Assignment 5.1 - Model Validation

Please submit your solution of this notebook in the Whiteboard at the corresponding Assignment entry as .ipynb-file and as .pdf.

Please do **NOT** rename the file!

State both names of your group members here:

[Rashid Harvey and S M Shameem Ahmed Khan]

In [ ]:

Grading Info/Details - Assignment 5.1:

The assignment will be graded semi-automatically, which means that your code will be tested against a set of predefined test cases and qualitatively assessed by a human. This will speed up the grading process for us.

- For passing the test scripts:
  - Please make sure to **NOT** alter predefined class or function names, as this would lead to failing of the test scripts.
  - Please do **NOT** rename the files before uploading to the Whiteboard!
- **(RESULT)** tags indicate checkpoints that will be specifically assessed by a human.
- You will pass the assignment if you pass the majority of test cases and we can at least confirm effort regarding the **(RESULT)**-tagged checkpoints per task.

In [ ]:

Task 5.1.1 - Binary Classification Evaluation

- Use model implementations of `sklearn` (or other) for Logistic Regression and SVM for classification tasks. Train both models on the `Breast Cancer` dataset. (see given imports) **(RESULT)**
- Evaluate the performance of both models using appropriate classification metrics and implement them using `numpy` only. Report at least on the following: accuracy, precision, recall, F1-score. **(RESULT)**

In [1]:
```
# Useful imports
import numpy as np
from sklearn.datasets import load_breast_cancer, load_diabetes, load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC, SVR
from sklearn.linear_model import LinearRegression, LogisticRegression
import matplotlib.pyplot as plt
```

In [ ]:
```
# Loading Dataset
data = load_breast_cancer()
X = data.data
Y = data.target   # 0 = malignant, 1 = benign

# Scaling the data
scaler = StandardScaler()
X = scaler.fit_transform(X, Y)

# 80/20 train-test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42, stratify=Y
)


# Train the Models(Logistic Regression and SVM)
# Logistic Regression
log_reg = LogisticRegression(max_iter=500)
log_reg.fit(X_train, y_train)
# SVM
svm_clf = SVC(kernel="linear")
svm_clf.fit(X_train, y_train)


# predictions
log_pred = log_reg.predict(X_test)
svm_pred = svm_clf.predict(X_test)


# Defining manual metric functions
def compute_confusion_matrix(y_true, y_pred):
    """Return TP, TN, FP, FN"""
    # true positive
    TP = np.sum((y_true == 1) & (y_pred == 1))
    # true negative
    TN = np.sum((y_true == 0) & (y_pred == 0))
    # false positive
    FP = np.sum((y_true == 0) & (y_pred == 1))
    # false negative
    FN = np.sum((y_true == 1) & (y_pred == 0))
    return TP, TN, FP, FN

def accuracy(TP, TN, FP, FN):
    # correct/total (like in a university exam)
    return (TP + TN) / (TP + TN + FP + FN)
```

```
def precision(TP, FP):
    # correct positive/total predicted positive
    # i.e. how often is the model correct when it predicts positive
    return TP / (TP + FP + 1e-12)  # avoiding division by zero

def recall(TP, FN):
    # true positives/actual positives
    return TP / (TP + FN + 1e-12)

def f1_score(prec, rec):
    # https://en.wikipedia.org/wiki/F-score
    return 2 * (prec * rec) / (prec + rec + 1e-12)


# Evaluation
def evaluate_model(y_true, y_pred, name="Model"):
    TP, TN, FP, FN = compute_confusion_matrix(y_true, y_pred)

    acc = accuracy(TP, TN, FP, FN)
    prec = precision(TP, FP)
    rec = recall(TP, FN)
    f1 = f1_score(prec, rec)

    print(f"\n{name} Evaluation")
    print("-"*30)
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1-score : {f1:.4f}")

# Results
evaluate_model(y_test, log_pred, "Logistic Regression")
evaluate_model(y_test, svm_pred, "SVM")
```

```
Logistic Regression Evaluation
------------------------------
Accuracy : 0.9825
Precision: 0.9861
Recall   : 0.9861
F1-score : 0.9861

SVM Evaluation
------------------------------
Accuracy : 0.9737
Precision: 0.9859
Recall   : 0.9722
F1-score : 0.9790
```

Task 5.1.2 - Multi-Class Classification Evaluation

- Do the same as Task 5.1.1 for the multiclass problem `Iris`. Report on the performance metrics: accuracy, precision, recall, F1-score. **(RESULT)**

