Lecture 8: Non-Linear Regression and SVM

Gonzalo De La Torre Parra, Ph.D.

Fall 2021

Nonlinear Regression

- Suppose you are given training data $(x_1, y_1), \ldots, (x_n, y_n)$. Assume both variables x and y are real numbers. Then, the linear regression fit between x and y is obtained by solving the following optimization problem:

$$\min_{a,b} \sum_{i=1}^{n} (y_i - ax_i - b)^2.$$

- **Bold assumption or approximation**: By seeking a linear fit between the variables x and y, we are either assuming that the true relationship between x and y is linear. Or we are assuming that the true relationship may be nonlinear, but we are only looking for a linear approximation to that relationship.
- **Elementary nonlinear regression**: If the linear fit leads to terrible predictions, then we can look for possibly nonlinear fit. The framework of linear regression is actually more general than it seems: it can allow us to obtain nonlinear fit as well. For example, the following two problems will give us a nonlinear fit:

$$\min_{a,b} \sum_{i=1}^{n} (y_i - a\frac{1}{x_i} - b)^2$$

$$\min_{a,b,c} \sum_{i=1}^{n} (y_i - ax_i - bx_i^2 - c)^2.$$

- **Nonlinear regression using linear regression**: More generally, we can transform the data using maps $\phi_1, \phi_2, \ldots, \phi_d$, etc and solve

$$\min_{\beta=\beta_1,\ldots,\beta_d} \sum_{i=1}^{n} (y_i - \beta_1\phi_1(x_i) - \beta_2\phi_2(x_i) - \cdots - \beta_d\phi_d(x_i))^2.$$

Here out $\mathbb{X}$ matrix will be

$$\mathbb{X} = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_d(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_d(x_2) \\ \vdots & & & \\ \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_d(x_n) \end{bmatrix}.$$

- **Example 1**: For example, for the problem

$$\min_{a,b} \sum_{i=1}^{n} (y_i - a\frac{1}{x_i} - b)^2,$$

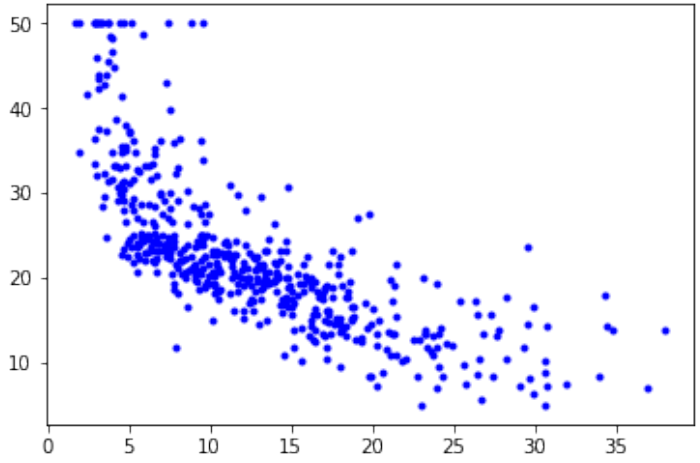
$d = 2$, $\phi_1(x) = \frac{1}{x}$, $\phi_2(x) = 1$.

- **Example 2**: For example, for the problem

$$\min_{a,b,c} \sum_{i=1}^{n} (y_i - ax_i - bx_i^2 - c)^2,$$

$d = 3$, $\phi_1(x) = x$, $\phi_2(x) = x^2$, and $\phi_3(x) = 1$.

- **Boston Housing Data**:



Support Vector Machines

- **Supervised Classification**: Recall the problem of classification in which we are given n pairs of data points:

$$(x_1, y_1), \ldots, (x_n, y_n).$$

We have to learn a relationship between x and y so as to predict y from x. In the problem of classification, y is a **discrete** variable.

- **Example: Iris classification**

6.2.2. Iris plants dataset

Data Set Characteristics:

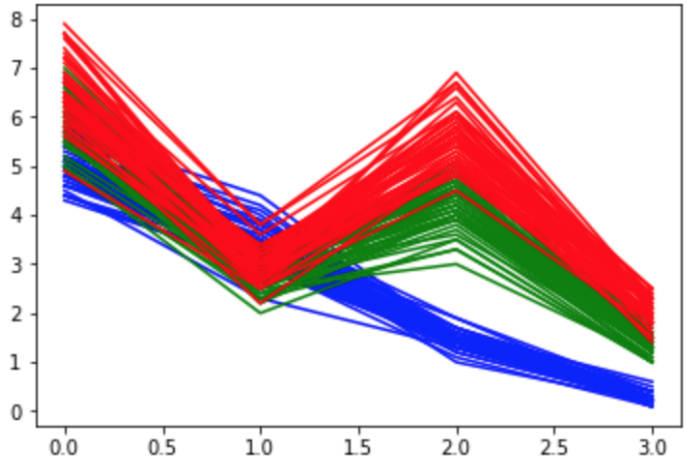
Number of Instances:
150 (50 in each of three classes)
Number of Attributes:
4 numeric, predictive attributes and the class
Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- **class:**
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

