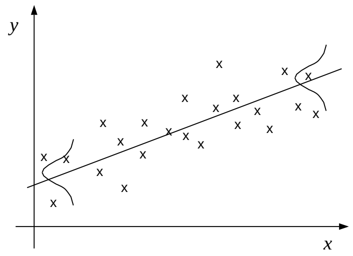
Lecture 4:

Regression I

October 4, 2005

Reminder: Regression

- Multiple inputs $\mathbf{x}$, mixed cts. and discrete.
- Continuous output(s) $\mathbf{y}$. (Consider each separately.)
- Goal: predict output on future unseen inputs.
- Still conditional density estimation: $p(\mathbf{y}|\mathbf{x})$ (c.f. classification)
- For now, consider continuous inputs and a single output...



Constant Model

- Constant model says $y = a$, independent of $\mathbf{x}$.
  (What is the constant model in classification?)
- Q: What should we use for a?
  The mean? The median? The mode (for quantized data)?
- A: Depends on your error function (noise model).
- For squared error, the mean is best:

$$
\begin{aligned}
e &= \sum_{n=1}^{N} (y_n - a)^2 \\
\frac{de}{da} &= 2 \sum_n (y_n - a) \\
a^* &= \frac{1}{N} \sum_n y_n
\end{aligned}
$$

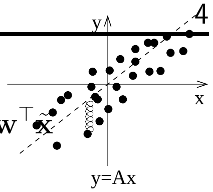
Absolute Error

- For abs error, we get the median:

$$
\begin{aligned}
e &= \sum_{n=1}^{N} |y_n - a| \\
\frac{de}{da} &= \sum_n \mathrm{sign}[a - y_n] \\
&= (\#y_n\text{smaller than a}) - (\#y_n\text{bigger than a}) \\
a^* &= \mathrm{median}[y_1 \ldots y_N]
\end{aligned}
$$

- What if there are an even number of datapoints?
  Or exact duplicates?
  Then any value between the middle two gives the same error.
- Even for constant model life is not so simple...

Linear Regression



- Linear model:

$$y = \sum w_i x_i + w_0 = \mathbf{w}^\top \mathbf{x} + w_0 = \mathbf{w}^\top \tilde{\mathbf{x}}$$

- Geometry: a line or hyperplane.
  The bias term w_0 offsets the line
  (hyperplane) from the origin.
  Think of vertical springs
  connecting y_n to line $\mathbf{w}^\top \mathbf{x}$.
  Goal: minimze energy.
- Usually augment $\mathbf{x}$ with constant and absorb bias into $\mathbf{w}$.
- Q: What's the best $\mathbf{w}$?
- A: As you know, it depends on your cost function!

(The constant model corresponds to $w_0 = a, w_i = 0$.)

Ordinary Least Squares

- For squared error, the problem is *linear least squares*, or *ordinary least squares*, and we can get a direct solution:

$$\begin{aligned} y &= \mathbf{w}^\top \mathbf{x} \\ e &= \sum_n (y_n - \mathbf{w}^\top \mathbf{x}_n)^2 \\ \mathbf{w}^* &= (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{X} \mathbf{y}^\top \end{aligned}$$

- $\mathbf{X}$ is matrix of inputs (one per column); $\mathbf{y}$ is output row vector.
- This is one of the most famous equations in all of linear algebra: the *discrete Weiner filter*.
- It says: take the correlation between inputs and outputs, but don't be fooled by large input-input correlations.
- Predicted values are $\hat{y}_n = \mathbf{y}\mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{x}_n$.

Probabilistic Model for Least Squares

- What probabilistic model corresponds to squared error? Gaussian:

$$\begin{aligned} p(y|\mathbf{x}, \mathbf{w}) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x})^2} \\ \log p(y|\mathbf{x}, \mathbf{w}) &= -\frac{1}{2}(y - \mathbf{w}^\top \mathbf{x})^2 + \text{const} \\ \log p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \sum_n \log p(y_n|\mathbf{x}_n, \mathbf{w}) \qquad \text{for iid data} \end{aligned}$$

- So minimzing squared error $\equiv$ maximum Gaussian noise likelihood
- What if we have multiple outputs (y is a vector)?
  Turns out we can just treat each one as a separate regression problem, even if the output noise is correlated.
  (However, if the output noises are correlated *and* the noise changes from case to case, then the solutions become coupled.)

