- Chapter 2 - Statistical Learning
  - Parametric Methods
  - Non-Parametric Methods
  - Assessing Model Accuracy
  - Classification
    - KNN
- Chapter 3 - Linear Regression
  - Qualitative Predictors
  - Variable Interaction
  - Common Problems
  - K-nearest neighbors regression (KNN regression)
  - Class Notes
- Chapter 4 - Classification
  - Logistic Regression
  - Linear Discriminant Analysis
  - Quadratic Discriminant Analysis
  - Evaluating Classification Methods
  - Classification Examples
- Chapter 5 - Resampling Methods
  - Cross Validation
    - Validation Set Approach
    - Leave One Out (LOOCV)
    - k-Fold CV
    - Choosing Optimal Model
    - One SE Rule
    - Wrong Way to do CV - ESL
  - Bootstrap
- Chapter 6 - Model Selection
  - Best Subset Selection
  - Stepwise Selection
  - Optimal Model Criteria
  - Shrinkage
    - Ridge Regression
    - Lasso
    - Comparing Ridge and Lasso
  - Dimensionality Reduction
    - Principal Components Regression
    - Partial Least Squares
  - High Dimensional Regression
- Chapter 7 - Non Linear Methods
  - Step Functions
  - Basis Functions
  - Piecewise Polynomials
  - Smoothing Splines
  - Local Regression
  - GAMs
  - Text Mining
- Chapter 8 - Tree-based Methods

        Regression Trees
        Classification Trees
        Evaluation of Trees
        Bagging
        Random Forests
        Boosting
    Chapter 9 - Support Vector Machines
        Maximal Marginal Classifier
        Support Vector Classifiers
        Support Vector Machines
        Support Vector Machines with > 2 Classes
        Relationship to Logistic Regression
        Generalizing Kernels
    Chapter 10 - Unsupervised Learning
        Principal Components Analysis (PCA)
        K-means Clustering
        Hierarchical Clustering
    ESL Chapter 14 - Non-Linear Dimensionality Reduction
        Kernel Principal Components
        Locally Linear Embeddings (LLE)
        Multidimensional Scaling
        Isomap
    ESL - Missing Data
        Methods for Adjustments
    Learning from Relational Data

Chapter 2 - Statistical Learning

- $Y = f(X) + \epsilon$ for a fixed but unknown function f of $X_1, \ldots, X_p$ and ϵ which is a random error term, mean 0 and independent of X.
- Prediction - we predict Y with $\hat{Y} = \hat{f}(X)$. Here f is sort of a black box, we care about the output but not necessarily what makes up the estimate of the function. The accuracy of the prediction depends on reducible and irreducible error - we can improve our modeling of f and this part of the error will decline, but the ϵ ensures we will not create a perfectly accurate predictor. Irreducible error comes from unmeasured variables, unmeasurable variation. Restrictive models may be better here too due to overfitting.
- Inference - looking to understand the relationship between X and Y, now f cannot be a black box but is the tool to decipher this relationship. Simple, parametric, and linear models are all more easily understood than their counterparts. Restrictive models are more interpretable and are often used for inference.
- Supervised - for each observation of the predictor measurements, there is an associated response measurement y_i. Fit a model that makes sense of this relationship
- Unsupervised - for every observation we observe a vector of measurements but no response measures.
- Regression problems have a quantitative response, while problems with qualitative response are classification problems. Note that some methods like logistic regression fall somewhere in the middle.

Parametric Methods

- First make an assumption about the shape of f, eg. f is linear.
- After selection a model, fit / train the model to the training data.
- Reduces the problem of estimating f down to estimating a smaller number of parameters. Tends to not truly match the form of f, but depends on the flexibility of the model. Fundamental limit to the quality of model fit.

Non-Parametric Methods

- Makes no assumptions about the form of f, simply fits to get close to the data points given some constraints. Can fit a wider range of f forms, but do not reduce the problem of estimating f to a few parameters and therefore need a large number of data points.
- Keep improving with more data

Assessing Model Accuracy

- $MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}\left(x_i\right)\right)^2$ The MSE will be small if the predicted responses are very close to the true responses, and will be large if for some of the observations, the predicted and true responses differ substantially.
- While training MSE can decline for changing model parameters, we are most interested in test MSE, ie. the average square distance between the test response measurements and the model estimates for the test inputs.
- As the flexibility of the statistical learning method increases, we observe a monotone decrease in the training MSE and a U-shape in the test MSE. When a given method yields a small training MSE but a large test MSE, we are said to be overfitting the data.
- We almost always expect the training MSE to be smaller than the test MSE because most statistical learning methods either directly or indirectly seek to minimize the training MSE.
- The expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental and positive quantities: the variance of $\hat{f}\left(x_0\right)$, the squared bias of $\hat{f}\left(x_0\right)$ and the variance of the error terms ϵ : $E\left(y_0 - \hat{f}\left(x_0\right)\right)^2 = \operatorname{Var}\left(\hat{f}\left(x_0\right)\right) + \left[\operatorname{Bias}\left(\hat{f}\left(x_0\right)\right)\right]^2 + \operatorname{Var}(\epsilon)$. This is minimized when MSE = $Var(\epsilon)$
- Variance $\operatorname{Var}\left(\hat{f}\left(x_0\right)\right) = E(\hat{f}(x_0) - E(\hat{f}(x_0)))^2$ refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set.
- Bias $\left[\operatorname{Bias}\left(\hat{f}\left(x_0\right)\right)\right]^2 = E(E[\hat{f}(x_0)] - f(x_0))^2$ refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. As we use more flexible methods, the variance will increase and the bias will decrease. Bias is the deviation of the average prediction from the truth

Classification

- Training error rate: $\frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right)$. Similar definition for test error rate.
- Bayes classifier: the test error rate is minimized, on average, by a very simple classifier that assigns each observation to the most likely class, given its predictor values. In two class setting, predicts class one if $\Pr(Y = 1|X = x_0) > 0.5$.
- The Bayes classifier produces the lowest possible test error rate, called the Bayes error rate. It is analogous to the irreducible error, since this is the lower bound to any classification method. In true modeling, we do not know these conditional probabilities, so Bayes is simply a theoretical gold standard.

KNN

- Given a positive integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0, represented by N_0. It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:
  $\Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I\left(y_i = j\right)$
- As K grows, the method becomes less flexible and produces a decision boundary that is close to linear. With K = 1, the KNN training error rate is 0, but the test error rate may be quite high.

Chapter 3 - Linear Regression

- Residual: $e_i = y_i - \hat{y}_i$. RSS = $e_1^2 + e_2^2 + \ldots + e_n^2$. Least squares chooses coefficients to minimize the RSS. Note that $MSE = \frac{1}{n} RSS$
- $\hat{\beta}_1 = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)}{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$
  $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$
- $\operatorname{SE}\left(\hat{\beta}_1\right)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$ where $\sigma^2 = Var(\epsilon)$. Notice in the formula that SE(B1) is smaller when the x_i are more spread out; intuitively we have more leverage to estimate a slope when this is the case.
- In general, σ^2 is not known, but can be estimated from the data. The estimate of σ is known as the residual standard error, and is given by the formula $RSE = \sqrt{RSS/(n-p-1)}$. Roughly speaking, it is the average amount that the response will deviate from the true regression line - it is a measure of the lack of fit of the model. We end up using $\hat{SE}$ since we do not know the true σ. For 95% CI's, we have $\hat{\beta}_1 \pm 2 \cdot \operatorname{SE}\left(\hat{\beta}_1\right)$
- t-stat - coefficient estimate over SE. If large, then the effect of the coefficient is large enough to overcome the uncertainty of its true value. Test statistic: $t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)} \sim t_{n-2}$. $H_0 : \beta_1 = 0$
- Rejecting H_0 does not mean the true relationship is linear. Not rejecting the null does not mean there is no relationship between x and y.
- $R^2 = \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} = corr(y, \hat{y})$ - measures the proportion of variability in Y that can be explained using X. A good value is highly dependent on the application. In simple regression, it is the same as the correlation between X and Y. Always increases with more predictors, but RSE can rise if the decrease in RSS is small relative to the increase in p.
- In the simple regression case, the slope term represents the average effect of a $1,000 increase in newspaper advertising, ignoring other predictors such as TV and radio. In contrast, in the multiple regression setting, the coefficient for newspaper represents the average effect of increasing newspaper spending by $1,000 while holding TV and radio fixed
- Variable selection: Forward starts with no predictors and at each step add predictor the variable that results in the lowest RSS. Backwards starts with all variables and removes the variable with the largest p-value. Mixed - start with no predictors and add using a forward rule, but if a p-value for existing predictor rises above a threshold, it is removed.

Qualitative Predictors

- If it only has 2 possible values, then we can use a dummy indicator variable that takes on 1 or 0. If we coded a multi level indicator as (0, 1, 2), then Y would go up by beta from level 0 to 1 and another beta for 1 to 2, but this makes no sense for qualitative variables. The dummy coefficient is the relative effect for that category on y compared to the chosen baseline category.

- If we pick 1 and -1 a simple linear regression, B_0 is the average value and I(1) indicates amount above the mean and I(-1) indicates amount below the mean for our subgroups
- For more than 2 levels for a variable, we simply create multiple true/false dummy variables. Always have one fewer dummy than the number of levels, since one can be the baselines when all dummies are false. Then we use the F test to determine whether the total of our dummies has significance beyond individual p-values

Variable Interaction

- Can think of synergy in a relationship as akin to the interaction between variables. Since standard regression looks at the effect of one variable while holding others constant, we do not include interaction effects automatically. The *additive assumption* is the feature of linear regression that assumes you can treat the variables as having no interaction - a single variable has a fixed effect on the response variable, regardless of the other variables.
- Adding an interaction term $\beta_3 X_1 X_2$ captures variable interaction. Good example of workers and production lines in a factory - to some extent both need to be expanded to increase capacity. We can interpret β_3 as the increase in the effectiveness of TV advertising for a one unit increase in radio advertising (or vice-versa).
- The **hierarchical principle** states that if we include an interaction in a model, we should also include the main effects, even if the p-values associated with their coefficients are not significant. This is akin to normalizing the variables being combined in the interaction term, since subtracting off the mean is the same as including a separate term for it.