IRIS Data from Scikit-learn Package

- **Plots of the 150 features each of length 4:**

```
(150, 4) (150,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

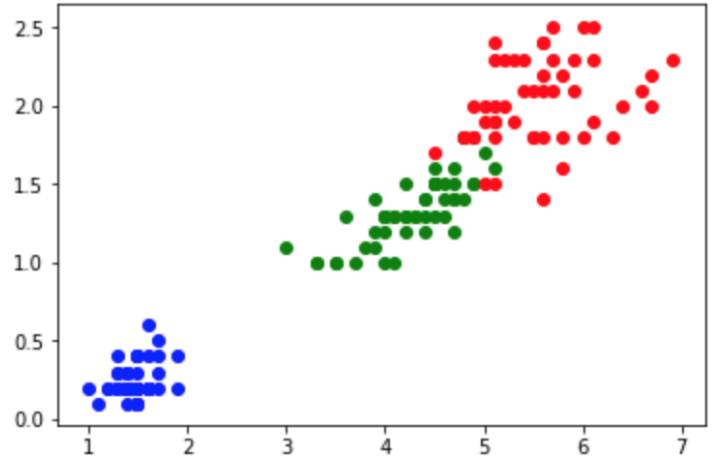


- **2-D Feature extraction**: We want to pick a low-dimensional representation so that we can plot the learned SVM model and appreciate the theory.
- **The third and fourth components have clear red, blue and green non-overlapping bands.**

- **Extracting the third and fourth components for each of the** 150 **training points and creating a scatter plot**:

```
X0 = iris.data[0:50, 2:4]
X1 = iris.data[50:100, 2:4]
X2 = iris.data[100:150, 2:4]
y = iris.target[0:100]
print(X0.shape, X1.shape, X2.shape)
plt.scatter(X0[:,0], X0[:,1], color='b')
plt.scatter(X1[:,0], X1[:,1], color='g')
plt.scatter(X2[:,0], X2[:,1], color='r')
plt.show()
```

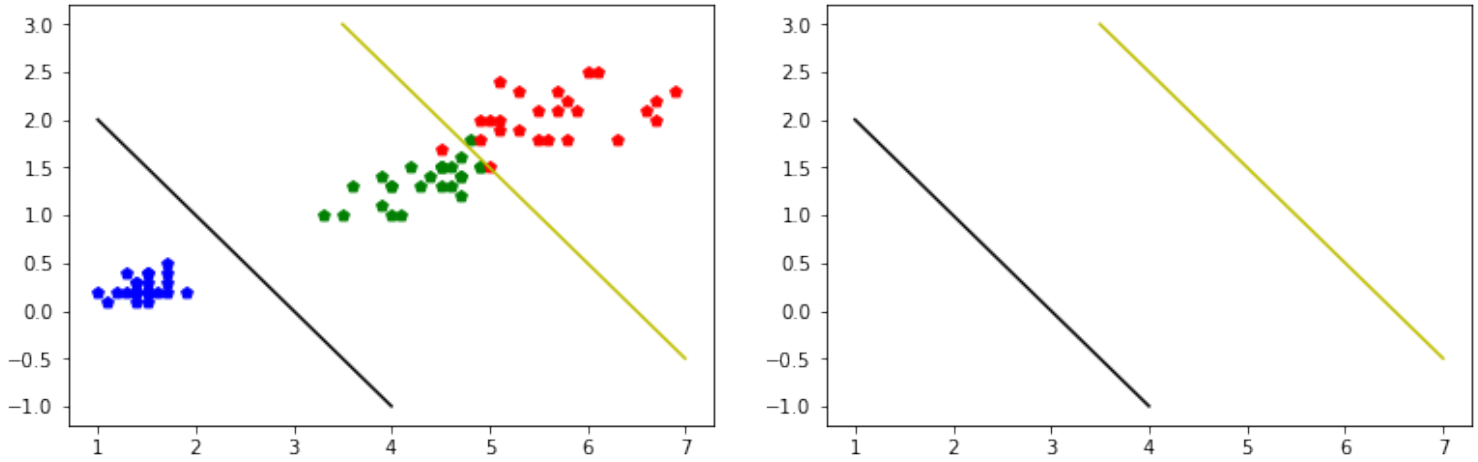
```
(50, 2) (50, 2) (50, 2)
```



- We want to understand how classification, especially SVM, works using 2-D data. Note that the actual classification will be performed using all the four features. The 2-D representation is used only for understanding.
- **Let us imagine we only have a 2D dataset.**

How to Train a Classifier Using 2D Data?

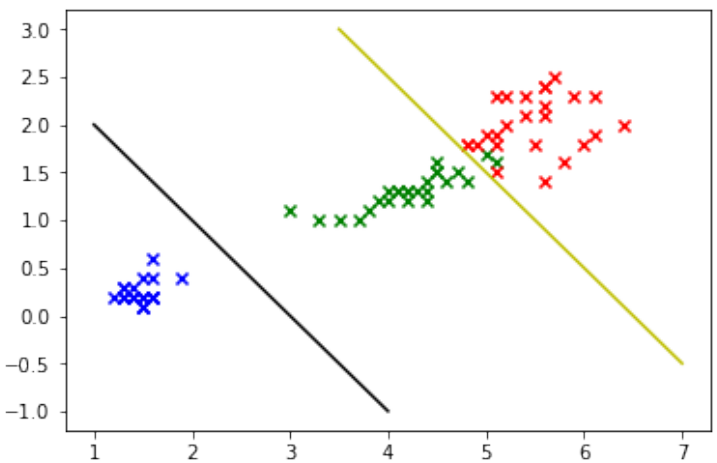
- **Drawing Lines**: One option for training is to 'draw' black and yellow lines between training points of different classes as shown in the figure. **We have used only the first** 25 **points from each class**.



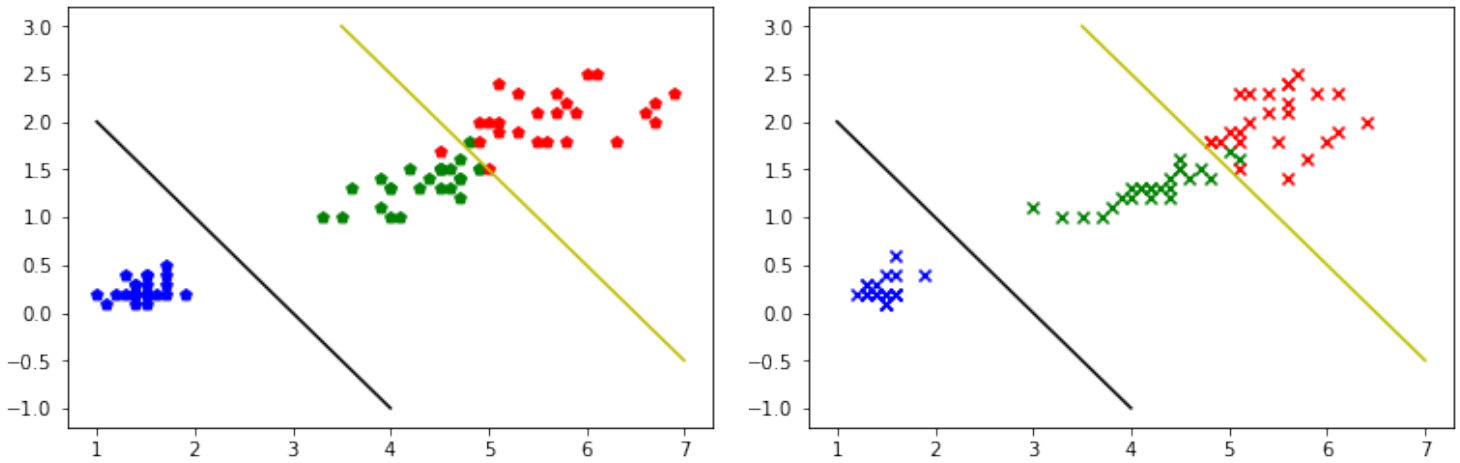
- **Inference or Classification**: Once we have drawn the lines the inference or classification can be done on new data as
  - If point below black line: choose class 0
  - If point above yellow line: choose class 2
  - If point between yellow and black line: choose class 1
- **The central problem in classification is to draw such lines or decision boundaries between data from different classes.**

Why Would We Succeed In Such an Approach?

- **Why do we think drawing such lines between points from different classes is such a great idea or will lead to good prediction?**
- Recall that we drew the lines by looking at the first 25 training points. Now, let us keep the black and yellow lines that we learned and now superimpose the next 25 points from each classes. **Note that the points in the figure are the next** 25 **points from each of the classes**.



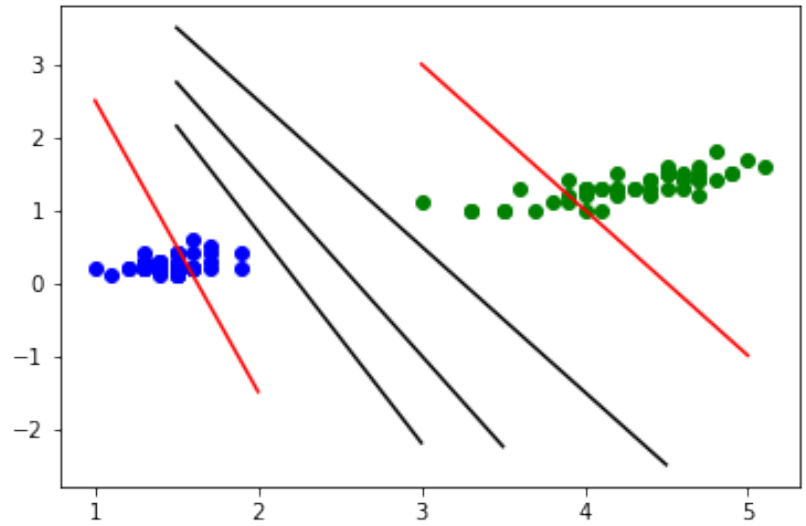
- A comparison reveals that
  - the blue crosses are close to the blue pentagons.
  - the green crosses are close to the green pentagons.
  - the red crosses are close to the red pentagons.



- If test data for a class is close to the old data or training data from the same class, the decision boundaries should successfully separate the points in the test data well.
- **How to draw the 'best' line that will lead to the best prediction?** The Support Vector Machine (SVM) method is one way to draw a good decision boundary.

Two-Class Problem

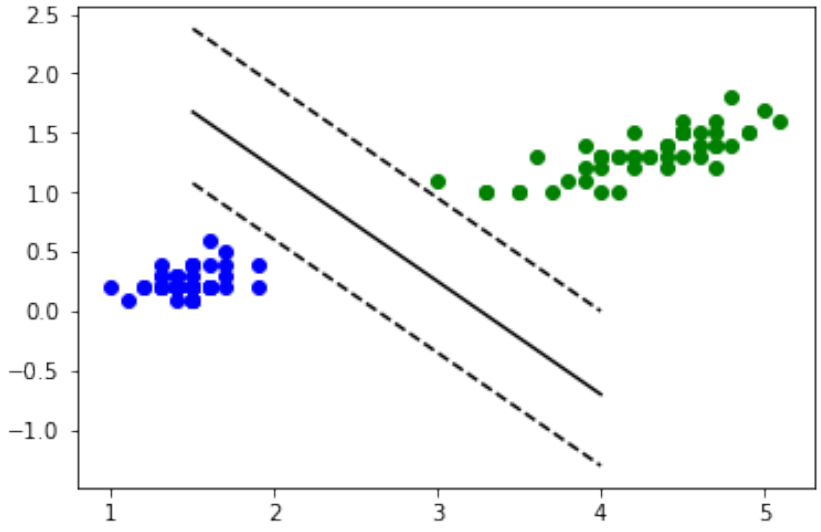
- **Two-Class Problem**: In order to understand the SVM method better, we just consider the first two classes: Class 0 and Class 1.
- **If we wish to draw lines between points from the two classes, which line should we draw? We are now thinking about an automatic ways of doing things.**



- In the figure above, black lines are better than red lines as black lines completely separate the data, but the red lines do not.
- **Conclusion: Black lines are better**.

Good vs Very Good Lines

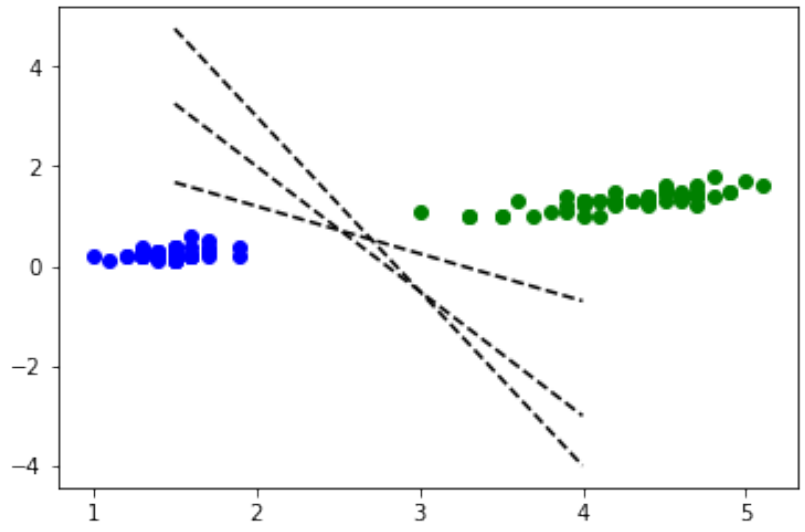
- **Good vs Very Good line**: Now that we understand that we need to draw lines that completely separate data from the two classes, which one should we choose from below?



- Selecting the dashed black lines may not be a good idea because they are quite close to data from one of the classes.
- So, intuitively, we should select the line that is farthest from both the classes.
- **Selecting bold black line is better than selecting either of the dashed black lines.**

The Best Line?

- **Which one is the best?** Now that we know that we should draw a line that is far from all the training data, which one should we draw?



- There are **infinitely many** lines that look like good candidates. We need a mathematically precise way to find the 'best' line. We also need to define what the definition of best line is.
- **In SVM, we define what best is and then find the best line by solving an optimization problem.**

Hyperplanes and Distance to a Hyperplane

- **Defining a line**: Before we discuss the definition of best line, we need to define what a line is in $\mathbb{R}^p$. The p-dimensional version of a line is called a hyperplane.
- **Hyperplane**: For a fixed $\beta \in \mathbb{R}^p$ and $c \in \mathbb{R}$, a hyperplane is a set $V_{\beta,c}$ such that

$$V_{\beta,c} = \left\{ x \in \mathbb{R}^p : \ \beta^T x + c = 0 \right\}.$$

- **SVM and Hyperplane**: In SVM, we wish to find a hyperplane that separates training data from the two classes and is farthest from each training point. To define farthest, we investigate distance of a point from a hyperplane.
- **Distance to a hyperplane**: Let $y \in \mathbb{R}^p$. The distance of y to the hyperplane $V_{\beta,c}$ is defined as

$$d(y, V_{\beta,c}) = \min_{x \in V_{\beta,c}} \|y - x\|,$$

  where $\|y - x\|$ is the Euclidean distance between x and y.
- We have this important lemma (proof will be given later):

  Lemma

  The distance of y from $V_{\beta,c}$ is

$$d(y, V_{\beta,c}) = \frac{|\beta^T y + c|}{\|\beta\|}.$$

Minimum Distance of Training Data

- From the lemma we have that the distance of any point x from the hyperplane $V_{\beta,c}$ is

$$d(x, V_{\beta,c}) = \frac{|\beta^T x + c|}{\|\beta\|}.$$

- **Distance of training points**: Given training points

$$(x_1, y_1), \ldots, (x_n, y_n),$$

the distance of x_i from $V_{\beta,c}$ is

$$d(x_i, V_{\beta,c}) = \frac{|\beta^T x_i + c|}{\|\beta\|}.$$

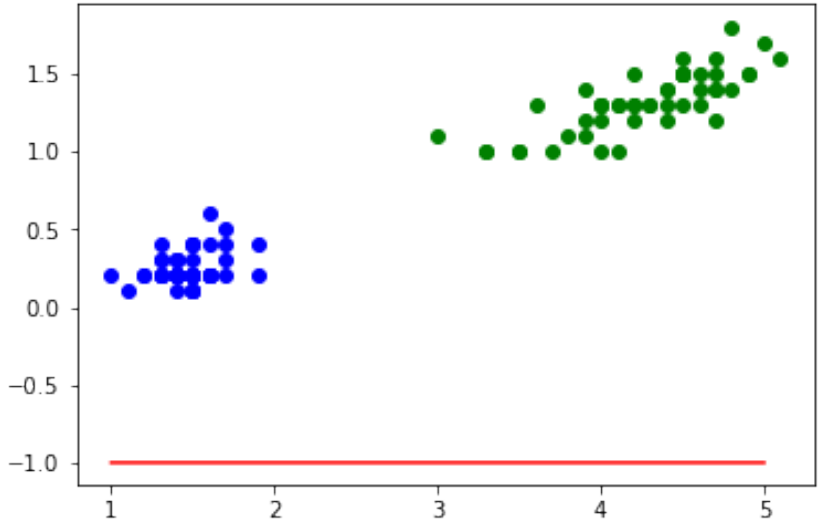
- **Possible SVM problem?** So, we can state the SVM problem as

$$\textbf{Problem 1}: \quad \max_{\beta,c} \min_{i=1,\ldots,n} \frac{|\beta^T x_i + c|}{\|\beta\|}.$$

Issues with Problem 1.

- **Issues with Problem 1**: Recall the Problem 1 where we wanted to maximize the minimum distance of the training data to the hyperplane:

$$\textbf{Problem 1}: \quad \max_{\beta, c} \quad \min_{i=1,\ldots,n} \frac{|\beta^T x_i + c|}{\|\beta\|}.$$



Solving Problem 1 can lead to bad solutions like the red line in the figure above. The line is far from the training data, but does not separate the blue and green dots.

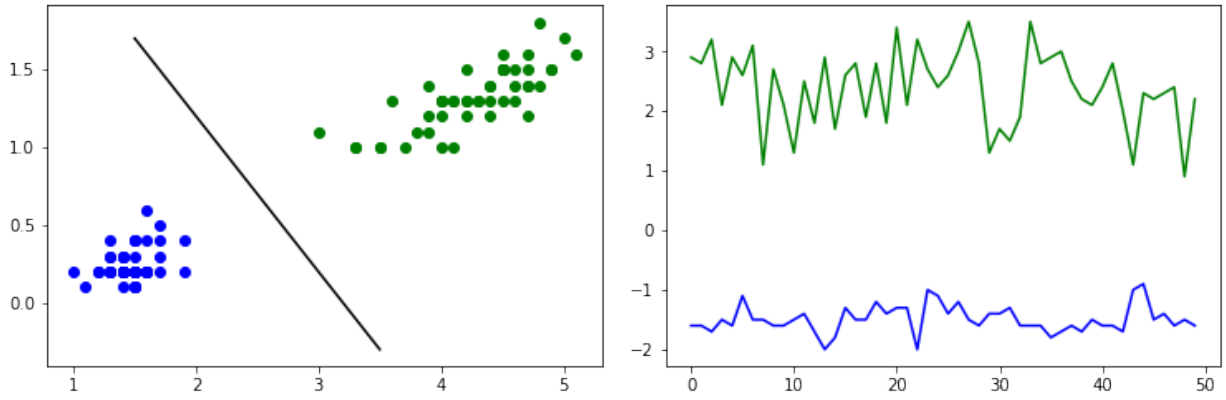
- We need to add constraints to make sure this does not happen.

The Constraints

- Define the function

$$f_{\beta,c}(x) = \beta^T x + c.$$

In the figure below, we have plotted this function for a hyperplane that correctly divides the data. The plots are values of $f_{\beta,c}(x_i)$ for blue and green training data.



- We have the following:

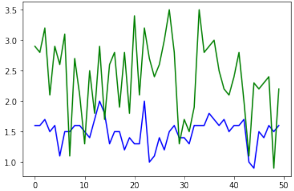
$$\begin{aligned} f_{\beta,c}(x) &= \beta^T x + c = 0, && \text{for } x \text{ on } V_{\beta,c} \\ &= \beta^T x + c > 0, && \text{for } x \text{ to the right of } V_{\beta,c} \\ &= \beta^T x + c < 0, && \text{for } x \text{ to the left of } V_{\beta,c}. \end{aligned}$$

- **Relabel the y values of the classes**: from 0 to -1 while keeping values 1 to 1.
- In the figure below, we have plotted

$$y_i f_{\beta,c}(x_i) = y_i(\beta^T x_i + c)$$

for blue and green training data.

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.
 -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1. -1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```



Note that values for both classes of data are positive. Thus,

$$y_i(\beta^T x_i + c) > 0, \quad \text{for } i = 1, 2, \ldots, n.$$

The SVM Problem?

- **A constrained problem**: Thus, the SVM problem should be posed as

$$\max_{\beta,c} \quad \min_{i=1,\ldots,n} \frac{|\beta^T x_i + c|}{\|\beta\|}$$
$$\text{Subject to} \quad y_i(\beta^T x_i + c) > 0, \quad \text{for } i = 1, 2, \ldots, n.$$

  In words, the SVM problem is to maximize the minimum distance of the training data to the hyperplace, subject to the constraint that the hyperplane divides the data.

- **But, there is another problem**:

$$V_{\beta,c} = V_{2\beta,2c} = V_{3\beta,3c} = V_{a\beta,ac}, \quad \forall a \in \mathbb{R}.$$

  This is because if $\beta^T x + c = 0$, then

$$(a\beta)^T + ac = a\left(\beta^T x + c\right) = a.0 = 0.$$

  Thus, the hyperplane does not have a unique representation. We can change scale both β and c by the same number and get another representation.

- This issue is resolved by normalization.

The SVM Problem, Finally ...

- **Normalized** β: If $\|\beta\| = 1$, then the distance value is

$$\frac{|\beta^T x_i + c|}{\|\beta\|} = |\beta^T x_i + c|.$$

- **The SVM Problem**:

$$\begin{aligned} \textbf{Problem 2:} \qquad & \max_{\beta, c} \quad \min_{i=1,\ldots,n} |\beta^T x_i + c| \\ \text{Subject to} \quad & \beta^T \beta = 1 \\ \text{and} \quad & y_i(\beta^T x_i + c) > 0, \quad \text{for } i = 1, 2, \ldots, n. \end{aligned}$$

  Assumption: This formulation assumed that we can always find hyperplanes that perfectly separate the training points of the two classes. This is called the separable case. If the data is not separable, you just have to solve a slightly different optimization problem. Non-separable case is more common, but this one is easier to understand.

- **(Theoretically) Correct SVM Problem**: The problem above, Problem 2, is the right SVM formulation we are looking for. Theoretically, we do not need to go any further. In practice, we need to solve this problem. With this implementation in mind, we will performs some transformation of the problem.

Modified SVM Problem

- If the data is really separable, then

$$|\beta^T x_i + c| = y_i(\beta^T x_i + c).$$

  Thus, the process of maximization of $\min_{i=1,\ldots,n} y_i(\beta^T x_i + c)$ makes sure that the minimum value is greater than zero. Thus, the constraint $y_i(\beta^T x_i + c) > 0$ will automatically be satisfied.

- **Modified SVM Problem**:

$$\textbf{Problem 3:} \qquad \max_{\beta, c} \quad \min_{i=1,\ldots,n} y_i(\beta^T x_i + c)$$
$$\text{Subject to} \quad \beta^T \beta = 1.$$

The Easily Solvable SVM Problem

It turns out the SVM problem can be further simplified to make it a **quadratic programming** problem. The latter can be solved very efficiently.

Theorem

The SVM problem

$$\textbf{Problem 3:} \qquad \max_{\beta, c} \quad \min_{i=1,\ldots,n} y_i(\beta^T x_i + c)$$
$$\textit{Subject to} \qquad \beta^T \beta = 1.$$

can be solved by solving

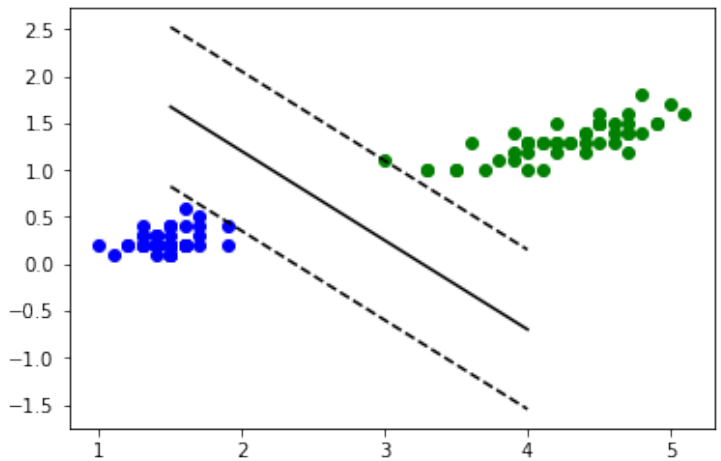
$$\textbf{Problem 4:} \qquad \min_{\beta, c} \quad \beta^T \beta$$
$$\textit{Subject to} \qquad y_i(\beta^T x_i + c) \geq 1, \qquad \textit{for } i = 1, 2, \ldots, n.$$

Note that Problem 3 is a maximization problem while Problem 4 is a minimization problem.

Note: The proofs of this theorem and all the other results will be discussed later.

Intuition Behind Problem 4

- In some books, Problem 4 above is treated as the SVM problem. The theorem states that the two problems are equivalent. Let us discuss the intution behind this problem using which we can also directly state Problem 4 as the SVM problem.



- Consider a hyperplane $V_{\beta,c} = \{x \in \mathbb{R}^p : \beta^T x + c = 0\}$ that separates the blue and green dots. The β selected is of arbitrary norm and not necessarily of norm 1.
- Recall that $\beta^T x + c = 0$ for x on the line, $\beta^T x + c > 0$ for x to the right and $\beta^T x + c < 0$ for x to the left.
- Now suppose we move the hyperplane to the right and to the left so as to touch the first training points on each side (see dashed lines in the figure). These vectors are called **support vectors**. The value of $\beta^T x + c$ at the support vectors is in general arbitrary.

Intuition Behind Problem 4: Continued

- Recall that many different values of β and c lead to the same hyperplane: just scale both by the same number.
- Now scale β and c so that the value $\beta^T x + c = \pm 1$ at the support vectors.
- Since the distance of any training point x_i to the hyperplane $V_{\beta,c}$ is $\frac{|\beta^T x_i + c|}{\|\beta\|}$, the distance of the support vectors to the hyperplane is

$$\frac{|\beta^T x_i + c|}{\|\beta\|} = \frac{1}{\|\beta\|}.$$

  In Problem 4, the objective is to maximize this distance (called margin).
- The constraints

$$y_i(\beta^T x_i + c) \geq 1, \quad \text{for } i = 1, \ldots, n,$$

  makes sure that every feasible hyperplane is scaled so that $\beta^T x + c = \pm 1$ for the support vectors and **it is at least as large for all other vectors**.
- **As mentioned earlier, in many books, SVM is motivated this way and Problem 4 is directly proposed.**

How to Solve the SVM Problem?

- **How to solve the SVM problem?** Now that we have identified

$$\textbf{Problem 4:} \qquad \min_{\beta, c} \quad \beta^T \beta$$
$$\text{Subject to} \qquad y_i(\beta^T x_i + c) \geq 1, \qquad \text{for } i = 1, 2, \ldots, n,$$

  as the problem of interest, how do we solve it?

- **The short answer is using duality theory of convex optimization**: Essentially, we start with the Lagrangian

$$L(\beta, c, \lambda) = \beta^T \beta + \sum_{i=1}^{n} \lambda_i \left(1 - y_i(\beta^T x_i + c)\right)$$

  - We first **minimize** $L(\beta, c, \lambda)$ with respect to β. This can be done by taking the gradient with respect to β and setting it equal to zero.
  - We then **maximize** the minimized Lagrangian with respect to λ.

Constraint Optimization: With Inequality and Equality Constraints

- A constrained optimization problem is defined as

$$\begin{aligned} \min_{x \in \mathbb{R}^p} \quad & f(x) \\ \text{subj. to} \quad & g_i(x) \leq 0, \quad i = 1, \cdots, n \\ \text{and} \quad & h_j(x) = 0, \quad j = 1, \cdots, m. \end{aligned}$$

