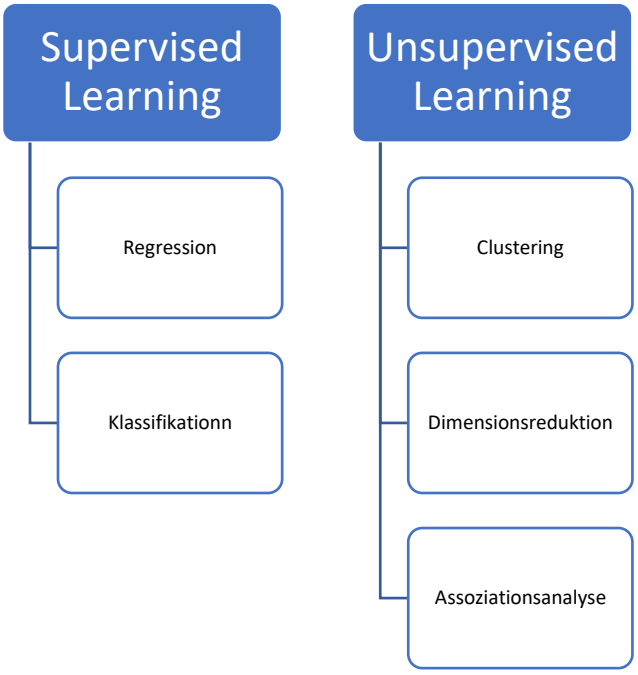
Machine Learning

- **Supervised Learning** (Daten gelabelt)
- **Unsupervised Learning** (Daten ungelabelt)
- **Reinforcement Learning** (bestärkendes Lernen)



Regression

- **Lineare Regression** (lineare Beziehung zwischen einer abhängigen Variablen und einer oder mehreren unabhängigen Variablen: y=mx+b)
- **Polynomiale Regression** (Polynom höheren Grades, anstatt lineare Funktion)
- **Ridge Regression** (Overfitting soll reduziert werden, indem eine Regularisierungsterm zur Verlustfunktion hinzugefügt wird, L2-Regularisierung)
- **Lasso Regression** (Ähnlich wie Ridge-Regression dient Lasso-Regression dazu, Overfitting zu reduzieren, indem eine Regularisierungsterm zur Verlustfunktion hinzugefügt wird. Allerdings verwendet Lasso eine L1-Regularisierung, was dazu führen kann, dass einige der Koeffizienten auf Null gesetzt werden, was eine Art von Feature-Auswahl darstellt.)
- **Elastic Net Regression** (Kombination aus Ridge- und Lasso-Regression und verwendet sowohl L1- als auch L2-Regularisierung, um eine ausgewogene Regularisierung zu erzielen.)
- **Support Vector Regression (SVR)** (soll Hyperplane finden, die möglichst viele Trainingsdatenpunkte innerhalb einer festgelegten Toleranzgrenze umfasst)
- **Decision Tree Regression**
- **Random Forest Regression** (Bagging Ansatz => Bootstrap Aggregating, zufällige Entscheidungsbäume werden parallel trainiert und mittels Mehrheitsentscheidung das beste Modell ausgewählt). Gini impurity bewertet die Qualität der Trennung (wie "rein" sind die durch die Trennung entstandenen Teilmengen in Bezug auf die Zielvariable).
- **Gradient Boosting Regression** (Entscheidungsbaum wird sequentiell nacheinander trainiert, indem das Modell immer weiter verbessert wird.)

