10 Regression, including Least-Squares Linear and Logistic Regression

REGRESSION aka Fitting Curves to Data

Classification: given point x, predict class (often binary)
Regression: given point x, predict a numerical value

[Classification gives a discrete prediction, whereas regression gives us a quantitative prediction, usually on a continuous scale.]
[We've already seen an example of regression in Gaussian discriminant analysis. QDA and LDA don't just give us a classifier; they also give us the probability that a particular class label is correct. So QDA and LDA implicitly do regression on probability values.]

- Choose form of regression fn $h(x; p)$ with parameters p (h = hypothesis)
  - like decision fn in classification [e.g. linear, quadratic, logistic in x]
- Choose a cost fn (objective fn) to optimize
  - usually based on a loss fn; e.g. risk fn = expected loss

Some regression fns:
(1) linear: $h(x; w, \alpha) = w \cdot x + \alpha$
(2) polynomial [equivalent to linear regression with added polynomial features]
(3) logistic: $h(x; w, \alpha) = s(w \cdot x + \alpha)$ recall: logistic fn $s(\gamma) = \frac{1}{1+e^{-\gamma}}$

[The last choice is interesting. You'll recall that LDA produces a posterior probability function with this expression. So the logistic function seems to be a natural form for modeling certain probabilities. If we want to model class probabilities, sometimes we use LDA; but alternatively, we could skip fitting Gaussians to points, and instead just try to directly fit a logistic function to a set of probabilities.]

Some loss fns: let z be prediction $h(x)$; y be true value
(A) $L(z, y) = (z - y)^2$ squared error
(B) $L(z, y) = |z - y|$ absolute error
(C) $L(z, y) = -y \ln z - (1 - y) \ln(1 - z)$ logistic loss, aka cross-entropy: $y \in [0, 1]$, $z \in (0, 1)$

Some cost fns to minimize:
(a) $J(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(X_i), y_i)$ mean loss [you can leave out the "$\frac{1}{n}$"]
(b) $J(h) = \max_{i=1}^{n} L(h(X_i), y_i)$ maximum loss
(c) $J(h) = \sum_{i=1}^{n} \omega_i L(h(X_i), y_i)$ weighted sum [some points are more important than others]
(d) $J(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(X_i), y_i) + \lambda |w|^2$ ℓ_2 penalized/regularized
(e) $J(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(X_i), y_i) + \lambda \|w\|_{\ell_1}$ ℓ_1 penalized/regularized

Some famous regression methods:

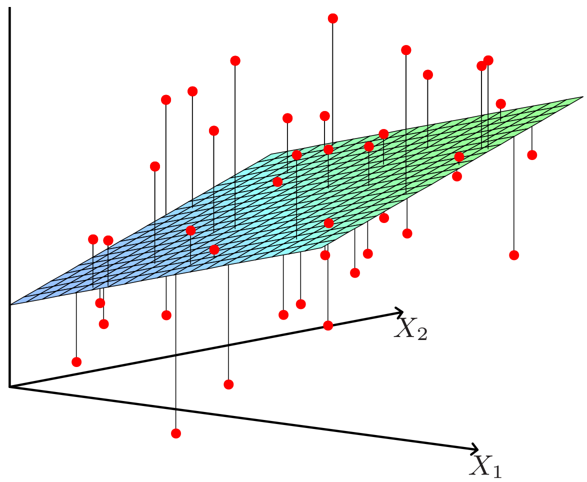
Least-squares linear regr.:	(1) + (A) + (a)	quadratic cost; minimize w/calculus
Weighted least-squ. linear:	(1) + (A) + (c)	
Ridge regression:	(1) + (A) + (d)	
Logistic regr.:	(3) + (C) + (a)	convex cost; minimize w/gradient descent
Lasso:	(1) + (A) + (e)	quadratic program
Least absolute deviations:	(1) + (B) + (a)	linear program
Chebyshev criterion:	(1) + (B) + (b)	

[I have given you several choices of regression function form, several choices of loss function, and several choices of objective function. These are interchangeable parts where you can snap one part out and replace it with a different one. But the optimization algorithm and its speed depend crucially on which parts you pick. Let's consider some examples.]

