständige Verbindung zum Frontend zu gewährleisten, wird FastAPI verwendet. Ebenso wie bei den Modellen wird dieses Modul über den Package Manager installiert:



Wichtig hierbei ist, dass FastAPI als Web-Framework auch einen ASGI-Server benötigt, um ausgeführt werden zu können. Uvicorn ist ein leichtgewichtiger ASGI-Server, der für FastAPI empfohlen wird:



**Erklärung ASGI:**

**ASGI (Asynchronous Server Gateway Interface)** ist eine Spezifikation für Web-Server und Python-Frameworks, die asynchrone Kommunikation unterstützt. Es ist die Weiterentwicklung des älteren WSGI (Web Server Gateway Interface), das die Grundlage für die meisten Python-Web-Frameworks bildet.

ASGI wurde entwickelt, um die Einschränkungen von WSGI zu überwinden und asynchrone Programmierung und parallele Verarbeitung von Anfragen zu ermöglichen. Dies ist besonders nützlich für Web-APIs, echte Echtzeit-Anwendungen wie Chat-Apps oder Spiele, sowie Anwendungen, die viele gleichzeitige Verbindungen benötigen.

##### Testexperiment

Am Anfang ist es wichtig, die Arbeit in diesem Bereich etwas kennenzulernen, ganz besonders wenn jemand noch nie zuvor damit in Berührung gekommen ist. Im Falle des DermaAI-Projektteams stellt die KI-Welt etwas Fremdes dar, daher wurde zu Beginn ein kleines Testexperiment mit Scikit-Learn implementiert, um den Umgang mit den Python-Bibliotheken zu verstehen.

Scikit-Learn stellt eine große Palette an unterschiedlichen KI-Algorithmen zur Verfügung:

Ein Bild, das Text, Screenshot, Grafikdesign, Design enthält.

Automatisch generierte Beschreibung

* + Es wird sich hier, wie schon erwähnt, auf die Classification-Algorithmen beschränkt

Anschließend wird ein Testskript in Python erstellt, um eine kleine Selektion der Scikit-Learn Algorithmen zu testen und deren Ergebnisse zu vergleichen (jegliche nicht definierte Funktionen sind aus dem Scikit-Learn-Framework oder anderen importiert):

**1. Classifier deklarieren**

Hier wird eine Liste von Klassifikatoren und ihre entsprechenden Namen erstellt. Diese Klassifikatoren werden später verwendet, um die trainierten Modelle zu evaluieren und die Entscheidungsgrenzen zu visualisieren.

Ein Bild, das Text, Screenshot, Schrift, Dokument enthält.

KI-generierte Inhalte können fehlerhaft sein.

* + **names** enthält die Namen der verschiedenen Klassifizierer, die im Code verwendet werden.
  + **classifiers** enthält die Instanzen der verschiedenen Klassifizierer aus sklearn

Diese Algorithmen sind auch Teil der fertigen DermaAI-Anwendung und werden im Kapitel XY mit ihren Parametern genauer erklärt.

**2. Datensätze generieren**

In diesem Schritt werden die Datensätze erstellt, die für das Training und Testen der Classifier verwendet werden. Die **make\_classification**, **make\_moons** und **make\_circles** Funktionen erzeugen synthetische Datensätze für Tests.

1. X, y = make\_classification(

2. n\_features=2, n\_redundant=0, n\_informative=2, random\_state=1, n\_clusters\_per\_class=1)

3. rng = np.random.RandomState(2)

4. X += 2 \* rng.uniform(size=X.shape)

5. linearly\_separable = (X, y)

6.

7. datasets = [

8. make\_moons(noise=0.3, random\_state=0),

9. make\_circles(noise=0.2, factor=0.5, random\_state=1),

10. linearly\_separable,

11. ]

12.

* + **make\_classification** erzeugt einen Datensatz für eine Klassifikationsaufgabe mit 2 informativen Features.
  + **make\_moons** und **make\_circles** erzeugen jeweils Datensätze, die für Klassifikationen von halbmond- oder kreisförmigen Daten verwendet werden.
  + **linearly\_separable** ist ein benutzerdefinierter Datensatz, der eine lineare Trennbarkeit aufweist.

**3. Erstellen einer Plot-Figur**

Hier wird die Plot-Figur mit einer bestimmten Größe erstellt, die später alle Subplots für die Datensätze und die Ergebnisse der Klassifikatoren enthalten wird.

