CSC2515 Fall 2007
Introduction to Machine Learning

Lecture 2: Linear regression

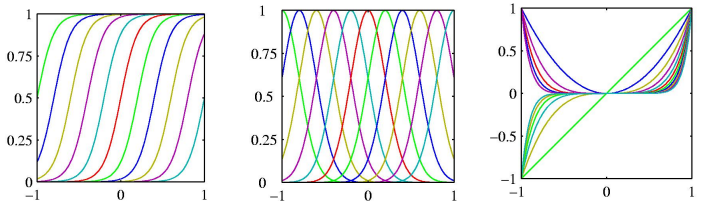
All lecture slides will be available as .ppt, .ps, & .htm at
www.cs.toronto.edu/~hinton

Many of the figures are provided by Chris Bishop
from his textbook: "Pattern Recognition and Machine Learning"

Linear models

- It is mathematically easy to fit linear models to data.
  - We can learn a lot about model-fitting in this relatively simple case.
- There are many ways to make linear models more powerful while retaining their nice mathematical properties:
  - By using non-linear, non-adaptive basis functions, we can get generalised linear models that learn non-linear mappings from input to output but are linear in their parameters – only the linear part of the model learns.
  - By using kernel methods we can handle expansions of the raw data that use a huge number of non-linear, non-adaptive basis functions.
  - By using large margin kernel methods we can avoid overfitting even when we use huge numbers of basis functions.
- But linear methods will not solve most AI problems.
  - They have fundamental limitations.

Some types of basis function in 1-D



Sigmoids Gaussians Polynomials

Sigmoid and Gaussian basis functions can also be used in multilayer neural networks, but neural networks learn the parameters of the basis functions. This is much more powerful but also much harder and much messier.

Two types of linear model that are equivalent with respect to learning

bias

$$y(\mathbf{x},\mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots = \mathbf{w}^T \mathbf{x}$$

$$y(\mathbf{x},\mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \ldots = \mathbf{w}^T \Phi(\mathbf{x})$$

- The first model has the same number of adaptive coefficients as the dimensionality of the data +1.
- The second model has the same number of adaptive coefficients as the number of basis functions +1.
- Once we have replaced the data by the outputs of the basis functions, fitting the second model is exactly the same problem as fitting the first model (unless we use the kernel trick)
  - So its silly to clutter up the math with basis functions

The loss function

- Fitting a model to data is typically done by finding the parameter values that minimize some loss function.
- There are many possible loss functions. What criterion should we use for choosing one?
  - Choose one that makes the math easy (squared error)
  - Choose one that makes the fitting correspond to maximizing the likelihood of the training data given some noise model for the observed outputs.
  - Choose one that makes it easy to interpret the learned coefficients (easy if mostly zeros)
  - Choose one that corresponds to the real loss on a practical application (losses are often asymmetric)

Minimizing squared error

$$y = \mathbf{w}^T \mathbf{x}$$

$$error = \sum_n (t_n - \mathbf{w}^T \mathbf{x}_n)^2$$

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

vector of target values

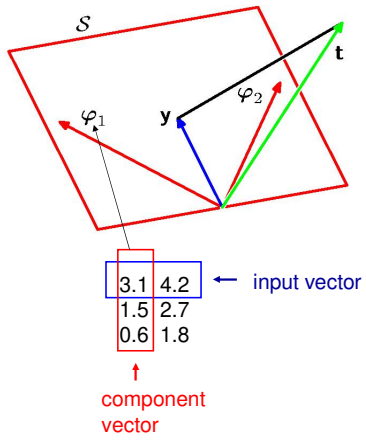
optimal weights

inverse of the covariance matrix of the input vectors

the transposed design matrix has one input vector per column

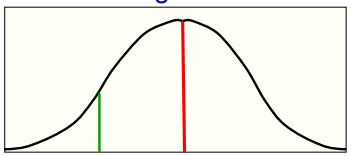
A geometrical view of the solution

- The space has one axis for each training case.
- So the vector of target values is a point in the space.
- Each vector of the values of one component of the input is also a point in this space.
- The input component vectors span a subspace, S.
  - A weighted sum of the input component vectors must lie in S.
- The optimal solution is the orthogonal projection of the vector of target values onto S.