Error Bars on Predictions

- Can we estimate the noise level? Yes. An unbiased estimate is:

$$\sigma^2 \approx \frac{1}{N - d - 1} \sum_n (y_n - \mathbf{w}^\top \mathbf{x})^2$$

- What about the variance of parameters? Yes, also:

$$\text{var}[\mathbf{w}] = (\mathbf{X}\mathbf{X}^\top)^{-1} \sigma^2$$

- This allows us to put some crude error bars on our predictions:
  $p(\hat{y}|\mathbf{x})$ is Gaussian with
  mean=$\mathbf{w}^\top \mathbf{x}$
  variance= $\mathbf{w}^\top (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{w} + \sigma^2$
- There is uncertainty in the prediction both from the output noise and from our uncertainty about our estimated parameters.

Absolute Error Linear Regression

- What if we use absolute error with the linear model? What's the equivalent of the median estimator?

$$\min_{\mathbf{w}} \sum_n |y_n - \mathbf{w}^\top \mathbf{x}_n|$$

- We need to solve a *linear programming problem*:

$$\min \sum_n t_n$$
$$\text{subject to} \qquad -t_n \leq y_n - \mathbf{w}^\top \mathbf{x}_n \leq t_n$$

- What probabilistic model corresponds to absolute error? Laplacian:

$$p(y|\mathbf{x}, \mathbf{w}) = a e^{-a|y - \mathbf{w}^\top \mathbf{x}|}$$
$$\log p(y|\mathbf{x}, \mathbf{w}) = -a|y - \mathbf{w}^\top \mathbf{x}| + \mathrm{const}$$

Regularization

- What? You thought the linear model was simple enough that we don't need to regularlize it? Everything needs regularization!
- Example 1: you have fewer training cases than input dimensions. Now $\mathbf{X}\mathbf{X}^\top$ will not be invertible.
- Example 2: certain input dimensions are useless (on average) at predicting the output. But because of noise or small samples, you can always reduce the training error a tiny bit by putting huge weights on these dimensions. At test time you get killed.
- Two common solutions:
  - input subset selection
  - parameter shrinkage

Subset Selection

- Use only a few x_i as inputs, discard the rest. Advantages: introduces inductive bias, produces small models. Disadvantage: high variability because of binary choices.
- Forward stepwise selection: start with constant and iteratively add the single x_i which most decreases error.
- Backward stepwise selection: start with all inputs and iteratively remove the single x_i which least increases error.
- Leaps-and-Bounds: Furnival and Wilson (74) came up with a very clever branch-and-bound trick for efficiently trying *all possible* subsets. Works for up to ≈ 40 variables.
- Choose subset size with cross validation or F-statistic tests.

Shrinkage/Ridge Regression

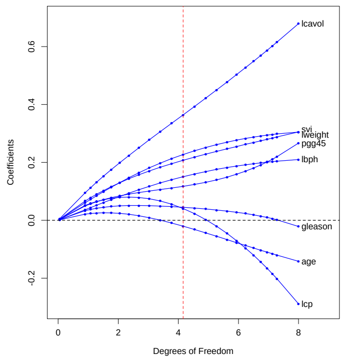
- Idea: pull ("shrink") estimated parameters towards some fixed values that do not depend on the data. ("Stein's paradox".)
- Usually we shrink towards zero (but sometimes towards the mean of some other set of weights).
- Shrinking to zero: penalize coefficients based on their size.
- For a penalty which is the sum of the squares of the weights, this is known as "weight decay" or "ridge regression":

$$y = \mathbf{w}^\top \mathbf{x}$$
$$e = \sum_n (y_n - \mathbf{w}^\top \mathbf{x})^2 + \lambda \sum_i w_i^2$$
$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^\top + \lambda I)^{-1} \mathbf{X}\mathbf{y}^\top$$

  where I is the identity matrix.
- We used the same trick when we were training Gaussian class-conditional classifiers.

Ridge Regression Practicalities

- Set λ with cross-validation. (There is a trick which lets you compute the leave-one-out error very efficiently without refitting N times. See "generalized cross-validation".)
- Don't shrink the constant (bias) term!