Non-Linear Relationships - Polynomial Regression: Can add a higher order polynomial term by taking a variable to a power. Still a linear model of terms, but capturing a non-linear relationship.

Common Problems

- Non-linearity - useful to look at residual plots $e_i = y_i - \hat{y_i}$ vs x_i. Hope for the residual plot to be white noise, but the presence of a pattern indicates a potential problem with the model. If residuals show non-linearity, could try a transformation of the predictors like log, sqrt, polynomial. Note for higher dimensions, we plot the model residuals against the predicted response variable.
- Correlated error terms will lead to underestimating the true SEs. Frequent problem with time series or spatial data. We can see in plots of the residuals that they exhibit tracking, adjacent residuals may have similar values.
- Heteroscedasticity - non-constant variance of the error terms, ie. correlation between x and ϵ. Often transform Y to $log(Y)$ or $\sqrt{Y}$. Typical to see in growth or compounding
- Outliers - have an unusual value of y_i given x_i. Can use studentized residuals, computed by dividing each residual ϵ_i by its estimated standard studentized error. Observations whose studentized residuals are greater than 3 in absolute value are possible outliers. Little impact on model parameters but could affect measures of model goodness of fit.
    - If we think the outlier is an error, then we can remove it. But it could indicate misspecification of the model

- High leverage points - have unusual value for x_i and can have a sizable impact on the regression line. This is hard to identify in multiple regression settings just by eyeballing. We can calculate a leverage statistic: $h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^{n}(x_i' - \bar{x})^2}$. Always between 1/n and 1 with average leverage over all points (p+1)/n. A high leverage point will have a high leverage stat.
    - If we have high leverage points for multiple predictors -> can make pairwise comparisons to see high leverage points between predictors. The leverage stat formalizes this idea for any dimension
- Collinearity - two or more predictors are closely related to one another. Reduces the accuracy of the estimates of the coefficients causing SE for coefficients to grow and the power of the hypothesis test is reduced. Useful to look at a correlation matrix of the variables, though if multicollinearity exists in the interaction of multiple variables this will not be caught. Compute the Variance Inflation Factor (VIF), ratio of variance of a beta when fitting the full model over the variance of beta if fit on its own. $VIF(\beta_j) = \frac{1}{1 - R^2_{X_J|X_{-j}}}$ where $R^2_{X_J|X_{-j}}$ is the R^2 from a regression of X_j onto all of the other predictors (Note X_{-j} indicates all x excluding the jth predictor). If this term is close to 1, then you have collinearity and VIF blow up to be large.
    - Collinearity causes coefficient estimates to become less certain. End up with exactly the same fit for multiple betas - now difficult to optimize the model for each beta. Elongates contour lines of fit, meaning betas of highly varying degree are producing the same goodness of fit / CI for estimate.

K-nearest neighbors regression (KNN regression)

- Given value K for prediction point x_0, identifies K training observations closest to x_o, represented by N_0. It then estimates $f(x_0)$ using the average of all the training responses in N_0, ie. $\hat{f}(x_0) = \frac{1}{K}\sum_{x_i \in N_0} y_i$.
- Value of K boils down to bias-variance tradeoff - small values of K more flexible with low bias and high variance. The parametric approach will outperform the nonparametric approach if the parametric form that has been selected is close to the true form of f.
- We would not use KNN regression with a linear relationship; no matter how you choose K, you will always have higher test MSE than linear regression, since the linear model has almost no bias and also resists variance better. Linear regression can still outperform for non-linear relationships; may simply depend on the choice of K. For extreme non-linearities, then linear regression tends to underperform for wide range of K.
- With small number of predictors, MSE for KNN remains somewhat constant over different f linearities while linear regression has much higher MSE for non-linear relationships. KNN can still underperform significantly with more predictors, however. Spreading 100 observations over p = 20 dimensions results in a phenomenon in which a given observation has no nearby neighbors—this is the so-called curse of dimensionality.

Class Notes

- Matrix notation for a regression -> $y = X\beta + \epsilon$ where $y = (y_1 \dots y_n)^T$ and $\beta = (\beta_1 \dots \beta_n)^T$ and X is our data matrix with an extra column of 1 for the intercept. X stores each variable in a column, so X is dimension n x (p+1) and B is dimension (p+1) vector. betas are always linear but the predictors X do not have to be linear.
- F-test: $F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n-p-1)}$ for omitted variables q. Intuition behind F-test - RSS_0 is the RSS for the model under the null hypothesis where we set all of the betas to zero, just fitting the intercept, ie. the average of y. RSS is just the model RSS. RSS_0 is always bigger since predictors improve the fit of the regression.

Comparing the two RSS stats, if the additional betas don't help much, then the RSS should be roughly the same. Otherwise we would expect a big improvement in the RSS over the replacement model with fewer betas. F-stat is the scaled ratio of the difference in the RSS over the RSS of the full model. In the specific case where we omit a single beta, then this is equivalent to a t-test for that beta. Note that since F depends on n, when n is large, an F-statistic that is just a little larger than 1 might still provide evidence against H0. In contrast, a larger F-statistic is needed to reject H0 if n is small.

- Multiple testing issue - Given 10 betas, could do 10 different t-tests, 1 for each beta. Over a large number of betas and at significance level of 5%, bound to find significance as a false positive. Over 10 tests, P(at least one is positive) = 1 - P(no false positives) = 1 - P(1st test accepts and 2nd...) = 1- P(1st test accepts)P(2nd test accepts)... = $1-(0.95)^{10}=0.40$. The F-statistic does not suffer from this problem because it adjusts for the number of predictors.
- p-value - if the null hypothesis is true, there is a p% chance of making a false positive, ie. rejecting H_0 in error.
- Stepwise approach - construct a sequence of p models and select the best among them. Performed through forward, backward, or mixed selection. Note for forward selection we are looking at the individual linear models to determine which betas to add at each step. If p > n, we must use one of these methods since cannot fit a full model using OLS in this case.
- Confidence interval vs prediction interval: they predict the same thing but CI is the output of $f(x_o)$ but the PI takes the full model $y=f(x_0)+\epsilon$, so it accounts for the additional error. The CI is the uncertainty around the estimate of a coefficient β, the PI is the uncertainty of the full model including the irreducible error.
- Residual standard error: $RSE=\sqrt{\frac{1}{n-p-1}RSS}$ compensation for the increase in P increasing the the R^2
- R note: `lm(y ~ .)` regresses the y against all x terms in the dataset.
- For higher dimensional residual analysis - plot the residuals against the $\hat{y}$ instead of x, look for a pattern. Pattern suggests a leftover structure to the y values that remains to be explained.
- Studentized residuals - $\hat{\epsilon_i}=y_i-\hat{y_i}$ is an estimate of the true epsilon. It has SE $\sigma\sqrt{1-h_{ii}}$. Therefore we can divide the estimate by the SE. Should follow a t-distribution with n-p-2 DoF (Note OLS has residuals with mean 0, so we are only concerned with the estimate SE).

Chapter 4 - Classification

- Bayes Classifier - P(Y|X) is known, then given input x_0 we predict the response $\hat{y_0}=argmax_yP(Y=y|X=x)$. Minimizes expected 0-1 loss $E\left[\frac{1}{m}\sum_1^m 1(\hat{y_i}\neq y_i)\right]$, ie a binary loss function of wrong or right. This is the best theoretical result.

Logistic Regression

- We can assign a probability threshold to a label - could be 0.5 or other depending on if the situation is asymmetric. Model is fit using MLE: we try to find $\hat{B}_0$ and $\hat{B}_1$ such that plugging these estimates into the model for $p(X)$ yields a number close to one for all individuals who defaulted, and a number close to zero for all individuals who did not. Uses likelihood function: $\ell\left(\beta_0,\beta_1\right)=\prod_{i:y_i=1}p\left(x_i\right)\prod_{i':y_{i'}=0}\left(1-p\left(x_{i'}\right)\right)$
- Mapping regression output to range of [0, 1]. To get a sigmoid from [0, 1], we use the **logistic function** $f(x)=\frac{e^x}{1+e^x}$. For small x, close to 0 and close to 1 as e^x grows. The odds is given by $\frac{p(X)}{1-p(X)}=e^{\beta_0+\beta_1X}$, ranging from low probability near 0 and high probability near ∞

- We can then generate log odds / logit: $log\left[\frac{P(Y=1|X)}{P(Y=0|X)}\right] = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$. We cannot use least squares because we do not know the conditional probability - LHS not observed.
    - B_1 does not correspond to the change in $p(X)$ associated with a one-unit increase in X - the amount that $p(X)$ changes not depends on the value of X.
    - The intercept is not typically of interest and is just an adjustment to the average of the fitted probabilities
    - Test statistic: z-stat $= \frac{\hat{\beta}_j}{SE(\hat{\beta}_j)}$
    - Predictions: $\hat{p}(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}} = \frac{e^{-10.6513+0.0055\times 1{,}000}}{1+e^{-10.6513+0.0055\times 1{,}000}} = 0.00576$
- Confounding - a variable that influences both the dependent and independent variables causing spurious association, eg. students are likely to have high card balances, people with high balances are more likely to default, given a high balance students are less likely to default. Running the regression with just student will give you a positive coefficient, and running with balance too will give a negative coefficient for student. In a simple logistic regression, the student is standing in for balance.
- Similar issues with collinearity - creates instability in estimating the coefficients and affects the convergence of the MLE fitting (as does trying to fit p > n).
- Note: training error rate can increase with more parameters since logit does not directly minimize the 0-1 error rate but uses MLE.
- Multinomial logistic regression - y is no longer binary. Use linear model for log odds against a base category.

