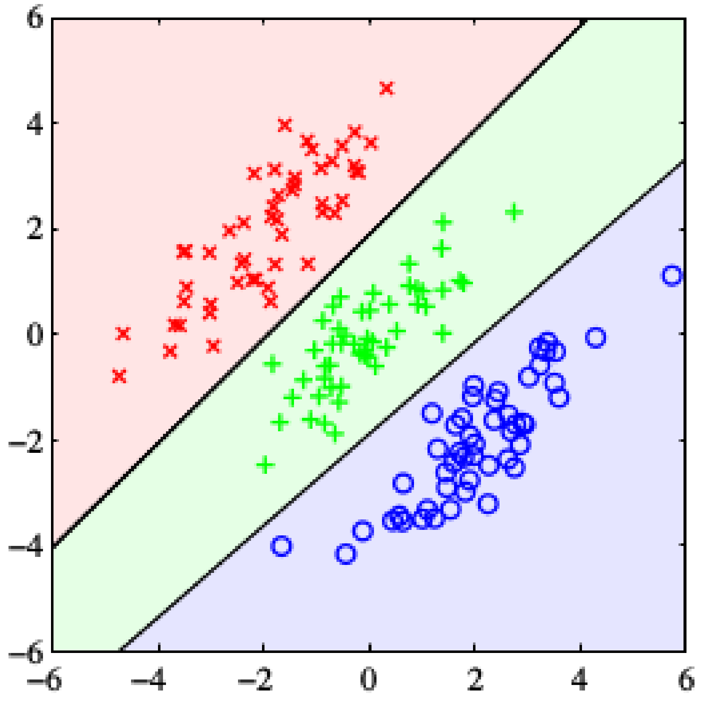
Lecture 5: Linear models for classification. Logistic regression. Gradient Descent. Second-order methods.

- Linear models for classification
- Logistic regression
- Gradient descent and second-order methods
- Kernelizing linear methods

Regression vs classification



- Regression: map $\mathbf{x} \in \mathbb{R}^n$ to $y \in \mathbb{R}$.
- Classification: map $\mathbf{x} \in \mathbb{R}^n$ to a *label* $y \in \mathcal{Y} = \{1, \cdots, k\}$ (k classes).
- Input space is divided into *decision regions*.
- Linear classification: decisions surfaces are linear (or affine) in $\mathbf{x}$.

- How to represent the output? If $k = 2$, $\mathcal{Y} = \{-1, 1\}$ can be convenient... One-hot encoding is another option...
- We will mainly talk about binary classification, i.e. $k = 2$.

Linear models for classification

- Hyperplane in $\mathbb{R}^n$: defined by the equation $\mathbf{w}^\top \mathbf{x} = 0$ for some $\mathbf{w} \in \mathbb{R}^n$.
- Least-squares for classification:

$$\mathcal{Y} = \{1, \cdots, k\} \quad \text{and} \quad h_{\mathbf{W}}(\mathbf{x}) = \arg\max_{i=1,\cdots,k} (\mathbf{W}^\top \mathbf{x})_i$$

  where $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_k \end{bmatrix} \in \mathbb{R}^{n \times k}$.

  - Learning: encode each output sample y_i to a vector in $\mathbb{R}^k$ using one-hot encoding and minimize the squared error (closed form solution).

Linear models for classification

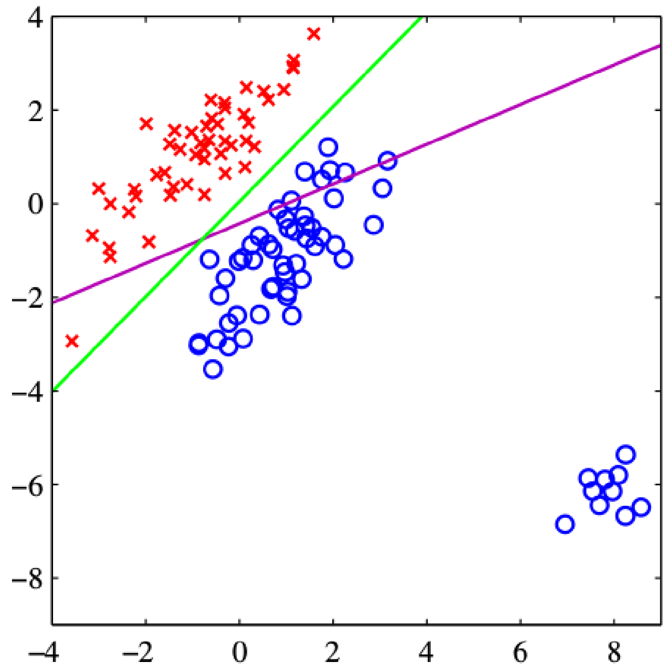
- <span>Least-squares</span> for classification:

$$\mathcal{Y} = \{1, \cdots, k\} \quad \text{and} \quad h_{\mathbf{W}}(\mathbf{x}) = \arg\max_{i=1,\cdots,k}(\mathbf{W}^\top \mathbf{x})_i$$

  where $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_k \end{bmatrix} \in \mathbb{R}^{n \times k}$.

  - Learning: encode each output sample y_i to a vector in $\mathbb{R}^k$ using one-hot encoding and minimize the squared error (closed form solution).

- Very sensitive to outliers:



Linear models for classification

- Least-squares for classification:

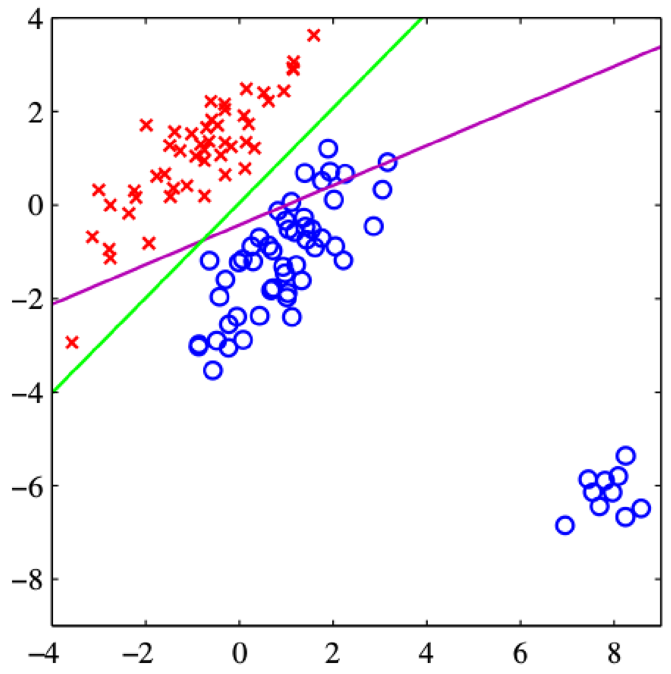
$$\mathcal{Y} = \{1, \cdots, k\} \quad \text{and} \quad h_{\mathbf{W}}(\mathbf{x}) = \arg\max_{i=1,\cdots,k} (\mathbf{W}^\top \mathbf{x})_i$$

  where $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_k \end{bmatrix} \in \mathbb{R}^{n \times k}$.

  – Learning: encode each output sample y_i to a vector in $\mathbb{R}^k$ using one-hot encoding and minimize the squared error (closed form solution).

- Very sensitive to outliers:

  Recall that minimizing the squared error corresponds to maximizing the likelihood under the assumption of a Gaussian conditional distribution.



Linear models for classification

- Hyperplane in $\mathbb{R}^n$: defined by the equation $\mathbf{w}^\top \mathbf{x} = 0$ for some $\mathbf{w} \in \mathbb{R}^n$.
- Binary linear discriminant function:

$$\mathcal{Y} = \{-1, 1\} \text{ and } h_{\mathbf{w}}(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^\top \mathbf{x}).$$

  - Learning: *Perceptron algorithm* (Rosenblatt, 1957)
    - iterative algorithm (closely related to *stochastic gradient descent*, more on that later...).
    - converges when the data is linearly separable.
  - Converges... ok, but to which hyperplane? What is the best one?

