Support Vector Machine (SVM)

Class 8

15/2/2025

Acknowledgement

The series of the IT & Japanese language course is Supported by AOTS and OEC.

AOTS

Ministry of Economy, Trade and Industry

OEC

Overseas Employment Corporation

What you have Learnt Last Week

We were focused on following points.

- Usage of control and loop flow statement
- Performing Linear Algebra in Numpy
- Inspecting and Understanding Data
- Basics of creating, loading, and exploring DataFrames
- Understanding of 1D, and 2D NumPy arrays
- Array indexing and slicing
- Linear Regression
- Cost Function

What you will Learn Today

We will focus on following points.

- Multi Linear regression
- Introduction to Support Vector Machines
- Explain difference between linear and non linear SVM
- Introduction to the Radial Basis Function (RBF) Kernel
- Practical implementation of support vector machine with example
- Upload code on Github
- Quiz
- Q&A Session

Linear Vs Multiple Linear Regression

Multiple Linear Regression models the relationship between a dependent variable (Y) and multiple independent variables

Difference Between Simple and Multiple Regression

Aspect	Simple Linear Regression	Multiple Linear Regression
Number of Predictors	One	More than one
Visualization	2D (Line)	Higher-dimensional Space
Complexity	Low	Higher

When a Multilinear Regression model is overfitted, the model performs well on training data but poorly on test data

Multiple Linear Regression

Multiple Linear Regression models the relationship between a dependent variable (Y) and multiple independent variables (X1, X2, ..., Xn).

[Why Multiple Linear Regression]

- More Accurate Predictions
- Better Understanding of Relationships
- Handling Complex Problems
- Improved Decision Making

Formula:

$$Y = b_0 + b_1X_1 + b_2X_2 + ... + b_nX_n + \epsilon$$

R-squared (R^2)commonly used to assess the performance of a Multilinear Regression model

Multiple Linear Regression

1. Import Libraries and sample dataset

```
import pandas as pd

from sklearn.model_selection import train_test_splitfrom

sklearn.linear_model import LinearRegression


# Sample dataset
data = {'X1': [1, 2, 3, 4, 5],
        'X2': [2, 4, 6, 8, 10],
        'Y': [3, 6, 9, 12, 15]}
df = pd.DataFrame(data)
```

Multiple Linear Regression

2. Defining features, target and then split the dataset

```
X = df[['X1', 'X2’]]
Y = df['Y’]
# Splitting dataset for training and testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

Multiple Linear Regression

3. Train the Model and then predict the model

```
# Model training
model = LinearRegression()model.fit(X_train, Y_train)

# Predictions
Y_pred = model.predict(X_test)print("Predictions:", Y_pred)

# Coefficients
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

Task-1

House Price Prediction

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

# Load dataset (House Price Prediction example)

data = pd.DataFrame ({

   'sqft_living': [1500, 2000, 1800, 2500, 1200, 3000, 2200, 2800, 1400, 3300],

   'bedrooms': [3, 4, 3, 5, 2, 4, 3, 5, 2, 6],

   'bathrooms': [2, 3, 2, 4, 1, 3.5, 2.5, 3, 1.5, 4],

   'floors': [1, 2, 2, 2, 1, 2, 1, 2, 1, 2],

   'age': [10, 5, 15, 8, 20, 2, 12, 7, 25, 3],

   'price': [250000, 400000, 320000, 600000, 180000, 750000, 450000, 700000,
200000, 900000]

})

# Select features and target variable

X = data[['sqft_living', 'bedrooms', 'bathrooms', 'floors', 'age']]

y = data['price’]

# Split 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)

# Initialize and train the model

model = LinearRegression()

model.fit(X_train, y_train)

# Make predictions

y_pred = model.predict(X_test)


# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


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

print(f"R-squared: {r2}")


