STATS 315A

STATS 315A

Chapter 1: Motivating Examples

- Prostate Cancer Dataset
  - Pairwise comparisons - can see binary vars like svi, gleason takes 4 values but only a single observation for value 3
  - For variables with zero values but non negative, take $log(x+\epsilon)$
  - Could split a variable - zero or non zero, then have the values for the non-zero portion as a new variable
  - Can see outliers in pairwise, investigate errors say in data entry
- Phoneme Classification
  - Discretize the analog spoken data via sampling, trying to classify sounds as “aa” or “ao”
  - For each observation have feature vector $x = (x_1, \dots, x_{256})$. Features are the values of the periodogram at each index.
  - Using logistic regression - take log odds: $log(\frac{P(v=aa)}{P(v=ao)}) = \beta_0 + \sum_{j=1}^{256} x_j \beta_j$. Fit the model via MLE to the data, can plot the estimates as a function of frequency. The estimates are variable - have many parameters and not a lot of data.
    - We sampled at 256 data points per analog curve - some arbitrariness, could have sampled at different frequency.
    - Correlation among the features - adjacent samples are coming from adjacent / similar parts of the curve. The higher the resolution in the sample the closer the values - expect collinearity in our samples, the autocorrelation means the coefficients are not very well determined.
    - Red curve - could introduce some regularization for coefficient sequence. For example constraint $\sum_{j=1}^{255} |\beta_j - \beta_{j+1}| \leq c$.
    - Better methods such as filtering will be introduced in chapter 5. Once we constrain the coefficients to be smooth, our sampling frequency no longer makes much of a difference and our correlation problem is not so large
- Handwritten digits - similar to phonemes, taking an analogue signal, choosing a resolution to discretize the data
  - With no information, could take the frequencies of the digits in a training set, predict the mode for each OOS. Null error rate - misclassification rate with naive guessing. With 10 digits, null rate is 10% accuracy
  - Linear methods were troubled by the variations in writing digits. KNN just needs to find the example that is closest to the OOS observation at hand. KNN works very well in many classification settings.
- Tumor Genetic Markers
  - Wide dataset over 20k genes. Organized into a matrix where each column is a sample, each row is a gene. Heat map, red overexpressed, green underexpressed. Hierarchical clustering leads to the groupings of red and green. Many more categories than observations so have to be careful about overfitting.
- Land Usage Map
  - Use another KNN type method. Take a pixel and the 8-neighborhood - the squares in a grid around it. Then shift the target pixel, build up big data set of 36-vectors corresponding to each pixel in the image. Then in 36-dimensions, predict the land usage based on NN in training set.

Chapter 2: Supervised Learning

- Goals to predict test cases, understand the inputs effect on the outcome, assess the quality of predictions and inferences
- Confidence could be derived from distance from decision boundary in classification. For regression, prediction interval smaller in center in dataset where we have many training observations, but a test point near the edge of our training set have much larger error bands. In higher dimensions, have all kinds of holes in the feature space.
- Netflix prize - somewhere in between supervised, unsupervised due to data sparsity. Led to development of matrix completion - SVD / PC to construct a low rank matrix approximation. X mxn - A mxk B kxn. $min_{A,B}||X - AB||_F^2$, min over Frobenius norm. $X = UDV^T$ and m >> n, then $U_{m\times n}, D_{n\times n}, V_{n\times n}^T$ - the D take values of 0 after we reach the rank k, then can distribute the orthogonal matrices into A and B and reconstruct. Problems? We did not handle the NAs and SVD will be performed on huge matrix. Can actually solve the problem with iterating least squares - given an A, our minimization is a least squares problem, then given B, solve for A.
    - To solve the missing values problem, can redefine the norm $min_{A,B}||X - AB||_{F,\Omega}^2$. Could try performing a weighted least squares $\sum_{i=1}^m w_i(x_{ij} - a_i^T b_j)^2$, setting missing values to weights 0. Perform for each column of x, not efficient since have to rerun for each movie.
    - Simpler idea: fill in NAs with some guess, then use SVD to find an A and B. Then use the A and B to fill in the missing values with better values. Then repeat - called hard impute. Not guaranteed to finda global minimum but get some improvement.
    - Could store the original matrix in a sparse matrix format so it takes up little space. Ω is just where we have observed values - we can make a matrix of binary values where values exist - then $P_\Omega(X)$ gets rid of NA values. $P_{\Omega^\perp}(AB)$, where $P_{\Omega^\perp}$ is the opposite of omega, where all are missing. Take $P_\Omega(X) - P_\Omega(AB) + AB$. The first two terms are sparse and AB is low rank - both stored easily.

Notation

- X: Input variable. If X is a vector, its components can be accessed by subscripts X_j
- Quantitative outputs will be denoted by Y , and qualitative outputs by G
- Observed values are written in lowercase; hence the ith observed value of X is written as x_i
- Matrices are represented by bold uppercase letters; for example, a set of N input p-vectors x_i, i = 1,...,N would be represented by the N ×p matrix $\mathbf{X}$.
- Since all vectors are assumed to be column vectors, the ith row of X is x_i^T , the vector transpose of x_i.

Least Squares

- Given vector of inputs $X^T = (X_1, X_2, \ldots, X_p)$, we predict Y with the model $\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^p X_j\hat{\beta}_j$. $\hat{\beta}_0$ is the intercept or bias.
- Including a constant variable 1 in X and $\hat{\beta}_0$ in the vector of coefficients $\hat{\beta}$, we can write the model as the inner product: $\hat{Y} = X^T\hat{\beta}$. (Make the inclusion assumption moving forward). $X^T = [1X_1X_2]$
- Here $\hat{Y}$ is a scalar since we model a single output, though $\hat{Y}$ could be a K-vector making $\hat{\beta}$ a p x K matrix.
- Least Squares minimizes RSS: $\mathrm{RSS}(\beta) = \sum_{i=1}^N \left(y_i - x_i^T\beta\right)^2 = \|y - X\beta\|^2 = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$
    - X is an N x p matrix with each row an input vector, y is an N-vector of the outputs of the training set
- Normal Equations: differentiate wrt β to get $\partial\mathbf{RSS}/\partial\beta = -2X^T(y - X\beta) = 0 \ = X^T(y - X\beta) = 0$
- For non-singular $\mathbf{X}^T\mathbf{X}$, the unique solution given by $\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$, with specific point prediction given by $\hat{y}\left(x_i\right) = x_i^T\hat{\beta}$
- Geometric interpretation
    - $\hat{Y} = X\hat{\beta}$ is the orthogonal projection of y onto the subspace $M \subset \mathbb{R}^n$ spanned by the columns of X. (True even if X is not full rank).
- We seldomly actually invert X^TX in practice - instead use QR decomposition

    - For full rank: $\|y - X\beta\|^2 = \left\|Q^T y - R\beta\right\|^2 = \left\|Q_1^T y - R_1\beta\right\|^2 + \left\|Q_2^T y\right\|^2 \Rightarrow \hat{\beta} = R_1^{-1} Q_1^T y$ and $\mathrm{RSS}(\hat{\beta}) = \left\|Q_2^T y\right\|^2$, $e = Q^T y$
    - $\hat{y} = Q_1 Q_1^T y = Hy = X\left(X^T X\right)^{-1} X^T y$ - H is the hat matrix bc it puts the hat on y
- For classification, could create a linear boundary with regression. If training data comes from bivariate Gaussians this model is pretty much optimal save for a few tweaks (which is somewhat surprising!), but if each class comes from a mixture of 10 low-variance Gaussians mixed together, need something more flexible - such as KNN.
- For mixed Gaussians $f(x) = \sum_{j=1}^{10} \pi_j \phi(\mu_j, \Sigma_j)$ where $\sum_{i=1}^{10} \pi_j = 1$ - ie a weighting of the various Gaussians with different means and covariances $\Sigma_j = \sigma^2 I$. Have means scattered with circular distributions around them - they mix together quite a bit. Flexible way of generating densities - nice for generative models, since have specified means, could be good for generating clusters.

Nearest Neighbors

- Memory based method, no model, just fit based on stored training data.
- Drawn decision boundary - contour drawn where blue and orange classifications split for discrete test points. With 1-NN, decision boundary drawn via Voronoi tesselation, the boundary bisects the distance between each pair of different class points.
- 1-NN can certainly overfit, but it many problems like image classification, 1-NN can do really well when the signal to noise ratio is really high.
- KNN model: $\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$. Here $N_k(x)$ is the neighborhood of x defined by the k closest points (euclidean distance) to xi in the training sample. Essentially just averaging over closest points.
- For K=1, each point x_i has a tile bounding the region for which it is the closest input point. The error on the training data will always be 0 for K=1
- While KNN appears to have a single parameter K, the effective number of parameters if N/k > p from least squares, decreasing with increasing k. Could imagine a one dimensional distribution of data, place them into bins and take majority votes within bin - then we are specifying N/k bin votes, which are our effective parameters
- Kernel smoother - instead of bins imagine an interval around the target test point, take a smooth weighting function over the points within the interval so that farther training points get a continuously smaller vote. Kernel smoothing differs from local linear regression most where data has a stronger trend than an average of the points in the region.
- Test set of 10k points - make simulated test sets huge to get a better reading of your method. No reason to stay with a small test set that will have its own variance problems.
- Optimal Bayes Error rate - can write a joint distribution since we simulated the data. Then use Bayes rule to work out the exact conditional probability: $P(Y = 1, X = x) = \frac{1}{2} f_1(x),\ P(Y = 0, X = x) = \frac{1}{2} f_0(x)$, then $P(Y = 1 | X = x) = \frac{\frac{1}{2} f_1(x)}{\frac{1}{2} f_1(x) + \frac{1}{2} f_0(x)} = \frac{f_1(x)}{f_1(x) + f_0(x)}$

Statistical Decision Theory

- $X \in \mathbb{R}^p$ - a real valued random input vector. $Y \in \mathbb{R}$ - real valued output vector. Joint distribution P(X,Y)
- Seek a function $f(x)$ for predicting Y from X, and loss function L(Y, f(x)) for penalizing errors in prediction
- Using squared error loss, $\mathrm{EPE}(f) = \mathrm{E}(Y - f(X))^2 = \int [y - f(x)]^2 \Pr(dx, dy) = E_{XY}(Y - f(X))^2 = \mathrm{E}_X \mathrm{E}_{Y|X}\left([Y - f(X)]^2 | X\right)$. We convert the squared error loss to an optimization objective. Note $E_X\left(E_{Y|x}(Y - E(Y|x))^2\right.$ does not contain f(x) - it is the irreducible error that does not depend on the form of f. (Had we used the absolute loss, we would get the conditional median instead of mean).
- Minimizing EPE point wise, $f(x) = \mathrm{argmin}_c\, \mathrm{E}_{Y|X}\left([Y - c]^2 | X = x\right)$, we get $f(x) = \mathrm{E}(Y | X = x)$
- The final conditional expectation is known as the regression function. Thus the best prediction of Y at any point X = x is the conditional mean, when best is measured by average squared error.