Linear Discriminant Analysis

- Defns: Prior = π_k, Likelihood/density of X $f_k(x) = Pr(X = x|Y = k)$ - ie. how likely is it that an observation in the kth class has $X = x$. Given the response, what is the distribution of the inputs.
- Bayesian estimate: $\Pr(Y = k|X = x) = p_k(x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$, for posterior $p_k(x)$
- $P(Y = k) = \pi_k$ - how likely are each of the categories in the training data - this gives us a prior. It is the overall probability that a randomly chosen observation comes from the kth class. In general, estimating π_k is easy if we have a random sample of Ys from the population: we simply compute the fraction of the training observations that belong to the kth class.
- We are assume Gaussian for $p_k(x)$ - think of heights (X) for men and women (Y), then conditioned on gender we have two normal curves.
- LDA approximates the Bayes classifier by plugging in estimates for the prior and parameters. Mean is sample mean, variance is sample variance for each class k in the training data. However, for sample variance, we assume that there is a shared variance σ^2 across all K classes, ie. $\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i;y_i=k} (x_i - \hat{\mu}_k)^2$. Note we are taking the sum of squared deviations within each k but then sum and normalize them across the different groups.
- Assuming a normal likelihood, we get a log-likelihood function $\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$. We then need to plug in estimates for these parameters using the sample mean, shared sample variance $\hat{\sigma}^2$, and $\hat{\pi}_k = \frac{n_k}{n}$. Notice the discriminant functions $\hat{\delta}_k(x)$ are linear functions of x. For a single predictor we assign observation x to the class for which $\hat{\delta}_k(x)$ is largest.

- To reiterate, the LDA classifier results from assuming that the observations within each class come from a normal distribution with a class-specific mean vector and a common variance σ^2, and plugging estimates for these parameters into the Bayes classifier.
- Multivariate case: will assume that $X = (X_1, X_2, \ldots, X_p)$ is drawn from a multivariate Gaussian distribution, with a class-specific multivariate mean vector and a common covariance matrix.
    - To indicate that a p-dimensional random variable X has a multivariate Gaussian distribution, we write $X \sim N(\mu, \Sigma)$. Here $E(X) = \mu$ is the mean of X (a vector with p components), and $Cov(X) = \Sigma$ is the p × p covariance matrix of X. The multivariate density is then: $f(x) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\right)$ and the log-likelihood function is given by $\delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log \pi_k$
- Looking for x values where discriminant equations are equal, eg $\delta_k(x) = \delta_l(x)$ - this defines the boundary between groups. This is where classes k, l do equally well at maximizing δ. The quadratic term drops out in the equality since it is the same for both discriminants, then we are left with comparing the linear values of x. Therefore the boundaries will be lines.

Quadratic Discriminant Analysis

- Assuming a common covariance matrix is restrictive - ensures boundaries are lines, while groups may be separated by non-linear boundaries. Remember a more flexible method only reduces bias in practice when we are dealing with non-linear Bayes boundaries.
- We no longer drop the quadratic term when comparing discriminant formulae since the covariance matrices are not the same anymore.
- QDA better for higher n or if we have reason to believe common covariance across K classes is unrealistic.

Evaluating Classification Methods

- 0-1 Loss doesn't tell you about if you are making the wrong prediction for some classes more than others. Does not distinguish between false positives and negatives.
- Use a confusion matrix - model the truth as either + or -. Then determine if you have True Negative, False Negative, False Positive, True Positive. FP = Type I error. True Positive = 1 - Type II Error.
- Sensitivity - percentage of correctly identified true (true positive). Specificity - percentage of correctly identified false
- Thus, the Bayes classifier, and by extension LDA, uses a threshold of 50 % for the posterior probability of default in order to assign an observation to the default class. However, if we are concerned about incorrectly predicting the default status for individuals who default, then we can consider lowering this threshold. For instance, we might label any customer with a posterior probability of default above 20 % to the default class.
- We can have very different error rates for FP and FN - think of Bank trying to determine who will default, cares much more about the error rate in predicting for the defaulting group than the non-defaulting. Can change the threshold for which $p(default = yes|X)$ sorts the data, say changing from 50% down to 20%. In this case it will increase the error for FP and decrease FN. This is an example of tuning - changing certain parameters to improve our model after fitting.

- The 0-1 error rate puts a lot of weight on the FP and little of the FN - this is why the 0-1 loss is not necessarily the best measure when we care about FN more. Especially lopsided when the overall percentage of data in one category is quite small.
- ROC Curve - False positive rate vs True positive rate (1 - false negative rate). Then the ideal point is (0, 1) - 0 FP and 0 FN. As we increase the FP rate along the x-axis, the FN decreases and the true positive rate approaches 1 monotonically. The better the classifier, the more it will hug the y-axis and y=1. A poor classifier follows x=y, which would be a randomized classifier. Considers performance over all possible thresholds.
    - Quantifying the goodness of classifier - take integral of curve to get the area under curve (AUC), closer it is to 1 the better, closer to 0.5 worse

Classification Examples

- In terms of flexibility, have logistic regression, LDA, QDA, KNN-CV, KNN-1 from least flexible to most. Parametric to nonparametric as well.
- Examples run simulations of the data, and compare the 0-1 error rates across methods.
- Scenario 1 Bivariate Normal without correlation - the boundary is simple, so the restrictive methods outperform. The more flexible methods are overfitting and since there was little bias to begin with, do not give us much benefit.
- Scenario 2 Bivariate Normal with correlation - does not impact our results, LDA just assumes same covariance matrix but that includes situations with correlation
- Scenario 3 independent t-distribution - QDA suffers from the heavy tails of t-stat. Optimal separation is still a line, so LDA and Logistic still do well. The sensitivity of QDA shows that simple methods may be better in a large data real-life setting - relaxed assumptions of normality do not hurt their fitting as much.
- Scenario 4 Bivariate Normal with different correlations - LDA assumption is violated, and QDA is made specifically for this situation. QDA does the best here, but the flexible KNN methods still do poorly.
- Scenario 5 uncorrelated SN with quadratic true decision boundary - QDA does well and KNN also do well.
- Scenario 6 Response sampled from nonlinear functions - KNN-CV is the best. Must be reserved for a complicated situation and we need to tune the parameter well. Even then it isn't outperforming by too much.

Chapter 5 - Resampling Methods

Cross Validation

- With a single training set, need to choose a method or tune a parameter to achieve the best test error rate. Looking for a proxy for the test MSE. This is specifically for supervised learning, since we are relying on misclassifications as errors.
- Tuning parameters - could be the K in KNN, number of predictors used in a regression, the order of a polynomial in polynomial regression

Validation Set Approach

- Can create a validation set that we hold out from the training data
- Divide the data, train on one part, compute the error on the other.
- Can use the `sample()` in r to pick random indices to include in one group or the other

- But 1) end up with different results based on the sample chosen 2) overestimate test error rate for the model trained on entire data set since training on smaller data set will reduce performance

Leave One Out (LOOCV)

- Train the model on every point except i and compute the test error on the held out point - deterministic, eliminating the randomness of the training data chosen.
- The LOOCV estimate for the test MSE is the average of these n test error estimates: $\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{MSE}_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i^{-i})^2$. The (-i) superscript indicates everything but the the observation i.
- Unlike validation set, always yields the same results and does not train on a particular subset of the available data. We generally fit the model n times, so could be expensive to implement if we have a lot of data, but OLS provides shortcut if we implement that type of model: $\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1-h_i}\right)^2$ where $\hat{y}_i$ is the ith fitted value from the original least squares fit, and h_i is the leverage statistic.
- Keep in mind the square root law - the SE of the average $SE(\bar{X}_n) = \frac{SD(X)}{\sqrt{n}}$. So averaging over n samples, we decrease the variation in the SE.
- For classification, error rate is given by $\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{Err}_i$ for $\mathrm{Err}_i = I\left(y_i \neq \hat{y}_i\right)$

k-Fold CV

- This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size.
- Iterate through the K groups, holding one out as a validation set each time, with total test error equal to $\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i$. Thus LOOCV is a special case of k-Fold with k=n.
- As we increase k we decrease the bias and increase the variance of the CV error. Diminishing marginal returns to size of training set up to the irreducible error.

Choosing Optimal Model

- Since we are repeating LOOCV on almost identical training sets, they are highly correlated. While this gives us a lower bias in our estimate of MSE, we get a higher variance compared to k-Fold with k < n.
- Typically, it has been shown that k = 5 or k = 10 is a good spot within the bias-variance tradeoff. K-Fold has more bias in its estimates of the test error since we are fitting on less data.
- Flexibility vs MSE - Flexibility is essentially the tuning of parameters. The error from CV compared to the test error, the curves may be off but the parameters that provide the minimum for the CV MSE is often very close to the parameters needed to minimize test MSE.
- For small sample sizes, cross validation pays a higher price than larger samples.

One SE Rule

- Choose the simplest model whose CV error is no more than one SE above the model with the lowest CV error. The CV error should be within a single SE of the individual misclassification error rates for each of the k-folds.
- The idea: We may continue to get some additional decrease in CV error with more variables, but quite marginal and not significant. Want to reign in model complexity and flexibility by sticking to the simplest error within some standard. Be careful when performing on KNN, since increasing K reduces flexibility.

Wrong Way to do CV - ESL

- Proposed strategy: logistic regression against all parameters, choose 20 with highest z-tests, use 10-fold CV to validate this model.
- We have many more predictors than sample size, so just by chance there will be correlations. If all predictors are actually random, it will still occur, even though each predictor has a small chance of showing correlation, among a large number just by chance some predictors will be correlated with the response.
- CV here will show a low error rate. Using variable selection against all of the data, some predictors with correlation will turn up in every fold, so CV will conclude these are significant relationships.
- How to fix it: Start by dividing the data into 10 folds before picking any parameters. Select the 20 most significant predictors in each K, compute the error on the fold. Then average the error across the 10 folds taken. The whole analysis has to be fresh for each fold.