In [ ]:
```
# multiclass
```

```
# Setup code copied from above

data = load_iris()
X = data.data
Y = data.target

# Scaling the data
scaler = StandardScaler()
X = scaler.fit_transform(X, Y)

# 80/20 train-test split (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42, stratify=Y
)

# Train the Models(Logistic Regression and SVM)
# Logistic Regression
log_reg = LogisticRegression(max_iter=500)
log_reg.fit(X_train, y_train)
# SVM
svm_clf = SVC(kernel="linear")
svm_clf.fit(X_train, y_train)


# predictions
log_pred = log_reg.predict(X_test)
svm_pred = svm_clf.predict(X_test)

# new code
def confusion_matrix_per_class(y_true, y_pred, num_classes):
    """
    Confusion matrix as above for each class but in a one-vs-rest manner.
    """
    matrices = []
    for cls in range(num_classes):
        TP = np.sum((y_true == cls) & (y_pred == cls))
        TN = np.sum((y_true != cls) & (y_pred != cls))
        FP = np.sum((y_true != cls) & (y_pred == cls))
        FN = np.sum((y_true == cls) & (y_pred != cls))
        matrices.append((TP, TN, FP, FN))
    return matrices

def aggregated_metrics(y_true, y_pred, num_classes):
    matrices = confusion_matrix_per_class(y_true, y_pred, num_classes)
    precisions, recalls, f1s = [], [], []

    for TP, TN, FP, FN in matrices:
        prec = TP / (TP + FP + 1e-12)
        rec = TP / (TP + FN + 1e-12)
        f1 = 2 * prec * rec / (prec + rec + 1e-12)
        precisions.append(prec)
        recalls.append(rec)
        f1s.append(f1)

    accuracy = np.mean(y_true == y_pred)
    precision = np.mean(precisions)
```

```
    recall = np.mean(recalls)
    f1 = np.mean(f1s)

    return accuracy, precision, recall, f1, matrices

# Evaluating Models
num_classes = len(np.unique(Y))

# Logistic Regression
log_acc, log_prec, log_rec, log_f1, log_matrices = aggregated_metrics(y_test, log_p
print(f"\nLogistic Regression Evaluation (Average)")
print("-"*30)
print(f"Accuracy : {log_acc:.4f}")
print(f"Precision: {log_prec:.4f}")
print(f"Recall   : {log_rec:.4f}")
print(f"F1-score : {log_f1:.4f}")

# SVM
svm_acc, svm_prec, svm_rec, svm_f1, svm_matrices = aggregated_metrics(y_test, svm_p
print(f"\nSVM Evaluation (Average)")
print("-"*30)
print(f"Accuracy : {svm_acc:.4f}")
print(f"Precision: {svm_prec:.4f}")
print(f"Recall   : {svm_rec:.4f}")
print(f"F1-score : {svm_f1:.4f}")
```

```
Logistic Regression Evaluation (Average)
------------------------------
Accuracy : 0.9333
Precision: 0.9333
Recall   : 0.9333
F1-score : 0.9333

SVM Evaluation (Average)
------------------------------
Accuracy : 1.0000
Precision: 1.0000
Recall   : 1.0000
F1-score : 1.0000
```

Task 5.1.3 - Regression Evaluation

- Now evaluate a trained `Linear Regression` and `SVM` model for the Regression task `Diabetes`. Report on the performance metrics: MSE, RMSE, MAE, R^2. **(RESULT)**

In [19]:
```
# regression

data = load_diabetes()
X = data.data
Y = data.target

# Scaling the data
scaler = StandardScaler()
X = scaler.fit_transform(X, Y)
```

```
# 80/20 train-test split (not stratified, because regression)
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

# Train the Models
# Linear (instead of logistic) regression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
# SVM (but for regression)
svm_reg = SVR(kernel="linear")
svm_reg.fit(X_train, y_train)


# predictions
lin_pred = lin_reg.predict(X_test)
svm_pred = svm_reg.predict(X_test)

# metrics
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

# root mean squared error
def rmse(y_true, y_pred):
    return np.sqrt(mse(y_true, y_pred))

# mean absolute error
def mae(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))

def r2(y_true, y_pred):
    return 1 - np.sum((y_true - y_pred) ** 2) / np.sum((y_true - np.mean(y_true)) *

# Evaluation
def evaluate_model(y_true, y_pred, name="Model"):
    mse_val = mse(y_true, y_pred)
    rmse_val = rmse(y_true, y_pred)
    mae_val = mae(y_true, y_pred)
    r2_val = r2(y_true, y_pred)

    print(f"\n{name} Evaluation")
    print("----------------------------")
    print(f"MSE : {mse_val:.4f}")
    print(f"RMSE: {rmse_val:.4f}")
    print(f"MAE : {mae_val:.4f}")
    print(f"R2  : {r2_val:.4f}")

evaluate_model(y_test, lin_pred, "Linear Regression")
evaluate_model(y_test, svm_pred, "SVM Regression")

print()
print("Reuslt: Linear regression performs better than SVM regression,\neven though
```

```
Linear Regression Evaluation
----------------------------
MSE : 2900.1936
RMSE: 53.8534
MAE : 42.7941
R2  : 0.4526

SVM Regression Evaluation
----------------------------
MSE : 2938.4619
RMSE: 54.2076
MAE : 43.3242
R2  : 0.4454

Reuslt: Linear regression performs better than SVM regression,
even though SVM was better for classification.
```

