Support Vector Machine SVM

1

Mathematical formula for a linear SVM :

Given a set of training data with input vectors X1, X2, ..., Xn, and corresponding binary labels y1, y2, ..., yn where yi ∈ {-1, 1}, we want to find a hyperplane that separates the positive and negative classes with the largest margin.

The hyperplane can be defined by the equation:

w^T * x + b = 0

where w is the weight vector perpendicular to the hyperplane, x is an input vector, b is a bias term, and the superscript T denotes the transpose of the vector.

The distance between a point x_i and the hyperplane can be calculated as:

d_i = |w^T * x_i + b| / ||w||

where ||w|| is the Euclidean norm of the weight vector w.

The goal of the SVM is to maximize the margin between the hyperplane and the closest data points from each class. This can be expressed as an optimization problem of finding the optimal values of w and b that satisfy the following constraints:

y_i(w^T * x_i + b) ≥ 1, for all i = 1, 2, ..., n

The objective function to be minimized is:

(1/2) * ||w||^2

where ||w||^2 is the squared Euclidean norm of the weight vector w.

The above optimization problem can be solved using quadratic programming techniques to obtain the optimal values of w and b that define the hyperplane with the maximum margin.

The objective function of a linear SVM

- The objective function of a linear Support Vector Machine (SVM) is to find the hyperplane that separates the positive and negative classes with the largest margin.
- The optimization problem of finding the optimal hyperplane can be expressed as minimizing the following objective function:

(1/2) * ||w||^2

where ||w|| is the Euclidean norm of the weight vector w.

The objective function can be thought of as a regularization term that controls the complexity of the model. The smaller the value of ||w||, the simpler the model, which helps to avoid overfitting.

The objective function is subject to the constraints that the hyperplane correctly separates the positive and negative classes. These constraints can be expressed as:

y_i(w^T * x_i + b) ≥ 1, for all i = 1, 2, ..., n

where y_i is the binary class label (+1 or -1) of the i-th training example, x_i is the input feature vector, and b is the bias term.

#

Kernel trick in SVM

- The kernel trick in Support Vector Machines (SVM) is a technique that allows nonlinear classification by implicitly mapping the input data into a higher-dimensional feature space.

- It is based on the observation that many problems that are not linearly separable in the original input space may become linearly separable in a higher-dimensional feature space.
- The kernel trick avoids the explicit computation of the high-dimensional feature space by defining a kernel function that measures the similarity between two input vectors in the original input space

There are several types of kernel functions that can be used in SVM, including:

- Linear kernel: K(x, y) = x^T y
- Polynomial kernel: K(x, y) = (α x^T y + c)^d, where d is the degree of the - - polynomial and α and c are constants.
- Radial basis function (RBF) kernel: K(x, y) = exp(-γ ||x - y||^2), where γ is a parameter that controls the smoothness of the decision boundary.

4

Role of support vectors in SVM.

Support vectors play a crucial role in Support Vector Machines (SVM) as they determine the location of the decision boundary, also known as the hyperplane, and the margin of the classifier.

- In SVM, the goal is to find the hyperplane that separates the positive and negative classes with the largest margin. The hyperplane is defined by a set of weights, w, and a bias term, b, such that the decision boundary is given by w^T x + b = 0, where x is an input vector.
- The margin of the classifier is defined as the distance between the decision boundary and the closest data point from either class.

During training, the SVM algorithm identifies the subset of training examples that lie on or close to the margin, known as support vectors. These support vectors are the ones that have the most influence on the location of the decision boundary and the width of the margin. The remaining training examples that are further away from the decision boundary do not have any influence on the location of the boundary.

Hyperplane, Marginal plane, Soft margin and Hard margin in SVM with Example

Hyperplane:

A hyperplane is a decision boundary that separates the input data into different classes. In a two-dimensional space, a hyperplane is a line that separates the data into two classes, and in a three-dimensional space, it is a plane that separates the data into two classes.

- In an n-dimensional space, a hyperplane is an (n-1)-dimensional plane that separates the data into two classes.

Example:

Let's consider a two-dimensional space with two classes of data, labeled as blue and red. The hyperplane is a line that separates the blue and red data points. The equation of the line can be written as y = mx + b, where m is the slope and b is the y-intercept. The hyperplane can be adjusted by changing the slope and the y-intercept.

