UiO : Department of Informatics
University of Oslo

INF 5860 Machine learning for image classification

Lecture 4 : From regression to classification

Anne Solberg

February 10, 2017

Today's reading material

- CS 231n note on linear classifiers
  http://cs231n.github.io/linear-classify/
- CS 229 note on supervised classification:
  http://cs229.stanford.edu/notes/cs229-notes1.pdf

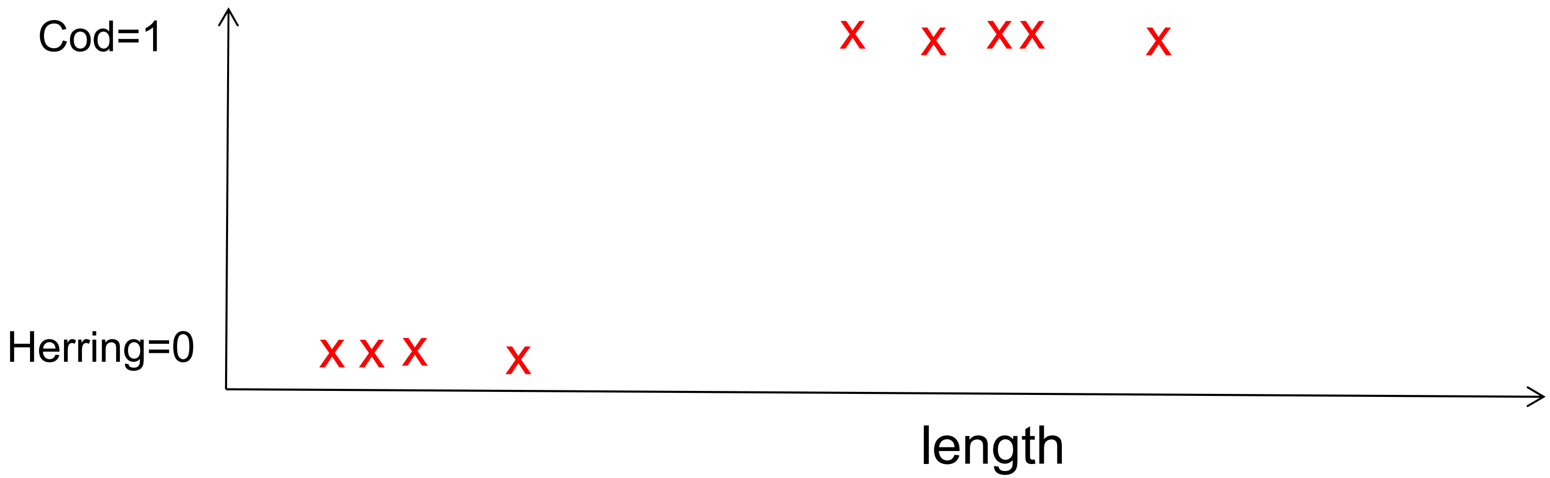
- SVM is included for reference, as it is a commonly used classifier. Details of this is not essential. See http://cs229.stanford.edu/notes/cs229-notes3.pdf

Topics

- Let us show how a regression problem can be transformed into a binary (2-class) classification problem using a nonlinear loss function.
- Then generalize to multiple classes using softmax
- Image-based classifiers f(X,W)
- Regularization terms in the loss function.

Introduction

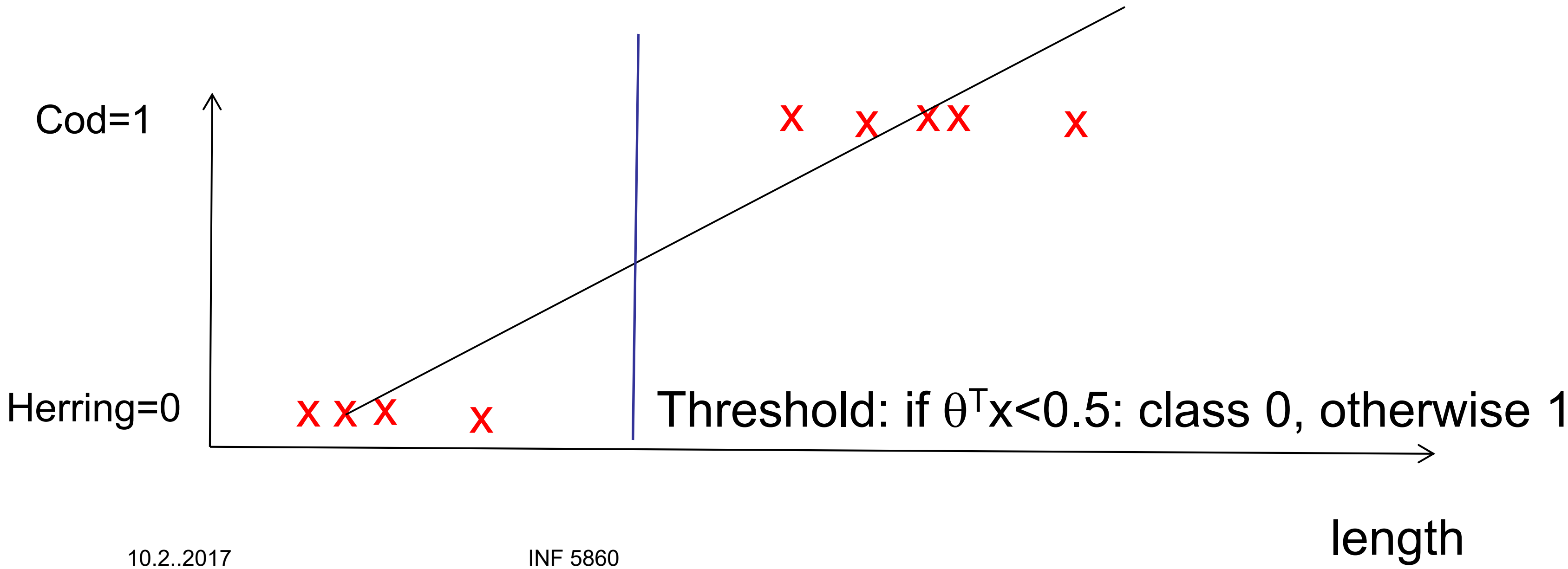
- Consider classification into 2 classes. Call the classes 0 and 1 (or negative and positive)
- Example: classify fish species based on length



What would linear regression give?

- Maybe we would threshold this?

