Classification: Support Vector Machines and Kernels

Siva Balakrishnan
Data Mining: 36-462/36-662

February 12th, 2018

Chapter 9.2 - 9.5 of ISL

A Quick Announcement

Recap: Naïve Bayes

- Another generative classifier, i.e. we estimate $f_k(x) := \mathbb{P}(X = x|Y = k)$, and $\pi_k := \mathbb{P}(Y = k)$ and classify according to:

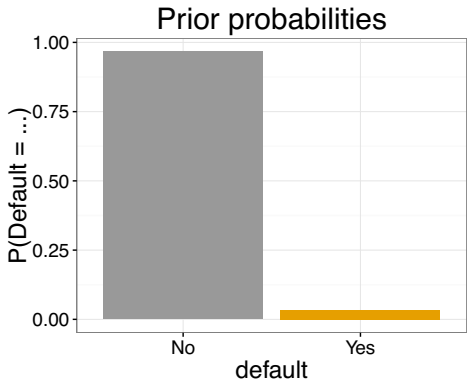
$$\widehat{f}(x) = \arg\max_k f_k(x)\pi_k.$$

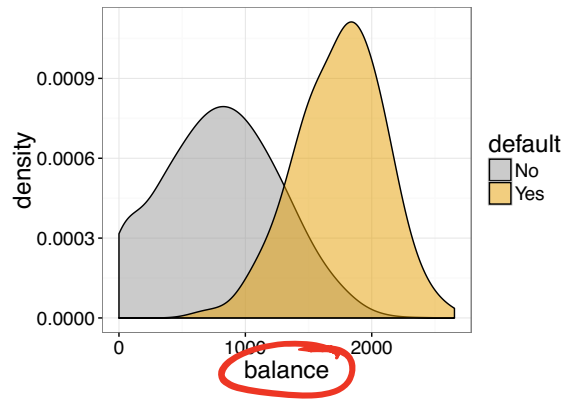
- In Naïve Bayes, we model that the features $X_1, \ldots, X_p$ are *independent conditional on the class label*.
- This means that:

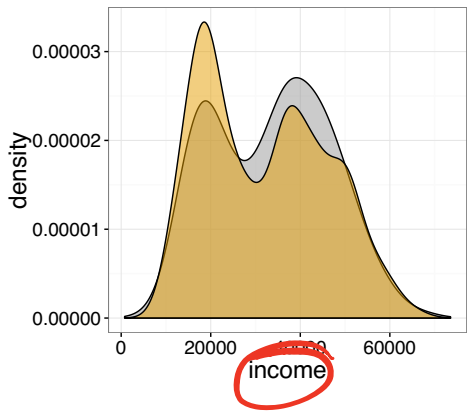
$$f_k(x) = \prod_{i=1}^{p} \mathbb{P}(X_i = x_i|Y = k),$$

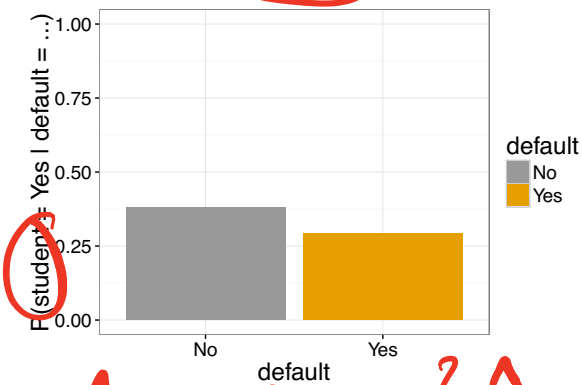
so we only need to estimate or model p univariate distributions per class. *Estimating univariate distributions is much much much easier than estimating a high-dimensional multivariate distribution.*

Recap: Naïve Bayes









- To classify a new point we use:

$$\widehat{f}_{\text{Yes}} = \widehat{f}_{\text{Yes}}(\text{income})\widehat{f}_{\text{Yes}}(\text{balance})\widehat{f}_{\text{Yes}}(\text{student})$$

$$\widehat{f}_{\text{No}} = \widehat{f}_{\text{No}}(\text{income})\widehat{f}_{\text{No}}(\text{balance})\widehat{f}_{\text{No}}(\text{student})$$

$\hat{f}_{yes} \cdot \hat{\pi}_{yes} \gtrless \hat{f}_{No} \hat{\pi}_{No}$

Recap: Gaussian Naïve Bayes

- When the covariates are all continuous, one version of the Naïve Bayes classifier assumes that:

$$f_k(X_i) \sim N(\mu_{ik}, \sigma_i^2),$$

  i.e. that each feature is (univariate) Gaussian with common variance across classes. This is called the *Gaussian Naïve Bayes* classifier.

- This classifier is identical to LDA with a *diagonal* covariance matrix. Even in this case, the *independence assumption* of Naïve Bayes reduces the number of parameters we need to estimate.

