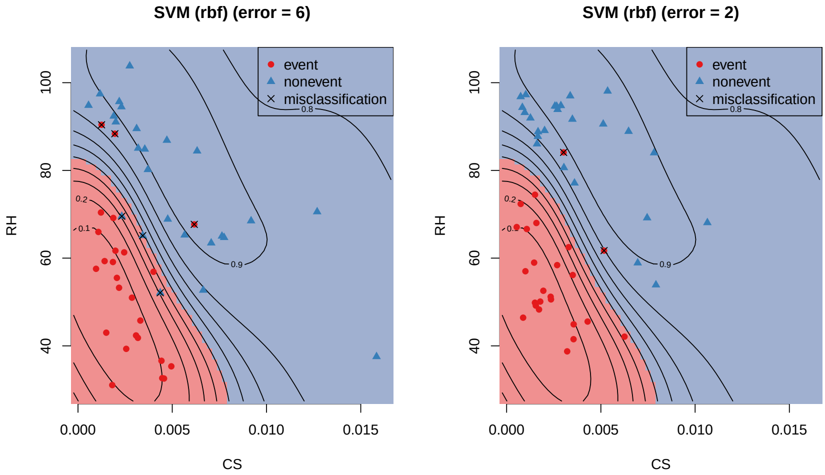
DATA11002 Introduction to Machine Learning

Kai Puolamäki

20 November 2020

Support vector machine



Support vector machines

- A refresher on linear models
- Feature transformations
- Linear classifiers (case: Perceptron)
- Maximum margin classifiers
- Surrogate loss functions (max margin *vs* log. reg.)
- SVM and the kernel trick

Linear models

- A refresher about linear models (see *linear regression*, L3):
- We consider features $x = (x_1, \ldots, x_p)^T \in \mathbb{R}^p$ throughout this lecture
- Function $f \colon \mathbb{R}^p \to \mathbb{R}$ is *linear* if for some $\beta \in \mathbb{R}^p$ it can be written as
$$f(x) = \beta^T x = \sum_{j=1}^{p} \beta_j x_j$$
- By including a constant feature $x_1 = 1$, we can express models with an intercept term using the same formula
- β is often called *coefficient* or *weight vector*

Multivariate linear regression

- We assume matrix $X \in \mathbb{R}^{n \times p}$ has n instances x_i as its rows and $y \in \mathbb{R}^n$ contains the corresponding labels y_i
- In the standard linear regression case, we write

$$y = X\beta + \epsilon$$

  where the *residual* $\epsilon_i = y_i - \beta^T x$ indicates the error of $f(x)$ on data point (x_i, y_i)
- Least squares: Find β which minimises the sum of squared residuals

$$\sum_{i=1}^{n} \epsilon_i^2 = |\epsilon|^2$$

- Closed-form solution (assuming $n \geq p$):

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Further topics in linear regression: Feature transformations

- Earlier (L3), we already discussed non-linear transformations e.g., a degree 5 polynomial of $x \in \mathbb{R}$

$$f(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \beta_5 x_i^5$$

- Likewise, we mentioned the possibility to include *interactions* via *cross-terms*

$$f(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_{12} x_{i1} x_{i2}$$

Further topics in linear regression: Dummy variables

- What if we have qualitative/categorical (instead of continuous) features, like gender, job title, pixel color, etc.?
- Binary features with two *levels* can be included as they are: $x_i \in \{0, 1\}$
- coefficient can be interpreted as the difference between instances with $x_i = 0$ and $x_i = 1$: e.g., average increase in salary
- When there are more than two levels, it doesn't usually make sense to assume linearity

$$f((x_1, x_2, 1)) - f((x_1, x_2, 0)) = f((x_1, x_2, 2)) - f((x_1, x_2, 1))$$

especially when the encoding is arbitrary: red $= 0$, green $= 1$, blue $= 2$

Further topics in linear regression: Dummy variables (2)

- For more than two levels, introduce *dummy* (or *indicator*) variables:

$$x_{i1} = \begin{cases} 1 & \text{if } i\text{th person is a student} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{i2} = \begin{cases} 1 & \text{if } i\text{th person is an actor/actress} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{i3} = \begin{cases} 1 & \text{if } i\text{th person is a data scientist} \\ 0 & \text{otherwise} \end{cases}$$