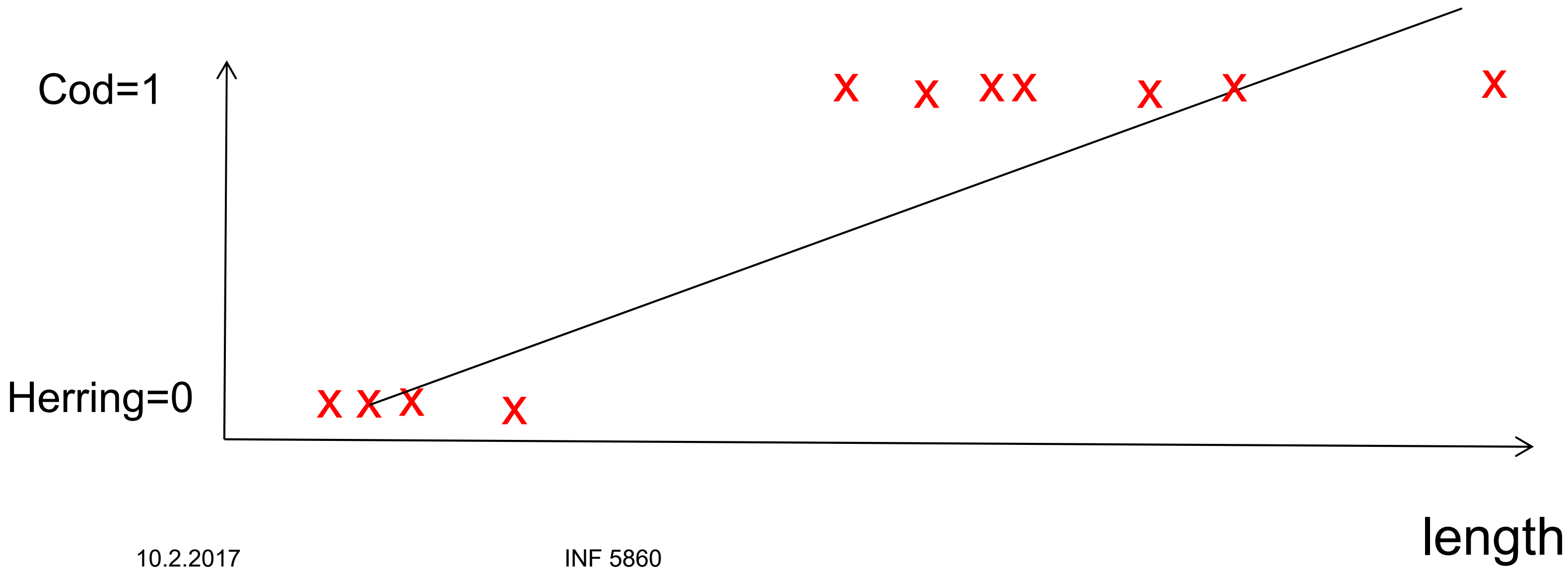
$$h_\theta(x) = \theta_0 + \theta_1 x$$



Threshold: if $\theta^T x < 0.5$: class 0, otherwise 1

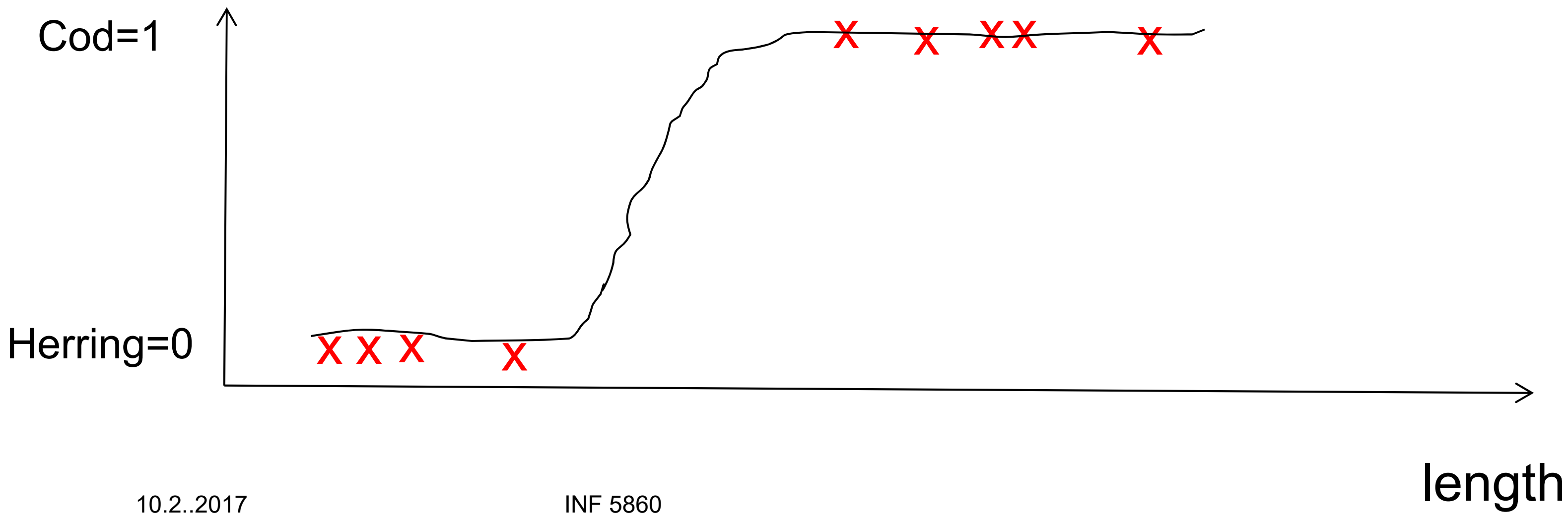
What would linear regression give?

- But what if we got more data?
  The line (and threshold) would change completely!



What if we fitted it to a function that is close to either 0 or 1?

- Hypothesis $h_\theta(x)$ is now a non-linear function of x
  Classification: y=0 or 1
  Threshold $h_\theta(x)$: if $h_\theta(x)$>1 : set y=1, otherwise set y=0
- Desirable to have $0 \leq h_\theta(x) \leq 1$



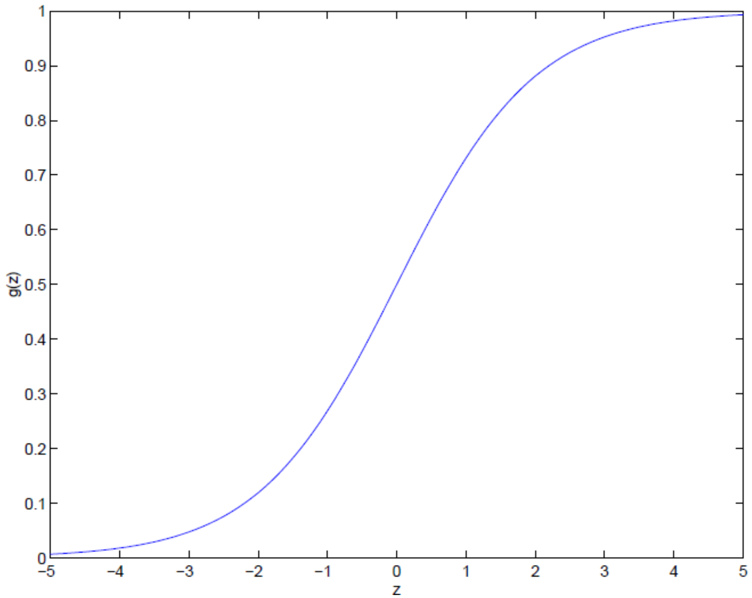
Logistic regression model

- Want $0 \le h_\theta(x) \le 1$
- Let

$$h_\theta(X) = g\left(\theta^T x\right)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$h_\theta(X) = \frac{1}{1+e^{-\theta^T x}}$$



- This is called the sigmoid function

Decisions for logistic regression

- Decide y=1 if $h_\theta(x)>$ 0.5, and y=0 otherwise

$$h_\theta(X) = g\left(\theta^T x\right)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$h_\theta(X) = \frac{1}{1+e^{-\theta^T x}}$$

