ASSIGNMENT-6

Title: Apply different Machine Learning approaches for the classification task. Compare the performance of different ML approaches in terms of accuracy, precision, and recall.

Objective: The objective of this lab assignment is to apply various Machine Learning (ML) approaches for a classification task and compare their performance in terms of accuracy, precision, and recall. You will gain hands-on experience in implementing and evaluating different ML algorithms, understanding their strengths and weaknesses, and interpreting their results.

1. Data Loading and Exploration
2. Data Preprocessing
3. Data Splitting
4. Model Training
5. Model Evaluation
6. Visualization, Comparison and Interpretation
7. Hyperparameter Tuning (Optional)

Bonus Tasks (Optional):
- Apply data normalization or feature scaling and observe its impact on the model's performance.
- Implement cross-validation to validate the model's robustness.

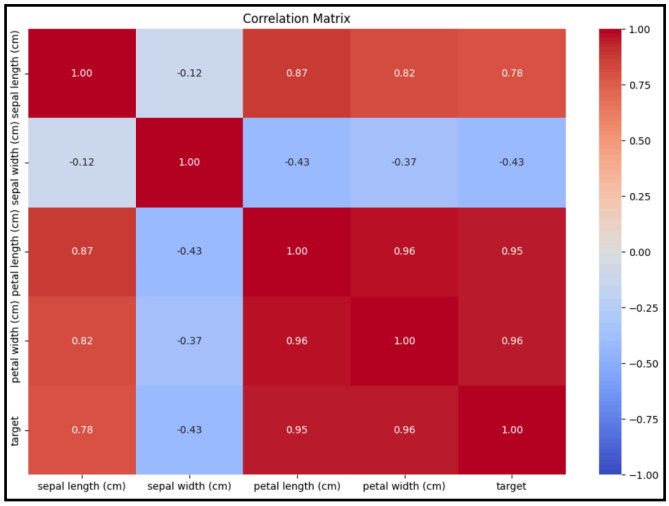
Tasks:

1) Data Loading and Exploration

```
import pandas as pd
import numpy as np
import seaborn as seaborn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.datasets import load_iris

data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
df.info()
df.describe()
df.isnull().sum()
```

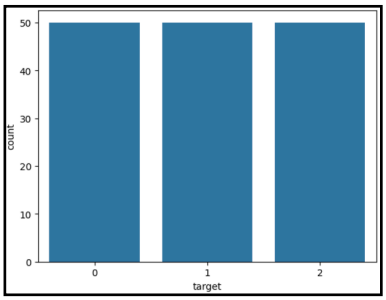
2) Data Preprocessing

```
corr_matrix = df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True,
cmap='coolwarm', fmt='.2f', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```

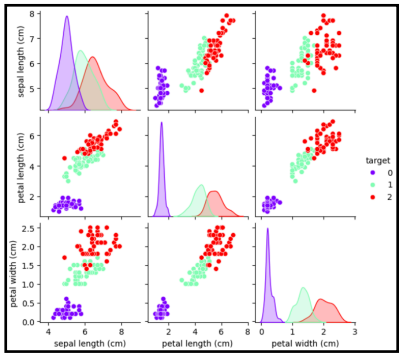



```
t = 0.75
target_corr = corr_matrix['target']
low_corr_features = target_corr[abs(target_corr) <
t].index
df1 = df.drop(columns=low_corr_features)
print("Before: ",len(df.columns),"\nAfter:
‘,len(df1.columns))
```

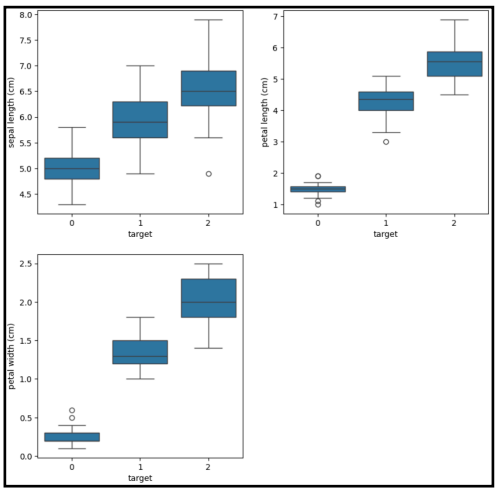
```
sns.countplot(x='target', data=df1)
plt.show()
```




