ASSIGNMENT 3

TASK 1. Tutorial that explains 'Complementary Slackness' condition.

Complementary Slackness

Complementary Slackness is a fundamental concept in linear programming and duality theory that helps us comprehend the relationship between primary (maximization) and secondary (minimization) problems. It essentially tells us how the ideal solutions to two problems that are linked and how resources are optimally distributed.

EXAMPLE:

Consider a product company that makes both mobile phones (X) and earphones (Y). The company's purpose is to maximize daily profit while adhering to labor and material consumption limits. The goal is to determine the optimal X and Y manufacturing amounts.

Primal Problem (Maximization):

In this scenario, the company wants to maximize its profit (P) by choosing what many mobile phones (X) and earphones (Y) to produce yet maintaining certain limits in account. Labor hours and material consumption are instances of these restrictions.

Maximize Profit: P=5X+3Y

> *Labor Constraint: 2X+3Y<=12 hours*
>
> - The labor constraint: 2X + 3Y <= 12 hours means that the company can't exceed 12 hours of labor in total.
>
> *Material Constraint: X+Y≤6 units*
>
> - The material constraint: X + Y <= 6 units means that they can't use more than 6 units of material.
>
> *Non-negativity Constraint: X>=0,Y>=0*

> - The goal is to determine the optimal values of X and Y while maximizing profit.

Dual Problem (Minimization): The company tries to minimize its costs (C), connected with labor and material requirement in the dual issue, subject to dual limitations, which are related to the resources (labor and material) available:

Minimize Cost: C=12A+6B

> *Labor Dual Constraint: 2A+B>=5 (associated with the labor constraint)*
>
> *Material Dual Constraint: 3A+B>=3 (associated with the material constraint)*
>
> *Non-negativity Constraint: A>=0,B>=0*

Complementary Slackness condition:

- If X (quantity of mobile phones) is produced in the primal problem:

  > In the primary context, X represents the quantity of mobile phones produced, with associated labor hours. To ensure labor hours don't exceed 12, a constraint is set as 2X + 3Y <= 12, and when X is produced, there must be "zero slack" (s1 = 0), meaning all 12 labor hours are fully utilized for phone production, emphasizing labor as the limiting factor.

- If Y (quantity of earphones) is produced in the primal problem:

  > Additionally, in the fundamental problem, Y denotes the number of earbuds manufactured. If the corporation decides to make earbuds, it signifies that they will use material units to do so. The material restriction of X + Y <= 6 units ensures that the total number of material units used in production does not exceed 6. If Y is generated, the material restriction should have no slack (s2 = 0).
  >
  > In the material constraint, "zero slack" means that the LHS (X + Y) equals 6 units, meaning that all available material units are fully employed in the creation of earphones (Y). This means that the material resource is a limiting restriction; they produce earbuds using all available material units.

- If the labor dual variable A is positive:

> The dual variable A in the dual problem corresponds to the labor constraint in the primal problem (2X + 3Y ≤ 12 hours). A positive value of A indicates a willingness to pay more for extra hours.
>
> The labor restriction in the primal problem (2X + 3Y <= 12 hours) should have no slack (s1 = 0). This signifies that all available labor hours are completely utilized because A (the labor price) is positive and the employer values more work hours. In other words, labor is a restricting constraint, and they are prepared to invest more to efficiently employ all available labor hours.

- If the material dual variable B is positive:

> The dual variable B in the dual problem is associated with the resource constraint from the primal problem (X + Y <= 6 units). When B has a positive value in the dual problem, it signifies the company's willingness to offer a higher price for additional material units.
>
> In the primordial problem, the material constraint (X + Y <= 6 units) should have no slack (s2 = 0). This signifies that all available material units have been fully utilized because B (the material price) is positive and the corporation values extra material units. In other words, material is a restricting constraint, and they are ready to spend more to make efficient use of all available materials.

This condition assures that the best solutions to the primary and dual problems are complimentary and consistent.

