Machine learning

Validation in Machine Learning

- Validation in ML is the process of *evaluating a model's performance on unseen data before final deployment*.
- It ensures the model generalizes well to new inputs and helps in selecting the best model while avoiding overfitting or underfitting.

Why Validation Important?

- **Prevents Overfitting:** Ensures the model does not memorize training data but learns meaningful patterns.
- **Hyperparameter Tuning:** Helps in selecting the best model parameters.
- **Improves Generalization:** Ensures the model performs well on new, unseen data.

Types of Validation Techniques

1. Holdout Validation

- Splits the dataset into three parts:
  - **Training Set** (e.g., 70%) → Used to train the model.
  - **Validation Set** (e.g., 15%) → Used to tune hyperparameters.
  - **Test Set** (e.g., 15%) → Used to evaluate final model performance.
- **Pros:** Simple and fast.
- **Cons:** Performance may depend on how the data is split.

2. k-Fold Cross-Validation (CV)

- Splits data into **k** subsets (folds).
- Trains the model on **(k-1)** folds and tests it on the remaining fold.
- Repeats **k** times, averaging results for better evaluation.
- **Common values of k:** 5, 10

Example (5-Fold CV):

1. Train on folds [1,2,3,4], test on fold [5].
2. Train on folds [1,2,3,5], test on fold [4].
3. … Repeat until all folds are tested.

3.Stratified k-Fold Cross-Validation

- Ensures class distribution is maintained in each fold (useful for imbalanced datasets).
- Example: In a binary classification problem, each fold has an equal proportion of positive and negative samples.

Stratified k-Fold Cross-Validation(Example)

Data: [A, A, A, A, A, B, B, B, B, B, C, C, C, C, C]

Each class A,B,C - **5 samples** **Divide Each Class into 3 Folds (k=3)**

- Split **A**: [A, A] (Fold 1), [A, A] (Fold 2), [A] (Fold 3)
- Split **B**: [B, B] (Fold 1), [B, B] (Fold 2), [B] (Fold 3)
- Split **C**: [C, C] (Fold 1), [C, C] (Fold 2), [C] (Fold 3)

Construct Folds

- **Fold 1** → [A, A, B, B, C, C] **Fold 2** → [A, A, B, B, C, C] **Fold 3** → [A, B, C]

Train & Test Rotations

- **Iteration 1**: Train on Fold 2 & 3, Test on Fold 1
- **Iteration 2**: Train on Fold 1 & 3, Test on Fold 2
- **Iteration 3**: Train on Fold 1 & 2, Test on Fold 3

4. Leave-One-Out Cross-Validation (LOO-CV)

- Uses **(n-1)** instances for training and 1 instance for testing, repeating for all data points
- **Pros:** Uses all data for training, maximizing learning.
- **Cons:** Extremely slow for large datasets.

Leave-One-Out Cross-Validation (LOO-CV)

- **For each data point** in the dataset:

  Use all the other points as the training set.

  Use the single remaining point as the test set.

- **Repeat this process N times**, each time selecting a different data point as the test set.
- **Calculate performance metrics** for each iteration.
- **Compute the final score** by averaging all iterations

5.Time Series Validation (Rolling Window Validation)

- Used for **time-dependent** data (e.g., stock prices, weather forecasting).
- Splits data in a way that respects the chronological order (no future data leaks).

Example:

- Train on **[Jan–June]**, test on **[July]**.
- Train on **[Feb–July]**, test on **[Aug]**.
- Repeat for all time windows.

Common Issues in Validation

- **Data Leakage:** Information from the test set is used in training, leading to overly optimistic results.
- **Imbalanced Data:** Class imbalance can mislead validation metrics (use stratified k-fold).
- **Improper Splitting:** Ensure validation data is representative of real-world distributions.

Dimensionality Reduction in Machine Learning

- Dimensionality reduction is the process of reducing the number of **features (variables)** in a dataset while retaining as much **information** as possible.
- It helps in improving computational efficiency and avoiding the **curse of dimensionality**.

Why is Dimensionality Reduction Important?

- ✓Reduces computation time & memory usage
- ✓Removes noise & redundant features
- ✓Avoids overfitting in ML model
- ✓Improves visualization of high-dimensional data

Types of Dimensionality Reduction Techniques

1.Feature Selection
Selects a subset of important features from the original dataset.

Example Methods:

- **Filter Methods** (Correlation, Chi-Square Test, Mutual Information)
- **Wrapper Methods** (Recursive Feature Elimination - RFE)
- **Embedded Methods** (Lasso Regression, Decision Tree Importance)

Filter Methods

These methods rank features based on statistical techniques and select the most relevant ones before model training.

