Machine learning scope

Theory

* Supervised learning: Classification and Regression

Supervised Learning model – is a model trained using labelled data, meaning every input is associated with output feature. Every observation in a training dataset must have an input and output object.

Classification – is a type supervised learning where the goal is to predict categorical outcome based on input features. It categorised input into discrete labels and it’s widely used automated decision-making, medical diagnosis, and fraud detection.

Regression - is a supervised learning technique specifically used for predicting continuous numerical outcomes. In regression, the model learns a mapping from a set of features to a continuous output.

* Overfitting and Underfitting and ways to mitigate these problems

Overfitting – The model learns noise instead of the underlying pattern, performing well on training data but poorly on new data.

* + Increase Training Data
  + Reduce Model Complexity
  + Regularization
  + Cross-Validation

Underfitting – The model is too simple to capture patterns in the data, leading to poor performance even on the training set.

* + Increase Model Complexity
  + Feature Engineering
  + Reduce Regularization
  + Increase Training Duration
* Machine learning libraries

NumPy is one of the fundamental packages for scientific computing in Python. It contains functionality for multidimensional arrays, high-level mathematical functions such as linear algebra operations and the Fourier transform, and pseudorandom number generators.

Pandas Pandas introduce additional data structures for managing datasets in Python. Its primary data structure is the DataFrame.

Scikit-learn

* Correctness 1.5 of study guide
* The Bias-Variance Trade-off 1.6 of study guide
* Median and mean imputation

Practical

* How to do standardization

**from sklearn.datasets import fetch\_california\_housing**

**from sklearn.preprocessing import StandardScaler**

**import pandas as pd**

**# Load Califonia Housing dataset**

**boston = pd.read\_csv("BostonHousing.csv")**

**boston.head()**

**# Seperating features and targets**

**X = boston.drop(columns=['medv']) # medv is a target variable**

**y = boston['medv']**

**# Standardize features**

**scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

**# Convert back to DataFrame for readability**

**X\_scaled\_df = pd.DataFrame(X\_scaled, columns=X.columns)**

**X\_scaled\_df.head()**

* Calculating performance metrics

**import pandas as pd**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import mean\_squared\_error, r2\_score**

**diabetes = pd.read\_csv("diabetes.csv")**

**diabetes.head()**

**# seperate features and targets**

**X = diabetes.drop(columns=['Outcome'])**

**y = diabetes.Outcome**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Train model**

**model = LinearRegression()**

**model.fit(X\_train, y\_train)**

**# Predict**

**y\_pred = model.predict(X\_test)**

**# Evaluate**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**r2 = r2\_score(y\_test, y\_pred)**

**print(f"Mean Square Error (MSE): {mse}")**

**print(f"R^2 Score: {r2}")**

* Step on how to train linear regression (e.g. load the houses data in a dataFrame, applying label-encoding, split the dataset, etc.)

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import mean\_squared\_error, r2\_score**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.preprocessing import LabelEncoder, StandardScaler**

**# Load dataset**

**data = pd.read\_csv("CaliforniaHousing.csv")**

**# Handling missing values**

**data = data.dropna()**

**# Label encode categorical features**

**le = LabelEncoder()**

**data['ocean\_proximity'] = le.fit\_transform(data['ocean\_proximity'])**

**# Seperate feature and target**

**X = data.drop(columns=['median\_house\_value'])**

**y = data.median\_house\_value**

**# Standardize the features**

**scaler = StandardScaler()**

**X\_scaled = scaler.fit\_transform(X)**

**# Split the data into train/test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)**

**# Train the Linear Regression Model**

**model = LinearRegression()**

**model.fit(X\_train, y\_train)**

**# Make prediction**

**y\_pred = model.predict(X\_test)**

**# Evaluate performance**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**r2 = r2\_score(y\_test, y\_pred)**

**print(f"Mean Squared Error: {mse}")**

**print(f"r2 Score: {r2}")**

**plt.figure(figsize=(8, 6))**

**plt.scatter(y\_test, y\_pred, alpha=0.5, color="teal")**

**plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linestyle='--')**

**plt.xlabel("Actual Median House Value")**

**plt.ylabel("Predicted Median House Value")**

**plt.title("Actual vs Predicted Values (Linear Regression)")**

**plt.grid(True)**

**plt.tight\_layout()**

**plt.show()**

* k-means clustering algorithm

**import pandas as pd**

**from sklearn.cluster import KMeans**

**import matplotlib.pyplot as plt**

**data = pd.read\_csv("iris.csv")**

**data.head()**

**# Seperate features and targets**

**X = data.drop(columns=['Species'])**

**kmeans = KMeans(n\_clusters=3, random\_state=42)**

**data['cluster'] = kmeans.fit\_predict(X)**

**# Select features for clustering**

**X = data[["SepalLengthCm", "SepalWidthCm"]]**

**# Apply K-Means**

**kmeans = KMeans(n\_clusters=3, random\_state=42)**

**data["cluster"] = kmeans.fit\_predict(X)**

**# Plot clusters**

**plt.figure(figsize=(8, 6))**

**for cluster in range(3):**

**clustered\_data = data[data["cluster"] == cluster]**

**plt.scatter(clustered\_data["SepalLengthCm"], clustered\_data["SepalWidthCm"], label=f"Cluster {cluster}")**

**# Plot centroids**

**centroids = kmeans.cluster\_centers\_**

**plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='black', marker='X', label='Centroids')**

**plt.xlabel("Sepal Length")**

**plt.ylabel("Sepal Width")**

**plt.title("K-Means Clustering of Iris Dataset")**

**plt.legend()**

**plt.grid(True)**

**plt.show()**

* Python code to Train the decision tree and logistic regression models for making predictions

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.datasets import load\_breast\_cancer**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score**

**import pandas as pd**

**cancer\_data = pd.read\_csv("BreastCancer.csv")**

**X = cancer\_data.drop(columns=['Class'])**

**y = cancer\_data["Class"]**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**tree\_model = DecisionTreeClassifier(random\_state=0)**

**tree\_model.fit(X\_train, y\_train)**

**print(f"Accuracy on training set: {tree\_model.score(X\_train, y\_train):.3f}")**

**print(f"Accuracy on test set: {tree\_model.score(X\_test, y\_test):.3f}")**

**X, y = load\_breast\_cancer(return\_X\_y=True)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**dt =DecisionTreeClassifier()**

**dt.fit(X\_train, y\_train)**

**dt\_pred = dt.predict(X\_test)**

**lr = LogisticRegression(max\_iter=10000)**

**lr.fit(X\_train, y\_train)**

**lr\_pred = lr.predict(X\_test)**

**print(f"Decision Tree Accuarcy: {accuracy\_score(y\_test, dt\_pred):.3f}")**

**print(f"Linear Regression Accuarcy: {accuracy\_score(y\_test, lr\_pred):.3f}")**