LEAST-SQUARES LINEAR REGRESSION (Gauss, 1801)

Linear regression fn (1) + squared loss fn (A) + cost fn (a).

$$\boxed{\text{Find } w, \alpha \text{ that minimizes } \sum_{i=1}^{n} (X_i \cdot w + \alpha - y_i)^2}$$



linregress.pdf (ISL, Figure 3.4) [An example of linear regression.]

Convention: X is $n \times d$ design matrix of sample pts
y is n-vector of scalar labels

$$\begin{array}{cc}
\begin{bmatrix}
X_{11} & X_{12} & \dots & X_{1j} & \dots & X_{1d} \\
X_{21} & X_{22} & & X_{2j} & & X_{2d} \\
\vdots & & & & & \\
X_{i1} & X_{i2} & & X_{ij} & & X_{id} \\
\vdots & & & & & \\
X_{n1} & X_{n2} & & X_{nj} & & X_{nd}
\end{bmatrix} \leftarrow \text{point } X_i^\top
&
\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\
\uparrow \ \text{feature column } X_{*j} & \uparrow \ y
\end{array}$$

Usually $n > d$. [But not always.]

Recall fictitious dimension trick [from Lecture 3]: rewrite $h(x) = x \cdot w + \alpha$ as

$$[x_1 \quad x_2 \quad 1] \cdot \begin{bmatrix} w_1 \\ w_2 \\ \alpha \end{bmatrix}.$$

Now X is an $n \times (d+1)$ matrix; w is a $(d+1)$-vector. [We've added a column of all-1's to the end of X.]
[We rewrite the optimization problem above:]

$\boxed{\text{Find } w \text{ that minimizes } |Xw - y|^2} = \text{RSS}(w)$, for residual sum of squares

Optimize by calculus:

$$\begin{aligned}
\text{minimize } \mathrm{RSS}(w) &= w^\top X^\top X w - 2y^\top X w + y^\top y \\
\nabla\, \mathrm{RSS} &= 2X^\top X w - 2X^\top y = 0 \\
&\Rightarrow \underbrace{X^\top X}_{(d+1)\times(d+1)}\ \underbrace{w = X^\top y}_{(d+1)-\text{vectors}} \qquad \Leftarrow \text{the } \underline{\text{normal equations}} \text{ [} w \text{ unknown; } X \text{ \& } y \text{ known]}
\end{aligned}$$

If $X^\top X$ is singular, problem is underconstrained
[because the sample points all lie on a common hyperplane. Notice that $X^\top X$ is always positive semidefinite.]

We use a linear solver to find $w = \underbrace{(X^\top X)^{-1} X^\top}_{X^+, \text{ the } \underline{\text{pseudoinverse}} \text{ of } X,\ (d+1)\times n} y$ [never actually invert the matrix!]

[We never compute X^+ directly, but we are interested in the fact that w is a linear transformation of y.]
[X is usually not square, so X can't have an inverse. However, every X has a pseudoinverse X^+, and if $X^\top X$ is invertible, then X^+ is a "left inverse."]

Observe: $X^+ X = (X^\top X)^{-1} X^\top X = I \quad \Leftarrow (d+1) \times (d+1)$ [which explains the name "left inverse"]

Observe: the predicted values of y are $\hat{y}_i = w \cdot X_i \quad \Rightarrow \quad \hat{y} = Xw = XX^+ y = Hy$
where $\underbrace{H}_{n\times n} = XX^+$ is called the hat matrix because it puts the hat on y

[Ideally, H would be the identity matrix and we'd have a perfect fit, but if $n > d + 1$, then H is singular.]

