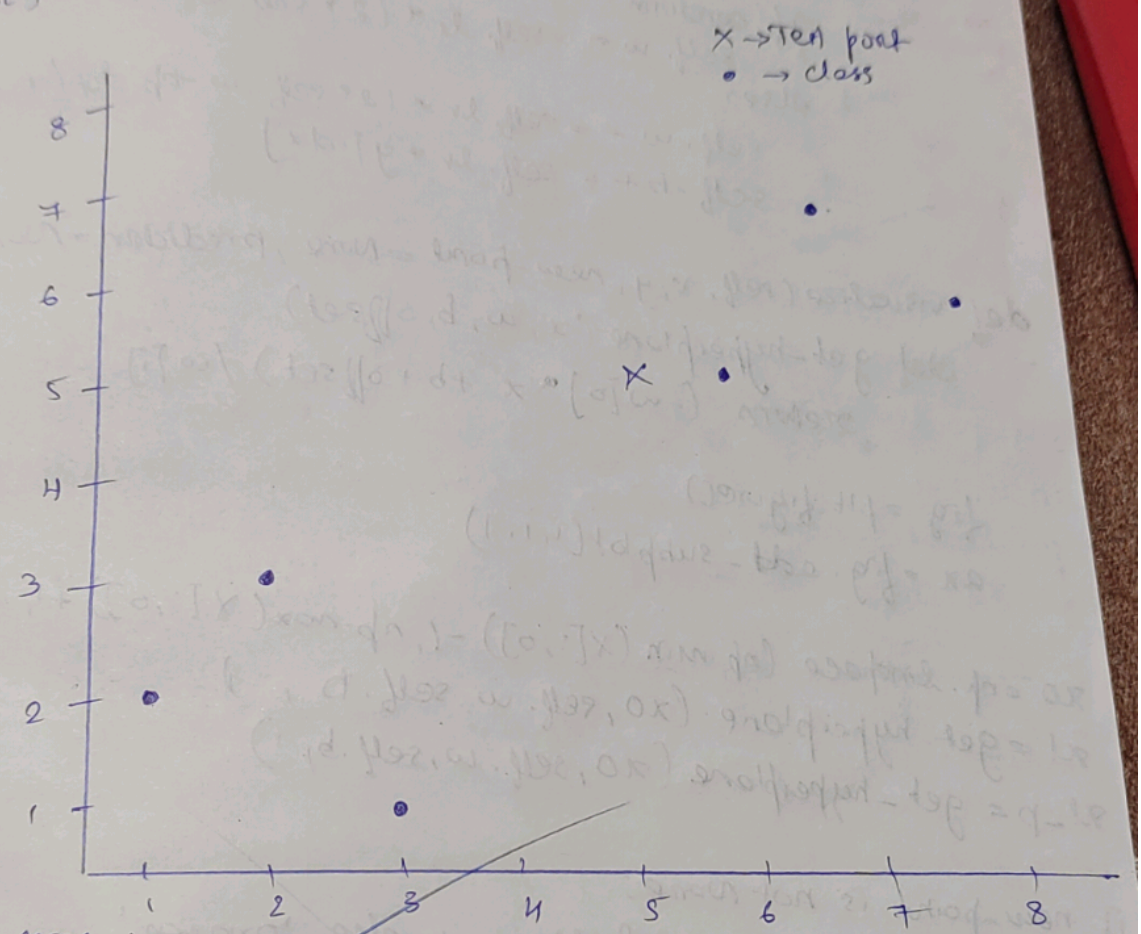
LAB-4

KNN Algorithm:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)

    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train
                     in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)

X_train = np.array([[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 6]])
y_train = np.array([0, 0, 0, 1, 1, 1])
X_test = np.array([[5, 5]])

knn = KNN(k=3)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)

plt.figure(figsize=(8, 6))
```

```
for i in range(len(X_train)):
    plt.scatter(X_train[i][0], X_train[i][1], color='red' if
        y_train[i]==0 else 'blue', label=f"Class
    {y_train[i]}" if f"Class {y_train[i]}" not in
    plt.gca().get_legend_handles_labels()[1] else "")
plt.scatter(X_test[0][0], X_test[0][1], color='green', s=200,
            marker='X', label='Test Point')

plt.legend()
plt.show()
```

Output:



SVM Algorithm

```
import numpy as np
import matplotlib.pyplot as plt

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None
```

```
def fit(self, X, y):
    y = np.where(y <= 0, -1, 1)
    n_samples, n_features = X.shape
    self.w = np.zeros(n_features)
    self.b = 0

    for _ in range(self.n_iters):
        for idx, x_i in enumerate(X):
            if condition:
                self.w -= self.lr * (2 * self.lambda_ * self.w)
            else:
                self.w -= self.lr * (2 * self.lambda_ * self.w - np.dot(x_i, y[idx]))
                self.b += self.lr * y[idx]

def visualize(self, X, y, new_point=None, prediction=None):
    def get_hyperplane(x, w, b, offset):
        return (-w[0] * x + b + offset) / w[1]

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)

x0 = np.linspace(np.min(X[:, 0]) - 1, np.max(X[:, 0]) + 1, 100)
x1 = get_hyperplane(x0, self.w, self.b, 0)
x1_p = get_hyperplane(x0, self.w, self.b, 1)

if new_point is not None:
    color = 'green' if prediction == 1 else 'orange'
    label = f'New Point: Class {"1" if prediction == 1 }'
    plt.scatter(new_point[0], new_point[1], label=label)
    plt.show()

#example
if __name__ == "__main__":
    X = np.array([[1, 7], [2, 8], [3, 8], [8, 1], [9, 2], [
    y = np.array([0, 0, 0, 1, 1, 1])
    new_point = np.array([[5, 5]])
```

```
svm = SVM()
svm.fit(X, y)
prediction = svm.predict(new_point)[0]
svm_visualize(X, y, new_point = new_point[0], prediction = prediction)
print(f"New point {new_point[0]} classified as : {'Class 1' if prediction == 1 else 'Class 0'}")
```

Output:

New point [5 5] classified as : Class 0