1. Correlation

•If two features are highly correlated (e.g., Pearson correlation > 0.9), one can be removed to avoid redundancy.

•**Use case:** Removing redundant columns in datasets like medical records or stock market data.

2. Chi-Square Test (For categorical features)

•The Chi-Square test measures the dependence between a feature and the target variable.

•**Use case:** In a customer churn dataset, selecting features like "Subscription Type" and "Payment Method" that significantly impact churn.

3. Mutual Information

•Measures how much information a feature provides about the target variable.

•**Use case:** Selecting the most informative features in text classification problems (e.g., choosing the most important words in spam detection).

Wrapper Methods

These use a model to evaluate different feature subsets iteratively.

Recursive Feature Elimination (RFE)

- RFE starts with all features and iteratively removes the least important ones based on model performance.
- **Use case:** In a financial fraud detection model, RFE can select the top 10 most important transaction attributes.

Embedded Methods

These select features during model training.

1. Lasso Regression (L1 Regularization)

•Shrinks the coefficients of less important features to zero, effectively removing them.

•**Use case:** In housing price prediction, Lasso can remove features like "House Color" if they don’t significantly impact price.

2. Decision Tree Feature Importance

•Decision trees assign an importance score to each feature based on how much it improves the decision-making process.

•**Use case:** In a customer segmentation dataset, tree-based models can identify key factors influencing customer preferences.

Feature Extraction (Projection Methods)

Transforms data into a lower-dimensional space while keeping important information.

Example Techniques:

(a) Principal Component Analysis (PCA)

•Finds new uncorrelated features (Principal Components) that capture the most variance.

•Used for image compression, facial recognition, etc.

Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA
import numpy as np

# Sample Data
X = np.random.rand(100, 5)  # 100 samples, 5 features

# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

print("Original Shape:", X.shape)  # (100,5)
print("Reduced Shape:", X_reduced.shape)  # (100,2)
```

(b)Linear Discriminant Analysis (LDA)

- Similar to PCA but focuses on **maximizing class separability**.
- Used in classification tasks

LDA works by:

1.Maximizing class separability – It projects data onto a lower-dimensional space where class separation is maximized.

2.Minimizing intra-class variance – It ensures that data points within the same class remain close to each other.

3.Maximizing inter-class variance – It pushes different classes farther apart.

(b)Linear Discriminant Analysis (LDA)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=1)  # Reduce to 1D
X_lda = lda.fit_transform(X, y)  # Requires class labels (y)
```

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Non-linear** technique used for **visualizing** high-dimensional data in 2D/3D.
- Great for clustering and exploratory analysis.

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2)
X_embedded = tsne.fit_transform(X)
```

3. Autoencoders (Deep Learning Approach)

•Uses a neural network to compress data into a **latent space** and reconstruct it.
•Good for image and text data.

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

input_layer = Input(shape=(5,))
encoded = Dense(2, activation='relu')(input_layer)  # Reduce to 2D
decoded = Dense(5, activation='sigmoid')(encoded)

autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
```

Data Compression in Machine Learning

Data compression is the process of **reducing the size** of data while preserving as much **important information** as possible.

It helps in reducing **storage space, transmission time, and computational cost** in ML applications.

Types of Data Compression

1. Lossless Compression (No Information Loss)

Original data can be **perfectly** reconstructed.

Used in text, structured data, and scientific data.

Examples:

- **Huffman Coding**
- **Run-Length Encoding (RLE)**
- **Lempel-Ziv-Welch (LZW) Algorithm**

Example: Compressing Text Data

```
import zlib

data = b"Machine learning compresses data effectively!"
compressed_data = zlib.compress(data)
print("Original Size:", len(data), "Compressed Size:", len(compressed_data))
```

Lossy Compression (Some Information Lost)

Reduces data size by removing **less important** details.

Used in images, videos, and audio (JPEG, MP3, MP4).

Examples:

- **JPEG Compression (Images)**
- **MP3 Compression (Audio)**
- **Deep Learning Autoencoders**

Example: Compressing an Image using JPEG

```
from PIL import Image



# Open Image

image = Image.open("image.jpg")



# Save with compression

image.save("compressed.jpg", "JPEG", quality=30)
```

Data Compression in ML: Principal Component Analysis (PCA) in Machine Learning

- Principal Component Analysis (PCA) is a **dimensionality reduction** technique that transforms high-dimensional data into a lower-dimensional space while preserving the most **important variance** in the dataset.
- Helps remove noise and redundant features
- Speeds up machine learning models
- Useful for visualization (reducing dimensions to 2D/3D)

Principal Component Analysis

Variance

- Variance measures the spread or dispersion of a single random variable around its mean (expected value).
- It quantifies how much the values of the variable deviate from the mean.
- A higher variance indicates that the data points are more spread out, while a lower variance signifies that they are closer to the mean.

