University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 8: Support Vector Machines, September 25th

Lecturer: Alexander Schwing Scribe: SHIVANK MISHRA

1 History of Support Vector Machines

Support Vector Machine (SVM) is a supervised machine learning algorithm in order to perform binary or multi-class classification and regression. The original SVM algorithm was invented by Vladimir Vapnik and Alexey Chervonenkis in 1963. However, *Vapnik et al.*, devised another way to create nonlinear classifiers in 1992. This method required the kernal trick to maximum margin hyperplanes [3]. Then in 1995, *Cortes et al.* introduced the concept of soft margin in SVM [4]. This paper dealt with non-linearly separable training data.

The goal of this scribe is to explain different aspects of SVMs. Subsequently understand the relationship between SVMs and logistic regression. Since SVM was later onturned into a constrained optimization problem, we will also explore the optimization primal and dual for SVM.

Support Vector Machines are named after the datapoints that are called support vectors. They are w* examples that are exactly at a distance $\frac{1}{\|w\|^2}$ from a separating hyperplane. In its core, it is a constrained optimization problem.

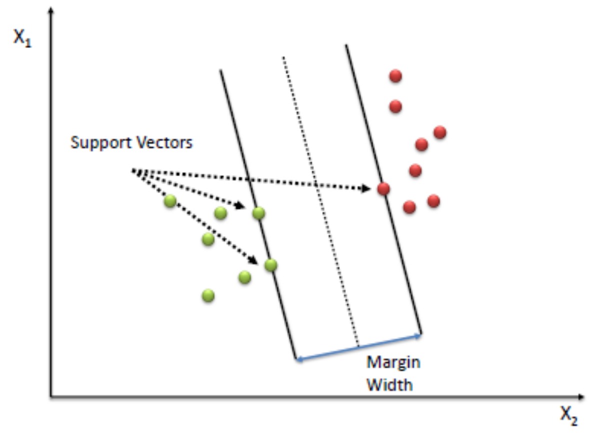


Figure 1: SVM with support vector points [1]

2 Introduction

The elementary idea of SVM is very similar to neural networks, in terms of finding the hyperplane that is most optimum for the linearly separable patterns.

2.1 High-level Understanding of SVMs

SVM discussed in paper is for classification purposes, that draws a single decision boundary to maximize the margin between two set ot more set of points.

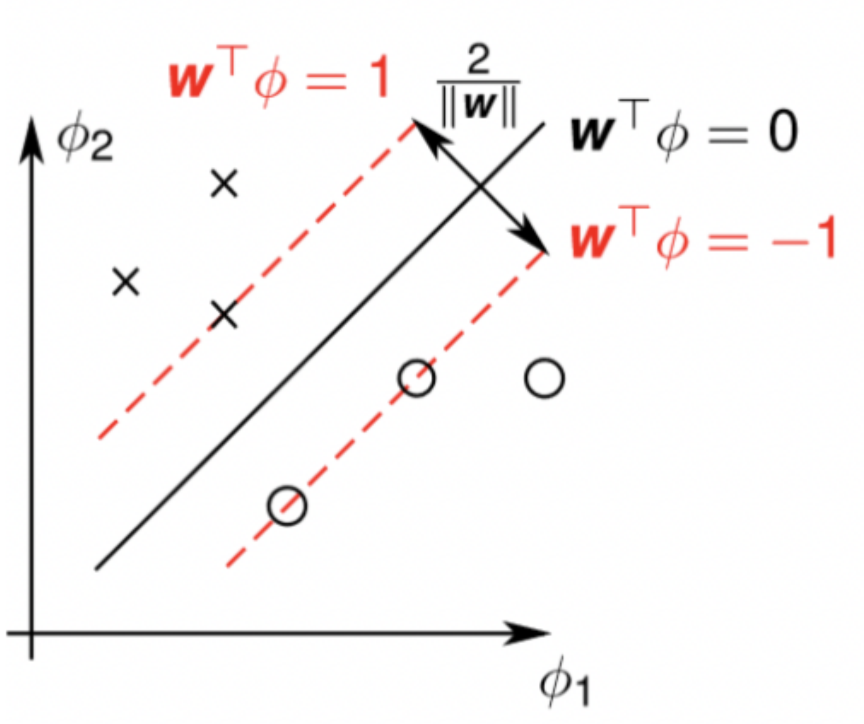

$\mathbf{w}^\top\phi = 1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{w}^\top\phi = 0$
$\mathbf{w}^\top\phi = -1$
ϕ_2
ϕ_1

Figure 2: SVM on the gutter lines/ margin that are at a distance +1 and -1

Consider m labeled points $(x_1, y_1), ..., (x_n, y_n)$ points in the plane, with $x_i \in \mathbb{R}^n$ and y_i {-1,+1}.

