The last 50 instences in iris data is one type "Virginica" and also The first 50 instences is "setosa". so the rest data is just two classes

- **The head 50 instences from data**

```
iris = sns.load_dataset("iris")
iris = iris.head(50)
np.unique(iris["species"])
```

```
array(['setosa'], dtype=object)
```

- **The tail 50 instences from data**

```
iris = sns.load_dataset("iris")
iris = iris.tail(50)
np.unique(iris["species"])
```

```
array(['virginica'], dtype=object)
```

Impelementation of Soft Margin SVM with Gradient Descent

HyperPlane Equation

$$h(x) = \beta^T X + b$$

```
def HyperPlane( X,beta,b):
    return X.dot(beta) + b
```

Cost Function or Loss Function (We try to Minimize this Function . to increase margin that help us to choose the best hyberPlane)

$$Loss = \frac{\beta^T \beta}{2} + C \sum \epsilon_i$$

- **Slack Variable :** $\epsilon_i = max(0, 1 - y_i(\beta^T X_i + b))$

Depend On

$$y_i(\beta^T X_i + b) >= 1 - \epsilon_i$$

```
def CostFunction( margin,beta,C):
    return (1 / 2) * beta.dot(beta) + C * np.sum(np.maximum(0, 1 - margin))
```

> **HyperParameter C can determine margin if c close to zero so we trying maximize margin and vice verse**

Margin

$$Margin = y_i(\beta^T X_i + b)$$

Where

$y_i(\beta^T X_i + b) \geqslant 1$ in Canonical HyperPlane or in general $y_i(\beta^T X_i + b) \geqslant margin$

```
def Margin(X, y,beta,b):
    return y * HyperPlane( X,beta,b)
```

Gradient Descent

as we learnt in the second lecture, We can update weightes by derivative of cost function w.r.t weights and learning rate or λ

$$Loss = \frac{\beta^T \beta}{2} + C \sum \epsilon_i$$

Derivative w.r.t Beta : $$\frac{\partial Loss}{\partial \beta} = \beta - C \sum y_i x_i$$

Update of beta : $$\beta = \beta - \lambda \frac{\partial Loss}{\partial \beta}$$

Derivative w.r.t b : $$\frac{\partial Loss}{\partial b} = -C \sum y_i$$

Update of b : $$b = b - \lambda \frac{\partial Loss}{\partial b}$$

> For sure, We update weights based on missclassified points

```
def gradientDescent(X,y,margin,beta,b,C,lr):
    misclassified_pts_idx = np.where(margin < 1)[0]
    d_beta = beta - C * y[misclassified_pts_idx].dot(X[misclassified_pts_idx])
    beta = beta - lr * d_beta

    d_b = - C * np.sum(y[misclassified_pts_idx])
    b = b - lr * d_b

    return beta , b
```

Fitting Function

That return Support vectors where the maximum margin with the best hyperPlane is found

```
def fit( X, y,C, lr=1e-3, epochs=1000):
        beta = np.random.randn(X.shape[1])
        b = 0
        loss_array = []
        for i in range(epochs):
            margin = Margin(X, y,beta,b)
            loss = CostFunction(margin,beta,C)
            loss_array.append(loss)
            beta,b=gradientDescent(X,y,margin,beta,b,C,lr)

        Support_vectors = np.where(Margin(X, y,beta,b) <= 1)[0]
        print(Support_vectors)
        return Support_vectors ,beta,b
```

Predict Function

That return only 1 if HyperPlaneequation > 0 or -1 if HyperPlaneequation <0

```
def predict(  X,beta,b):
    return np.sign(HyperPlane( X,beta,b))
```

