SVM (Support vector machines)

It is a superised machine learning problem where we try to find a hyperplane that best separates the two classes.

Soft margin: Can misclassify few data points but corecrly classifies majority points comsider a student (not well) = 35 marks
Hard margin: None of the observations are misclassified. (boundary is not smooth)

Support vectors: these are the points that are closest to the hyperplane.
(planes stop when they find the 1st point)

A separating line will be defined with the help of these data points.

Margin: It is the distance between the hyperplane and the observations closest to the hyporplane.(support vector)
In svm larg margin is a good margin.

Find the equation of hyper plane:margin max
How data points are classified, after getting decision boundary

#######

Steps :

find the vector w perpendicular to all hyper planes.(we know only derection)

consider a unknown point(vector u)(have durection)

we need to find the class for vector u

project your vector u on vector w (w*u)*(dot product)*

calculate the distance from origin to decision boundary. Beyond (+ve) or negative

distance is unknown

dot product(w*u)>=c(unknown distance)*

w*u-c>=0*
*w*c+b >=0 (decision boundary)

(linear separable tricks (kernal tricks))

In [ ]:

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

In [ ]:

```
class SVM:
    def __init__(self, kernel='linear', learning_rate=0.01, lambda_param=0.01, n_iters
        self.kernel = kernel
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.alpha = None
        self.b = None
        self.gamma = gamma
        self.X = None
        self.y = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.X = X
        self.y = y
        self.alpha = np.zeros(n_samples)
        self.b = 0

        if self.kernel == 'linear':
            K = np.dot(X, X.T)
        elif self.kernel == 'rbf':
            K = np.zeros((n_samples, n_samples))
            for i in range(n_samples):
                for j in range(n_samples):
                    K[i,j] = np.exp(-self.gamma*np.linalg.norm(X[i] - X[j])**2)
        else:
            raise ValueError('Invalid kernel type.')

        # Gradient descent
        for _ in range(self.n_iters):
            for idx in range(n_samples):
                condition = y[idx]*(np.sum(self.alpha*y*K[idx,:])-self.b)>= 1
                if condition:
                    self.alpha[idx] -= self.lr*(2*self.lambda_param*self.alpha[idx])
                else:
                    self.alpha[idx] -= self.lr*(2*self.lambda_param*self.alpha[idx]-y[
                    self.b -= self.lr*y[idx]

    def predict(self, X):
        if self.kernel == 'linear':
            linear_output = np.dot(X, self.X.T)
        elif self.kernel == 'rbf':
            n_samples = X.shape[0]
            linear_output = np.zeros(n_samples)
            for i in range(n_samples):
                kernel = np.exp(-self.gamma*np.linalg.norm(self.X-X[i], axis=1)**2)
                linear_output[i] = np.sum(self.alpha*self.y*kernel)-self.b
        else:
            raise ValueError('Invalid kernel type.')

        return np.sign(linear_output)
```

In [ ]:

```
```

In [ ]:

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

In [ ]:

```
class SVM:
    def __init__(self, kernel="linear", learning_rate=0.01, lambda_param=0.01, n_iters
        self.kernel = kernel
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.gamma = gamma
        self.degree = degree
        self.w = None
        self.b = None

    def linear_kernel(self, x1, x2):
        return np.dot(x1, x2)

    def polynomial_kernel(self, x1, x2):
        return (self.gamma*np.dot(x1, x2) + 1)**self.degree

    def rbf_kernel(self, x1, x2):
        return np.exp(-self.gamma*np.linalg.norm(x1 - x2)**2)

    def kernel_function(self, x1, x2):
        if self.kernel == "linear":
            return self.linear_kernel(x1, x2)
        elif self.kernel == "polynomial":
            return self.polynomial_kernel(x1, x2)
        elif self.kernel == "rbf":
            return self.rbf_kernel(x1, x2)

    def fit(self, X, y):
        n_samples, n_features = X.shape

        y_ = np.where(y <= 0, -1, 1)

        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(self.w, self.kernel_function(x_i, X))-self
                if condition:
                    self.w -= self.lr*(2*self.lambda_param*self.w)
                else:
                    self.w -= self.lr*(2*self.lambda_param*self.w - y_[idx]*self.kerne
                    self.b -= self.lr*y_[idx]

    def predict(self, X):
        approx = np.dot(X, self.w)-self.b
        return np.sign(approx)
```