- More generally, Naïve Bayes uses a flexible model for the univariate distributions (which lowers bias relative to LDA) but assumes the features are independent given the label (which increases bias).

Recap: Hyperplanes

- Many classifiers we have seen so far have the form:

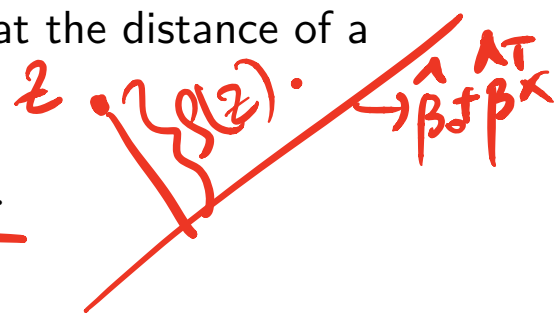
$$\widehat{f}(x) = \begin{cases} 1 & x^T\widehat{\beta} + \widehat{\beta}_0 > 0 \\ -1 & x^T\widehat{\beta} + \widehat{\beta}_0 < 0 \end{cases} = \text{sign}(x^T\widehat{\beta} + \widehat{\beta}_0)$$

- Hyperplanes are unchanged if we multiply all the coefficients by some positive number. We will adopt the convention that:

$$\sum_{j=1}^{p} \widehat{\beta}_j^2 = 1.$$

- With this standardization, we showed that the distance of a point z to the hyperplane is given by:

$$\rho(z) := |z^T\widehat{\beta} + \widehat{\beta}_0|.$$



Recap: Support Vector Machines – Linearly Separable

- When the data is linearly separable the SVM just attempts to draw a line through the middle – i.e. one that has largest distance to the closest point.
- This distance of a plane to the closest point is sometimes called the *margin*. So SVMs are sometimes called "maximum-margin classifiers".
- (Recall $y \in \{-1, 1\}$). To find the SVM classifier in the separable case we solve the following optimization problem:

$$\text{Maximize } M$$

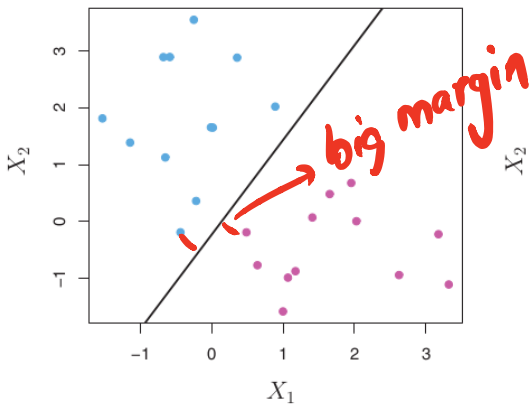
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

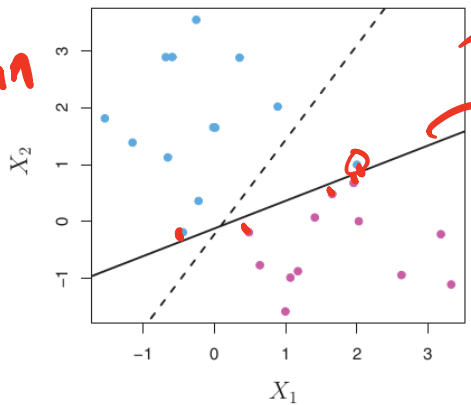
$$y_i(\beta_0 + x_i^T \beta) \geq M.$$

Since in the separable case, $y_i(\beta_0 + x_i^T \beta) \geq 0$, this is just the margin of the i-th point.

Recap: Support Vector Machines – Main Issues

- When the data is not linearly separable, there is no solution to the optimization problem with $M \geq 0$.
- The other problem is that the hard-margin SVM (i.e. the SVM on separable data) can produce undesirable results.





(ISL pg.345)

We'd like to sometimes allow a few mis-classifications as long as our final classifier has a large margin (i.e. we'd like to be able to trade these off).

Support vector classifier

ϵ_i – slack variables.

$$\text{Maximize}_{M,\beta,\beta_0,\epsilon}\ M$$

$$\text{subject to } \sum_{j=1}^{p}\beta_j^2 = 1,\ \varepsilon_i \geq 0,\ \sum_{i=1}^{n}\varepsilon_i \leq C$$