In order to classify the points correctly, let us draw a straight line between the points in an attempt to correctly classify the points. However, draw two lines above and below the classifier line respectively.

Let the distance between the two lines and the original classifier line be δ. The aim is to maximize margin δ, in between the classifier line.

3 Basics

3.1 Convex Optimization

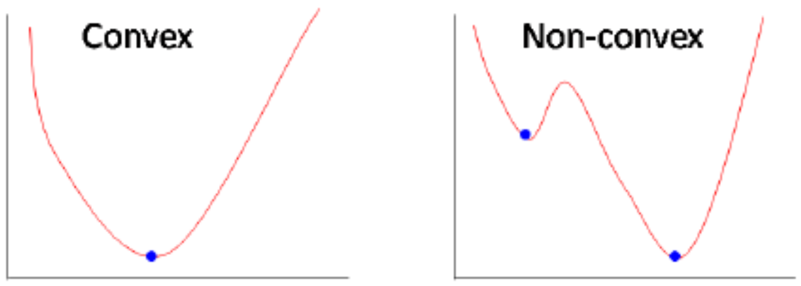


Figure 3: Examples of convex and non-convex functions

The idea of optimization problem is as follows:

$$\begin{aligned} \min_x \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \quad \text{for } i \in 1, 2, \ldots, K, \end{aligned}$$

where f_0 and f_i $(i \in \{1, 2, \ldots, K\})$ are convex functions. This is called the Optimization 'primal' problem.

The convex optimization problems also have following three requirements:

- The objective function must be convex.
- The inequality constraint functions must be convex.
- The equality constraint functions $h_i(x) = a_i^T x b_i$ must be affine (affine transformation is a function between affine spaces that preserves points, straight lineas and planes e.g. a set of parallel lines remain parallel after affine transformation).

The Lagrangian is used to compute optima analytically. It utilizes the KKT condition of optimiality. The Lagrange function follows the KKT conditions of optimiality and implies a dual problem. :

$$L(x, \lambda) = f_0(x) + \sum_{i=1}^{K} \lambda_i f_i(x)$$

The Lagrange dual function, it is derived from using nonnegative Lagrange multipliers to add constraints to the objective function. The solution give primal form :

$$g(\lambda) = \min_x L(x, \lambda)$$

The dual function $g(\lambda)$ is concave. We can obtain the minima of a convex primal optimization problem by maximizing the dual function $g(\lambda)$. Thus the dual optimization problem is as follows:

$$\begin{aligned} \max_\lambda \quad & g(\lambda) \\ \text{s.t.} \quad & \lambda_i \geq 0, \quad \text{for } i \in 1, 2, \ldots, K. \end{aligned}$$

3.2 Karush-Kuhn-Tucker Conditions

In order to understand the Optimization for SVM. It will also help to survey the Karush-Kuhn-Tucker (KKT) condition.

These are the first order necessary conditions for for a solution in nonlinear programming to be optimal in primal and dual functions.

If f_0 and f_i's are convex functions that are differentiable, and the feasible set has some interior points that satify the Slater's condition (in order to reduce them ro make Langrange multiplier rule work)

the x^* and λ_i^*'s are the optimal solutions of the primal and dual problems if and only if they satisfy the following conditions:

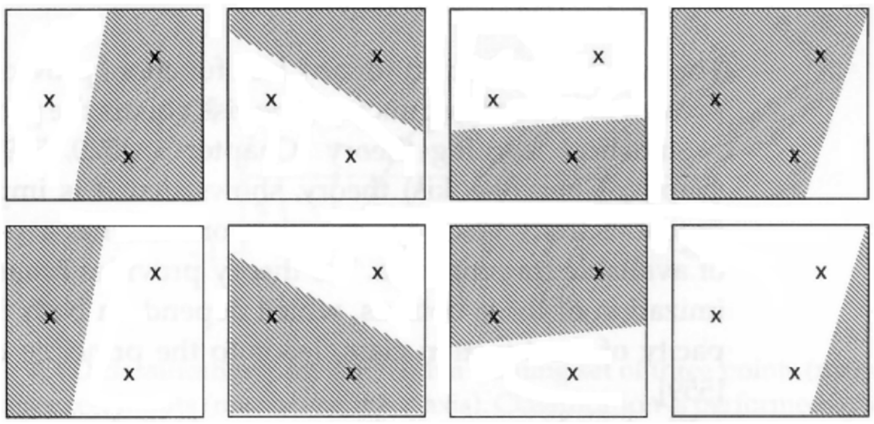


Figure 4: There are 8 ways of assigning 3 points to the two classes. Since the VC dimension can shatter 3 points, so the VC dimension is 3 [5]