Interpretation as a projection:
- $\hat{y} = Xw \in \mathbb{R}^n$ is a linear combination of columns of X (one column per feature)
- For fixed X, varying w, Xw is subspace of $\mathbb{R}^n$ spanned by columns

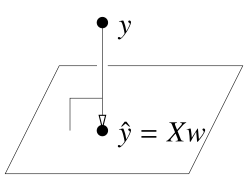


Figure in n-dimensional space (1 dim/sample)
NOT d-dimensional feature space

$\Leftarrow$ subspace spanned by X's columns
(at most $d + 1$ dimensions)

- Minimizing $|\hat{y} - y|^2$ finds point $\hat{y}$ nearest y on subspace
  $\Rightarrow$ projects y orthogonally onto subspace
  [the vertical line is the direction of projection and the error vector]
- Error is smallest when line is perpendicular to subspace: $X^\top (Xw - y) = 0$
  $\Rightarrow$ the normal equations!
- Hat matrix H does the projecting. [H is sometimes called the projection matrix.]

Advantages:
- Easy to compute; just solve a linear system.
- Unique, stable solution. [. . . except when the problem is underconstrained.]

Disadvantages:
- Very sensitive to outliers, because errors are squared!
- Fails if $X^\top X$ is singular.

[Apparently, least-squares linear regression was first posed and solved in 1801 by the great mathematician Carl Friedrich Gauss, who used least-squares regression to predict the trajectory of the planetoid Ceres. A paper he wrote on the topic is regarded as the birth of modern linear algebra.]

<u>LOGISTIC REGRESSION</u> (David Cox, 1958)

Logistic regression fn (3) + logistic loss fn (C) + cost fn (a).
Fits "probabilities" in range (0, 1).

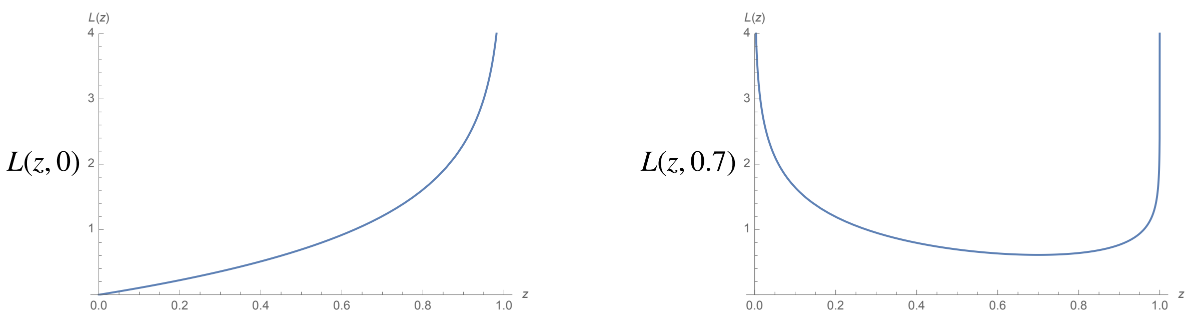
Usually used for classification. The input y_i's *can* be probabilities,
but in most applications they're all 0 or 1.

QDA, LDA: generative models
logistic regression: discriminative model
[We've learned from LDA that in classification, the posterior probabilities are often modeled well by a logistic function. So why not just try to fit a logistic function directly to the data, skipping the Gaussians?]

With X and w including the fictitious dimension; α is w's last component . . .

Find w that minimizes

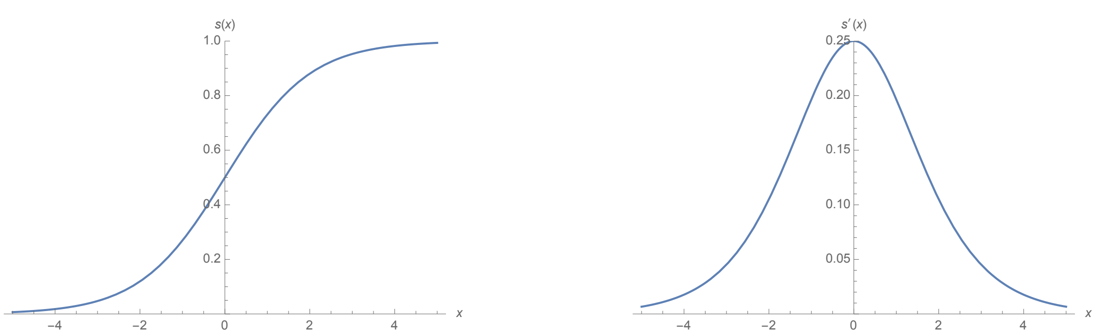
$$J = -\sum_{i=1}^{n} \left(y_i \ln s(X_i \cdot w) + (1 - y_i) \ln (1 - s(X_i \cdot w)) \right)$$



logloss0.pdf, losspt7.pdf [Plots of the loss $L(z, y)$ for $y = 0$ (left) and $y = 0.7$ (right). As you might guess, the left function is minimized at $z = 0$, and the right function is minimized at $z = 0.7$. These loss functions are always convex.]