C → tuning parameter

$$y_i(\beta_0 + x_i^T\beta) \geq M(1-\varepsilon_i)$$

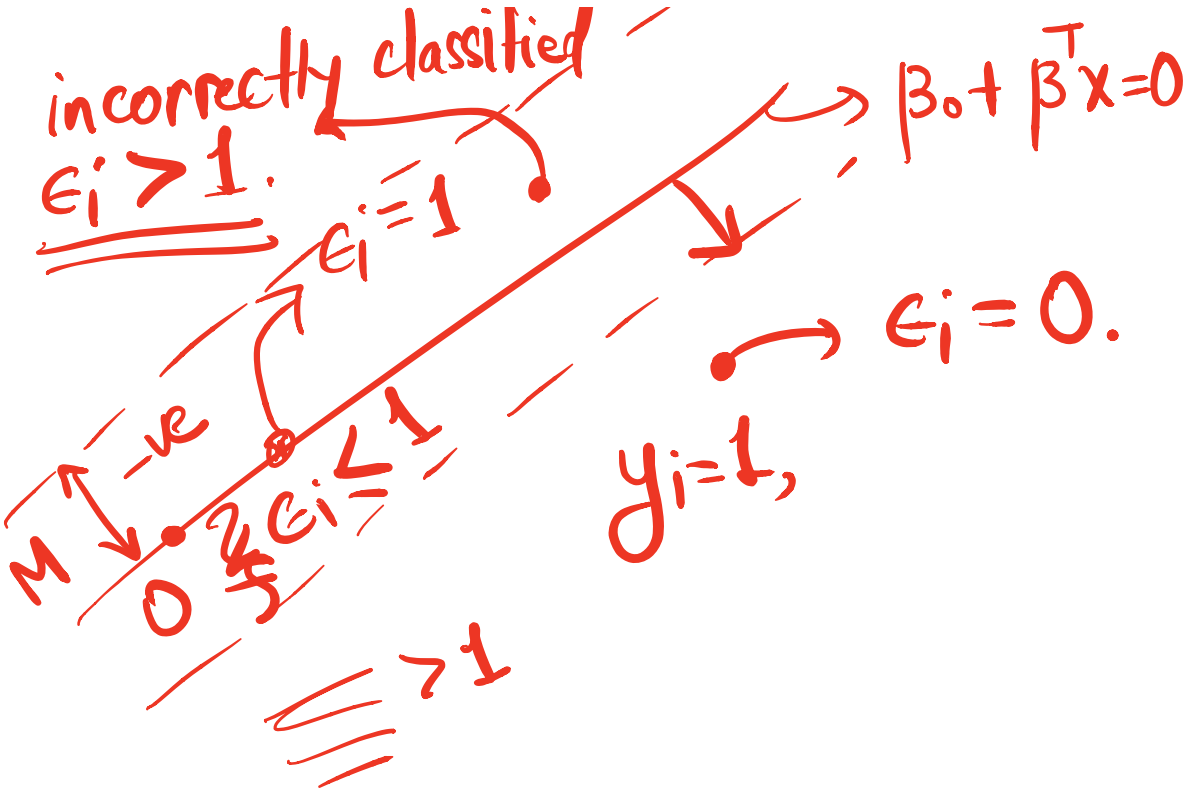
if $\epsilon_i > 0$ then margin $(x_i) < M$.

- Parameter C determines "softness" of the margin. Big C makes it easier to cross. In particular, no more than C observations cross because...

  can be $\epsilon_i \geq 0$, $\epsilon_i > 1$.

- Variable ε_i encodes point location: $\varepsilon_i = 0$ outside margin, $\varepsilon_i > 0$ inside margin, $\varepsilon_i > 1$ across boundary.

If $\epsilon_i = 1 \Rightarrow y_i(\beta_0 + \beta^T x_i) = 0 \Rightarrow \beta^T x_i + \beta_0 = 0.$



→ C bounds the # of training errors.

– every mis-classified $\epsilon_i > 1$.

more than C

$\Rightarrow \sum_i \epsilon_i > C$. :-C

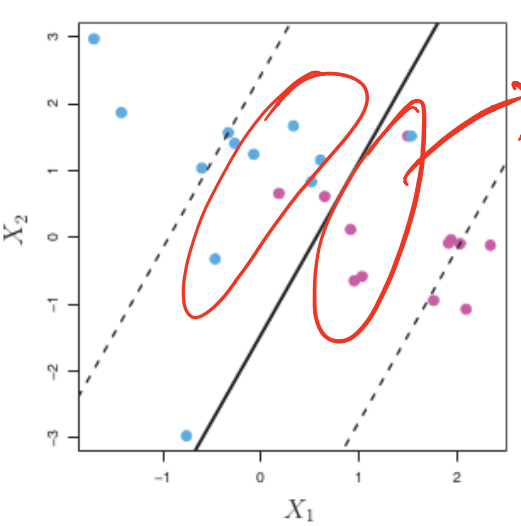
big C → large M but meaningless.