In [104…
```
# Import libraries
import numpy as np
```

Primal problem

In [105…
```
# Mean Accuracy (K-Fold Cross-Validation)
from scipy.optimize import linprog
primal_objective_coefficients = [-5, -3]
primal_constraint_coefficients = [
    [2, 3],
    [1, 1]
]
primal_constraint_rhs = [12, 6]
x_variable_bounds = (0, None)
y_variable_bounds = (0, None)
primal_result = linprog(primal_objective_coefficients, A_ub=primal_constrain

# Optimal solution for the primal problem
X_optimal, Y_optimal = primal_result.x
P_optimal = -primal_result.fun
```

Dual Problem

In [106…
```
dual_objective_coefficients = [12, 6]
dual_constraint_coefficients = [
    [-2, -1],
    [-3, -1]]
dual_constraint_rhs = [-5, -3]
a_variable_bounds = (0, None)
b_variable_bounds = (0, None)
dual_result = linprog(dual_objective_coefficients, A_ub=dual_constraint_coef

# Optimal solution for the dual problem
A_dual_optimal, B_dual_optimal = dual_result.x
C_optimal = dual_result.fun
```

In [107…
```
complementary_slackness_condition = np.dot(dual_result.fun, primal_result.fu
```

In [108…
```
print("Primal Optimal Solution (X*, Y*):", (X_optimal, Y_optimal))
print("Primal Objective Value (P*):", P_optimal)
print("Dual Optimal Solution (A*, B*):", (A_dual_optimal, B_dual_optimal))
print("Dual Objective Value (C*):", C_optimal)
print("Complementary Slackness Condition: P* * C* =", complementary_slacknes
```

```
Primal Optimal Solution (X*, Y*): (6.0, 0.0)
Primal Objective Value (P*): 30.0
Dual Optimal Solution (A*, B*): (0.0, 5.0)
Dual Objective Value (C*): 30.0
Complementary Slackness Condition: P* * C* = -900.0
```