Marginal plane:

In SVM, the distance between the hyperplane and the closest data points is called the margin. The marginal plane is the boundary between the hyperplane and the closest data points.

- The margin is maximized by finding the hyperplane that is farthest from the closest data points. The points that lie on the marginal plane are called support vectors.

Example:

Let's consider a two-dimensional space with two classes of data, labeled as blue and red. The marginal plane is the boundary between the hyperplane and the closest data points. The margin is maximized by finding the hyperplane that is farthest from the closest data points. The points that lie on the marginal plane are called support vectors.

Soft margin:

In some cases, the data points may not be linearly separable, and it may not be possible to find a hyperplane that separates the data perfectly. In such cases, a soft margin SVM can be used.

- In a soft margin SVM, the margin is allowed to be violated by a certain amount, and a penalty is imposed on the margin violation.

Example:

Let's consider a two-dimensional space with two classes of data, labeled as blue and red. The data points are not linearly separable, and it is not possible to find a hyperplane that separates the data perfectly. In such cases, a soft margin SVM can be used, where the margin is allowed to be violated by a certain amount.

Hard margin:

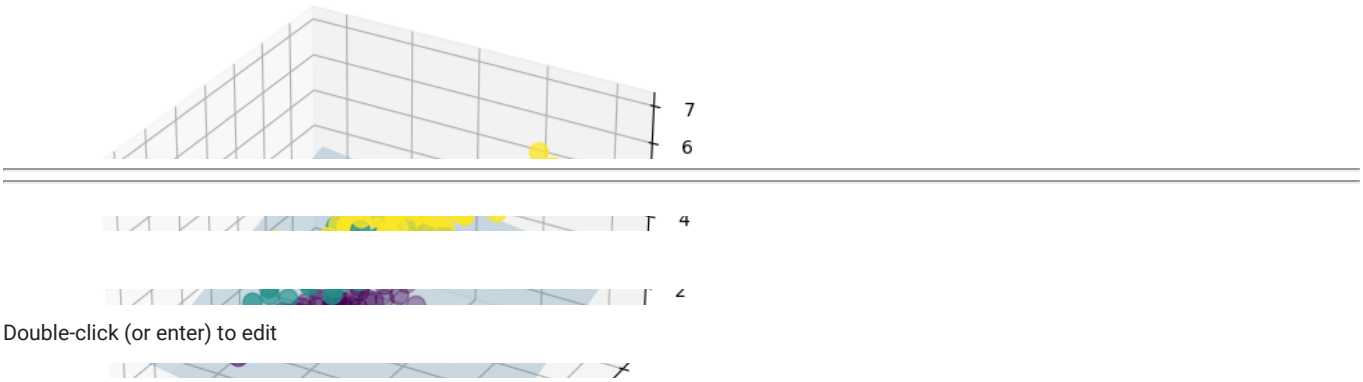
In some cases, the data points may be linearly separable, and it is possible to find a hyperplane that separates the data perfectly. In such cases, a hard margin SVM can be used.

- In a hard margin SVM, the margin is not allowed to be violated, and the hyperplane is chosen to maximize the margin.

Example:

Let's consider a two-dimensional space with two classes of data, labeled as blue and red. The data points are linearly separable, and it is possible to find a hyperplane that separates the data perfectly. In such cases, a hard margin SVM can be used, where the margin is not allowed to be violated.

```
from sklearn import datasets
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

# Load the iris dataset
iris = datasets.load_iris()
X = iris.data[:, :3]
y = iris.target

# Train a linear SVM classifier
clf = SVC(kernel='linear')
clf.fit(X, y)

# Plot the hyperplane and support vectors in 3D
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
z_min, z_max = X[:, 2].min() - 0.5, X[:, 2].max() + 0.5
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.2),
                     np.arange(y_min, y_max, 0.2))
zz = (-clf.intercept_[0]-clf.coef_[0][0]*xx-clf.coef_[0][1]*yy)/clf.coef_[0][2]

ax.plot_surface(xx, yy, zz, alpha=0.2)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap='viridis', s=100)
ax.set_xlabel('Sepal length')
ax.set_ylabel('Sepal width')
ax.set_zlabel('Petal length')
plt.show()
```