$$\begin{aligned} f_i(x^*) &\leq 0 \\ \lambda_i^* &\geq 0,\ \forall i \in 1, \dots, K \\ xL(x^*, \lambda_1^*, \dots, \lambda_K^*) &= 0 \\ \lambda_i^* f_i(x^*) &= 0 \end{aligned}$$

3.3 VC Dimension

The Vapnik-Chernybov Dimension is described as measure of capacity of space that is learned by the classification algorithm. Alternatively, it can be explained as a measure of model's complexity.(from http://www.svms.org/vc-dimension/)

If we have N points in our dataset, that can be labelled in 2_N ways in binary fashion. let us say '+' or '-'. We would like to find hypothesis that cab separate the two classes, then we can say that H shatters/fragments N points. Ideally, the hypothesis should be able to shatter between classes cleanly. However, that is not always the case

High level view of VC dimension is as follows:

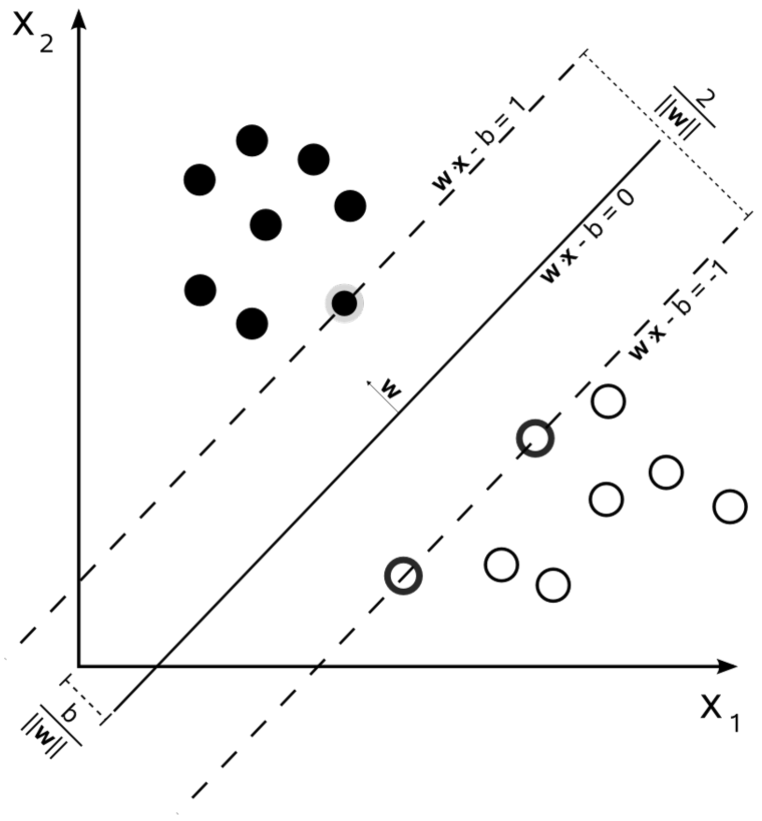
- Higher VC dimension corresponds to higher Confidence Interval
- Lower VC dimension corresponds to High bias.