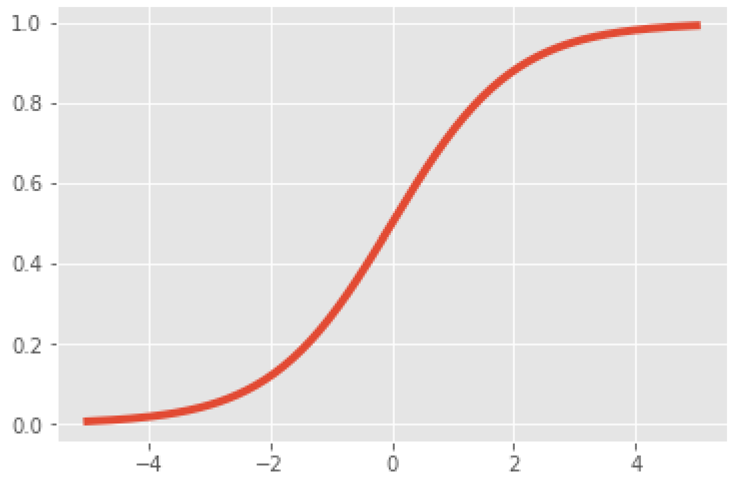
  $\Rightarrow$ *Support Vector Machines* (next lecture)

- More generally, Generalized linear models:

$$h_{\mathbf{w}}(\mathbf{x}) = \psi(\mathbf{w}^\top \mathbf{x}) \text{ for some } \textit{activation function } \psi.$$

$\rightarrow$ If ψ takes its values in $(0, 1)$, we could interpret $h_{\mathbf{w}}(\mathbf{x})$ as the conditional probability $P(y = 1|x)$!

Logistic sigmoid (recall): $\sigma(z) = \frac{1}{1+\exp(-z)}$



- Symmetry: $\sigma(-z) = 1 - \sigma(z)$
- Inverse: $z = \ln\left(\frac{\sigma(z)}{1-\sigma(z)}\right)$
- Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Logistic regression

- Suppose we represent the hypothesis itself as a logistic function of a linear combination of inputs:

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}$$

  This is also known as a *sigmoid neuron*
- Suppose we interpret $h(\mathbf{x})$ as $P(y = 1|\mathbf{x})$
- Then the log-odds ratio,

$$\ln\left(\frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})}\right) = \mathbf{w}^\top \mathbf{x}$$

  is linear in $\mathbf{x}$ (nice!)
- The optimum weights will maximize the *conditional likelihood* of the outputs, given the inputs.

The cross-entropy error function

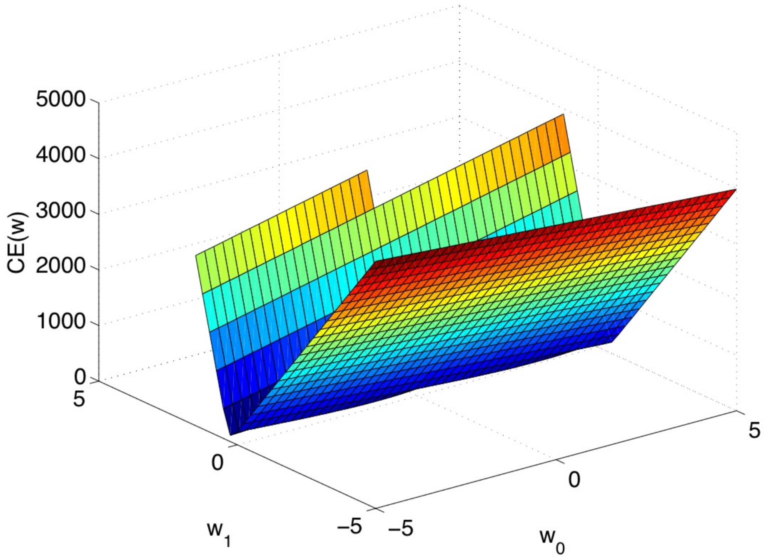
- Suppose we interpret the output of the hypothesis, $h(\mathbf{x}_i)$, as the probability that $y_i = 1$
- Setting $\mathcal{Y} = \{0, 1\}$ for convenience, the log-likelihood of a hypothesis h is:

$$\begin{aligned} \log L(h) &= \sum_{i=1}^{m} \log P(y_i | \mathbf{x}_i, h) = \sum_{i=1}^{m} \begin{cases} \log h(\mathbf{x}_i) & \text{if } y_i = 1 \\ \log(1 - h(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases} \\ &= \sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i)) \end{aligned}$$

