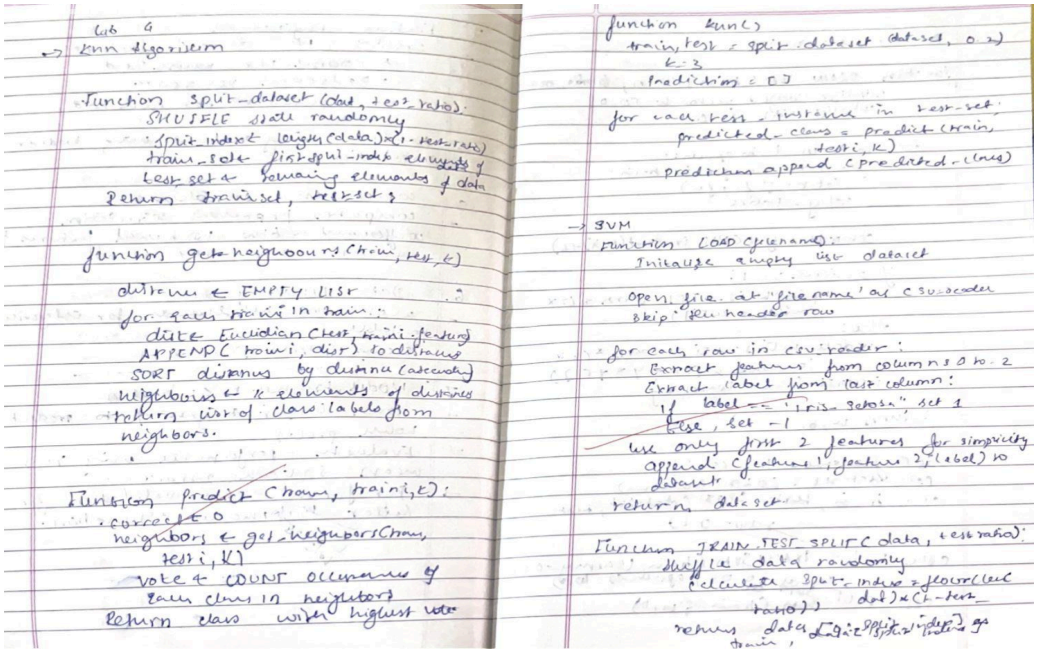
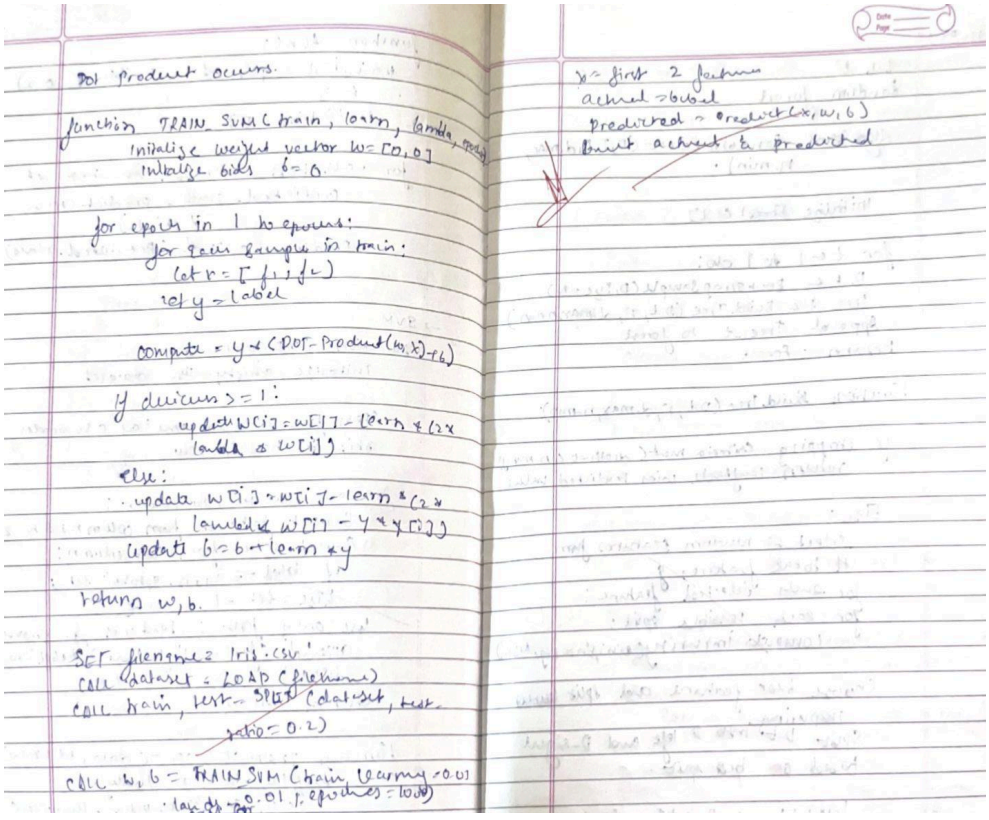
Build KNN Classification model and SVM for a given dataset





Code:

```
import math
```

```
import matplotlib.pyplot as plt

#AGE, RESTING HEARTRATE, isAthlete
data = [
    [25, 55, 1],
    [30, 60, 1],
    [22, 58, 1],
    [35, 85, 0],
    [40, 90, 0],
    [45, 88, 0],
    [25, 60, 1],
    [31, 70, 0],
    [19, 50, 1],
    [40, 80, 0]
]

def euclidean_distance(p1, p2):
    return math.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)

def knn_predict(data, test_point, k=3):
    distances = []
    for point in data:
        dist = euclidean_distance(point, test_point)
        distances.append((dist, point[2]))
    distances.sort(key=lambda x: x[0])
    k_nearest = distances[:k]

    # Voting
    votes = {0: 0, 1: 0}
    for _, label in k_nearest:
        votes[label] += 1
    predicted_class = max(votes, key=votes.get)
    return predicted_class

# Step 4: Predict test point
test_point = [41, 81]
prediction = knn_predict(data, test_point, k=3)
print(f"Predicted Class for point {test_point}: {prediction}")

# Separate data by class for color grading
```

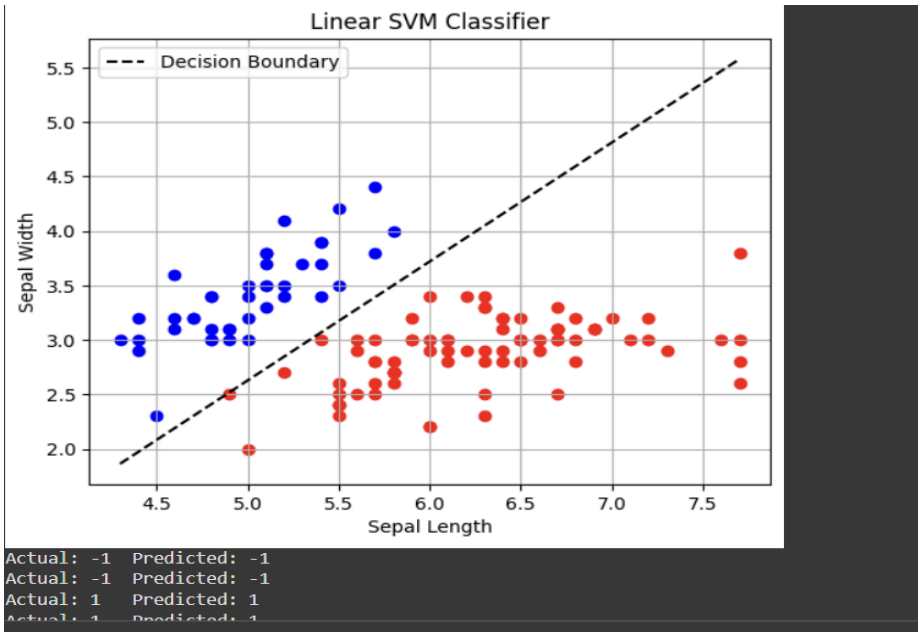