4 The Problem

4.1 Training Linear SVM

Let there be input of set N training vectors $\{x_n\}_{n=1}^N$ with their corresponding class labels $\{y_n\}_{n=1}^N$, where $x_n \in \mathbb{R}^D$ (Real numbers matrix of length D)and $y_n \in \{-1, 1\}$. Initially we assume that the two classes are linearly separable. The hyperplane separating the two classes can be represented as:

$$w^T x_n + b = 0, where w_0 could be interpreted as b(bias)$$



X_2
X_1
$\mathbf{w}\cdot\mathbf{x} - b = 1$
$\mathbf{w}\cdot\mathbf{x} - b = 0$
$\mathbf{w}\cdot\mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Figure 5: SVM with 2 hyperplanes

such that the equations of two hyper planes are:

$$\begin{aligned} w^T x_n + b &\geq 1 \quad \text{for} \quad y_n = +1, \\ w^T x_n + b &\leq -1 \quad \text{for} \quad y_n = -1. \end{aligned}$$

Let H_1 and H_2 be the two hyperplanes (Figure 6) that separates the binary classes in a way that there is no other data point between them. Our goal is to maximize the margin M between the two classes. The objective function:

$$\begin{aligned} \max_{w} \quad & M \\ \text{s.t.} \quad & y^n (w^T x_n + b) \geq M, \\ & w^T w = 1. \end{aligned}$$

The margin M is $\frac{2}{\|w\|}$. As shown in the figure it spans between the two hyperplanes H_1 and H_2.

The objective function can modified from above to:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2} w^T w \\ \text{s.t.} \quad & y^n (w^T x_n + b) \geq 1 \end{aligned}$$

Above loss functions and conditions work well until the data is not linearly separable. Let us now include a variable called slack $\{\xi_n\}_{n=1}^{N}$ and spread few points that now in the different side of the hyperplane

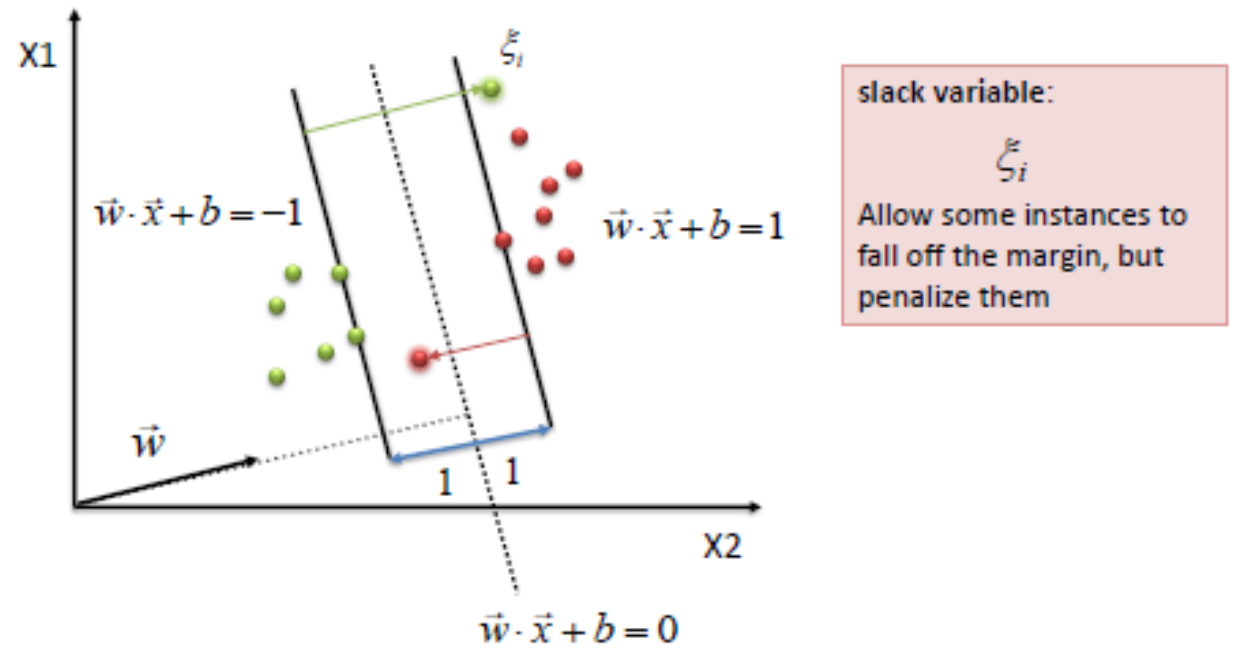

X1
ξ_i
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}$
1
1
X2
$\vec{w}\cdot\vec{x}+b=0$
slack variable:
ξ_i
Allow some instances to fall off the margin, but penalize them

Figure 6: The figure shows a linear SVM classifier for two linearly separable classes. The hyperplane $w^T x + b$ is the solid line between H_1 and H_2, and the the margin is M. [1]

The new objective function now includes sum of slack variables. Whereas C is an hyperparameter that can be tuned using training-validation set :

$$\begin{aligned} \min_{w} \quad & \frac{1}{2} w^T w + C \sum_{n=1}^{N} \xi_n \\ \text{such that} \quad & y^n (w^T x_n + b) + \xi_n \geq 1, \\ & \xi_n \geq 0, \quad \forall n. \end{aligned}$$

The above equations are the optimization primal for the Support Vector Machines.

