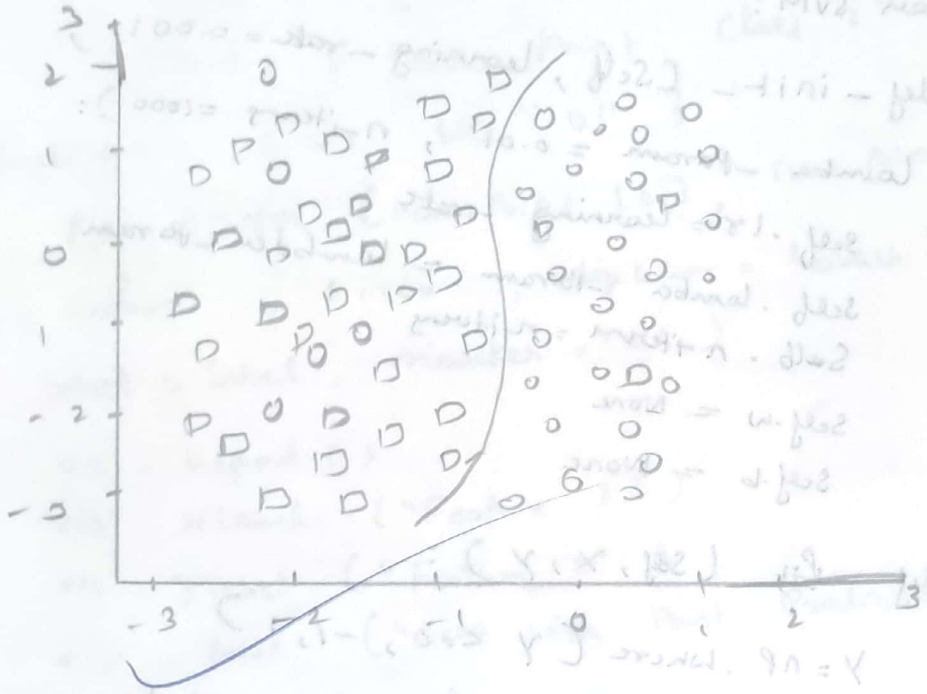
Lab-4

KNN (K Nearest Neighbour algorithm)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X, y = make_classification (n_samples=200, n_features=2,
    n_classes=2, random_state=42,
    n_informative=2, n_redundant=0, n_repeated=0)

X_train, X_test, y_train, y_test = train_test_split
    (X, y, test_size=0.3, random_state=42)

Scaler = StandardScaler()
X_train = Scaler.fit_transform(X_train)
X_test = Scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

h=0.2
x_min, x_max = X_train[:,0].min()-1,
    X_train[:,0].max()+1,
y_min, y_max = X_train[:,1].min()
    -1, X_train[:,1].max()+1

xx, yy = np.meshgrid(np.arange(x_min, x_max,
    h), np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(),
    yy.ravel()])
```

```
plt.figure(figsize=(10,6))
plt.xlabel("Linear Decision boundary")
plt.ylabel("feature 1")
plt.legend()
plt.show()
```



SVM

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

class SVM:
    def __init__(self, learning_rate=0.001,
    lamba_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lamba_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):
                condition = y[idx] * (np.dot(x_i, self.w)
                + self.w) + self.b) >= 1

                if condition:
                    self.w -= self.lr * (2 * self.lambda_param
                    self.w)

                else:
                    self.w -= self.lr * (2 * self.lambda_
                    param * self.w - np.dot(x_i, y[idx]))
```

```
def Predict (self, x):
    approx = np.dot (x, self.w) + self.b
    return np.sign (approx)

if new_point is not None:
    color = 'green' if Prediction == 1 else 'orange'
    label = f'New point: Class {"1" if
Prediction == 1 else "0"}'
    plt.scatter [new_point [0], new_point [1],
c = color, s = 100, edgecolor = 'black',
label = label, marker = '*')

ax.legend ()
plt.xlabel ("Feature 1")
plt.ylabel ("Feature 2")
plt.title ("SVM with Point Prediction")
plt.grid (True)
plt.show ()
```