Double-click (or enter) to edit

SVM Implementation through Iris dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


import warnings as warns
warns.filterwarnings("ignore")
```

Loading the dataset from scikit-learn library

```
from sklearn import  datasets
from sklearn.model_selection import train_test_split

# load the dataset
iris = datasets.load_iris()

# convert into feature datafreme

df = pd.DataFrame(iris.data , columns = iris.feature_names)
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=4
```

```
X_train.shape ,y_train.shape
```

```
((120, 4), (120,))
```

```
X_test.shape, y_test.shape
```

```
((30, 4), (30,))
```

Train a linear SVM classifier on the training set and predicting the labels for the testing set

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score


svm_clf = LinearSVC()

# train the linear SVM

trained_model = svm_clf.fit(X_train ,y_train)

trained_model
```

```
LinearSVC()
```

```
# predict the labels for the testing data

y_pred = trained_model.predict(X_test)
y_pred
```

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

Accuracy of model on testing set

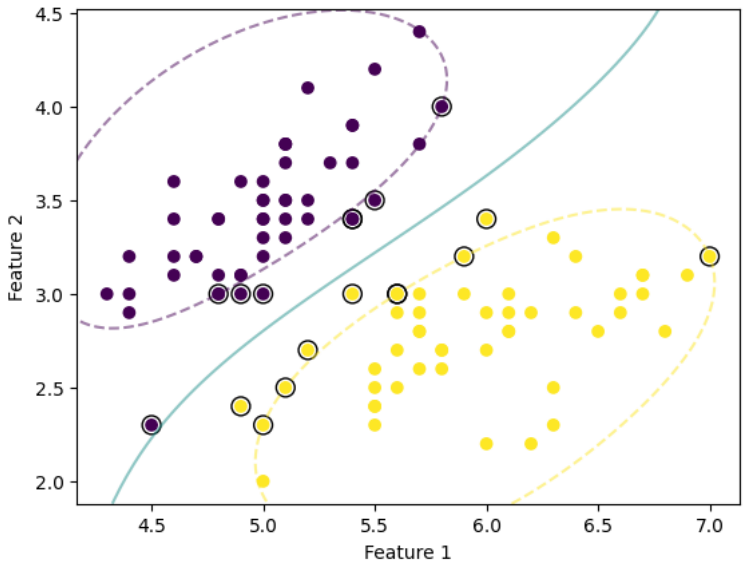
```
accuracy  = accuracy_score(y_test, y_pred)
print("Accuracy: {0:.3f}%".format(accuracy*100))
```

```
Accuracy: 100.000%
```

Plot the decision boundaries of the trained model using two of the featuresl

```
from sklearn.svm import SVC

# Train an SVM model with an RBF kernel
svm = SVC(kernel='rbf', gamma=1.0, C=1.0)
X = X[y != 2]
y = y[y != 2]
svm.fit(X, y)

# Visualize the decision boundary and the support vectors
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')

# Plot the decision boundary
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# Create a grid of points to evaluate the model
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100),
                     np.linspace(ylim[0], ylim[1], 100))
Z = svm.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
# Plot the decision boundary and the margins
ax.contour(xx, yy, Z, levels=[-1, 0, 1], linestyles=['--', '-', '--'], alpha=0.5)
ax.scatter(svm.support_vectors_[:, 0], svm.support_vectors_[:, 1], s=100, linewidth=1, facecolors='none',

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Accuracy at diferenct value of c:

```
# Try different values of C and record the accuracy for each value

C_values = [0.001, 0.01, 0.1, 1, 10, 100]
accuracies = []
for C in C_values:
    svm_clf = LinearSVC(C=C)
    svm_clf.fit(X_train, y_train)
    y_pred = svm_clf.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))
    print("Accuracy with C =", C, ":", accuracies[-1])

# Plot the accuracy vs. C values
plt.plot(C_values, accuracies, '-o')
plt.xscale('log')
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. C values')
plt.show()
```

```
Accuracy with C = 0.001 : 0.7
Accuracy with C = 0.01 : 0.8333333333333334
Accuracy with C = 0.1 : 1.0
Accuracy with C = 1 : 1.0
Accuracy with C = 10 : 1.0
Accuracy with C = 100 : 1.0
```