# 11 More Regression; Newton's Method; ROC Curves

## LEAST-SQUARES POLYNOMIAL REGRESSION

Replace each $X_i$ with feature vector $\Phi(X_i)$ with all terms of degree $0 \dots p$

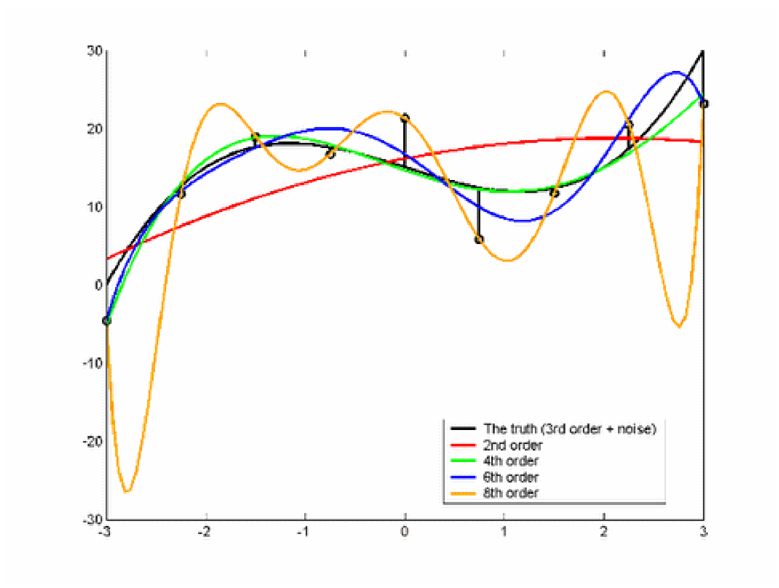
e.g., $\Phi(X_i) = [X_{i1}^2 \quad X_{i1}X_{i2} \quad X_{i2}^2 \quad X_{i1} \quad X_{i2} \quad 1]^\top$

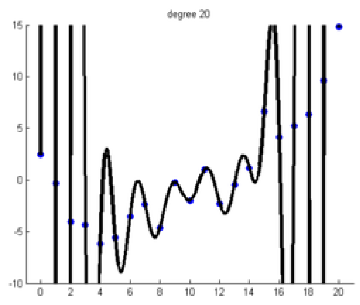
[Notice that we've added the fictitious dimension "1" here, so we don't need to add it again to do linear or logistic regression. This basis covers all polynomials quadratic in $X_{i1}$ and $X_{i2}$.]

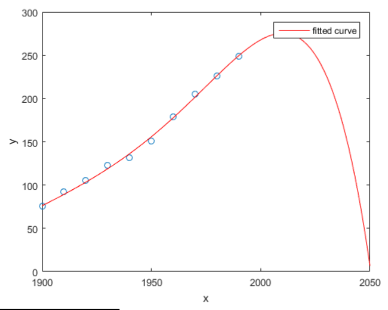
Can also use non-polynomial features (e.g., edge detectors).
Otherwise just like linear or logistic regression.

Log. reg. + quadratic features = same logistic posteriors as QDA.

Very easy to overfit!







overunder.png, degree20.png, UScensusquartic.png

[Here are some examples of polynomial overfitting, to show the importance of choosing the polynomial degree very carefully. At left, we have sampled points from a degree-3 curve (black) with added noise. We show best-fit polynomials of degrees 2, 4, 6, and 8 found by regression of the black points. The degree-4 curve (green) fits the true curve (black) well, whereas the degree-2 curve (red) underfits and the degree-6 and 8 curves (blue, yellow) overfit the noise and oscillate. The oscillations in the yellow degree-8 curve are a characteristic problem of polynomial interpolation.]

[At upper right, a degree-20 curve shows just how insane high-degree polynomial oscillations can get. It takes a great deal of densely spaced data to tame the oscillations in a high degree curve, and there isn't nearly enough data here.]

[At lower right, somebody has regressed a degree-4 curve to U.S. census population numbers. The curve doesn't oscillate, but can you nevertheless see a flaw? This shows the difficulty of *extrapolation* outside the range of the data. As a general rule, extrapolation is much harder than interpolation. The $k$-nearest neighbor classifier is one of the few that does extrapolation decently without occasionally returning crazy values.]

## WEIGHTED LEAST-SQUARES REGRESSION

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

[The idea of weighted least-squares is that some sample points might be more trusted than others, or there might be certain points you want to fit particularly well. So you assign those more trusted points a higher weight. If you suspect some points of being outliers, you can assign them a lower weight.]

Assign each sample pt a weight $\omega_i$; collect them in $n \times n$ diagonal matrix $\Omega$.