When is minimizing the squared error equivalent to Maximum Likelihood Learning?

- Minimizing the squared residuals is equivalent to maximizing the log probability of the correct answer under a Gaussian centered at the model's guess.



t = the correct answer

y = model's estimate of most probable value

$$y_n = y(\mathbf{x}_n, \mathbf{w})$$

$$p(t_n \mid y_n) = p(y_n + noise = t_n \mid \mathbf{x}_n, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t_n - y_n)^2}{2\sigma^2}}$$

$$-\log p(t_n \mid y_n) = \log \sqrt{2\pi} + \log \sigma + \frac{(t_n - y_n)^2}{2\sigma^2}$$

can be ignored if sigma is fixed

can be ignored if sigma is same for every case

Multiple outputs

- If there are multiple outputs we can often treat the learning problem as a set of independent problems, one per output.
  - Not true if the output noise is correlated and changes from case to case.
- Even though they are independent problems we can save work by only multiplying the input vectors by the inverse covariance of the input components once. For output k we have:

$$\mathbf{w}_k^* = \boxed{(\mathbf{X}^T\ \mathbf{X})^{-1}\ \mathbf{X}^T}\mathbf{t}_k$$

does not depend on a

Least mean squares: An alternative approach for really big datasets

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \boxed{\nabla E_{n(\tau)}}$$

weights after seeing training case tau+1

learning rate

vector of derivatives of the squared error w.r.t. the weights on the training case presented at time tau.

- This is called "online" learning. It can be more efficient if the dataset is very redundant and it is simple to implement in hardware.
  - It is also called stochastic gradient descent if the training cases are picked at random.
  - Care must be taken with the learning rate to prevent divergent oscillations, and the rate must decrease at the end to get a good fit.

Regularized least squares

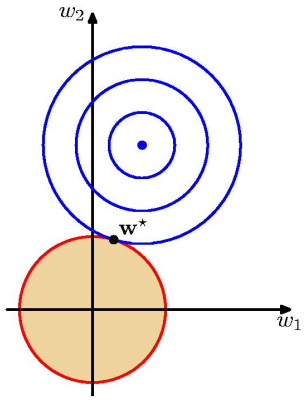
$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

The penalty on the squared weights is mathematically compatible with the squared error function, so we get a nice closed form for the optimal weights with this regularizer:

$$\mathbf{w}^* = (\lambda \mathbf{I} + \mathbf{X}^T\ \mathbf{X})^{-1}\ \mathbf{X}^T\ \mathbf{t}$$

identity matrix

A picture of the effect of the regularizer



- The overall cost function is the sum of two parabolic bowls.
- The sum is also a parabolic bowl.
- The combined minimum lies on the line between the minimum of the squared error and the origin.
- The regularizer just shrinks the weights.

A problem with the regularizer

- We would like the solution we find to be independent of the units we use to measure the components of the input vector.
- If different components have different units (e.g. age and height), we have a problem.
  - If we measure age in months and height in meters, the relative values of the two weights are very different than if we use years and millemeters. So the squared penalty has very different effects.
- One way to avoid the units problem: Whiten the data so that the input components all have unit variance and no covariance. This stops the regularizer from being applied to the whitening matrix.

$$\mathbf{X}^T_{whitened} = (\mathbf{X}^T \mathbf{X})^{-\frac{1}{2}} \mathbf{X}^T$$

  - But this can cause other problems when two input components are almost perfectly correlated.
  - We really need a prior on the weight on each input component.

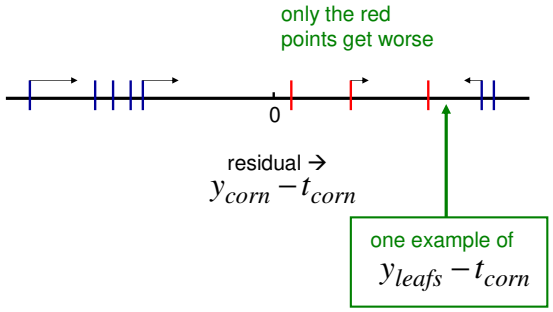
Why does shrinkage help?