Bootstrap

- Rather than repeatedly obtaining independent data sets from the population, we instead obtain distinct data sets by repeatedly sampling observations from the original data set. The sampling is done with replacement, so the same observation can occur more than once in the bootstrap data set.
- Example: average height of all people in the US. To estimate the answer, sample 50 people at random, $X_1, \ldots, X_n$ for n=50, and our estimate is $\bar{X}_n = 71\,in$. How much are we off, we need a measure of uncertainty with SE for our estimator. Suppose I can sample repeatedly, say B=200 times: $X_1^{(1)}, \ldots, X_n^{(1)} \rightarrow \bar{X}_n^{(1)}$, $X_1^{(2)}, \ldots, X_n^{(2)} \rightarrow \bar{X}_n^{(2)}$, ..., $X_1^{(B)}, \ldots, X_n^{(B)} \rightarrow \bar{X}_n^{(B)}$. Take the 200 estimates of $\bar{X}$ and take the SD, and this is the desired approximation of the SE. Now replace the population with the sample - the histogram of our sample of 50 resembles that of the underlying population.
- X_1^* (* indicates bootstrap) is drawn at random from $X_1, \ldots, X_n$, etc. You may see some observations multiple times while others do not show up at all. Main result: the SD of the bootstrap means is a good estimate of the SE
- Each resampled dataset Z^{*b} is called a bootstrap replicate. The bootstrap substitutes computational effort for theoretical calculation.

Chapter 6 - Model Selection

Best Subset Selection

- Fit OLS regression for each possible combination of the p predictors

1. Fit M_0 containing no predictors, simply uses the sample mean
2. For $k \in 1, \ldots, p$, fit all $\binom{p}{k}$ models that contain exactly k predictors. Pick the best among these (by RSS / R^2) and call it M_k.
3. Select a single best model among the M_k using some metric, AIC, BIC, CV, adjusting R^2

- Optimizing K metrics are standins for test error, but are much easier to compute than full CV
- When not talking about linear regression, we use deviance, which is $-2log(lik)$
- Suffers from computational limitations as possible models grow at 2^p. Also considering such a large search space make it more likely to find a model that fits the training data by chance without any predictive power over test data - extreme overfitting, similar to multiple testing problem, ie high variance.

Stepwise Selection

- Forward stepwise selection begins with a model containing no predictors, and then adds predictors to

the model, one-at-a-time, until all of the predictors are in the model. Consider p-k models that augment the predictors in M_k with one additional parameter, choose the one that decreases RSS the most. Though forward stepwise tends to do well in practice, it is not guaranteed to find the best possible model out of all 2p models containing subsets of the p predictors. Can be used when n < p since do not have to fit a model with all predictors.
- Backward stepwise selection begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time. Also not guaranteed to select the best model out of all possible models. Backward selection requires that the number of samples n is larger than the number of variables p so the full model can be fit.
- Both forward and backward are greedy algos and are unlikely to give you the same sequence - can use both to see where you end up.
- Mixed stepwise - take predictors out that become non-significant as we add more predictors. Can make it harder to remove than add to prevent loops of adding and removing

Optimal Model Criteria

- We can either estimate the test error by making adjustments to the training error to account for bias, or we can directly estimate the test error with CV.
- For a fitted least squares model containing d predictors, the C_p estimate of test MSE is computed using the equation $C_p = \frac{1}{n}\left(\mathrm{RSS} + 2d\hat{\sigma}^2\right)$. sigma hat is an estimate of the variance of the error epsilon. Essentially a penalty to the training RSS, and increasing function with the number of parameters used.
- $\mathrm{AIC} = \frac{1}{n\hat{\sigma}^2}\left(\mathrm{RSS} + 2d\hat{\sigma}^2\right)$ For OLS this criterion is proportional to Cp, differ only by a scaling factor
- $\mathrm{BIC} = \frac{1}{n\hat{\sigma}^2}\left(\mathrm{RSS} + \log(n)d\hat{\sigma}^2\right)$. BIC statistic generally places a heavier penalty on models with many variables, and hence results in the selection of smaller models than Cp.
- $\text{Adjusted } R^2 = 1 - \frac{\mathrm{RSS}/(n-d-1)}{\mathrm{TSS}/(n-1)}$ a large value of adjusted R2 indicates a model with a small test error. Adding additional noise variables only decreases RSS minimally and will be outweighed by d in the denominator.
- A way to do some validation that is much less expensive than CV. Rely on asymptotic arguments and model assumptions like normality of errors - a theoretical approach to limiting computation.

Shrinkage

- Keep all predictors but shrink them towards zero. Why is this better? Think of $Var(k\theta) = k^2 Var(\theta)$
- Introduces bias but may decrease the variance of the estimates - if the variance shrinkage is larger this decreases the test error. Imagine statistic T to estimate parameter μ, $T = \bar{X}_n$. You get a sampling distribution of that statistic T, say a bell curve histogram of T. Often T is restricted to unbiased estimators, but it was found that it might be advantageous to use a biased estimator with a much tighter sampling distribution - most of the time will be closer to true μ even though in expectation $T \neq \mu$. What we care about is the MSE, which is both bias and variance, so our biased T still might reduce the MSE.
- Bayesian motivations - the prior tends to shrink the parameters

Ridge Regression

- Ridge solves the optimization: $\min_\beta \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{i,j}\right)^2 + \lambda \sum_{j=1}^p \beta_j^2$. Note we could use constraint $\sum_{j=1}^p \beta_j^2 < s$ instead of a penalty.
- Start with least squares equation, minimizing the sum of squares but now add a multiple of the sum of

the beta-squares - the L2 norm of B, $||B||_2^2$. This makes the matrix invertible, but also has a statistical effect of shrinking the betas towards zero.
- As we increase lambda, the solution becomes smaller and smaller betas - lambda is a tuning parameter that can be chosen by CV. Lambda = 0 is just LS. However the betas do not become zero for $\lambda < \infty$
- Scale each variable st it has sample variance = 1 before running regression. Unlike normal regression, scaling matters since the beta terms are not dependent on X.
- The choice of λ is the trade-off between bias and variance - CV should tell us how to optimize this problem. CV tries a variety of lambdas and picks the one that minimizes the CV error. Lambda = 0 equivalent to $||\hat{B}_\lambda^R||_2 / ||\hat{B}^2||_2$ = 1

Lasso

- Lasso solves the optimization $\min_\beta \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{i,j}\right)^2 + \lambda \sum_{j=1}^p |\beta_j|$. Note we could use constraint $\sum_{j=1}^p |\beta_j| < s$ instead of a penalty.
- Here the penalty term is the L1 norm of beta - $||B||_1 = \sum_{j=1}^p |B_j|$
- Lasso shrinks some coefficients all the way to zero - alternative to best subset selection or stepwise. We can focus on a smaller subset of predictors and obtain a simpler model. Not only simple but performs well in predictions.
- As you increase lambda, more predictors drop out of the model entirely. For large enough lambda you can eliminate all of the predictors. No guarantee that the shrinkage is monotone as a function of lambda but most often is monotone.
- Can think of Lasso as a relaxation of best subset, since best subset is equivalent to an L0 norm penalty. We have seen that method can be quite computational taxing, so Lasso is next best solution that is actually feasible and gives good results.

Comparing Ridge and Lasso

- Lasso and Ridge are really constrained optimization using Lagrange multipliers - the penalty < constant s is the constraint for a fixed lambda.
- Best subset can be included here, minimization constrained by the L0 norm < s, ie $\sum_{j=1}^p 1(B_j \neq 0) < s$
- Visualizing lasso and ridge with 2 predictors
    - β contour lines, each level curve defines a level of equal RSS. The constraints Ridge: $\sum_{j=1}^p B_j^2 < s$, Lasso: $\sum_{j=1}^p |B_j| < s$ are expanding areas from the origin. The squared ridge constraint is a circle, and the point where it touches the level curves is the optimum. Since it is circular this is unlikely to occur at 0. The L1 ball is a diamond and the level curves are likely to intersect the diamond at 0. Think of in 100 dimensions, there are many vertices and likely to hit one when we intersect with the beta level curves. Note if the beta-hat were constrained in the constraint space, then the beta is optimal and we have least squares.
- Comparison in situations
    - R^2 of training data is used to plot both ridge and lasso together, since plotting against lambda is infeasible since the lambdas are actually different between the two models.
    - All coefficients are non-zero - then ridge is likely better, since the variance of the ridge is smaller, the bias is about the same, so the ridge MSE is smaller.
    - Only 2 of 45 coefficients are non-zero - the Lasso is better on all counts, generally lasso especially effective when most variables are not useful for prediction.

- Special case n = p, matrix X = I. Therefore y_j depends only on B_j and vice versa, Using Ridge, we get $\hat{B}_j^R = \frac{y_j}{1+\lambda}$. Taking the least squares fit and shrinking it by a positive number greater than 1. Using Lasso, either subtracting off a positive number for positive y or adding a small number to a negative y - soft thresholding.
- Bayesian interpretation
  - Gives us a distribution of a solution for betas, not a single point estimate.
  - Ridge: $\hat{B}^R$ is the posterior mean with a Normal prior. We take the distribution, take the mean, this is the Ridge beta estimate.
  - Lasso: $\hat{B}$ is the posterior mode with a Laplacian prior.

Dimensionality Reduction

- Added penalties are very common in statistics and optimization to create function that can be worked with.
- Increasing lambda increases bias, but it reduces variance
- Idea: define a small set of M predictors which summarize the information in all p predictors - we saw this with PCA, using a transformation to combine information from many predictors into a smaller set of new predictors
- Let $Z_1, Z_2, \ldots, Z_M$ represent $M < p$ linear combinations of our original reduction linear combination p predictors: $Z_m = \sum_{j=1}^{p} \phi_{jm} X_j$
- Fit the regression model $y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, \quad i = 1, \ldots, n$
- All dimension reduction methods work in two steps. First, the transformed predictors $Z_1, Z_2, \ldots, Z_M$ are obtained. Second, the model is fit using these M predictors. However, the choice of $Z_1, Z_2, \ldots, Z_M$ or equivalently, the selection of the ϕ_{jm}'s, can be achieved in different ways.

