🧩 *Day 7 – K-Nearest Neighbors (KNN) & Support Vector Machine (SVM)*

📅 Training Summary

On Day 7, we explored two widely used classification techniques in machine learning — **K-Nearest Neighbors (KNN)** and **Support Vector Machine (SVM)**. We implemented, tuned, and evaluated both models using the well-known **Iris flower dataset**, leveraging powerful tools from the scikit-learn library. Visualization and error metrics were used for detailed comparison.

□ Theory Recap

📌 K-Nearest Neighbors (KNN)

- It is a **lazy learner** that stores the training data and delays learning until prediction time.
- It classifies a sample based on the **majority class** among its nearest 'k' neighbors.
- Uses **distance metrics** like:
  - Euclidean Distance: $\sqrt{(x2-x1)^2 + (y2-y1)^2}$
  - Manhattan Distance: $|x2-x1| + |y2-y1|$

📌 Support Vector Machine (SVM)

- A **margin-based classifier** that aims to find the best hyperplane separating classes.
- Excellent at handling **high-dimensional** datasets.
- Uses **kernels** to handle non-linear boundaries. Common ones:
  - Linear
  - Polynomial
  - RBF (Radial Basis Function)

□ KNN Implementation – with Extended Functionality

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix, classification_report, mean_squared_error,
mean_absolute_error
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Initialize KNN with k=6
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(x_train, y_train)

# Predict and evaluate
y_pred = knn.predict(x_test)
print(f"KNN Accuracy (k=6): {knn.score(x_test, y_test):.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
plt.figure(figsize=(5,5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="YlGnBu", fmt="d")
plt.title("KNN Confusion Matrix (k=6)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Testing multiple k-values
k_vals = list(range(1, 16))
scores = [KNeighborsClassifier(n_neighbors=k).fit(x_train, y_train).score(x_test, y_test) for k in
k_vals]

plt.plot(k_vals, scores, marker='o', linestyle='dashed', color='green')
plt.title("KNN Accuracy across k-values")
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()

# Final Model with k=4
final_knn = KNeighborsClassifier(n_neighbors=4)
final_knn.fit(x_train, y_train)
final_preds = final_knn.predict(x_test)

# Encode for MAE/MSE
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
true_encoded = le.fit_transform(y_test)
pred_encoded = le.transform(final_preds)

mae_knn = mean_absolute_error(true_encoded, pred_encoded)
mse_knn = mean_squared_error(true_encoded, pred_encoded)

print("KNN (k=4) Final Accuracy:", final_knn.score(x_test, y_test))
print("MAE (KNN):", mae_knn)
print("MSE (KNN):", mse_knn)
```

```
method_names.append("KNN")
method_scores.append(final_knn.score(x_test, y_test))
```

▯ SVM Implementation – with Enhancements

```
from sklearn.svm import SVC

# Linear Kernel SVM
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(x_train, y_train)
svm_preds = svm_model.predict(x_test)

# Accuracy
print(f"SVM Accuracy (Linear Kernel): {svm_model.score(x_test, y_test):.2f}")

# Confusion Matrix & Visual
plt.figure(figsize=(5,5))
sns.heatmap(confusion_matrix(y_test, svm_preds), annot=True, cmap="Reds", fmt="d")
plt.title("SVM Confusion Matrix (Linear Kernel)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Errors
svm_mae = mean_absolute_error(le.fit_transform(y_test), le.transform(svm_preds))
svm_mse = mean_squared_error(le.fit_transform(y_test), le.transform(svm_preds))

print("MAE (SVM):", svm_mae)
print("MSE (SVM):", svm_mse)

method_names.append("SVM")
method_scores.append(svm_model.score(x_test, y_test))
```

📊 Visual Comparison of Models

```
plt.figure(figsize=(6,4))

sns.barplot(x=method_names, y=method_scores, palette="Set2")

plt.title("Model Comparison on Iris Dataset")

plt.ylabel("Accuracy")

plt.ylim(0.8, 1.0)

plt.show()
```

✅ Final Observations

KNN showed strong accuracy, especially around k=4. Hyperparameter tuning helped optimize performance.

SVM with a linear kernel gave very high precision and clean class separation using maximum margin.

Both algorithms are effective, but SVM may outperform on more complex, high-dimensional datasets.

MAE and MSE further verified prediction consistency for both models.