- With KNN, we use $\hat{f}(x) = \text{Ave}(y_i|x_i \in N_k(x))$ - we approximate the expectation by an average and conditioning at a point relaxed to conditioning on a region "close" to the target point. This average becomes more stable as N grows, converges to expectation.
- We often do not have enough data for convergence or p grows too large, slowing the rate of convergence
- For regression, $f(x) \approx x^T\beta$, then $\beta = \left[\text{E}\left(XX^T\right)\right]^{-1}\text{E}(XY)$. The least squares solution amounts to replacing these expectations with averages over the training data.
- Using L1 loss instead of L2, the solution becomes $\hat{f}(x) = \text{median}(Y|X = x)$ - more robust than conditional mean.
- Classification
  - Loss function K x K matrix L, zero on diagonal and non negative elsewhere. $L(k, \ell)$ is price paid for assigning l to a k observation - loss matrix k x k.
  - Using 0-1 loss, all misclassifications are charged a single unit:
    $\text{EPE} = \text{E}[L(G, \hat{G}(X))] = \text{E}_X \sum_{k=1}^{K} L\left[\mathcal{G}_k, \hat{G}(X)\right] \Pr(\mathcal{G}_k|X)$
  - Solved by $\hat{G}(x) = \text{argmin}_{g\in\mathcal{G}} \sum_{k=1}^{K} L\left(\mathcal{G}_k, g\right) P\left(\mathcal{G}_k|X = x\right)$ or can think of the posterior as $\hat{G}(x) = \mathcal{G}_k \text{ if } \Pr(\mathcal{G}_k|X = x) = \max_{g\in\mathcal{G}} \Pr(g|X = x)$ - the Bayes classifier. Error rate of the Bayes classifier is called the Bayes rate
  - Conditioning on x, say in 2 dimensions, we look at the one dimensional distribution of Y above that x. Our best estimate then is the mean of that one dimensional distribution of Y. For classification, we then just look at the probabilities of the class labels, and with 0-1 loss, select the class with highest probability.
  - This is all fine given a known distribution. With KNN, since we don't know the distribution of Y given any x, we relax these expectations and assume the density is locally smooth in a small neighborhood around the target point.

Local Methods in High Dimensions

- Curse of dimensionality - try to capture fraction r of observations, edge length of volume needed given by $e_p(r) = r^{1/p}$. For even moderately large p, we have to capture the majority of the range of the each input variable. Secondly, most data points are closer to the boundary of the sample space than to any other data point. Thirdly, the sampling density is proportional to $N^{1/p}$, where p is the dimension of the input space and N is the sample size - in high dimensions all feasible training samples sparsely populate the input space.
- MSE at a point:
  $\text{MSE}(x_0) = \text{E}_{\mathcal{T}}\left[f(x_0) - \hat{y}_0\right]^2 = \text{E}_{\mathcal{T}}\left[\hat{y}_0 - \text{E}_{\mathcal{T}}\left(\hat{y}_0\right)\right]^2 + \left[\text{E}_{\mathcal{T}}\left(\hat{y}_0\right) - f\left(x_0\right)\right]^2 = \text{Var}_{\mathcal{T}}(\hat{y}_0) + \text{Bias}^2(\hat{y}_0)$
- In low dimensions and with N = 1000, the nearest neighbor is very close to 0, and so both the bias and variance are small. As the dimension increases, the nearest neighbor tends to stray further from the target point, and both bias and variance are incurred.
- Bias increases as dimension increases with 1-NN, but variance can decrease - function gets very flat far from the test point, so the predictions don't vary much once all distances are huge. (Page 25). Alternatively, with a different function, the bias is small and the variance takes off - very scenario dependent.

Statistical Models

- Additive error model for regression $Y = f(X) + \varepsilon$ where random error epsilon has $E(\varepsilon) = 0$, independent of X. This assumptions imply: $f(x) = \text{E}(Y|X = x)$. $P(Y|X)$ depends on X only though f(X) as an assumption - we get the same shape distribution for the errors given any value of X. These model's approximation that means all unmeasured variables are captured by ε. N realizations from the model and assume errors are independent - $Var(\varepsilon_i) = \sigma^2$
- Generally there will be other unmeasured variables that also contribute to Y, including measurement error. The additive model assumes that we can capture all these departures from a deterministic relationship via the error epsilon.
- Function approximation - The goal is to obtain a useful approximation to f(x) for all x in some region of $\mathbb{R}^p$, given the representations in T. Allows us to use geometrical concepts and probabilistic inference.

- We can write down a loss function without making any assumptions - doesn't necessarily imply a statistical model. But it is often useful to employ a model instead of pure inference learning from data.
- MLE: $P(G = k|X = x) = p_{k,\theta}(x)$ - vector of functions for G = k. Maximizes log likelihood to pick the parameter.
- Linear basis expansions $f_\theta(x) = \sum_{k=1}^{K} h_k(x)\theta_k$.
  - h_k are a suitable set of functions or transformations of the input vector x, such as polynomials, trig functions. Or non-linear expansions such as the sigmoid $h_k(x) = \frac{1}{1+\exp(-x^T\beta_k)}$
  - θ - set of parameters modified to suit the data at hand, say $\theta = \beta$ for regression. Could estimate through OLS
- We imagine our parameterized function as a surface in p+1 space and we just see noisy realizations of it.
- MLE - for random sample, density $y_i, i = 1, \ldots, N,\ \Pr_\theta(y)$, get log probability $L(\theta) = \sum_{i=1}^{N} \log \Pr_\theta(y_i)$
  - For additive model, LS equivalent to MLE using conditional likelihood $\Pr(Y|X, \theta) = N\left(f_\theta(X), \sigma^2\right)$

Structured Regression Models

- For arbitrary f, minimizing $\mathrm{RSS}(f) = \sum_{i=1}^{N} \left(y_i - f\left(x_i\right)\right)^2$ has infinite solutions since any f passing through the points is min. We need to restrict the class of function considered. If our prediction fits every training point, under a different training set where for each x there is a slightly different y value, every prediction will have error.
- However If there are multiple observation pairs $x_i, y_{i\ell},\ \ell = 1, \ldots, N_i$ at each value of x_i, the risk is limited - solutions pass through average of $y_{i\ell}$. If sample size is infinite, this is guaranteed, but for finite N, need to reduce set of f's.
- There are still infinite possible restrictions, so the amiguity still exists. In general the constraints imposed by most learning methods can be described as complexity restrictions of one kind or another. This usually means some kind of regular behavior in small neighborhoods of the input space. How small a neighborhood do we draw within which the function is constant. Very often we impose smoothing conditions - either explicity like setting a linear model or splines, implicity with some nonparametric method.
- Any method that attempts to produce locally varying functions in small isotropic neighborhoods will run into problems in high dimensions—again the curse of dimensionality. And conversely, all methods that overcome the dimensionality problems have an associated—and often implicit or adaptive—metric for measuring neighborhoods, which basically does not allow the neighborhood to be simultaneously small in all directions. If we restrict our attention to a single dimension or a few, we can look closely at distance in those directions.

Classes of Restricted Estimators

- Roughness penalty and Bayesian methods
  - Penalize RSS with a roughness penalty: $\mathrm{PRSS}(f; \lambda) = \mathrm{RSS}(f) + \lambda J(f)$. Instead of minimizing RSS through interpolation between all points, we force some smoothing considerations.
  - $J(f)$ will be large for functions that vary too rapidly over small regions of input space. Eg. cubic smoothing spline $\mathrm{PRSS}(f; \lambda) = \sum_{i=1}^{N} \left(y_i - f\left(x_i\right)\right)^2 + \lambda \int \left[f''(x)\right]^2 dx$
  - Penalty function, or regularization methods, express our prior belief that the type of functions we seek exhibit a certain type of smooth behavior, and indeed can usually be cast in a Bayesian framework.
- Kernel Methods and Local Regression
  - Explicitly providing estimates of the regression function or conditional expectation by specifying the nature of the local neighborhood, and of the class of regular functions fitted locally.
  - Kernel function specifies local neighborhood $K_\lambda\left(x_0, x\right)$. Assigns weights to points x in region around x_0.
  - Eg. local regression minimizing $\mathrm{RSS}(f_\theta, x_0) = \sum_{i=1}^{N} K_\lambda\left(x_0, x_i\right)\left(y_i - f_\theta\left(x_i\right)\right)^2$
- Basis Functions and Dictionary Methods
  - Includes the familiar linear and polynomial expansions, but more importantly a wide variety of more flexible models. Model f is a linear expansion of basis functions $f_\theta(x) = \sum_{m=1}^{M} \theta_m h_m(x)$. M can be a tuning parameter, choose the # of basis functions.
  - Radial basis functions are symmetric p-dimensional kernels located at particular centroids: $f_\theta(x) = \sum_{m=1}^{M} K_{\lambda_m}\left(\mu_m, x\right)\theta_m$

Model Selection and Bias-Variance Tradoff

- Many flexible methods have a complexity parameter described above, since RSS would produce a perfectly interpolating function. Can have low penalty in low noise situations and come close to fitting data exactly - think of tight fitting of NN for image classification. Such a high signal to noise ratio that danger of overfitting is low.
- $\mathrm{EPE}_k(x_0) = \mathrm{E}\left[\left(Y - \hat{f}_k(x_0)\right)^2 | X = x_0\right] = \sigma^2 + \left[\mathrm{Bias}^2\left(\hat{f}_k(x_0)\right) + \mathrm{Var}_{\mathcal{T}}\left(\hat{f}_k(x_0)\right)\right] = \sigma^2 + \left[f(x_0) - \frac{1}{k}\sum_{\ell=1}^{k} f\left(x_{(\ell)}\right)\right]^2 +$ for $\mathcal{T}$= training data. $Y|x_0$ is independent of $\mathcal{T}$. $\sigma^2_{x_0}$ is the variance of Y arouund x0.
- Note $\hat{f}(X_0)$ is a RV, it depends on the training data - this leads directly to our understanding of bias and variance
- The first term σ^2 is the irreducible error. The second and third terms are under our control, and make up the mean squared error, broken down into a bias component and a variance component.
- Bias: $\left[\mathrm{E}_{\mathcal{T}}\left(\hat{f}_k(x_0)\right) - f(x_0)\right]^2$ square difference between true mean and expected value of the estimate.
- When we calculated EPE, consider what is random. Treating $x_i's$ as fixed, the randomness if from the training Y's that are associated with the x's in the training set. Then we can say
  $\mathrm{E}_{Y|X}\left[\left(Y - \hat{f}_k(x_0)\right)^2 | X = x_0\right] = \sigma^2 + \mathrm{Bias}^2\left(\hat{f}_k(x_0)\right) + \mathrm{Var}_{\mathcal{T}}\left(\hat{f}_k(x_0)\right) = \sigma^2 + \left[f(x_0) - \frac{1}{k}\sum_{\ell=1}^{k} f\left(x_{(\ell)}\right)\right]^2 + \frac{\sigma^2}{k}$
  As k gets big in KNN, the variance term declines but the bias term increases since we are averaging over more f(x) values.
- The variance term is simply the variance of an average here, and decreases as the inverse of k.
- Often prediction error (EPE) is a vehicle for getting at MSE, which tells how well we are estimating f(x), our real interest.
- Performance of OLS vs KNN in Bias Variance Terms
  - OLS: $EPE(X_0) \approx \sigma^2 \times \frac{P}{N}\sigma^2 + bias^2_{OLS}(x_0)$
    - How did we get P/N: $\hat{\beta} = (X^T X)^{-1} X^T y$ and $\hat{f} = X\hat{\beta}$. Say $Y \sim f(x) + \epsilon$ for $\epsilon \sim (0, \sigma^2)$. Treat X's in training sample as fixed, condition on those X's, then $\hat{f} = X\hat{\beta} = Ay$. $Cov(Y) = \sigma^2 I_n$ so $Cov(\hat{f}) = \sigma = A\sigma^2$. What is the avg variance of $\hat{f}_i$? $Var(\hat{f}_i) = tr(A)\sigma^2/n$ where $tr(A) = p =$ # of parameters. So this is close to P/N sigma squared
  - KNN: $EPE(X_0) \geq \sigma^2 + \sigma^2 + bias^2_{1-NN}(x_0)$
    - Sigma squared - both irreducible error and the error from the closest neighbor.
  - Concluding $\sigma^2 + P\sigma^2/N \approx \sigma^2 < 2\sigma^2$ - OLS better on variance. f(x) determines the bias, but for large N KNN could be optimal.