- One level is usually left without a dummy variable since otherwise the model is *over-parametrized*
  - Adding a constant α to all coefficients of variable X_i and subtracting α from the intercept has net effect zero
- Read Sec. 3.3.1 (Qualitative Predictors) of the textbook

Linear classification via regression

- As we have seen, minimising squared error in linear *regression* has a nice closed form solution (if inverting a $p \times p$ matrix is feasible)
- How about using the linear predictor $f(x) = \beta^T x$ for *classification* with a binary class label $y \in \{-1, 1\}$ through

$$\hat{y} = \mathrm{sign}(f(x)) = \begin{cases} +1 & \text{if } \beta^T x \geq 0 \\ -1 & \text{if } \beta^T x < 0 \end{cases}$$

- Given a training set $(x_1, y_1), \ldots, (x_n, y_n)$, it is computationally intractable to find the coefficient vector w that minimises the 0–1 loss

$$\sum_{i=1}^{n} I[y_i(\beta \cdot x_i) < 0]$$

  - The indicator function $I[\Box] = 1$ if $\Box$ is true and $I[\Box] = 0$ otherwise.

Linear classification via regression (2)

- One approach is to replace 0-1 loss $I[y_i(\beta \cdot x_i) < 0]$ with a **surrogate loss function** - something similar but easier to optimise
- In particular, we could replace $I[y_i(\beta \cdot x_i) < 0]$ by the squared error $(y_i - \beta^T x_i)^2$
  - learn β using least squares regression on the binary classification data set (with $y_i \in \{-1, +1\}$)
  - use β in linear classifier $\hat{c}(x) = \mathrm{sign}(\beta^T x)$
  - **advantage:** computationally efficient
  - **disadvantage:** sensitive to outliers (in particular, "too good" predictions $y_i(\beta^T x) \gg 1$ get heavily punished, which is counterintuitive)
- We'll return to this a while

The Perceptron algorithm (briefly)

NB: The perceptron is just mentioned in passing — not required content. However, the concepts introduced here (**linear separability** and **margin**) will be useful in what follows.

- The *perceptron algorithm* is a simple iterative method which can be used to train a linear classifier
- If the training data $(x_i, y_i)_{i=1}^n$ is *linearly separable*, i.e., there is some $\beta \in \mathbb{R}^p$ such that $y_i(\beta^T x_i) > 0$ for all i, the algorithm is guaranteed to find such a β
- The algorithm (or its variations) can be run also for non-separable data but there is no guarantee about the result

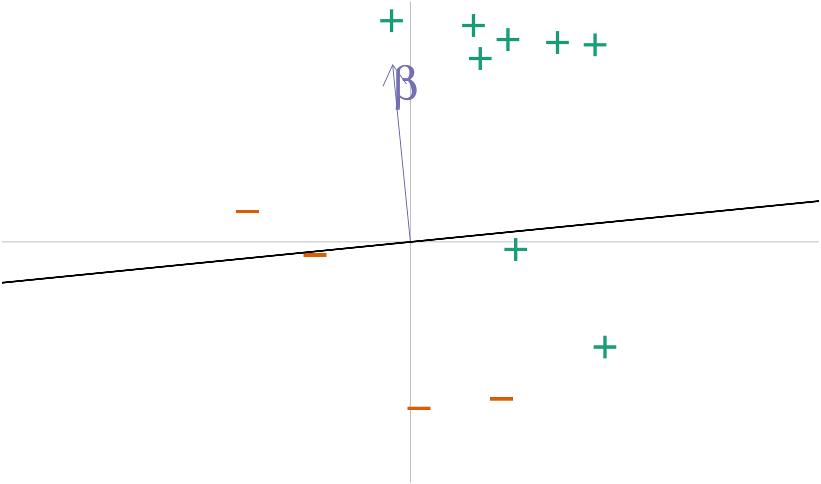
Perceptron algorithm: Main ideas

- The algorithm keeps track of and updates a weight vector β
- Each input item is shown once in a *sweep* over the training data. If a full sweep is completed without any misclassifications then we are done, and return β that classifies all training data correctly.
- Whenever $\hat{y}_i \neq y_i$ we update β by adding $y_i x_i$. This turns β towards x_i if $y_i = +1$, and away from x_i if $y_i = -1$
- NB: Notice that the vector β is normal to the separating hyper-plane.
  - proof: if x and x' are on hyperplane, meaning that $\beta^T x = \beta^T x' = 0$, then vector $v = x - x'$ is tangential to the hyperplane and $\beta^T v = \beta^T (x - x') = \beta^T x - \beta^T x' = 0$, i.e., β is orthogonal to v.