```
sns.pairplot(df1,hue='target',
height=2,palette='rainbow')
```

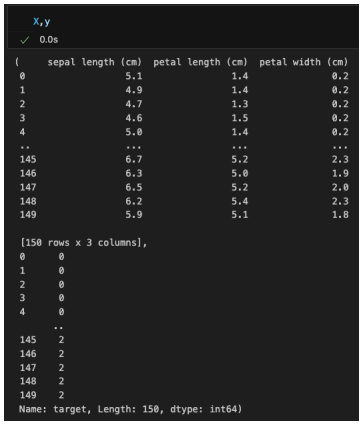



```
def graph(y):
    sns.boxplot(x="target", y=y, data=df1)

plt.figure(figsize=(10,10))
plt.subplot(221)
graph('sepal length (cm)')
plt.subplot(222)
graph('petal length (cm)')
plt.subplot(223)
graph('petal width (cm)')

plt.show()
```



3) Data Splitting

```
from sklearn.model_selection import train_test_split

X = df1.drop(columns='target')
y = df1['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=66)
```

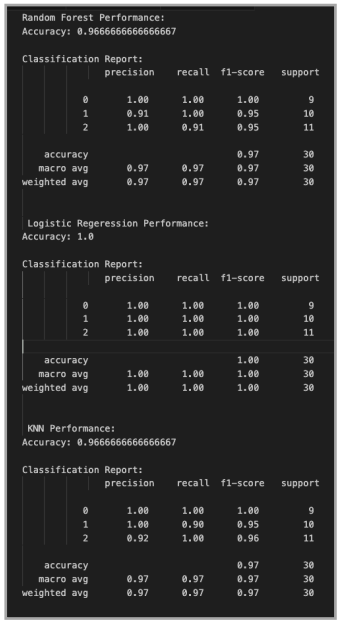


4) Model Training

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

rf = RandomForestClassifier(random_state=22)
rf.fit(X_train, y_train)
lr = LogisticRegression(max_iter=200)
lr.fit(X_train,y_train)
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

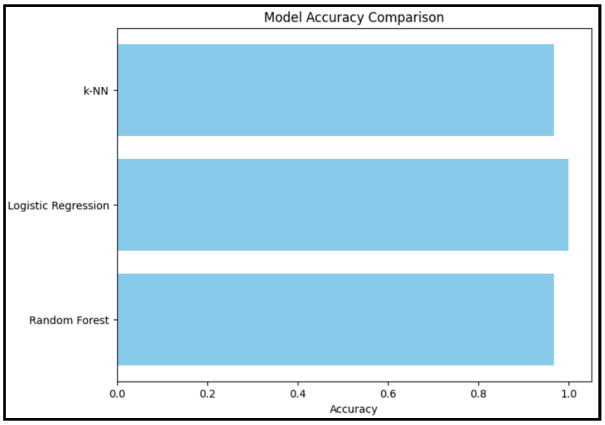
5) Model Evaluation

```
print(f"Random Forest Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))

print(f"\n Logistic Regeression Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("\nClassification Report:\n", classification_report(y_test,
y_pred_lr))

print(f"\n KNN Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("\nClassification Report:\n", classification_report(y_test,
y_pred_knn))
```

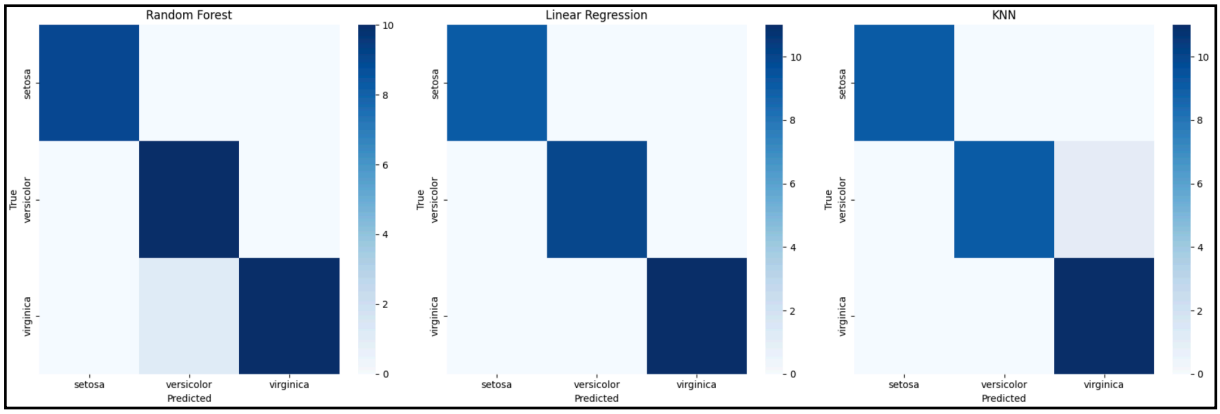


6) Visualization, Comparison and Interpretation