Chapter 3: Linear Methods for Regression

- Function in nature are generally not linear, so we should always think of our model as an approximation.
- RSE is divided by n-p-1 - we have already done some fitting, so we have reduced our degrees of freedom by p. This ensures RSE is unbiased estimate.
- Hat matrix is conditonal on X - assuming that x is not random. So when we do $Var(\hat{\beta}) = (X^T X)^{-1}\sigma^2$, $\hat{\beta} = My$, $Var(\hat{\beta}) = MVar(y)M^T = \sigma^2 MM^T = \sigma^2(X^T X)^{-1}$ since $Var(y) = \sigma^2 I_n$ and $M = (X^T X)^{-1} X^T$

Least Squares

- Basic model: $f(X) = \beta_0 + \sum_{j=1}^{p} X_j\beta_j$ - the model is linear in the parameters, but X can be a number of different forms
- Minimize RSS: $\mathrm{RSS}(\beta) = \sum_{i=1}^{N}(y_i - f(x_i))^2 = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$ and when X has full rank, $\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$
- The outcome vector y is orthogonally projected onto the hyperplane spanned by the input vectors x1 and x2. The projection $\hat{y}$ represents the vector of the least squares predictions: $\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} = Hy$

- The p predictors span a p-dimensional subspace of N space. Want to find an approximation of y that lies in that subspace - a linear combination of the columns of X. Necessarily the closest y is orthogonal to the subspace - $\hat{y} \perp$ subspace. Essentially $(y - \hat{y}) \perp x_j \forall j$ - e is perp to all x vectors that form a basis for the subspace. Could choose many betas that project onto the subspace - but the one with the smallest e is the least square solution.
- If two inputs perfectly correlated, X won't be full rank. The least square projection no longer works, but $\hat{y}$ is still the projection of y onto C(X), there is just more than one way to express the projection in terms of the X columns.
- To do more with this model, we need to assume the conditional expectation of Y is linear in $X_1, \ldots, X_p$ and deviations of Y around its expectation are additive and Gaussian - $\varepsilon \sim N\left(0, \sigma^2\right)$. Then $\hat{\beta} \sim N\left(\beta, \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \sigma^2\right)$
- F-Statisic: $F = \frac{(\mathrm{RSS}_0 - \mathrm{RSS}_1)/(p_1 - p_0)}{\mathrm{RSS}_1/(N - p_1 - 1)}$ measures the change in residual sum-of-squares per additional parameter in the bigger model, and it is normalized by an estimate of σ^2.
    - More general from F test: say $H_0 : \beta_{1,\ldots,p-q} \neq 0$ and last q = 0. Then $(RSS_0 - RSS/q)/(RSS/n - p) \sim F_{n-p}, q$ for RSS_0 the RSS for our subset model forcing the last q to 0. When q = 1, $F_{n-p,1} = t^2_{n-p}$ - equivalence between t and F tests. Then the p values can be interpreted as the effect you get leaving that variable out of the model and keeping the others.
    - Cannot simply remove the insignificant variables - if they are linearly dependent, then removing one might make the other significant.
- Properties of OLS
    - OLS is **equivariant** under non-singular linear transformations of X. If we make a transformation and solve we can undo the transformation to recover the solution for the original system. If $\hat{\beta}$ is OLS solution for X then $\beta^* = A^{-1}\hat{\beta}$ is OLS solution for $X^* = XA$ for A p x p nonsingular.
        - Many models are not equivariant - PCA is not equivariant. $z_1 = Xa_1$ - if we standardized the cols of X we get a solution but not recoverable if we didn't standardize the cols of X. Lasso and ridge are also not equivariant.
    - Given $\mathbf{z}_p = \mathbf{x}_p - \mathbf{X}_{(p)}\gamma$ for $\mathbf{X}_{(p)}$ submatrix of X exluding last columns for any gamma. Then OLS coef for x_p is the same as for z_p. This follows from equivariance. We have $X = (X_{(p)}, x_p)$, $\tilde{X} = (X_{(p)}, z_p)$. Regression on X and $\tilde{X}$, then x_p, z_p fill the same role in the regression.
        - Now let γ be the OLS coef of x_p on $X_{(p)}$, then z_p is the residual obtained by adjusting x_p for other variables in the model. z_p then orthogonal to $X_{(p)}$.
        - Why do we care? The coef of a variable picks up what is left off after removing the effect of others. The variance of a coefficient depends on the size of the norm of z_p. If the residual is small after adjusting for the other variables, the norm gets small and the variance of the coef blows up - highly uncertain when not much unexplained signal left in the residuals. This is true for any of the variates. The more correlated stuff you throw into your model, the less interpretable your model will be.

Assumptions for Inference

- Assume linearity, normality, constant variance and independent of errors with X's.
- Often the last is ignored - given some model $y_i = \beta_0 + \sum_j x_{ij}\beta_j + \left(\sum_k x^*_{ik}\theta_k + \varepsilon_i\right)$ we often just lump in the last ones with the error.
- Also the x's are considered fixed, but just an assumption of convenience - most of the time x's are random.
- **Bootstrap can help here**
- We assume additivity, but predictors usually change together - correlation in the data can confound interpretation. Can often end up with negative effect on one predictor when two correlated predictors are both positively correlated with the response. Can render a model uninterpretable

Model Performance

- Measure of estimator: $\mathrm{Mse}[\hat{\mathbf{f}}(x)] = \mathrm{E}[\hat{\mathbf{f}}(x) - \mathbf{f}(x)]^2 = \mathrm{Var}[\hat{\mathbf{f}}(x)] + [\mathrm{E}\hat{\mathbf{f}}(x) - \mathbf{f}(x)]^2$
- If linear model is correct, Gauss-Markov theorem applies and the LS prediction is unbiased and has the lowest variance among all unbiased estimators that are linear functions of y.
- Application: Time Series

- Fitting AR(k) model - k lags as predictors.
- We regress over many correlated variables, the t-stats are not significant, but prediction is our goal, not inference.

Gauss Markov Theorem

- Assumptions: linear model is correct, unbiased estimate, linear estimate. These are hard assumptions to live up to in practice. Always we can create a biased estimate with smaller MSE, James-Stein theorem says we can always improve an estimator's MSE through regularization / shrinkage.
- The least squares estimates of the parameters β have the smallest variance among all linear unbiased estimates.
- The Gauss-Markov theorem implies that the least squares estimator has the smallest mean squared error of all linear estimators with no bias. However, there may well exist a biased estimator with smaller mean squared error.

Multiple Regression

- From simple regression: $Y = X\beta + \varepsilon,\ \hat{\beta} = \dfrac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle},\ \mathbf{r} = \mathbf{y} - \mathbf{x}\hat{\beta}$ for residuals r.
- When the inputs of X are orthogonal, they have no effect on each other's parameter estimates in the model. Very rare in observational data, so need to orthogonalize them.
- Saying formally, X_1, X_2 mutually orthogonal, then $X_1^T(y - X\hat{\beta}) = X_1^T(y - X_1\hat{\beta}_1) = 0$ and same for β_2. You can do the multiple regression by doing the univaritate regression with orthogonal columns.
- Gram Schmidt for multiple regression
  - Initialize $z_0 = x_0 = 1$.
  - For $j \in 1, \ldots, p$, regress x_j onto $z_0, \ldots, z_j - 1$ to produce coefficients $\hat{\gamma}_{lj} = \langle \mathbf{z}_\ell, \mathbf{x}_j \rangle \,/\, \langle \mathbf{z}_\ell, \mathbf{z}_\ell \rangle$ for l in 0 to j-1 and residual vector $z_j = \mathbf{x}_j - \sum_{k=0}^{j-1} \hat{\gamma}_{kj}\mathbf{z}_k$. This procedure is equivalent to $\mathbf{X} = \mathbf{Z\Gamma} = \mathbf{ZD}^{-1}\mathbf{D\Gamma} = \mathbf{QR}$
  - Regress y on the residual z_p to get estimate $\hat{\beta}_p$
- The algorithm produces $\hat{\beta}_p = \frac{\langle \mathbf{z}_p, \mathbf{y} \rangle}{\langle \mathbf{z}_p, \mathbf{z}_p \rangle}$ Since the zj are all orthogonal, they form a basis for the column space of X, and hence the least squares projection onto this subspace is $\hat{y}$.
- The multiple regression coefficient β_j represents the additional contribution of xj on y, after xj has been adjusted for x0, x1,..., xj−1, xj+1,..., xp. If xp is highly correlated with some of the other xk's, the residual vector zp will be close to zero, and from (3.28) the coefficient $\hat{\beta}_p$ will be very unstable. See via this formula $\mathrm{Var}\left(\hat{\beta}_p\right) = \frac{\sigma^2}{\langle \mathbf{z}_p, \mathbf{z}_p \rangle} = \frac{\sigma^2}{\|\mathbf{z}_p\|^2}$

Regression via QR Decomposition

- Think of relation of Q and R. The first column of x is the first column of Q times a single number. Vector $q_1 = \frac{x_1}{||x_1||_2}$, the top element of R is $||x_1||_2$ to bring us back to X. The first p columns of X for a basis for the columns of X.
- The second column of X is a linear combination of the first two columns of Q - hence Q's second column has the elements of the first column substracted off.
- If X is not full rank, then first r columns of R look the same, but then we get a deficient triangle and get 0's towards the bottom of the triangle. Don't need the later columns in the linear combinations that produce X. This assumes that the columns are in a certain order that allow us to do this - we can shuffle the columns and pull the independent columns of X forward.
- With an L2 norm, can insert an orthogonal matrix inside of it and it won't change the norm: $||a||_2^2 = ||Qa||_2^2 \implies a^TQ^TQa = a^Ta$
- Using this fact, for the full rank X, $\|y - X\beta\|^2 = \left\|Q^Ty - R\beta\right\|^2 = \left\|Q_1^Ty - R_1\beta\right\|^2 + \left\|Q_2^Ty\right\|^2$. (Steps to get here: $X = QR \implies y = QR\beta \implies Q^Ty = Q^TQR\beta \implies y = R\beta$). Then $\hat{\beta} = R_1^{-1}Q_1^Ty$ from the first term leaving $\mathbf{RSS}(\hat{\beta}) = \left\|Q_2^Ty\right\|^2$
  - When you have a euclidean norm, you can always multiply inside on the left by an ortho matrix. Why? $||Z||^2 = Z^TZ \implies (QZ)^TQZ = Z^TZ$
  - R_1 above is p x p nonsingular. The inverse of triangular matrix is trivial.