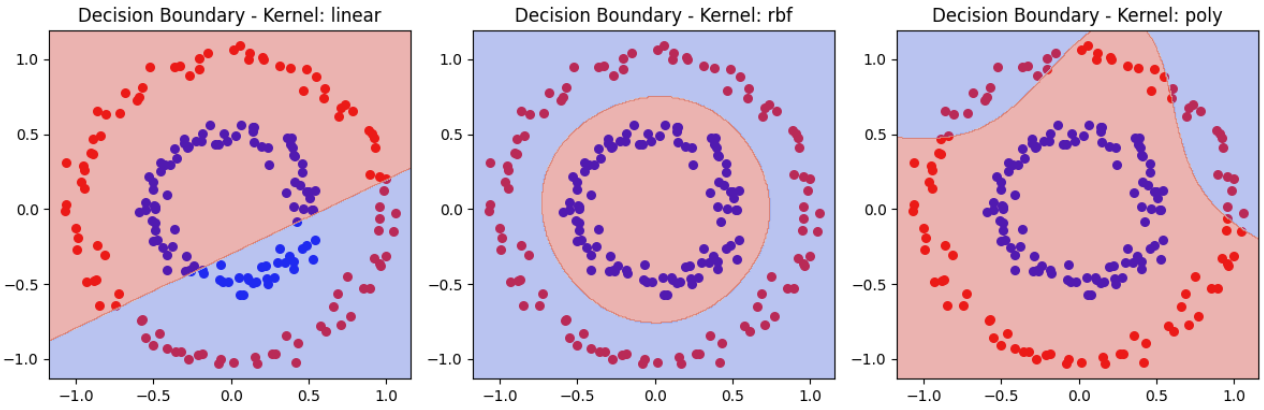
TASK 2

Support Vector Machine

In [109…
```
from sklearn.datasets import make_circles, make_moons
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

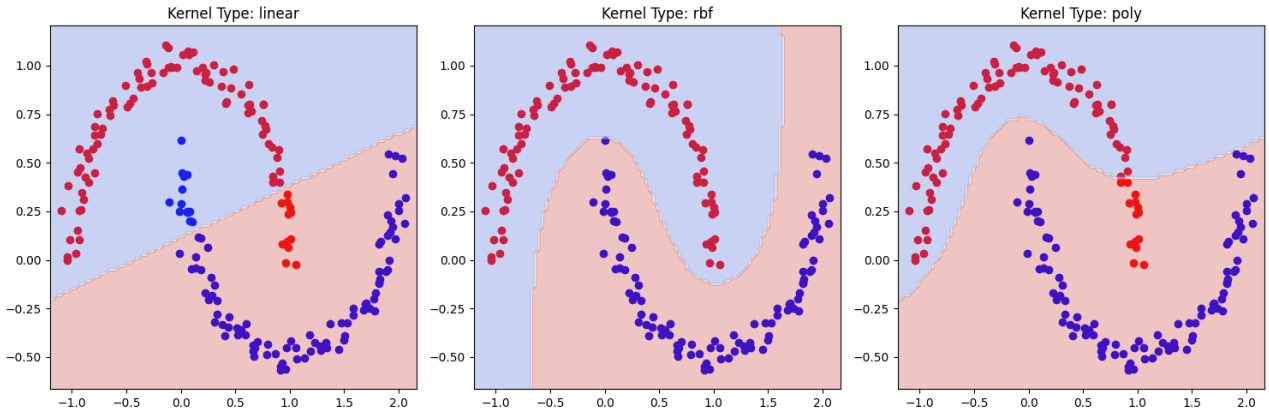
Dataset: make_circles

In [122…
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

# Generate the dataset
X_circles, y_circles = make_circles(n_samples=200, noise=0.05, factor=0.5, r

# List of kernel types to try
kernel_types = ['linear', 'rbf', 'poly']

# Create subplots for decision boundary visualization
fig, decision_boundary_plots = plt.subplots(1, len(kernel_types), figsize=(1

for i, kernel_type in enumerate(kernel_types):
    svm_classifier = SVC(kernel=kernel_type, C=1.0)
    svm_classifier.fit(X_circles, y_circles)

    # Separate data points by class
    decision_boundary_plots[i].scatter(X_circles[y_circles == 0][:, 0], X_ci
    decision_boundary_plots[i].scatter(X_circles[y_circles == 1][:, 0], X_ci

    # Define mesh grid
    x_min, x_max = X_circles[:, 0].min() - 0.1, X_circles[:, 0].max() + 0.1
    y_min, y_max = X_circles[:, 1].min() - 0.1, X_circles[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300), np.linspace(y_min,

    # Predict and plot the decision boundary
    Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    decision_boundary_plots[i].contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alp
    decision_boundary_plots[i].set_title(f"Decision Boundary - Kernel: {kern

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



In [136…

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

# Generate the make_moons dataset
features, labels = make_moons(n_samples=200, noise=0.05, random_state=42)

kernel_types = ['linear', 'rbf', 'poly']

fig, subplots = plt.subplots(1, len(kernel_types), figsize=(15, 5))

for i, kernel_type in enumerate(kernel_types):
    # Create and train the Support Vector Machine (SVM) model
    support_vector_classifier = SVC(kernel=kernel_type, C=1.0)
    support_vector_classifier.fit(features, labels)

    # Plot the data points with the specified colors and no labels for the l
    subplots[i].scatter(features[labels == 0][:, 0], features[labels == 0][:
    subplots[i].scatter(features[labels == 1][:, 0], features[labels == 1][:

    # Create a grid to visualize the decision boundary
    x_min, x_max = features[:, 0].min() - 0.1, features[:, 0].max() + 0.1
    y_min, y_max = features[:, 1].min() - 0.1, features[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,

    # Predict the values for each point in the grid
    decision_values = support_vector_classifier.predict(np.c_[xx.ravel(), yy
    decision_values = decision_values.reshape(xx.shape)

    # Plot the decision boundary
    subplots[i].contourf(xx, yy, decision_values, cmap=plt.cm.coolwarm, alph
    subplots[i].set_title(f"Kernel Type: {kernel_type}")

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



In [140…

```
moons_features, moons_labels = make_moons(n_samples=200, noise=0.05, random_
circles_features, circles_labels = make_circles(n_samples=200, factor=0.5, n

datasets = {
    'Moons': (moons_features, moons_labels),
    'Circles': (circles_features, circles_labels)
}

kernel_types = ['linear', 'rbf', 'poly']
num_folds = 5  # Number of folds for cross-validation

for dataset_name, (dataset_features, dataset_labels) in datasets.items():
    print(f"Dataset: {dataset_name}")
    print()
    for kernel_type in kernel_types:
        print(f"Kernel Type: {kernel_type}")

        # Create and train the Support Vector Machine (SVM) model
        support_vector_classifier = SVC(kernel=kernel_type, C=1.0)

        # Perform k-fold cross-validation
        cross_validation_scores = cross_val_score(support_vector_classifier,

        print(f"Cross-Validation Scores: {cross_validation_scores}")
        print()
```

```
Dataset: Moons

Kernel Type: linear
Cross-Validation Scores: [0.95  0.9   0.825 0.825 0.875]

Kernel Type: rbf
Cross-Validation Scores: [1. 1. 1. 1. 1.]