↓

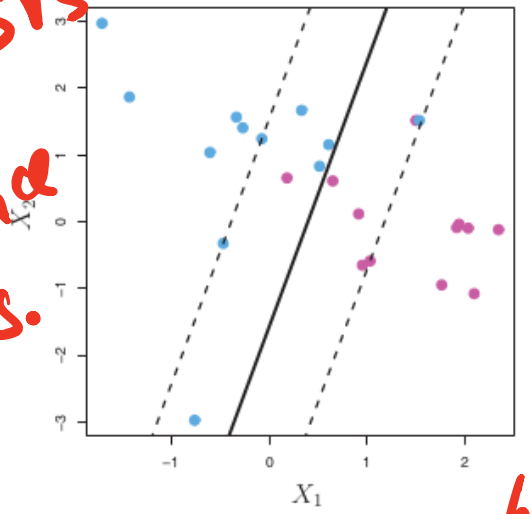
large # of SVs

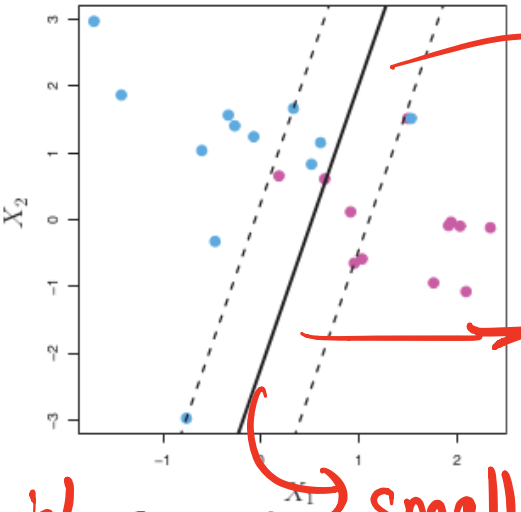
↓

low variance high bias.



all support vectors





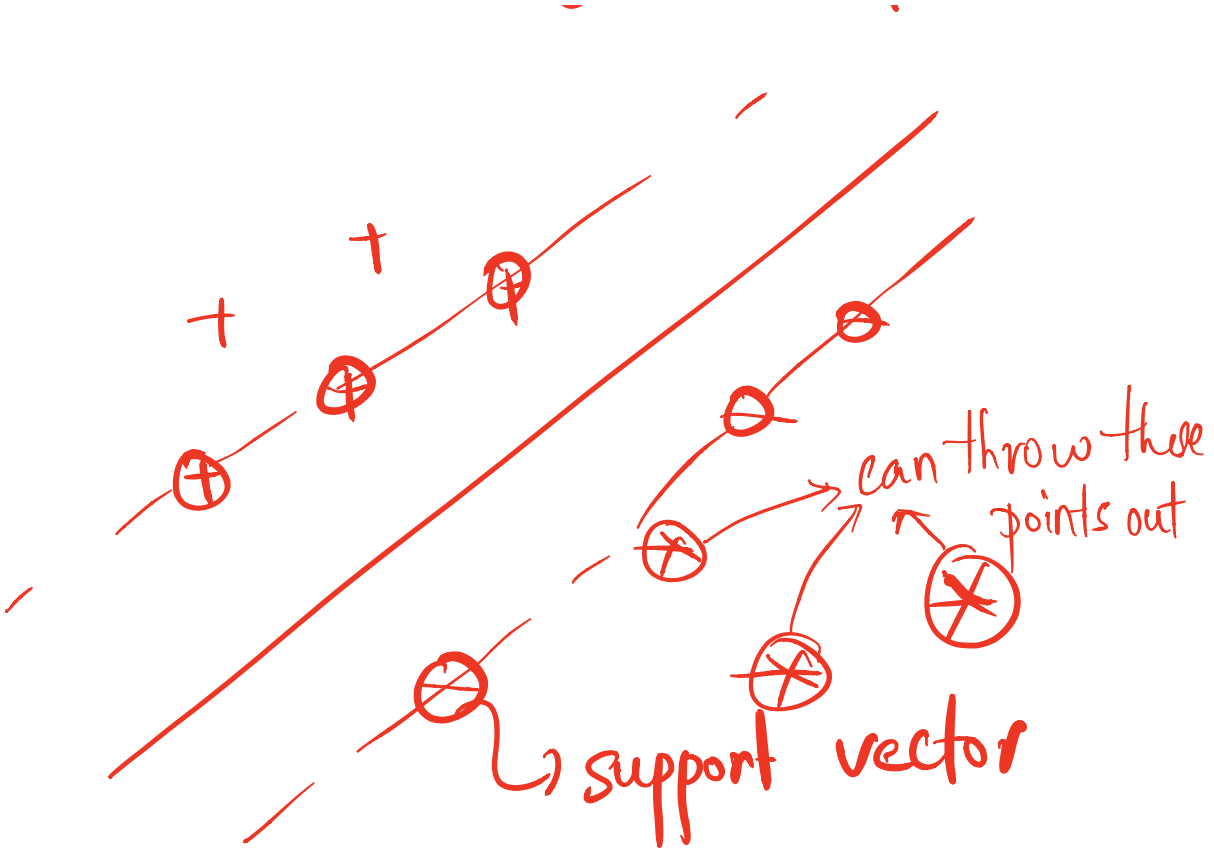
→ Small C.

→ Smaller margin

→ small number of SVs.

higher variance.

