582631 — 5 credits

Introduction to Machine Learning

Lecturer: Teemu Roos
Assistant: Ville Hyvönen

Department of Computer Science
University of Helsinki

(based in part on material by Patrik Hoyer and Jyrki Kivinen)

November 1st–December 16th 2016

Support Vector Machines

Outline

- A refresher on linear models
- Feature transformations
- Linear classifiers:
  - surrogate loss functions
  - case Perceptron
- Maximum margin classifiers
- SVM and the kernel trick

Linear models

- A refresher about linear models (see *linear regression*, Lecture 3):

- We consider features $\mathbf{x} = (x_1, \ldots, x_p) \in \mathbb{R}^p$ throughout this chapter

- Function $f : \mathbb{R}^p \to \mathbb{R}$ is *linear* if for some $\boldsymbol{\beta} \in \mathbb{R}^p$ it can be written as

$$f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x} = \sum_{j=1}^{p} \beta_j x_j$$

- By including a constant feature $x_1 \equiv 1$, we can express models with an intercept term using the same formula

- $\boldsymbol{\beta}$ is often called *coefficient* or *weight vector*

Multivariate linear regression

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

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

  where the *residual* $\epsilon_i = y_i - \boldsymbol{\beta} \cdot \mathbf{x}_i$ indicates the error of $f(\mathbf{x})$ on data point $(\mathbf{x}_i, y_i)$

- Least squares: Find $\boldsymbol{\beta}$ which minimises the sum of squared residuals

$$\sum_{i=1}^{n} \epsilon_i^2 = \|\boldsymbol{\epsilon}\|_2^2$$

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

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

Further topics in linear regression: Feature transformations

- Earlier (Lecture 3), we already discussed non-linear transformations:
  e.g., a degree 5 polynomial of $x \in \mathbb{R}$

$$f(x_i) = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 x_i + \boldsymbol{\beta}_2 x_i^2 + \boldsymbol{\beta}_3 x_i^3 + \boldsymbol{\beta}_4 x_i^4 + \boldsymbol{\beta}_5 x_i^5$$

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

$$f(\mathbf{x}_i) = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 x_{i1} + \boldsymbol{\beta}_2 x_{i2} + \boldsymbol{\beta}_{12} x_{i1} x_{i2}$$

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:

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

$$x_{i2} = \begin{cases} 1 & \text{if } i\text{th person is a physician} \\ 0 & \text{otherwise} \end{cases}$$

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

- 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 $f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x}$ for *classification* with a binary class label $y \in \{-1, 1\}$ through

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

- Given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, it is computationally intractable to find the coefficient vector $\boldsymbol{\beta}$ that minimises the 0–1 loss

$$\sum_{i=1}^{n} I_{[y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i) < 0]}$$

Linear classification via regression (2)

- One approach is to replace 0-1 loss $I_{[y_i(\boldsymbol{\beta}\cdot\mathbf{x}_i)<0]}$ with a **surrogate loss function** — something similar but easier to optimise

- In particular, we could replace $I_{[y_i(\boldsymbol{\beta}\cdot\mathbf{x}_i)<0]}$ by the squared error $(y_i - \boldsymbol{\beta}\cdot\mathbf{x}_i)^2$
  - learn $\boldsymbol{\beta}$ using least squares regression on the binary classification data set (with $y_i \in \{-1, +1\}$)
  - use $\boldsymbol{\beta}$ in linear classifier $\hat{c}(\mathbf{x}) = \mathrm{sign}(\boldsymbol{\beta}\cdot\mathbf{x})$
  - **advantage:** computationally efficient
  - **disadvantage:** sensitive to outliers (in particular, "too good" predictions $y_i(\boldsymbol{\beta}\cdot\mathbf{x}_i) \gg 1$ get heavily punished, which is counterintuitive)

- We'll return to this a while

The Perceptron algorithm (briefly)

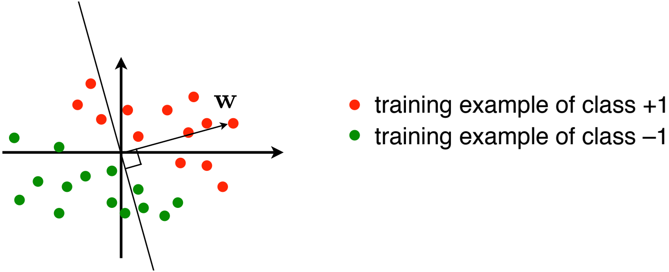
NB: The perceptron is just mentioned in passing — not required for the exam. However, the concepts introduced here (**linear separability** and **margin**) will be useful in what follows.