$J(w)$ is convex! Solve by gradient descent.
[To do gradient descent, we'll need to compute some derivatives.]

$$\begin{aligned} s'(\gamma) &= \frac{\mathrm{d}}{\mathrm{d}\gamma} \frac{1}{1 + e^{-\gamma}} = \frac{e^{-\gamma}}{(1 + e^{-\gamma})^2} \\ &= s(\gamma)\,(1 - s(\gamma)) \end{aligned}$$



logistic.pdf, dlogistic.pdf [Plots of $s(\gamma)$ (left) and $s'(\gamma)$ (right).]

Let $s_i = s(X_i \cdot w)$

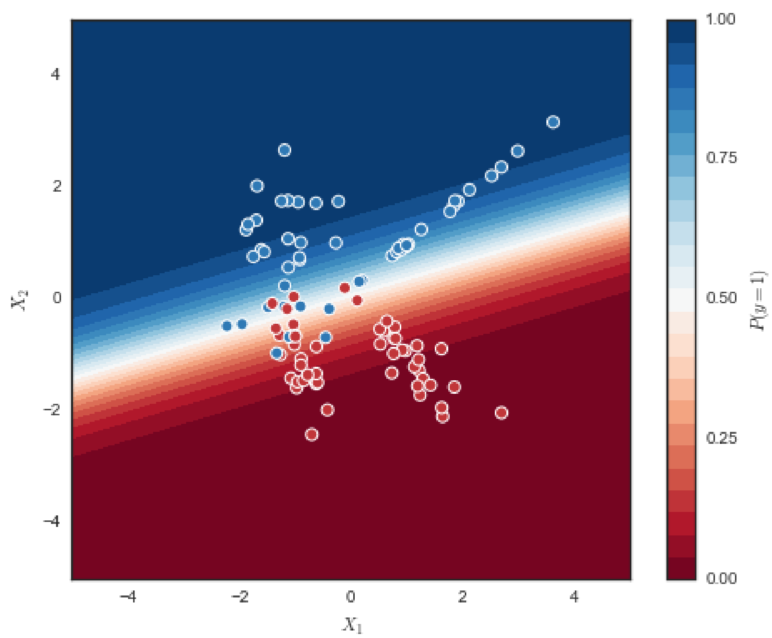
$$\begin{aligned}
\nabla_w J &= -\sum \left(\frac{y_i}{s_i} \nabla s_i - \frac{1-y_i}{1-s_i} \nabla s_i \right) \\
&= -\sum \left(\frac{y_i}{s_i} - \frac{1-y_i}{1-s_i} \right) s_i (1-s_i)\, X_i \\
&= -\sum (y_i - s_i)\, X_i \\
&= -X^\top (y - s(Xw)) \qquad \text{where } s(Xw) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \qquad \text{[applies } s \text{ component-wise to } Xw\text{]}
\end{aligned}$$

Gradient descent rule: $w \leftarrow w + \epsilon\, X^\top (y - s(Xw))$

Stochastic gradient descent: $w \leftarrow w + \epsilon\, (y_i - s(X_i \cdot w))\, X_i$
Works best if we shuffle points in random order, process one by one.
For very large n, sometimes converges before we visit all points!

[This looks a lot like the perceptron learning rule. The only difference is that the "$-s_i$" part is new.]

Starting from $w = 0$ works well in practice.



problogistic.png, by "mwascom" of Stack Overflow

http://stackoverflow.com/questions/28256058/plotting-decision-boundary-of-logistic-regression

[An example of logistic regression.]