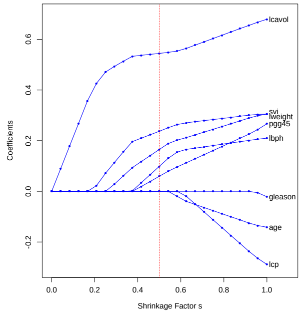
Warning: ridge regression is *not* invariant to input rescaling.
Often we want to "whiten/sphere" inputs first (i.e. rescale them so their sample covariance is a multiple of the identity matrix).

Lasso

- Shrinkage has less variance but doesn't give sparse models like subset selection does. Can we get the best of both worlds?
- Lasso: squared error with absolute weight penalty.

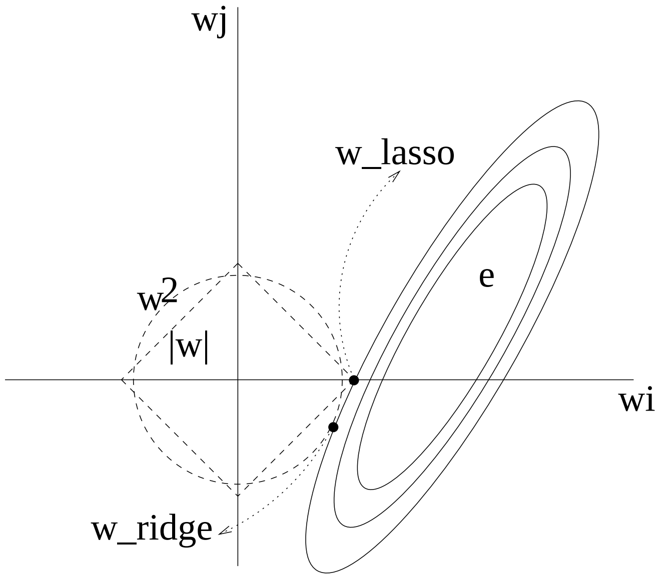
$$e = \sum_n (y_n - \mathbf{w}^\top \mathbf{x})^2 + \lambda \sum_i |w_i|$$

- Requires quadratic programming to solve, but still unique optimum.



A very cool thing happens. As you increase the lasso penalty many coefficients go exactly to zero. Why is that? Geometry!

Error Function Geometry



Beyond Linear Regression

- We can augment the inputs, not just with a constant to get a bias term, but with lots of other things.
- If we decide beforehand on how to augment the input, this is still linear regression:

$$y = \sum_j w_j h_j(\mathbf{x})$$

- For linear regression, just use $h_0 = 1, h_j = x_j$.
- Optimal weights are still easy to find:

$$e = \sum_n (y_n - \mathbf{w}^\top \mathbf{h}(\mathbf{x}))^2$$

$$\mathbf{w}^* = (\mathbf{H}\mathbf{H}^\top)^{-1}\mathbf{H}\mathbf{y}^\top$$

where $\mathbf{h}(\mathbf{x})$ is a vector of basis function outputs and $\mathbf{H}$ is a matrix with columns $\mathbf{h}(\mathbf{x}_n)$.

Generalized Linear Models

- Any fixed, generalized basis set that depends on the inputs can be used to make a *generalized linear model*.
- Elements of basis are called *dictionary functions*.
- Examples include splines, radial basis functions, wavelets, etc.
- Common things to add: quadratic or other polynomial terms, sinusoidal terms, exponentials, square roots, logarithms, etc.
- Terms can depend on more than one input e.g.
  $h_j(\mathbf{x}) = x_2 x_8 x_9$
  $h_j(\mathbf{x}) = \|\mathbf{x}\|^2$
  $h_j(\mathbf{x}) = [a \leq x_i \leq b][c \leq x_j \leq d]$
- These models can also be use for classification:
  as inputs to logistic/softmax regression, as a space for Fisher discriminants/Gaussians class-conditionals, KNN, etc.
- This is a simple version of the "kernel" idea (hopefully more later).