1. figure = plt.figure(figsize=(27, 9))

2. i = 1

* + **plt.figure(figsize=(27, 9))** legt eine Abbildung fest, die Platz für Subplots bietet.
  + **i** ist der Index für die Subplots, der später bei der Zuordnung der Plot-Positionen verwendet wird:

**4. Plotten der Datensätze**

In diesem Schritt wird jeder der drei Datensätze geplottet, wobei die Trainings- und Testdaten durch unterschiedliche Farben unterschieden werden:

1. for ds\_cnt, ds in enumerate(datasets):

2. # preprocess dataset, split into training and test part

3. X, y = ds

4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(

5. X, y, test\_size=0.4, random\_state=42

6. )

7.

8. x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

9. y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

10.

11. cm = plt.cm.RdBu

12. cm\_bright = ListedColormap(["#FF0000", "#0000FF"])

13. ax = plt.subplot(len(datasets), len(classifiers) + 1, i)

14. if ds\_cnt == 0:

15. ax.set\_title("Input data")

16. ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k")

17. ax.scatter(

18. X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cm\_bright, alpha=0.6, edgecolors="k"

19. )

20. ax.set\_xlim(x\_min, x\_max)

21. ax.set\_ylim(y\_min, y\_max)

22. ax.set\_xticks(())

23. ax.set\_yticks(())

24. i += 1

* + **train\_test\_split** teilt die Daten in Trainings- und Testdaten (60% Training und 40% Test).
  + Die **scatter** Methode wird verwendet, um die Datenpunkte zu visualisieren.
  + **x\_min, x\_max, y\_min, y\_max** definieren den Bereich der x- und y-Achse, um sicherzustellen, dass alle Daten sichtbar sind.

**5. Trainieren und Auswerten der Klassifikatoren**

Für jeden Klassifikator wird ein Pipeline-Objekt erstellt, das den StandardScaler für die Datenvorverarbeitung und den jeweiligen Klassifikator beinhaltet. Der Klassifikator wird trainiert und auf den Testdaten evaluiert:

1. for name, clf in zip(names, classifiers):

2. ax = plt.subplot(len(datasets), len(classifiers) + 1, i)

3.

4. clf = make\_pipeline(StandardScaler(), clf)

5. clf.fit(X\_train, y\_train)

6. score = clf.score(X\_test, y\_test)

7. DecisionBoundaryDisplay.from\_estimator(

8. clf, X, cmap=cm, alpha=0.8, ax=ax, eps=0.5

9. )

10.

11. ax.scatter(

12. X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k"

13. )

14. ax.scatter(

15. X\_test[:, 0],

16. X\_test[:, 1],

17. c=y\_test,

18. cmap=cm\_bright,

19. edgecolors="k",

20. alpha=0.6,

21. )

22.

23. ax.set\_xlim(x\_min, x\_max)

24. ax.set\_ylim(y\_min, y\_max)

25. ax.set\_xticks(())

26. ax.set\_yticks(())

27. if ds\_cnt == 0:

28. ax.set\_title(name)

29. ax.text(

30. x\_max - 0.3,

31. y\_min + 0.3,

32. ("%.2f" % score).lstrip("0"),

33. size=15,

34. horizontalalignment="right",

35. )

36. i += 1

* + **make\_pipeline** erstellt eine Pipeline, die den **StandardScaler** zur Normalisierung der Eingabedaten und den Klassifikator selbst enthält.
  + **clf.fit(X\_train, y\_train)** trainiert den Klassifikator auf den Trainingsdaten.
  + **DecisionBoundaryDisplay.from\_estimator** zeigt die Entscheidungsgrenze des Klassifikators.
  + Die **ax.text** Methode zeigt die Genauigkeit des Klassifikators auf dem Plot an.

**6. Layout anpassen und Plot anzeigen**

1. plt.tight\_layout()

2. plt.show()

* + **plt.tight\_layout()** stellt sicher, dass alle Subplots ordentlich und ohne Überlappung angezeigt werden.
  + **plt.show()** zeigt den Plot auf dem Bildschirm an.