- The *perceptron algorithm* is a simple iterative method which can be used to train a linear classifier
- If the training data $(\mathbf{x}_i, y_i)_{i=1}^n$ is *linearly separable*, i.e. there is some $\boldsymbol{\beta} \in \mathbb{R}^p$ such that $y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i) > 0$ for all i, the algorithm is guaranteed to find such a $\boldsymbol{\beta}$
- 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

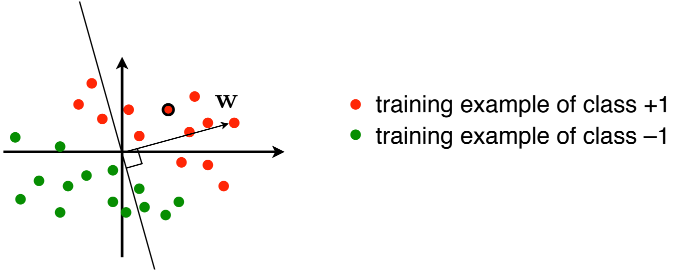
- The algorithm keeps track of and updates a weight vector $\boldsymbol{\beta}$
- Each input item is shown once in a *sweep* over the training data. If a full sweep is completed without any misclassifications then we are done, and return $\boldsymbol{\beta}$ that classifies all training data correctly.
- Whenever $\hat{y}_i \neq y_i$ we update $\boldsymbol{\beta}$ by adding $y_i \mathbf{x}_i$. This turns $\boldsymbol{\beta}$ towards $\mathbf{x}_i$ if $y_i = +1$, and away from $\mathbf{x}_i$ if $y_i = -1$

Perceptron algorithm: Illustration



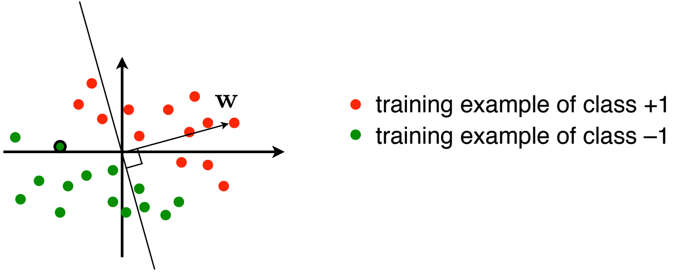
Current state of β (denoted by **w** in the figure)

Perceptron algorithm: Illustration



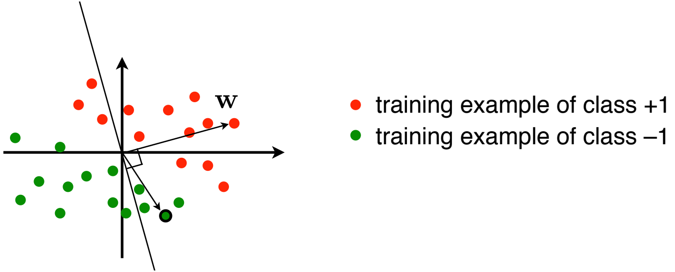
Red point classified correctly, no change to $\boldsymbol{\beta}$

Perceptron algorithm: Illustration



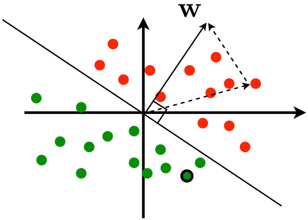
Green point classified correctly, no change to $\boldsymbol{\beta}$

Perceptron algorithm: Illustration



Green point misclassified, will change β as follows...