Consequently, we can write the Lagrangian for the primal problem as follows:

$$L\left(w, b, \xi, \alpha, \mu\right) = \frac{1}{2} w^T w + C \sum_{n=1}^{N} \xi_n - \sum_{n} \alpha_n \left[y_n (w^T x_n + b) \right] - \sum_{n} \alpha_n \xi_n + \sum_{n} \alpha_n - \sum_{n} \mu_n \xi_n,$$

where α_n and μ_n, $1 \leq n \leq N$ are Lagrange multipliers.

Differentiating the Lagrangian with respect to the different variables are as follows:

$$wL\left(w, b, \xi, \alpha, \mu\right) = w - \sum_{n} \alpha_n y_n x_n = 0$$

$$bL\left(w, b, \xi, \alpha, \mu\right) = - \sum_{n} \alpha_n y_n = 0$$

$$\xi_n L(w,b,\xi,\alpha,\mu) = C - \alpha_n - \mu_n = 0$$

When we solve the following equations we obtain the following:

$$w = \sum_n \alpha_n y_n x_n \tag{1}$$

$$\sum_n \alpha_n y_n = 0$$

$$\alpha_n = C - \mu_n \tag{2}$$

In this step we plug-in some values to get a dual function:

$$\begin{aligned} g(\alpha,\mu) &= \frac{1}{2}\sum_n\sum_m \alpha_n\alpha_m y_n y_m {x_n}^T x_m - \sum_n\sum_n \alpha_n\alpha_m y_n y_m {x_n}^T x_m + \sum_n \alpha_n \\ &= \sum_n \alpha_n - \frac{1}{2}\sum_n\sum_m \alpha_n\alpha_m y_n y_m {x_n}^T x_m \end{aligned} \tag{3}$$

Lastly, we utilize the KKT conditions and the equations (2) and (3) in order t obtain optimization dual problem for the SVMs

$$\begin{aligned} \max_\alpha \quad & \sum_n \alpha_n - \frac{1}{2}\sum_n\sum_m \alpha_n\alpha_m y_n y_m {x_n}^T x_m \\ \text{s.t.} \quad & 0 \le \alpha_n \le C. \end{aligned}$$

In the end we get concave function that we need to maximize, in order to solve the optimization. The dual variables (α_n) lie within a box with side C.

We also need to manipulate two values α_i and α_j and numerically optimize the dual function.

We must then plug back the values of α_n^* into get the equation (1) where we can now obtain w^* solution in primal form

4.2 Non- Linear Classifier

Linear SVM utlize dot product with 'w' weight vectors in order to classify the datapoints. However, dataset cannot always be linearly separable. In order to handle non-linearity we create a non-linear classifier.

Therefore, the non-linearity can be handled by replacing the dot-product with a kernal function.

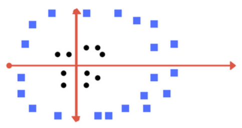


Figure 7: A non-linearly separable dataset

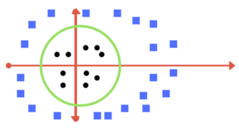


Figure 8: Separated using a kernal function

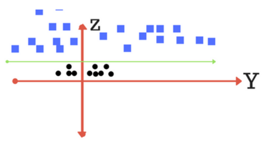


Figure 9: Plot of zy axis

5 Support Vector Machine Implementation on Python

As discussed in the above sections SVMs can used for both classification and regression algorithms. However, let us explore the implementation of classification on linearly separable data with no noise.