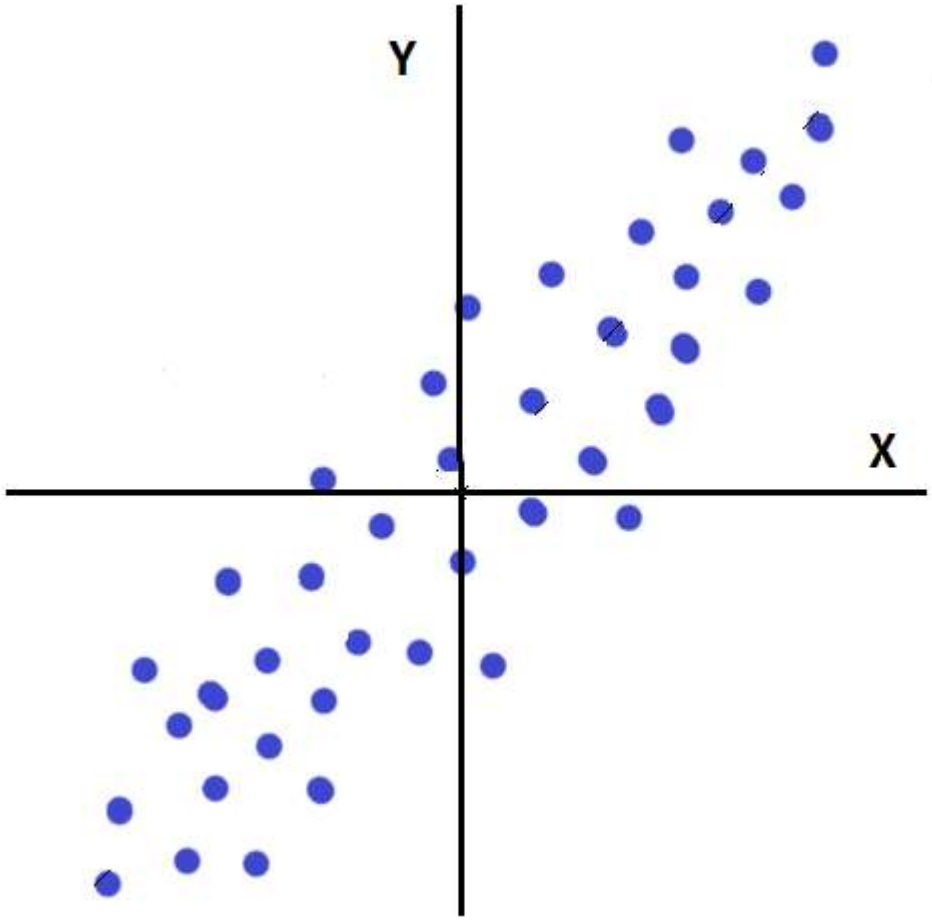
Covariance

- Covariance assesses the degree to which two random variables change together.
- It indicates whether an increase in one variable corresponds to an increase or decrease in another variable.