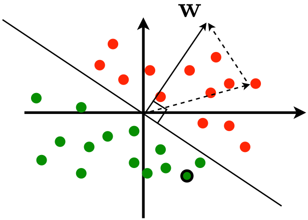
Perceptron algorithm: Illustration



- training example of class +1
- training example of class –1

Adding $y_i \mathbf{x}_i$ to current weight vector $\boldsymbol{\beta}$ to obtain new weight vector

Perceptron algorithm: Illustration



- training example of class +1
- training example of class –1

Adding $y_i\mathbf{x}_i$ to current weight vector $\boldsymbol{\beta}$ to obtain new weight vector

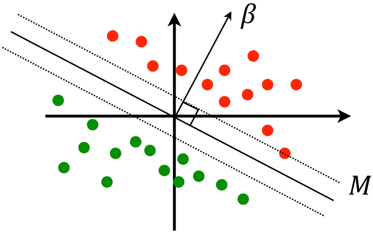
Note that the *length* of $\boldsymbol{\beta}$ is irrelevant for classification

Margin

- Given a data set $(\mathbf{x}_i, y_i)_{i=1}^n$ and $\gamma > 0$, we say that a coefficient vector $\boldsymbol{\beta}$ separates the data with *margin* M if for all i we have

$$\frac{y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i)}{\|\boldsymbol{\beta}\|_2} \geq M$$

- Explanation
  - $y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i) \geq 0$ means we predict the correct class
  - $|\boldsymbol{\beta} \cdot \mathbf{x}_i| \,/\, \|\boldsymbol{\beta}\|_2$ is Euclidean distance between point $\mathbf{x}_i$ and hyperplane $\boldsymbol{\beta} \cdot \mathbf{x} = 0$



Max margin classifier and SVM: Terminology

- **Maximal margin classifier** (Sec. 9.1.3): Find $\boldsymbol{\beta}$ that classifies all instances correctly and maximizes the margin M

Its special cases:

- **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:

$$y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i) \geq M(1 - \epsilon_i)$$

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

$$\text{subject to } \|\boldsymbol{\beta}\|_2 = 1$$

- **Support vector machine** (SVM; Sec. 9.3): Non-linear version of the support vector classifier obtained by defining a **kernel function** $K(\mathbf{x}_i, \mathbf{x}_j)$

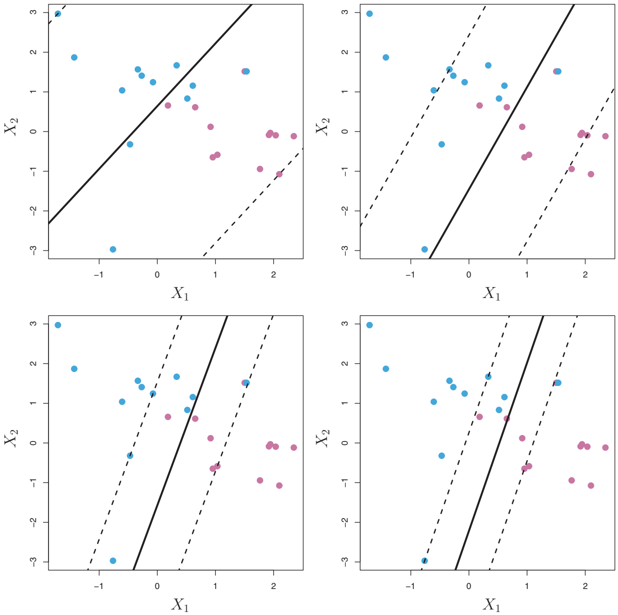


Fig. 9.7 in (James et al., 2013)

Observations on max margin classifiers

- Consider the linearly separable case $\epsilon_i \equiv 0$.
- The maximal margin touches a set of training data points $\mathbf{x}_i$, which are called **support vectors**

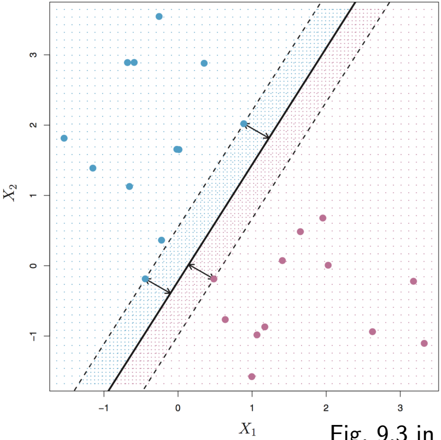


Fig. 9.3 in (James et al., 2013)

Observations on max margin classifiers (2)

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

$$\hat{\boldsymbol{\beta}} = \sum_{i=1}^{n} c_i y_i \mathbf{x}_i,$$

with some $c_i \geq 0$, where $c_i > 0$ 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 c_i, and for points that do not violate the margin, $c_i = 0$