Perceptron algorithm formally (extra material)

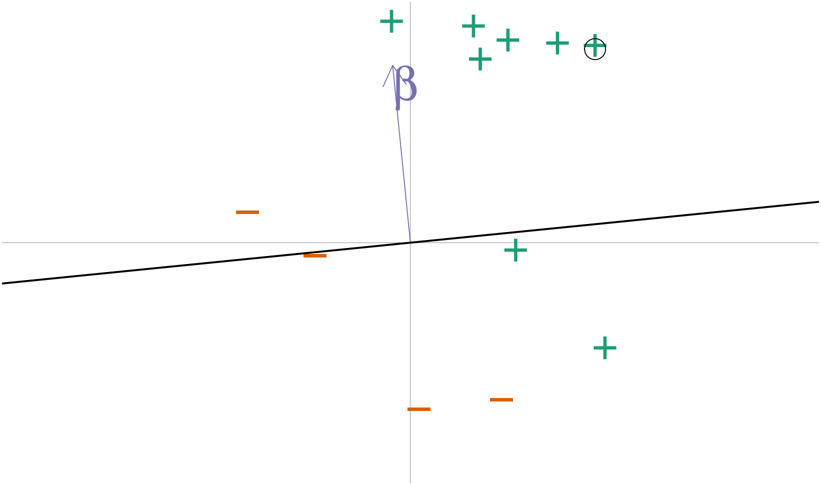
- Classifier $\hat{y}(x) = \mathrm{sign}(\beta^T x) \in \{-1, +1\}$ (use column of 1s for intercept)
- Define $\theta(t) = t$ for $t \geq 0$ and $\theta(t) = 0$ for $t < 0$.
  - $\theta'(t) = H(t) = 1$ for $t \geq 0$ and $H(t) = 0$ for $t < 0$.
- Use loss $L(\beta) = \sum_{i=1}^n \theta(-y_i \beta^T x_i) = -\sum_{i \in M} y_i \beta^T x_i$, where M are the misclassified points (for which $-y_i \beta^T x_i > 0$).
- *Gradient descent:* minimize $L(\beta)$ by iteratively updating $\beta \to \beta - \rho \partial L / \partial \beta = \beta + \rho \sum_{i \in M} y_i x_i$, where the learning rate is $\rho > 0$ and the gradient $\partial L / \partial \beta = -\sum_{i \in M} y_i x_i$.
- *Rosenblatt's perceptron learning algorithm* is *stochastic* gradient descent where at each iteration we pick i from the set of misclassified points M in random and update $\beta \to \beta + \rho y_i x_i$ (we can take $\rho = 1$ here for simplicity).