Principal Components Regression

- We assume that the directions in which $X_1, \ldots, X_p$ show the most variation are the directions that are associated with Y .
- In the USArrests data, we saw the loadings for the first PC were heavy for the various crimes, low for criminal pop. Therefore the score for the first PC represents the overall rate of crime in each state. Each M (state) gets a score, each predictor assigned loadings
- In the regression, we replace our p predictors with M score vectors. Obtain then M coefficients $\theta_0, \ldots, \theta_M$. We can rearrange terms to see that we are still doing a regression on the original space of x, but now the betas are restricted. The coefficients are interdependent, and we only have M θ terms across p X terms to vary. Think of constrained to working in a plane in three dimensional space. Coefficients are constrained by $\beta_j = \sum_{m=1}^{M} \theta_m \phi_{jm}$ where $M \leq p$. Introduces bias but can lower the variance of the model.
- Usefulness is dependent on the actual dataset. The Credit dataset has best error at 10 components instead of the full 11 - there is almost no dimensionality reduction here. If the best error is with all parameters, then just have least squares. So PCR should only improve on the least squares fit since OLS is always an option in selection number of components.
- In the case of 45 predictors with 42 significant, PCR performs worse than Ridge. In 45 predictors with 2 significant, PCR does worse than Lasso. PCR will tend to do well in cases when the first few principal components are sufficient to capture most of the variation in the predictors as well as the relationship with the response.
- Note this is not a feature selection method, since the principal components are linear combinations of

all p of the original features. Note also since PCR is unsupervised, there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.

Partial Least Squares

- PLS is a supervised alternative to PCR: PCR only uses information in the X predictors, but now we include information from the Y responses as well.
- Regress Y onto X_j, then our ϕ_{j1} is the coefficient from this regression. Define $Z_1 = \sum_{j=1}^{p} \phi_{j1} X_j$.
- Look at the direction that explains Y best, instead of the direction of greatest X variance. Gives more weight to the variables / residuals that are more correlated with the response.
- Take the residuals of the simple linear regression, and repeat. Run a regression of Y against the residuals. Continue to define $Z_1, Z_2, \ldots$ Then do a regression of Y onto the Z vectors.
- To form each component, PLS give more weight to the variables / residuals that are more correlated with the response. To form each residual, PLS removes the information explained by the previous components.
- Compared to PCR, PLS has less bias, more variance ie. tendency to overfit. The Y responses have noise, and now we increase the chance of fitting to that noise as well.

High Dimensional Regression

- Bag of words - count the words in data, turns it into quantitative measures. Number of predictors is as many as the words in the dictionary, so p >> n
- When p > n, formally we have no solution, but if we assume the effect is simple enough, the real p that drives the effect can be found, say through regularization / shrinkage. In actuality the regression line becomes too flexible, since we have so many predictors, we overfit to our exact data set of n. Measures of training error become bad proxies for test error.
- Plots of Lasso performance vs increasing predictors - adding many noise predictors hurts Lasso performance of the regression. Lasso looking at all possible options of selecting variables, but too much noise to find the signal when p >> n
- With p unknowns and n equations, there is always multicollinearity - some predictors will have to be defined in terms of others. There are then many ways to write out a good solution, we should not take the fitted model as the one true model given our data.

Chapter 7 - Non Linear Methods

- Wage vs Age - could fit a polynomial regression to show the concave down shape. But also have a classification problem, since the high earning group is very split off from the lower earning group. Could use logistic regression with higher order terms and get a flexible classification model, finding probability of being in higher age group given age. Standard error blows up on the upper end, possible due to the matrix becoming singular in that region
- Could instead use piecewise indicator functions, get averages over certain age groups.

Step Functions

- We break the range of X into bins, and fit a different constant in each bin. This amounts to converting a continuous variable into an ordered categorical variable.
- Create cutpoints then use indicator functions for points that lie within each range:
  $C_1(X) \quad = \quad I\left(c_1 \leq X < c_2\right)$ - ie. dummy variables

- $y_i = \beta_0 + \beta_1 C_1 (x_i) + \beta_2 C_2 (x_i) + \ldots + \beta_K C_K (x_i) + \epsilon_i$

Basis Functions

- The idea is to have at hand a family of functions or transformations that can be applied to a variable $X : b_1(X), b_2(X), \ldots, b_K(X)$.
- Fit the model $y_i = \beta_0 + \beta_1 b_1 (x_i) + \beta_2 b_2 (x_i) + \beta_3 b_3 (x_i) + \ldots + \beta_K b_K (x_i) + \epsilon_i$. Think of generalized version of polynomial or piecewise / indicator functions

Piecewise Polynomials

- Fit cubic polynomial up to age 50, then look at the data above age 50 and fit another polynomial to that data.
- Piecewise continuous - force the functions to meet at 50 to make function continuous
- A piecewise cubic polynomial with a single knot at a point c takes the form:
  $y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c \end{cases}$
- Cubic spline - fits polynomials on each data set, and has continuous point at 50 with 1st and 2nd derivatives continuous too.
  - define knots - ie. the break points, 1 - K
  - Fit a cubic polynomial $Y = f(x)$ between each pair of knots, s.t. 1st and 2nd derivatives exist
  - Can write f in terms of K+3 basis functions:
    $f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 h (X, \xi_1) + \cdots + \beta_{K+3} h (X, \xi_K)$ where
    $h(x, \xi) = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise} \end{cases}$. At each knot point can change the higher order terms while keeping the continuity conditions.
- Linear spline - simple linear fits continuous at the break point.
- Natural cubic splines - At the end points, use linear spline instead of cubic, use cubic on the rest. Helps control the SEs at the end points, since polynomials tend to become more erratic at their endpoints - Gibbs phenomenon. More stable for extreme values of X.
- Choosing knots - back to bias-variance tradeoff, chosen through CV. Locations are typically at quantiles of X. Graph with DF vs MSE, usually MSE drops dramatically with added DF and then flattens.
- Splines can fit complex functions without the weird behavior of a very high degree polynomial.

Smoothing Splines

- Writes an optimization problem as the addition of the sum of squares differences and a penalty: $\sum_i^n (y_i - f(x_i))^2 + \lambda \int f''(x)^2 dx$
- The term $\sum_i^n (y_i - f(x_i))^2$ is a loss function (RSS) that encourages g to fit the data well, and the term $\lambda \int f''(x)^2 dx$ is a penalty term similar to ridge regression
- The second derivative defines the curvature of the model - penalize flexibility through the curvature of the function. A large second derivative is a more flexible model. Since all linear functions of $f''(x) = 0$, they have no penalty. Integration over the whole line to capture the curvature over the whole domain.
- For large lambda, we force the second derivative to be small and we force the model towards least squares. Very small lambdas allow for very flexible models that could interpolate between every data point for an exact fit to the training data - overfitting.

- The minimizer $\hat{f}$ of this function is a natural cubic spline with knots at each sample point $x_1, \ldots, x_n$. Contrast that to piecewise knots where we had a fixed number of knots over the data set. Even though we have many knots, we are forcing them to be smooth so still has limited flexibility.
- Choosing the regularization parameter λ - chosen with CV. The tuning parameter λ controls the roughness of the smoothing spline, and hence the effective degrees of freedom. Although a smoothing spline has n parameters and hence n nominal degrees of freedom, these n parameters are heavily constrained or shrunk down. Shortcut `in choosing λ for LOOCV. Can often get a similar fit with smaller effective DF

Local Regression

- At each point, use regression function fit only to nearest neighbors of that point - this is a generalization of KNN regression. While KNN fit an average of y_i, we now fit a least squares line through those neighbors - could end up with pretty different y values depending on how the data is distributed around the point.
- Span - fraction of training samples used in each regression. Smaller span increases flexibility.
- Weighting function K - 0 outside of the nearest neighbors, and decreases away from our x_i within the collection of nearest neighbors - seems similar to kernel density functions. Leaves us with a weighted least squares problem: $\sum_{i=1}^{n} K_i(y_i - \beta_0 - \beta_1 x_i)^2$ - while we sum over all n, the weight is 0 outside of nearest neighbors, but this allows us to simply use the least squares optimization.

GAMs

- Extension of non-linear models to multiple predictors. Take functions of our predictors - don't use $f(x_1, x_2, x_3)$, instead use $f_1(x_1) + f_2(x_2) + f_3(x_3)$ hence the name additive.
- Still restrictive - no interaction since it is an additive model. If we wanted it to be more flexible, we would have to combine the variables into a single function - but this leads to the curse of dimensionality, overfitting becomes exponentially worse, etc. May want to start by including interaction terms to two variables to keep the dimensions constrained.
- Could use splines, polynomials, step functions, etc for the f applied, and then have a basis representation. Then we are back to least squares in fitting the model.
- If we use a function without a basis representation, can use backfitting.
    - Keep $f_2, \ldots f_p$ fixed and fit f_1 using the partial residuals $y_i - \beta_0 - f_2\left(x_{i2}\right) - \cdots - f_p\left(x_{ip}\right)$ as the response
    - Iterate for each f_i, fixing the other functions and fitting on partial residuals.
    - Could start out say with the identity function, fit it, then refine on the partial residuals. This works for smoothing splines and local regression.
- Somewhere between linear regression and a fully nonparametric method.
- In practice we can examine each predictor against its f output to see how the transformation performs.