- Suppose you have an unbiased estimator for the price of corn and an unbiased estimator for the number of fouls committed by the leafs.
- You can improve each estimate by taking a weighted average with the other:

$$\tilde{y}_{corn} = (1-\varepsilon)\ y_{corn} + \varepsilon\ y_{leafs}$$

- For some positive epsilon, this estimate will have a smaller squared error (but it will be biased).

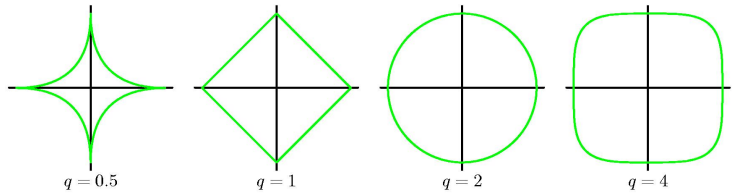
Why shrinkage helps



If we move all the blue residuals towards the green arrow by an amount proportional to their difference, we are bound to reduce the average squared magnitudes of the residuals. So if we pick a blue point at random, we reduce the expected residual.

Other regularizers

- We do not need to use the squared error, provided we are willing to do more computation.
- Other powers of the weights can be used.

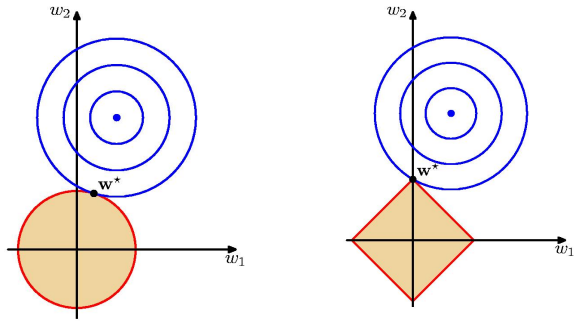


The lasso: penalizing the absolute values of the weights

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \lambda \sum_i |\mathbf{w}_i|$$

- Finding the minimum requires quadratic programming but its still unique because the cost function is convex (a bowl plus an inverted pyramid)
- As lambda is increased, many of the weights go to exactly zero.
  - This is great for interpretation, and it is also pretty good for preventing overfitting.

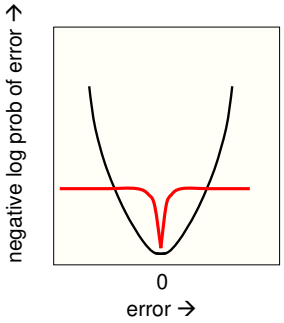
A geometrical view of the lasso compared with a penalty on the squared weights



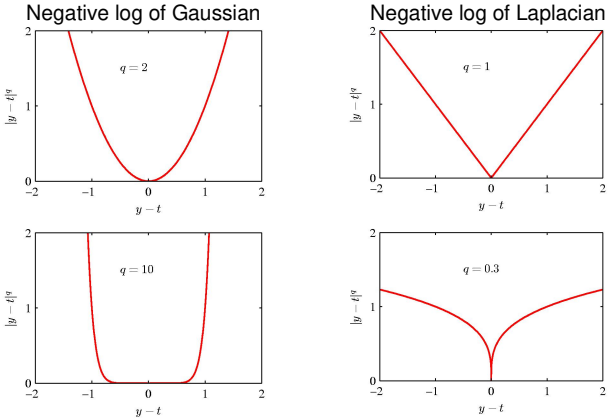
Notice that w1=0 at the optimum

An example where minimizing the squared error gives terrible estimates

- Suppose we have a network of 500 computers and they all have slightly imperfect clocks.
- After doing statistics 101 we decide to improve the clocks by averaging all the times to get a least squares estimate
  - Then we broadcast the average to all of the clocks.
- Problem: The probability of being wrong by ten hours is more than one hundredth of the probability of being wrong by one hour. In fact, its about the same!