```
class_0 = [point for point in data if point[2] == 0]
class_1 = [point for point in data if point[2] == 1]

# Extract values for plotting
age_0 = [point[0] for point in class_0]
hr_0 = [point[1] for point in class_0]
age_1 = [point[0] for point in class_1]
hr_1 = [point[1] for point in class_1]

# Plot with color grading
plt.figure(figsize=(8,6))
scatter0 = plt.scatter(age_0, hr_0, c='lightcoral', s=120,
label='Non-Athlete (Class 0)', edgecolors='black')
scatter1 = plt.scatter(age_1, hr_1, c='skyblue', s=120, label='Athlete
(Class 1)', edgecolors='black')

plt.xlabel("Age")
plt.ylabel("Resting Heart Rate (bpm)")
plt.title("Athlete Classification Based on Age and Resting HR")
plt.legend()
plt.grid(True)
plt.show()
```

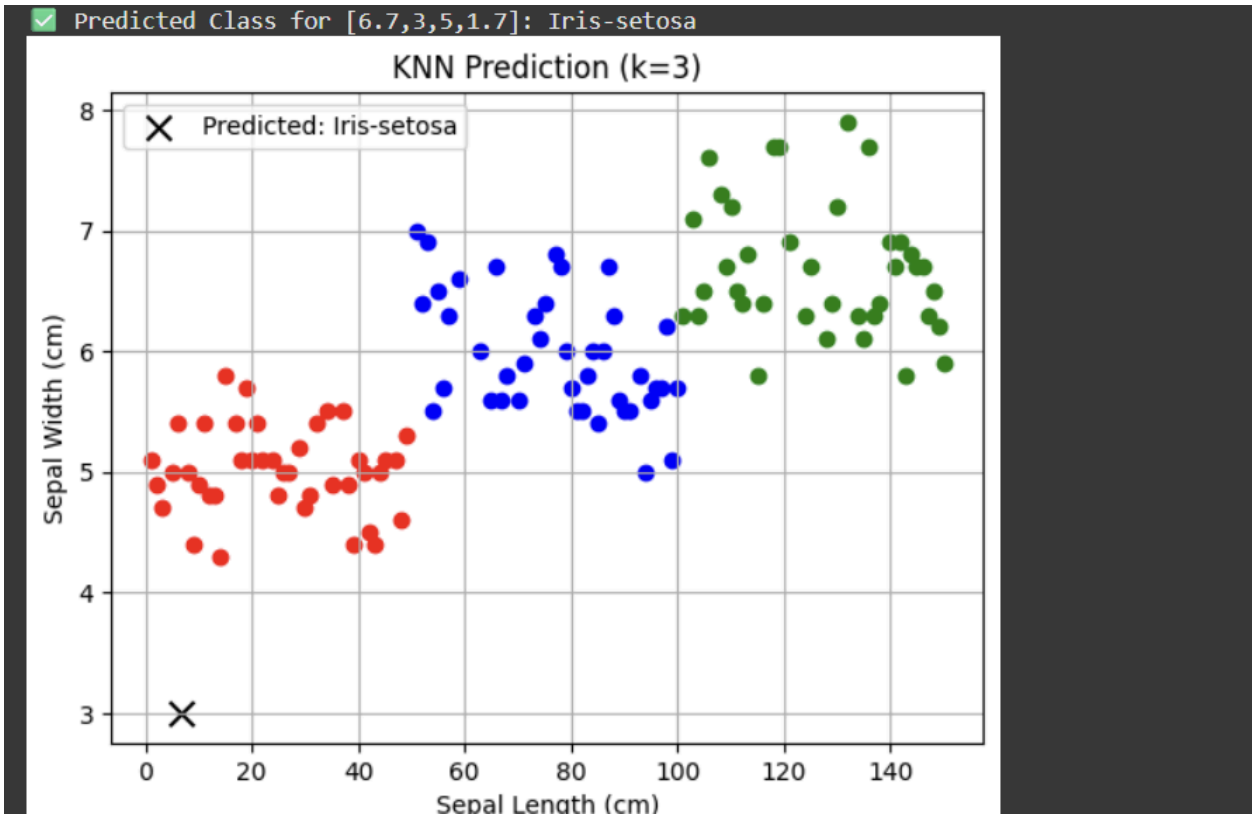
```
import csv
import random
import matplotlib.pyplot as plt

# Load dataset
def load_dataset(filename):
    dataset = []
    with open(filename, 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)  # skip header
        for row in csv_reader:
            features = list(map(float, row[1:-1]))  # Assuming format: ID,
f1, f2, f3, f4, label
            label = 1 if row[-1] == 'Iris-setosa' else -1  # Binary
classification