Text Mining

- Given a large corpus of documents, how do we gain an unsupervised understanding of the content and relationships
- Latent variables - hidden variables, are there certain underlying topics that drive the content of certain documents
- Turn each document into a bag of words - count how often each word comes up in the document in a

long vector. Characterizes the article (perhaps almost uniquely).
- Create a matrix, row for each document, each column a word count in the article.
- Apply PCA to the matrix - reduction of dimensionality of the matrix.
- Latent Dirichlet Allocation - can model these documents by topics. Say we have K topics, and each topic is a distribution over words. Then each document is a distribution over topics.

Chapter 8 - Tree-based Methods

Regression Trees

- Find a partition of the space of predictors, predict a constant in each set of the partition. Defined by splitting the range of one predictor at a time - draw a single partition over one predictor then iterate, ie recursive splits.
- Could get splits that are concentrated in one section of the $X_1,\ X_2$ grid, where there is a lot of variability in the response variable in that area and relatively static response values elsewhere. Can be a simple method to work with high dimensions
- We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$
- For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j.
- To split, we take a top-down, greedy approach that is known as recursive binary splitting. We first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS. For any j and s, $R_1(j,s) = \{X|X_j < s\}$ and $R_2(j,s) = \{X|X_j \geq s\}$, we seek j and s that minimize $\sum_{i:x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i:x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2$. Repeat for new best predictor and new cutpoints. Terminate when there are 5 observations or fewer in each region (or some other stopping criterion) in a top down greedy approach.
- This doesn't encounter the same curse of dimensionality, since we consider one axis at a time - we aren't looking for near neighbors over many dimensions at once.
- This tends to overfit - so we grow a large tree and then prune it back. Cost complexity pruning: For each value of α there corresponds a subtree $T \subset T_0$ such that $\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2 + \alpha|T|$ is as small as possible. |T| is the number of terminal nodes / leaves. When $\alpha = \infty$ we select the null tree. When $\alpha = 0$, we select the full tree. α penalizes having many terminal nodes. Then can choose the optimal α by CV.
- Alternative pruning approach starts with the full tree T_0 and replaces a subtree with a leaf node. Minimize $\frac{RSS(T_1) - RSS(T_0)}{|T_0| - |T_1|}$. Turns out you get the same sequence of trees from this procedure as the other pruning procedure. While α moves continuously and this procedure is discrete, since the trees are discrete they produce the same sequence. These methods are very similar to Lasso.
    - Wrong way to do CV: construct trees for range of values on full dataset, split the training points into 10 folds, for each tree use every fold except kth to estimate the averages in each region and calculate the RSS of the kth fold. We cannot build the trees on the full dataset before splitting - splitting into folds always comes first.
    - Correct way: split into 10 folds, for k in 1 - 10, using every fold except the nth, construct a sequence of trees for range of alphas, calculate RSS on test set, then select the alpha that minimizes the

average test error across test folds.

- Other ideas don't work as well: CV across all trees still overfits due to too many possibilities. Also CV would select a tree that is best fit for the training data, but trees are not especially resilient to changing samples / data. Stopping the growth of the tree once we have diminishing returns to the decrease in RSS may prevent us from finding good cuts after bad ones.
- For the baseball data - our model fit predicts if year below 4.5, we make a single prediction. Above this number of years, we split into two regions based on number of hits. Like KNN, we use the average response as the prediction for a region.

Classification Trees

- We predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. We use the error function $E = 1 - \max_k (\hat{p}_{mk})$. Here $\hat{p}_{mk}$ represents the proportion (on 0 to 1) of training observations in the mth region that are from the kth class. Misclassification error $\sum_{m=1}^{|T|} \sum_{x_i \in R_m} \mathbf{1} \left(y_i \neq \hat{y}_{R_m} \right)$
- This is often not sensitive enough so can use the Gini index $G = \sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk} \left(1 - \hat{p}_{mk}\right)$, a measure of total variance across the K classes. q_m is the proportion of samples in R_m The Gini index is a measure of node purity - a small value indicates that a node contains predominantly observations from a single class. If instead of predicting the most likely class, we predict a random sample from the distribution, the Gini index is the expected misclassification rate.
- Entropy can also be used: $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$. Similar measure of node purity. Cross entropy: $-\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$
- Typically use Gini / Entropy to evaluate the quality of a split due to their node purity sensitivity, but classification error rate is best for pruning when prediction accuracy is the goal.
- Splits can yield the same predicted value, because it leads to increased node purity. This split does not reduce classification error but improves node purity - ie. we are more certain of our predicted value along one of those split paths.

Evaluation of Trees

- Tend to work better than linear regression when we have a highly non-linear, complex relationship (say a square boundary) between features the response. Also may be preferred simply for interpretability or visualization
- However tend to not have the prediction accuracy of other approaches and are not robust - small changes in the data can cause a large change in the estimated tree. This can be improved through bagging, random forests, and boosting.
- Missing data - can use 2nd or 3rd best response variable when we are missing the output for an observations. Can still propagate the observation down the tree using the 2nd best split - ie instead of deciding based on heart rate, then look at the second best split for a possible tree like resting beats per minute.

Bagging

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method
- Recall $Var(\bar{X}) = \frac{\sigma^2}{n}$ - averaging over observations reduces variance. Bagging simply takes samples from the training data set and trains our method on the bth bootstrapped training set, then averages across all predictions: $\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$. For classification, take the majority vote from the set

of predictions $\hat{y}_0^{(1)}, \ldots, \hat{y}_0^{(B)}$

- Bagging is not specific to trees, could be used for any learning method. Simply average the predictions of the model fit to many bootstrap samples. Popular to take the output of different algorithms and average their results - reduces the variance of predictions.
- To apply bagging to regression trees, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance.
- The number of trees B is not a critical parameter with bagging; using a very large value of B will not lead to overfitting. In practice we use a value of B sufficiently large that the error has settled down.
- On average, each bagged tree makes use of around two-thirds of observations - the remaining third are referred to as the out-of-bag (OOB) observations. This is simply due to the probability of not drawing an observation when sampling with replacement. We predict the response for the ith observation averaging (regression) / voting (classification) across tree in which it was an OOB observation. Can get the test error by looking at this measure across all n observations without resorting the CV: $\left(y_i - \hat{y}_i^{\text{obb}}\right)^2$
- Bagging reduces interpretability of the model, since it is no longer clear which variables are most important. Instead, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor. For classification, use Gini index instead of RSS.

Random Forests

- Random forests provide an improvement over bagged trees by way of a random small tweak that de-correlates the trees.
- Uses similar procedure as bagging, but when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ - that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors
- Bagged trees may be highly correlated if there is one big predictor, since almost every tree will use this as its top split. Averaging across correlated quantities does not reduce the variance as much. Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, on average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. If we choose m = p, then the random forest is exactly the same as bagging.
- Using a small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors.
- Gain more in the variance reduction than you lose in the bias introduced by doing splits over less significant predictors. This is a constant in ML - introducing some jitter to get a better result, even at the cost of potentially worse intermediate steps.

Boosting

- Boosting works similarly to bagging, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set. Like bagging, it can be applied to any learning method, not just trees.
- Given the current model, we fit a decision tree to the residuals from the model. By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.
- 3 tuning parameters:
    - B: # of trees. Can overfit if B is too large. B is selected through CV
    - λ: shrinkage parameter, a small positive number. Controls the rate of learning, the small lambda requires larger B
    - d: number of splits in each tree. Controls the complexity of the boosted ensemble. Often use d=1 and each tree is a single split, called a stump, which is just an additive model. More generally d is the interaction depth, controlling the interaction order of the overall model.
- Procedure
    - Set $\hat{f}(x) = 0 \text{ and } r_i = y_i$ for all i
    - Looping over 1 to B:
        - Fit a tree with d splits to the training residual data.
        - Update model $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$
        - Update the residuals $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$
    - Total model: $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$

Chapter 9 - Support Vector Machines

Maximal Marginal Classifier

- In a p-dimensional space, a hyperplane is a flat affine subspace of dimensions p − 1, defined as $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$. It divides a $p-$dimensional space into two halves.
- Normal vector $\beta = (\beta_1, \ldots, \beta_p)$ is a unit vector ($\sum_{j=1}^{p} \beta_j^2 = 1$) $\perp$ to hyperplane. If the hyperplane goes through the origin, the signed distance between a point and the hyperplane is the dot product. The sign of the dot product tells us which side of the hyperplane on which the point lies. Recall the dot product could be thought of as the projection of X onto the β vector.
- Suppose that we have a $n \times p$ data matrix X that consists of n training observations in $p-$dimensional space, and that these observations fall into two classes—that is, $y_1, \ldots, y_n \in \{-1, 1\}$ where −1 represents one class and 1 the other class. We also have a test observation, a p-vector of observed features
- A separating hyperplane has the property $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} > 0 \text{ if } y_i = 1$ and $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} < 0 \text{ if } y_i = -1$, ie. $y_i (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0$
- Then we can classify a test observation based on the sign of where it lies relative to the hyperplane. The magnitude of the distance from the hyperplane tells us something about the confidence of the prediction.
- If a separating hyperplane exists, there are infinitely many choices. A natural choice is the maximal margin hyperplane (also known as the optimal separating hyperplane), which is the separating hyperplane that is farthest from the training observations. The margin is the minimal distance from the observations to the hyperplane. In a sense, the maximal margin hyperplane represents the mid-line of

the widest “slab” that we can insert between the two classes.

- Support vectors are the observation(s) that have minimal distance to the margin hyperplane - they determine the plane used. Interestingly, the maximal margin hyperplane depends directly on the support vectors, but not on the other observations
- Construction of MMH: $\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\text{maximize}} M$ subject to $\sum_{j=1}^{p} \beta_j^2 = 1$ and $y_i \left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}\right) \geq M \, \forall i = 1, \ldots, n$. The constraints ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane. Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, \ldots, \beta_p$ to maximize M. Since $y_i = \pm 1$ we get a positive distance greater than M since the dot product of x and beta is also signed.
- This can be reformulated into a quadratic optimization problem that is simpler to solve. Could also use KKT multipliers similar to lagrange multipliers - can take the partial derivatives set to 0. The optimal normal vector is a linear combination of the training points on the margin.