- The *cross-entropy error function* is the opposite quantity:

$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i)) \right)$$

Cross-entropy error surface for logistic function

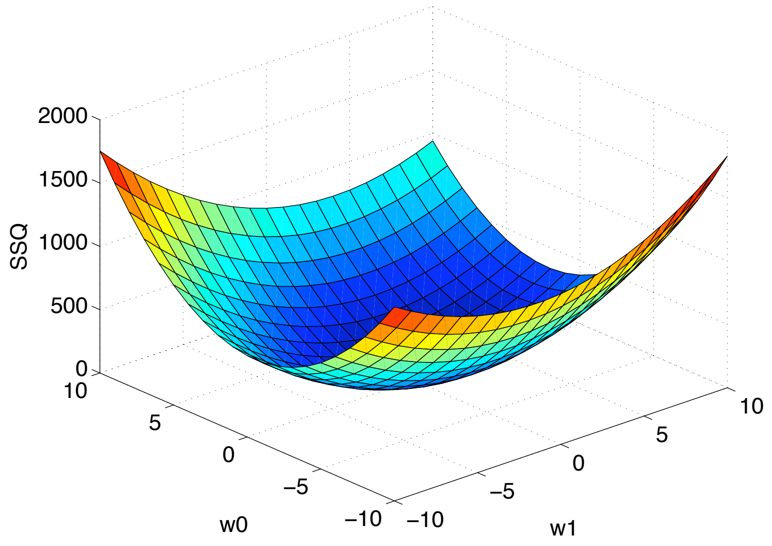


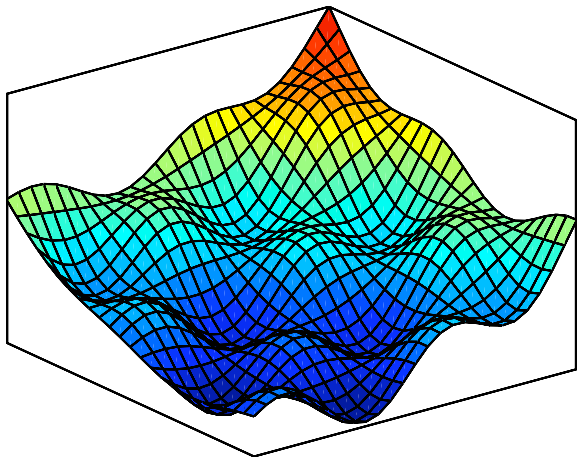
$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_i))\right)$$

Nice error surface, unique minimum, but *cannot be solved in closed form*

Gradient descent

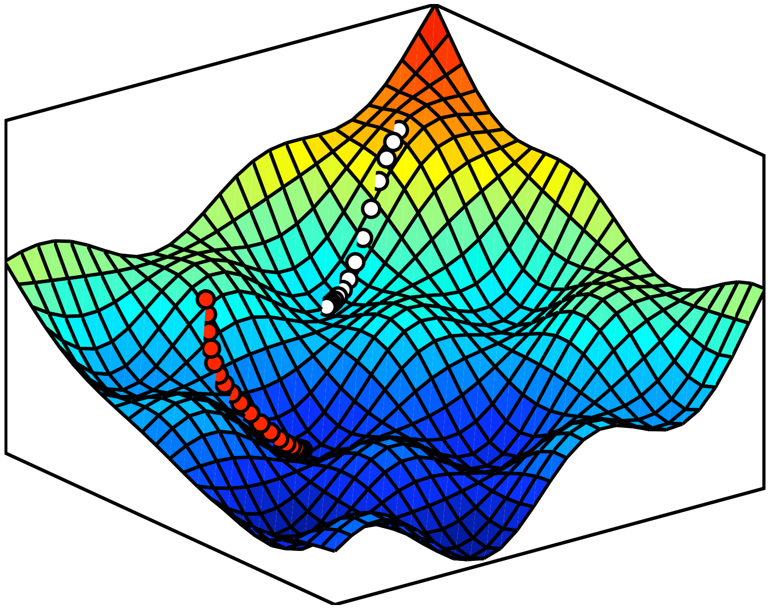
- The gradient of J at a point $\mathbf{w}$ can be thought of as a vector indicating which way is "uphill".