(ISL pg. 348)


can throw these points out
support vector

The parameter C trades off bias and variance.

It essentially determines how far from the boundary we should be looking when forming our line.

We can tune C using cross-validation.

We now have yet another linear classifier. This one requires even fewer assumptions, but still allows tuning and gives reasonable answers.

How do these linear classifiers compare?

Though the decision rules are all linear, these methods produce different classifications because they rely on different assumptions.

In order from strongest to weakest assumptions:

1. LDA: Works well if the groups are in "clumps" so that the Gaussian distribution is reasonable. Also assumes the shapes are similar.

2. Logistic: Only requires that $\log \frac{P(Y=1|X=x)}{P(Y=0|X=x)} \approx x_i^T \beta$, which is strictly weaker. LDA will do better when it's assumptions are reasonable, but otherwise worse. Logistic focuses more on the boundary points.

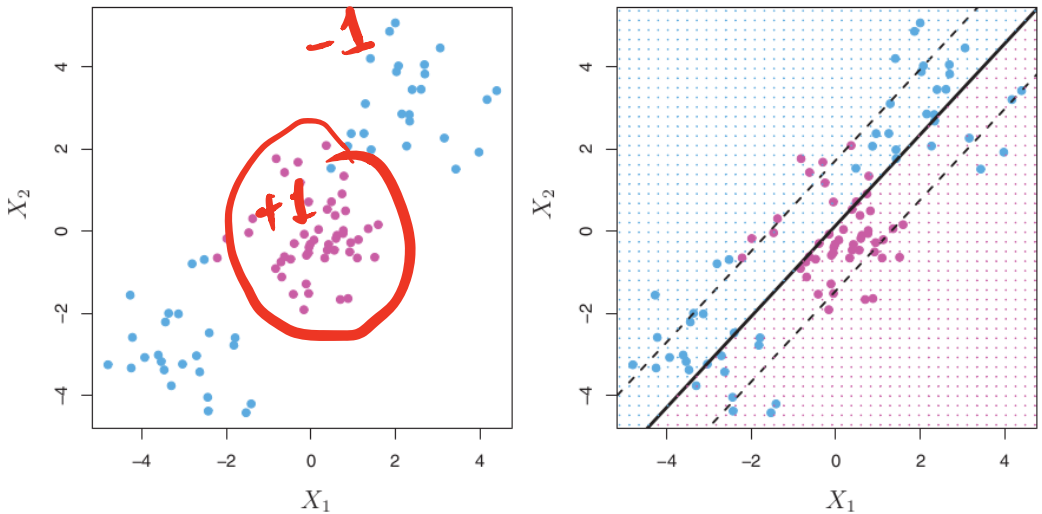
3. SVM: The fewest assumptions, and entirely focuses on points near the boundary. Can work well with the other models are far from reasonable. Also most common to use kernels in this one.

Multiclass SVM

- Unlike logistic regression and LDA there is no particularly natural way to take the (binary) SVM and use it in multi-class settings.
- Two popular methods are:
  1. One-versus-all classification: Here we fit K different SVMs $\{(\widehat{\beta}_{01}, \widehat{\beta}_1), \ldots, (\widehat{\beta}_{0K}, \widehat{\beta}_K)\}$ by comparing each class to all the other classes.
     To finally classify a point we use:
     $$\widehat{f}(x) = \arg\max_k \widehat{\beta}_{0k} + \widehat{\beta}_k^T x.$$
  2. One-versus-one classification: Here we fit $\binom{K}{2}$ different SVMs, by comparing each class with every other class.
     To classify a new test example: we classify it according to each of the classifers and pick the class that is chosen most often.

Non-Linear SVMs

Support Vector Classifiers give linear boundaries. What are we going to do if we want something non-linear?



(ISL pg.349)

Non-Linear SVMs

feature expansion.

We could start making up transformations of the x values. For example, polynomials in the x, like $x_1^2, x_2^2, x_1 x_2$.

$(x_1, x_2) \rightarrow (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

Linear boundaries in this higher dimensional space of transformed x values are equivalent to non-linear boundaries in lower dimensional space. You've seen this before in regression when adding terms like quadratics.

In high dimensions, this can be a lot to keep track of and to compute. Even constructing reasonable expansions can be difficult.

The "kernel trick" gives a convenient way around this.

Feature Expansions

- Suppose we added quadratic features and solved the SVM then we would solve:

$$\text{Maximize}_{M,\beta,\widetilde{\beta},\beta_0,\epsilon} \; M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 + \sum_{j=1}^{p} \sum_{k=1,k>j}^{p} \widetilde{\beta}_{jk}^2 = 1, \; \varepsilon_i \geq 0, \; \sum_{i=1}^{n} \varepsilon_i \leq C$$

$$y_i(\beta_0 + x_i^T \beta + \sum_{j=1}^{p} \sum_{k=1,k>j}^{p} \widetilde{\beta}_{jk} x_{ik} x_{ij}) \geq M(1 - \varepsilon_i)$$

- We now have to optimize over roughly p^2 variables, and this can explode very quickly (if we keep adding new features).

