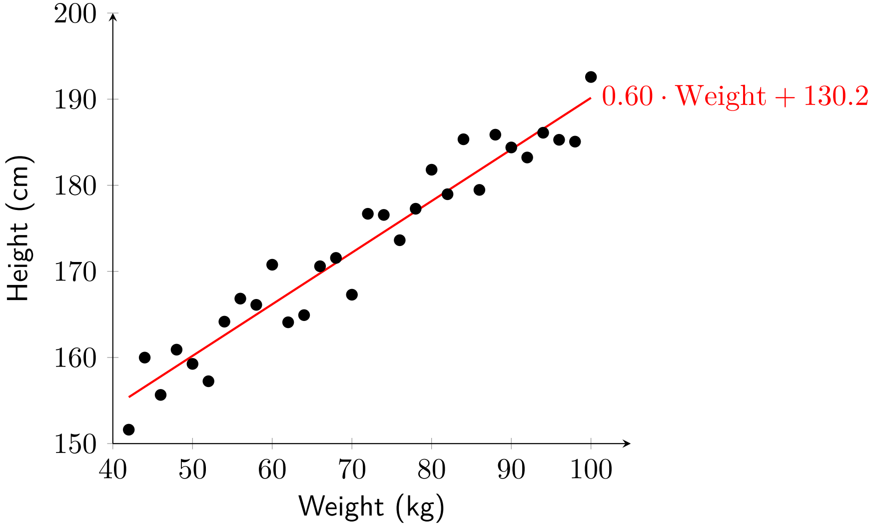
Linear regression



Linear Regression $Height = g(Weight)$ finds a trend in data.

Linear regression classifiers

- Two-class classification - domain linear partition into the positive and negative regions by comparing against the threshold $g(a) = \langle \mathbf{wa} \rangle = \langle w^1_{n+1} a^1_{n+1} \rangle$, where $a_{n+1} = 1$ for the intercept term $b = w_0 = w_{n+1}$ from $g(a) = w^1_n a^1_n + w_0$

$$h(a) = H(g(a)) = \begin{cases} 1 & \text{if } g(a) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Class probability modeling - the logit representation $logit(p) = \ln \frac{p}{1-p}$, especially its inverse function called the logistic function

$$P(1|a) = logit^{-1}(g(a)) = \frac{e^{g(a)}}{e^{g(a)} + 1}$$

Optimization problem with constraints

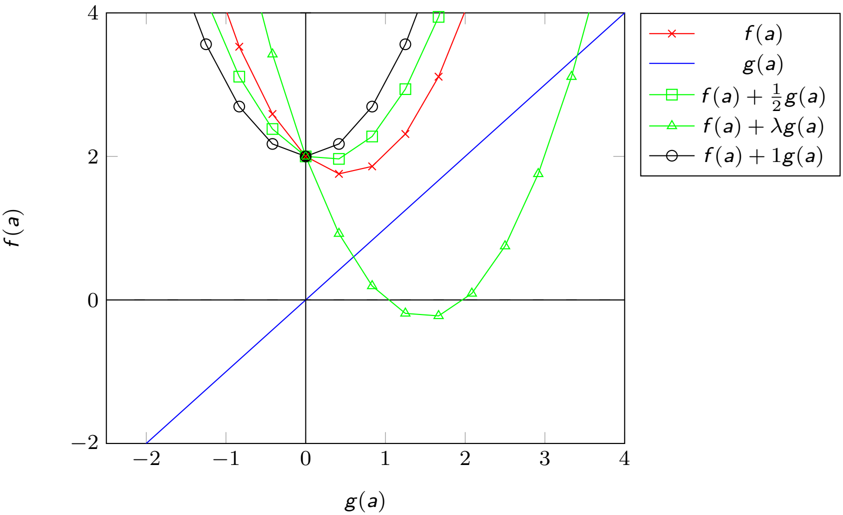
- Minimize f(a) as a primal optimization problem

$$\begin{array}{ll} \min\limits_{a} & f(a) \\ \text{with constraints:} & g_i(a) = 0, \ i = 1, \ldots, m. \end{array}$$