Perceptron algorithm: illustration



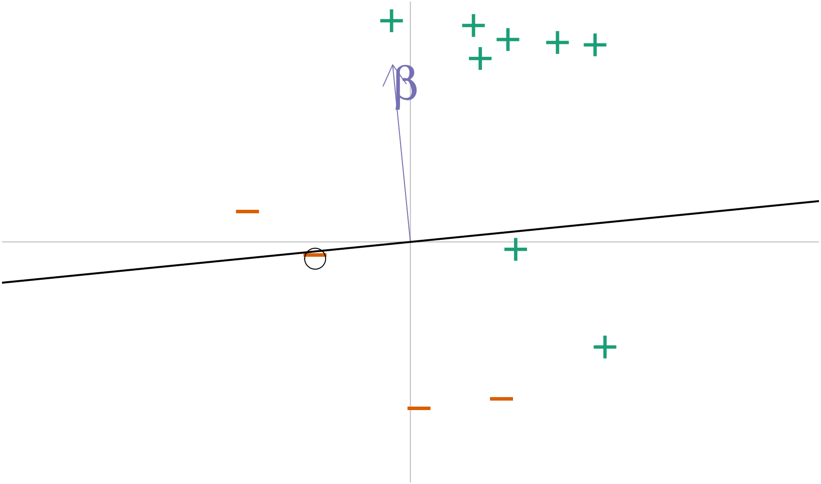
Current state β

Perceptron algorithm: illustration



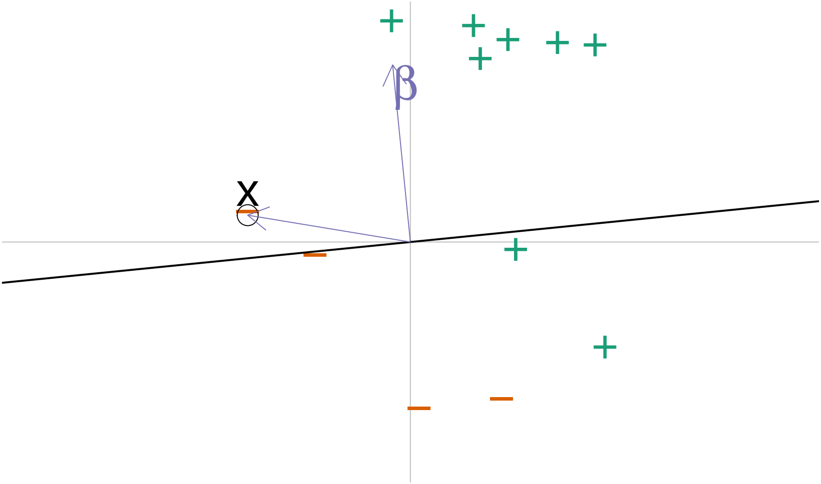
"+" classified correctly, no change to β

Perceptron algorithm: illustration



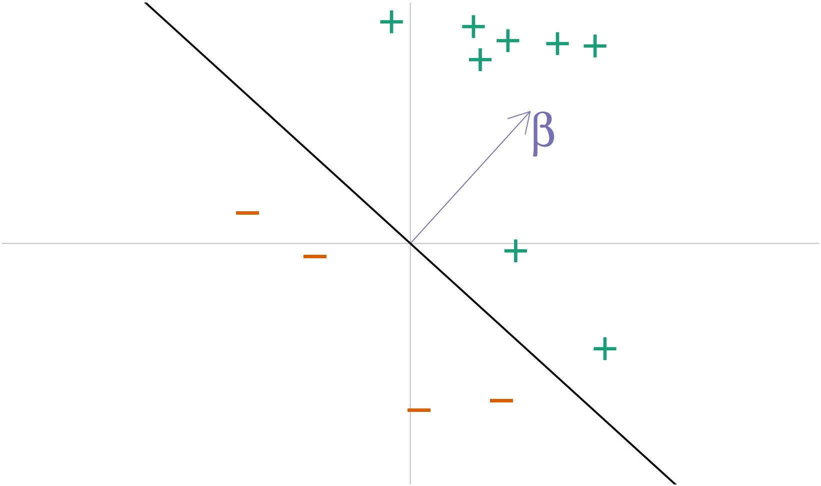
“-” classified correctly, no change to β

Perceptron algorithm: illustration



"-" classified incorrectly, will change $\beta \leftarrow \beta + yx$ (here $y = -1$)

Perceptron algorithm: illustration



Everything is classified correctly and we have converged. Notice that the *lenght* of β is irrelevant for classification.

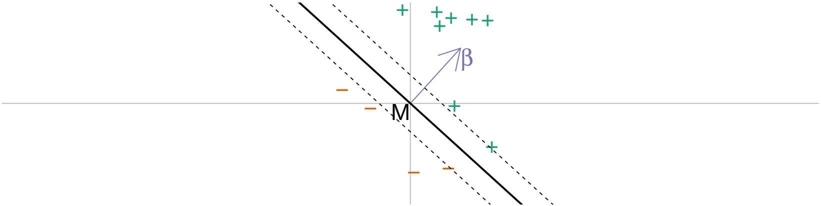
Perceptron algorithm: Comments

- Upsides:
  - Can be used for **online learning**, i.e., the case where data arrives in an endless stream. Very useful if memory is a problem.
  - For linearly separable data is guaranteed to find a separating hyperplane.
- Downsides:
  - Is sensitive to initialisation of β.
  - May converge very slowly.
  - When data are not (linearly) separable, may not even converge.

For separable data the perceptron finds a separating hyperplane, but there are several (infinitely many!) of such hyperplanes! How to choose the best one?