Kernelization - Prelude

- The kernel trick, or kernelization, is roughly a way to obtain non-linear methods from linear ones without the extra computational burden of doing big feature expansions.
- At some point, a big fraction of the ML papers published were on "kernelizing" various linear algorithms.
- It was a very mysterious trick for a while, and worked quite well in practice (somewhat like deep learning today).

Kernelization

- Before we get to kernelization, we need a simple fact: the optimal solution to the SVM problem (in both the separable and non-separable cases) is of the form:

$$\widehat{\beta} = \sum_{i=1}^{n} \alpha_i x_i,$$

Representer thm. $x_i \in \mathbb{R}^p$, $\alpha_i \in \mathbb{R}$.

where $x_i \in \mathbb{R}^p$ are the training data. This fact is true if the vectors x_i span $\mathbb{R}^p$ (since then every vector can be written in this form) but is also more generally true.

- This in turn means that the SVM hyperplane can be written in the form:

$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}^T x = \hat{\beta}_0 + \left(\sum_{i=1}^{n} \alpha_i x_i\right)^T x.$

$$\widehat{f}(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i x_i^T x.$$

inner products between x & x_i.

- This has many nice implications. One simple one is that now instead of optimizing over β in the SVM program we can optimize over the coefficients α. This is often called the *dual* form of the SVM.

Kernelization continued

- If we know that, $\widehat{\beta} = \sum_{i=1}^{n} \alpha_i x_i$, then we can re-write the SVM program:

$$\text{Maximize}_{M,\beta,\beta_0,\epsilon}\ M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,\ \varepsilon_i \geq 0,\ \sum_{i=1}^{n} \varepsilon_i \leq C$$

$$y_i(\beta_0 + x_i^T \beta) \geq M(1 - \varepsilon_i)$$

as

$$\text{Maximize}_{M,\alpha,\beta_0,\epsilon}\ M$$

$$\text{subject to } \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j x_i^T x_j = 1,\ \varepsilon_i \geq 0,\ \sum_{i=1}^{n} \varepsilon_i \leq C$$

$$y_i(\beta_0 + \sum_{j=1}^{n} \alpha x_i^T x_j) \geq M(1 - \varepsilon_i).$$

Curious fact:

The kernel trick finally

$(x_1, x_2) \rightarrow (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Suppose we knew a nice, high dimensional transformation $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and we wanted to fit a linear SVM in that $\mathbb{R}^m$ space to get better separation.

We would still only need the inner products in that higher dimensional space!

What if we cheat? Can we just compute the inner products directly, without ever forming the high-dimensional vectors?

$x_i^T x_j \rightarrow$ in bigger space.

$x_i^T x_j$ — measuring how similar are x_i & x_j.

We define a kernel function – *just an inner product in a higher dimensional space* – which measures the similarity of two observations. For example:

Linear kernel: $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$ is our usual inner product

Polynomial kernel: $K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$

Radial kernel: $K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$

All of these correspond to forming higher dimensional vectors by transforming x, and then taking usual inner products. However, we never need to actually form the vectors!

Example: Polynomial kernel of order 2

Suppose we observe points (x_1, x_2), but we want to work in the space

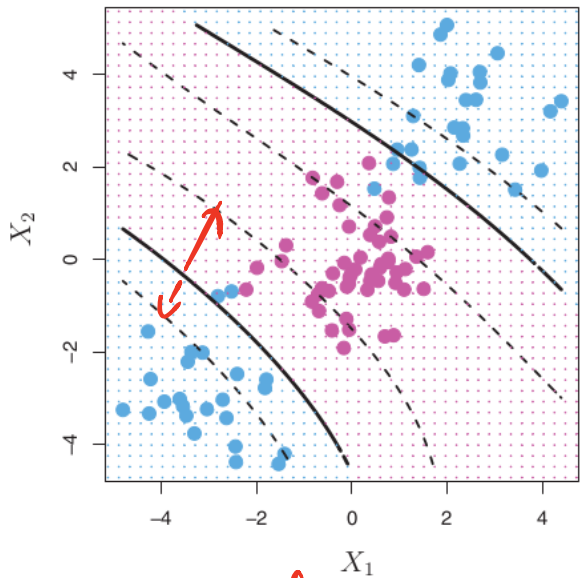
$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2),$$