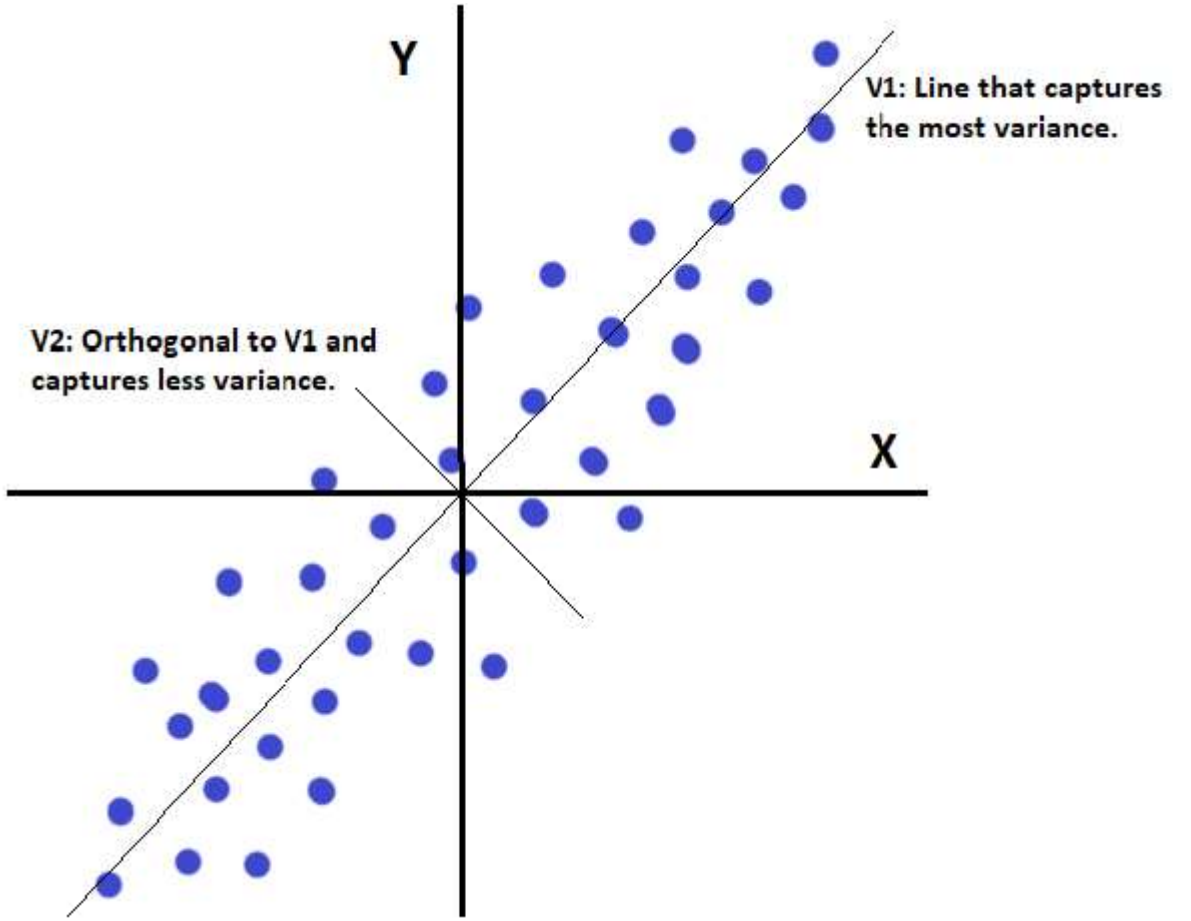
- **Positive Covariance**: Both variables tend to increase or decrease together.
- **Negative Covariance**: One variable increases while the other decreases.
- **Zero Covariance**: No linear relationship between the variables.

Principal Component Analysis (PCA)

- In PCA, principal component is a linear combination of the original variables.
- The analysis produces a series of components: the first component explains the most variance, the second component explains the second most variance, and so on.
- first several components that explain the greatest amount of variation
- Principal component analysis works by rotating the axes to produce a new coordinate system.
- such as standardizing all variables, assessing their correlation matrix, and transforming the original data values



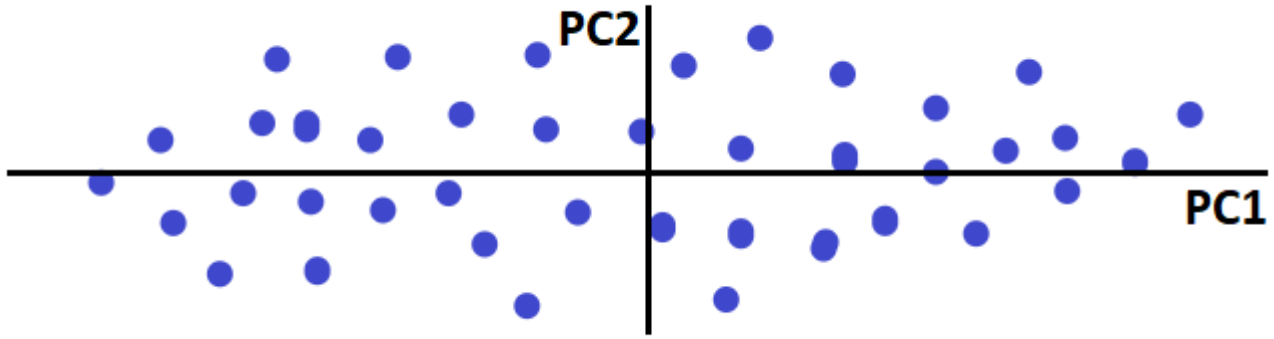
Here’s our original data. Clearly, the variables are correlated. This graph uses the native axes, where each one represents an original variable.