Task 5.1.4 - Cross-Validation (BONUS)

- Set up a cross-validation pipeline for the `Linear Regression` and `SVM` models on the `Diabetes` dataset. (Regression) **(RESULT)**
- Set up a cross-validation pipeline for the `Logistic Regression` and `SVM` models on the `Iris` dataset. (Classification) **(RESULT)**
- Report the performance metrics on all folds (minimum 5-fold) for each model and dataset. **(RESULT)**

In [ ]:
```python
# Cross-validation means doing many different train-test splits

# Most code copied

def cross_validate_diabetes(X, Y, num_folds=5):
    # Scaling the data
    scaler = StandardScaler()
    X = scaler.fit_transform(X, Y)

    _random_state = 42  # fixed seed for reproducibility

    for fold in range(num_folds):
        random_state = fold * 10 + _random_state

        # 80/20 train-test split (not stratified, because regression)
        X_train, X_test, y_train, y_test = train_test_split(
            X, Y, test_size=0.2, random_state=random_state
        )

        # Train the Models
        # Linear (instead of logistic) regression
        lin_reg = LinearRegression()
        lin_reg.fit(X_train, y_train)
        # SVM (but for regression)
        svm_reg = SVR(kernel="linear")
        svm_reg.fit(X_train, y_train)
```

```
        # predictions
        lin_pred = lin_reg.predict(X_test)
        svm_pred = svm_reg.predict(X_test)

        # using metrics defined above

        # Evaluation
        def evaluate_model(y_true, y_pred, name="Model"):
            mse_val = mse(y_true, y_pred)
            rmse_val = rmse(y_true, y_pred)
            mae_val = mae(y_true, y_pred)
            r2_val = r2(y_true, y_pred)

            print(f"\n{name} Evaluation")
            print("----------------------------")
            print(f"MSE : {mse_val:.4f}")
            print(f"RMSE: {rmse_val:.4f}")
            print(f"MAE : {mae_val:.4f}")
            print(f"R2  : {r2_val:.4f}")

        evaluate_model(y_test, lin_pred, f"Linear Regression Fold {fold+1}")
        evaluate_model(y_test, svm_pred, f"SVM Regression Fold {fold+1}")

cross_validate_diabetes(X, Y, num_folds=5)
```

```
Linear Regression Fold 1 Evaluation
----------------------------
MSE : 0.0641
RMSE: 0.2532
MAE : 0.1969
R2  : 0.7271

SVM Regression Fold 1 Evaluation
----------------------------
MSE : 0.0687
RMSE: 0.2620
MAE : 0.1980
R2  : 0.7077

Linear Regression Fold 2 Evaluation
----------------------------
MSE : 0.0569
RMSE: 0.2384
MAE : 0.1854
R2  : 0.7474

SVM Regression Fold 2 Evaluation
----------------------------
MSE : 0.0574
RMSE: 0.2395
MAE : 0.1811
R2  : 0.7452

Linear Regression Fold 3 Evaluation
----------------------------
MSE : 0.0540
RMSE: 0.2324
MAE : 0.1800
R2  : 0.7721

SVM Regression Fold 3 Evaluation
----------------------------
MSE : 0.0599
RMSE: 0.2448
MAE : 0.1872
R2  : 0.7472

Linear Regression Fold 4 Evaluation
----------------------------
MSE : 0.0912
RMSE: 0.3019
MAE : 0.2278
R2  : 0.6119

SVM Regression Fold 4 Evaluation
----------------------------
MSE : 0.0853
RMSE: 0.2921
MAE : 0.2196
R2  : 0.6369
```

```
Linear Regression Fold 5 Evaluation
----------------------------
MSE : 0.0487
RMSE: 0.2208
MAE : 0.1738
R2  : 0.7807

SVM Regression Fold 5 Evaluation
----------------------------
MSE : 0.0542
RMSE: 0.2328
MAE : 0.1748
R2  : 0.7560
```