One dimensional cross-sections of loss functions with different powers



Minimizing the absolute error

$$\min_{over\ \mathbf{w}} \sum_n |\, t_n - \mathbf{w}^T \mathbf{x}_n \,|$$

- This minimization involves solving a linear programming problem.
- It corresponds to maximum likelihood estimation if the output noise is modeled by a Laplacian instead of a Gaussian.

$$p(t_n \mid y_n) = a\, e^{-a|t_n - y_n|}$$

$$-\log p(t_n \mid y_n) = -a\,|t_n - y_n| + const$$

The bias-variance trade-off
(a figment of the frequentists lack of imagination?)

- Imagine that the training set was drawn at random from a whole set of training sets.
- The squared loss can be decomposed into a "bias" term and a "variance" term.
  - Bias = systematic error in the model's estimates
  - Variance = noise in the estimates cause by sampling noise in the training set.
- There is also an additional loss due to the fact that the target values are noisy.
  - We eliminate this extra, irreducible loss from the math by using the average target values (i.e. the unknown, noise-free values)

The bias-variance decomposition

model's estimate for test case n when trained on dataset D

average target value for test case n

The "bias" term is the squared error of the average, over all training datasets, of the estimates.

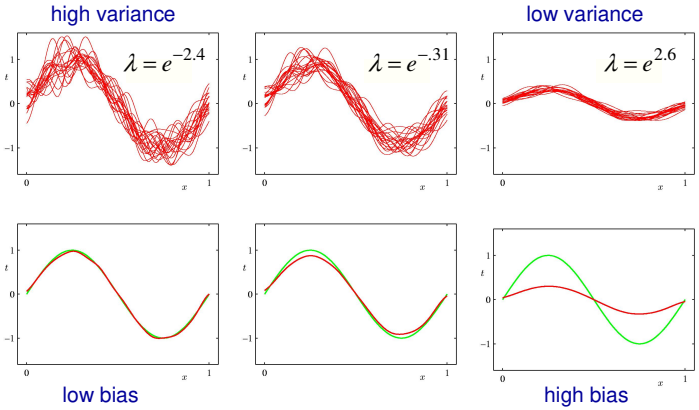
$$\left\langle \{y(\mathbf{x}_n; D) - \bar{t}_n\}^2 \right\rangle_D = \boxed{\{\langle y(\mathbf{x}_n; D)\rangle_D - \bar{t}_n\}^2} + \boxed{\left\langle \{y(\mathbf{x}_n; D) - < y(\mathbf{x}_n; D) >_D\}^2 \right\rangle_D}$$

angle brackets are physics notation for expectation over D

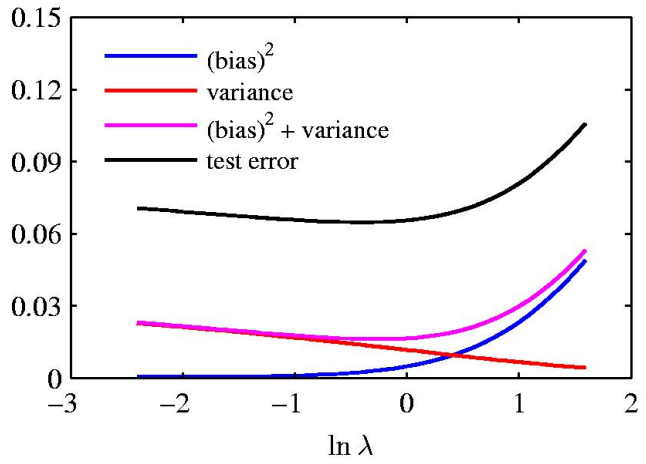
The "variance" term is the variance, over all training datasets, of the model's estimate.

see Bishop page 149 for a derivation using a different notation

How the regularization parameter affects the bias and variance terms



An example of the bias-variance trade-off



Beating the bias-variance trade-off

- We can reduce the variance term by averaging lots of models trained on different datasets.
  - This seems silly. If we had lots of different datasets it would be better to combine them into one big training set.
    - With more training data there will be much less variance.
- Weird idea: We can create different datasets by bootstrap sampling of our single training dataset.
  - This is called "bagging" and it works surprisingly well.
- But if we have enough computation its better to do the right Bayesian thing:
  - Combine the predictions of many models using the posterior probability of each parameter vector as the combination weight.