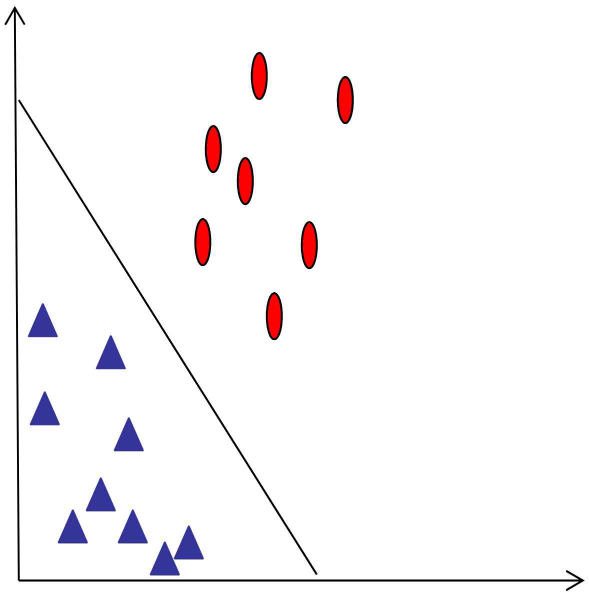
- g(z)>0.5 if z>0
  - $\theta^T x>0$

g(z)<0.5 if z<0

$\theta^T x<0$

$\theta^T x=0$ gives the decision boundary

An example with 2 features



Notation: X(i,j) is feature j for sample i

$$h_\theta(x) = g(\theta_0 + \theta_1 X(i,1) + \theta_2 X(i,2))$$

Predict y=1 if $x_1+2x_2-4 \geq 0$

$$\left(\theta^T x\right)$$

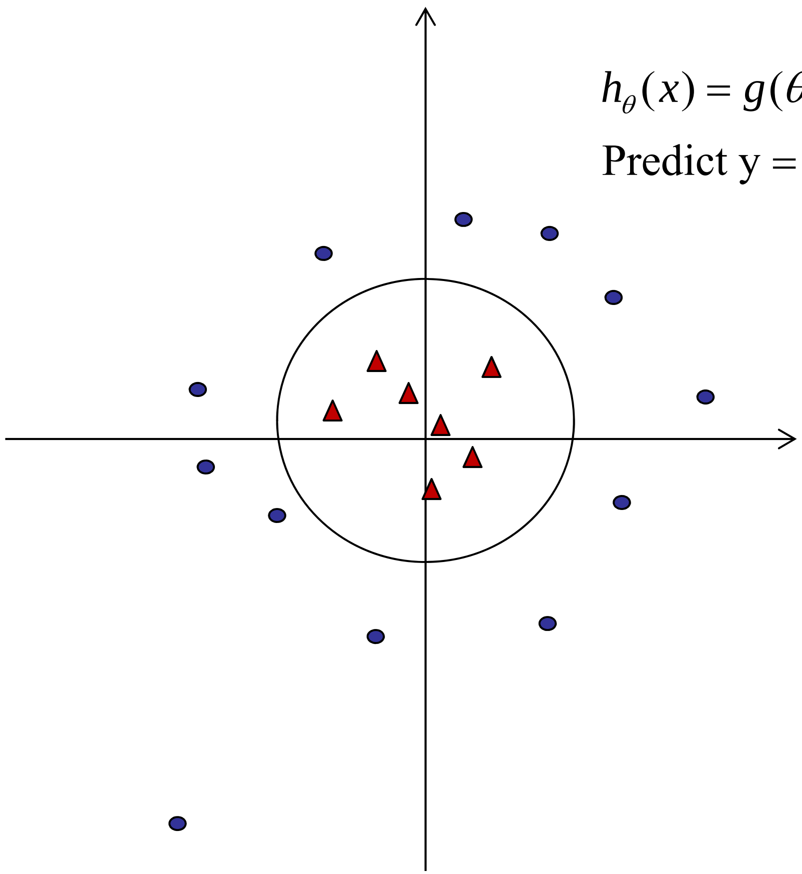
Decision boundary $x_1+2x_2-4=0$

If we KNOW θ_0, θ_1 and θ_2 classification is based on which **side** of the boundary we are on, in terms of the **sign of** $\boldsymbol{\theta^T x}$

Nonlinear boundary by including higher-order terms

$$h_\theta(x) = g(\theta_0 + \theta_1 X(i,1) + \theta_2 X(i,2)) + \theta_3 X(i,1)^2 + \theta_4 X(i,2)^2)$$

$$\text{Predict y} = 1 \text{ if } -1 + X(i,1)^2 + X(i,2)^2 \geq 0$$



Logistic cost function

- Training set

$$X = \begin{bmatrix} X(1,0) & X(1,1) & .. & X(1,n) \\ X(2,0) & X(2,1) & \dots & X(2,n) \\ \vdots & \vdots & \dots & \vdots \\ X(m,0) & X(m,1) & \cdots & X(m,n) \end{bmatrix} \qquad y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(m) \end{bmatrix}$$

$X(:, j)$ Column j : all samples for feature j

X(i,:) Row i : all features for sample i

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

- How do we set θ to have high classification accuracy?

Logistic regression cost

Minimize $$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(Cost(h_\theta(x_i), y_i)\right)$$

Due to the sigmoid function g(z), this is a non-quadratic function, and non-convex.

Set
$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$$\text{Cost} = 0 \text{ if } y = 1 \text{ and } h_\theta(x) = 1$$

$$\text{If } y = 1 \text{ and } h_\theta(x) \to 0 : \text{Cost} \to \infty$$

$$\text{Cost} = 0 \text{ if } y = 0 \text{ and } h_\theta(x) = 0$$

$$\text{If } y = 0 \text{ and } h_\theta(x) \to 1 : \text{Cost} \to \infty$$

Mimick a probability

We skip deriving this cost,
it is derived by maximizing the
log-likelihood that θ fits the data

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if y} = 1 \\ -\log(1 - h_\theta(x)) & \text{if y} = 0 \end{cases}$$

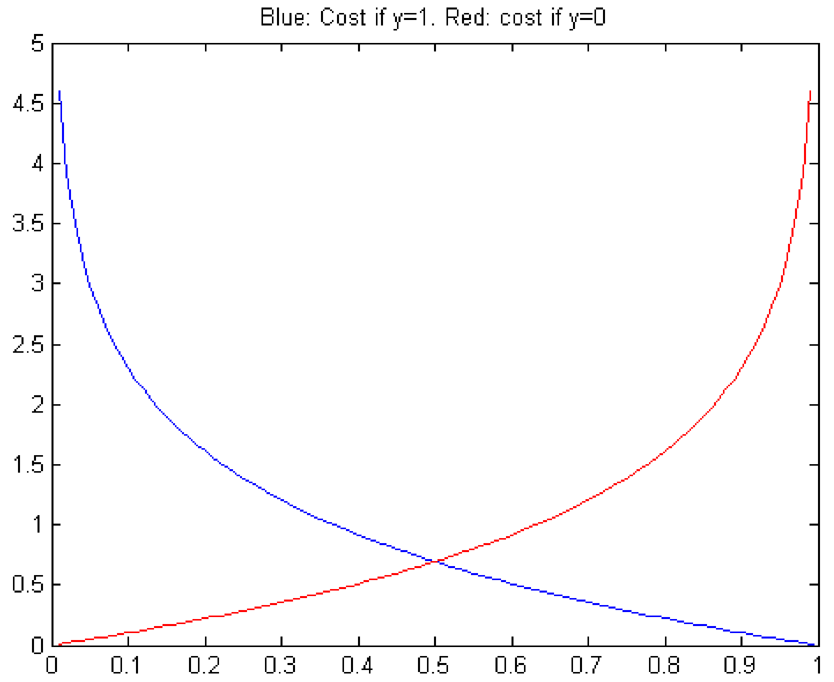
$\text{Cost} = 0 \text{ if } \text{y} = 1 \text{ and } h_\theta(x) = 1$

$\text{If y} = 1 \text{ and } h_\theta(x) \rightarrow 0 : \text{Cost} \rightarrow \infty$

$\text{Cost} = 0 \text{ if } \text{y} = 0 \text{ and } h_\theta(x) = 0$

$\text{If y} = 0 \text{ and } h_\theta(x) \rightarrow 1 : \text{Cost} \rightarrow \infty$

Mimick a probability



Cost function-compact notation

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta(X(i,:), y(i))\right)$$

$$= -\frac{1}{m}\left[\sum_{i=1}^{m} y(i)\log h_\theta(X(i,:)) + (1-y(i))\log(1-h_\theta(X(i,:)))\right]$$

To find θ: find θ that minimize J(θ)

To classify a new sample X(i,:):

$$\text{Output } h_\theta(X(i,:)) = \frac{1}{1+e^{-\theta^T X(i,:)}}$$

Gradient descent of J(θ)

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y(i)\log h_\theta(X(i,:) + (1-y(i)))\log(1-h_\theta(X(i,:))\right]$$

To find θ : find θ that minimize J(θ) using gradient descent

Repeat :

$$\theta_j = \theta_j - \varepsilon\frac{\partial}{\partial\theta_j}$$

$$\theta_j - \varepsilon\frac{1}{m}\sum_{i=1}^{m}((h_\theta(X(i,:)) - y(i))X(i,j))$$