```
import matplotlib.pyplot as plt

m = { 'Random Forest': rf,
   'Logistic Regression': lr,
   'k-NN': knn }
# Calculate accuracies for each model
accuracies = {model_name:
accuracy_score(y_test,
model.predict(X_test)) for model_name,
model in m.items()}

plt.figure(figsize=(8, 6))
plt.barh(list(accuracies.keys()),
list(accuracies.values()), color='skyblue')
plt.xlabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.show()
```

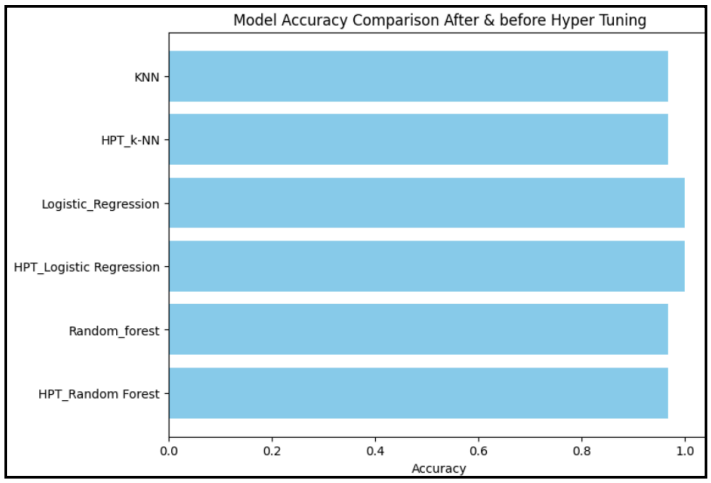



```
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm_rf = confusion_matrix(y_test,y_pred )
cm_lr = confusion_matrix(y_test,y_pred_lr )
cm_knn = confusion_matrix(y_test,y_pred_knn)

fig, axes = plt.subplots(1,3, figsize=(18,6))
sns.heatmap(cm_rf,fmt='d',cmap='Blues', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[0])
axes[0].set_title('Random Forest')
axes[0].set_ylabel('True')
axes[0].set_xlabel('Predicted')

sns.heatmap(cm_lr,fmt='d',cmap='Blues', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[1])
axes[1].set_title('Linear Regression')
axes[1].set_ylabel('True')
axes[1].set_xlabel('Predicted')