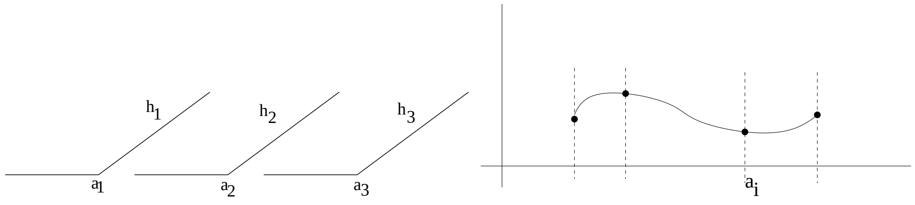
Splines

- You can construct a special basis set that gives piecewise constant functions between pre-specified split points ("knots") a_i:

  $h_1 = (x{-}a_1)_+ \quad h_2 = (x{-}a_2)_+ \ldots h_k = (x{-}a_k)_+ \quad h_{k+1} = x \quad h_{k+2} = 1$

  where $(x - a_i)_+$ is the positive part of $(x - a_i)$.
- To enforce continuity up to the $(r-1)^{st}$ derivative, use

  $h_1 = (x - a_1)^r_+ \ldots h_k = (x - a_k)^r_+ \quad h_{k+1} = x^r \ldots h_{k+r+1} = 1$
- Most common: *cubic splines*, corresponding to $r = 3$.
- Can also enforce linearity beyond edges: *natural cubic spline*.

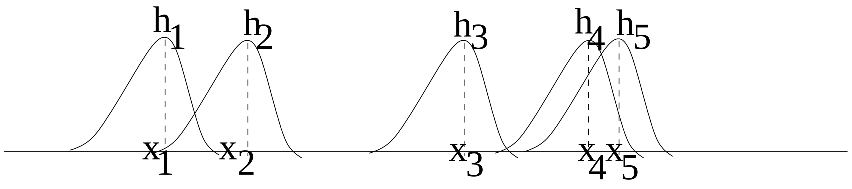


Radial Basis Functions

- One way to generate a nice automatic basis is to place a dictionary element on each input datapoint, whose value depends on the distance of the input from the point it is on top of:

$$h_n(\mathbf{x}) = \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{x}_n\|^2\right]$$

- Tricky part is setting σ^2.



Neural Networks

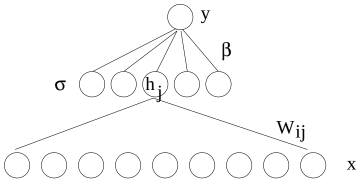
- Another generalized linear model, this time with the basis set:

$$h_j = g\left(\sum_i w_{ij} x_i\right)$$

  with $g()$ a "squashing" function with limited outputs, e.g.

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

- The outputs h_j are known as the "hidden layer".



Learning the Basis

- In all the examples above, the basis functions were fixed.
- Ideally, we'd like to adjust the basis set also.
  E.g. where are the knots for splines, the centres for RBF's, what are the input-to-hidden weights for neural networks?
- Three strategies: (1) shrinkage, (2) subset selection, (3) adaptive.

1. We can do *ridge regression* on a large generalized basis set, by penalizing the coefficients. For splines, this technique gives *smoothing splines*.
2. We can also start with a huge dictionary and try to pick a few elements. This is *subset selection* from a broader choice set.
3. Lastly, we can have a fixed number of *adaptive* elements. Next lecture we can see how to do this, for certain error functions, using *gradient descent*. But the solutions are no longer optimal.

Multivariate Adaptive Regression Splines (MARS)

- Piecewise constant $1D$ splines with knots at each data point value in each dimension. Also the "reflected pairs".

$$h_{ni}(\mathbf{x}) = (x_i - x_{ni})_+ \qquad h_{2ni}(\mathbf{x}) = (x_{ni} - x_i)_+$$

- Now use forward stepwise regression, chosing from these basic elements *and* any product between them and an existing dictionary element.
- Then do backwards deletion.
- Another Stanford masterpiece. (What's in the water in Palo Alto?)

Things I won't cover

- Regression trees (very similar to MARS but worse).
- Partial Least Squares
- Empirical Bayes (ML-II)
  Automatic Relevance Determination (ARD)
- Canonical Correlation Analysis

More reading: Hastie et al. Ch4,5,9.4