This algorithm looks similar to linear regression, but now

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

How to include regularization

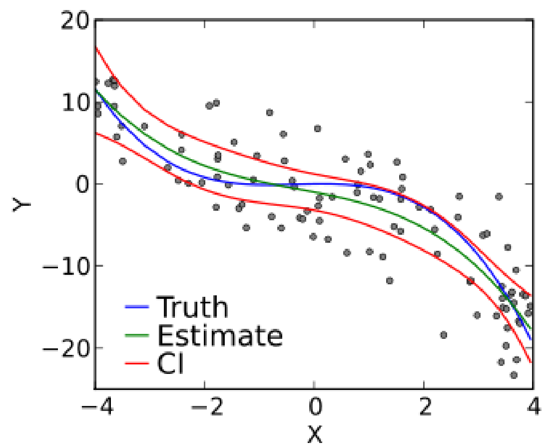
- Last week: the importance of not overfitting to training data.
- Too many parameters: risk of overfitting

Repetition: Polynomial regression

- If a linear model is not sufficient, we can extend to allow higher-order terms or cross-terms between the variables by changing our hypothesis $h_\theta(x)$

$$h_\theta(x) = \theta^0 + \theta^1 x^1 + \theta^2 (x^1)^2 + \theta^3 (x^1)^3 \ldots$$

$$h_\theta(x) = \theta^0 + \theta^1 x^1 + \theta^2 \sqrt{x^1}$$

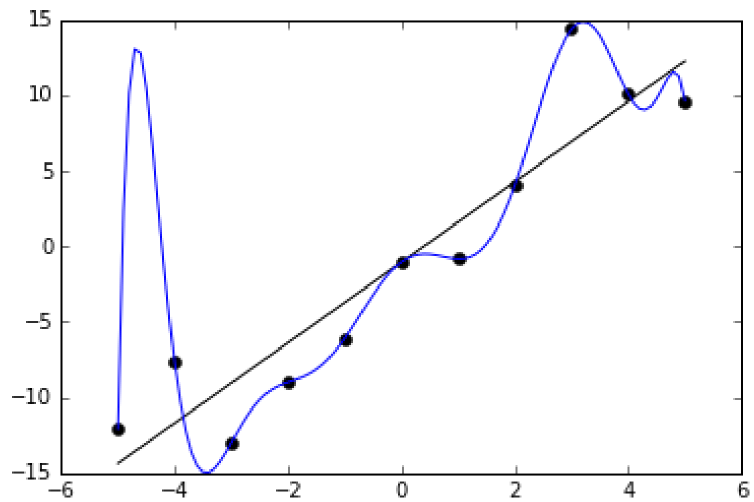


The danger of overfitting

A higher-order model can easily overfit the training data

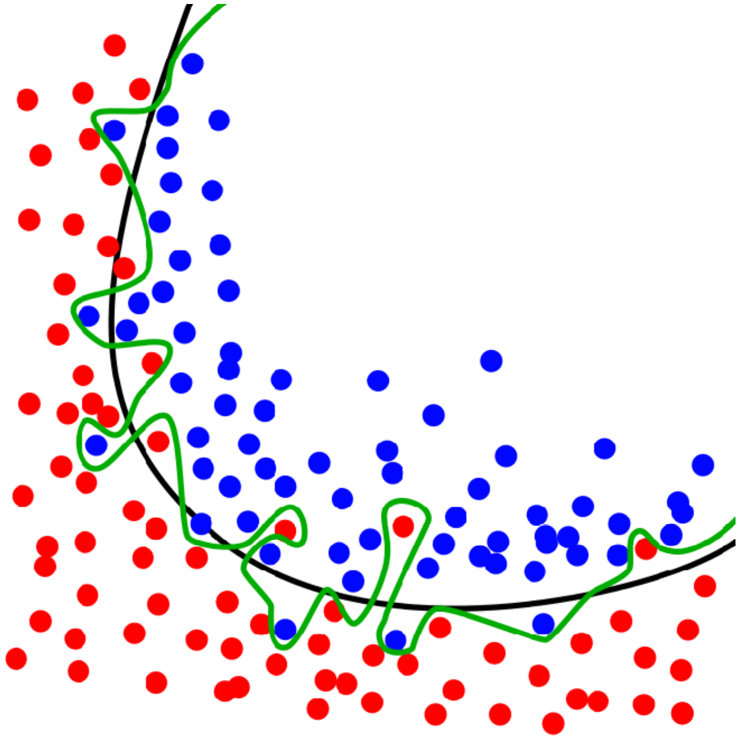
For the higher order terms:

- The higher the value of the coefficients, the more the curve can fluctuate
- This is not valid for the first two coefficients
- Restricting only the value of higher-order terms is difficult in general (e.g. for neural nets)
- But we can restrict the magnitude of the coefficients (except θ_0).

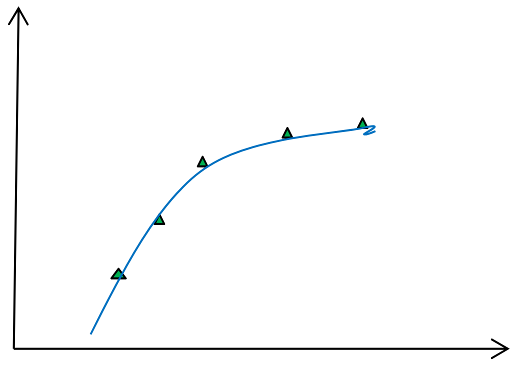


Overfitting for classification

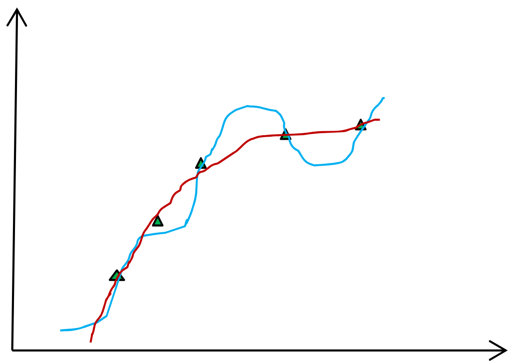
- Overfitting must be avoided for classifiation also – this is partly why we start with simple linear models



Regularization - intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Suppose we add a penalty to restrict θ_3 and θ_4

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(X(i,:)) - y(i)\right)^2 + 100\theta_3 + +100\theta_4$$

To minimize, θ_3 and θ_4 must be small

Regularized cost function

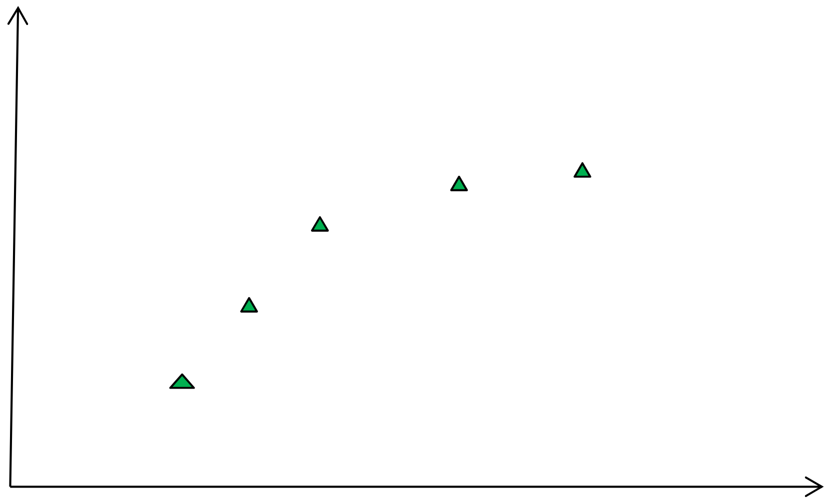
- Simplify the hypothesis by having small values for $\theta_1, \ldots \theta_n$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(X(i,:)) - y(i)\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^{\,2}$$

- λ is the regularization parameter
- This is L2-regularization, later we will see
  - Dropout, max norm…
- Question: Should we restrict θ_0?
  - Think about a linear model $\theta_0 + \theta_1 x$

What if λ is very large?