Kernel Type: poly
Cross-Validation Scores: [0.975 0.95  0.85  0.9   0.95 ]

Dataset: Circles

Kernel Type: linear
Cross-Validation Scores: [0.475 0.55  0.55  0.375 0.475]

Kernel Type: rbf
Cross-Validation Scores: [1. 1. 1. 1. 1.]

Kernel Type: poly
Cross-Validation Scores: [0.625 0.65  0.625 0.575 0.575]
```

In [111…

Find best kernel function:

The RBF (Radial Basis Function) kernel emerges as the optimal choice for handling both the Half-Circles and Moons datasets due to its remarkable ability to cope with complex, non-linear patterns in the data.

- Circles Dataset:

> In the case of the Circles dataset, the RBF kernel demonstrates its prowess. This dataset inherently exhibits a non-linear boundary – it forms two half circles that are interconnected. Trying to separate these half circles using a straight line would be an arduous task. However, the RBF kernel excels precisely in these situations. It can adapt to the data's non-linearities and create decision regions that are circular or elliptical in shape. It's as if the RBF kernel molds itself around the data, resembling the way you might draw a circle to encompass these half circles. The result is an accurate and well-defined decision boundary that effectively distinguishes the inner circle from the outer one.

- Moons Dataset:

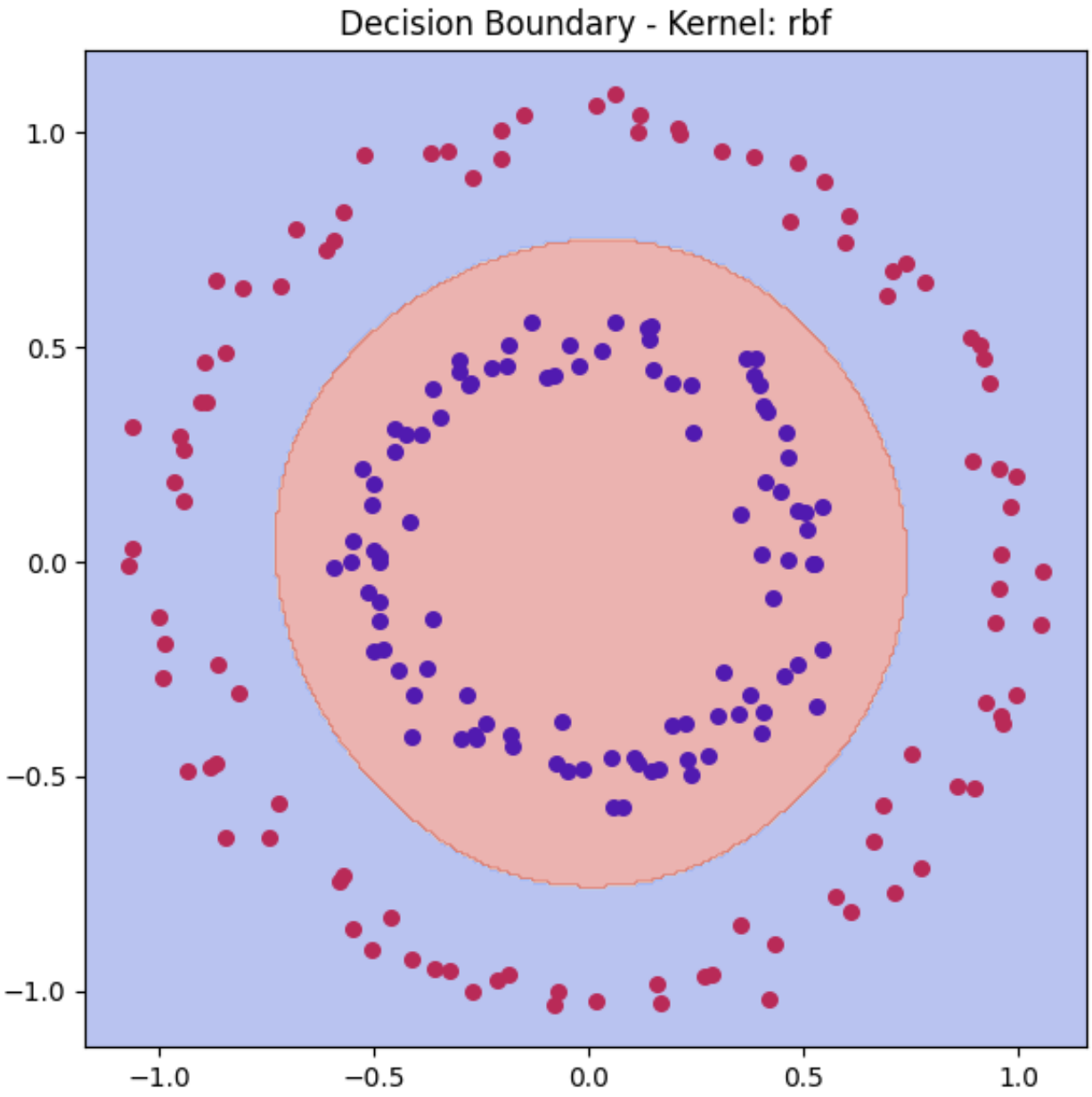
> The RBF kernel appears as the best match when dealing with the Moons dataset, which presents a complex scenario of two interlocking half circles. The Moons dataset is likewise a non-linear pattern. The RBF kernel's strength in representing non-linear decision boundaries is highlighted here. It's as if it can follow the contour of the data instinctively, capturing the delicate shape of the moons. This accurately differentiates the two classes, resulting in a clear and precise decision boundary that closely corresponds to the data's curves and patterns.

The RBF kernel is the best choice for both the Half-Circles and Moons datasets due to its adaptability to complicated, non-linear data patterns, capacity to bend decision boundaries according to the structure of the data, and efficacy in collecting subtle features. It embodies the essence of machine learning: the ability to comprehend and model complicated relationships within data to produce good and accurate predictions

Visualization

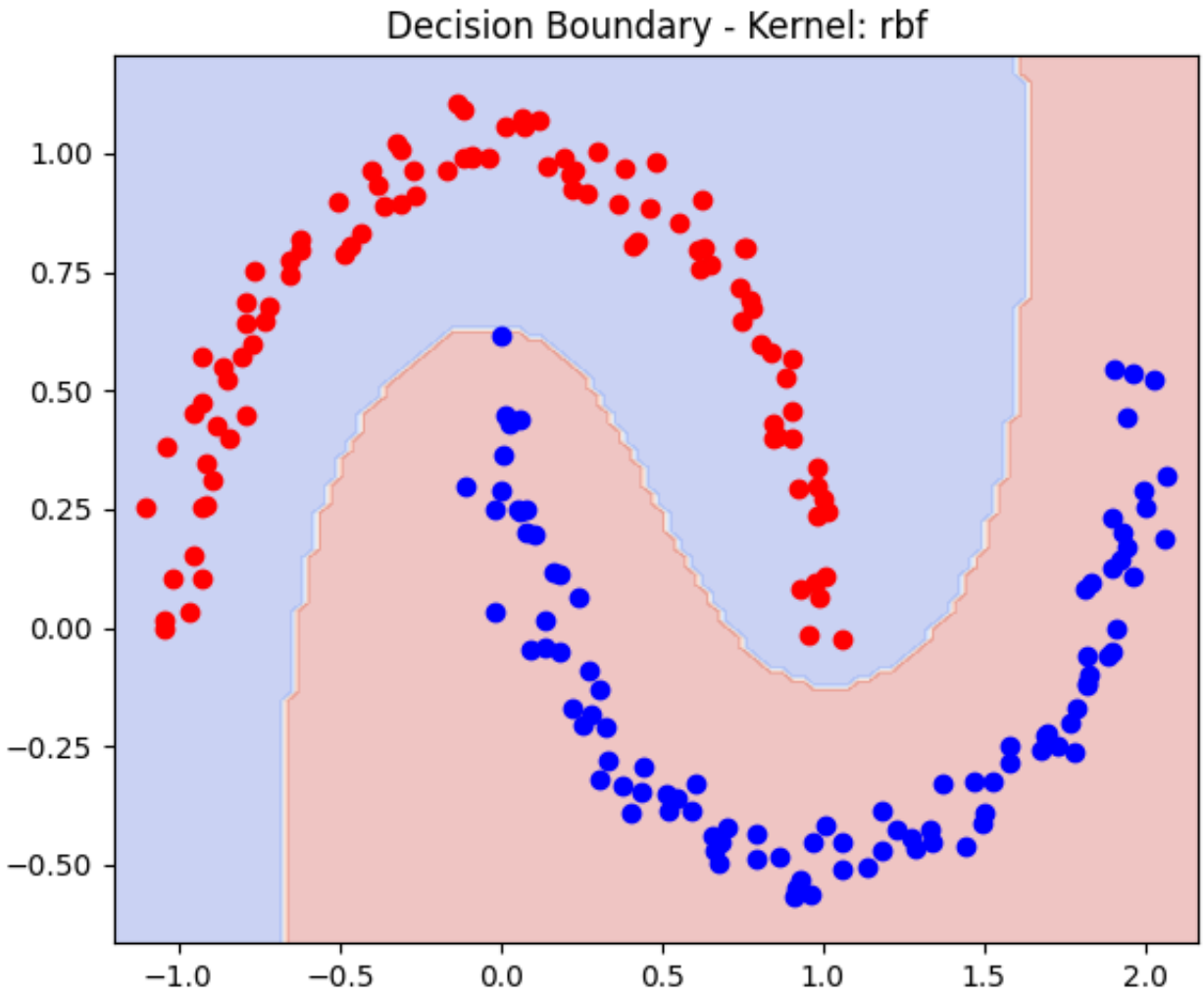
In [153…