Margin

- Given a data set $\{(x_i, y_i)\}_{i=1}^n$ and $M > 0$, we say that a coefficient vector β separates the data with *margin* M if for all i we have $y_i(\beta^T x_i)/|\beta| \geq M$
- Explanation
  - $\beta^T x_i/|\beta|$ is the **scalar projection** of x_i onto vector β
  - $y_i(\beta^T x_i) \geq 0$ means we predict the correct class
  - $|\beta^T x_i|/|\beta|$ is Euclidean distance between point x_i and the decision boundary, i.e., hyperplane $\beta^T x = 0$



Margin formally (extra material)

- Optimization problem: find β that maximizes the margin M subject to $y_i \beta^T x_i / |\beta| \geq M$ for all i, where $|\beta| = \sqrt{\beta^T \beta}$ and $M \in \mathbb{R}_+$.
- If β is a solution then also $t\beta$ is a solution for any $t \in \mathbb{R}_+$. We can therefore choose the norm freely. Choose $|\beta| = 1/M$.
- Equivalent optimization problem: find β that minimizes $|\beta|^2/2$ subject to $y_i x^T \beta \geq 1$ for all i.
- This is a convex optimization problem (quadratic criterion with linear inequality constraints), for which efficient solvers exists.
  - "Given $\mathbf{D} \in \mathbb{R}^{n \times n}$, $d \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n \times k}$, and $b_0 \in \mathbb{R}^k$, find $b \in \mathbb{R}^n$ that minimizes $b^T \mathbf{D} b/2 - d^T b$ subject to $\mathbf{A}^T b \geq b_0$."
  - See, e.g., R package "quadprog".

Margin formally (extra material)

- Lagrange (primal function) to be minized is $L_P = |\beta|^2/2 - \sum_{i=1}^n \alpha_i(y_i\beta^T x_i - 1)$ subject to $\alpha_i(y_i\beta^T x_i - 1) = 0$, where $\alpha_i \geq 0$ are KKT multipliers.
- Solve $\partial L_P/\partial\beta = 0$, resulting $\beta = \sum_{i=1}^n \alpha_i y_i x_i$.
  - Estimate $\hat{y}(x) = \mathrm{sign}(\beta^T x) = \sum_{i=1}^n \alpha_i y_i \mathbf{x_i^T x}$.
- Inserting β into L_P we obtain the so-called Wolfe dual: find α_i that maximizes $L_D = \sum_{i=1}^n \alpha_i - \sum_{i=1}^n \sum_{j=1}^n \alpha_i\alpha_j y_i y_j \mathbf{x_i^T x_j}/2$ subject to $\alpha_i \geq 0$.
- Observations:
  - Data point x_i is either on margin $(y_i\beta^T x_i - 1 = 0)$ or $\alpha_i = 0$.
  - We call a data point x_i *support vector*, if $\alpha_i > 0$.
  - β is a linear combination of support vectors, $\beta = \sum_{i=1}^n \alpha_i y_i x_i$.
  - The Wolfe dual L_D and the estimate $\hat{y}(x) = \sum_{i=1}^n \alpha_i y_i \mathbf{x_i^T x}$ can be expressed in terms of dot products of $x^T x'$ only!

See https://en.wikipedia.org/wiki/Duality_(optimization)

Observations on max margin classifiers

- Consider the linearly separable case (i.e., $\epsilon_i = 0$ in the next slides).
- The maximal margin touches a set of training data points x_i, which are called **support vectors**

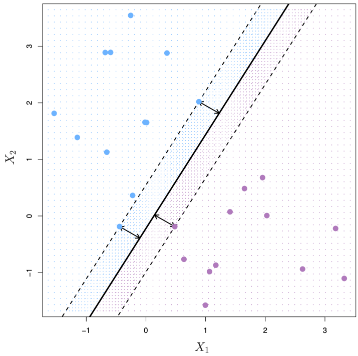


Fig. 9.3 from James et al.

Max margin classifier and SVM: Terminology

- **Maximal margin classifier** (Sec. 9.1.3): Find β that classifies all instances correctly and maximizes the margin M.
- **Support vector classifier** (Sec. 9.2): Maximize the **soft margin** M allowing some points to violate the margin (and even be misclassified), controlled by a tuning parameter C:

  $$\max_{\beta} M$$

  subject to

  $$y_i(\beta^T x_i)/|\beta| \geq M(1 - \epsilon_i),$$

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

- **Support vector machine** (SVM; Sec. 9.3): Non-linear version of the support vector classifier.