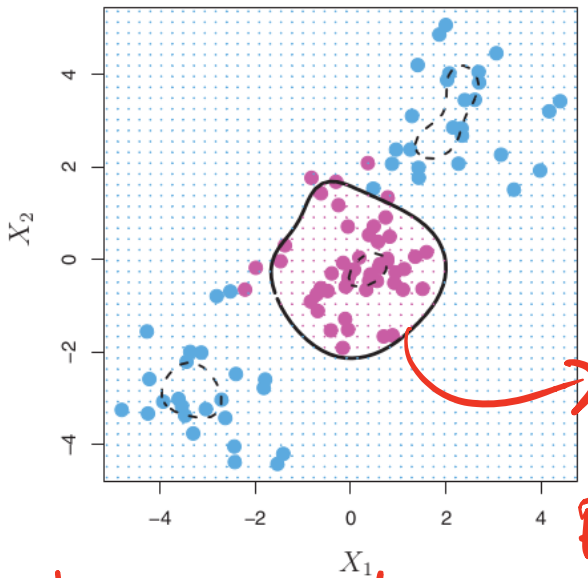
so that we can have quadratic boundaries. (Constants are just chosen to make the math nice)

Compare taking inner products in the larger space to the kernel $K(x, y) = (1 + \sum_{i=1}^{p} x_i y_i)^2$

22

We can look for good linear boundaries in the high dimensional space (with no new math!), which will correspond to non-linear boundaries in our new space.



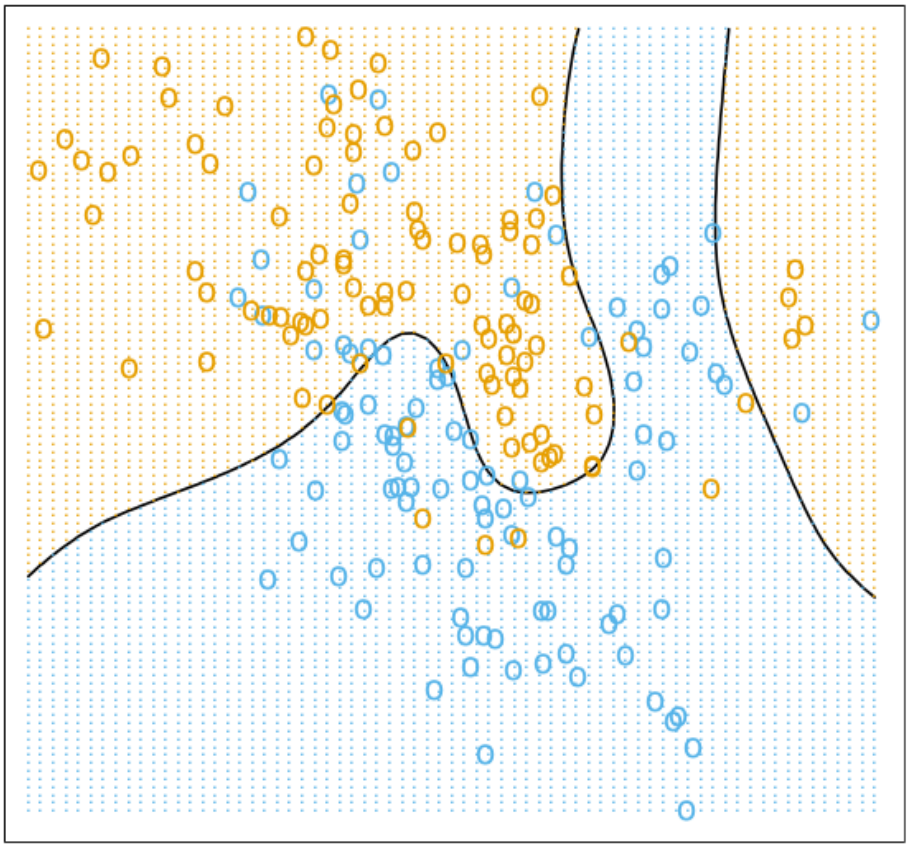


SVM with RBF kernel.