**Ergebnis:**

**Teil 1:**

**Ein Bild, das Screenshot, Kunst, Mosaik enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**Teil 2:**

**Ein Bild, das Screenshot, Mosaik, Kunst, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.**

Das gezeigte Bild stellt eine Visualisierung verschiedener Klassifikationsalgorithmen dar, die auf drei verschiedene Datensätze angewendet wurden. Jeder Algorithmus versucht, eine Entscheidungsgrenze zu lernen, um die beiden Klassen (rot und blau) korrekt zu trennen.

**1. Aufbau der Visualisierung**

Die Darstellung besteht aus einer **Matrix von Plots**, wobei:

* **Jede Zeile** einen anderen Datensatz repräsentiert (von oben nach unten: Mondförmige Daten (make\_moons), Kreisförmige Daten (make\_circles), Linear trennbare Daten (make\_classification)).
* **Jede Spalte** einen anderen Klassifikationsalgorithmus repräsentiert (von links nach rechts: Unterschiedliche Methoden wie k-Nächste Nachbarn, SVMs, Entscheidungsbäume, neuronale Netze etc.).
* Das erste Bild in jeder Zeile zeigt den **rohen Datensatz**, also die ursprünglichen Punkte ohne eine Klassifikation.
* Die übrigen Bilder zeigen die **Entscheidungsgrenzen**, die durch die jeweiligen Algorithmen gelernt wurden. Die eingefärbten Regionen repräsentieren die Vorhersagen des jeweiligen Klassifikators:
  + **Rote Regionen** → Klassifikation als Klasse 1 (rot).
  + **Blaue Regionen** → Klassifikation als Klasse 2 (blau).
  + Die Zahlen in den Diagrammen repräsentieren die Genauigkeit (Accuracy) des jeweiligen Klassifikators auf dem Testdatensatz.

**2. Analyse der Algorithmen und Entscheidungsgrenzen**

**Nearest Neighbors (K-Nearest Neighbors, KNN)**

* **Charakteristik:** KNN basiert darauf, dass ein Punkt die gleiche Klasse wie seine **nächsten Nachbarn** annimmt.
* **Entscheidungsgrenze:** Sehr flexibel und passt sich stark an die Struktur der Daten an, daher eher **unregelmäßige Grenzen**.
* **Stärken:** Gute Leistung bei komplexen, nicht-linearen Datensätzen.
* **Schwächen:** Kann übermäßig an die Trainingsdaten angepasst sein (Overfitting), besonders bei verrauschten Daten.

**Linear SVM (Support Vector Machine mit linearem Kernel)**

* **Charakteristik:** Versucht, eine **lineare Trennlinie** zu finden, die die beiden Klassen optimal trennt.
* **Entscheidungsgrenze:** Gerade Linie – sehr gut für den linearen Datensatz (unterste Zeile), aber **ungeeignet für nicht-lineare Daten** (Mond- und Kreisförmige Daten).
* **Stärken:** Sehr gut für **linear separierbare Daten**.
* **Schwächen:** Funktioniert nicht für komplexe, nicht-lineare Strukturen.

**RBF SVM (Support Vector Machine mit radialem Basisfunktions-Kernel)**

* **Charakteristik:** Erweitert die lineare SVM, indem sie einen nicht-linearen Kernel verwendet, der die Daten in eine höhere Dimension transformiert.
* **Entscheidungsgrenze:** Sehr geschwungene, **komplexe Entscheidungsmuster**, die sich gut an nicht-lineare Strukturen anpassen.
* **Stärken:** Sehr leistungsfähig für nicht-lineare Probleme (wie hier für make\_moons und make\_circles).
* **Schwächen:** Erfordert sorgfältige Wahl der Hyperparameter (gamma, C).

**Gaussian Process Classifier (GPC)**

* **Charakteristik:** Probabilistischer Ansatz, der Wahrscheinlichkeitsverteilungen über Funktionen modelliert.
* **Entscheidungsgrenze:** Sehr **glatte, weiche Trennungen**, besonders sichtbar bei den kreisförmigen Daten.
* **Stärken:** Liefert nicht nur eine Klassifikation, sondern auch eine **Unsicherheitsabschätzung**.
* **Schwächen:** Hoher Rechenaufwand für große Datensätze.

**Decision Tree (Entscheidungsbaum)**

* **Charakteristik:** Hierarchische Struktur, die schrittweise die Daten in Bereiche unterteilt.
* **Entscheidungsgrenze:** **Stufenförmige, eckige Trennungen**, da Entscheidungsbäume durch einfache Regeln arbeiten.
* **Stärken:** Schnell zu trainieren, interpretiert leicht.
* **Schwächen:** Neigt zum Overfitting, besonders bei kleinen Datenmengen.