- Will we get overfit or underfit?



Gradient descent with regularization: linear regression

To find θ :   find θ that minimize J(θ) using gradient descent

Note : NO penalty on θ_0

Repeat :

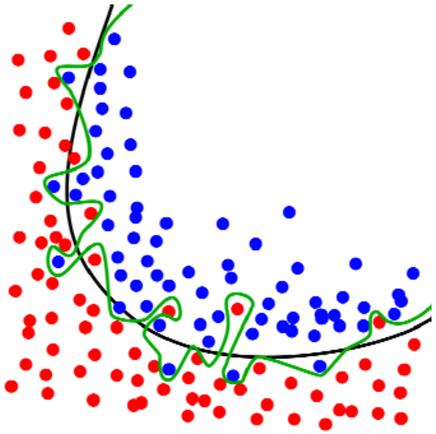
$$\theta_j = \theta_j - \varepsilon \frac{\partial}{\partial \theta_j}$$

$$\theta_0 = \theta_0 - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j))$$

$$\theta_j = \theta_j - \varepsilon \left[\frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j)) + \frac{\lambda}{m} \theta_j \right]$$

$$= \theta_j \left(1 - \varepsilon \frac{\lambda}{m} \right) - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j))$$

Regularized logistic regression



$$h_\theta(x) = \theta_0 + \theta_1 X(:,1) \ + \theta_2 X(:,1)^2 + \theta_3 X(:,1)^2 X(:,2) \ + \theta_4 X(:,1) \ X(:,2)^2 + \cdots$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y(i)\log h_\theta(X(i,:) + (1-y(i))\log(1-h_\theta(X(i,:))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Regularized logistic regression: gradient descent

$$\text{Repeat :}$$

$$\theta_0 = \theta_0 - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,0))$$

$$\theta_j = \theta_j - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j)) + \frac{\lambda}{m}\theta_j$$

$$j = 1,....n$$

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}}$$

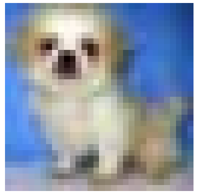
One vs all classification

- From 2 to C classes:
  - Train a logistic classifier $h_{\theta,c}(x)$ for each class c to predict the probability for y=c.
  - Classify new sample x by picking the class c that maximize

$$\max_c h_{\theta,c}(x)$$

Introducing classifying CIFAR images

- CIFAR-10 images: 32x32x3 pixels



- Stack one image into a vector x of length 32x32x3=3072
- Classification will be to find a mapping f(W,x,b) from image space to a set of C classes.
- For CIFAR:

$$x = \begin{bmatrix} \text{pixel 1} \\ \\ \\ \text{pixel 3072} \end{bmatrix} \quad W = \begin{bmatrix} \text{weight for pixel 1 for class 1} & \ldots & \text{weight for pixel 3072 for class 1} \\ \\ \\ \text{weight for pixel 1 for class 10} & & \text{weight for pixel 3072 for class 10} \end{bmatrix} \quad b = \begin{bmatrix} \text{b1} \\ \\ \\ \text{b10} \end{bmatrix}$$

Small example 2 classes

$$\text{Graylevel image} = \begin{bmatrix} 40 & 36 \\ 16 & 12 \end{bmatrix} \quad x = \begin{bmatrix} 40 \\ 36 \\ 16 \\ 12 \end{bmatrix} \quad W = \begin{bmatrix} 0.5 & -1.2 & 0.1 & 2.0 \\ 1.0 & 0.2 & -0.5 & 0.3 \end{bmatrix} \quad \text{b} = \begin{bmatrix} 2.1 \\ 0.3 \end{bmatrix}$$

$$\begin{bmatrix} \text{Score for class 1} \\ \text{Score for class 2} \end{bmatrix} = \begin{bmatrix} 0.5 & -1.2 & 0.1 & 2.0 \\ 1.0 & 0.2 & -0.5 & 0.3 \end{bmatrix} \begin{bmatrix} 40 \\ 36 \\ 16 \\ 12 \end{bmatrix} + \begin{bmatrix} 2.1 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 4.5 \\ 43.1 \end{bmatrix}$$

W: 2x4

One weight w(i,j) for pixel j for class i

- If color image, append the r,g,b bands into one long vector x.
- Note: no spatial information concerning pixel neighbors is used here.
  – Convolutional nets use spatial information.
- All images are standarized to the same size!
  – For CIFAR-10 it is 32x32.
  – If a classifier is trained on CIFAR and we have a new image to classify, resize to 32x32.

W for multiclass image classification

- W is a Cx(n+1)-matrix (C classes, n pixels in the image plus 1 for b)
- We train one linear model pr. class, so each class has a different $\theta_{c,i}$-vector
- If $\theta_{c,i}$- is a vector of length (n+1)

$$W = \begin{bmatrix} \theta_{c=1}{}^T \\ \theta_{c=2}{}^T \\ \vdots \\ \theta_{c=C}{}^T \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ pixel1 \\ \\ \text{pixel 3072} \end{bmatrix} \quad \mathrm{W} = \begin{bmatrix} b_1 & \text{weight for pixel 1 for class 1} & \ldots & \text{weight for pixel 3072 for class 1} \\ & & & \\ b_C & \text{weight for pixel 1 for class C} & & \text{weight for pixel 3072 for class C} \end{bmatrix}$$

Cx(n+1)

Let the score for class s_c be f(W,x)=W(c,:)x (b is included in W and x)

From 2 to multiple classes: Softmax

- The common generalization to multiple clasess is the **softmax classifier**.
- We want to predict the class label y_i={1,…C} for sample X(i,:), y can take one of C discrete values, so it follow a multinomial distribution.
- This is derived from an assumption that the probability of class y=k is

$$h_\theta(x) = p(y = k \mid x, \theta) = \frac{e^{\theta_k^T x}}{\sum_{j=1}^{C} e^{\theta_j^T x}}$$

- The score or loss function for class i is

$$L_i = -\log\left(\frac{e^{\theta_i^T X(i,:)}}{\sum_{j=1}^{k} e^{\theta_j^T X(i,:)}} \right)$$

This is called the cross-entropy loss

Cross-entropy

- From information theory, the cross entropy between a true distribution p and an estimated distribution q is:

$$H(p,q) = -\sum_x p(x) \log q(x)$$

- Softmax minimize the cross-entropy between the estimated class probabilities and the ‘true’ distribution (the distribution where all the mass is in the correct class).

Softmax

- From a training data set with m samples, we formulate the log-likelihood function that the model fits the data:

$$l(\theta) = \sum_{i=1}^{m} \log(p(y_i \mid X(i,:), \theta))$$

- We can now find θ that maximize the likelihood using e.g. gradient ascent of the log-likelihood function.
  – Or we can minimize –l(θ) using gradient descent

Loss and gradient descent for softmax

- The cost function for softmax, including regularization:

$x_i = X(i,:)^T$, the n pixel values for image i , let $\theta_j = W(j,:)$, the row for class j

$$J(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=1}^{C} I(y_i = j)\log\left(\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{C} e^{\theta_l^T x_i}}\right)\right] + \frac{\lambda}{2}\sum_{i=1}^{C}\sum_{j=0}^{n} W(i,j)^2$$

$$\nabla J_{\theta_j} = -\frac{1}{n}\sum_{i=1}^{n} x_i\left(I(y_i = j) - p(y_i = j \mid x_i, W)\right) + \lambda\theta_j$$

- I(y=j) is the indicator function that is 1 if y=j and zero otherwise.
- See http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression

Link to Gaussian classifiers

- In INF 4300, we used a traditional Gaussian classifier
  - This type of models is called generative models, where a specific distribution is assumed.

