Support Vector Machine

March 2022

1 Machine Learning Lab#11

1.1 Aim: To implement SVM using scikit-learn library and train it to classify the given dataset.

2 Introduction

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. The main objective of SVM is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

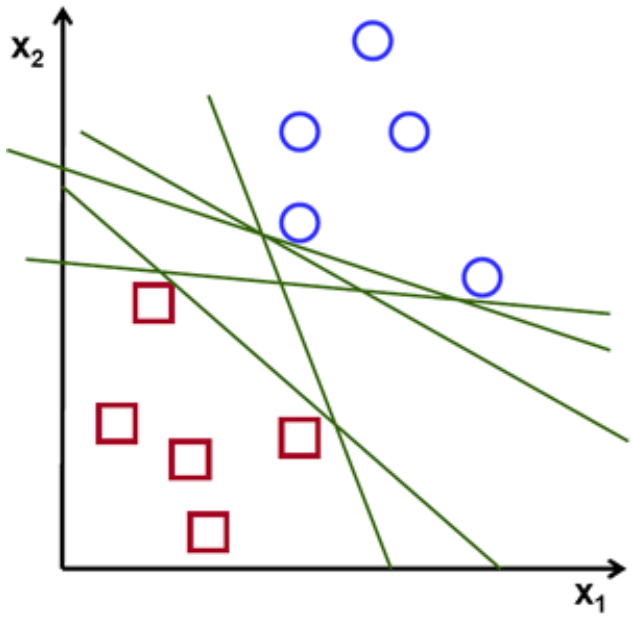
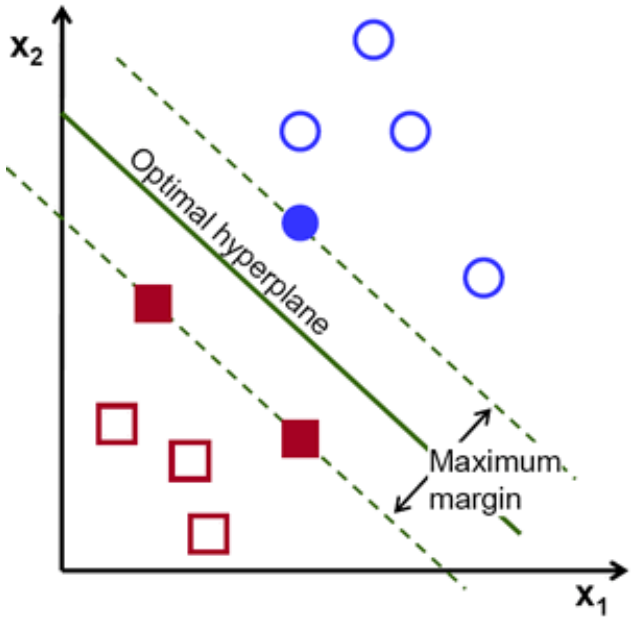


Figure 1 represents all possible hyperplanes for the given dataset. But, using SVM, only that hyperplane will be choosen for which the margin distance is maximum.

Figure 2 shows the choosen hyperplane which maximizes the margin.

SVM can be of two type, Linear SVM where the data is linearly separable and Non-linear SVM where the dataset is not linearly separable using a straight line or a hyperplane.



When the data is linearly separable, and we don't want to have any misclassifications, we use SVM with a hard margin. However, when a linear boundary is not feasible, or we want to allow some misclassifications in the hope of achieving better generality, we can opt for a soft margin for our classifier.

Suppose the data is not linearly separable in original space, then the possible way to solve this is by mapping the data to some higher dimensional space. After mapping the data to higher dimension, the class of data are now linearly separable in higher dimensional feature space. So, now we can fit a decision boundary to separate the features. However, to train our support vector machine and find optimal values of parameters, we would have to perform operations with the higher dimensional vectors in the transformed space. This will lead to extremely high and impractical computational costs.
The kernel trick provides a solution to this problem. The "trick" is that kernel methods represent the data only through a set of pairwise similarity comparisons between the original data observations x (with the original coordinates in the lower dimensional space), instead of explicitly applying the transformations $\Phi(x)$ and representing the data by these transformed coordinates in the higher dimensional feature space. Our kernel function accepts inputs in the original lower dimensional space and returns the dot product of the transformed vectors in the higher dimensional space. This is one of the most useful property of SVM.

In this lab tutorial we will perform implementation of SVM using scikit-learn library and train it to classify the given dataset and also observe the working of kernel trick.

#Aim: To implement SVM using scikit-learn library.

##Key Terms:

Hyperplane: A hyperplane is a decision plane which separates between a set of objects having different class memberships.

Support Vectors : Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins.

Margin : A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

SVM Kernel : The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form.

Linear Kernel : A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

$$K(x, xi) = sum(x * xi)$$

Polynomial Kernel : A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

$$K(x, xi) = 1 + sum(x * xi)^d$$

RBF (Radial Basis Function) Kernel : The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification.RBF can map an input space in infinite dimensional space.

$$K(x, xi) = exp(-gamma * sum((xxi^2))$$

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.

Part A: Basic SVM with Linear Kernel

[ ]:
```
import sys, os
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
import numpy as np
```

[ ]:
```
# importing scikit learn with make_blobs
from sklearn.datasets import make_blobs
# creating datasets X containing n_samples
# Y containing two classes

# plotting scatters

# Split data to train and test on 80-20 ratio
```

[ ]:
```
# Create a linear SVM classifier
clf = svm.SVC(kernel='linear')
```

[ ]:
```
# Train classifier


## Plot decision function on training and test data
#plot_decision_function(X_train, y_train, X_test, y_test, clf)
```

[ ]:
```
# Make predictions on unseen test data
#clf_predictions = ##############################
print("Accuracy: {}%".format(clf.score(X_test, y_test) * 100 ))
```

[ ]:
```
def make_meshgrid(x, y, h=.02):
  x_min, x_max = x.min() - 1, x.max() + 1
  y_min, y_max = y.min() - 1, y.max() + 1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
  return xx, yy
```

```
def plot_contours(ax, clf, xx, yy, **params):
  Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
  Z = Z.reshape(xx.shape)
  out = ax.contourf(xx, yy, Z, **params)
  return out

fig, ax = plt.subplots()
# title for the plots
title = ('Decision surface of linear SVC ')
# Set-up grid for plotting.
X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)
plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
ax.scatter(X0, X1, c=Y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
ax.set_ylabel('y label here')
ax.set_xlabel('x label here')
ax.set_xticks(())
ax.set_yticks(())
ax.set_title(title)
ax.legend()
plt.show()
```

Part B : Breast Cancer Prediction Example

[ ]:
```
#Import scikit-learn dataset library
from sklearn import datasets
#Load dataset
cancer = datasets.load_breast_cancer()
```

[ ]:
```
# print the names of the 13 features

# print the label type of cancer('malignant' 'benign')
```

[ ]:
```
# print data(feature)shape
```

[ ]:
```
# print the cancer labels (0:malignant, 1:benign)
```

[ ]:
```
# plotting scatters
```

[ ]:
```
#Import svm model
from sklearn import svm
#Create a svm Classifier
clf = ######### # Linear Kernel
#Train the model using the training sets
####################
#Predict the response for test dataset
y_pred = ###################
```

[ ]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

[ ]:
```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))
# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

Assignment:

Try SVM classifier on MNIST dataset, compare the performance of linear, polynomial and RBF kernels.