- $e = Q^T y$ - coordinates of y on columns of Q - ie. in an orthonormal basis. $H = Q_1 Q_1^T$. For rank of X k < p, we solve $Q_1^T y = R_{11}\beta_1 + R_{12}\beta_2$ where Q_1 has r columns - this has infinite solutions.
- $\hat{y}$ is a projection of y into the columns space of X, so $\hat{y} = Q_1 Q_1^t y = Hy = X(X^T X)^{-1} X^T$ for projection matrix hat H. Clearly the orthornormal version of H is more convenient. When rank X is deficient, this is especially useful. Q_1 is the first r columns of Q, and all columns beyond that are going to be zero when multiplied by R. Then $Q_1^T y = R_{11}\beta_1 + R_{12}\beta_2$ has infinite solutions, need to set $\beta_2 = 0$ and solve for β_1, but this is an arbitrary choice. We could have chosen other solutions. We can do something less arbitrary - finding the β with the smallest norm (see Strang for min solution).
- Despite the non-uniqueness of the solution, the fit is well defined - we still are projecting onto the same subspace $Q_1 Q_1^T y$.

Distributional Aspects

- $\varepsilon \sim (0, \sigma^2)$ iid, though we can add normality. Given the X's (fixed), $Cov(\hat{\beta}) = (X^T X)^{-1}\sigma^2 = (R^T R)^{-1}\sigma^2$
- Adding the normal assumption for errors, we get the beta distribution $\hat{\beta} \sim N(\beta, (X^T X)^{-1}\sigma^2)$
- The effects e also normal $N(R\beta, \sigma^2 I)$ (since y was distributed $Q\beta$). Can break e into e1 and e2, wheree1 has mean $R_1\beta$ and e2 have mean zero since remaining R is zero. Then $||e_2||^2 \sim \sigma^2 \chi^2_{N-p}$
- Note $\sigma^2 = RSS/(n-p)$, denom to make unbiased. e1 distributed with a non zero mean, but under H0, $\beta = 0$, so e1 also chi distributed. e1 and e2 independent since uncorrelated Normals are independent. Then $\frac{\|e_1\|^2}{p} / \frac{\|e_2\|^2}{N-p} \sim F_{p,N-p}$ which we can use to test all the coefficients are simultaneously 0.

Subset Selection

- Prediction accuracy: the least squares estimates often have low bias but large variance, may be improved by taking on some bias via shrinkage and selection. Interpretation: With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects.
- Best / all subset regression finds for each $k \in \{0, 1, 2, \ldots, p\}$ the subset of size k that gives smallest residual sum of squares.
    - The null model has the full variance of the dataset - just predicted by the mean. The number of models increases around the middle of subset size k. The smaller k's have clear winners, but by middle k's the models are bunch up in performance. Good to note since the "ideal" subset is not particularly differentiated from other model choices.
- Forward stepwise selection starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit. Computational: for large p we cannot compute the best subset sequence, but we can always compute the forward stepwise sequence. Statistical: forward stepwise is a more constrained search, and will have lower variance, but perhaps more bias.
- Backward-stepwise selection starts with the full model, and sequentially deletes the predictor that has the least impact on the fit. The candidate for dropping is the variable with the smallest Z-score. Backward selection can only be used when N>p, while forward stepwise can always be used.
- Could view subset size a tuning parameter for all of these methods. For each size we have the "best" model, then pick the right size.
- Generally need to perform model assessment, could use validation and test sets. Validation use for choosing k, then test set for reporting the performance of the chosen model, since otherwise using one test/validation set includes some bias in selection. With insufficient data, we turn to CV.
- Forward-stagewise regression (FS) is even more constrained than forward stepwise regression. At each step the algorithm identifies the variable most correlated with the current residual. It then computes the simple linear regression coefficient of the residual on this chosen variable, and then adds it to the current coefficient for that variable. This is continued till none of the variables have correlation with the residuals (the least square fit when N > p). Unlike forward-stepwise regression, none of the other variables are adjusted when a term is added to the model. As a consequence, forward stagewise can take many more than p steps to reach the least squares fit - slow fitting.

Model Performance - CV + Bootstrap

- K-fold CV - divide the data into K equal parts (5,10), for each k fit the model with parameter λ to the other K-1 parts, computing the error in predicting the kth part. The total CV error: $CV(\lambda) = (1/K) \sum_{k=1}^{K} E_k(\lambda)$
    - For best subset selection, say, these subsets will be different. The subset of 3 fit to the k in 1,2,3 could be different from the one fit on 2,3,4. But the idea is we are finding the best subset number - not the actual make-up of the subset. The best model within a K may not be too far from another model.
    - Once we have selected the subset size, we go back to the full data set, fit the full sequence of models and choose the model for the tuned subset size. The actual model comes from the full data
    - We tend to prefer 10 to 5 for K for this reason - fitting to 9/10 of the data is closer to fitting to your full training set.
    - Could go all the way with LOOCV, but computationally difficult (at least for best subset, though not necessarily for something like ridge). LOOCV could also have too high variance compared to K-fold, but the best method depends on the goal and situation.
- Boostrap Approach
    - Can use bootstrap for prediction error estimation. We get a full size training sample since we sample with replacement.
    - For each bootstrap sample, estimate errors using only observations excluded from the boostrap sample.
    - Pretends the training sample is the population distribution, draws a sample, and the population is the full training sample.
    - Can be much more computationally expensive, but often can have slightly better results
- CV Issues
    - Example: 1) Have 5000 predictors, find 200 predictors having the largest correlation with class labels. 2) Then carry about nearest centroid classification using those 200 predictors
        - Wrong approach is to apply CV to step 2 - need to apply CV to 1 and 2.
    - Need a CV and test set - if we run CV and get a CV error, it will be slightly biased downwards. If our model has many tuning parameters, we will have to CV multiple times - we have overfit the data for fitting the models and for CV. We need to report our methods error on the test set, a dataset reserved for final evaluation, and of course this could still overreport performance since future datasets may not be the same as today's.
    - If we have little data, a separate test set may be expensive. Could instead use nested CV, using say K-fold, select lambda and train model using selected lambda on the 9/10, test on the 1/10. Grows exponentially in computation needed
    - Example: Does CV really work?
        - Null situation, N = 20 p = 500, where labels independent of data - expect 50% error rate since no signal
        - Take a stump - univaritate classifier that is a single split minimizes the misclassification error. Among all features, pick the feature and best split point to minimize error by a single division.
        - Fit to entire training set, by chance we can find a split that fits data well. In 5-fold CV expect good separation on this feature too. Does this mean CV is useless?
        - With very high number of noise variables, these things will happen by chance. Running this process many times, on average get 50% misclassification but the error rate has high variance - from 0.2 to 0.8. Have to realize with small data can have huge variance, but CV does not produce biased estimate.
        - In practice, just have one number, the CV error, not the full variance of the CV error. In practice, often use variance from the 10 folds but this tends to be too optimistic. Somewhat of an open question about how to improve on that. The ten folds are independent but the estimates are correlated since they rely on overlapping observations - we have bias in our estimates.
- 1 SE Rule - find model with the CV minimum, then take the simplest model within the 1 SE bounds of this best model.

Shrinkage

Ridge Regression

- $\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$ or equivalently
  $\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$ st $\sum_{j=1}^{p} \beta_j^2 \leq t$
- Here $\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage: the larger the value of λ, the greater the amount of shrinkage.
- Ridge solutions dependent on scale - should standardize inputs! Note β_0 is not in penalty term, otherwise adding a constant to Y would affect results non linearly. Easiest way is to center variables - remove mean from X and y.
- In matrix notation, take centered inputs $x_{ij} - \bar{x}_j$ less the constant. Using centered X,
  $\text{RSS}(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta$ and the solutions are $\hat{\beta}^{\text{ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$
  - In this form we can simply take the gradient, $-2X^T(y - X\beta) + 2\lambda\beta = 0$ and solve.
- There is always a value of λ that reduces the MSE more than the OLS estimator.
  - $y \sim X\beta + \varepsilon$, then some value of lambda > 0 minimizes $E||\beta - \hat{\beta}_\lambda||_2^2$
- Note by adding a constant to X^TX, it is always non-singular and is always invertible. In the case of orthonormal inputs, ridge edstimates are just scaled LS estimates $\hat{\beta}^{\text{ridge}} = \hat{\beta}/(1 + \lambda)$
- Effective degrees of freedom: $\text{df}(\lambda) = \text{tr}\left[\mathbf{X}\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\right] = \text{tr}(\mathbf{H}_\lambda) = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}$ for singular values d.
  - When lambda is 0, the df is the number of predictors in the model. When lambda infinity, all the coefficients are 0, so df=0.
  - In the linear model, we get $\hat{y} = Hy$ where H is a projection operator. The trace of H gives you the number of parameters in the model. Use similar idea for df here - $df(\lambda) = tr(H_\lambda)$

Lasso

- $\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$ or equivalently
  $\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$ st $\sum_{j=1}^{p} |\beta_j| \leq t$
- Small t -> coefficients shrink to 0, form of subset selection. Larger t, end up with LS estimates.
- Acts in a nonlinear manner on the outcome unlike ridge. It sets some coefficients to 0 for large enough penalty - acts in between subset selection and ridge regularization. Can think of it as a convex relaxation of the subset size constraint.
- Can think of ridge as the OLS line with a smaller slope, The lasso is the OLS line with a parallel shift downwards, setting negative values to zero - kinked.
- For ridge and lasso imporant to standardize predictors to mean zero and SD 1. If all predictors are in the same units, could just center but not scale.
- Degrees of freedom for lasso is harder. Can use the more general definition $df(\hat{y}) = \sum_i Cov(y_i, \hat{y}_i/\sigma^2)$ - average covariance over a point in the data and its fitted value
  - Covariance on repeated training samples. If a method overfits, prediction closer to y in data and covariance is high. If we use the null model then covariance will be quite small.
  - For ridge regression, have y and $\hat{y}_\lambda = H_\lambda y$, then want the $Cov(\hat{y}_\lambda, y) = Cov(H_\lambda y, y) = \sigma^2 H_\lambda$ for $y \sim (f, \sigma^2 I)$. So the sum of the covariances is $Tr(\sigma^2 H_\lambda)$, which is what we found before, so this is a generalization of the ridge dfs
  - Take size of the active set (# of parameters fit in lasso / size of subset) $\hat{k}_\lambda$ is unbiased in the df - $E(\hat{k}_\lambda) = df(\lambda)$
  - This is not true for best subset, searched over a state space. In Lasso, we not only get a subset but we also get shrinkage in the coefficients. This is enough to bring the degrees of freedom down to k.
- Notice in the lasso coefficient graph - coefficients start growing at different times. In ridge, all coefficients started growing once we alter the shrinkage factor.