- If this is an error function, we want to move "downhill" on it, i.e., in the direction opposite to the gradient

Example gradient descent traces



- If the error function is *convex*, gradient descent converges to the global minimum (under some conditions on the learning rates)
- For more general hypothesis classes, there may be local optima
- In this case, the final solution may depend on the initial parameters

Gradient descent algorithm

- The basic algorithm assumes that ∇J is easily computed
- We want to produce a sequence of vectors $\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3, \ldots$ with the goal that:
  - $J(\mathbf{w}^1) > J(\mathbf{w}^2) > J(\mathbf{w}^3) > \ldots$
  - $\lim_{i\to\infty} \mathbf{w}^i = \mathbf{w}$ and $\mathbf{w}$ is locally optimal.
- The algorithm: Given $\mathbf{w}^0$, do for $i = 0, 1, 2, \ldots$

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \alpha_i \nabla J(\mathbf{w}^i) \ ,$$

where $\alpha_i > 0$ is the *step size* or *learning rate* for iteration i.

Maximization procedure: Gradient ascent

- First we compute the gradient of $\log L(\mathbf{w})$ wrt $\mathbf{w}$:

$$\begin{aligned}
\nabla \log L(\mathbf{w}) &= \sum_i y_i \frac{1}{h_{\mathbf{w}}(\mathbf{x}_i)} h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \\
&\quad + (1 - y_i)\frac{1}{1 - h_{\mathbf{w}}(\mathbf{x}_i)} h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \\
&= \sum_i \mathbf{x}_i(y_i - y_i h_{\mathbf{w}}(\mathbf{x}_i) - h_{\mathbf{w}}(\mathbf{x}_i) + y_i h_{\mathbf{w}}(\mathbf{x}_i)) \\
&= \sum_i (y_i - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \\
&= \mathbf{X}^\top(\mathbf{y} - \hat{\mathbf{y}})
\end{aligned}$$

where $\hat{\mathbf{y}} \in \mathbb{R}^m$ is defined by $(\hat{\mathbf{y}})_i = h_{\mathbf{w}}(\mathbf{x}_i)$.

- The update rule (because we maximize) is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla \log L(\mathbf{w}) = \mathbf{w} + \alpha \mathbf{X}^\top (\mathbf{y} - \hat{\mathbf{y}})$$

  where $\alpha \in (0,1)$ is a step-size or learning rate parameter
- This is called *logistic regression*
- If one uses features of the input, we simply have:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{\Phi}^\top (\mathbf{y} - \hat{\mathbf{y}})$$

Roadmap: theoretical guarantees for gradient descent and second-order methods

- If the cost function is convex then gradient descent will converge to the optimal solution for an appropriate choice of the learning rates.
- We will show that the cross-entropy error function is convex.
- We will see how we can use a second-order method to choose "optimal" learning rates.

Convexity

- A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if for all $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, $\lambda \in [0, 1]$:

$$f(\lambda\mathbf{a} + (1 - \lambda)\mathbf{b}) \leq \lambda f(\mathbf{a}) + (1 - \lambda) f(\mathbf{b}).$$

- f is concave if $-f$ is convex.
- If f and g are convex functions, $\alpha f + \beta g$ is also convex for any real numbers α and β.

Characterizations of convexity

- *First-order* characterization:

$$f \text{ is convex} \Leftrightarrow \text{for all } \mathbf{a}, \mathbf{b} : \quad f(\mathbf{a}) \geq f(\mathbf{b}) + \nabla f(\mathbf{b})^\top (\mathbf{a} - \mathbf{b})$$

  (the function is globally above the tangent at $\mathbf{b}$).
- *Second-order* characterization:

$$f \text{ is convex} \Leftrightarrow \text{the Hessian of } f \text{ is positive semi-definite.}$$

  – The Hessian contains the second-order derivatives of f:

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

  – $\mathbf{H}$ is positive semi-definite $\Leftrightarrow \mathbf{a}^\top \mathbf{H} \mathbf{a} \geq 0$ for all $\mathbf{a} \in \mathbb{R}^d$
  $\Leftrightarrow \mathbf{H}$ has only non-negative eigenvalues.