In this graph, V1 and V2 represent new vectors and are the principal components. The distance that data spread out along each vector represents the amount of variance the component captures. More variance equals more information.

- V1 represents the first component. The standard vertical distance between the data points and the new axis is less than their distance to the original X-axis.
- Technically, the vector minimizes the sum of the squared errors like a least squares regression line.
- The length of the second component’s line (V2) is shorter than the first. The relative lengths indicate that the first component accounts for significantly more variance than the second.

- Finally, the two vectors are perpendicular, making them orthogonal.



- The data points represent the same observations as before but now use new coordinates based on the orthogonal vectors. As you can see, there's no correlation because the data points have no slope.
- Next, we want to simplify the data and use only one principal component. A single principal component score (a value on the new X-axis) represents each data point. That value is a linear combination of both original variables.

- As you saw in the rotation process, the new axis uses the relationship between the original variables to combine information from both, but you can’t directly interpret the new values. However, you can use the principal component scores in subsequent analyses.
- When your dataset contains more dimensions, simply extend the process. Each subsequent axis transformation must satisfy the following:
- It accounts for more variance than any other potential new axis.
- It must be perpendicular/orthogonal to all previous axes.

- Each principal component has a pair of these values. In the context of the geometric example:
- Eigen vectors signify the orientation of the new axes.
- Eigen values represent the line length or the amount of variance/information the new axis explains.
- Principal component analysis computes these values from the correlation matrix.
- Most statistical software ranks the principal components by their eigen vectors from largest to smallest. This process creates a list of components ordered from explaining the most to least variance.

How PCA Works

1. **Standardize Data** – Scale features to have **zero mean and unit variance**.
2. **Compute Covariance Matrix** – Identifies relationships between features.
3. **Calculate Eigenvectors & Eigenvalues** – Determines new feature directions.
4. **Select Principal Components** – Choose top eigenvectors (highest variance).
5. **Transform Data** – Map original data onto the new principal components.

Principal Component Analysis (PCA) - Feature Compression

```
from sklearn.decomposition import PCA
import numpy as np

X = np.random.rand(100, 10)  # 100 samples, 10 features

# Reduce to 3 features
pca = PCA(n_components=3)
X_compressed = pca.fit_transform(X)

print("Original Shape:", X.shape, "Compressed Shape:",
X_compressed.shape)
```

Example: PCA on a Sample Dataset

We'll apply PCA on the **Iris dataset** to reduce it from **4D to 2D** for visualization.

Step 1: Load Data & Preprocess

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris

# Load dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target  # Target labels

# Standardize data (PCA works better with normalized data)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=['species']))
```

Step 2: Apply PCA (Reduce 4D → 2D)

```
# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Convert to DataFrame
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df_pca['species'] = df['species']
```

Step 3: Visualize PCA Results

```
# Plot the PCA-transformed data
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue=df_pca['species'], palette='viridis')
plt.title("PCA: Iris Dataset (4D → 2D)")
plt.show()
```

Interpreting PCA Components

PCA finds new **orthogonal axes (principal components)** that maximize variance. We can check how much variance each component explains:

```
explained_variance = pca.explained_variance_ratio_
print("Explained Variance:", explained_variance)  # Shows % variance
captured by each PC
```

Example Output: [0.72, 0.23]

🔹 PC1 captures **72%** of variance, PC2 captures **23%**, meaning we kept **95%** of information!

When to Use PCA?

When you have **many features** (high-dimensional data)

To **reduce noise** and redundant features

For **faster training** of ML models

For **visualization** of high-dimensional data

Autoencoders

- Autoencoders use **neural networks** to encode data into a smaller representation
- One of the popular methods of dimensionality reduction is auto-encoder, which is a type of **ANN** or **artificial neural network**, and its main aim is to copy the inputs to their outputs.
- In this, the input is compressed into latent-space representation, and output occurs using this representation.
- It has mainly two parts:

Encoder: The function of the encoder is to compress the input to form the latent-space representation.

Decoder: The function of the decoder is to recreate the output from the latent-space representation.

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

# Define Autoencoder
input_layer = Input(shape=(10,))
encoded = Dense(3, activation='relu')(input_layer)  # Compressed to 3D
decoded = Dense(10, activation='sigmoid')(encoded)

autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mse')

# Print model summary
autoencoder.summary()
```