The Bayesian approach

- Consider a very simple linear model that only has two parameters:

$$y(x, \mathbf{w}) = w_0 + w_1 x$$

- It is possible to display the full posterior distribution over the two-dimensional parameter space.
- The likelihood term is a Gaussian, so if we use a Gaussian prior the posterior will be Gaussian:
  - This is a conjugate prior. It means that the prior is just like having already observed some data.

Gaussian

variance of output noise

$$p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathrm{N}(t_n \mid \mathbf{w}^T \mathbf{x}_n, \beta^{-1})$$

← likelihood

$$p(\mathbf{w} \mid \alpha) = \mathrm{N}(\mathbf{w} \mid 0, \alpha^{-1}\mathbf{I})$$

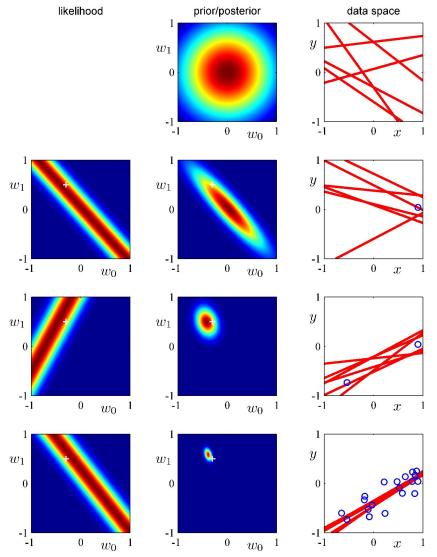
← conjugate prior

inverse variance of prior

$$-\ln p(\mathbf{w} \mid \mathbf{t}) = \frac{\beta}{2} \sum_{n=1}^{N} (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + const$$

The Bayesian interpretation of the regularization parameter:

$$\lambda = \frac{\alpha}{\beta}$$



- With no data we sample lines from the prior.
- With 20 data points, the prior has little effect

Using the posterior distribution

- If we can afford the computation, we ought to average the predictions of all parameter settings using the posterior distribution to weight the predictions:

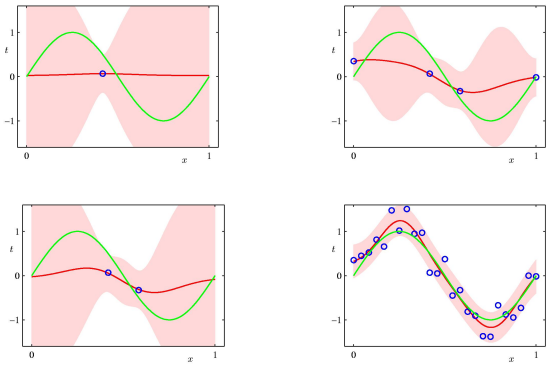
$$p(t_{test} \mid x_{test}, \alpha, \beta, D) = \int p(t_{test} \mid x_{test}, \beta, \mathbf{w})\ p(\mathbf{w} \mid \alpha, \beta, D)\, d\mathbf{w}$$

training data

precision of output noise

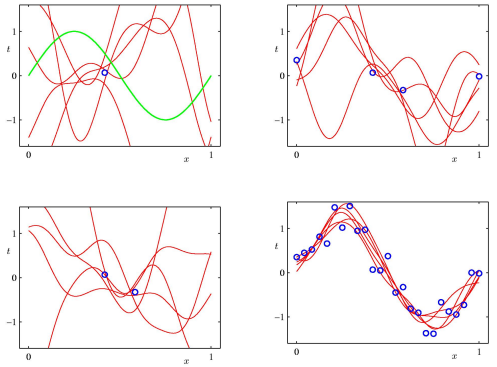
precision of prior

The predictive distribution for noisy sinusoidal data modeled by a linear combination of nine radial basis functions.