sns.heatmap(cm_knn,fmt='d',cmap='Blues', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[2])
axes[2].set_title('KNN')
axes[2].set_ylabel('True')
axes[2].set_xlabel('Predicted')
plt.tight_layout()
plt.show()
```



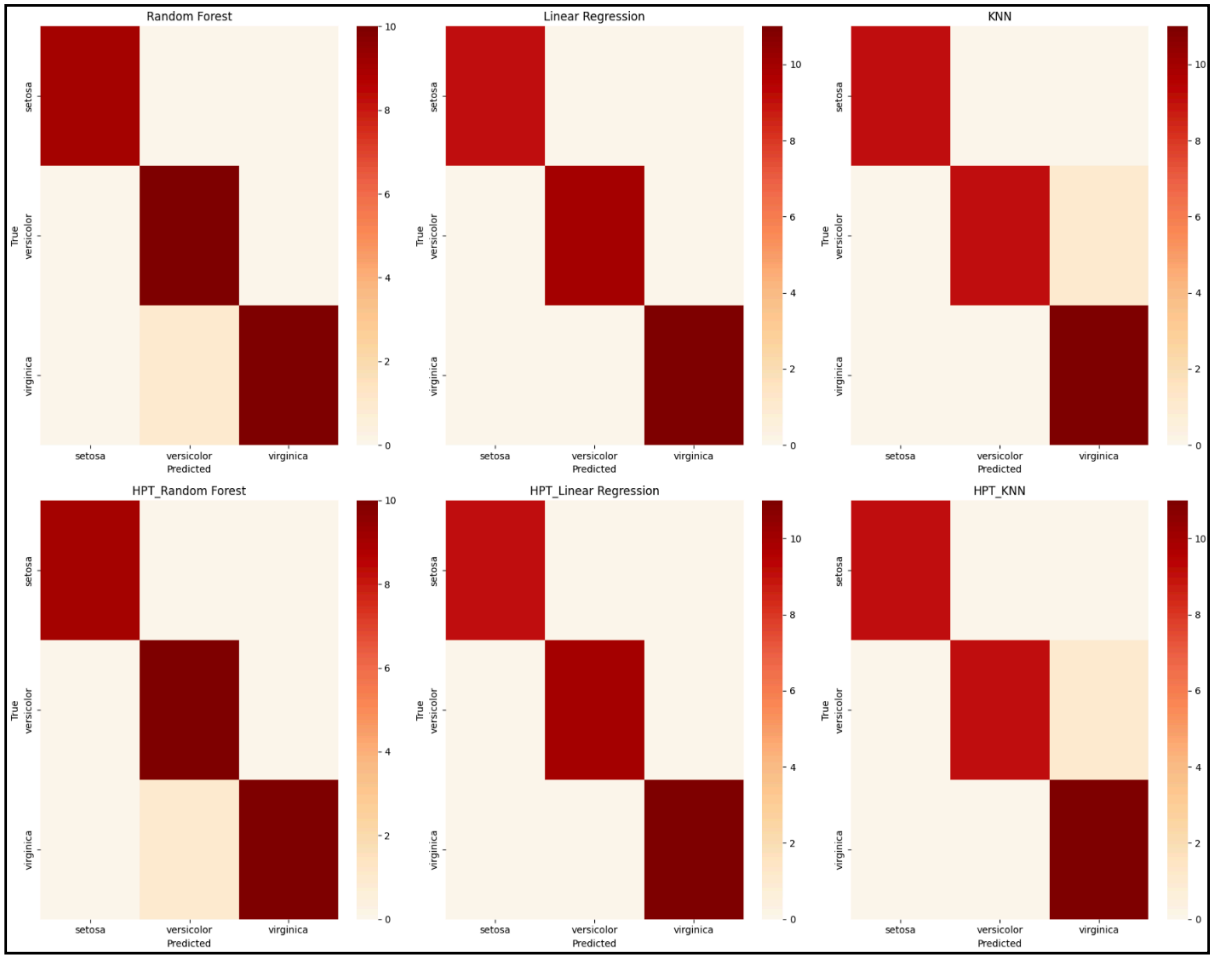
7) Hyperprarameter & Cross Validation Tuning

```
from sklearn.model_selection import GridSearchCV

rf_param_grid = {
   'n_estimators': [20, 40, 60],
   'max_depth': [3, 5, 7, 9],
   'min_samples_split': [2, 5, 8],
   'min_samples_leaf': [1, 2, 4] }

log_reg_param_grid = {
   'C': [0.1, 1, 10, 100],
   'solver': ['liblinear', 'lbfgs'],
   'penalty': ['l2'],
   'max_iter': [50, 100, 150] }

knn_param_grid = {
   'n_neighbors': [3, 6, 9],
   'weights': ['uniform', 'distance'],
   'metric': ['euclidean', ‘manhattan'] }

# Model 1: Random Forest
rf_grid_search = GridSearchCV(RandomForestClassifier(), rf_param_grid, cv=5, n_jobs=-1,
verbose=1)
rf_grid_search.fit(X_train, y_train)
hpt_rf = rf_grid_search.best_estimator_

# Model2 : Logistic Regression
lr_grid_search = GridSearchCV(LogisticRegression(), log_reg_param_grid, cv=5, n_jobs=-1,
verbose=1)
lr_grid_search.fit(X_train, y_train)
hpt_lr = lr_grid_search.best_estimator_