Greater $\omega_i$ → work harder to minimize $|\hat{y}_i - y_i|^2$ recall: $\hat{y} = Xw$ [$\hat{y}_i$ is predicted label for $X_i$]

$$\boxed{\text{Find } w \text{ that minimizes } (Xw - y)^\top \Omega (Xw - y)} = \sum_{i=1}^{n} \omega_i \, (X_i \cdot w - y_i)^2$$

[As with ordinary least-squares regression, we find the minimum by setting the gradient to zero, which leads us to the normal equations.]
Solve for $w$ in normal equations: $X^\top \Omega X w = X^\top \Omega y$

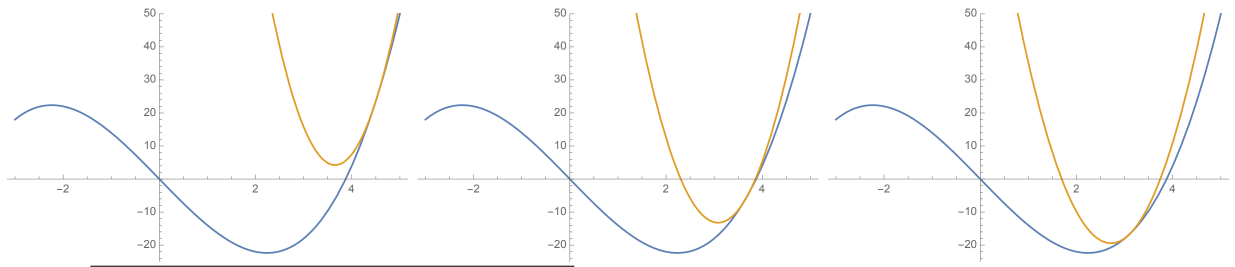
[Once again, you can interpret this method as a projection of $y$.]
Note: $\Omega^{1/2}\hat{y}$ is the pt nearest $\Omega^{1/2}y$ on subspace spanned by columns of $\Omega^{1/2}X$.

[If you stretch the $n$-dimensional space by applying the linear transformation $\Omega^{1/2}$, $\hat{y}$ is an orthogonal projection of $y$ onto the stretched subspace.]

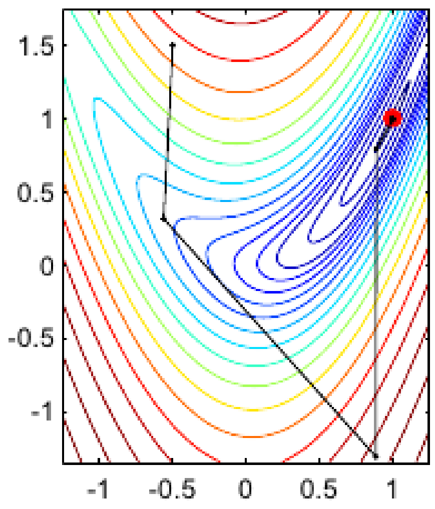
## NEWTON'S METHOD

Iterative optimization method for smooth fn $J(w)$.
Often much faster than gradient descent. [We'll use Newton's method for logistic regression.]

Idea: You're at point $v$. Approximate $J(w)$ near $v$ by quadratic fn.
Jump to its unique critical pt. Repeat until bored.



$\boxed{\text{newton1.pdf, newton2.pdf, newton3.pdf}}$ [Three iterations of Newton's method in one-dimensional space. We seek the minimum of the blue curve, $J$. Each brown curve is a local quadratic approximation to $J$. Each iteration, we jump to the bottom of the brown parabola.]



newton2D.png [Steps taken by Newton's method in two-dimensional space.]

Taylor series about $v$:

$$\nabla J(w) = \nabla J(v) + (\nabla^2 J(v))\,(w - v) + O(|w - v|^2)$$

where $\nabla^2 J(v)$ is the Hessian matrix of $J$ at $v$.

Find critical pt $w$ by setting $\nabla J(w) = 0$:

$$w = v - (\nabla^2 J(v))^{-1}\,\nabla J(v)$$

[This is an iterative update rule you can repeat until it converges to a solution. As usual, we don't really want to compute a matrix inverse. Instead, we solve a linear system of equations, typically by Cholesky factorization or the conjugate gradient method.]

Newton's method:

```
pick starting point $w$
repeat until convergence
    $e \leftarrow$ solution to linear system $(\nabla^2 J(w))\,e = -\nabla J(w)$
    $w \leftarrow w + e$
```

Warning: Doesn't know difference between minima, maxima, saddle pts.
Starting pt must be "close enough" to desired solution.

[If the objective function $J$ is actually quadratic, Newton's method needs only one step to find the correct answer. The closer $J$ is to quadratic, the faster Newton's method tends to converge.]

