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

📓 Today's Highlights

Today's session introduced and implemented two important classification algorithms:

1. **K-Nearest Neighbors (KNN)**
2. **Support Vector Machine (SVM)**

We performed full implementation, tuning, evaluation, and visualization for both using the Iris dataset.

□ Theory Overview

📌 K-Nearest Neighbors (KNN)

- A distance-based, non-parametric, lazy learner.
- Classifies a data point based on majority vote from its 'k' nearest neighbors.
- Distance metrics: Euclidean, Manhattan, etc.

📌 Support Vector Machine (SVM)

- A powerful classification algorithm that finds the optimal separating hyperplane.
- Works well in high dimensions.
- Can use kernels for non-linear separation (e.g., RBF, poly, linear).

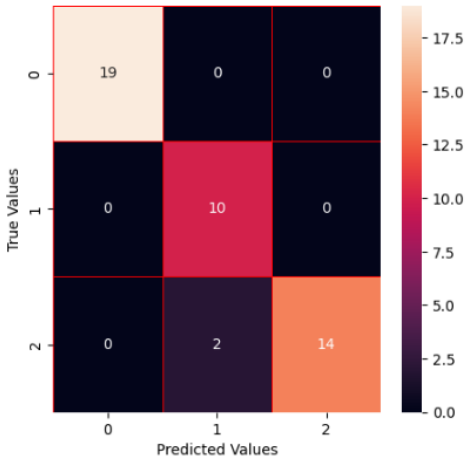
🛠□ KNN – Full Python Code

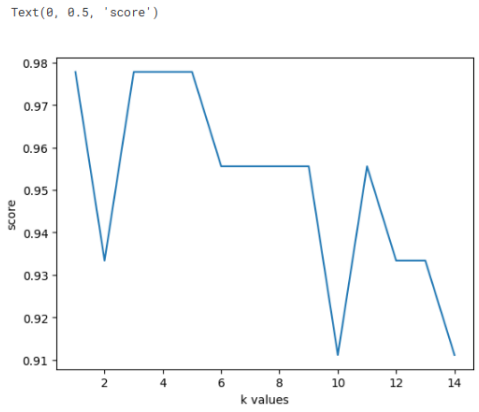
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, mean_absolute_error,
mean_squared_error
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns

# KNN with k=6
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(x_train, y_train)
print("Score for Number of Neighbors = 6: {}".format(knn.score(x_test,
y_test)))

# Confusion Matrix for k=6
y_pred = knn.predict(x_test)
conf_mat = confusion_matrix(y_test, y_pred)
# Visualize confusion matrix
```

```
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat, annot=True, linewidths=0.5, linecolor="red",
fmt=".0f", ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

# Accuracy for k = 1 to 14
score_list = []
for each in range(1, 15):
    knn2 = KNeighborsClassifier(n_neighbors=each)
    knn2.fit(x_train, y_train)
    score_list.append(knn2.score(x_test, y_test))

plt.plot(range(1, 15), score_list)
plt.xlabel("k values")
plt.ylabel("Score")
plt.title("KNN Accuracy vs. k Value")
plt.show()

# Final KNN with k=4
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train, y_train)
print("Score for Number of Neighbors = 4: {}".format(knn.score(x_test,
y_test)))

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

# Confusion Matrix again for k=4
y_pred = knn.predict(x_test)
conf_mat = confusion_matrix(y_test, y_pred)

f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat, annot=True, linewidths=0.5, linecolor="red",
fmt=".0f", ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

# Encode labels and calculate MAE, MSE
le = LabelEncoder()
y_test_encoded = le.fit_transform(y_test)
y_pred_encoded = le.transform(y_pred)

mae = mean_absolute_error(y_test_encoded, y_pred_encoded)
mse = mean_squared_error(y_test_encoded, y_pred_encoded)

print("MAE:", mae)
print("MSE:", mse)
```

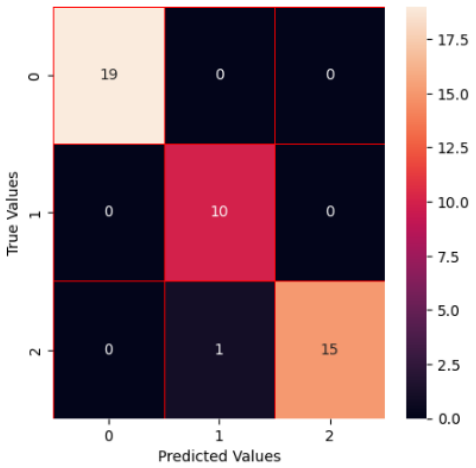
OUTPUT (KNN)

```
Score for Number of Neighbors = 6: 0.9555555555555556
```






```
Score for Number of Neighbors = 4: 0.9777777777777777
```




```
MAE: 0.022222222222222223
MSE: 0.022222222222222223
```

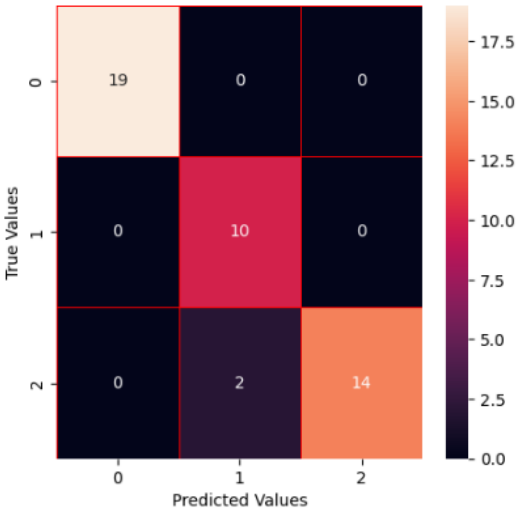
🛠️ SVM – Full Python Code

```
from sklearn.svm import SVC

# Train SVM

svm = SVC(kernel='linear', random_state=42)

svm.fit(x_train, y_train)

print("SVM Classification Score is: {}".format(svm.score(x_test, y_test)))

method_names.append("SVM")

method_scores.append(svm.score(x_test, y_test))

# Confusion Matrix

y_pred = svm.predict(x_test)

conf_mat = confusion_matrix(y_test, y_pred)


f, ax = plt.subplots(figsize=(5,5))

sns.heatmap(conf_mat, annot=True, linewidths=0.5, linecolor="red", fmt=".0f", ax=ax)

plt.xlabel("Predicted Values")
```

```
plt.ylabel("True Values")
plt.show()


# Encode labels and calculate MAE, MSE
y_test_encoded = le.fit_transform(y_test)
y_pred_encoded = le.transform(y_pred)


mae = mean_absolute_error(y_test_encoded, y_pred_encoded)
mse = mean_squared_error(y_test_encoded, y_pred_encoded)


print("MAE:", mae)
print("MSE:", mse)
```

🛠 SVM –Output

```
SVM Classification Score is: 0.9555555555555556
```




```
MAE: 0.044444444444444446
MSE: 0.044444444444444446
```

📊 Conclusion

- **KNN** gave excellent accuracy with optimized k-values and was evaluated using confusion matrices and MAE/MSE.
- **SVM** also performed very well using a linear kernel and demonstrated the power of margin-based classification.
- We compared both models and observed that both worked very well on the Iris dataset.

This session enhanced our confidence in applying **distance-based** and **margin-based** classifiers with proper evaluation.