Score Function

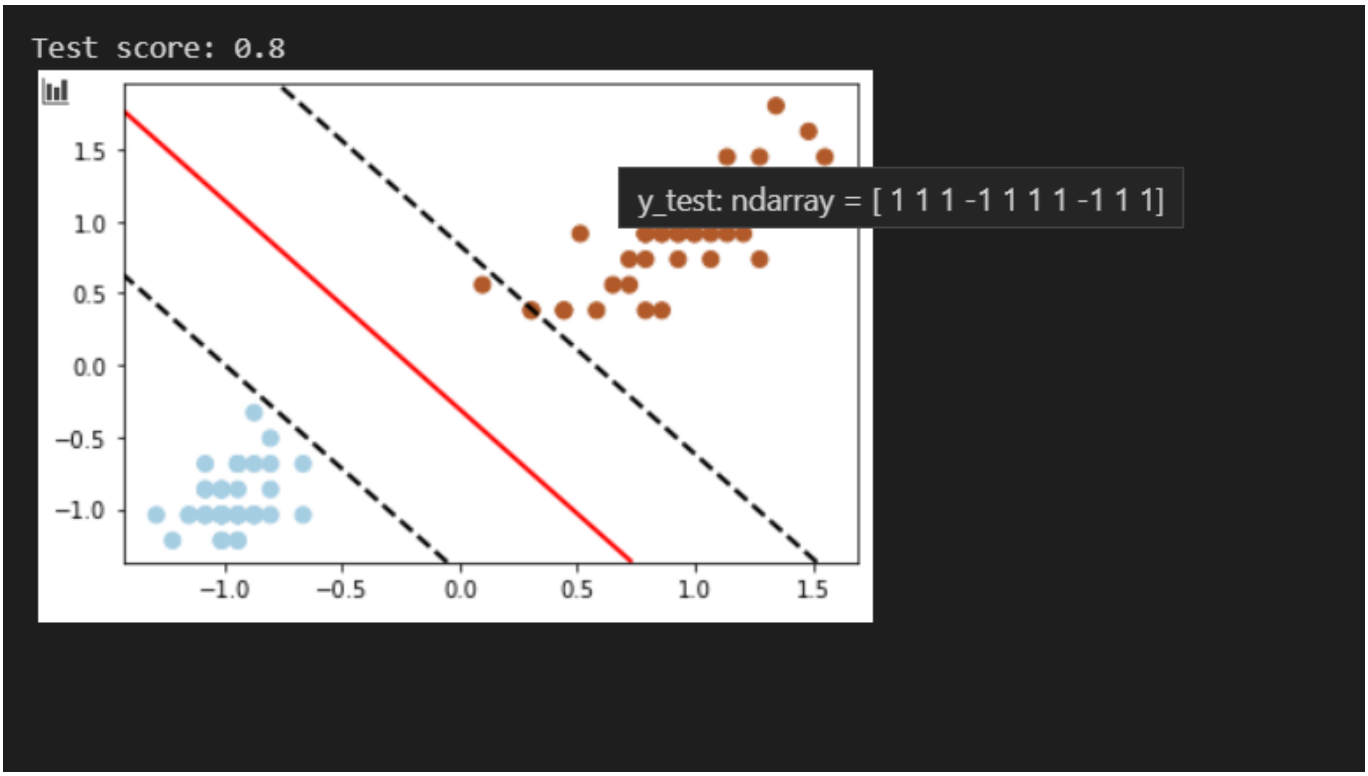
```
def score( y,y_p):
    return np.mean(y == y_p)
```