A way to see the covariance of the predictions for different values of x

We sample models at random from the posterior and show the mean of the each model's predictions



Bayesian model comparison

- We usually need to decide between many different models:
  - Different numbers of basis functions
  - Different types of basis functions
  - Different strengths of regularizers
- The frequentist way to decide between models is to hold back a validation set and pick the model that does best on the validation data.
  - This gives less training data. We can use a small validation set and evaluate models by training many different times using different small validation sets. But this is tedious.
- The Bayesian alternative is to use all of the data for training each model and to use the "evidence" to pick the best model (or to average over models).
- The evidence is the marginal likelihood with the parameters integrated out.

Definition of the evidence

- The evidence is the normalizing term in the expression for the posterior probability of a weight vector given a dataset and a model class

$$p(\mathbf{w} \mid D, M_i) = \frac{p(\mathbf{w} \mid M_i)\, p(D \mid \mathbf{w}, M_i)}{p(D \mid M_i)}$$

$$p(D \mid M_i) = \int p(D \mid \mathbf{w}, M_i)\; p(\mathbf{w} \mid M_i)\, d\mathbf{w}$$

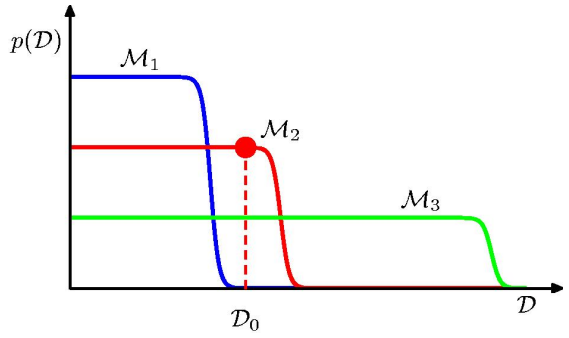
Using the evidence

- Now we use the evidence for a model class in exactly the same way as we use the likelihood term for a particular setting of the parameters
  - The evidence gives us a posterior distribution over model classes, provided we have a prior.

$$p(M_i \mid D) \propto p(M_i)\, p(D \mid M_i)$$

  - For simplicity in making predictions we often just pick the model class with the highest posterior probability. This is called model selection.
    - But we should still average over the parameter vectors for that model class using the posterior distribution.

How the model complexity affects the evidence



Increasingly complicated data →

Determining the hyperparameters that specify the variance of the prior and the variance of the output noise.

- Ideally, when making a prediction, we would like to integrate out the hyperparameters, just like we integrate out the weights
  - But this is infeasible even when everything is Gaussian.
- Empirical Bayes (also called the evidence approximation) means integrating out the parameters but maximizing over the hyperparameters.
  - Its more feasible and often works well.
  - It creates ideological disputes.

Empirical Bayes

target and input on test case; precision of output noise; precision of prior; training data

$$p(t \mid \mathbf{x}, D) = \int \int \int p(t \mid \mathbf{x}, \beta, \mathbf{w}) \; p(\mathbf{w} \mid \alpha, \beta, D) \; p(\alpha, \beta \mid D) \, d\mathbf{w} \, d\alpha \, d\beta$$

- The equation above is the right predictive distribution (assuming we do not have hyperpriors for alpha and beta).
- The equation below is a more tractable approximation that works well if the posterior distributions for alpha and beta are highly peaked (so the distributions are well approximated by their most likely values)

point estimates of alpha and beta that maximize the evidence

$$p(t \mid \mathbf{x}, D) \approx p(t \mid \mathbf{x}, D, \hat{\alpha}, \hat{\beta}) = \int p(t \mid \mathbf{x}, \hat{\beta}, \mathbf{w}) \; p(\mathbf{w} \mid \hat{\alpha}, \hat{\beta}, D) \; p(\hat{\alpha}, \hat{\beta} \mid D) \, d\mathbf{w} \, d\alpha \, d\beta$$