# Plot actual vs predicted values

plt.scatter(y_test, y_pred)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual vs Predicted Prices")

plt.show()
```

Support Vector Machine (SVM)

A supervised learning algorithm for classification and regression

[Why SVM]

- Finds the optimal hyperplane to separate different classes.
- Works well for high-dimensional data.

[Key Concepts]

- **Hyperplane:** Decision boundary between classes.
- **Support Vectors:** Data points closest to the hyperplane.
- **Margin:** Distance between support vectors and the hyperplane.
- **High-dimensional data:** datasets that have a large number of features

Linear SVM vs Non-Linear SVM

Linear SVM uses a straight-line hyperplane for linearly separable data, while Non-Linear SVM uses the kernel trick to handle complex decision boundaries.

Linear SVM vs Non-Linear SVM

Feature	Linear SVM	Non-Linear SVM
Usage	Works when data is linearly separable.	Used for complex decision boundaries.
Hyperplane	Uses a straight-line hyperplane.	Uses the **kernel trick** to map data to higher dimensions.
Computation	Faster and computationally efficient.	More computationally intensive.
Common Kernels	Not applicable.	Polynomial, RBF, Sigmoid.

The **kernel trick** is a technique in SVM that transforms data into a higher-dimensional space, making it possible to find a linear boundary for data that is not linearly separable in its original space. It allows SVM to handle complex decision boundaries efficiently without explicitly computing the transformation

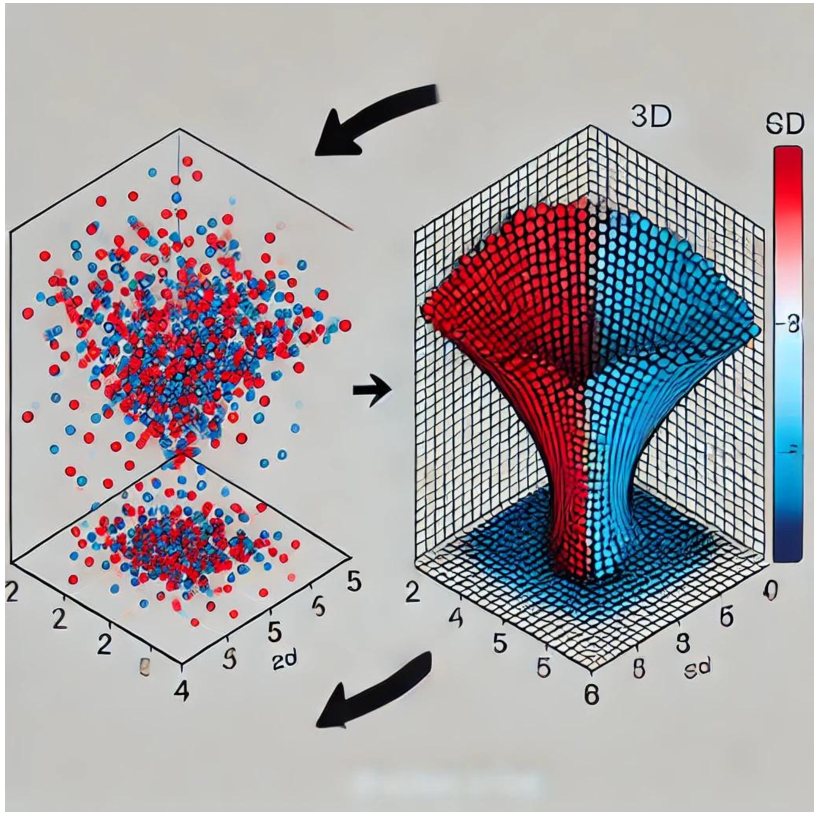
Kernel Trick

Transforms non-linearly separable data into higher dimensions.