Metriken für Regression

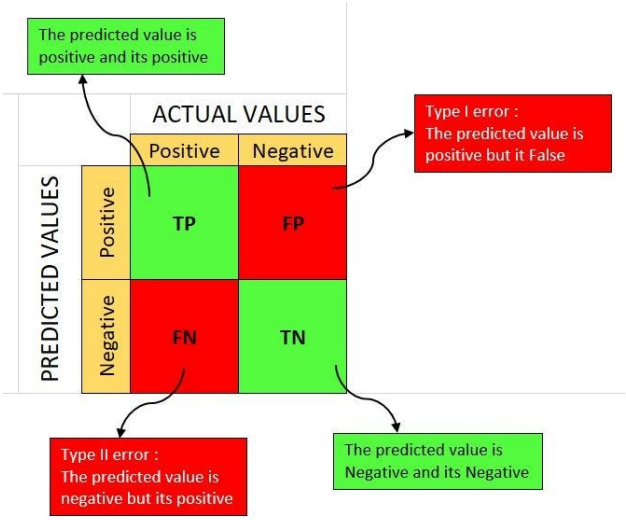
- **Mean Absolute Error (MAE)** (durchschnittliche absolute Abweichung zwischen den Vorhersagen des Modells und den tatsächlichen Werten)
- **Mean Squared Error (MSE)** (durchschnittliche quadratische Abstand zwischen den Vorhersagen des Modells und den tatsächlichen Werten.)
- **Root Mean Squared Error (RMSE)** (Die Wurzel des mittleren quadratischen Fehlers ist einfach die Quadratwurzel des MSE.)
- **Mean Absolute Percentage Error (MAPE)** (gibt die durchschnittliche prozentuale Abweichung zwischen den Vorhersagen des Modells und den tatsächlichen Werten an.)
- **R^2-Score (Bestimmtheitsmaß)** (gibt Auskunft darüber, wie gut die abhängige Variable mit den betrachteten unabhängigen Variablen vorhersagt werden kann.)
- **Residual Sum of Squares (RSS)** (quantifiziert die Diskrepanz zwischen den beobachteten Werten und den durch das Modell vorhergesagten Werten, indem die vorhergesagten Werte von den beobachten Werten subtrahiert und quadriert und anschließend addiert werden)

Klassifikation

- **Logistische Regression** (Im Gegensatz zur linearen Regression, die dazu verwendet wird, eine kontinuierliche Zielvariable vorherzusagen, modelliert die logistische Regression die Wahrscheinlichkeit, dass eine Beobachtung einer bestimmten Klasse angehört; Sigmoid-Funktion: 1/(1+e^-x).)
- **Decision Tree** (Entscheidungsbaum)
- **Random Forest** (zufällige Entscheidungsbäume)
- **Support Vector Machines (SVM)** (optimale Trennung zwischen Klassen, indem eine oder mehrere Trennflächen (Hyperebenen) im Merkmalsraum identifiziert werden)
- **K-Nearest Neighbors (KNN)** (wähle K, dann wird ein neuer Datenpunkt der Klasse zugeordnet, die unter seinen K nächsten Nachbarn am häufigsten vorkommt.)
- **Naive Bayes** (basiert auf dem Bayes'schen Theorem: bedingte Wahrscheinlichkeit => wie wahrscheinlich ist ein Ereignis unter der Bedingung eines anderen Ereignisses)

Metriken für Klassifikation

- **Confusion Matrix**
  from sklearn.metrics import confusion_matrix

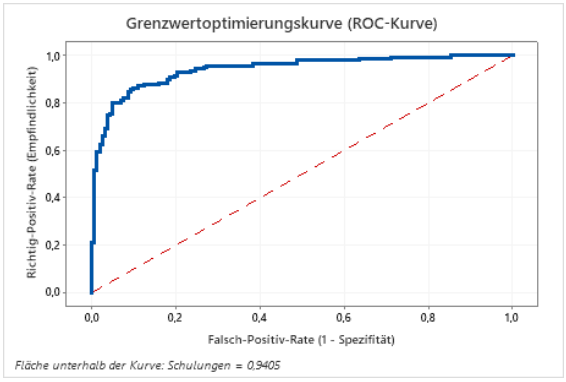


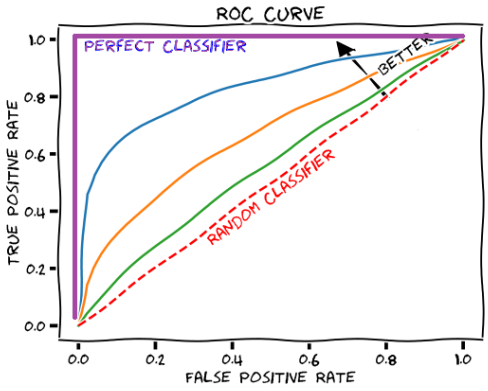
- **Genauigkeit (Accuracy)** (Anzahl korrekter Vorhersagen/ Gesamtanzahl der Vorhersagen => (TP+TN)/(TP+TN+FP+FN))
- **Präzision (Precision)** (Anzahl korrekter positiver Vorhersagen/ Anzahl aller positiven Vorhersagen z.B. 100 positive Vorhersagen, davon 85 korrekt: 85/100=85% z.B. Spam-Erkennung => TP/(TP+FP))
- **Recall (Sensitivität)** (Wahrscheinlichkeit, dass ein Test eine Person, die die Krankheit hat, richtig diagnostiziert; Anzahl korrekter positiver Vorhersagen/ Anzahl aller tatsächlich positiven Fälle z.B. 100 haben die Krankheit, das Modell identifiziert 80 Pateienten als krank, von denen 70 tatsächlich erkrankt sind: 70/100=70%, z.B. Medizinische Diagnose =>TP/(TP+FN))
- **F1-Score** (harmonische Mittel aus Präzision und Recall; 2 * ((Precision * Recall) / (Precision + Recall))