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

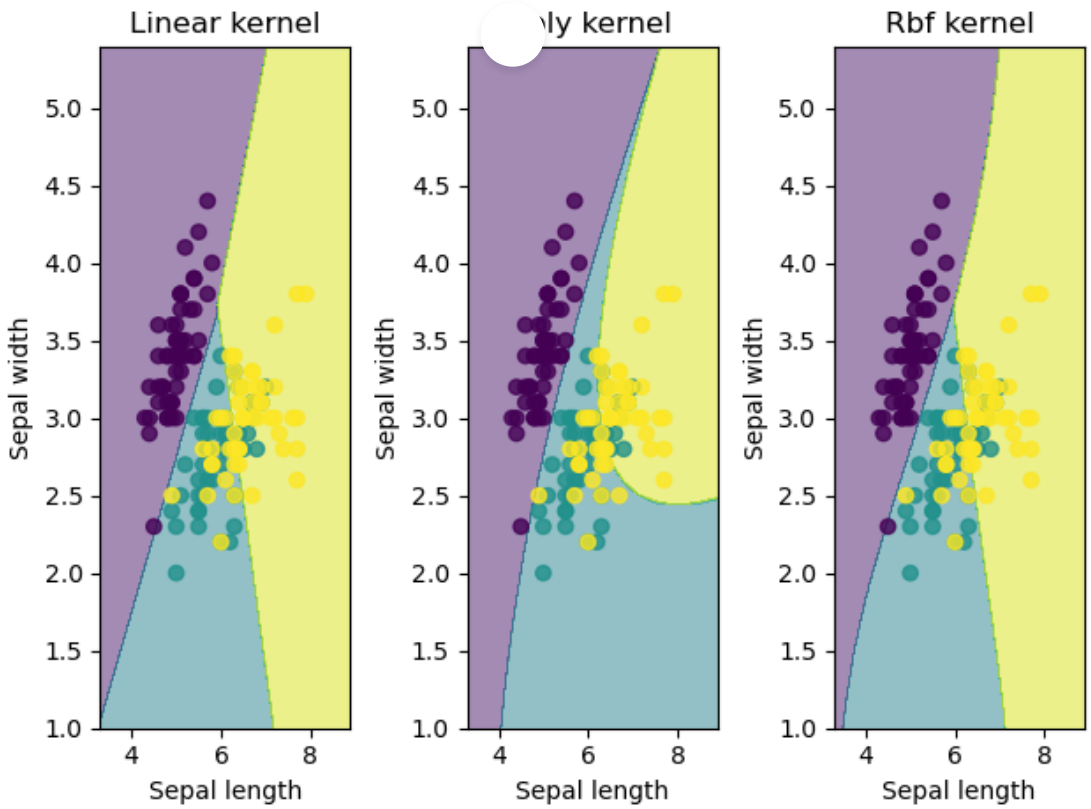
In [ ]:

```

```

In [54]:

```
# Load Iris dataset
iris = load_iris()

# Extract the first two features (sepal length and sepal width)
X = iris.data[:, :2]
y = iris.target

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Train the SVM model with different kernels
kernels = ["linear", "poly", "rbf"]
for i, kernel in enumerate(kernels):
    plt.subplot(1, 3, i + 1)

    # Train the SVM model
    svm = SVC(kernel=kernel, C=1, gamma="auto")
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)

    # Compute the accuracy of the model
    accuracy = np.mean(y_pred == y_test)
    print(f"Accuracy ({kernel} kernel): {accuracy:.2f}")

    # Plot the decision boundary
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x1_min, x1_max, 0.01), np.arange(x2_min, x2_max, 0.
    Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.5)
    plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8)
    plt.xlabel("Sepal length")
    plt.ylabel("Sepal width")
    plt.title(f"{kernel.capitalize()} kernel")

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

```
Accuracy (linear kernel): 0.90
Accuracy (poly kernel): 0.80
Accuracy (rbf kernel): 0.90
```


Linear kernel
ly kernel
Rbf kernel
Sepal width
Sepal length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
6
8

In [ ]: