Lecture 21: Boosting, Support Vector Classifiers

Reading: Chapters 8, 9.1, 9.2

STATS 202: Data mining and analysis

November 11, 2019

Bagging decision trees

- Draw a bootstrap sample by sampling with replacement.
- Fit a decision tree to each bootstrap replicate (growing the tree, and pruning).
- **Regression:** To make a prediction for an input point x, average the predictions of all the trees:

$$\hat{f}^{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{(b)}(x)$$

- **Classification:** To make a prediction for an input point x_0, take the majority vote from the set of predictions:

$$\hat{y}_0^{(1)}, \ldots, \hat{y}_0^{(B)}.$$

Random Forests

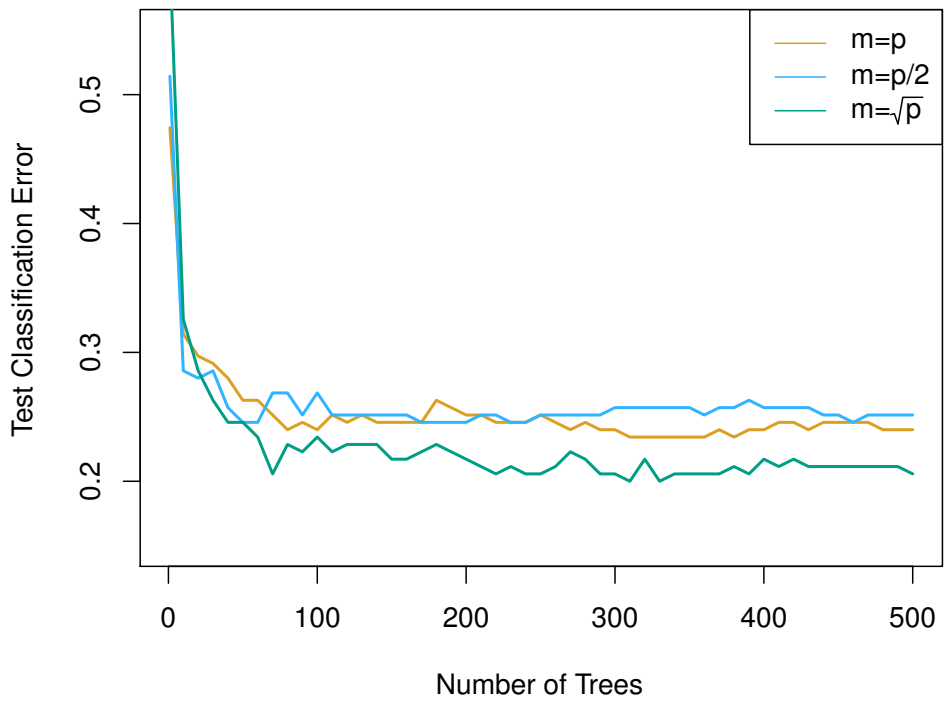
Bagging has a problem:

$\rightarrow$ The trees produced by different Bootstrap samples can be very similar.

Random Forests:

- We fit a decision tree to different Bootstrap samples.
- When growing the tree, we select a random sample of $m < p$ predictors to consider in each step.
- This will lead to very different (or "uncorrelated") trees from each sample.
- Finally, average the prediction of each tree.

Random Forests, choosing m



The optimal m is usually around $\sqrt{p}$,
but this can be used as a tuning parameter.

Boosting

Sequential procedure for combining ("ensembling") decision trees:

1. Choose a *learning rate* λ.
2. Set $\hat{f}(x) = 0$, and $r_i = y_i$ for $i = 1, \ldots, n$.
3. For $b = 1, \ldots, B$, iterate:
   - 3.1 Fit a decision tree $\hat{f}^b$ with d splits to the response $r_1, \ldots, r_n$.
   - 3.2 Update the prediction to:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

   - 3.3 Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

4. Output the final model:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

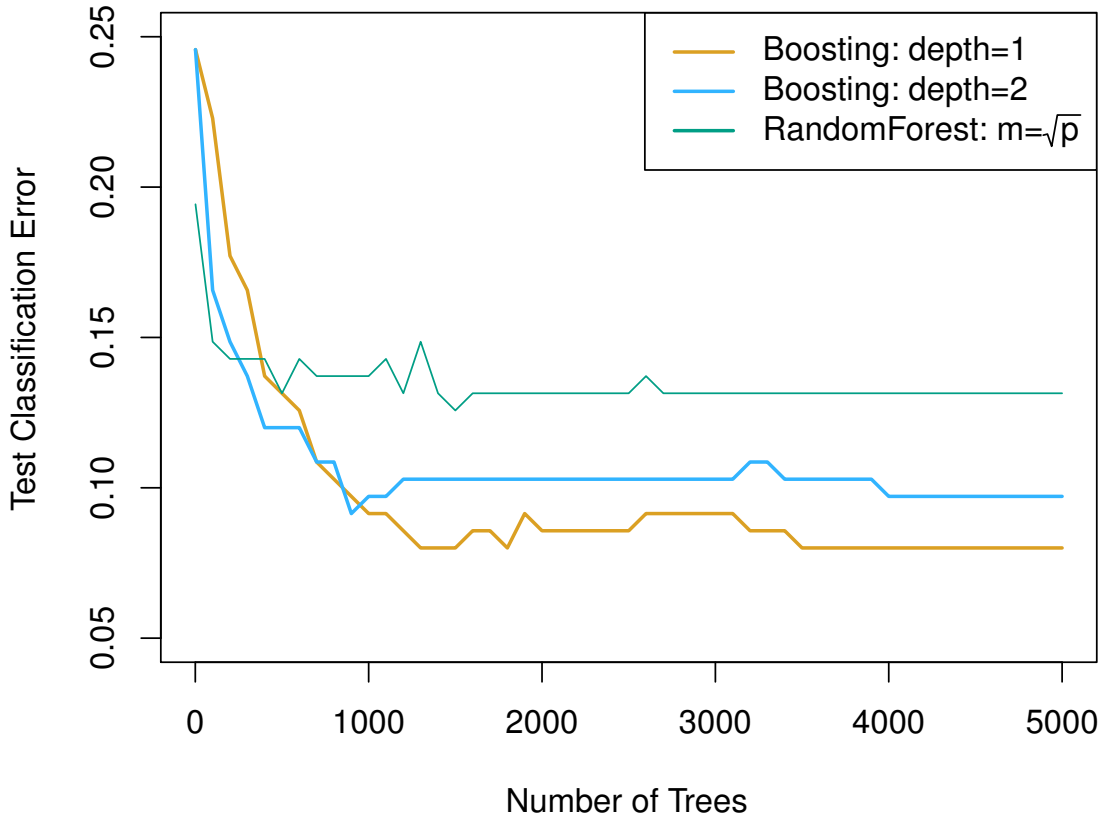
Boosting, intuitively

Boosting learns *slowly*:
We typically fit small (low variance but high bias) trees in each round.

Boosting learns sequentially, trains on errors:
Each iteration learns to predict the mistakes of the previous iteration.

You can boost anything:
You can apply boosting to any predictive learning method, not just decision trees!

Boosting vs. random forests



The parameter $\lambda = 0.01$ in each case.
We can tune the model by CV using λ, d, B.

Hyperplanes and normal vectors

- Consider a p-dimensional space of predictors.
- A **hyperplane** is a (p-1)-dimensional affine space which separates the space into two regions.
- The normal vector $\beta = (\beta_1, \ldots, \beta_p)$, is a unit vector $\sum_{j=1}^{p} \beta_j^2 = 1$ which is perpendicular to the hyperplane.
- If the hyperplane goes through the origin, the (signed) distance between a point $(x_1, \ldots, x_p)$ and the hyperplane is the dot product:

$$x \cdot \beta = x_1\beta_1 + \cdots + x_p\beta_p.$$

- The sign of the dot product tells us on which side of the hyperplane the point lies.

Hyperplanes and normal vectors

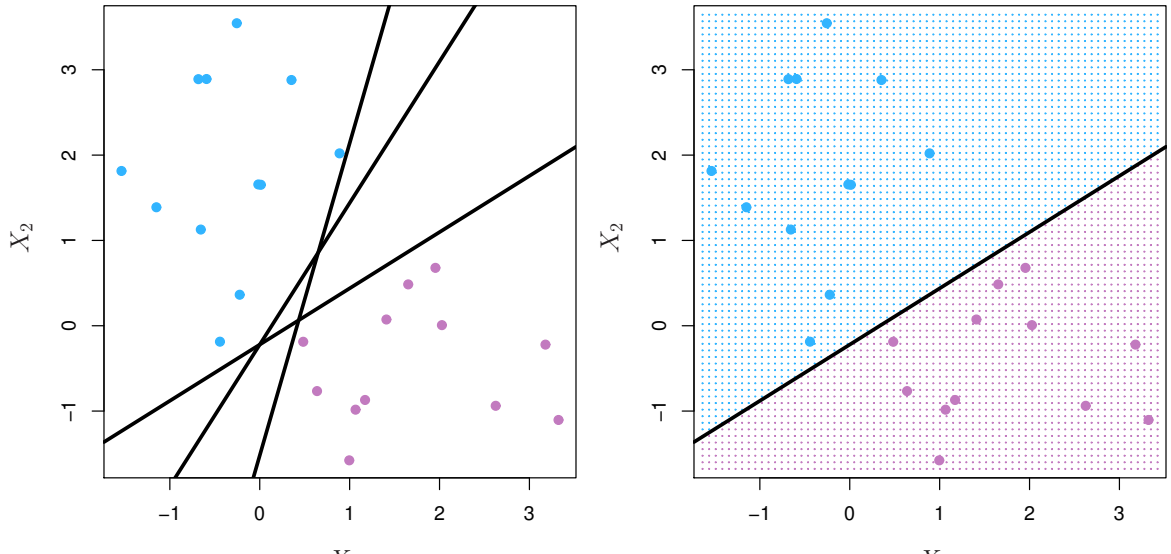
- Consider a p-dimensional space of predictors.
- A **hyperplane** is a (p-1)-dimensional affine space which separates the space into two regions.
- The normal vector $\beta = (\beta_1, \ldots, \beta_p)$, is a unit vector $\sum_{j=1}^{p} \beta_j^2 = 1$ which is perpendicular to the hyperplane.
- If the hyperplane is displaced from the origin by $-\beta_0$ along the normal vector (i.e. it goes through the point $-\beta_0\beta$), then the (signed) distance between a point $(x_1, \ldots, x_p)$ and the hyperplane is:

$$\beta_0 + x_1\beta_1 + \cdots + x_p\beta_p.$$

- The sign tells us on which side of the hyperplane the point lies.

Maximal margin classifier

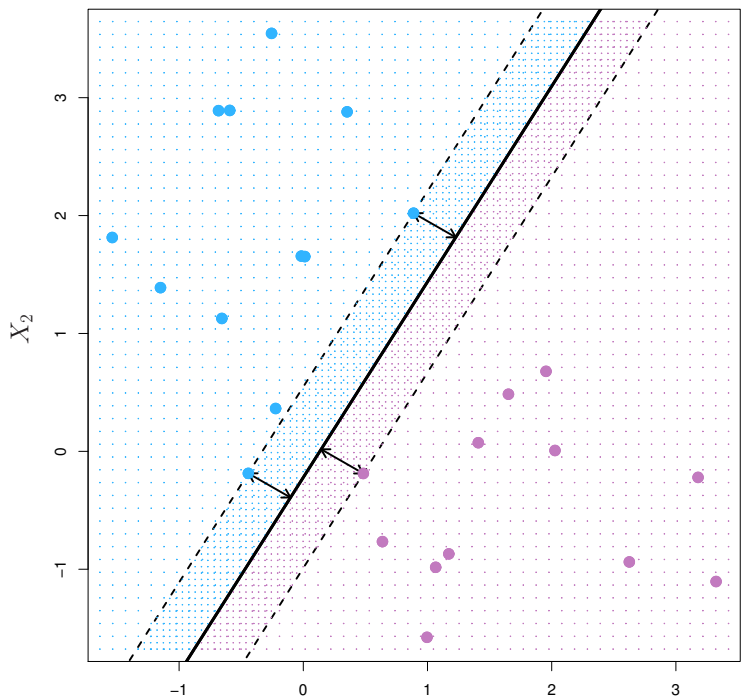
- Suppose we have a classification problem with response $Y = -1$ or $Y = 1$.
- Our goal is to construct a classifier with a linear boundary, i.e. with a **separating hyperplane**.
- If the classes can be separated with a linear boundary, there will typically be an infinite number of hyperplanes separating the classes.
- Which classification rule should we choose?