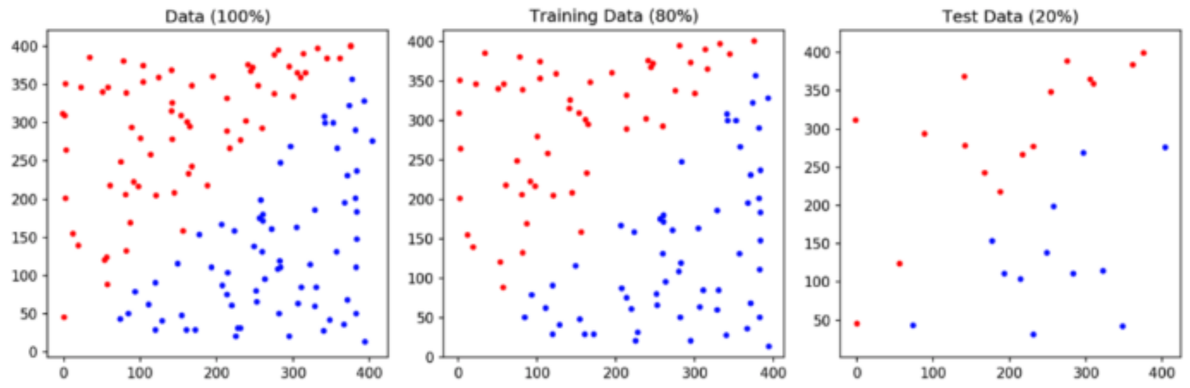
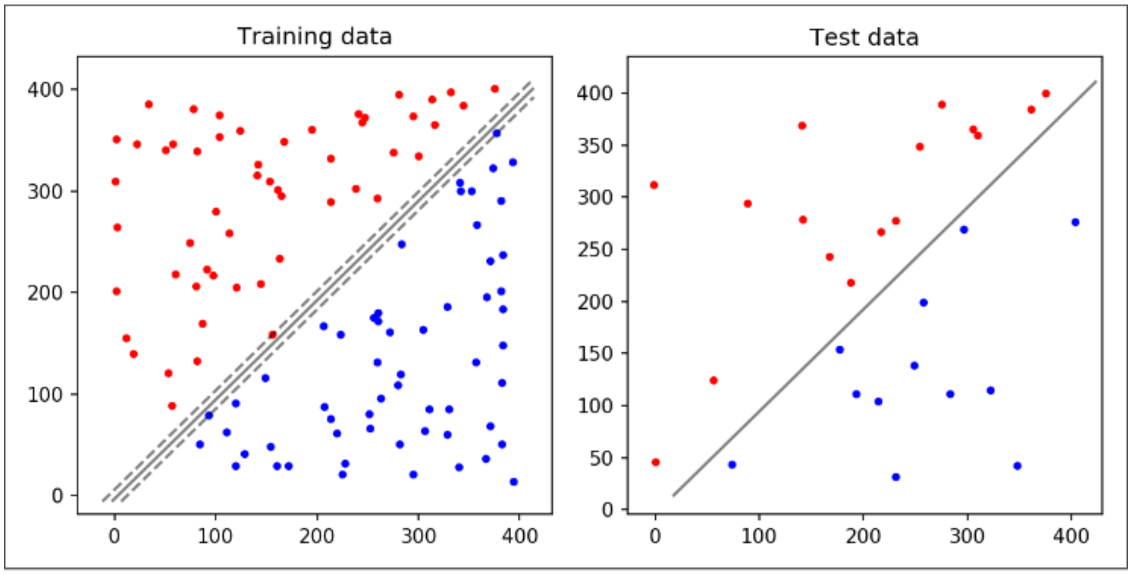


Figure 10: Linearly separable data. The dataset is randomly split into 80% training and 20% test.



6 Conclusion

A Support Vector Machine in the above examples is discussed as discriminative classifier that is in form of a separating hyperplane. We utilize this even on the non-linear regions that use "kernal functions". The algorithm is turned into an optimization problem, where the goal is to output the optimal hyperplane for the given dataset.

Point we should consider is that SVM is what [2] calls it "unnatural" from a probabilistic standpoint. Firstly, they encode the sparsity in the loss function, then Secondly, they encode the kernal trick by using an algorithmic trick. Lastly, SVMs have problems in the multi-class classification setting. Nonetheless, it still remains the most widely used algorithms in Machine Learning