- Lagrangian is made of f(a) and constraints: $L(a, \lambda) = f(a) + \lambda g(a)$
- Minimize $L(a, \lambda)$ in a domain a in a dual optimization problem.
- After minimization with derivatives of Lagrangian $L(a, \lambda)$ equal to 0, maximize $L(a, \lambda)$ in a domain *lambda*.

$$\min_{a} \max_{\lambda} \quad f(a) + \lambda g(a)$$

Optimization primal and dual problem example



- $\lambda = 1$ for Lagrangian $\max_{\lambda > 0} \min_a f(a) + \lambda g(a)$ dual optimal solution.
- Result is the same for primal and dual problem solution: (0,2)

VC (Vapnik–Chervonenkis) dimension of hyperplanes in $\mathbb{R}^n$

- For 4 points in 2D (a square or a matrix 2x2) and two classes:
  - it is not always possible to separate them with lines.

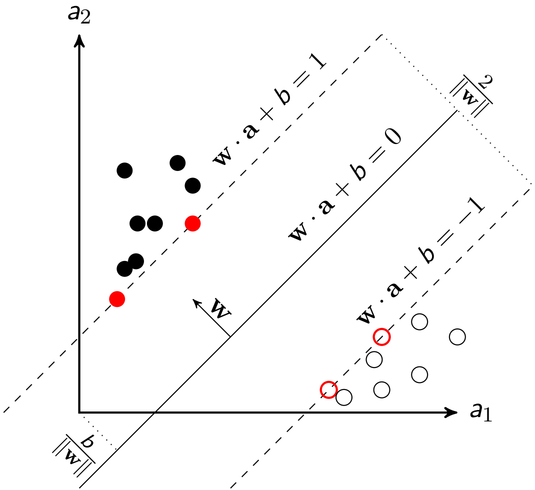
$$\begin{array}{cc} c_1 & c_2 \\ c_2 & c_1 \end{array} \quad ==> \quad \begin{array}{c|c} c_1 & c_2 \\ c_2 & c_1 \end{array} \quad or \quad \begin{array}{cc} c_1 & c_2 \\ \hline c_2 & c_1 \end{array}$$

- $n+1$ points in $\mathbb{R}^n$ can not be separated linearly
- The VC dimension of hyperplanes in $\mathbb{R}^n$ is $n+1$

Support Vector Machines: basic linear examples

- Linear separation for two classes
- Some training data $\{\mathbf{a}_i, c_i\}_m^i$, $\mathbf{a}_i \in \mathbb{R}^n$, and $c_i \in \{-1, 1\}$
- Train to obtain the separation hyperplane:
  - Minimize $d_+ + d_-$ to receive the shortest distances from the hyperplane
    - to closest positive point d_+
    - to closest negative point d_-
- The goal is to find the separating hyperplane: $\mathbf{wa} + b = 0$, where
  - a vector w is normal to the hyperplane
  - $\frac{|b|}{||\mathbf{w}||}$ is the distance to origin $(0, 0)$
  - $||\mathbf{w}||$ the length of the vector $\mathbf{w}$
- Designing a road between trees on the left and rocks on the right.

Support Vector Machines



- max $\frac{2}{\|w\|}$ - maximize the margin between parallel lines

Support Vector Machines

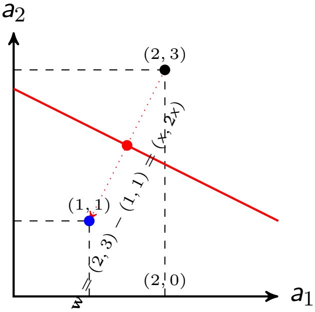
- d_+, d_- the shortest distances from labeled points to hyperplane
- A margin $m = d_+ + d_-$
- The optimal separating hyperplane maximizes m and minimizes the VC dimension
- For the separating hyperplane:

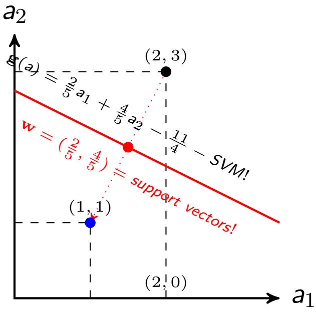
$$\mathbf{wa}_i + b \geq +1, \quad c_i = +1 \tag{1}$$
$$\mathbf{wa}_i + b \leq -1, \quad c_i = -1 \tag{2}$$
$$\equiv \tag{3}$$
$$c_i(\mathbf{wa}_i + b) - 1 \geq 0, \qquad \forall i \tag{4}$$

- For the closest points the equalities are satisfied, so:

$$d_+ + d_- = \frac{|1-b|}{\|w\|} + \frac{|-1-b|}{\|w\|} = \frac{2}{\|w\|} \tag{5}$$

Support Vector Machines: the linear example





$$\mathbf{w} = (2,3) - (1,1) = (x, 2x) \tag{1}$$
$$\textit{for point } (2,3) : \mathbf{wa} + b = +1 \tag{2}$$
$$2x + 6x + b = 1 \tag{3}$$
$$b = 1 - 8x \tag{4}$$
$$\textit{for point } (1,1) : \mathbf{wa} + b = -1 \tag{5}$$
$$x + 2x + b = -1 \tag{6}$$
$$x + 2x + 1 - 8x = -1 \tag{7}$$
$$x = \frac{2}{5} \quad b = -\frac{11}{5} \tag{8}$$

$$\underline{\textit{Support vectors}} : w = (\frac{2}{5}, \frac{4}{5}) \tag{9}$$
$$\underline{\textit{SVM}} : g(a) = \mathbf{wa} + b \tag{10}$$
$$\underline{\textit{SVM}} : g(a) = \frac{2}{5}a_1 + \frac{4}{5}a_2 - \frac{11}{4} \tag{11}$$
$$\forall c(a) \in c_1 \; g(a) >= 1 \tag{12}$$
$$\forall c(a) \in c_2 \; g(a) <= -1 \tag{13}$$

Support Vector Machines: Lagrangian

- The constraints is more easy to handle.
- Using dual form SVM models predictions and calculations can be performed without using any attribute values other than inside dot products.
- Instead of calculating model parameters Lagrange multipliers are used.
- Prepared for the kernel trick to swap the dot product with the kernel special function.

Support Vector Machines: Lagrangian

- For $\{\mathbf{a}_i, c_i\} \in \mathbb{R}^N \times \{-1, 1\}$ a primal problem minimization:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{Subject to} \quad & c_i\left(\langle \mathbf{w}, \mathbf{a}\rangle + b\right) \geq 1. \end{aligned} \tag{1}$$

- An unconstrained problem with the Lagrange multipliers for minimization $\min_{\mathbf{w},b} L(\mathbf{w}, b, \lambda)$

$$L(\mathbf{w}, b, \lambda) = \frac{1}{2}||\mathbf{w}||^2 - \sum_i \lambda_i \left(c_i(\langle \mathbf{w}, \mathbf{a}\rangle + b) - 1\right), \tag{2}$$

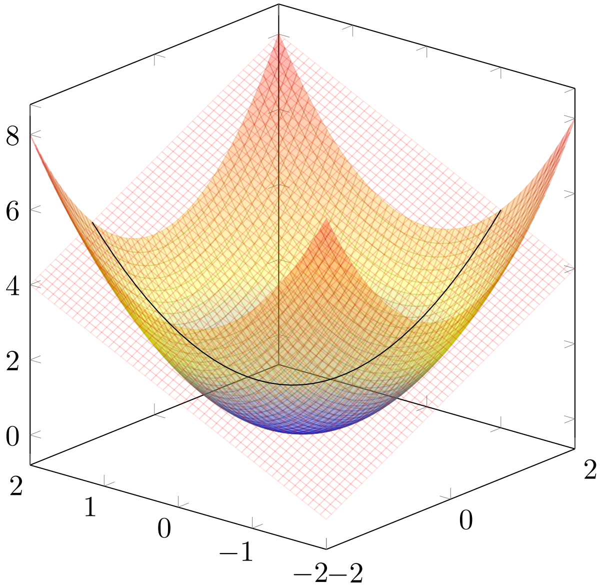
- Convex quadratic programming dual problem $\max_{\lambda > 0} L_D$