```
X_circles, y_circles = make_circles(n_samples=200, noise=0.05, factor=0.5, r

# Create subplots for decision boundary visualization
fig, decision_boundary_plot = plt.subplots(figsize=(6, 6), facecolor='white'

# Create and train the SVM model with the 'rbf' kernel
svm_classifier = SVC(kernel='rbf', C=1.0)
svm_classifier.fit(X_circles, y_circles)

# Separate data points by class
decision_boundary_plot.scatter(X_circles[y_circles == 0][:, 0], X_circles[y_
decision_boundary_plot.scatter(X_circles[y_circles == 1][:, 0], X_circles[y_

# Define mesh grid
x_min, x_max = X_circles[:, 0].min() - 0.1, X_circles[:, 0].max() + 0.1
y_min, y_max = X_circles[:, 1].min() - 0.1, X_circles[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300), np.linspace(y_min, y_ma

# Predict and plot the decision boundary
Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
decision_boundary_plot.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.4)
decision_boundary_plot.set_title("Decision Boundary - Kernel: rbf")

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


Decision Boundary - Kernel: rbf
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

In [156…

```
features, labels = make_moons(n_samples=200, noise=0.05, random_state=42)

# Create and train the Support Vector Machine (SVM) model with the 'rbf' ker
support_vector_classifier = SVC(kernel='rbf', C=1.0)
support_vector_classifier.fit(features, labels)

# Create a grid to visualize the decision boundary
x_min, x_max = features[:, 0].min() - 0.1, features[:, 0].max() + 0.1
y_min, y_max = features[:, 1].min() - 0.1, features[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_ma
decision_values = support_vector_classifier.predict(np.c_[xx.ravel(), yy.rav
decision_values = decision_values.reshape(xx.shape)
plt.figure(figsize=(6, 5))
plt.contourf(xx, yy, decision_values, cmap=plt.cm.coolwarm, alpha=0.3)
plt.title("Decision Boundary - Kernel: rbf")
plt.scatter(features[labels == 0][:, 0], features[labels == 0][:, 1], color=
plt.scatter(features[labels == 1][:, 0], features[labels == 1][:, 1], color=
plt.tight_layout()
plt.show()
```



The RBF kernel is a flexible option in SVMs because it can create a decision boundary that successfully captures and isolates complex, non-linear clusters of data points by giving greater similarity values to points close by and lower values to those that are farther apart. This versatility enables the SVM to model and categorize subtle patterns in data, making it a valuable tool for a wide range of classification problems.