Load Data Function

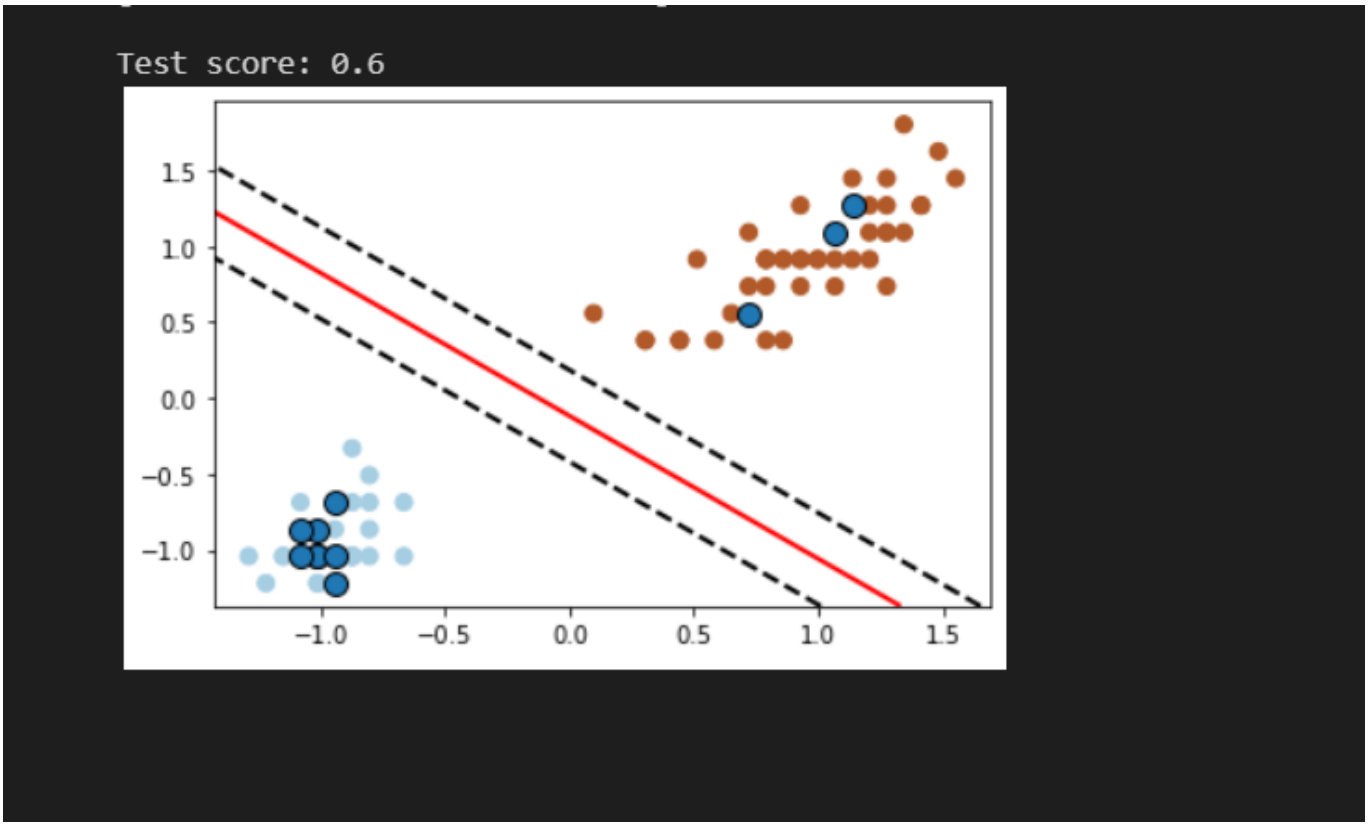
I used the first 100 Instences from iris data Split Data with 10% for testing data and 90% for training data

```
def load_data(cols):
    iris = sns.load_dataset("iris")
    iris = iris.head(100)
    le = preprocessing.LabelEncoder()
    y = le.fit_transform(iris["species"])

    X = iris.drop(["species"], axis=1)

    if len(cols) > 0:
        X = X[cols]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

    return X_train.values, X_test.values, y_train, y_test
```