- Homotopy path of the lasso is piecewise linear. In between where active set changes get a piecewise linear function, notice knots on graph. When the set changes, all of the coefficients change across variables in the model. This means you can compute the whole lasso path in the same time as the least squares fit - lars algorithm. We know the order in which to add predictors.

Regression Restriction Comparisons

- Ridge regression does a proportional shrinkage. Lasso translates each coefficient by a constant factor λ, truncating at zero. This is called “soft thresholding”
- Both ridge and lasso, centering allows us to forget the intercept.
- Both methods find the first point where the elliptical contours hit the constraint region. Unlike the disk, the diamond has corners; if the solution occurs at a corner, then it has one parameter β_j equal to zero.
- Family of Shrinkage Estimators: Consider penalty $\lambda \sum_{j=1}^{p} |\beta_j|^q$; each $|\beta_j|^q$ can be a log-prior density for β_j. The lasso, ridge regression and best subset selection are Bayes estimates with different priors.
- This generic penalty constructs different constraint contours, but little values for q > 2 and between 0 and 1 becomes non convex.
- Elastic-net Penalty - compromise between ridge and lasso: $\lambda \sum_{j=1}^{p} \left(\alpha\beta_j^2 + (1-\alpha)\,|\beta_j|\right)$. Has constraint contour like lasso with slightly rounded edges. Of course you get another parameter alpha, but often don’t need a fine search over it like you do for lambda.
- If you do ridge with identical predictors, they end up sharing the coefficient equally. Lasso does not - two variables identical in the model, could split a coefficient any number of ways, ambivalent about the linear combination. If you have predictors in groups, and want the group to be selected, elastic-net removes some of the arbitrariness of lasso selection among correlated groups.

Derived Input Direction Methods

PC Regression

- Principal components depend on the scaling of the inputs, so typically we first standardize them.
- For X data matrix and $\tilde{X}$ the data matrix with centered columns. The largest PC direction v max’s $\hat{Var}(Xv) = \frac{1}{N} v^T \tilde{X}^T \tilde{X} v$ subject to $||v||_2 = 1$. Then $z_1 = \tilde{X} v_1$ is the largest eigenvector of the coviance matrix. $z_2 = \tilde{X} v$ where we want the variance between z1 and z2 to be zero. So take $\frac{1}{N} v^T \tilde{X}^T \tilde{X} v = \frac{d^2}{n_1} v_1^T v = 0$. Do the full eigendecomposition and get the PCs. All of the PCs defined by eigenvector equations $\frac{1}{N}\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{v}_j = d_j^2 \mathbf{v}_j, \quad j = 1, \ldots, p$. Also note that the SVD of the centered X provides the PCs of X.
- PCR - regress y on z1,...zj. Since all zj are orthogonal, it is just a sum of univariate regressions.
- If M = p, we would just get back the usual least squares estimates; for M<p we get a reduced regression.
- Implicit assumption that the response will change more in the direction that the x’s change most.
- We see that principal components regression is very similar to ridge regression: both operate via the principal components of the input matrix. Ridge regression shrinks the coefficients of the principal components, shrinking more depending on the size of the corresponding eigenvalue; principal components regression discards the p − M smallest eigenvalue components.
    - Ridge: for $\tilde{X} = UDV^T,\ Z = \tilde{X}V = UD$ - the left singular vectors scaled by the singular values gives you Z. We are regressing on Z, so scaling by D does not matter, so can just use U, an orthonormal basis. So $\hat{y}_\lambda = \sum_{j=1}^{p} u_j \frac{d_j^2}{d_j^2+\lambda} \langle u_j, y \rangle$ - gives more weight to leading PCs but then shrinkage increases. Bigger lambda, shrinkage squeezes more.
    - For PCR: $\hat{y} = \sum_{j=1}^{k} u_j \langle u_j, y \rangle$ - gives weight 1 to the first k components, weight 0 to the rest. As we shrink k, number of PCs included, more will drop out.

Partial Least Squares

- Also depends on the scaling of the inputs, so typically we first standardize them to mean 0, var 1

- Akin to PCR but takes the response into account instead of using an unsupervised preprocessing. Fit univariate coefficients then construct z = sum of coefs times x_i. This is the first z, z1. Then regression of y on z1, to get coef beta1. Then orthogonalize y and x's wrt z1 - nothing left of z1 in any portion of the data. Then repeat the process.
- As with principal-component regression, if we were to construct all M = p directions, we would get back a solution equivalent to the usual least squares estimates; using $M < p$ directions produces a reduced regression.
- It can be shown that partial least squares seeks directions that have high variance and have high correlation with the response, but typically the variance dominates making it close to PCR.
- If the input matrix X is orthogonal, then partial least squares finds the least squares estimates after m = 1 steps.

Comparison of Selection and Shrinkage

- For minimizing prediction error, ridge regression is generally preferable to variable subset selection, principal components regression and partial least squares.
- PLS, PCR and ridge regression tend to behave similarly. Ridge regression may be preferred because it shrinks smoothly, rather than in discrete steps. Lasso falls somewhere between ridge regression and best subset regression, and enjoys some of the properties of each.

Chapter 4: Linear Methods for Classification

- Logit transformation: $\log \frac{\Pr(G=1|X=x)}{\Pr(G=2|X=x)} = \beta_0 + \beta^T x$

Linear Regression of Indicator Matrix

- $\hat{\mathbf{Y}} = \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$. Could view it as estimate of conditional expectation, but can exceed the domain of probability densities.
- Response variable g that takes values in an unordered set of size k. We encode the response in an indicator matrix Y n x k - one hot encoding. We regress this matrix of indicators on X. For the kth column get a linear model $f_k(x) = \beta_{k0} + \beta_k^T x \quad k = 1, \ldots, K$
- The decision boundaries - get $\binom{k}{2}$ boundaries where $\{x : f_k(x) = f_l(x)\}$.
- We fit $min_B||Y - XB||_F = min \sum_{k=1}^{K} ||y_k - X\beta_k||_2^2$ - have a coefficient for each of the responses instead of a coefficient vector.
- Note $E(Y|X = x)$ for a given one hot encoding Y for a class is the vector of conditional probabilities across possible classes.
- We could construct targets for each class t_k, and try to reproduce the correct target for an observation fit via OLS.
  - Could write the frobenius norm also as $\sum_{i=1}^{N} ||y_i = B^T x_i||_2^2$
  - Classifying to closest target equivalent to highest probability for class
- As number of classes K increases, classes can be masked by others - linear model misses linear boundaries completely. Falls out as a curiosity of using least squares and a linear model for this problem.
  - Reduce this problem down to 1 dimension by projecting on a line through the three classes. The rug plot shows the fitted regression line relative to that line. The green and orange points have lines with non zero slopes - the likelihood varies. But the middle blue class has a line without slope and never is the most likely. The problem goes away if we fit quadratic functions, but we shouldn't need quadratics because something like LDA can fit easily with linear functions. This is a problem as in higher dimensions we may not see we have this problem at all.
  - We shouldn't have to introduce non-linearity to remove an artifact from our method - best to use a method without an artifact at all.
- A loose but general rule is that if K ≥ 3 classes are lined up, polynomial terms up to degree K − 1 might be needed to resolve them.

Linear Discriminant Analysis

- From Bayes' Theorem, $\Pr(G=k|X=x) = \frac{f_k(x)\pi_k}{\sum_{\ell=1}^K f_\ell(x)\pi_\ell}$
- $f_k(x)$ is our density when G = k. $\pi_k = Pr(G=k)$ - the marginal probability that G equals k. If we know both of these for 1,...,k, then we know everything about the joint distribution between G and X.
  $P_X(G=k) = P_X(X=x|G=k)P(G=k) = f_k(x)\pi_k$
- Then by Bayes, $P(G_k|X=x) = \frac{P_X(G=k,X=x)}{P(X=x)} = \frac{f_k(x)\pi_k}{\sum_{l=1}^k f_l(x)\pi_l}$ where the marginal is just summing over all k to get rid of it from the joint. However the density estimation is hard, and in high dimensions it is especially hard. Generative models - if you have the formula for the densities easy to generate the boundaries.
- Take the log ratio of the conditional probabilties - if this log ratio is 0, then we have equal probability and determines the decision boundary. Think of two normal bells overlapping - we take the boundary as the intersection of the densities - but notice that some will be misclassified where the density of one slips below the other.
- Suppose we model each class density as multivariate Gaussian. Linear discriminant analysis (LDA) arises in the special case when we assume that the classes have a common covariance matrix $\Sigma_k = \Sigma\ \forall k$.
- MVN: $f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)}$
- In comparing two classes k and l, it is sufficient to look at the log-ratio, and we see that
  $\log\frac{\Pr(G=k|X=x)}{\Pr(G=\ell|X=x)} = \log\frac{\pi_k}{\pi_\ell} - \frac{1}{2}(\mu_k+\mu_\ell)^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_\ell) \quad + x^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_\ell) = \delta_k(x) - \delta_\ell(x)$ where
  $\delta_k(x) = \log\pi_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + x^T\Sigma^{-1}\mu_k$. The convenient assumptions give us a nice linear form for the function.
- Linear discriminant functions: $\delta_k(x) = x^T\mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}^{-1}\mu_k + \log\pi_k$, equivalent to the decision rule using $G(x) = \text{argmax}_k\,\delta_k(x)$
- This is the population model, but obviously we need to plug in estimates from our data. The covariance matrix needs to be invertible, then we multiply by μ_k and get linear coefficients for x. Have a sample $x_i, g_i,\ i \in 1,\ldots,n$, for the covariance matrix need to take into account that it is the same estimate across all classes. Say
  $S_k = \frac{1}{n_k}\sum_{g_i=k}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^T$ for n_k obs in class k. Then $\hat{\Sigma} = \sum_{k=1}^K \pi_k S_k$ - pooled within class covariance matrix.
- Need to estimate the Gaussian parameters from the training data since they are unknown.
  - $\hat{\pi}_k = N_k/N$
  - $\hat{\mu}_k = \sum_{g_i=k} x_i/N_k$
  - $\hat{\mathbf{\Sigma}} = \sum_{k=1}^K\sum_{g_i=k}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^T/(N-K)$
- Choose the cut-point that empirically minimizes training error for a given dataset. For two classes, we can just subtract one discriminant function from the other and categorize based on positive or negative value, eg classify to class 2 if $x^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) > \frac{1}{2}(\hat{\mu}_2+\hat{\mu}_1)^T\hat{\Sigma}^{-1}(\hat{\mu}_2-\hat{\mu}_1) - \log\frac{N_2}{N_1}$. Note that the discriminant direction minimizes overlap in distribution for Gaussian data - skew projection, see that the density overlap is minimized in this direction. The $\hat{\mu}_2-\hat{\mu}_1$ is a point between the two mean vectors then modulated by the relative prize $\log\frac{N_2}{N_1}$ - if one class is much bigger than the other, want to scale the density to favor the bigger class and minimize total number of misclassified data. The presence of Σ makes it a skew projection instead of a standard projection.
- With more than two classes, LDA is not the same as linear regression of the class indicator matrix, and it avoids the masking problems associated with that approach
- We can enrich the LDA - augment the two variables with their squares and cross product - go from $\mathbb{R}^2 \to \mathbb{R}^5$. You get some weird correlation and you can project the linear $\mathbb{R}^5$ boundaries back down to $\mathbb{R}^2$ to get quadratic boundaries. They can perform pretty well. Of course, not really necessary since we can turn to QDA with more flexible assumptions.
- Without assumption of equal variance matrices, get QDA when the quadratic terms no longer cancel between discriminant functions. The estimates for QDA are similar to those for LDA, except that separate covariance matrices must be estimated for each class. When p is large this can mean a dramatic increase in parameters.