In [112…

TASK 3 : MLP to solve the classification

MLP

In [157…
```
from sklearn.neural_network import MLPClassifier
```

Dataset Moon

In [158…
```
X_moons, y_moons = make_moons(n_samples=100, noise=0.3, random_state=42)

X_train_moons, X_test_moons, y_train_moons, y_test_moons = train_test_split(

mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=10000

mlp_classifier.fit(X_train_moons, y_train_moons)

y_pred_moons = mlp_classifier.predict(X_test_moons)

# Calculate the performance
accuracy_moons = accuracy_score(y_test_moons, y_pred_moons)
print(f"Accuracy for 'make_moons' dataset: {accuracy_moons * 100:.2f}%")
```

```
Accuracy for 'make_moons' dataset: 80.00%
```

Dataset: Half_circles

In [159…

```
X_circles, y_circles = make_circles(n_samples=100, noise=0.2, factor=0.5, ra

X_train_circles, X_test_circles, y_train_circles, y_test_circles = train_tes

mlp_classifier_circles = MLPClassifier(hidden_layer_sizes=(100, 100), max_it

mlp_classifier_circles.fit(X_train_circles, y_train_circles)

y_pred_circles = mlp_classifier_circles.predict(X_test_circles)

# Calculate the performance
accuracy_circles = accuracy_score(y_test_circles, y_pred_circles)
print(f"Accuracy for 'make_circles' dataset: {accuracy_circles * 100:.2f}%")
```

```
Accuracy for 'make_circles' dataset: 85.00%
```

Training process:

- Firsly we have imported dataset from sklearn called "make_moons" and "make_circles. The make_circles have data looked like circular cluster and the other is the moon shaped cluster. Here, we are supposed to perform MLP classification.
- To enable MLP classification, the dataset is initially partitioned into training and testing subsets with an 80:20 ratio using the train_test_split library.
- Next step is to decide the Machine learning model: MLP classifier. The MLP classifier is basically a neural network model which performs feed forward operations with more than one hidden layers.
- For the classifier above, there are hundred to hundred hidden layers, each 100 among two hidden layers, depening upon the how complex the dataset is.
- Number of iterations are kept to a thousand, the more the number of iterations, the more the model is trained upto the maximum.
- Basically, the training process is done only using the training dataset added to the fit method in the model, here the model optimizes the biases values, weight values and reduces the loss on each iteration. This way te model is improved for the classification. This procedure is trained until the last iteration.
- Next step, is to prdict and evaluate accuracy for the model. The trained model can be used for prediction using the testing dataset under predict model.
- We use a function that would evaluate the accuracy or we can use the accuracy library to know how well the model's performace is based on the classification

- To visualize the prediciton, we use decision boundry, a mesh grid covering the whole feature space is generated, and the model's predictions are obtained for each point in the grid.
- The decision boundary is shown graphically, with different colors marking different class regions. On the graph, the original data points are also shown to show their distribution in the feature space.

Basically, the goal of this training method is to create and test an MLP classifier for binary classification on the'make_moons' dataset. The architecture used, including how many of layers that are hidden and neurons per layers and the highest amount of iterations, are affected by the complexity of the task and can be changed through testing. The accuracy score gives an objective evaluation of how well the model performs, and the decision boundary representation provides an intuitive comprehension of the model's classification areas.