Effect of parameter C

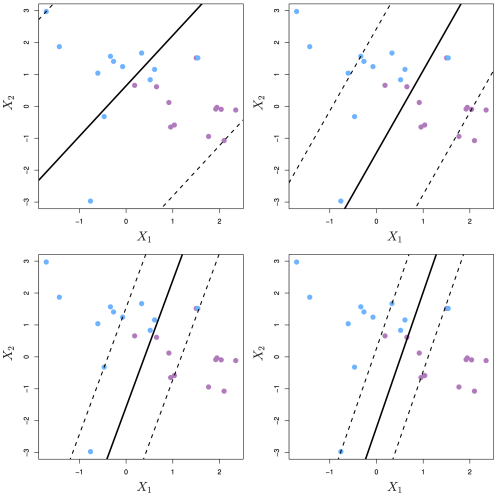


Fig. 9.7 from James et al.

Observations on max margin classifiers (2)

- Given a set of support vectors, the coefficients defining the hyperplane can be defined as

$$\hat{\beta} = \sum\nolimits_{i=1}^{n} \alpha_i y_i x_i,$$

 with some $\alpha_i \geq 0$, where $\alpha_i > 0$ only if the ith data point touches the margin
- In other words, **the classifier is defined by a few data points!**
- A similar property holds for the soft margin: the more the ith point violates the margin, the larger α_i, and forpoints that do not violate the margin, $\alpha_i = 0$

Observations on max margin classifiers (3)

- The optimization problem for both hard and soft margin can be solved efficiently using the *Lagrange method*
- The details are beyond our scope (but interesting!)
- A key property is that the solution only depends on the data through the inner products $\langle x_i, x_j \rangle = x_i^T x_j$ (and the values y_i)
- This follows from the expression of the coefficient vector $\hat{\beta}$ as a linear combination of the support vectors.
- Given a new (test) data point x, we can classify it using

$$\hat{y}(x) = \mathrm{sign}(\beta^T x) = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle$$

Relation to other linear classifiers

- The soft margin minimization problem of the support vector classifier can be rewritten as an unconstrained problem

$$\min_\beta \left\{ \sum_{i=1}^n \max(0, 1 - y_i(\beta^T x_i)) + \lambda |\beta|_2^2 \right\}$$

- Compare this to penalized logistic regression

$$\min_\beta \left\{ \sum_{i=1}^n \log(1 + \exp(-y_i(\beta^T x_i))) + \lambda |\beta|_2^2 \right\}$$

- or ridge regression

$$\min_\beta \left\{ \sum_{i=1}^n (y_i - \beta^T x_i)^2 + \lambda |\beta|_2^2 \right\}$$

- These are all examples of common *surrogate loss functions*

Relation to other linear classifiers (2)

- Compare the **hinge loss** ("SVM Loss") $\max(0, 1 - y_i(\beta^T x))$ (black) and the logistic loss $\log(1 + \exp(-y_i(\beta^T x_i)))$ (green)

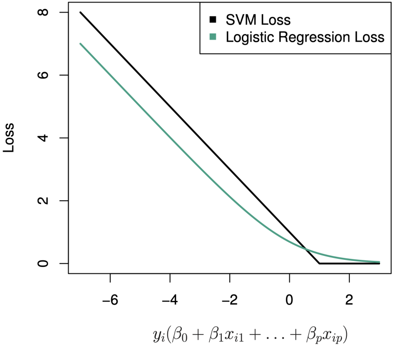


Fig. 9.12 from James et al.

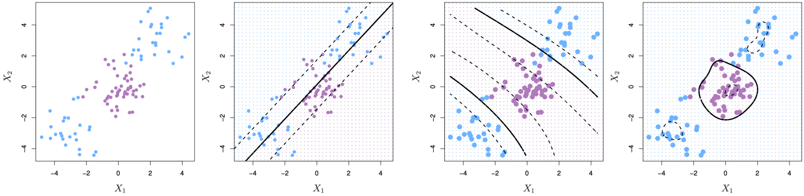
Kernel trick

- Since the data only appear through $\langle x_i, x_j \rangle$, we can use the following **kernel trick**
- Imagine that we want to introduce non-linearity by mapping the original data into a higher-dimensional representation
  - remember the polynomial example $x_i \mapsto 1, x_i, x_i^2, x_i^3, \ldots$
  - interaction terms are an another example: $(x_i, x_j) \mapsto (x_i, x_j, x_i x_j)$
- Denote this mapping by $\Phi : \mathbb{R}^p \to \mathbb{R}^q$, $q > p$
- Define the kernel function as $K(x_i, x) = \langle \Phi(x_i), \Phi(x) \rangle$
- The trick is to evaluate $K(x_i, x)$ without actually computing the mappings $\Phi(x_i)$ and $\Phi(x)$