Helps SVM classify data in complex patterns

[Commonly used kernels]

1. **Linear Kernel:** For simple cases.
2. **Polynomial Kernel:** When relationships involve curves.
3. **RBF Kernel:** Works well in most cases.



Here's a visual representation of the kernel trick! On the left, you can see how data is not linearly separable in 2D, and on the right, the transformation into a higher-dimensional space allows a linear separation

The Radial Basis Function (RBF) Kernel

Maps data into an infinite-dimensional space

Mathematical Formula:

$$K(x, y) = \exp\left(-\gamma \|x - y\|^2\right)$$

where:

- $K(x, y)$ is the kernel function.
- $\|x - y\|^2$ is the squared Euclidean distance between two data points.
- γ (gamma) controls how far the influence of a data point extends. A **high** γ makes the model focus on nearby points, while a **low** γ makes it consider more distant points.

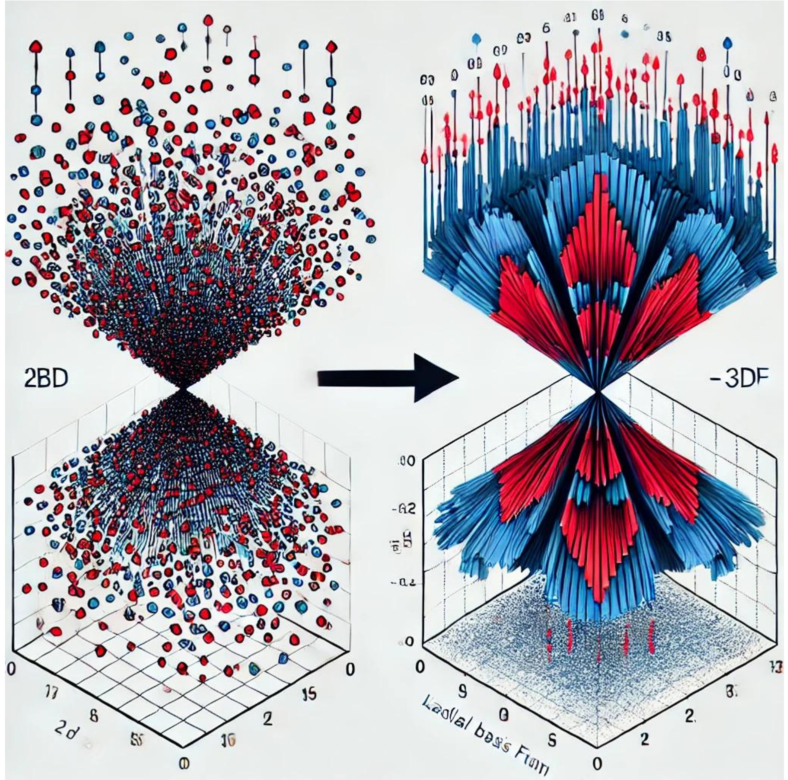
It is useful when the decision boundary is highly complex and non-linear

The Radial Basis Function (RBF) Kernel

It measures the similarity between two points based on their distance.

Why Use RBF Kernel?

1. Works well even when the data is not linearly separable.
2. Can model highly complex decision boundaries.
3. Avoids the need to manually define polynomial features.



The left side shows non-linearly separable data in 2D, while the right side illustrates how the kernel lifts the data into a higher-dimensional space, allowing a linear separation

Practical Implementation of SVM

1. Import Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import make_classification
```