FROM INF 4300:Discriminant functions for the Gaussian density

- When finding the class with the highest probability, these functions are equivalent:

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})}$$

$$g_i(\mathbf{x}) = p(\mathbf{x} \mid \omega_i)P(\omega_i)$$

$$g_i(\mathbf{x}) = \ln p(\mathbf{x} \mid \omega_i) + \ln P(\omega_i)$$

- With a multivariate Gaussian we get:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t \Sigma_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln\left|\Sigma_i\right| + \ln P(\omega_i)$$

- If we assume all classes have equal diagonal covariance matrix, the discriminant function is a linear function of x:

$$\frac{1}{(\sigma^2)}\boldsymbol{\mu}_j^T\mathbf{x} - \frac{1}{2(\sigma^2)}\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j + \ln P(\omega_j)$$

Gaussian classifier vs. logistic regression

- These Gaussian with diagonal covariance and the logistic regression/softmax classifier will result in different linear decision boundaries.

- If the Gaussian assumption is correct, we will expect that this classifier has the lowest error rate.
- The logistic regresion might be better if the data is not entirely Gaussian.
- NOTE: SOFTMAX reduces to logistic regression if we have 2 classes.

Support Vector Machine classifiers

- Another popular classifier is the Support Vector Machine (SVM) formulation, which also can be formulated in terms of loss functions
- The following foils are for completeness, only a basic understand of the SVM as a maximum-margin classifier is expected in this course.

Background SVM

Background SVM

- If the two classes are linearly separable, there exist a hyperplane $w^{*T}x=0$ such that:

$$w^{*T} x > 0 \quad \forall x \in \omega_1$$
$$w^{*T} x < 0 \quad \forall x \in \omega_2$$

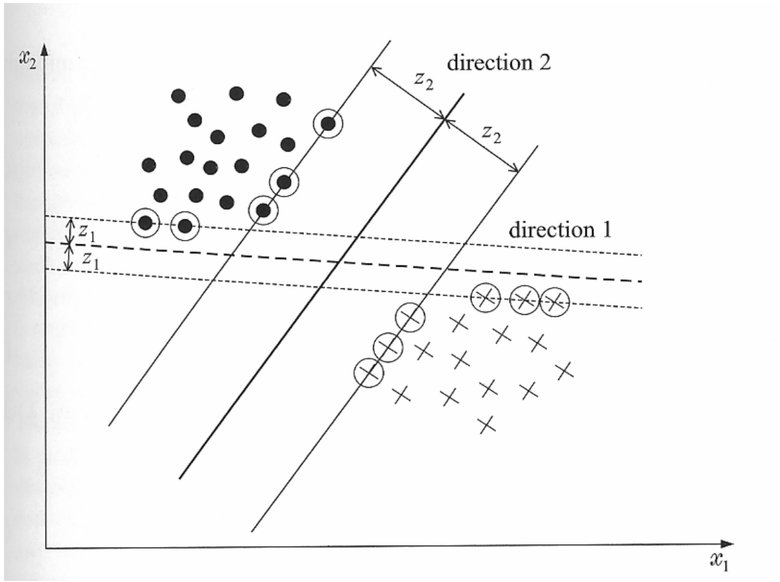
- The above also covers the situation where the hyperplane does not cross the origin, $w^{*T}x+w_0=0$, since this can reformulated as $x'=[x,1]^T$, $w'=[w^{*T},w_0]^T$. Then $w^{*T}x+w_0=w'^Tx'$.

Hyperplanes and margins

Background SVM

- A hyperplane is defined by its direction (w) and exact position (w_0).
- If both classes are equally probable, the **distance from the hyperplane to the closest points** in both classes should be equal. This is called the margin.
- The margin for direction 1 is $2z_1$, and for direction 2 it is $2z_2$.
- The distance from a point to a hyperplane is

$$z = \frac{|g(x)|}{\|w\|}$$



Hyperplanes and margins

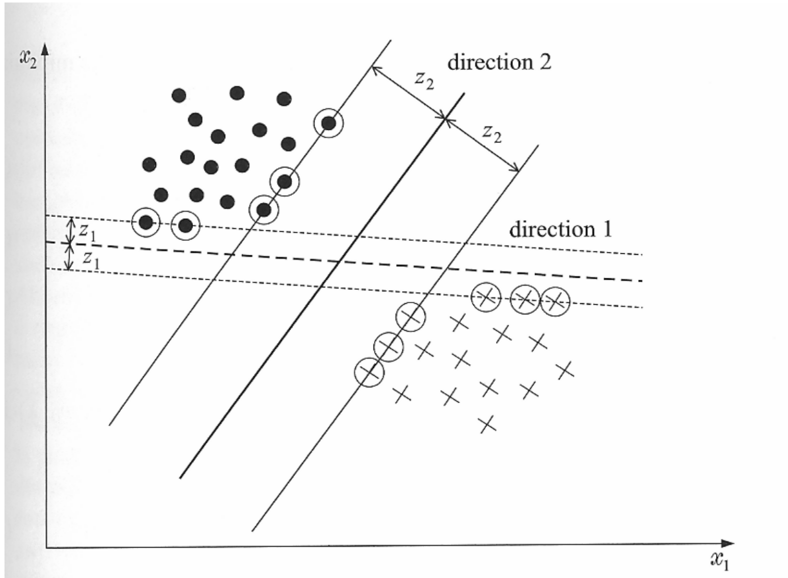
Background SVM

- We can scale w and w_0 such that g(x) will be equal to 1 at the closest points in the two classes. This is equivalent to:

1. Have a margin of $\frac{1}{\|w\|}+\frac{1}{\|w\|}=\frac{2}{\|w\|}$
2. Require that

$$w^T x + w_0 \geq 1, \quad \forall x \in \omega_1$$
$$w^T x + w_0 \leq -1, \quad \forall x \in \omega_2$$

- **Goal: find w and w_0**



Support Vector Machine loss

- A SVM loss function can be formulated by having as large margin as possible.
- This is generalized to multiple classes so the SVM 'wants' the correct class to have a score higher than the scores for the incorrect classes by som margin Δ
- If s_i is the score for class i, the loss function for SVM is

$$L_i = \sum_{j \neq i} \max(0, s_j - s_{y_i} + \Delta)$$

This is called the **hinge** loss

Background SVM

The optimization problem with margins

- The class indicator for pattern i, y_i, is defined as 1 if y_i belongs to class ω_1 and -1 if it belongs to ω_2.
- The best hyperplane with margin can be found by solving the optimization problem with respect to w and w_0 :

$$\text{minimize} \quad J(w) = \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad y_i(w^T x_i + w_0) \geq 1, \quad i = 1,2,\ldots N$$

The optimization problem with margins

$$\text{minimize} \quad J(w) = \frac{1}{2}\|w\|^2$$
$$\text{subject to} \quad y_i(w^T x_i + w_0) \geq 1, \quad i = 1,2,\ldots N$$

- This is a quadratic optimization task with a set of linear inequality contraints.
- It can be shown that the solution has the form:

$$\mathrm{w} = \sum_{i=1}^{N} \lambda_i y_i x_i \quad \text{where} \quad \sum_{i=1}^{N} \lambda_i y_i = 0$$

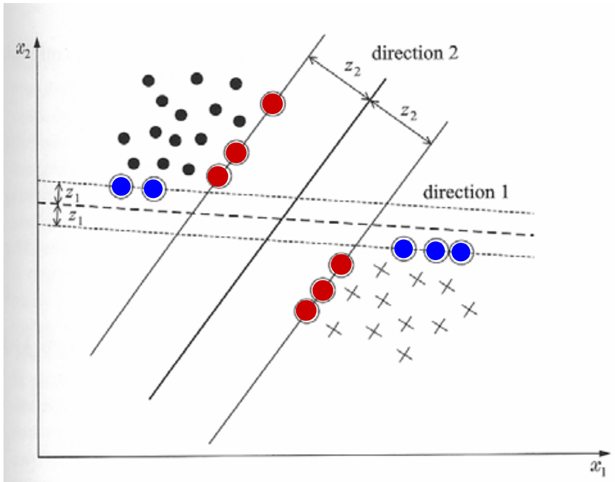
- The λ_i's are called Lagrange multipliers.
- The λ_i's can be either 0 or positive.
- We see that the solution w is a linear combination of $N_S \leq N$ feature vectors associated with a $\lambda_i > 0$.