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 $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i \cdot \mathbf{x}_j$ (and the values y_i)
- This follows from the expression of the coefficient vector $\hat{\boldsymbol{\beta}}$ as a linear combination of the support vectors.
- Given a new (test) data point $\mathbf{x}$, we can classify it based on the sign of

$$\hat{f}(\mathbf{x}) = \hat{\boldsymbol{\beta}} \cdot \mathbf{x} = \left(\sum_{i=1}^{n} c_i y_i \mathbf{x}_i \right) \cdot \mathbf{x} = \sum_{i=1}^{n} c_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle$$

Relation to other linear classifiers

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

$$\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i)] + \lambda \|\boldsymbol{\beta}\|_2^2 \right\}$$

- Compare this to penalized logistic regression

$$\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} \ln(1 + \exp(-y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i))) + \lambda \|\boldsymbol{\beta}\|_2^2 \right\}$$

- or ridge regression

$$\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} (y_i - \boldsymbol{\beta} \cdot \mathbf{x}_i)^2 + \lambda \|\boldsymbol{\beta}\|_2^2 \right\}$$

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

Relation to other linear classifiers (2)

- Compare the **hinge loss** $\max[0, 1 - y_i(\boldsymbol{\beta} \cdot \mathbf{x})]$ (black) and the logistic loss $\exp(-y_i(\boldsymbol{\beta} \cdot \mathbf{x}))$ (green)

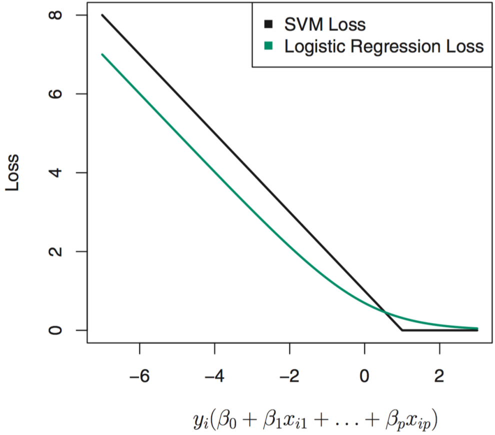


Fig. 9.12 in the (James et al., 2013)

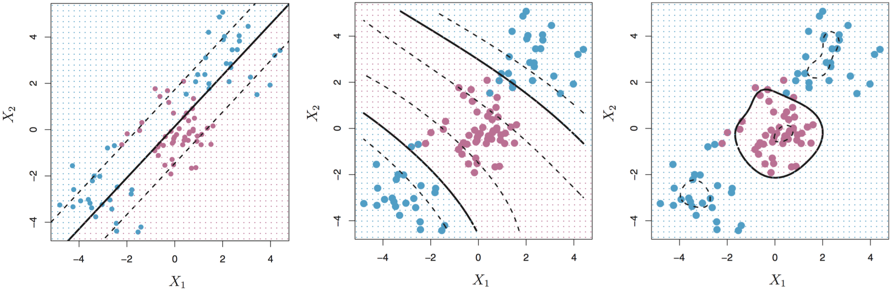
Kernel trick

- Since the data only appear through $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$, 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 $K(\mathbf{x}_i, \mathbf{x}) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle$
- The trick is to evaluate $K(\mathbf{x}_i, \mathbf{x})$ without actually computing the mappings $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x})$

Kernels

- Popular kernels:
  - linear kernel: $K(\mathbf{x}_i, \mathbf{x}) = \langle \mathbf{x}_i, \mathbf{x} \rangle$
  - polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}) = (\langle \mathbf{x}_i, \mathbf{x} \rangle + 1)^d$
  - (Gaussian) radial basis function: $K(\mathbf{x}_i, \mathbf{x}) = \exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x} \right\|_2^2)$
- 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 $\mathbf{x}$
  - perceptron
  - linear (ridge) regression
  - Gaussian process regression
  - principal component analysis (PCA)
  - ...

SVM: Example



From (James et al., 2013)

- Three SVM results on the same data, from left to right: Linear kernel, polynomial kernel $d = 3$, RBF

```
library(e1071)
fit = svm(y∼., data=D, kernel="radial", gamma=1, cost=1)
plot(fit, D)
```

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 $K(\mathbf{x}_i, \mathbf{x}_j)$ may become intractable for large samples but fast techniques are available
- SVM is one of the overall best out-of-the-box classifiers
- Since the kernel trick allows complex, non-linear decision boundaries, regularization is absolutely crucial:
  - the tuning parameter C is typically chosen by cross-validation