$$L_D = \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{l} \lambda_i \lambda_j c_i c_j \langle \mathbf{a}_i \mathbf{a}_j \rangle \tag{3}$$

$$C \geq \lambda_i \geq 0 \tag{4}$$

$$\sum_i \lambda_i c_i = 0 \tag{5}$$

The dot product surface $x^2 + y^2$ cut by a plane $x + y + 4$



Support Vector Machines: Lagrangian

- From dual problem

$$\max_{\lambda>0} \min_{\mathbf{w},b} \qquad L(\mathbf{w}, b, \lambda)$$

- to a quadratic optimization problem

$$\max_{\lambda>0} \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \alpha^t K \alpha$$

$$K_{ij} = \langle \mathbf{a}_i \mathbf{a}_j \rangle$$

$$\alpha_i = \lambda_i c_i$$

- Solution only depends on support vectors $\mathbf{w}_{\lambda>0}$
- All others have $\lambda_i = 0$ and can be moved arbitrarily far from the decision hyperplane or removed
- Such the dual problem is solved with the help of a gradient descent, which is appropriate for dot products.
- The kernel trick can be applied and K can be replaced by an inner product $\phi(x_i).\phi(x_j)$ in a higher dimensional space.

Support Vector Machines: space transformation

- Take points from $\mathbb{R}^d$ to some space $\mathcal{H}$:

$$\Phi : \mathbb{R}^d \to \mathcal{H} \tag{1}$$

- Choose kernel function K such that

$$K(a_i, a_j) = \Phi(\mathbf{a}_i)\Phi(\mathbf{a}_j) \tag{2}$$

- Since in the Lagrangian formula we only have a_i in dot products, we don't even need to know Φ!

Support Vector Machines: kernel

- Replacing $\langle \mathbf{a}_i \mathbf{a}_j \rangle$ with $K(\mathbf{a}_i, \mathbf{a}_j)$ everywhere do all the magic.
- Training is identical and takes almost similar time.
- Separation is still linear, but in a different space (infinite-dimensional!)
- Kernel examples:
  - Gaussian Kernel

$$K(\mathbf{a}_i, \mathbf{a}_j) = e^{\frac{-\|\mathbf{a}_i - \mathbf{a}_j\|^2}{2\sigma^2}} \tag{1}$$

  - Polynomial

$$K(\mathbf{a}_i, \mathbf{a}_j) = (\gamma \langle \mathbf{a}_i \mathbf{a}_j \rangle + b)^p \tag{2}$$

  - Based on neural net elements

$$K(\mathbf{a}_i, \mathbf{a}_j) = \tanh(\kappa \langle \mathbf{a}_i \mathbf{a}_j \rangle - \delta)^p \tag{3}$$

SVM For Multiple Classes

- Build n "1-vs-all" classifiers:
  - It costs n times the complexity of one classifier and the most confident answer should be chosen.
- Build $\frac{n(n-1)}{2}$ "1-vs-1" classifiers:
  - The instances are assigned by voting, so many classifiers have a small number of instances.
- Large Margin DAG's for Multiclass Classifications (Platt) means the Decision Directed Acyclic Graph (DDAG), which is used to combine many two-class classifiers into a multiclass classifier.
- Probabilities are calibrated by logistic regression on the SVM's scores.

Conclusions

- Effective in cases where number of dimensions is greater than the number of samples.
- Support Vector Machines have different performance depending on the scaling of the data.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Complexity dependent on the number of support vectors (and also the kernel type) and is generally between $O(n^2)$ and $O(n^3)$ with n the amount of training instances.
- Performance depends on choice of kernel and parameters.
- If the number of features is much greater than the number of samples, the method is likely to give poor performances.