            dataset.append(features[:2] + [label])  # Only use first 2
features for plotting
```

```
    return dataset

# Split into training and test
def train_test_split(data, test_ratio=0.2):
    random.shuffle(data)
    split_index = int(len(data) * (1 - test_ratio))
    return data[:split_index], data[split_index:]

# SVM Training using SGD
def train_svm(train_set, learning_rate=0.01, lambda_param=0.01,
epochs=1000):
    w = [0, 0]
    b = 0

    for epoch in range(epochs):
        for features in train_set:
            x = features[:2]
            y = features[2]

            condition = y * (dot_product(w, x) + b) >= 1
            if condition:
                w = [w[i] - learning_rate * (2 * lambda_param * w[i]) for
i in range(len(w))]
            else:
                w = [w[i] - learning_rate * (2 * lambda_param * w[i] - y *
x[i]) for i in range(len(w))]
                b = b + learning_rate * y
    return w, b

def dot_product(a, b):
    return sum([a[i] * b[i] for i in range(len(a))])

# Prediction
def predict(x, w, b):
    return 1 if dot_product(w, x) + b >= 0 else -1

# Plotting
def plot_svm(train_set, w, b):
    for point in train_set:
        x1, x2, label = point
```

```
        color = 'blue' if label == 1 else 'red'
        plt.scatter(x1, x2, c=color)

    # Decision boundary
    x_vals = [min(p[0] for p in train_set), max(p[0] for p in train_set)]
    y_vals = [(-w[0] * x - b) / w[1] for x in x_vals]
    plt.plot(x_vals, y_vals, '--k', label='Decision Boundary')

    plt.xlabel("Sepal Length")
    plt.ylabel("Sepal Width")
    plt.title("Linear SVM Classifier")
    plt.grid(True)
    plt.legend()
    plt.show()

# MAIN
filename = "/content/Iris.csv"  # Replace with your actual file path
dataset = load_dataset(filename)
train_set, test_set = train_test_split(dataset)

w, b = train_svm(train_set)
plot_svm(train_set, w, b)

# Testing
for point in test_set:
    x = point[:2]
    actual = point[2]
    pred = predict(x, w, b)
    print(f"Actual: {actual:<2}  Predicted: {pred}")
```

Output:


Linear SVM Classifier
Decision Boundary
Sepal Width
Sepal Length
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
Actual: -1 Predicted: -1
Actual: -1 Predicted: -1
Actual: 1 Predicted: 1


Predicted Class for [6.7,3,5,1.7]: Iris-setosa
KNN Prediction (k=3)
Predicted: Iris-setosa
Sepal Width (cm)
Sepal Length (cm)
8
7
6
5
4
3
0
20
40
60
80
100
120
140