Logistic regression: convexity of the cost function

$$J(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_i))\right)$$

where σ is the sigmoid function.

- We show that $-\log \sigma(\mathbf{w}^\top \mathbf{x})$ and $-\log(1-\sigma(\mathbf{w}^\top \mathbf{x}))$ are convex in $\mathbf{w}$:

$$\nabla_{\mathbf{w}}\left(-\log \sigma(\mathbf{w}^\top \mathbf{x})\right) = -\frac{\nabla_{\mathbf{w}}\left(\sigma(\mathbf{w}^\top \mathbf{x})\right)}{\sigma(\mathbf{w}^\top \mathbf{x})} = -\frac{\sigma'(\mathbf{w}^\top \mathbf{x})\nabla_{\mathbf{w}}(\mathbf{w}^\top \mathbf{x})}{\sigma(\mathbf{w}^\top \mathbf{x})} = (\sigma(\mathbf{w}^\top \mathbf{x})-1)\mathbf{x}$$

$$\nabla^2_{\mathbf{w}}\left(-\log \sigma(\mathbf{w}^\top \mathbf{x})\right) = \nabla_{\mathbf{w}}\left(\sigma(\mathbf{w}^\top \mathbf{x})\mathbf{x}\right) = \sigma(\mathbf{w}^\top \mathbf{x})(1-\sigma(\mathbf{w}^\top \mathbf{x}))\mathbf{x}\mathbf{x}^\top$$

$\Rightarrow$ It is easy to check that this matrix is positive semi-definite for any $\mathbf{x}$.

Convexity of the cost function

$$J(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_i))\right)$$

where $\sigma(z) = 1/(1+e^{-z})$ (check that $\sigma'(z) = \sigma(z)(1-\sigma(z))$).

- Similarly you can show that

$$\nabla_{\mathbf{w}}\left(-\log(1-\sigma(\mathbf{w}^\top \mathbf{x}))\right) = \sigma(\mathbf{w}^\top \mathbf{x})\mathbf{x}$$

$$\nabla^2_{\mathbf{w}}\left(-\log(1-\sigma(\mathbf{w}^\top \mathbf{x}))\right) = \sigma(\mathbf{w}^\top \mathbf{x})(1-\sigma(\mathbf{w}^\top \mathbf{x}))\mathbf{x}\mathbf{x}^\top$$

$\Rightarrow$ $J(\mathbf{w})$ is convex in $\mathbf{w}$.

$\Rightarrow$ The gradient of J is $\mathbf{X}^\top(\hat{\mathbf{y}} - \mathbf{y})$ where $\hat{\mathbf{y}}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i) = h(\mathbf{x}_i)$.

$\Rightarrow$ The Hessian of J is $\mathbf{X}^\top \mathbf{R} \mathbf{X}$ where $\mathbf{R}$ is diagonal with entries $\mathbf{R}_{i,i} = h(\mathbf{x}_i)(1-h(\mathbf{x}_i))$.

Another algorithm for optimization

- Recall Newton's method for finding the zero of a function $g : \mathbb{R} \to \mathbb{R}$
- At point w^t, approximate the function by a straight line (its tangent)
- Solve the linear equation for where the tangent equals 0, and move the parameter to this point:

$$w^{t+1} = w^t - \frac{g(w^t)}{g'(w^t)}$$

Application to machine learning

- Suppose for simplicity that the error function J has only one parameter
- We want to optimize J, so we can apply Newton's method to find the zeros of $J' = \frac{d}{dw}J$
- We obtain the iteration:

$$w^{t+1} = w^t - \frac{J'(w^t)}{J''(w^t)}$$

- Note that there is *no step size parameter*!
- This is a *second-order method*, because it requires computing the second derivative
- But, if our error function is quadratic, this will find the global optimum in one step!