References

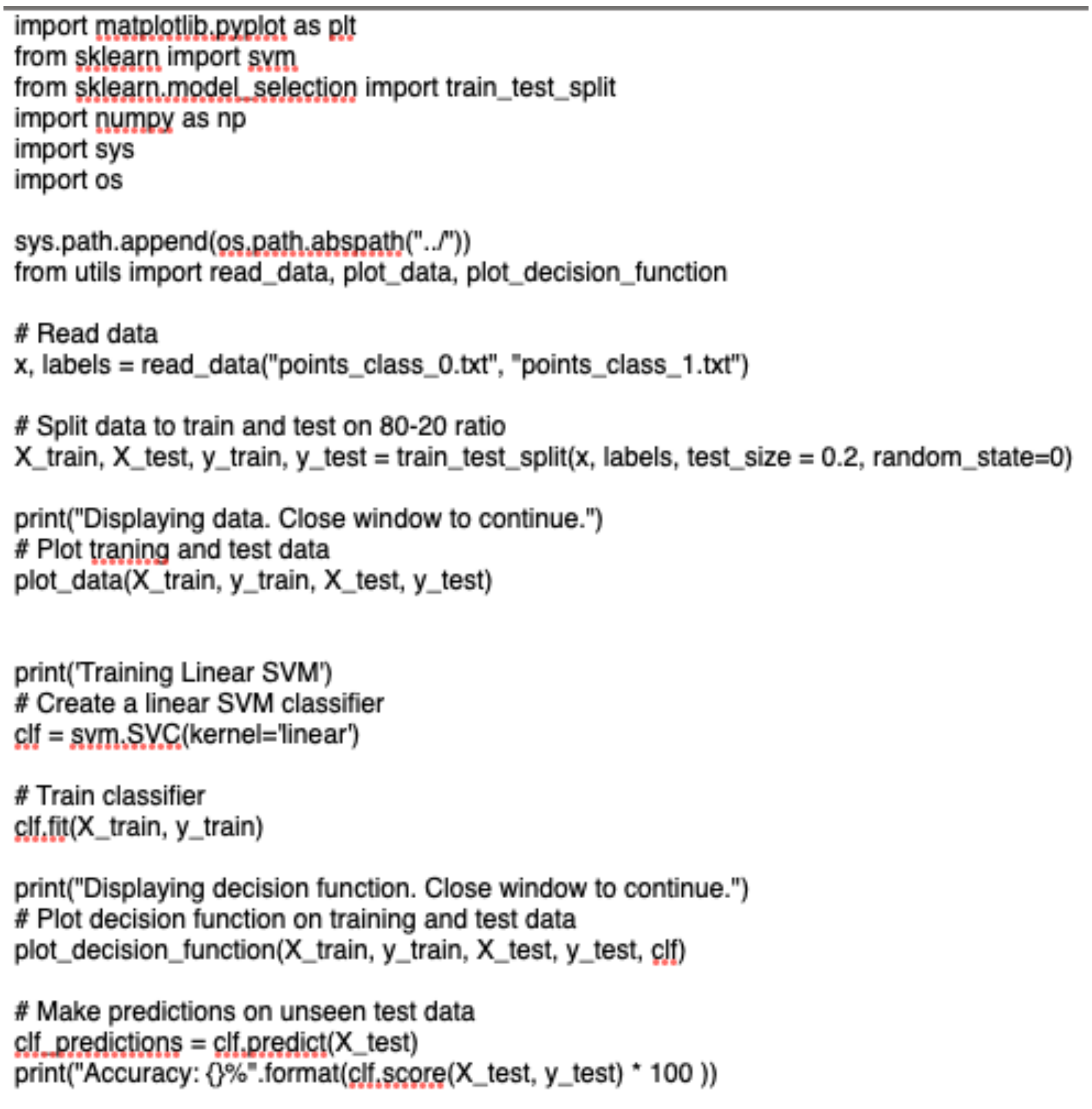
[1] Support vector machine,https://www.saedsayad.com/support_vector_machine.htm.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, Sept. 1995.

[5] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2001.



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

sys.path.append(os.path.abspath("../"))
from utils import read_data, plot_data, plot_decision_function

# Read data
x, labels = read_data("points_class_0.txt", "points_class_1.txt")

# Split data to train and test on 80-20 ratio
X_train, X_test, y_train, y_test = train_test_split(x, labels, test_size = 0.2, random_state=0)

print("Displaying data. Close window to continue.")
# Plot traning and test data
plot_data(X_train, y_train, X_test, y_test)


print('Training Linear SVM')
# Create a linear SVM classifier
clf = svm.SVC(kernel='linear')

# Train classifier
clf.fit(X_train, y_train)

print("Displaying decision function. Close window to continue.")
# 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 = clf.predict(X_test)
print("Accuracy: {}%".format(clf.score(X_test, y_test) * 100 ))
```

Figure 11: https://www.learnopencv.com/svm-using-scikit-learn-in-python/