Maximal margin classifier

Idea: Out of all possible hyperplanes that separate the 2 classes, choose the one with the widest **margin**.

- Margin = minimum distance of any point to hyperplane.
- Margin literally represents your classifier margin of error, how far datapoints must be shifted before classifier makes a mistake
- Hope is that if margin is large, test datapoints will be more likely to fall on correct side of hyperplane.



Maximal margin classifier

This can be written as an optimization problem:

$$\max_{\beta_0,\beta_1,\ldots,\beta_p} \quad M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$\underbrace{y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})}_{\text{How far is } x_i \text{ from the hyperplane}} \geq M \quad \text{for all } i = 1, \ldots, n.$$

M is simply the width of the margin in either direction.

Finding the maximal margin classifier

We can reformulate the problem by defining $b_0 = \beta_0/M$ and a vector $w = (w_1, \ldots, w_p) = \beta/M$:

$$\min_{b_0, w} \quad \frac{1}{2}\|w\|^2$$

subject to

$$y_i(b_0 + w \cdot x_i) \geq 1 \quad \text{for all } i = 1, \ldots, n.$$

This is a quadratic optimization problem.

Finding the maximal margin classifier

$$\min_{b_0, w} \quad \frac{1}{2}\|w\|^2$$

subject to

$$y_i(b_0 + w \cdot x_i) \geq 1 \quad \text{for all } i = 1, \ldots, n.$$

Introducing Karush-Kuhn-Tucker (KKT) multipliers, $\alpha_1, \ldots, \alpha_n$, this is equivalent to:

$$\max_{\alpha} \min_{b_0, w} \quad \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(b_0 + w \cdot x_i) - 1]$$

subject to $\alpha_i \geq 0$.

Finding the maximal margin classifier

$$\max_{\alpha} \min_{b_0, w} \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(b_0 + w \cdot x_i) - 1]$$

subject to $\alpha_i \geq 0$.

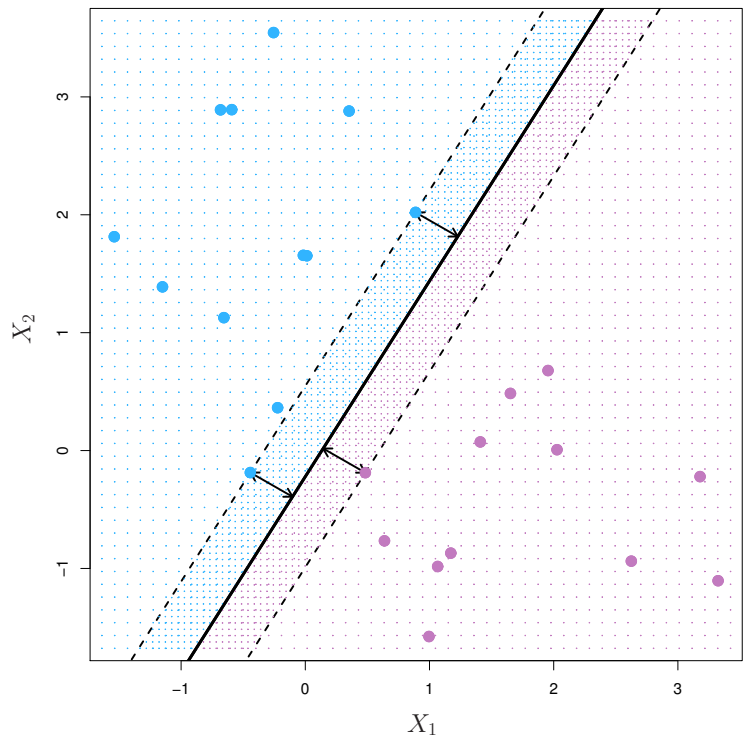
- Setting the partial derivatives with respect to w and b_0 to 0, we get:

$$\hat{w} = \sum_{i=1}^{n} \alpha_i y_i x_i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

- So optimal normal vector $\hat{w}$ is a linear combination of training points!
- Furthermore, one of the KKT conditions yields $\alpha_i > 0$ if and only if $y_i(b_0 + w \cdot x_i) = 1$, that is, if x_i falls on the margin.
- **Take-away:** Optimal normal vector $\hat{w}$ is a linear combination of training points **on the margin**!