Support Vector Classifiers

- We can extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called soft margin. The generalization of the maximal margin classifier to the non-separable case is known as the support vector classifier.
- A soft margin can provide greater robustness to individual observations and better classification of most of the training observations
- An observation can be not only on the wrong side of the margin, but also on the wrong side of the hyperplane. In fact, when there is no separating hyperplane, such a situation is inevitable.
- Construction: $\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} M$ subject to $\sum_{j=1}^{p} \beta_j^2 = 1$ and $y_i \left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}\right) \geq M \left(1 - \epsilon_i\right)$ and $\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$
- C is a nonnegative tuning parameter. M is the width of the margin; we seek to make this quantity as large as possible. $\epsilon_1, \ldots, \epsilon_n$ are slack variables that allow individual observations to be on slack the wrong side of the margin or the hyperplane.
- The slack variable tells us where the ith observation is located, relative to the hyperplane and relative to the margin. If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin, and we say that the ith observation has violated the margin. If $\epsilon_i > 1$ then it is on the wrong side of the hyperplane (since then $1 - \epsilon_i$ becomes negative). C bounds the sum of the i’s, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate. We can think of C as a budget for the amount that the margin can be violated by the n observations. In practice, C is treated as a tuning parameter that is generally chosen via cross-validation.
- When the tuning parameter C is large, then the margin is wide, many observations violate the margin, and so there are many support vectors that determine the hyperplane. This classifier has low variance but higher bias. The more points that fall into the margin, the more points that contribute to the budget and reduce the variance across samples. Perfect separation (C=0) has high variation, since the addition of a single data point could shift the boundary significantly.

Support Vector Machines

- Automatically converts a linear decision boundary into non-linear ones.
- We could address the problem of possibly non-linear boundaries between classes by enlarging the feature space using quadratic, cubic, and even higher-order polynomial functions of the predictors. In the enlarged feature space, the decision boundary that results is in fact linear. But in the original feature

space, the decision boundary is of the form $q(x) = 0$, where q is a quadratic polynomial, and its solutions are generally non-linear.

- The inner product of two r-vectors a and b is defined as $\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$. The inner product of two observations is $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$ and the linear support vector classifier can be represented as $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$.
- It turns out that α_i is nonzero only for the support vectors in the solution—that is, if a training observation is not a support vector, then its α_i equals zero. So we just need to calculate inner products for the support points
- Define K some kernel - a function that quantifies the similarity of two observations. The SVC uses kernel $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$, but we could instead choose a nonlinear kernel, like this polynomial kernel: $K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$. Another popular choice is a radial kernel $K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2\right)$ which takes a more circular shape, notice it is minimizing a euclidean distance.
- Then can take $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$ to $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + B_4 X_2^2 = 0$. This is a linear boundary in the augmented variable set (X_1, X_2, X_1^2, X_2^2) but a quadratic boundary in (X_1, X_2). Now are fitting a hyperplane in a shifted space. Basically fitting a more complex model in the same space is equivalent to keeping a linear boundary in a higher dimensional space.
- Kernels have a computational advantage, important since SVMs often work with very large enlarged feature spaces.
- The kernel matrix contains all of the inner products computed, always positive, semi-definite. Can make new kernels from sums and products of kernel matrices we know to be positive, sd. Not every similarity function is a valid kernel - must meet those matrix properties with their inner products.
- The bottom line of kernels - since our optimization comes down to inner products, we can use complicated kernels to make SVMs more useful. Inner products replaced with kernels now applied to many other learning methods like PCA. Take $\langle x_i, x_j \rangle \rightarrow k(x_i, x_j)$
- A kernel defines a similarity between the samples x_i and x_k. Define families of kernels which capture similarity between non-standard types of data - text, images, molecules, graphs, histograms. The expansion Φ can be infinite dimensional but the kernel is still computable.
- String dot kernel - count number of times word u occurs a string and kernel computed in $O(Lp)$ time
- Gap weight kernel - for each word u of length p we look for the subsequences that create the word in a string - count the number of gaps splitting the word. Computed in $O(L^2 p)$

Support Vector Machines with > 2 Classes

- One-vs-one or all pairs approach: construct $\binom{k}{2}$ SVMs, comparing all pairs of classes. Classify as test observation using each of the classifiers and tally the number of times that the test observation is assigned to each of K classes - take the winning vote.
- One-vs-all: fit K SVMs, comparing one of K classes to the remaining K-1 classes. Assign test observation for which linear combination of parameters is highest - hyperplane where we have most confidence of separation.

Relationship to Logistic Regression

- We can rewrite the criterion for fitting the SVC as $\underset{\beta_0, \beta_1, \ldots, \beta_p}{\text{minimize}} \left\{ \sum_{i=1}^{n} \max\left[0, 1 - y_i f\left(x_i\right)\right] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$
- In logistic regression we minimize the loss $min_{\beta_o, \beta} \sum_{i=1}^{n} log[1 + e^{y_i} f(x_i)]$. Recall we fit logistic regression using MLE, but this loss function is equivalent.

- Comparing the losses - SVM has a kink due to its maximization, but the shape is quite similar - declining loss in $y_i(\beta_0 + \ldots + \beta_p x_{ip})$
- When λ is large then $\beta_1, \ldots, \beta_p$ are small, more violations to the margin are tolerated, and a low-variance but high-bias classifier will result. When λ is small then few violations to the margin will occur; this amounts to a high-variance but low-bias classifier.
- Uses a similar penalty term as ridge regression - we have set up a similar loss minimizing procedure. Here we are using hinge loss: $L(\mathbf{X}, \mathbf{y}, \beta) = \sum_{i=1}^{n} \max\left[0, 1 - y_i\left(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}\right)\right]$
- When the classes are well separated, SVMs tend to behave better than logistic regression; in more overlapping regimes, logistic regression is often preferred. There is in fact an extension of the SVM for regression (i.e. for a quantitative rather than a qualitative response), called support vector regression.

Generalizing Kernels

- Perform PCA with an expanded set of predictors defined by the mapping Φ - replacing the data vectors with mapped data
- Normal PCA only accounts for linear variability. Kernel PCA allows for nonlinear curves that best explain data variability.
- Can expand kernels to many other learning methods - anytime a procedure depends on inner products it can be generalized with the kernel trick.

Chapter 10 - Unsupervised Learning

- PCA looks to find a low dimensional representation of the observations that explain a good fraction of the variance. Clustering looks to find homogeneous subgroups among the observations. Both seek to simplify the data via a small number of summaries
- Clustering involves making a lot decisions - dissimilarity measures, choice of K, scaling.

Principal Components Analysis (PCA)

- When faced with a large set of correlated variables, principal components allow us to summarize this set with a smaller number of representative variables that collectively explain most of the variability in the original set.
- PCA seeks a small number of dimensions that are as interesting as possible, where the concept of interesting is measured by the amount that the observations vary along each dimension.
- The first PC of features $X_1, \ldots, X_p$ is the normalized linear combination of the features $Z_1 = \phi_{11} X_1 + \ldots + \phi_{p1} X_P$ that has the largest variance. It is the direction in space in which the data vary most. By normalized, we mean $\sum_{j=1}^{p} \phi_{j1}^2 = 1$, constrained since otherwise would have arbitrarily large variance. The ϕ elements are the loadings of the first principal component, together the principal component loading vector.
    - Solves the problem $\underset{\phi_{11}, \ldots, \phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} x_{ij} \right)^2 \right\}$ subject to $\sum_{j=1}^{p} \phi_{j1}^2 = 1$
- We look for the linear combination of sample feature values z_{ij} , ie $\frac{1}{n} \sum_{i=1}^{n} z_{i1}^2$ subject to the constraint $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. The z-values are the scores of the first principal component. We assume that each of the X's has **mean zero** or we can center variables by subtracting off the mean. The objective that we are maximizing is just the sample variance of the n values of z_{i1}.

- The loading vector ϕ_1 with elements $\phi_{11}, \phi 21, \ldots, \phi p1$ defines a direction in feature space along which the data vary the most. If we project the n data points $x_1, \ldots, x_n$ onto this direction, the projected values are the principal component scores $z_{11}, \ldots, z_{n1}$ themselves. The score variable is the z - the data in the new coordinate system.
- The second principal component is the linear combination of $X_1, \ldots, X_p$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1. Therefore $\phi_2 \perp \phi_1$ - creating a new coordinate system along the lines of most variance
- 1) Find the principal components 2) Plot them against each other to produce low dimensional views of the data.
- An alternative interpretation for principal components can also be useful: principal components provide low-dimensional linear surfaces that are closest to the observations. The first principal component loading vector has a very special property: it is the line in p-dimensional space that is closest to the n observations in euclidean distance, ie. that minimizes the average squared length to all points. Together the first M principal component score vectors and the first M principal component loading vectors provide the best M-dimensional approximation (in terms of Euclidean distance) to the ith observation x_{ij}.
- Scaling - Because it is undesirable for the principal components obtained to depend on an arbitrary choice of scaling, we typically scale each variable to have standard deviation one before we perform PCA. If all variables are measured in the same units, we may not want to scale.
- Each principal component loading vector and each score vector is unique, up to a sign flip.
- We look at a biplot of the first PC vs the second PC to see how the original predictors align around components.
- Proportion of variance explained (PVE):
  - Total variance $\sum_{j=1}^{p} \mathrm{Var}(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2$
  - Variance explained by mth PC: $\frac{1}{n} \sum_{i=1}^{n} z_{im}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{jm} x_{ij} \right)^2$
  - PVE of the mth PC is the ratio of these quantities. We can sum over the first M PVEs to get an answer for first M PCs. Sum of all PVEs is 1. We can use a scree plot with the marginal PVE vs PC number to see where we get added benefit from additional PCs.
- PCA less suited when we expect some variables to be closer to others or some correlations would be more surprising than others.