Second-order methods: Multivariate setting

- If we have an error function J that depends on many variables, we can compute the *Hessian matrix*, which contains the second-order derivatives of J:

$$\mathbf{H}_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

- The inverse of the Hessian gives the "optimal" learning rates
- The weights are updated as:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}\nabla_{\mathbf{w}} J$$

- This is also called Newton-Raphson method for logistic regression, or Fisher scoring

Which method is better?

- Newton's method usually requires significantly fewer iterations than gradient descent
- Computing the Hessian requires a batch of data, so there is no natural on-line algorithm
- Inverting the Hessian explicitly is expensive, but almost never necessary
- Computing the product of a Hessian with a vector can be done in linear time (Pearlmutter, 1993), which helps also to compute the product of the inverse Hessian with a vector without explicitly computing $\mathbf{H}$

Newton-Raphson for logistic regression

- Leads to a nice algorithm called *iteratively reweighted least squares* (or iterative recursive least squares)
- The Hessian has the form:

$$\mathbf{H} = \mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi}$$

  where $\mathbf{R}$ is the diagonal matrix of $h(\mathbf{x}_i)(1 - h(\mathbf{x}_i))$ (you can check that this is the form of the second derivative).
- The weight update becomes:

$$\mathbf{w} \leftarrow \mathbf{w} - (\mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top (\hat{\mathbf{y}} - \mathbf{y})$$

  which can be rewritten as the solution of a weighted least square problem:

$$\mathbf{w} \leftarrow (\mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{R} (\mathbf{\Phi} \mathbf{w} - \mathbf{R}^{-1} (\hat{\mathbf{y}} - \mathbf{y}))$$

Regularization for logistic regression

- One can do regularization for logistic regression just like in the case of linear regression
- Recall regularization makes a statement about the weights, so does not affect the error function
- Eg: L_2 regularization will have the optimization criterions:

$$J(\mathbf{w}) = J_D(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$

Probabilistic view of logistic regression

- Consider the additive noise model we discussed before:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon$$

  where ϵ are drawn iid from some distribution
- At first glance, log reg does not fit very well
- We will instead think of a latent variable $\hat{y}_i$ such that:

$$\hat{y}_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon$$

- Then the output is generated as:

$$y_i = 1 \text{ iff } \hat{y}_i > 0$$

Generalized Linear Models

- Logistic regression is a special case of a generalized linear model:

$$E[Y \mid \mathbf{x}] = g^{-1}(\mathbf{w}^\top \mathbf{x}).$$

  g is called the *link function*, it relates the mean of the response to the linear predictor.

- Linear regression: $E[Y \mid \mathbf{x}] = E[\mathbf{w}^\top \mathbf{x} + \varepsilon \mid \mathbf{x}] = \mathbf{w}^\top \mathbf{x}$ (g is the identity).
- Logistic regression: $E[Y \mid \mathbf{x}] = P(Y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$
- Poisson regression: $E[Y \mid \mathbf{x}] = \exp(\mathbf{w}^\top \mathbf{x})$ (for count data).
- ...

Linear regression with feature vectors revisited

- Recall: we want to minimize the (regularized) error function:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^\top(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

- Using the identity $(\mathbf{M}^\top\mathbf{M} + \alpha\mathbf{I})^{-1}\mathbf{M}^\top = \mathbf{M}^\top(\mathbf{M}\mathbf{M}^\top + \alpha\mathbf{I})^{-1}$, the solution can be rewritten as

$$\mathbf{w} = (\mathbf{\Phi}^\top\mathbf{\Phi} + \lambda\mathbf{I}_n)^{-1}\mathbf{\Phi}^\top\mathbf{y} = \mathbf{\Phi}^\top(\mathbf{\Phi}\mathbf{\Phi}^\top + \lambda\mathbf{I}_m)^{-1}\mathbf{y} = \mathbf{\Phi}^\top\mathbf{a}$$

  with $\mathbf{a} \in \mathbb{R}^m$.

$\Rightarrow$ Inversion of an $m \times m$ matrix instead of $n \times n$!

$\Rightarrow$ The solution $\mathbf{w}$ is a linear combination of input points!