- Doing it with augmented LDA vs QDA - get very similar boundaries, or at very least similar misclassification rates. Generally look at the marginals of the variable, do they look roughly Gaussian, probably can fit this model. Otherwise perhaps look for other methods or transform the data to look more Gaussian (of course remember the transformation come test time) - Gaussiam copula model.

LDA computations

- Diagonalize $\hat{\mathbf{\Sigma}}$ or $\hat{\mathbf{\Sigma}}_k$, using eigendecomposition $\hat{\mathbf{\Sigma}}_k = \mathbf{U}_k \mathbf{D}_k \mathbf{U}_k^T$

Regularized DA

- With big number of parameters, problematic to estimate covariance matrices
- QDA: shrink the separate covariances towards the common covariance matrix by parameter α. For $\alpha = 1$ get QDA and for $\alpha = 0$ get LDA.
- May even be $\hat{\Sigma}$ is still too hard just for LDA. Shrink the pool covariance matrix towards a scalar covariance matrix - a scaled version of I, can only do if vars are in the same units. γ parameter determines level of shrinkage, like a ridge version of DA.
- Vowel data - the full $\alpha = 1$ has bigger misclassification rate, but just below 1 we get best performance. Regularization solving some instability in using separate covariance matrices.

Reduced Rank LDA

- The K centroids in p-dimensional input space lie in an affine subspace of dimension ≤ K − 1, and if p is much larger than K, this will be a considerable drop in dimension.
- Project X^* onto the centroid spanning subspace since we only care about relative distances to centroids - we need only consider data in a subspace of dimension at most K-1
- We could make subspace even smaller by finding the PCs of the centroids themselves. Say we have three centroids that are nearly collinear - instead of projecting into $\mathbb{R}^3$, we aren't going to lose much if we project onto $\mathbb{R}^1$ instead.
- We see in the example, in the first few coordinates there is good class separation and not much beyond that - looking at 9 vs 10 is quite messy. Then we can project into that smaller space.
- Fisher: Find the linear combination $Z = a^T X$ st the between class variance is maximized relative to the within class variance. The between class variance is the variance of the class means of Z, and the within class variance is the pooled variance about the means.
- For W, common covariance matrix (within class covariance), the between-class variance of Z is $a^T \mathbf{B} a$ and within class variance is $a^T \mathbf{W} a$ where $B + W = T$, total covariance matrix of X. Fisher's problem amounts to maximizing the Rayleigh quotient, $\max_a \frac{a^T \mathbf{B} a}{a^T \mathbf{W} a}$
- B is defined as $B = M^T D_\pi M$, $M_{(K \times p)} = \begin{bmatrix} \bar{x}_1^T \\ \bar{x}_2^T \\ \vdots \\ \bar{x}_K^T \end{bmatrix}$, D diagonal matrix of π_l. W = the pooled variance, assumed constant for the classes. Want between to be big and within to be small.
- Reformulated as $max\, a^T B a$ st $a^T W a = 1$ - becomes a generalized eigenvalue problem. $a^* = W^{1/2} a,\ a^{*T} a^* = 1$. So $a^T B a \rightarrow a^{*T} W^{-1/2} B W^{-1/2} a^* = a^{*T} B a^*$
- Dimension reduction allows visualization and can improve classification performance. Do dimension reduction, perform LDA in this space. Training error may not be monotonic since we are not optimizing on this metric - we are focused on B and W.
- Null error rate - classify via the distribution of the priors. For 11 class vowel data, this is over 90%
- Looking at the vowel task, linear regression performs worse likely due to the masking of the classes.
- While old school, this is still useful in Gaussian estimation processes, simplifies classification once we have started modeling using Gaussians
- We can also generalize this idea of the Rayleigh quotient for the problem at hand. Some measure that combines a dual objective - punish one metric and reward another.

Comparing DA Methods

- Say we have two classes, look at P(G=1|x) vs x. Can have a horrible bumpy function, but all we really want to know is where is it above 0.5. Estimating this with a smooth function will be highly biased - but could have the same classifications since below or above 0.5 is all that matters for this classification.

Example: Classification of Microarray Samples

- Four varieties of tumors - BL, EWS, NB, RMS - 4 classes. Small dataset but 2300 genes, very wide data. LDA not possible, pooled covariance is 2300 x 2300 - rank is very low.
- Row is gene, column sample, grouped by classes. We can see from the image there is some grouping within classes. The paper fit a NN, and way too many parameters. Used bagging on the NN which is why there are many learning curves. Ended with 0 training and test error. They fit the network on a projection onto PCs - Hastie got same performance with ridge logistic regression - similarly depends on PCs.
- Class centroid plot - centered the classes. Then with shrinkage, the blue lines are centroids that remained in the model for genes, while genes without blue shrunk out of model

Shrunken Centroids

- Have within class centroids and overall centroid
- For each class and component, take the deviation between component for kth class less the overall mean for variable j standardized by sj, the pooled variance for that gene.
- Shrinkage is soft-thresholding - just like Lasso. If positive, shrink it down by an amount delta, if negative shrink it up by delta. But if it crosses zero in the transform, it is set to 0.
- For a test sample x^* with p components, the discriminant score for class k $\delta_k\left(x^*\right)=\log \pi_k-\frac{1}{2} \sum_{j=1}^p \frac{\left(x_j^*-\bar{x}_{j k}^{\prime}\right)^2}{s_j^2}$ then map to class with largest discriminant function. If for variable j, all shrunk to the overall centroid, then we can throw variable j out of the model since it contributes nothing new to the model.
- As shrinkage increases on genes, get better performance until a threshold where the genes really matter, then error shoots up with more shrinkage.
- The shrinkage also highlights the important genes, those that survived in the model. Interestingly, there were different sets of genes that survived for each of the classes.
- The ridge model worked very well, but a little inconvenient bc did not perform feature selection. This method focuses more on simplicity with feature selection.

Logistic Regression

- Intercept tends to be bundled into X. Logit nice to squeeze probabilities between 0 and 1, but also has nice properties for certain applications
- Conditioning on the X's, so the loglikelihood just works with the randomness in the Y's. Y = 1 or 0 sets one term in the sum to zero. Non-linear in beta, but is convex, so we can use a Newton algorithm to optimize. This is equivalent to IRLS - iteratively reweighted least squares.
- IRLS - Initialize beta, set all but intercept to 0, set linearezed responses (residual on the y scale). We repeat weighted linear regression to get all of the betas
- Historically, took empirical probability r_i, could take $log\frac{r_1}{1-r_i}$ - empirical logits -> linear regression. Cannot optimize so took the taylor series approximation for $log(y_i)$
- What do the weights do - similar to the δ-method for variance approx. Given RV z and $var(z)=\sigma^2$. Say we want var(f(z)) for some transformation. Do some TS approx of f(z) around the mean of z - $f(z) \approx f(\mu_z)+f'(\mu_z)(z-\mu_z)+\ldots$ so $Var(f(z)) \approx [f'(\mu_z)]^2 var(z)$
- So the variance of logit of y $=p(1-p)\frac{1}{[p(1-p)]^2}$ - so we are in effect weighting by the inverse of the variance.
- Newton Method: $l(p)=\sum_{i=1}^n y_i log\frac{p_i}{1-p_i}+log(1-p_i)=y^T X\beta-log(1+e^{X^T\beta})$ since $(1-p(x_i))=\frac{1}{1+e^{X^T\beta}}$. How we get to the gradient of l(p) wrt beta. Hessian is a diagonal matrix with elements $w_i=p_i\left(1-p_i\right)$.
- Score equations - since x has a 1's columns in there, the sum of y's = sum of p's there.
- Asymptotic $Cov(\hat{\beta})=(X^T W X)^{-1}$. From asymptotic theory of MLE, relies on fact that the model is correct though - need a linear model for this to be true.
- When the Newton algorithm converges, it is an approximation of the asymptotic log likelihood and we can just read

off the statistics from there.
- If the 2 classes are linearly separable, solution is undefined - MLE tries to achieve probabilties of 0,1 and for this some beta must go to pos/min infinity. The point where you start bending $log(\frac{p}{1-p})\beta_o + \beta_1 x$. Slope becomes infinite for the bend. If nearly separable, the variance explodes for the coefficients.
- Called a forward or genertive model vs LDA which is backwards or discriminative.
- Inference is similar to linear regression. The SEs are from the weighted least square procedure, the z-score is treating this as a Gaussian and doing a normal t-test. Except here there is no global σ^2
- Deviance: $dev(y, \hat{p}) = -2l(\hat{\beta})$, would be the RSS in the linear model.
- Null hypothesis: first q components of beta are non-zero. Then $\mathrm{dev}(y, \hat{p}_0) - \mathrm{dev}(y, \hat{p}_1) \sim \chi^2_{p-q}$ and we get chi-square distributions instead of F-distributions.
- The Chi-Square statistic $\sum_{i=1}^n w_i \left(z_i - x_i^T \hat{\beta}\right) = \sum_{i=1}^n \frac{(y_i - \hat{p}_i)^2}{\hat{p}_i(1-\hat{p}_i)}$ - squared residuals divided by the variance. Can get there from the delta method and the residuals.
- If you don't trust these asymptotics, can simply use the bootstrap to get a better approximaton of the true variance.

Case Control Sampling

- Case control intercept correction - add in log odds of the true proportion in the population less the log odds of the sample ratio. Sampling more controls than cases reduces the variance of the parameter estimates up to a point of 5/1 where the reduction drops off. Useful for add click through rates that are very small - can just take a null sample 10:1 and not worry about taking more than that.
- Disease D we are targeting. Assuming logistic model is correct. We go out and sample separately from disease and non-disease groups. Z is indicator that you are in the sample - given disease, the probability of being in the sample is independent - doesn't depend on x, can just be a static π_0
- See odds ratios for similar types of work.

