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 do regression on probability values.]

– Choose form of regression fn \(h(x; w)\) with parameters \(w\) \(\text{ (} h \text{ = hypothesis) }\)
  – 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., empirical risk = expected loss on data

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 \(s(y) = \frac{1}{1+e^{-y}}\)

[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 posterior 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 label
(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) \(J(h) = (a), (b), or (c) + \lambda ||w||^2\) \(\ell_2\) penalized/regularized
(e) \(J(h) = (a), (b), or (c) + \lambda ||w||_{\ell_1}\) \(\ell_1\) penalized/regularized

Some famous regression methods:
Least-squares linear regress: \(1\) + (A) + (a) \(\quad \text{quadratic cost; minimize } w/\text{calculus}\)
Weighted least-squares linear: \(1\) + (A) + (c) \(\quad \text{quadratic program}\)
Ridge regression: \(1\) + (A) + (a) + (d) \(\quad \text{convex cost; minimize } w/\text{gradient descent}\)
Lasso: \(1\) + (A) + (a) + (e) \(\quad \text{linear program}\)
Logistic reg: \(3\) + (C) + (a) \(\quad \text{convex cost; minimize } w/\text{gradient descent}\)
Least absolute deviations: \(1\) + (B) + (a) \(\quad \text{linear program}\)
Chebyshev criterion: \(1\) + (B) + (b)

[I have given you several choices of regression function, several choices of loss function, and several choices of objective function. 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).

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

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

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

Usually $n > d$. [But not always.]  
Recall fictitious dimension trick [from Lecture 3]: rewrite $h(x) = x \cdot w + \alpha$ as

$$
\begin{bmatrix}
x_1 & x_2 & 1
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
\alpha
\end{bmatrix}
$$

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:]  

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

\[
\begin{align*}
\minimize \text{RSS}(w) &= w^T X^T X w - 2y^T X w + y^T y \\
\nabla \text{RSS} &= 2X^T X w - 2X^T y = 0 \\
\Rightarrow \quad X^T X w &= X^T y \quad \text{ <=> the normal equations [w unknown; X & y known]}
\end{align*}
\]

If \( X^T X \) is singular, problem is underconstrained.

[... because the sample points all lie on a common subspace (through the origin).]

[Notice that \( X^T X \) is always positive semidefinite, but not always positive definite.]

We use a linear solver to find \( w = (X^T X)^{-1} X^T y \).

[never actually invert the matrix!]

\( X^+, \) the pseudoinverse of \( X \), \((d+1)\times(n)

[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 \( X^T X \) is invertible, then \( X^+ \) is a “left inverse.”]

Observe: \( X^+ X = (X^T X)^{-1} X^T X = I \) \( \text{[which explains the name “left inverse”]} \)

Observe: the predicted values of \( y \) are \( \hat{y}_i = w \cdot X_i \) \( \Rightarrow \hat{y} = Xw = XX^+ y = H y \)

where \( H = XX^+ \) 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.]

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 \( X^T X \) is singular. [Which means the problem is underconstrained, has multiple solutions.]

[In discussion section 6, we’ll address how to handle the underconstrained case where \( X^T X \) 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.]

**LOGISTIC REGRESSION** (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 are 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; $\alpha$ is $w$’s last component . . .

Find $w$ that minimizes

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

$J(w)$ is convex! Solve by gradient descent.

[To do gradient descent, we’ll need to compute some derivatives.]

$$s'(\gamma) = \frac{d}{dy} \frac{1}{1 + e^{-\gamma}} = \frac{e^{-\gamma}}{(1 + e^{-\gamma})^2} = s(\gamma)(1 - s(\gamma))$$

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

$$\nabla_w J = - \sum \left( \frac{y_i}{s_i} \nabla s_i - \frac{1 - y_i}{1 - s_i} \nabla s_i \right)$$

$$= - \sum \left( \frac{y_i}{s_i} - \frac{1 - y_i}{1 - s_i} \right) s_i(1 - s_i) X_i$$

$$= - X^\top (y - s(Xw)) \quad \text{where } s(Xw) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \quad \text{[applies } s \text{ component-wise to } Xw]$$
Gradient descent rule: \( w \leftarrow w + \epsilon X^T (y - s(Xw)) \)

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.

If sample pts are linearly separable and \( w \cdot x = 0 \) separates them (with decision boundary touching no pt), scaling \( w \) to have infinite length causes \( s(X_i \cdot w) \rightarrow 1 \) for a pt \( i \) in class C, \( s(X_i \cdot w) \rightarrow 0 \) for a pt not in class C, and \( J(w) \rightarrow 0 \) [in the limit as \( ||w|| \rightarrow \infty \)].

[Moreover, making \( w \) grow extremely large is the only way to get the cost function \( J \) to approach zero.]

Therefore, logistic regression always separates linearly separable pts!

[In this case, the cost function \( J(w) \) has no finite local minimum, but gradient descent will “converge” to a solution, in the sense that the cost \( J \) will get arbitrarily close to zero, though of course the weight vector \( w \) will never become infinitely long. Mathematically speaking, \( w \) doesn’t converge at all—it diverges—though \( J(w) \) does converge to zero.]

[A 2018 paper by Soudry, Hoffer, Nacson, Gunasekar, and Srebro shows that gradient descent applied to logistic regression eventually converges to the maximum margin classifier, but the convergence is very, very slow. A practical logistic regression solver should use a different optimization algorithm.]