from sklearn.metrics import classification_report

	precision	recall	f1-score	support
0	0.77	0.86	0.81	37584
1	0.84	0.75	0.79	37577
accuracy			0.80	75161
macro avg	0.81	0.80	0.80	75161
weighted avg	0.81	0.80	0.80	75161

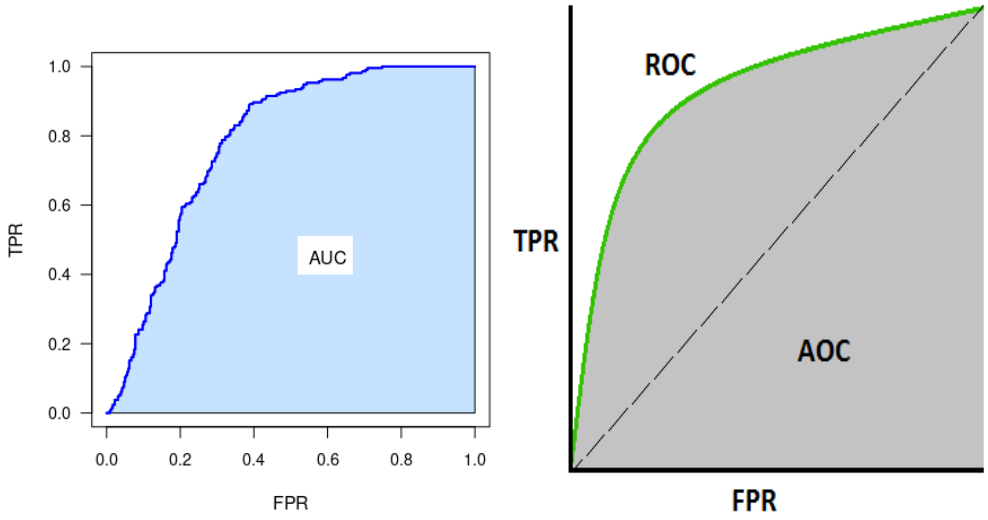
- **Spezifität** (Wahrscheinlichkeit, dass ein Test eine Person, die nicht erkrankt ist, korrekt erkennt; misst die Fähigkeit Modells, alle tatsächlich negativen Fälle zu identifizieren => tn / (tn + fp))
- **ROC-Kurve (Receiver Operating Characteristic)**





Sie zeigt das Verhältnis zwischen der wahren positiven Rate TPR (Recall) und der falsch positiven Rate FPR (FP / (FP + TN))
from sklearn.metrics import roc_curve, roc_auc_score

- **AUC (Area Under the Curve)**



Die AUC misst die gesamte Fläche unter der ROC-Kurve. Sie variiert zwischen 0 und 1
Eine AUC von 1 bedeutet, dass der Klassifikator perfekt ist. Das heißt, er erreicht eine TPR von 1 (keine falsch negativen Ergebnisse) und eine FPR von 0 (keine falsch positiven Ergebnisse). Dies würde bedeuten, dass der Klassifikator die positiven und negativen Fälle vollständig voneinander unterscheiden kann.

scikit-learn Beispiele supervised Learning

Klassifikation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
#1.  Daten laden
df=pd.read_csv("data.csv")
X = df.drop(columns =["name","is_legendary"])
y = df["is_legendary"]
#2. Trainings- und Testdaten aufteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
#3.  Feature-Skalierung auf X_train und X_test
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
#4. k-NN Klassifikator erstellen und trainieren
knn = KNeighborsClassifier(n_neighbors=3)
#5. Fitting auf den Trainings Daten
knn.fit(X_train_scaled, y_train)
#6. Vorhersagen machen auf X_test
predictions = knn.predict(X_test_scaled)
#7.  Genauigkeit berechnen mittels y_test
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