Support vectors

The vectors that fall on the margin are called **support vectors**; the maximal margin classifier is fully determined by the support vectors:



Finding the maximal margin classifier

$$\max_{\alpha} \min_{b_0, w} \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(b_0 + w \cdot x_i) - 1]$$

$$\text{subject to} \quad \alpha_i \geq 0.$$

If we plug in the optimality condition $\hat{w} = \sum_{i=1}^{n} \alpha_i y_i x_i$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$, we obtain the dual optimization problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} (x_i \cdot x_{i'})$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad \sum_{i} \alpha_i y_i = 0.$$

Take-away: Maximal margin classifier only depends on input vectors via their dot products!

Summary

We've reduced the problem of finding w, which describes the hyperplane and the size of the margin, to finding a set of coefficients $\alpha_1, \ldots, \alpha_n$ through:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} (x_i \cdot x_{i'})$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad \sum_{i} \alpha_i y_i = 0.$$

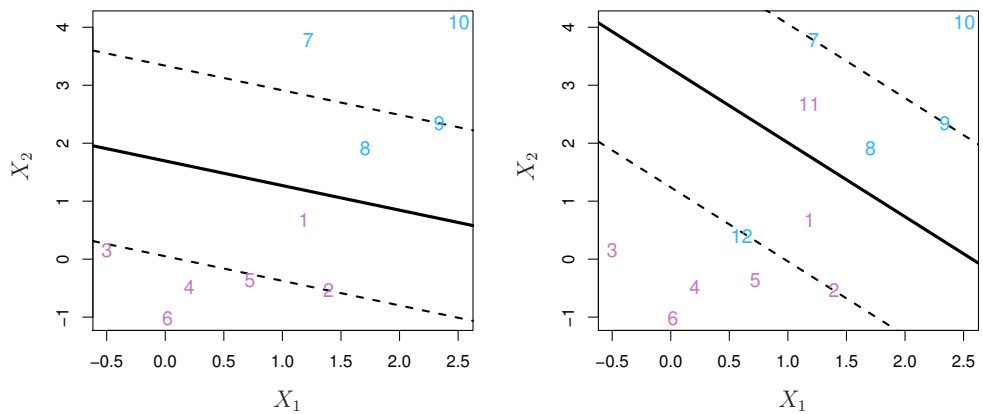
This only depends on the training sample inputs through the inner products $x_i \cdot x_j$ for every pair i, j.

Support vector classifier

Problem: It is not always possible to separate the points using a hyperplane.

Support vector classifier:

- Relaxation of the maximal margin classifier.
- Allows a number of points points to be on the wrong side of the margin or even the hyperplane.



Support vector classifier

This can be written as an optimization problem:

$$\max_{\beta_0,\beta,\epsilon} \quad M$$

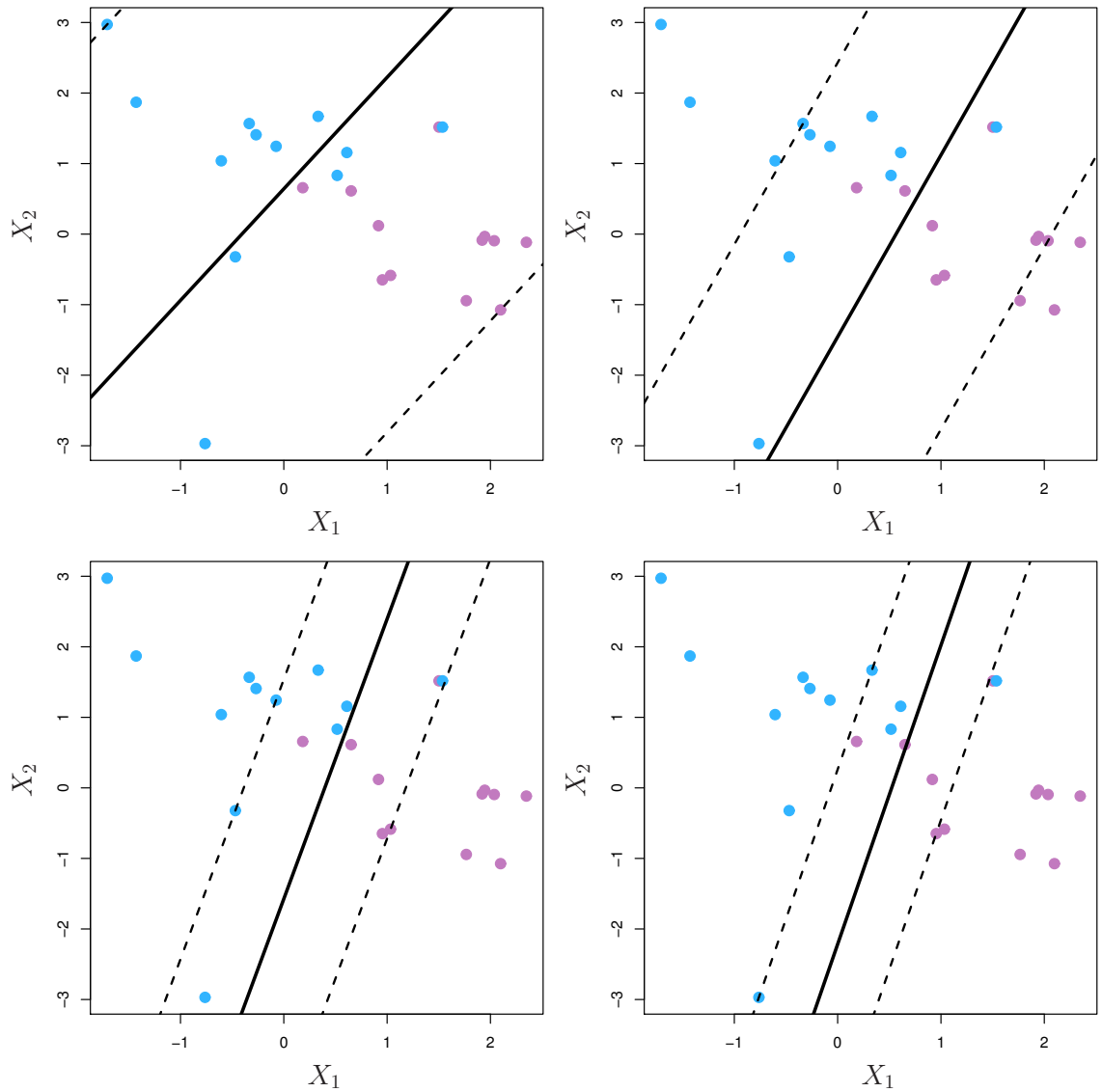
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$\underbrace{y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})}_{\text{How far is } x_i \text{ from the hyperplane}} \geq M(1-\epsilon_i) \quad \text{for all } i = 1, \ldots, n$$

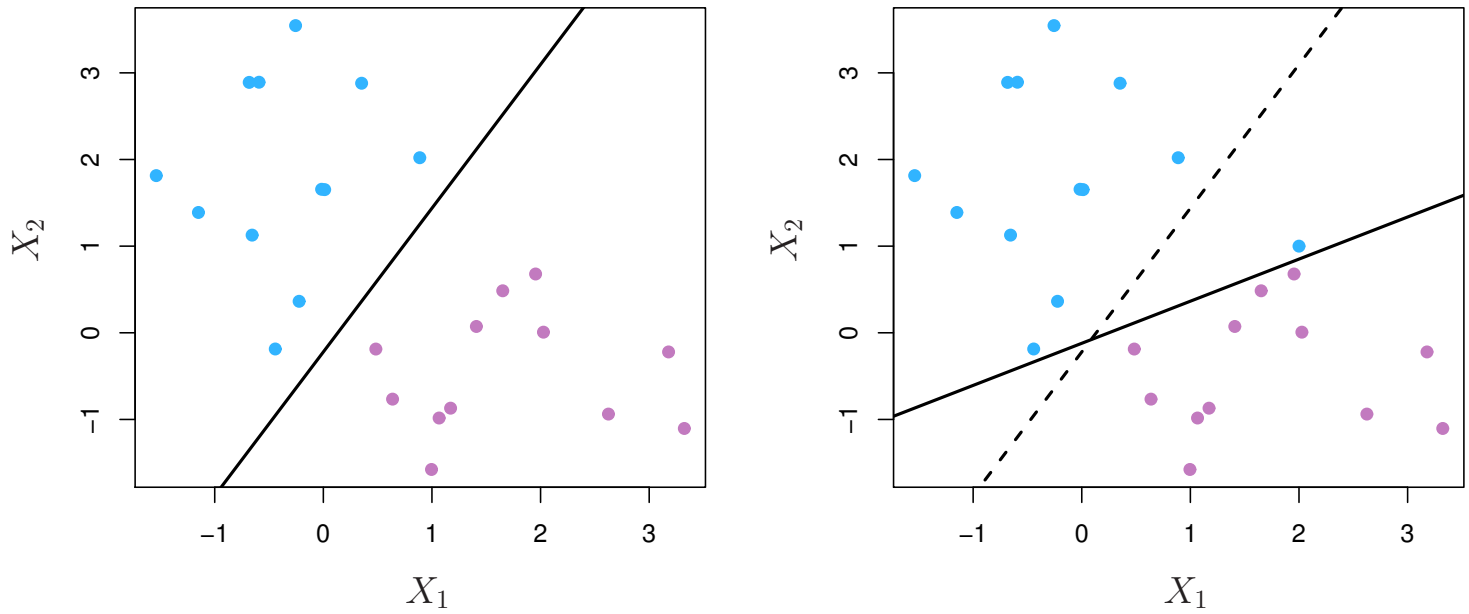
$$\epsilon_i \geq 0 \text{ for all } i = 1, \ldots, n, \quad \sum_{i=1}^{n} \epsilon_i \leq C.$$