Multiple Logistic Regression

- Make one class a base class (arbitrary) and take log odds of each class against the base. Model j - 1 logits (since the jth will be the base class vs the base class and equal 1).
- Changing the base classes - MLE unchanged, change the coefficients by adjusting
- With p >> n, linear models often sufficient - often still too flexible and we need to regularize with ridge or lasso penalty. Since trying to maximize log likelihood we subtract off the penalty.
- With p > n, the data is separable, so we have a problem fitting as we had before. Adding a positive lambda ensures a solution exists, so regularization is simply necessary for finding a solution.
- With penalties for the two class case, can use IRLS algo for ridge and LARS similar algo for Lasso. Modular algorithms that can fit a large variety of models.
- With penalties, there is symmetric representation of the model instead of setting base class to 0 - end up with the softmax $P(G = j|x) = e^{\eta_j(x)} / \sum_{\ell=1}^K e^{\eta_\ell(x)}$ for $\eta_j(x) \sim \log P(G = j|x) = x^T \beta_j$
- The mean minimizes the sum of squares - so with a quadratic ridge penalty, we subtract off the mean from each β_j. For ℓ_1 we use the median

Risk Estimates

- Risk for new observation $\hat{\eta}\left(x_0\right) = x_0^T \hat{\beta}$. and $\hat{\Pr}\left(Y = 1 | X = x_0\right) = e^{\hat{\eta}(x_0)} / \left(1 + e^{\hat{\eta}(x_0)}\right)$.
- To classify new obs, threshold at 0.5, but in practice we adjust using ROC. ROC is conducted on the test data set, FP vs TP graph
- Sensitivity : TP rate, Specificity: TN rate = 1- FP rate
- AUC measures the performance without any indication of how the classifier will be used. You could use partial AUC if there is a range of FP rates you will tolerate.

Name	Definition	Synonyms
False Pos. Rate	FP/N	Type I Error, $1-$ Specificity
True Pos. Rate	TP/P	$1-$ Type II Error, Power, Sensitivity, Recall
Pos. Pred. Value	TP/P'	
Neg. Pred. Value	TN/N'	

- F measure combines precision and recall, harmonic mean: $F = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Naive Bayes

- If your goal is classification, you can put up with a lot of bias. Care only about placing in correct class. This has high bias but we get to have low variance in return.
- Naive Bayes $logit \frac{P(y=1x)}{P(y=0|x)} = \alpha + \sum_{j=1}^{p} \beta_j(x_i)$ to GAM. The Naive Bayes is actually producing a GAM, though the fitting is done differently. Naive Bayes uses full likelihood with conditional independence assumption.

Separating Hyperplanes

- Direct approach to classification. Perceptron learning algorithm - would find a separator then stop. Minimization criteria - if misclassified it is positive, negative for correct classification. Uses gradient descent to optimize the loss.
- Geometry - any two x on the line, then $\beta^T(x_2 - x_1) = 0$ so β is a normal to the line, since dot product is zero. Then the signed distance of a point to the line is projected onto the normal vector
- Optimal separating hyperplane - $y_i \left(x_i^T \beta + \beta_0\right) \geq C$ multiplying by y removes the sign and we are left with just the distance to the hyperplane, and we want to maximize it. The optimal solution for β is a linear combination of only the support vectors on the boundary / margin

Coordinate Descent for the Lasso

- $\frac{1}{2N} \sum (r_{ij} - x_{ij}\beta_j)^2 + \lambda|\beta_j|$
- Then add and subtract OLS beta: $(r_{ij} - x_{ij}\beta_j^* + x_{ij}\beta_j^* - x_{ij}\beta_j)^2$ - complete the square
- Cross term: OLS residual in parens $(r_{ij} - x_{ij}\beta_j^*)x_{ij}(\beta_j^* - \beta_j) = 0$ since least squares residual is orthogonal to the x.
- So get $C + 1/2(\beta_j^* - \beta_j) + \lambda|\beta_j|$.
- Suppose $\beta_j > 0$. Then differentiating $-(\beta_j^* - \beta_j) + \lambda = 0$. If $\beta_j < 0$ flip the sign on $lambda$: $-(\beta_j^* - \beta_j) - \lambda = 0$.
- So $\beta_j = \beta_j^* - \lambda$ if $\beta_j > 0$, else $\beta_j = \beta_j^* + \lambda$. Notice this only works if $|\beta_j^*| > \lambda$. This is the process of soft-thresholding.
- We can do something similar for elastic net
- λ max - the smallest value of lambda for which all coefficients are still zero is lasso. It's the biggest β_j in the regression. The first coefficient to survive the thresholding will be the biggest coefficient since we need $|\beta_j^*| > \lambda$. In elastic net, if alpha is 0 and we have pure ridge, this lambda is infinity.
- All variables in the active set will have inner products between x and residuals equal to lambda. All those out of the active set are less than lambda.

Chapter 5: Basis Functions

- Controling complexity of the model - restriction methods limit the class of functions considered (such as additivity), Selection methods - scan possible functions and only use basis functions that contribute significantly to fit of model (CART, boosting), Regularization methods - scan all functions but restrict the coefficients
- Basis expansions, could be polynomials, radial function ||X||, log transform
- Indicator basis function $h_m(X) = I\left(L_m \leq X_k < U_m\right)$ for a domain. This fits a piecewise constant function over the domain. Used to be a popular way of fitting non linear functions, just throw some knots in there.
- If we use a large expansion, we need to include some regularization.

Piecewise Polynomials and Splines

- Dividing the domain of X into contiguous intervals, and representing f by a separate polynomial in each interval.
- Understanding piecewise knots - adjusted by ξ_i, forces value to 0 until X = ξ_i. Starts off at zero at the knot then from the knot onwards changes the slope of the linear function. Since added to the rest of the linear model is just a slope adjustment that doesn't kick in until you hit the knot. Continuity constrain enacted in this way to get the spline.
- Parameter count: regions x parameters per region - knots x contraints per knot. For example 3 regions x 4 parameters per region - 2 knots x 3 constraints per knot on a 2 knot, cubic spline fit on cubic model of X (4 params)
- More generally, an order-M spline with knots ξj , j = 1,...,K is a piecewise-polynomial of order M, and has continuous derivatives up to order M − 2.
- These fixed-knot splines are also known as regression splines. One needs to select the order of the spline, the number of knots and their placement.
- In the cubic case, the basis function is zero left of the knot, and adds a cubic beyond the knot. Continuous at the knot, but looking at the first derivative $3(x-\xi_1)_+^2$ still continuous across 0 and non zero parts. Second derivative $6(x-\xi_1)_+$ - still continuous. Third derivative = 6, not continuous from left 0 portion.
- A natural cubic spline adds additional constraints, namely that the function is linear beyond the boundary knots. This frees up four degrees of freedom (two constraints each in both boundary regions), which can be spent more profitably by sprinkling more knots in the interior region. Bias on the boundaries but linear approximation is not bad for most fits.
    - There are separate boundary knots, beyond these knots the function is linear. Still have continuity at the boundary knot. We get two constraints in each of the boundary regions, reducing the number parameters by 4.
    - Since the boundary knots decrease the number of parameters, we can fit more interior knots to match the df of the cubic spline for comparison.
    - For $f(x) = \sum_{j=1}^m h_j(x)\theta_j$ and have an additive error model $y = f(x) + \epsilon$ iid constant variance. Fit via least squares get $\hat{\theta} = (H^TH)^{-1}H^Ty$, $Var(\hat{\theta}) = (H^TH)^{-1}\sigma^2$. Evaluate you basis functions at that point x and make a prediction. This gives us the pointwise variance of the fitted function. Comparison of variances - cubic spline has high variance in tails, little in the middle. Natural cubic spline has somewhat higher variance in the middle due to more knots but lower variance in the tails.
- B Splines - made in recursive fashion, start as piecewise constant then iterated to raise the degree. (An order 4 polynomial is a degree 3 polynomial). Taking $h_1(x), h_2(x), \dots h_7(x)$, but returns N x 6 matrix to remove intercept. If we have x1, x2 and want a cubic spline in both. If we included intercept, there would be some arbitrariness. Instead of specifying knots we can specify degree of freedom, and knots will be dispersed to quantiles. `lm( y ~ bs(x_1, df=4) + bs(x_2, df=4))`. Not really perferred to natural splines though.
- Can get pointwise standard errors around each point for a natural spline. Certainly would not want to extrapolate the spline prediction outside of the fitted data range, would be a worse fit than a linear model.

Smoothing Splines

- $\text{RSS}(f,\lambda) = \sum_{i=1}^N \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 dt$
- The first term measures closeness to the data, while the second term penalizes curvature in the function, and λ establishes a tradeoff between the two.
- Remarkably, it can be shown that (5.9) has an explicit, finite-dimensional, unique minimizer which is a natural cubic spline with knots at the unique values of the x_i: $f(x) = \sum_{j=1}^N N_j(x)\theta_j$
- $\text{RSS}(\theta,\lambda) = (\mathbf{y}-\mathbf{N}\theta)^T(\mathbf{y}-\mathbf{N}\theta) + \lambda\theta^T\mathbf{\Omega}_N\theta$, for $\{\mathbf{\Omega}_N\}_{jk} = \int N_j''(t)N_k''(t)dt$
- Solution $\hat{\theta} = \left(\mathbf{N}^T\mathbf{N} + \lambda\mathbf{\Omega}_N\right)^{-1}\mathbf{N}^T\mathbf{y}$
- Reinsch form - reparametrization, had $||y - H\theta||_2^2 + \lambda\theta^T\Omega\theta$. Now take $f = H\theta, \theta = H^{-1}f$ -> $||y-f||_2^2 + \frac{1}{K}\lambda f^T(H^{-1}\Omega H^{-1})f$. Then say $K = UDU^T$. If you know the eigens of K, you know it of $I + \lambda K$.
- S has two eigenvalues exactly equal to 1 - this creates a subspace. $S_\lambda v = v$. S_λ is smoothing some response as a function of X, so can think of v as a function of X. If v is linear in X, the smoother applies no penalty, can simply reproduce the line since it makes no error on the loss and the penalty is 0. The space of functions spanned by the constant and linear functions in X get the eigenvalues of 1, anything else gets penalized.
- For $S_\lambda v$, you wouldn't want the smoother to give you back something bigger than v - which is why we have largest

eigenvalues of 1. But also all are bigger than 0, not reversing the sign.
- Rows of the smoother matrix: fitted vector $\hat{f}_\lambda = S_\lambda y$, then each fitted vector is a sum of the y's across a row of S. We call this an equivalent kernel, interpreting the smoothing function as a weighted average. We see that it has local properties across x_i
- CV vs EPE on a simulated example - we see EPE is far below CV error but minimizer df is around the same place. CV error has more variance, but the minimum is still a good choice.
- LOOCV can be done efficiently with smoothing splines. $S_\lambda(i,i)$ - self influence. For small lambda, this fit becomes very local and close to 1 around for the locality around an x_i.
- Generalized additive model RSS, $\sum_{i=1}^n (y_i - \sum_{j=1}^p f_j(x_{ji}))^2 + \sum_{j=1}^p \lambda_j \int f_j''(t_j)^2 dt$. Can be fit using smoothing splines. Y is linear in functions wrapped around each X in an additive way.

Automatic Selection of Smoothing Spline

- Fixing degrees of freedom: $\mathrm{df}_\lambda = \mathrm{trace}(\mathbf{S}_\lambda)$ monotone for lambda, so can simply invert and specify a df.

Tensor Product Basis

- Fit model in higher dimensions, NSC say. Specify basis of function on each of the coordinates. Basis functions say for variable 1 and 2. Then for every basis function for 1 and every for 2, form a pairwise product
- Then have a basis across both spaces and can fit a linear model
- Easy to do in r - `lm(y ~ ns(x1, df =4) * ns(x2, df=4))`
- df = 4 x 4 = 16 vs linear model df = 1 + (4-1) + (4-1) = 7 (only want to count the intercept once here).

