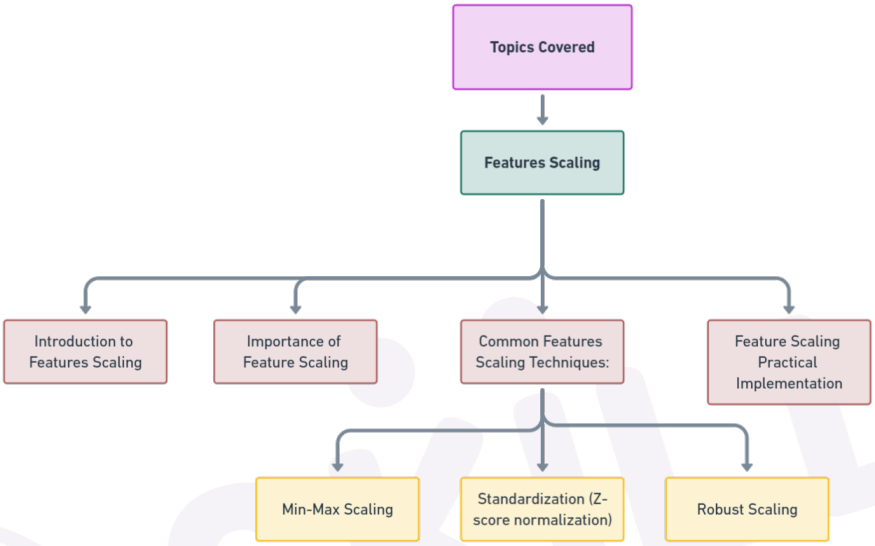
PW SKILLS

Lesson Plan

Feature Scaling

Topic to covered:

- Introduction to Features Scaling
- Importance of Feature Scaling
- Common Features Scaling Techniques:
  - Min-Max Scaling
  - Standardization (Z-score normalization)
  - Robust Scaling
- Feature Scaling Practical Implementation



Introduction to Feature Scaling

- In the context of machine learning, features refer to the input variables or attributes used to make predictions.
- Examples of features could include height, weight, temperature, etc., depending on the problem.
- Features are usually represented in the form of a feature matrix, where each row corresponds to an instance or data point, and each column corresponds to a specific feature.
- Feature scaling is the process of normalizing or standardizing the range of independent variables or features of the data.
- The goal is to ensure that no particular feature dominates others, preventing biased model training.

Importance of Feature Scaling

- Many machine learning algorithms are sensitive to the scale of features.
- Algorithms like k-nearest neighbors, support vector machines, and neural networks can be influenced by the magnitude of features.

- Feature scaling ensures that all features contribute equally to the distance computations or optimization processes during model training.
- Without scaling, features with larger scales may overshadow smaller ones.
- Feature scaling aids in the faster convergence of gradient-based optimization algorithms.
- Algorithms like gradient descent converge more quickly when features are on a similar scale.
- Scaling can lead to improved model performance, as it reduces the chances of models being misled by features with larger scales.
- Models may become more accurate and generalize better to unseen data.

Common Features Scaling Techniques

Min-Max Scaling:

- Min-Max Scaling, also known as Min-Max normalization or Rescaling, transforms the features by scaling them to a specific range, usually [0, 1].
- Formula:

$$X_{\text{new}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

- Apply the formula to scale the feature to the desired range.
- Pros:
  - Simple and intuitive.
  - Preserves the relationships between data points.
- Cons:
  - Sensitive to outliers.
  - Might not work well if the minimum and maximum values are not representative of the overall data.

• Standardization (Z-score normalization):

- Standardization transforms features to have a mean (μ) of 0 and a standard deviation (σ) of 1.
- Formula:

$$\overline{X_{\text{new}}} = \frac{X - \mu}{\sigma}$$

- Calculate the mean (μ) and standard deviation (σ) of the feature.
- Apply the formula to standardize the feature.

- **Pros:**
  - Robust to outliers.
  - Suitable for algorithms that assume normally distributed data.

- **Cons:**
  - Does not bound the data to a specific range.
  - Changes the interpretation of the data (values are in terms of standard deviations)

- **Robust Scaling**
  - Robust Scaling, or Robust Standardization, is a technique that scales features based on the median and interquartile range (IQR), making it resistant to outliers.
  - Formula:

$$X_{\text{new}} = \frac{X - \text{median}}{\text{IQR}}$$

- Calculate the median and IQR of the feature.
- Apply the formula to robustly scale the feature.
- Pros:
  - Robust to outliers.
  - Preserves the shape of the data distribution.
- Cons:
  - Does not produce values in a fixed range.
  - Sensitivity to the distribution of the data.

Practical Implementation Of Feature Scaling

Code:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
RobustScaler

# Load the Iris dataset
iris = datasets.load_iris()
data = pd.DataFrame(data= np.c_[iris['data'], iris['target']], columns=
iris['feature_names'] + ['target'])

# Display the first few rows of the dataset
print("Original Iris Dataset:")
print(data.head())
# Extract features and target variable
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Apply Min-Max Scaling
```

```
minmax_scaler = MinMaxScaler()
X_train_minmax_scaled = minmax_scaler.fit_transform(X_train)
X_test_minmax_scaled = minmax_scaler.transform(X_test)

# Apply Standardization (Z-score normalization)
standard_scaler = StandardScaler()
X_train_standardized = standard_scaler.fit_transform(X_train)
X_test_standardized = standard_scaler.transform(X_test)

# Apply Robust Scaling
robust_scaler = RobustScaler()
X_train_robust_scaled = robust_scaler.fit_transform(X_train)
X_test_robust_scaled = robust_scaler.transform(X_test)
# Display the scaled datasets
print("\nMin-Max Scaled Training Data:")
print(pd.DataFrame(X_train_minmax_scaled, columns=X_train.columns).head())

print("\nStandardized Training Data:")
print(pd.DataFrame(X_train_standardized, columns=X_train.columns).head())

print("\nRobust Scaled Training Data:")
print(pd.DataFrame(X_train_robust_scaled, columns=X_train.columns).head())
```

Output:

Min-Max Scaled Training Data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.088235	0.666667	0.000000	0.041667
1	0.411765	1.000000	0.087719	0.125000
2	0.705882	0.458333	0.596491	0.541667
3	0.147059	0.583333	0.105263	0.041667
4	0.029412	0.500000	0.052632	0.041667

Standardized Training Data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-1.473937	1.203658	-1.562535	-1.312603
1	-0.133071	2.992376	-1.276006	-1.045633
2	1.085898	0.085709	0.385858	0.289218
3	-1.230143	0.756479	-1.218701	-1.312603
4	-1.717731	0.309299	-1.390618	-1.312603

Robust Scaled Training Data:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.884615	1.000000	-0.902778	-0.733333
1	-0.038462	2.333333	-0.763889	-0.600000
2	0.730769	0.166667	0.041667	0.066667
3	-0.730769	0.666667	-0.736111	-0.733333
4	-1.038462	0.333333	-0.819444	-0.733333