- **NumPy (np)** → Used for numerical operations.
- **Matplotlib (plt)** → Used for plotting decision boundaries.
- **SVC (Support Vector Classifier)** → Implements **SVM** from sklearn.
- **make_classification** → Generates synthetic dataset for classification.

Practical Implementation of SVM

2. Generate a Synthetic Dataset

```
X, y = make_classification(n_samples=100, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_clusters_per_class=1,
                           class_sep=1.5, random_state=42)
```

- **n_samples=100** → 100 data points.
- **n_features=2** → Each point has 2 features (makes visualization possible).
- **n_informative=2** → Both features contain useful information for classification.
- **n_redundant=0** & **n_repeated=0** → No unnecessary features.
- **n_clusters_per_class=1** → One cluster per class for simplicity.
- **class_sep=1.5** → Controls separation between classes (higher = more separable).
- **random_state=42** → Ensures reproducibility.

Practical Implementation of SVM

3. Train SVM Models

```
#Creates an SVM classifier with a linear kernel
linear_svm = SVC(kernel='linear')
linear_svm.fit(X, y)

#Creates an SVM classifier with the RBF (Gaussian) kernel.
rbf_svm = SVC(kernel='rbf', gamma='scale')
rbf_svm.fit(X, y)
```

- **Creates an SVM classifier with a linear kernel.**
- **Trains it on the dataset (X, y).**
- **Creates an SVM classifier with the RBF (Gaussian) kernel.**
- **Uses gamma='scale'**, which determines how much influence each data point has.

Practical Implementation of SVM

4. Define a Function to Plot Decision Boundaries

```
def plot_decision_boundary(model, X, y, title):
    xx, yy = np.meshgrid(np.linspace(X[:,0].min()-1, X[:,0].max()+1, 100),
                         np.linspace(X[:,1].min()-1, X[:,1].max()+1, 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(6, 4))
    plt.contourf(xx, yy, Z, alpha=0.3, levels=1, cmap='coolwarm')
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap='coolwarm')
    plt.title(title)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()
```

Practical Implementation of SVM

5. Plot the Decision Boundaries

```
plot_decision_boundary(linear_svm, X, y, "Linear SVM Decision Boundary")
plot_decision_boundary(rbf_svm, X, y, "RBF Kernel SVM Decision Boundary")
```

[Expected Output]

- **Linear SVM** → A **straight-line** decision boundary.
- **RBF Kernel SVM** → A **curved boundary** that adapts to complex data.

[Key Takeaways]

1. **Linear SVM** → Works well for linearly separable data.
2. **RBF Kernel SVM** → Handles complex patterns by mapping data to higher dimensions.
3. **Decision boundaries show how models classify data points.**

Task-2

Classifying Handwritten Digits with SVM

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Load the handwritten digits dataset
digits = datasets.load_digits()

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

# Standardize the feature values for better model performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train an SVM model using the RBF kernel
svm_digits = SVC(kernel='rbf', gamma=0.01, C=10)
svm_digits.fit(X_train, y_train)

# Predict on the test set
y_pred = svm_digits.predict(X_test)

# Display classification metrics
print(classification_report(y_test, y_pred))

# Visualize random predictions
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for ax in axes.flat:
    index = random.randint(0, len(y_test) - 1)  # Ensure valid index
    ax.imshow(digits.images[index], cmap='gray')  # Use original image data
    ax.set_title(f"Pred: {y_pred[index]}")
    ax.axis("off")

plt.tight_layout()
plt.show()
```

Quiz Section

Quiz

Everyone student should click on submit button before time ends otherwise MCQs will not be submitted

[Guidelines of MCQs]

1. There are 20 MCQs
2. Time duration will be 10 minutes
3. This link will be share on 6:10pm (Pakistan time)
4. MCQs will start from 6:15pm (Pakistan time)
5. This is exact time and this will not change
6. Everyone student should click on submit button otherwise MCQs will not be submitted after time will finish
7. Every student should submit Github profile and LinkedIn post link for every class. It include in your performance

Assignment

Assignment should be submit before the next class

[Assignments Requirements]

1. Create a post of today's lecture and post on LinkedIn.
2. Make sure to tag @Plus W @Pak-Japan Centre and instructors LinkedIn profile
3. Upload your code of assignment and lecture on GitHub and share your GitHub profile in respective

   your region group WhatsApp group
4. If you have any query regarding assignment, please share on your region WhatsApp group.
5. Students who already done assignment, please support other students

Q&A Session

ありがとうございます。

Thank you.

شکریا

For the World with Diverse Individualities