Kernels

- Polynomial ridge regression for example. X in p dimensions, h(X) basis expansion mapping to Rm, for m huge. Solution requres inverting an M x M matrix of basis. $H^T H$ is big and low rank (at most n). But we can rewrite the solution to be N x N - this is the heart of the kernel trick
- H is N x M matrix, N num observations. Each entry is the feature vector for the ith observation. So HH^T is all of the pairwise inner products for the rows in H (also called a Gram matrix).
- Our kernel function is a bivariate function that outputs a positive definite matrix $\mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$. K computes the inner products for you - K does this without needing to perform HH^T.
- All the computations just require K itself, don't actually use the basis functions, just know they exist
- These K's exist for a good number of basis functions. In the radial basis function, there in an infinite implicit basis, but the kernel does not need to consider these.
- Get a linear combination of the kernels in our model.
- There is a kernel for the smoothing spline, though it is only positive semi-definite (null space is the unpenalized dimensions)

Chapter 6: Kernels

Kernel Smoothing

- NN kernel - discontinuous since discrete. Wiggly, not very attractive, but easy to compute, O(n) for whole curve.
- General family of kernel functions that assign a weight between 0 and some number, indexed by target point x_0$ and another point for which we want to assign a weight.
- Bandwidth parameter h - depends on lambda and target point, D function assigns weight. h can be a constant that acts like a scale parameter, amounts to keeping the same width as we slide along function - metric kernel. This is a biased estimate, since if function is concave then average will be down, but for constant we get bias approximately the same everywhere. The variance can still vary since the number of points can change, cause us to average over different number of observations
- Can instead keep number of points we average over constant - constant variance but bias decreases for regions with a lot of data, increases in areas with less data.

- Weightings generally give weight 0 outside of range.

Local Regression

- At the boundary, kernel will reach more to the right (or left), but in a concave function say, the weighted average will be biased upwards, since more points to the up and right than nearer to the boundary.
- Instead of fitting a constant, let's fit a linear regression to the data with our kernel weights. Then the fitted value at x_0 is the value of the smoother.
- W a diagonal weight matrix, get a closed form minimization criterion. We get an equivalent kernel, one for assigning the observation weights and one that performs the weighted least squares. If there are more points on the left than the right, the equivalent kernel can compensate by giving the right values more weight, even on the interior for additional bias correction.
- Since the linear fit has bias in curved regions, could go further and do a local quadratic. Has less effect of "trimming down the hills, filling in the valleys." Of course the returned bias is traded for variance.
- We see local linear and local constant have similar variances on the interior. Local quadratic is somewhat higher on the interior and much higher on the boundaries.
- Generalization to two dimensions is basically automatic, while for smoothing splines this is much more complicated. Basic idea for local regression can simply be extended to incorporate more data directions. If you go to 3+ dimensions may get nearest neighbor type dimensionality problems.
- Conditional plots - for higher dimensional data, can condition on other predictors and let one vary within each window, conditioning on different values for the variables.

Varying Coefficient Models

- Coefficients conditioned on a varying factor in the model. Can condition on multiple factors like sex and depth in the aorta model.
- Then let the diameter of the aorta vary against the age, for different conditioned values of sex and depth
- How are they fit? Local regression gives us a way $y \sim B_0(S, D) + B_1(S, D) \times age$. For a given depth for males, could get your weighting kernel and fit a linear regression with weighting for $d_0, d_1, \ldots$.
- Any time you have a model and want it to change smoothly with one or more variables, we can use this mechanism - define a weighting function in terms of the that variable, then refit the overall model using these local observation weights. As long as the optimization can take observation weights, we can make any model local.

Kernel Density Classification

- Old fashioned method today
- Posterior given two densities. Density ratio not guaranteed to be smooth, get weird results when data is not distributed evenly across range.
- Can hide structure in class densities

Gaussian Mixture Models and EM Algorithm

- GMMs useful for modeling non-standard densities, eg multi-modal. Useful for generation but also for estimation of odd densities
- Form of non-parametric density estimation: $f(x) \sim \sum_{j=1}^{k} \pi_j \phi(x, \mu_j, \Sigma_j)$ - for ϕ Gaussian density. With a few Gaussians, we can approximate a weird density, but need a method for fitting GMM to data
- K-means and vector quantization - image in pixels and we want to compress it. Break up image into non-overlapping blocks of pixels -> each block is a vector of pixel values. Run KMC on the vectors, K is often large in this application. All the vectors then approximated by their centroids in the encoder. The codebook has the centroids and cluster assignments, which is a small dataset. Then a decoder reconstructs the image, looking up its cluster centroid values.
    - In a lossy way, we find clusters with some error, maybe some are pure white but many colors might be approximated. Lossless - we keep going until all clusters are pure, centroids and data points exactly match in vector values.

- Mixture Models - can be seen as a soft version of KMC.
    - Soft assignment of responsibilities - have two centers with a Gaussian density over each. Have a point and we want to assign it to a class. If we knew the Gaussians, could assign based on the posterior. A soft assignment wouldn't assign it just to one - would assign it with a probability responsibilty to each class given the strength of the density. K-means is the limit as the bandwidth of the Gaussian shrinks to 0.
- EM Algorithm
    - Like KMC; make soft assignments, give points weights for a cluster. Go to each cluster and compute a weighted mean across points, using their weight contribution to the cluster.
    - There is always some missing information - supposing we had it the whole problem would be easy. Some unobserved latent variables, naming them, we can write a joint density with observed values. Averaging over the data you have, you want to get the original mixture density you started with.
    - We don't know the values, so we work iteratively, substituting the expected value of the parameter given the parameters we know so far and the observed data.
    - If we have observations missing at random in a dataset. We could just use the mean of the observed values for each variable, though not perfect - if we now estimate the covariance, it will be biased, variances will be too small.
    - For $x_i \sim N_p(\mu, \Sigma),\ i = i, \dots, n$, if I knew μ, Σ, I could come up with a more educated guess for the missing values given the others. I would know the conditional distribution of the missing data given the rest. For $x = [x_{(1)}\ x_{(2)}],\ (\mu_1, \mu_2), (\Sigma_{11}, \Sigma_{12}, \Sigma_{21}, \Sigma_{22})$, the distribution $x_{(2)}|x_{(1)} \sim N(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12})$ - can think of this as the outcome of a linear regression. If there is correlation in these variables, then this distribution informs us about the missing values.
    - Good case for EM - pretend we know the missing data, and run EM. If O_i is the indexed set of the observed values, then $X_{i,O_i} \sim N_{O_i}(\mu_{O_i}, \Sigma_{O_i,O_i})$ - distribution of the observed values of the ith observation. Get a bunch of different sized normal densities for the observed log likelihood - quite a mess. By making up values we get a full loglikelihood

Chapter 9: GAMs

- Intercept and a general function for each of the predictors
- A linear model is an additive model trivially.
- Semi-parametric model - more interested in the parametric variables with their betas, but there are other factors we want to account for, the wrapped term
- Then can look at plots for each of the linear functions separately, some will be in single variables, some will be packed together interaction terms wrapped in a function
- There is a price to additivity, we ignore some interaction, can get different levels for a given pair of values. But the reduced flexibility of the additive model may reduce overfitting to the dataset.
- Can do the same for logistic regression - any linear model can be replaced with a GAM

Fitting

- Similar to coordinate descent, but in block fashion: referred to as backfitting or block coordinate descent
- Suppose we knew values of $f_2 \dots f_p$ and we just needed to fit f_1. Then we could subtract off those values from y and be left with $f_1 + \epsilon$. Then we get $f_1(x_1)$ as a smooth residual $S_1(y - f_2 \dots . f_p)$.
- Then the smoothers are our fitting funciton - smoothing splines, lowess, etc.
- We could start off with all functions = 0, then f_1 residual is just a smooth value of y.
- Then fitting the next function, we take f_1 value fitted, subtract from y, and fit f_2. Pass through one full time to get estimates for each function. Then cycle back to f_1 and subtract off the estimated values of the other functions, repeat, repeat until convergence.
- With two predictors, we alternate between them, each updated based on the last fit for the other predictor.
- Fitting on logistic regression - outer layer of iteration corresponding to the Newton algorithm. We previously used IRLS, here we use a weighted additive model. Newton-Raphson algorithm for a penalized log-likelihood problem

Interpretation

- Fitting a logistic additive model to spam, get the linear effects through the z-scores, non-linear effects through non-linear p values (test of non-linearity of $\hat{f}$)

Chapter 12: SVMs

- Note that we know code responses in two classes as ± 1, since this determine the size of the separating hyperplane
- Building out from the maximum margin classifier; necessarily some points from each class exactly on the margin.
- Then can formulate the problem as $\max\ C$ st $y_i(x_i^T\beta + \beta_0) \geq C$. Now since y is in -1 or +1, it is a signed distance for each class from the margin.
- Reformulate into the min $||\beta||$ st $y_i(x_i^T\beta + \beta_0) \geq 1$, since we are also working around the norm vector to the hyperplane. Easy transform since $C = \frac{1}{||\beta||}$
- Maximum margin works well when p > n since then the data is always linearly separable.
- Once you know the support points, you have everything you need to define the separating line. Then the solution coefficient $\hat{\beta}$ is defined just in terms of the support set S, the set of support points.
- Soft margin - need to allow some errors with non-seperable data, and can prevent overfitting generally. Give yourself a budget for how much overlap you will allow, the max the margin subject to being within that budget.
- The larger B, the more of the data that determines the orientation of the line - can then see B as a regularization parameter. Decreasing variance since relying on more of the data
- Full SVMs: Generally, the higher the dimension, the more points sit on the margin. Using the radial basis expansion, get very good performance
- Notice since the observations are only referenced through their inner products, can replace with a kernel / gram matrix
- Here the budget B determines the wiggliness of the solution in the kernel - if B is large it is less wiggly, small more wiggly
- Just like lasso / ridge, we have a tuning parameter and we can get the svmpath cheaply, using svmpath in R. Reduces budget parameter down to 0, refitting in a piecewise fashion. At the end left with the optimal separating hyperplane. If you write the solution in the right way, you can show that adjusting the B will change the fit in a piecewise linear fashion. When a new point crosses the margin, it crosses from a support point to no longer being a support point (positive alpha to 0 alpha). This then changes the linear path.
- SVM via loss / penalty: $\min_{\beta_0,\beta} \sum_{i=1}^{N} \left[1 - y_i f\left(x_i\right)\right]_+ + \lambda\|\beta\|^2$ - we see $\left[1 - y_i f\left(x_i\right)\right]_+$ is 1 minus the margin. Get charged a linear loss for values less than 1, then add a ridge type penalty. The hinge loss criterion is equivalent to the SVM.
    - We can compare to the binomial deviance loss - like a smooth version of the hinge loss.
    - If the data is separable, in the limit we get the same orientation of the margin - logistic regression and svm converge as lambda goes to zero.
- If the data is separable and you want a separating hyperplane, definitely want to use SVM to get a more stable solution. If not separable, often the logistic regression is a better solution and easier to tune.
- Logistic regression also has an edge in multiclass, has a natural extension. But SVM has more hacky ways of achieving multiclass, though Kernel Logistic Regression is more computationally heavy.