# Model 3: k-NN
knn_grid_search = GridSearchCV(KNeighborsClassifier(), knn_param_grid, cv=5, n_jobs=-1,
verbose=1)
knn_grid_search.fit(X_train, y_train)
hpt_knn = knn_grid_search.best_estimator_

hpt_rf_pred = hpt_rf.predict(X_test)
hpt_lr_pred = hpt_lr.predict(X_test)
hpt_knn_pred = hpt_knn.predict(X_test)
```




```
# Visualising Model Accuracy
m = {
   'HPT_Random Forest': hpt_rf,
   'Random_forest' : rf,
   'HPT_Logistic Regression': hpt_lr,
   'Logistic_Regression' : lr,
   'HPT_k-NN': hpt_knn,
   'KNN' : knn
}


# Calculate accuracies for each model
```

```
accuracies = {model_name: accuracy_score(y_test, model.predict(X_test)) for model_name, model in
m.items()}

plt.figure(figsize=(8, 6))
plt.barh(list(accuracies.keys()), list(accuracies.values()), color='skyblue')
plt.xlabel('Accuracy')
plt.title('Model Accuracy Comparison After & before Hyper Tuning')
plt.show()


#Visualizing Confusion Matrix Before & After Hyper-Parameter Tuning and Cross Validation
cm_rf = confusion_matrix(y_test,y_pred )
cm_lr = confusion_matrix(y_test,y_pred_lr )
cm_knn = confusion_matrix(y_test,y_pred_knn)

cm_hpt_rf = confusion_matrix(y_test,hpt_rf_pred )
cm_hpt_lr = confusion_matrix(y_test,hpt_lr_pred )
cm_hpt_knn = confusion_matrix(y_test,hpt_knn_pred)

fig, axes = plt.subplots(2,3, figsize=(18,14))
sns.heatmap(cm_rf,fmt='d',cmap='OrRd', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[0][0])
axes[0][0].set_title('Random Forest')
axes[0][0].set_ylabel('True')
axes[0][0].set_xlabel('Predicted')

sns.heatmap(cm_lr,fmt='d',cmap='OrRd', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[0][1])
axes[0][1].set_title('Linear Regression')
axes[0][1].set_ylabel('True')
axes[0][1].set_xlabel('Predicted')

sns.heatmap(cm_knn,fmt='d',cmap='OrRd', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[0][2])
axes[0][2].set_title('KNN')
axes[0][2].set_ylabel('True')
axes[0][2].set_xlabel('Predicted')

sns.heatmap(cm_hpt_rf,fmt='d',cmap='OrRd', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[1][0])
axes[1][0].set_title('HPT_Random Forest')
axes[1][0].set_ylabel('True')
axes[1][0].set_xlabel('Predicted')

sns.heatmap(cm_hpt_lr,fmt='d',cmap='OrRd', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[1][1])
axes[1][1].set_title('HPT_Linear Regression')
axes[1][1].set_ylabel('True')
axes[1][1].set_xlabel('Predicted')

sns.heatmap(cm_hpt_knn,fmt='d',cmap='OrRd', xticklabels=data.target_names,
yticklabels=data.target_names, ax=axes[1][2])
axes[1][2].set_title('HPT_KNN')
axes[1][2].set_ylabel('True')
axes[1][2].set_xlabel('Predicted')

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



This shows that after applying hyperparameter tuning and cross-validation the result remained the same as our dataset was iris, Which was quite balanced, Simple,and included only 3 classes.

Conclusion:

In this assignment, we explored the application of various machine learning approaches—Logistic Regression, Random Forest, and k-nearest Neighbors (k-NN)—for a classification task using the Iris dataset. The goal was to compare the performance of these models based on accuracy, precision, and recall.

After loading and preprocessing the data, we split it into training and testing sets, trained the models, and evaluated their performance. The results indicated that all models performed well due to the simplicity and balanced nature of the Iris dataset. Random Forest and k-NN showed slightly lower performance compared to Logistic Regression, with minor differences in accuracy, precision, and recall.

We also applied hyperparameter tuning and cross-validation to optimize the models further. However, the improvements were minimal, which can be attributed to the straightforward structure of the Iris dataset, containing only three classes with well-separated features.

In summary, while hyperparameter tuning and cross-validation are essential steps in model optimization, their impact may be limited when working with simple and balanced datasets like Iris.