[Newton's method is superior to blind gradient descent for some optimization problems for several reasons. First, it tries to find the right step length to reach the minimum, rather than just walking an arbitrary distance downhill. Second, rather than follow the direction of steepest descent, it tries to choose a better descent direction.]

[Nevertheless, it has some major disadvantages. The biggest one is that computing the Hessian can be quite expensive, and it has to be recomputed every iteration. It can work well for low-dimensional weight spaces, but you would never use it for a neural network, because there are too many weights. Newton's method also doesn't work for most nonsmooth functions. It particularly fails for the perceptron risk function, whose Hessian is zero, except where the Hessian is not even defined.]

## LOGISTIC REGRESSION (continued)

[Let's use Newton's method to solve logistic regression faster.]

Recall: $s'(\gamma) = s(\gamma)\,(1 - s(\gamma))$, $\qquad s_i = s(X_i \cdot w)$, $\qquad s = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}$,

$$\nabla_w J = -\sum_{i=1}^{n} (y_i - s_i)\, X_i = -X^\top (y - s)$$

[Now let's derive the Hessian too, so we can use Newton's method.]

$$\nabla_w^2 J(w) = \sum_{i=1}^{n} s_i(1 - s_i)\, X_i X_i^\top = X^\top \Omega X \qquad \text{where } \Omega = \begin{bmatrix} s_1(1-s_1) & 0 & \dots & 0 \\ 0 & s_2(1-s_2) & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & s_n(1-s_n) \end{bmatrix}$$

$\Omega$ is +ve definite $\forall w \quad \Rightarrow \quad X^\top \Omega X$ is +ve semidefinite $\forall w \quad \Rightarrow \quad J(w)$ is convex.
[The logistic regression cost function is convex, so Newton's method finds a globally optimal point if it converges at all.]

Newton's method:

```
$w \leftarrow 0$
repeat until convergence
    $e \leftarrow$ solution to normal equations $(X^\top \Omega X)\, e = X^\top (y - s)$        Recall: $\Omega$, $s$ are fns of $w$
    $w \leftarrow w + e$
```

[Notice that this looks a lot like weighted least squares, but the weight matrix $\Omega$ and the right-hand-side vector $y - s$ change every iteration. So we call it . . . ]
An example of iteratively reweighted least squares.

$\Omega$ prioritizes points with $s_i$ near 0.5; tunes out pts near 0/1.
[In other words, sample points near the decision boundary have the biggest effect on the iterations. Meanwhile, the iterations move the decision boundary; in turn, that movement may change which points have the most influence. In the end, only the points near the decision boundary make a big contribution to the logistic fit.]

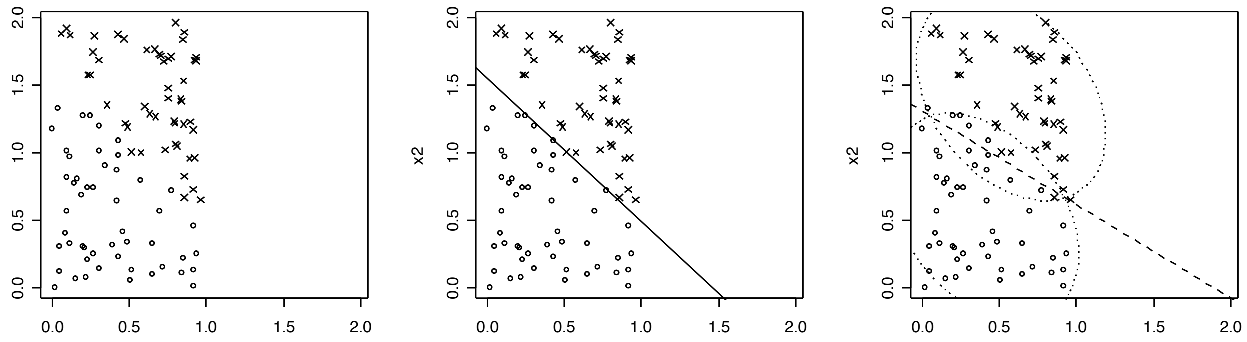
Idea: If $n$ very large, save time by using a random subsample of the pts per iteration. Increase sample size as you go.

[The principle is that the first iteration isn't going to take you all the way to the optimal point, so why waste time looking at *all* the sample points? Whereas the last iteration should be the most accurate one.]

## LDA vs. Logistic Regression

Advantages of LDA:
- For well-separated classes, LDA stable; log. reg. surprisingly unstable
- $> 2$ classes easy & elegant; log. reg. needs modifying (softmax regression)
- LDA slightly more accurate when classes nearly normal, especially if $n$ is small

Advantages of log. reg.:
- More emphasis on decision boundary [though not as much as a hard-margin SVM]
  [When we analyzed Newton's method, we learned that points near the decision boundary have greater influence on the boundary. By contrast, LDA gives all the sample points equal weight when fitting Gaussians to them.]