K-means Clustering

- We seek to partition the observations into a pre-specified, non-overlapping number of clusters. Each observation belongs to exactly one cluster. A good clustering minimizes the within-cluster variation, ie $\underset{C_1, \ldots, C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}$ where $W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} \left(x_{ij} - x_{i'j} \right)^2$. Note this is equivalent to twice the squared distance from points to their centroids - this value is stricly non-increasing as we iterate.

1. Randomly assign cluster to each of the observations
2. Iterate over
    1. For each of K clusters, compute the cluster mean or centroid.
    2. Assign each observation to the cluster whose centroid is closest in euclidean distance.

- In Step 2(b), reallocating the observations can only improve the sum of distances from the centroids. This means that as the algorithm is run, the clustering obtained will continually improve until the result no longer changes; the objective function will never increase.
- Note this finds a local minimum, not a global - therefore results are dependent on the initial cluster assignments. Therefore we usually run the algorithm multiple times to increase probability we get the best assignments possible. The local minimum convergence is guaranteed, since we have a monotone decreasing sequence bounded below by 0, improving at each step.
- Works best for convex clusters

Hierarchical Clustering

- Bottom up agglomerative clustering
- As we move higher up the tree, branches themselves fuse, either with leaves or other branches. The earlier (lower in the tree) fusions occur, the more similar the groups of observations are to each other. We cannot draw conclusions about the similarity of two observations based on their proximity along the horizontal axis. Rather, we draw conclusions about the similarity of two observations based on the location on the vertical axis where branches containing those two observations first are fused.
- Hierarchical works best when the clusters are nested, can yield worse results when clusters slice populations in different ways - eg. gender and nationality.
- Starting out at the bottom of the dendrogram, each of the n observations is treated as its own cluster. The two clusters that are most similar to each other are then fused so that there now are n−1 clusters.

1. Begin with n observations and a dissimilarity measure. Treat each observation as its own cluster
2. For each cluster, examine all pairwise inter-cluster dissimilarities and identify the pair of clusters that are least dissimilar. Fuse these two clusters. The dissimilarity measure between these two clusters indicates the height in the dendrogram where the fusion is placed.
3. Repeat for remaining i-1 clusters.

- Measures of dissimilarity between clusters
    - Complete - among all pairwise dissimilarities in two clusters, take the largest
    - Single - among all pairwise dissimilarities in two clusters, take the smallest (can result in extended trailing clusters)
    - Average - among all pairwise dissimilarities in two clusters, take the average
    - Centroid - dissimilarity between centroids in two clusters (can result in bad inversions)
- Correlation-based distance considers two observations to be similar if their features are highly correlated, even though the observed values may be far apart in terms of Euclidean distance. Might use correlation based distance for two shoppers with similar tastes / spending patterns on very different scales, while euclidean distance would cluster shoppers by amount spent without regard to items.
- Scaling - might want to scale to SD 1 if on different units.

ESL Chapter 14 - Non-Linear Dimensionality Reduction

- Non linear dimensionality reduction methods are useful for datasets with high signal to noise ratios - the structure is pronounced, like the swiss roll example. If noise around the manifold increases, it is increasingly hard to see the structure of the lower dimensions.
- Suppose the data are clustered on a low dimensional manifold embedded in higher dimension space - may want the geodesic distance, the shortest distance on the manifold instead of in the higher dimensional space.

- Ambient dimensionality is the dim of the whole set of parameters. But the data may fall on a lower dimension - its intrinsic dimensionality.

Kernel Principal Components

- Kernel PCA expand the scope of PCA, mimicking what we would obtain if we were to expand the features by non-linear transformations and then apply PCA in this transformed space.
- Example - shells, concentric circles of classes. PCA wouldn't capture the circular patterns since all directions have equal variance. Throw in quadratic / some non-linear terms - it is the usual PCA algorithm applied to a new set of features remapped using a kernel $\langle \Phi(x_i), \Phi(x_k) \rangle$. Each circle is essentially a one dimensional space in two dimensions, but if noise increases too much then the distinct groups will be harder to see.
- Plug in the radial basis function and PCA can separate the circles into separate components in two dimensional space. Dependent on the tuning parameter $c = \frac{1}{\gamma}$ - captures the distance from the center.
- Choosing a kernel is not easy, but there are newer methods trying to learn the structure of the kernel. Other methods like LLE, Isomap reduce to kernel PCA with a certain kernel.

Locally Linear Embeddings (LLE)

- Take a higher dimensional space to a lower such that all of the relevant features are kept intact. Think of linearizing curves in calculus to work with a line around a certain point - assumption holds locally.
- Represent each sample as a linear combination of its neighbors - $x_i \approx \sum_{k=1}^{n} x_k W_{ik}$, then $W_{ik} \neq 0 \iff x_i, x_k$ are neighbors.
- Map each sample x_i to a point z_i in a low dimensional space st the local linear representation holds approximately, ie. $z_i \approx \sum_{k=1}^{n} z_k W_{ik}$. Maintain the relationships while mapping to a lower dimension.
- Find the weights W, fix W and find the optimal low dimensional embedding (solved by eigendecomposition). Number of weights is a hyperparameter
- Used for images of faces for example. Image is a matrix of pixels with grayscale values. Create 2D projection with LLE with 16 neighbors. While there are 560 features, they are all faces and should have lower dimensionality, certain regularity should be expected.

Multidimensional Scaling

- Project data onto a low-dimensional space, approx preserving the distance between every pair of samples.
- $d(i, j)$ is the distance between x's, then find z_i of every sample minimizing $\sum_{i,j} (d(i, j) - ||z_i - z_j||)^2$. Simply a least squares problem, but have to minimize over different mappings onto lower dimensions (say a 2D manifold parametrized somehow). The distance can be any distance, not just euclidean.

Isomap

- Modern version of multidimensional scaling
- Data clustered in low dimension manifold in higher dimensional space - swiss roll eg.
- Take the geodesic distance between samples as our $d(i, j)$.
- We don't know the manifold a priori, but the nearest neighbor graph approximates the shortest path. We can use this as a proxy for geodesic distance and apply multidimensional scaling
- Hands dataset - large dimensionality n x n pixels per image but all are visualizing very similar things. The images have data in similar locations.
- Signal to noise is good for many probles in computer vision, where components are distinct.

ESL - Missing Data

- We have missing data everywhere - from surveys, recommendation systems. Yelp has a skewed sample of reviewers. Dropout from longitudinal studies. Netflix prize - filling in the ratings of people without any.
- Missing completely at random
    - Pattern of missingness is independent of missing values and the values of any measured variables.
    - Eg. rate only 4 out of 20 drinks, but the 4 rated were chosen at random from the 20
- Missing at Random
    - Depends on other predictors but conditional on observed variables missingness is independent of missing value
    - Poor subjects less likely to answer question about drug use than the wealthy ones. Conditioning on income, whether they answer are not is independent of actual drug use.
- Missing not at random
    - Related to the missing variable even after correcting for measured variables.
    - Higher earners are less likely to report their income - we are interested in income and we are less likely to have data on it for high earners.
    - Censoring - follow people over time to see time until an event, but at some time point the study ends or there is dropout. We no longer observe the result after a time period.

Methods for Adjustments

- Categorical - treat missing as another category
- Surrogate - use other variables to account for missing variables. CART methods can have backup variables when data is missing for the original best split. We can also do this for other methods.
- Delete the observations missing values, but reduces the dataset size, add bias.
- Variable deletion - delete whole variables with missing values. Again can bias the feature space, may be a valuable variables with large differentiation.
- Single imputation - try to replace the missing value with a single number. Obviously in the hard sciences, this is not an acceptable solution, but depending on application this may work.
    - Replace with the mean or median of the column
    - Replace with a random sample from the non-missing values
    - Replace missing values in X_j with regression estimate from other predictors X_{-j}. Sort of the most logical imputation of a variable when we have other data.
    - However, 1 and 2 might introduce bias if the missing values are not completely random. Also the inference uncertainty parameters do not account for this additional uncertainty. Good application for bootstrap, the imputation step becomes part of the bootstrap function, then incorporates the imputation into the confidence intervals.
- Multiple Imputation - Assume the data follows a distribution. Generate many imputed datasets and sample from them
- Missing data in multiple variables
    - Iterative multiple imputation - start with a simple imputation, the iterate imputing the first variable given everything else, impute the 2nd given everything else... Think similar to backfitting using a regression of a variable against all others

    - Model based - posit a joint model for all variables, but rarely worth the trouble
    - Low rank matrix completion
        - $\hat{y}$ can be understood as a projection of y onto the space spanned by the columns of X in multiple regression. The columns space matters more than the X
        - If the predictor matrix is roughly low rank (ie. points lie near a lower dimensional subspace) then one can approximately recover X and its col space even if many entries are missing.
        - If sparse matrix of 1000 x 100, but the matrix is roughly rank 5, then it is reasonable to fill in the missing values using a similar matrix of rank 5.
        - Find a matrix X' which is similar to X in its non-missing values and has a low dimensional col space: $min_{st\ rank(X')=k}||X' - X||$ where $||X' - X||$ is the sum of squared diffs of the non-missing entries. The rank is unknown ahead of time and could be a tuning parameter.
        - Can relax to a convex optimization: $min||X' - X|| + \lambda \sum_1^p \sigma_p$ where $\sigma_1, \dots \sigma_p$ are the singular values of X'. Now this is solvable. Lambda is inversely proportional to the rank and can be a tuning parameter.
- Practical considerations
    - Plotting the data to see a pattern in missingness is worthwhile - say a doctor's office may have systematic errors. A histogram even might show a tight distribution of values, followed by some weird systematic issues. Could actually be informative and used as a dummy variable.
    - Weight variables by amount of missingness - weighted least squares. More error variance can be accounted for in weighting that down weights those variables.

Learning from Relational Data

- Observations have the form of a graphl. Each vertex can contain metadata beyond the relationships displayed in the graph.
- PageRank - uses a graph of links between websites to rank by importance.