Background SVM

- The feature vectors x_i with a corresponding $\lambda_i>0$ are called the support vectors for the problem.
- The classifier defined by this hyperplane is called a Support Vector Machine.
- Depending on y_i (+1 or -1), the support vectors will thus lie on either of the two hyperplanes

  $$w^Tx+w_0=\pm 1$$

- The support vectors are the points in the training set that are closest to the decision hyperplane.
- The optimization has a unique solution, only one hyperplane satisfies the conditions.



The support vectors for hyperplane 1 are the blue circles.
The support vectors for hyperplane 2 are the red circles.

Solving the optimization problem

Background SVM

- The optimization problem

$$\text{minimize} \quad J(w) = \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad y_i(w^T x_i + w_0) \geq 1, \quad i = 1,2,\ldots N$$

has a dual representation with equality constraints:

$$\text{maximize} \quad L(w, w_0, \lambda)$$

$$\text{subject to} \quad \mathrm{w} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \lambda_{\mathrm{i}} y_i x_i$$

$$\sum_{\mathrm{i}=1}^{\mathrm{N}} \lambda_{\mathrm{i}} y_i = 0 \quad \text{and} \quad \lambda_i \geq 0 \forall i$$

- This is easier to solve and can be reformulated as:

$$\max_{\lambda}\left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^T x_j\right)$$

$$\text{subject to} \quad \sum_{\mathrm{i}=1}^{\mathrm{N}} \lambda_{\mathrm{i}} y_i = 0 \quad \text{and} \quad \lambda_i \geq 0 \forall i$$

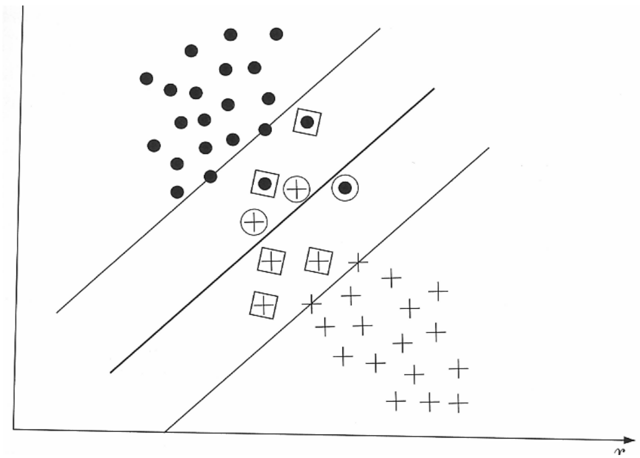
- Note that the training samples x_i and x_j occurr as inner products of pairs of feature vectors. **The solution does not depend on the dimensionality of the feature vector, only on the inner product.**
- The computational complexity can be expected to depend on the number of pixels in the training data set, N.
- In this setting we just accept that the solution can be found in optimization theory.

Background SVM

The nonseparable case

- If the two classes are nonseparable, a hyperplane satisfying the conditions $w^Tx-w_0=\pm 1$ cannot be found.
- The feature vectors in the training set are now either:

1. Vectors that fall outside the band and are correctly classified.
2. Vectors that are inside the band and are correctly classified. They satisfy $0 \leq y_i(w^Tx+w_0)<1$ □
3. Vectors that are misclassified – expressed as $y_i(w^Tx+w_0)<0$ ○



□ Correctly classified

○ Erroneously classified

Background SVM

- The three cases can be treated under a single type of contraints if we introduce slack variables ξ_i:

$$y_i[w^T x + w_0] \geq 1 - \xi_i$$

  – The first category (outside, correct classified) have ξ_i=0
  – The second category (inside, correct classified) have $0 \leq \xi_i \leq 1$
  – The third category (inside, misclassified) have $\xi_i > 1$

- The optimization goal is now *to keep the margin as large as possible and the number of points with $\xi_i > 0$ as small as possible.*

Cost function – nonseparable case

Background SVM

- The cost function to minimize is now

$$J(w, w_0, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

$$\text{where} \qquad I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases}$$

and ξ is the vector of parameters ξ_i.

- C is a parameter that controls how much misclassified training samples is weighted.
- We skip the mathematics and present the alternative dual formulation:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j x_i^T x_j\right)$$

$$\text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \quad \text{and} \quad 0 \le \lambda_i \le C \quad \forall i$$

- All points between the two hyperplanes ($\xi_i>0$) can be shown to have $\lambda_i=C$.

Background SVM

Nonseparable vs. separable case

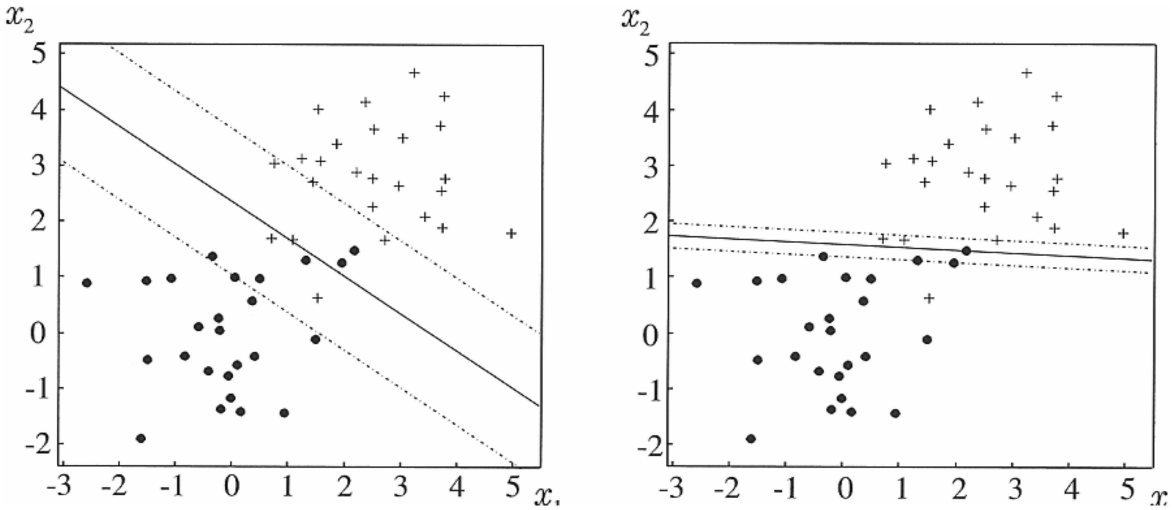
- Note that the slack variables ξ_i does not enter the problem explicitly.
- The only difference between the linear separable and the non-separable case is that the Lagrange-multipliers are bounded by C.
- Training a SVM classifier consists of solving the optimization problem.
  - The problem is quite complex since it grows with the number of training pixels.
  - It is computationally heavy.

Background SVM

An example – the effect of C

- C is the misclassification cost.

C=0.2



C=100

- Selecting too high C will give a classifier that fits the training data perfect, but fails on different data set.
- The value of C should be selected using a separate validation set. Separate the training data into a part used for training, train with different values of C and select the value that gives best results on the validation data set. Then apply this to new data or the test data set. (explained later)

SVM and gradient descent

- We can also solve the SVM using gradient descent also, we will not cover this, but see http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

SVMs: The nonlinear case

Background SVM

- We have now found a classifier that is not defined in terms of the class centres or the distributions, **but in terms of patterns close to the borders between classes, the support vectors.**
- It gives us a solution in terms of a hyperplane. This hyperplane can be expressed as a inner product between the training samples:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j x_i^T x_j\right)$$

$$\text{subject to} \quad \sum_{i=1}^{N}\lambda_i y_i = 0 \quad \text{and} \quad 0 \le \lambda_i \le C \quad \forall i$$

- The training samples are l-dimensional vectors.
- What if the classes overlap in l-dimensional space:
  - Can we find a mapping to a higher dimensional space, and use the SVM framework in this higher dimensional space?