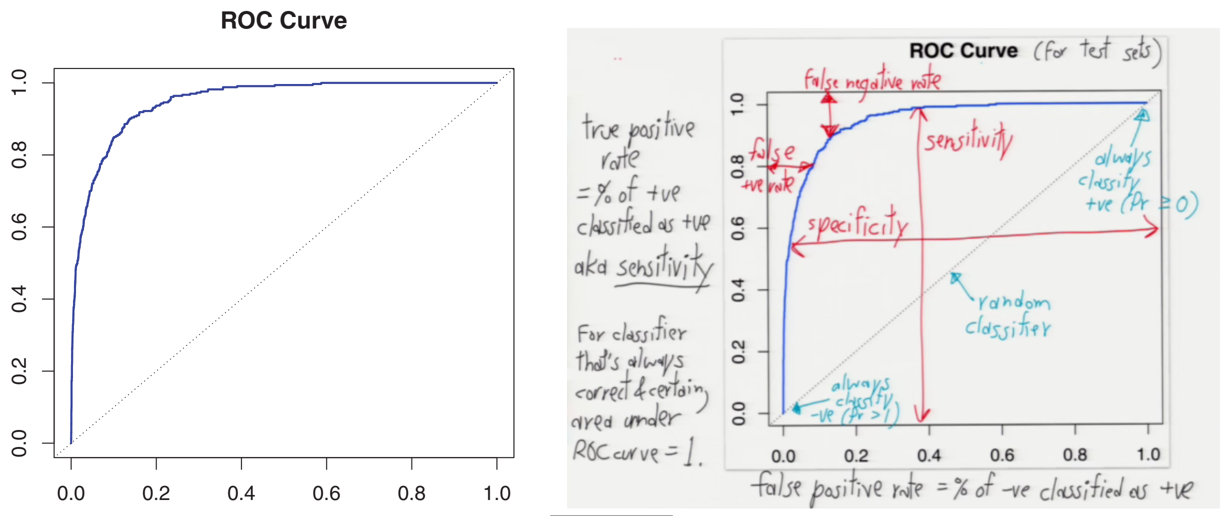


$\boxed{\text{logregvsLDAuni.pdf}}$ [Logistic regression vs. LDA for a linearly separable data set with a very narrow margin. Logistic regression (center) succeeds in separating these classes, because points near the decision boundary have more influence. (Some luck is involved too; unlike a hard-margin SVM, logistic regression doesn't always separate separable points.) In this example, LDA (right) misclassifies some of the training points.]

- Hence less sensitive to most outliers
  [including outliers far out on the wrong side of the decision boundary . . . compare with SVMs]
- Easy & elegant treatment of "partial" class membership; LDA pts are all-or-nothing
  [because logistic regression accommodates labels between 0 and 1]
- More robust on some non-Gaussian distributions (e.g., dists. w/large skew)

[When you use logistic regression with quadratic features, you get a quadric decision boundary, just as you do with QDA. Based on what I've said here, do you think logistic regression with quadratic features gives you exactly the same classifier as QDA?]

## ROC CURVES (for test sets)



ROC.pdf

[This is a ROC curve. That stands for receiver operating characteristics, which is an awful name but we're stuck with it for historical reasons.
A ROC curve is a way to evaluate your classifier after it is trained.
It is made by running a classifier on the test set or validation set.
It shows the rate of false positives vs. true positives for a range of settings.
We assume there is a knob we can turn to trade off these two types of error. For our purposes, that knob is the posterior probability threshold for Gaussian discriminant analysis or logistic regression.
However, neither axis of this plot is that knob.]

$x$-axis: "false positive rate = % of −ve classified as +ve"
$y$-axis: "true positive rate = % of +ve classified as +ve aka sensitivity"
"false negative rate": vertical distance from curve to top [1− sensitivity]
"specificity": horizontal distance from curve to right [1− false positive rate; "true negative rate"]
[You generate this curve by trying *every* probability threshold; for each threshold, measure the false positive & true positive rates and plot a point.]

upper right corner: "always classify +ve (Pr $\geq$ 0)"
lower left corner: "always classify −ve (Pr > 1)"
diagonal: "random classifiers"
[A rough measure of a classifier's effectiveness is the area under the curve. For a classifier that is always correct, the area under the curve is one. For the random classifier, the area under the curve is $1/2$, so you'd better do better than that.]

[IMPORTANT: In practice, the trade-off between false negatives and false positives is usually negotiated by choosing a point on this plot, based on real test data, and NOT by taking the choice of threshold that's best in theory.]