In [26]:
```python
def cross_validate_iris(X, Y, num_folds=5):
    # Scaling the data
    scaler = StandardScaler()
    X = scaler.fit_transform(X, Y)

    _random_state = 42  # fixed seed for reproducibility

    for fold in range(num_folds):
        random_state = fold * 10 + _random_state

        # 80/20 train-test split (not stratified, because regression)
        X_train, X_test, y_train, y_test = train_test_split(
            X, Y, test_size=0.2, random_state=random_state
        )

        # Train the Models(Logistic Regression and SVM)
        # Logistic Regression
        log_reg = LogisticRegression(max_iter=500)
        log_reg.fit(X_train, y_train)
        # SVM
        svm_clf = SVC(kernel="linear")
        svm_clf.fit(X_train, y_train)

        # predictions
        log_pred = log_reg.predict(X_test)
        svm_pred = svm_clf.predict(X_test)

        def confusion_matrix_per_class(y_true, y_pred, num_classes):
            """
            Confusion matrix as above for each class but in a one-vs-rest manner.
            """
            matrices = []
            for cls in range(num_classes):
                TP = np.sum((y_true == cls) & (y_pred == cls))
                TN = np.sum((y_true != cls) & (y_pred != cls))
                FP = np.sum((y_true != cls) & (y_pred == cls))
                FN = np.sum((y_true == cls) & (y_pred != cls))
                matrices.append((TP, TN, FP, FN))
            return matrices

        def evaluate_model(y_true, y_pred, num_classes, name="Model"):
            matrices = confusion_matrix_per_class(y_true, y_pred, num_classes)
```

```
            precisions, recalls, f1s = [], [], []

            for TP, TN, FP, FN in matrices:
                prec = TP / (TP + FP + 1e-12)
                rec = TP / (TP + FN + 1e-12)
                f1 = 2 * prec * rec / (prec + rec + 1e-12)
                precisions.append(prec)
                recalls.append(rec)
                f1s.append(f1)

            accuracy = np.mean(y_true == y_pred)
            precision = np.mean(precisions)
            recall = np.mean(recalls)
            f1 = np.mean(f1s)

            print(f"\n{name} Fold {fold+1} Evaluation (Average)")
            print("-"*30)
            print(f"Accuracy : {accuracy:.4f}")
            print(f"Precision: {precision:.4f}")
            print(f"Recall   : {recall:.4f}")
            print(f"F1-score : {f1:.4f}")

        num_classes = len(np.unique(Y))
        evaluate_model(y_test, log_pred, num_classes, name="Logistic Regression")
        evaluate_model(y_test, svm_pred, num_classes, name="SVM")


cross_validate_iris(X, Y, num_folds=5)
```

```
Logistic Regression Fold 1 Evaluation (Average)
------------------------------
Accuracy : 0.9737
Precision: 0.9742
Recall   : 0.9697
F1-score : 0.9719

SVM Fold 1 Evaluation (Average)
------------------------------
Accuracy : 0.9561
Precision: 0.9516
Recall   : 0.9556
F1-score : 0.9535

Logistic Regression Fold 2 Evaluation (Average)
------------------------------
Accuracy : 0.9912
Precision: 0.9934
Recall   : 0.9872
F1-score : 0.9902

SVM Fold 2 Evaluation (Average)
------------------------------
Accuracy : 0.9737
Precision: 0.9682
Recall   : 0.9738
F1-score : 0.9709

Logistic Regression Fold 3 Evaluation (Average)
------------------------------
Accuracy : 0.9825
Precision: 0.9861
Recall   : 0.9773
F1-score : 0.9813

SVM Fold 3 Evaluation (Average)
------------------------------
Accuracy : 0.9825
Precision: 0.9861
Recall   : 0.9773
F1-score : 0.9813

Logistic Regression Fold 4 Evaluation (Average)
------------------------------
Accuracy : 0.9649
Precision: 0.9673
Recall   : 0.9581
F1-score : 0.9623

SVM Fold 4 Evaluation (Average)
------------------------------
Accuracy : 0.9474
Precision: 0.9482
Recall   : 0.9394
F1-score : 0.9435
```

```
Logistic Regression Fold 5 Evaluation (Average)
------------------------------
Accuracy : 0.9912
Precision: 0.9935
Recall   : 0.9868
F1-score : 0.9901

SVM Fold 5 Evaluation (Average)
------------------------------
Accuracy : 0.9737
Precision: 0.9810
Recall   : 0.9605
F1-score : 0.9698
```

Congratz, you made it! :)