Background SVM

- Assume that there exist a mapping from l-dimensional feature space to a k-dimensional space **(k>l**) :

$$x \in R^l \rightarrow y \in R^k$$

- Even if the feature vectors are not linearly separable in the input space, they might be separable in a higher dimensional space.
- Classification of a new pattern x is to be computed by computing the sign of

$$g(x) = w^T x + w_0$$
$$= \sum_{i=1}^{N_s} \lambda_i y_i x_i^T x_i + w_0$$

- In k-dimensional space, this involves the inner product between two k-dimensional vectors.
- Can it really help to go to a higher dimensional space?

An examle: from 2D to 3D

Background SVM

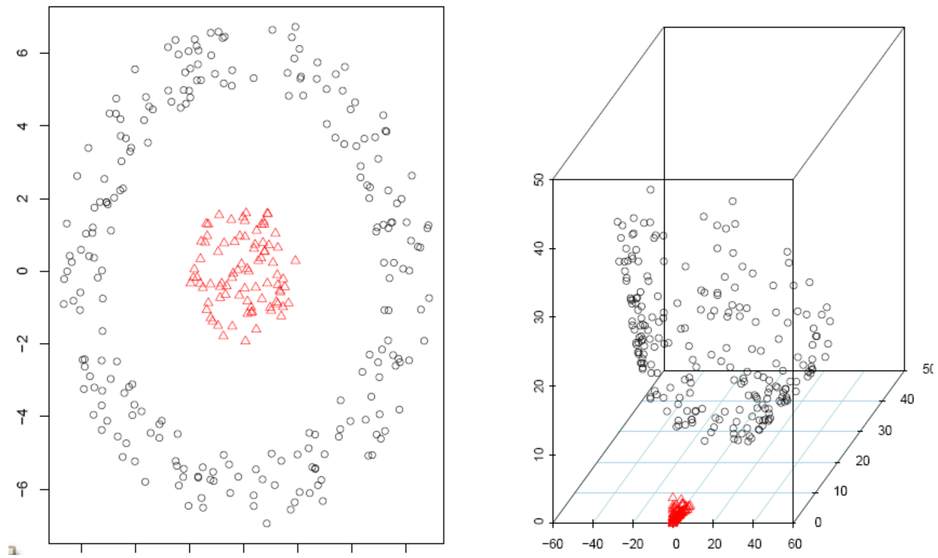
- Let *x* be a 1x2 vector $x=[x_1,x_2]$.
- Consider the transformation

$$y = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

- On the toy example, the two classes can not be linearly separated in the original 2D space.
- It can be shown that

$$y_i^T y_j = \left(x_i^T x_j\right)^2$$

- Given the transformation above, these points in a 3D space CAN actually be separated by a hyperplane.
- In 2D, we would need an ellipse to separate the classes.
- In 3D, this ellipse can be expressed as a linear function of y.



Remark: we don’t know yet how to construct this mapping or other useful mappings.

A useful trick: Mercer's theorem – finding a mapping to the high-dimensional space using a kernel

Background SVM

What we need from all this math is just that the inner product can be computed using the kernel K(x,z). Someone has also identified some useful kernels.

Assume that ϕ is a mapping:

$$x \to \phi(x) \in H$$

where H is an Euclidean space.

The inner product has an equivalent representation

$$\sum_r \phi_r(x)\phi_r(z) = K(x,z)$$

where $\phi_r(x)$ is the r-component of the mapping $\phi(x)$ of x, and K(x,z) is a symmetric function satisfying

$$\int K(x,z)g(x)g(z)dxdz \geq 0$$

for any g(x), $x \in R^l$ such that

$$\int g(x)^2 dx < +\infty$$

K(x,z) defines a inner product. K(x,z) is called a kernel. Once a kernel has been defined, a mapping to the higher dimensional space is defined.

Radial basis kernel for classification

- Radial basis function kernels (most commonly used)

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{\sigma^2} \right)$$

- The most common type of kernel is the radial basis function. It has an extra parameter σ that must be tuned.
- Use software packages like libsvm to solve.

The kernel formulation of the cost function

Background SVM

- Given the appropriate kernel (e.g. Radial with width σ) and the cost of misclassification C, the optimization task is:

$$\max_{\lambda}\left(\sum_i \lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j K(x_i, x_j)\right)$$

$$\text{subject to} \quad 0 \le \lambda_i \le C, \quad i = 1,....N$$

$$\sum_i \lambda_i y_i = 0$$

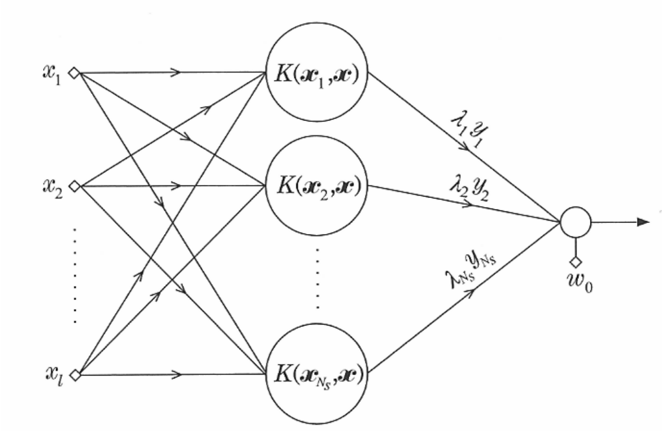
- The resulting classifier is:

$$\text{assign x to class } \omega_1 \text{ if } g(x) = \sum_{i=1}^{N}\lambda_i y_i K(x_i, x) + w0 > 0 \text{ and to class } \omega_2 \text{ otherwise}$$

Background SVM

SVM architecture

- Notice how the kernels are used to compute the inner product between all pairs of samples x_i in the training data set.



Link to Gaussian classifiers

- In INF 4300, we used a traditional Gaussian classifier
  – This type of models is called generative models, where a specific distribution is assumed.

FROM INF 4300:Discriminant functions for the Gaussian density

- When finding the class with the highest probability, these functions are equivalent:

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})}$$

$$g_i(\mathbf{x}) = p(\mathbf{x} \mid \omega_i)P(\omega_i)$$

$$g_i(\mathbf{x}) = \ln p(\mathbf{x} \mid \omega_i) + \ln P(\omega_i)$$

- With a multivariate Gaussian we get:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^t \Sigma_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i| + \ln P(\omega_i)$$

- If we assume all classes have equal diagonal covariance matrix, the discriminant function is a linear function of x:

$$\frac{1}{(\sigma^2)}\boldsymbol{\mu}_j^T\mathbf{x} - \frac{1}{2(\sigma^2)}\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j + \ln P(\omega_j)$$

Gaussian classifier vs. logistic regression

- These Gaussian with diagonal covariance and the logistic regression classifier will result in different linear decision boundaries.
- If the Gaussian assumption is correct, we will expect that this classifier has the lowest error rate.
- The logistic regresion might be better if the data is not entirely Gaussian.

Learning goals

- Understand logistic regression and the loss function for binary classification.
- Have knowledge about Softmax classification.
- Know what Support Vector Machines optimize and recognize the loss function in the linear case (without the kernel trick).
- Understand the need for regularization, and how to incorporate this in the loss function.

Next two weeks:

- Next week: Image representation/feature extraction
- In 2-3 weeks: basic neural nets
  - Reading material:
    - http://cs231n.github.io/neural-networks-1/
    - http://cs231n.github.io/neural-networks-2/
    - http://cs231n.github.io/optimization-2/
    - Deep learning Chapter 6