↳ SVM with polynomial kernel

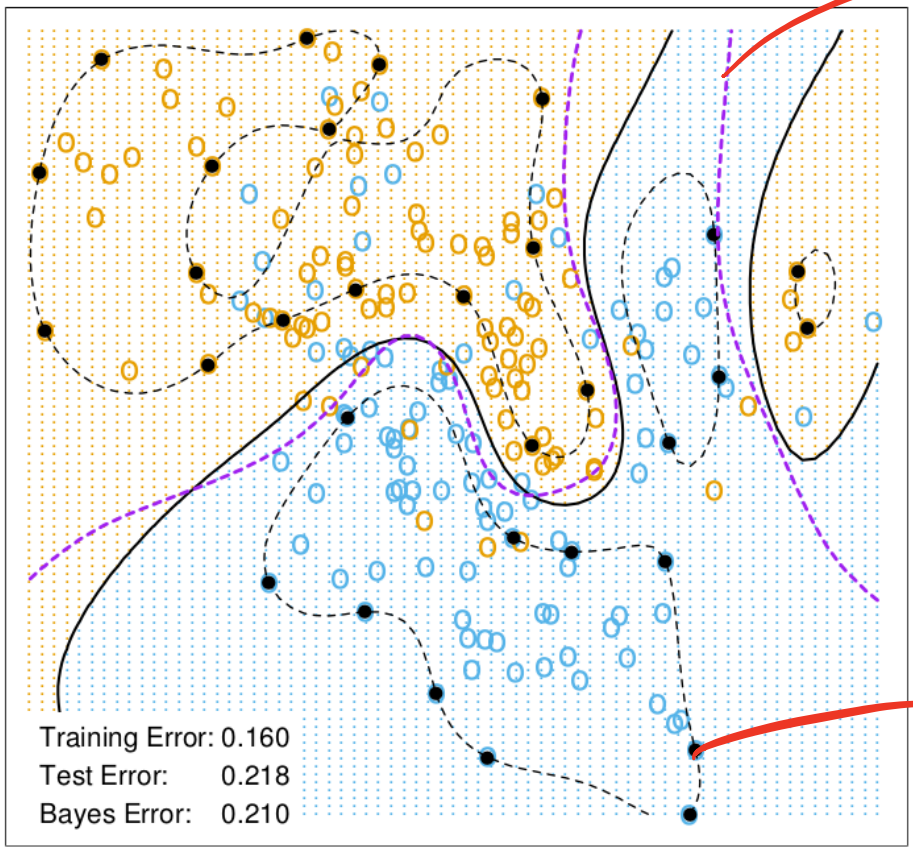
(ISL pg. 353)

The end result can be quite flexible



(ESL pg. 21)

The end result can be quite flexible



Best possible classifier

support vectors.

(ESL pg. 425)

How should we think about this?

- Our hyperplane used to have the form:

$$\widehat{f}(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i x_i^T x.$$

- Once we kernelize it has the form:

$$\widehat{f}(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i K(x_i, x).$$

Imagine we use an RBF kernel:

$$K(x, x') = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$$

This function is tiny if x and x' are far apart, so the classifier is doing something quite intuitive – it is essentially ignoring points that are far away from the point x – and only using ones close by in making its classification decisions.

Can we kernelize other linear classifiers (and regressors)?

- The short answer is yes.
- Lets see another example: suppose we wanted to kernelize logistic regression. We would suppose that:

$$\widehat{\beta} = \sum_{i=1}^{n} \alpha_i x_i,$$

then our conditional probability just becomes:

$$\mathbb{P}(Y = 1|X = x) = \frac{\exp(\beta^T x)}{1 + \exp(\beta^T x)} = \frac{\exp\left(\left(\sum_i \alpha_i x_i\right)^T x\right)}{1 + \exp\left(\left(\sum_i \alpha_i x_i\right)^T x\right)}$$

Replace $x_i^T x$ with $K(x_i, x)$

- So to kernelize it we would simply use:

$$\mathbb{P}(Y = 1|X = x) = \frac{\exp\left(\sum_i \alpha_i K(x_i, x)\right)}{1 + \exp\left(\sum_i \alpha_i K(x_i, x)\right)}$$

and optimize the log-likelihood over the coefficients α. This is called kernel logistic regression.

SVM: a different perspective

There is another important way of thinking about the linear SVM. It turns out that we can re-write the optimization problem (its a bit of work) as solving:

$$\min_{\beta} \sum_{i=1}^{n} (1 - y_i f(x_i))_+ + \frac{\lambda}{2}\|\beta\|_2^2$$

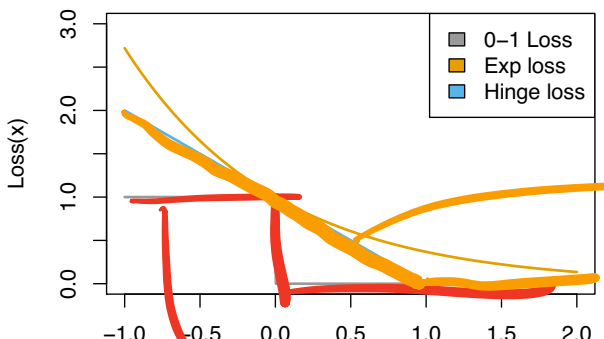
loss

$\beta_0 + \beta^T x_i$

ridge regularizer.

Hinge loss.

This is like replacing 0-1 loss with a hinge function and adding a ridge penalty to keep things regularized!



$\max\{1 - yf(x), 0\}$.

hinge loss, easy to min.

$yf(x)$.

0/1 loss.

Empirical risk minimization

This general pattern:

1. We want to minimize:

$$\mathbb{E}\mathbf{1}\{Y \neq f(X)\}$$

2. And so we actually try to minimize its empirical version:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{y_i \neq f(x_i)\}$$

3. But we can't even do that for classification. So we introduce a nicer loss L and minimize

$$\frac{1}{n}\sum_{i=1}^{n}L(y_i, f(x_i))$$

The first two steps are known as *empirical risk minimization*. The last step almost always follows for classification.

Empirical risk minimization