Result



> When I used the first 100 instences from iris data

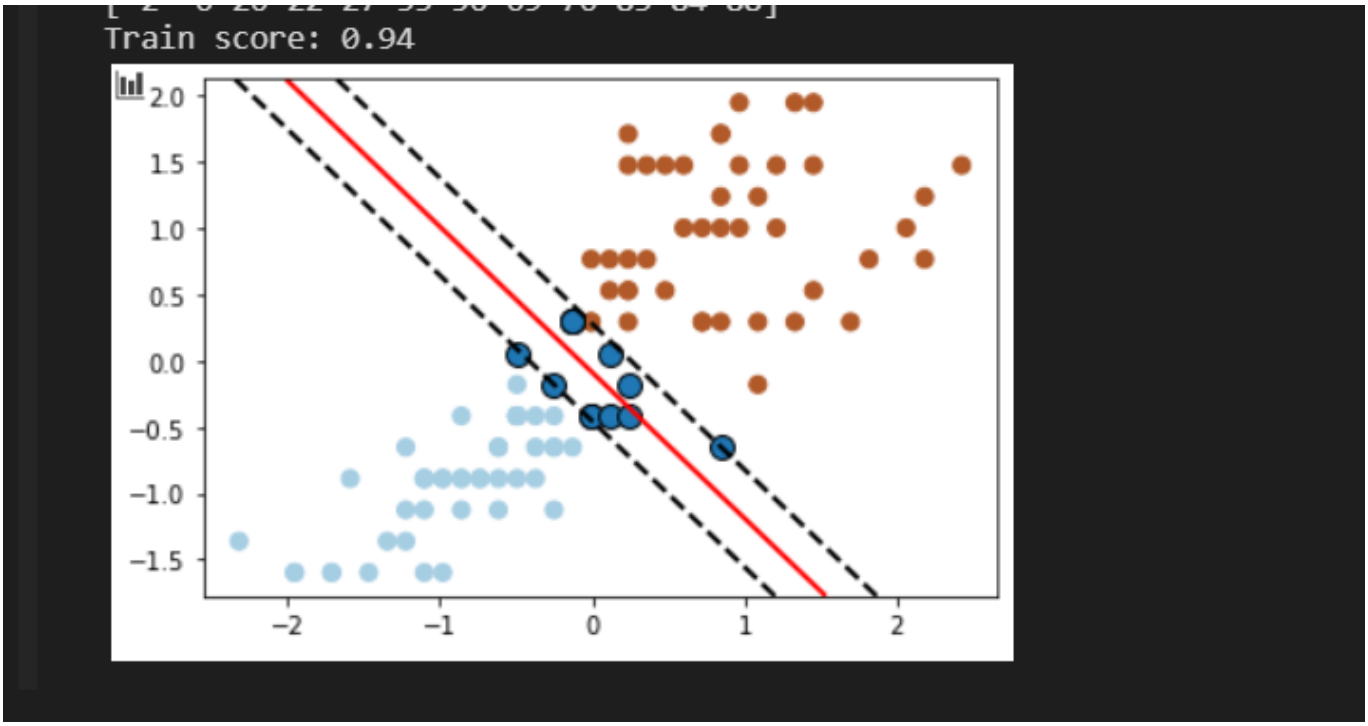


> when i used the last 100 instences from iris data

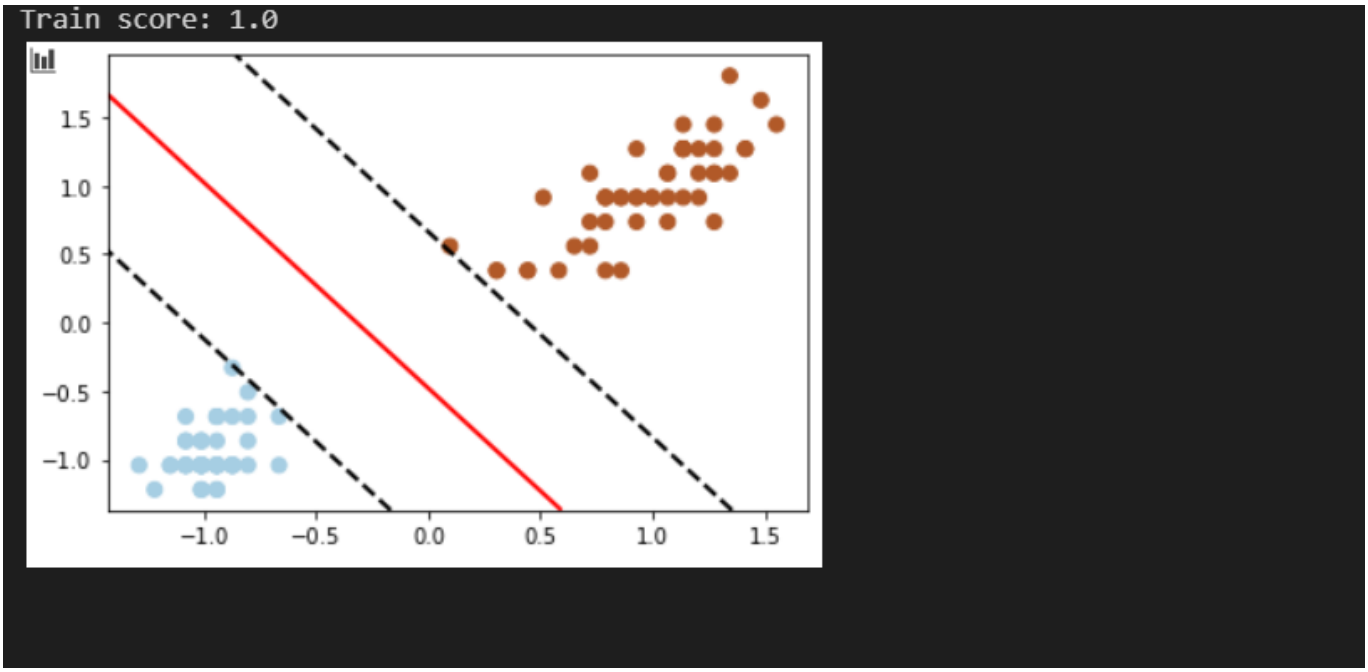
All Data

When I used all data not splitting it to train and test set. because data is very small,just 100 instences and plotting it, it was result

> When i used the last 100 instences



> When i used the first 100 instences



Conclusion

- **AS We saw in result section, Data with the first 100 instences is more better and without error so it svm with hard margin and we dont need port of slack variable becouse data in this case linearly seperable.**
- **and with the the last 100 instences data isn't linearly seperable so I used soft margin svm**
- **I implemented soft margin SVM for this data to be more effective with two part from data (the last and the first 100 instences)**