Regression

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
#1. Daten laden
boston = load_boston()
X, y = boston.data, boston.target
#2. Trainings- und Testdaten aufteilen
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
#3. Feature-Skalierung auf X_train und X_test
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
#4. Lineare Regression erstellen und trainieren
lr = LinearRegression()
#5. Fitten auf Trainingsdaten
lr.fit(X_train_scaled, y_train)
#6. Vorhersagen machen auf X_test
predictions = lr.predict(X_test_scaled)
#7. Mittlere quadratische Abweichung berechnen
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

Skalierung

- **StandardScaler:** Skaliert die Features, so dass sie eine mittlere von 0 und eine Standardabweichung von 1 haben. Bringt die Daten auf eine Standardnormalverteilung, wobei der Mittelwert 0 und die Standardabweichung 1 beträgt. Es ist robust gegenüber Ausreißern in den Daten.
- **MinMaxScaler**: Skaliert die Features in einen festgelegten Bereich, normalerweise zwischen 0 und 1 (es sei denn, Sie legen andere Grenzen fest). Wandelt die Daten linear so um, dass der kleinste Wert 0 und der größte Wert 1 ist. Erhält die Struktur der ursprünglichen Daten und ist daher besonders geeignet, wenn Sie wissen, dass Ihre Daten auf einen bestimmten Bereich begrenzt sind.

Cross Validation

Hyperparameter-Optimierung

- **GridSearchCV**
- **Random Search**
- **Bayesian Optimization**

Clustering

- **K-Mean** (Datenpunkte werden in k Cluster gruppiert, wobei jedes Cluster durch seinen Zentroiden repräsentiert wird. Es versucht, die Daten so aufzuteilen, dass die Summe der quadratischen Abstände der Datenpunkte zu ihren jeweiligen Zentroiden minimiert wird.)
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** (dichtebasierter Clustering-Algorithmus, der Cluster anhand der Dichte der Datenpunkte im Raum identifiziert.)

Dimensionsreduktion

- **Principal Component Analysis (PCA)** (nutzt die Korrelation zwischen den Merkmalen, um neue, orthogonal zueinanderstehende Dimensionen (Hauptkomponenten) zu erstellen. Diese neuen Dimensionen werden so angeordnet, dass die erste Hauptkomponente die größte Varianz im Datensatz erklärt, die zweite die zweitgrößte Varianz und so weiter.)
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** (Im Wesentlichen wandelt t-SNE hochdimensionale Daten in eine zweidimensionale oder dreidimensionale Darstellung um, die leichter interpretiert und visualisiert werden kann.)

Assoziationsanalyse

- **Assoziationsanalyse** (Er identifiziert häufige Muster in Daten, indem er alle möglichen Kombinationen von Variablen durchsucht und unterstützende Regeln extrahiert. Diese Regeln haben typischerweise die Form "Wenn A, dann B".)
- **Eclat (Equivalence Class Transformation)** (Idee der äquivalenten Klassenbildung)
- **FP-Growth (Frequent Pattern Growth)** (verwendet eine spezielle Datenstruktur namens FP-Baum, um häufige Muster in den Daten zu identifizieren)

NLP

- **Tokenization** (Aufteilung eines Textes in einzelne Wörter oder Sätze)
- **Stemming und Lemmatization** (soll Grundformen von Wörtern finden, indem sie Suffixe entfernen (Stemming) oder Wörter in ihre lexikalische Grundform umwandeln (Lemmatization)
- **Part-of-Speech Tagging** (Prozess der Zuordnung von grammatikalischen Tags wie Nomen, Verben, Adjektiven usw. zu den Wörtern in einem Text)
- **Named Entity Recognition (NER)** (Identifizierung und Klassifizierung von benannten Entitäten wie Personen, Orten, Organisationen usw. in einem Text)
- **Sentiment Analysis** (automatische Erkennung und Klassifizierung von Stimmungen oder Emotionen in einem Text)
- **Topic Modeling** (Themen oder verborgene Strukturen in einem Textkorpus sollen damit identifiziert werden. Ein bekannter Algorithmus dafür ist Latent Dirichlet Allocation (LDA).)
- **Word Embeddings** (Word Embeddings sind Vektordarstellungen von Wörtern in einem Vektorraum, die semantische Ähnlichkeiten zwischen Wörtern erfassen)

Bekannte NLP Bibliotheken in Python NLTK (Natural Language Toolkit), spaCy, TextBlob, Gensim, Transformers (Hugging Face)

Time Series

- **ARIMA (Autoregressive Integrated Moving Average)**
- **SARIMA (Seasonal ARIMA)**
- **Prophet**