**Random Forest (Ensemble von Entscheidungsbäumen)**

* **Charakteristik:** Kombiniert mehrere Entscheidungsbäume, um eine robustere Klassifikation zu ermöglichen.
* **Entscheidungsgrenze:** Ähnlich wie Decision Trees, aber **glatter und stabiler**, weil mehrere Bäume kombiniert werden.
* **Stärken:** Reduziert Overfitting, robuste Performance.
* **Schwächen:** Weniger interpretierbar als ein einzelner Entscheidungsbaum.

**Neural Network (Mehrschichtige neuronale Netze, MLPClassifier)**

* **Charakteristik:** Mehrere Schichten von Neuronen lernen **hochgradig nicht-lineare Entscheidungsgrenzen**.
* **Entscheidungsgrenze:** Sehr **geschwungene, flexible Grenzen**, besonders sichtbar bei make\_moons und make\_circles.
* **Stärken:** Kann sehr komplexe Muster lernen.
* **Schwächen:** Erfordert viel Rechenleistung und Daten für effektives Training.

**AdaBoost (Adaptive Boosting)**

* **Charakteristik:** Ensemble-Methode, die schwache Klassifikatoren kombiniert, um eine starke Gesamtentscheidung zu treffen.
* **Entscheidungsgrenze:** Etwas eckige Trennung, aber deutlich robuster als ein einzelner Entscheidungsbaum.
* **Stärken:** Verbessert schwache Klassifikatoren und funktioniert gut auf moderaten Datensätzen.
* **Schwächen:** Sensitiv gegenüber Ausreißern.

**Naive Bayes (Gaussian Naive Bayes)**

* **Charakteristik:** Basierend auf **Bayes‘ Theorem**, nimmt an, dass Merkmale unabhängig sind.
* **Entscheidungsgrenze:** Glatte, oft ovale oder kreisförmige Trennung.
* **Stärken:** Schnell zu trainieren, funktioniert oft überraschend gut.
* **Schwächen:** Die Annahme der **Merkmalsunabhängigkeit** ist oft nicht realistisch.

**QDA (Quadratic Discriminant Analysis)**

* **Charakteristik:** Erweiterung von Naive Bayes, erlaubt **quadratische** Trennungen.
* **Entscheidungsgrenze:** Starke, geschwungene Muster (besonders sichtbar in der letzten Spalte).
* **Stärken:** Erlaubt nicht-lineare Trennungen.
* **Schwächen:** Funktioniert nicht gut, wenn die Daten nicht den zugrundeliegenden Modellannahmen entsprechen.

**3. Fazit und Interpretation der Ergebnisse**

* **Die besten Algorithmen für nicht-lineare Daten (make\_moons, make\_circles)** sind **RBF SVM, Gaussian Process, Random Forest und Neural Networks**, da sie geschwungene Entscheidungsgrenzen erzeugen.
* **Lineare SVM funktioniert nur für lineare Datensätze** und ist für make\_moons und make\_circles nicht geeignet.
* **Decision Trees und Random Forests neigen zu kantigen Entscheidungsgrenzen**, funktionieren aber trotzdem gut.
* **Probabilistische Methoden wie Naive Bayes und QDA** zeigen eher weiche, geschwungene Entscheidungsgrenzen, sind aber nicht immer die besten.

Die Wahl des besten Klassifikators hängt also stark von der Datenstruktur ab. Während lineare Modelle für einfache Trennungen gut sind, sind komplexe, nicht-lineare Modelle für anspruchsvollere Muster nötig.

## Variantenbildung

## Machbarkeitsstudie

## Projektziele

## Benötigte Ressourcen

## Entwicklungsmethodik

## Kommunikations- und Berichterstattungsstrategie

## Softwarearchitektur

## Aktivitätsdiagramme

## Sequenzdiagramme

## Komponentendiagramme

## Verteilungsdiagramme

## Softwarediagramme / Komponenten

## SW Programme

## SW Komponenten

# Projektdurchführung

## Daniel Jessner (KI & Python)

# Projektabschluss

## Installation / Software deployment

# Literaturverzeichnis

# Abbildungsverzeichnis