Linear regression with feature vectors revisited (cont'd)

- Let $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top \in \mathbb{R}^{m \times m}$ be the so-called Gram matrix.
  - $\mathbf{K}_{i,j} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ measures the similarity between inputs i and j.
  - No need to compute the feature map, just need to know how to compute $\phi(\mathbf{u})^\top \phi(\mathbf{v})$ for any $\mathbf{u}, \mathbf{v}$.
- Solution of regularized least squares: $\mathbf{w} = \mathbf{\Phi}^\top \mathbf{a}$ with $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}$.
- The predictions for the input data are given by

$$\hat{\mathbf{y}} = \mathbf{\Phi}\mathbf{w} = \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{a} = \mathbf{K}(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}.$$

- The prediction for a new input point $\mathbf{x}$ is given by

$$y = \phi(\mathbf{x})^\top \mathbf{\Phi}^\top \mathbf{a} = \mathbf{k}_\mathbf{x}^\top \mathbf{a} \quad \text{where } \mathbf{k}_\mathbf{x} \in \mathbb{R}^m \text{ is defined by } (\mathbf{k}_\mathbf{x})_i = \phi(\mathbf{x})^\top \phi(\mathbf{x}_i)$$

$\Rightarrow$ Never need to compute the feature map ϕ explicitly!

Another derivation: Linear regression with feature vectors revisited

- Find the weight vector $\mathbf{w}$ which minimizes the (regularized) error function:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^\top(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

- Instead of closed-form solution, take the gradient and rearrange the terms
- Setting the learning rate $\alpha = \frac{1}{\lambda}$, the solution takes the form:

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{i=1}^{m}(\mathbf{w}^\top\phi(\mathbf{x}_i) - y_i)\phi(\mathbf{x}_i) = \sum_{i=1}^{m} a_i\phi(\mathbf{x}_i) = \mathbf{\Phi}^\top\mathbf{a}$$

  where $\mathbf{a}$ is a vector of size m (with $a_i = -\frac{1}{\lambda}(\mathbf{w}^\top\phi(\mathbf{x}_i) - y_i)$)
- Main idea: *use $\mathbf{a}$ instead of $\mathbf{w}$ as parameter vector*

Another derivation: Re-writing the error function

- Instead of $J(\mathbf{w})$ we have $J(\mathbf{a})$:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{a} - \mathbf{a}^\top \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{y} + \frac{1}{2}\mathbf{y}^\top \mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{a}$$

- Denote $\mathbf{\Phi}\mathbf{\Phi}^\top = \mathbf{K}$
- Hence, we can re-write this as:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top \mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^\top \mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^\top \mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top \mathbf{K}\mathbf{a}$$

- This is quadratic in $\mathbf{a}$, and we can set the gradient to 0 and solve.

Another derivation: Dual-view regression

- By setting the gradient to 0 we get:

$$\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1}\mathbf{y}$$

- Note that this is similar to re-formulating a weight vector in terms of a linear combination of instances
- Again, the *feature mapping is not needed* either to learn or to make predictions!
- This approach is useful if the feature space is very large

Kernel functions

- Whenever a learning algorithm can be written in terms of dot-products, it can be generalized to kernels.
- A *kernel* is any function $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to a dot product for some feature mapping ϕ:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) \text{ for some } \phi.$$

- Conversely, by choosing a feature map $\phi : \mathbb{R}^n \to \mathbb{R}^\ell$, we implicitly choose a kernel function (ℓ may even be infinite!)
- Recall that $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = \cos \angle(\mathbf{x}_1, \mathbf{x}_2)$ where $\angle$ denotes the angle between the vectors, so a kernel function can be thought of as a notion of *similarity*.

Example: Quadratic kernel

- Let $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.
- Is this a kernel?

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) = \sum_{i,j \in \{1 \ldots n\}} x_i z_i x_j z_j \\ &= \sum_{i,j \in \{1 \ldots n\}} (x_i x_j)(z_i z_j) \end{aligned}$$

- Hence, it is a kernel, with feature mapping:

$$\phi(\mathbf{x}) = \langle x_1^2,\ x_1 x_2,\ \ldots,\ x_1 x_n,\ x_2 x_1,\ x_2^2,\ \ldots,\ x_n^2 \rangle$$

  Feature vector includes all squares of elements and all cross terms.
- Note that computing ϕ takes $O(n^2)$ but *computing K takes only $O(n)$*!