- **Primal Feasible set**: The set of points satisfying the inequality and equality constraints is called the primal feasible set:

$$D = \{x : g_i(x) \leq 0, \quad h_j(x) = 0, \quad i = 1, \cdots, n, \;\; j = 1, \cdots, m\}.$$

- **Cost of Primal Program**: The cost of primal program is denoted by f^*

$$f^* = \min_{x \in D} f(x).$$

- **Lagrangian**: A central concept in convex optimization is the concept of a Lagrangian. It is defined as

$$L(x, \lambda, \nu) = f(x) + \sum_{i=1}^{n} \lambda_i \; g_i(x) + \sum_{j=1}^{m} \nu_j \; h_j(x).$$

  Here $\lambda = [\lambda_1, \cdots, \lambda_n]$ and $\nu = [\nu_1, \cdots, \nu_m]$ with

$$\lambda_i \geq 0, \quad \forall i.$$

- **Lagrange Multipliers**: The constants $\lambda = [\lambda_1, \cdots, \lambda_n]$ and $\nu = [\nu_1, \cdots, \nu_m]$ are called the Lagrange multipliers. Note that the multipliers $\lambda_i \geq 0$.

Lemma

We have an important equality

$$f^* = \min_{x \in D} f(x) = \min_{x \in \mathbb{R}^p} \max_{\lambda > 0, \nu} L(x, \lambda, \nu),$$

where $\lambda > 0$ *is used to denote* $\lambda_i \geq 0, \quad \forall i.$

- **Proof**: Consider the maximization (note that x is fixed)

$$\max_{\lambda > 0, \nu} L(x, \lambda, \nu) = \max_{\lambda > 0, \nu} \quad f(x) + \sum_{i=1}^{n} \lambda_i \, g_i(x) + \sum_{j=1}^{m} \nu_j \, h_j(x).$$

  If $h_j(x) \neq 0$ for any j or $g_i(x) > 0$ for any i, then

$$\max_{\lambda > 0, \nu} L(x, \lambda, \nu) = \infty.$$

  Thus, only when $x \in D$ we get finite value in the maximization. If $x \in D$, then

$$\max_{\lambda > 0, \nu} f(x) + \sum_{i=1}^{n} \lambda_i \, g_i(x) + \sum_{j=1}^{m} \nu_j \, h_j(x) = \max_{\lambda > 0, \nu} f(x) + \sum_{i=1}^{n} \lambda_i \, g_i(x) = f(x).$$

Weak and Strong Duality

- **Weak Duality**: Interchanging the order of minimization and maximization in the lemma leads to a profound result in constraint optimization called the weak duality:

$$f^* = \min_{x \in \mathbb{R}^p} \max_{\lambda > 0, \nu} L(x, \lambda, \nu) \geq \max_{\lambda > 0, \nu} \min_{x \in \mathbb{R}^p} L(x, \lambda, \nu) = g^*.$$

  This result is true because if we start with $L(x, \lambda, \nu)$

$$\max_{\lambda > 0, \nu} L(x, \lambda, \nu) \geq \min_{x \in \mathbb{R}^p} L(x, \lambda, \nu).$$

  Now, realize that the left hand side is not longer a function of λ, ν and the right hand side is no longer a function of x.

- **Strong Duality** is when we have equality in weak duality:

$$f^* = g^*.$$

  While weak duality is always true, strong duality does not hold in general. The difference $f^* - g^*$ is called the **duality gap**. Thus, strong duality is the case when the duality gap is zero.