M is the width of the margin in either direction.
$\epsilon = (\epsilon_1, \ldots, \epsilon_n)$ are called *slack* variables.
C is called the *budget*.

Tuning the budget, C (high to low)



If the budget is too low, we tend to overfit



Maximal margin classifier, $C = 0$. Adding one observation dramatically changes the classifier.

Take-away: Higher budget reduces variance, increases robustness to variation in dataset.

Finding the support vector classifier

We can reformulate the problem by defining $b_0 = \beta_0/M$ and a vector $w = (w_1, \ldots, w_p) = \beta/M$:

$$\min_{b_0, w, \epsilon} \quad \frac{1}{2}\|w\|^2 + D\sum_{i=1}^{n} \epsilon_i$$

subject to

$y_i(b_0 + w \cdot x_i) \geq (1 - \epsilon_i)$ for all $i = 1, \ldots, n$,

$\epsilon_i \geq 0$ for all $i = 1, \ldots, n$.

The penalty $D \geq 0$ serves a function similar to the budget C, but is inversely related to it.

Finding the support vector classifier

$$\min_{b_0,w,\epsilon} \quad \frac{1}{2}\|w\|^2 + D\sum_{i=1}^{n}\epsilon_i$$

subject to

$y_i(b_0 + w \cdot x_i) \geq (1 - \epsilon_i)$ for all $i = 1, \ldots, n$.

$\epsilon_i \geq 0$ for all $i = 1, \ldots, n$.

Introducing Karush-Kuhn-Tucker multipliers, α_i and μ_i, this is equivalent to:

$$\max_{\alpha,\mu} \min_{b_0,w,\epsilon} \quad \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i[y_i(b_0 + w \cdot x_i) - 1 + \epsilon_i] + \sum_{i=1}^{n}(D - \mu_i)\epsilon_i$$

subject to $\alpha_i \geq 0, \mu_i \geq 0,$ for all $i = 1, \ldots, n$.

Finding the support vector classifier

$$\max_{\alpha,\mu} \min_{b_0,w,\epsilon} \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i[y_i(b_0 + w \cdot x_i) - 1 + \epsilon_i] + \sum_{i=1}^{n}(D - \mu_i)\epsilon_i$$