Kernels

- Popular kernels:
  - linear kernel: $K(x_i, x) = \langle x_i, x \rangle$
  - polynomial kernel: $K(x_i, x) = (\langle x_i, x \rangle + 1)^d$
  - (Gaussian) radial basis function: $K(x_i, x) = \exp(-\gamma |x_i - x|_2^2)$
- For example, the radial basis function (RBF) kernel corresponds to a feature mapping of infinite dimension!
- The same kernel trick can be applied to any learning algorithm that can be expressed in terms of inner products between the data points x
  - perceptron
  - linear (ridge) regression
  - Gaussian process regression
  - principal component analysis (PCA)
  - ...

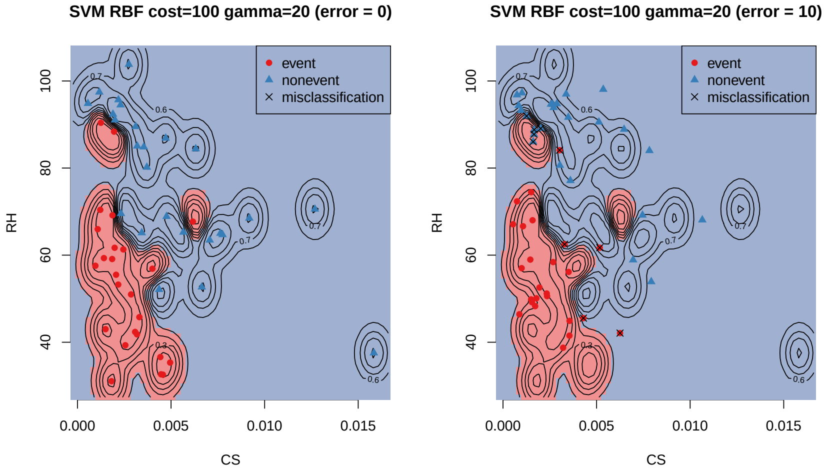
SVM: Example



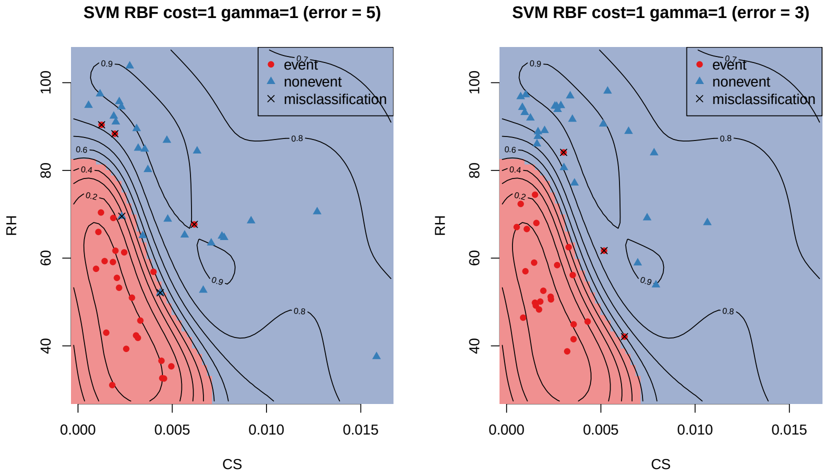
From left to right: original data, linear kernel, polynomial kernel ($d = 3$), radial basis function (Figs. 9.8-9 from James et al.)

```
library(e1071)
fit <- svm(class2 ~ .,data=data,kernel="radial",gamma=1,cost=1)
plot(fit,data)
```

Support vector machine



Support vector machine



SVMs: Properties

- The use of the hinge loss (soft margin) as a surrogate for the 0-1 loss leads to the support vector classifier
- With a suitable choice of kernel, the SVM can be applied in various different situations
  - string kernels for text, structured outputs, ...
- The computation of pairwise kernel values $K(x_i, x_j)$ may become intractable for large samples but fast techniques are available
- SVM is one of the overall best out-of-the-box classifiers (together with random forest)
- Since the kernel trick allows complex, non-linear decision boundaries, regularization is absolutely crucial
  - the tuning parameter C is typically chosen by cross-validation

What is the best classifier?

- Fernández-Delgado et al. (2014) Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?. JMLR.
- Hand (2006) Classifier Technology and the Illusion of Progress. Statistical Science.