K FOLD CROSS VALIDATION and COMPUTATIONAL COST

SVM Model

In [116…
```
from sklearn.model_selection import train_test_split, KFold,cross_val_score
import time
```

In [162…

```
def perform_kfold_cross_validation(features, labels, kernel, num_folds=5):
    start_time = time.time()
    support_vector_classifier = SVC(kernel=kernel, C=1.0)
    cross_validation_scores = cross_val_score(support_vector_classifier, fea

    mean_accuracy = np.mean(cross_validation_scores)

    end_time = time.time()
    computational_cost = end_time - start_time

    return mean_accuracy, computational_cost


half_circles_features, half_circles_labels = make_circles(n_samples=200, noi
moons_features, moons_labels = make_moons(n_samples=200, noise=0.05, random_

datasets = {
    'Half Circles': (half_circles_features, half_circles_labels),
    'Moons': (moons_features, moons_labels)
}

kernel_type = 'rbf'
num_folds = 5

for dataset_name, (dataset_features, dataset_labels) in datasets.items():
    mean_accuracy, computational_cost = perform_kfold_cross_validation(datas
    print(f"Dataset: {dataset_name}, Kernel: {kernel_type}")
    print(f"Mean Accuracy: {mean_accuracy}")
    print(f"K-Fold Cross-Validation Scores: {cross_val_score(SVC(kernel=kern
    print(f"Computational Cost: {computational_cost} seconds")
    print()
```

```
Dataset: Half Circles, Kernel: rbf
Mean Accuracy: 1.0
K-Fold Cross-Validation Scores: [1. 1. 1. 1. 1.]
Computational Cost: 0.019808530807495117 seconds

Dataset: Moons, Kernel: rbf
Mean Accuracy: 1.0
K-Fold Cross-Validation Scores: [1. 1. 1. 1. 1.]
Computational Cost: 0.018218040466308594 seconds
```

MLP Model

In [164…

```
def perform_kfold_cross_validation(features, labels, num_folds=5):
    start_time = time.time()

    mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=1

    cross_validation_scores = cross_val_score(mlp_classifier, features, labe

    mean_accuracy = np.mean(cross_validation_scores)

    end_time = time.time()
    computational_cost = end_time - start_time

    return mean_accuracy, computational_cost

half_circles_features, half_circles_labels = make_circles(n_samples=200, noi

moons_features, moons_labels = make_moons(n_samples=200, noise=0.05, random_

datasets = {
    'Half Circles': (half_circles_features, half_circles_labels),
    'Moons': (moons_features, moons_labels)
}

num_folds = 5

for dataset_name, (dataset_features, dataset_labels) in datasets.items():
    mean_accuracy, computational_cost = perform_kfold_cross_validation(datas
    print(f"Dataset: {dataset_name}, MLP")
    print(f"Mean Accuracy: {mean_accuracy}")
    print(f"K-Fold Cross-Validation Scores: {cross_val_score(MLPClassifier(h
    print(f"Computational Cost: {computational_cost} seconds")
    print()
```

```
Dataset: Half Circles, MLP
Mean Accuracy: 1.0
K-Fold Cross-Validation Scores: [1. 1. 1. 1. 1.]
Computational Cost: 3.7543275356292725 seconds

Dataset: Moons, MLP
Mean Accuracy: 1.0
K-Fold Cross-Validation Scores: [1.    0.975 1.    1.    1.   ]
Computational Cost: 4.7500810623168945 seconds
```

In the comparative analysis of SVM and MLP models applied to the "Half Circles" and "Moons" datasets through k-fold cross-validation, both models demonstrate strong performance with high accuracy levels.

SVM models, in particular, exhibit near-perfect mean accuracies for both datasets, indicating their effectiveness in segregating data points.

Notably, SVMs boast a significantly lower computational cost, taking only approximately 0.02 seconds for each dataset.

Alternatively, MLP models also deliver good accuracy but come with a higher computational overhead, with execution times of approximately 3.75 seconds for the "Half Circles" dataset and 4.7 seconds for the "Moons" dataset.

When selecting between these models, it is essential to consider factors such as the available computational resources, the need for interpretability, and the desired level of accuracy. SVMs stand out for their computational efficiency, making them suitable for resource-constrained scenarios. In contrast, MLPs provide enhanced flexibility and adaptability, particularly for addressing intricate and challenging tasks, albeit at the cost of higher computational demands.

In [165…
```
!jupyter nbconvert --to html /content/drive/MyDrive/CS6140_assignment_3_GA.i
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/CS6140_assignment_
3_GA.ipynb to html
[NbConvertApp] Writing 889158 bytes to /content/drive/MyDrive/CS6140_assignm
ent_3_GA.html
```