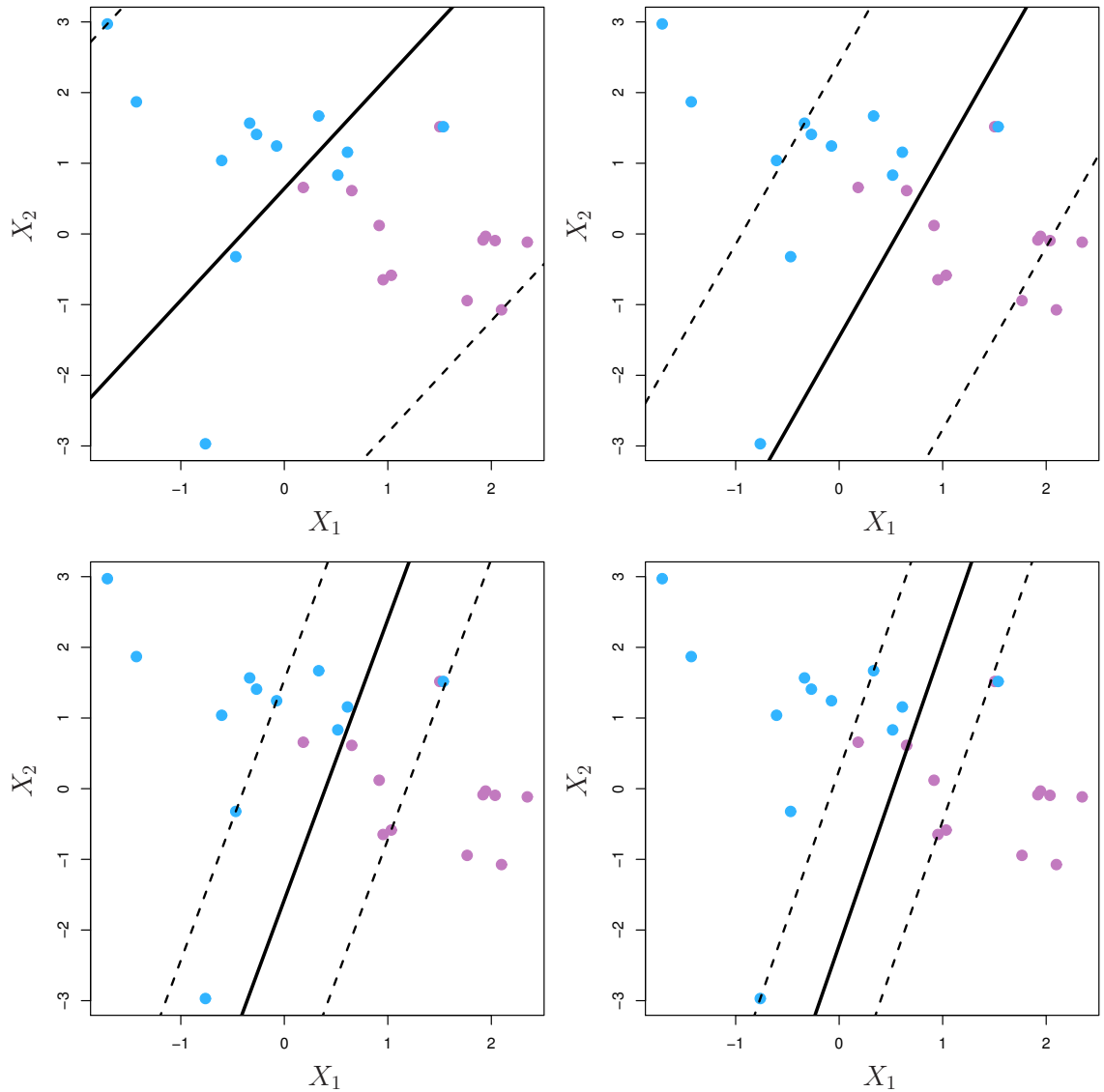
subject to $\alpha_i \geq 0, \mu_i \geq 0, \quad$ for all $i = 1, \ldots, n$.

- Setting the derivatives with respect to w, b_0, and ϵ to 0, we obtain :

$$\hat{w} = \sum_{i=1}^{n} \alpha_i y_i x_i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \mu_i = D - \alpha_i$$

- Furthermore, the KKT conditions yield $\alpha_i > 0$ if and only if $y_i(b_0 + w \cdot x_i) \leq 1$, that is, if x_i falls on the wrong side of the margin.
- **Take-away:** Optimal normal vector $\hat{w}$ is a linear combination of training points that are **on the margin or violate the margin** (these are support vectors!)

Support vectors



The problem only depends on $x_i \cdot x_{i'}$

As with the Maximal Margin Classifier, the problem can be reduced to finding $\alpha_1, \ldots, \alpha_n$:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} (x_i \cdot x_{i'})$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq D \quad \text{for all } i = 1, \ldots, n,$$

$$\sum_{i} \alpha_i y_i = 0.$$

As before, this only depends on the training sample inputs through the inner products $x_i \cdot x_j$ for every pair i, j.