Review Session

Prof. Alexandra Chouldechova
95-791: Data Mining

December 8, 2017

Agenda

- **Central themes**
- **Review of Supervised learning**
  - LDA, QDA
  - Regression, Classification methods list
  - Model selection approaches
  - Trees, Bagging, Random forests, Boosting
- **Review of Unsupervised learning**
  - Methods list
  - K-means, GMM's
  - Hierarchical clustering
  - Dimensionality Reduction
- **Overview: How to approach a particular task?**

Central themes

1. **Generalizability**
   - We want to construct models that:
     - Capture useful trends in the data (don't *underfit*)
     - Ignore meaningless random fluctuations in the training data (don't *overfit*)
   - To avoid underfitting:
     - Consider more flexible models
     - Try doing some feature engineering
   - To avoid overfitting:
     - Use Cross-validation to identify models with low test error
     - Perform variable selection to reduce model variability
     - Try regularized models (with CV to select regularization parameter)
2. **Bias-Variance tradeoff**
   - The *reducible* component of expected prediction error takes the form $\mathrm{bias}(\hat{f})^2 + \mathrm{var}(\hat{f})$
   - Using a model with high bias risks *underfit*
   - Using a model with with high variance risks *overfitting*

Central themes

3. **Interpretability vs. Flexibility**
   - Highly structured models tend to be easy to interpret. Examples:
     - Linear and Logistic regression (with small p)
     - Regularized regression
     - Small decision trees
     - Additive models (with small p)
   - Highly flexible models tend to do a better job at prediction, but are hard to interpret. Examples:
     - Large trees, Regression with large p
     - Bagging, Random Forests, Boosting

4. **Feature engineering**
   - In practice, whether a model gives good results depends *a lot* on which features are fed into the model
   - Transforming existing features can help performance *a lot*
     - Step functions, Polynomial regression, Splines, Additive models
   - Data Visualization and Unsupervised learning techniques can help us to construct new features (e.g., PCA, Clustering)

Supervised learning methods: Regression

- Linear regression
- Polynomial regression, Step functions, Splines, Local regression
- Additive models
- Regularized regression (Lasso)
- k-Nearest Neighbours
- Decision trees
- Bagging
- Random forests
- Boosting

Supervised learning methods: Classification

- Logistic regression
  - Generalized Additive Models
  - Regularized logistic regression
- K-nearest neighbours
- Bayes methods:
  - Linear discriminant analysis
  - Quadratic discriminant analysis
  - Naive Bayes
- Classification trees
- Bagging
- Random forests
- Boosting

Model selection approaches

- Cross-validation
  - Calculate CV error for a bunch of models, and pick the one with the lowest CV error
  - Or: use the 1-SE rule
  - Note: If you are considering many different models, you should assess your final model on an unseen set of test data

- Best subset selection, Forward stepwise selection (to select variables)
  - Both methods give us the *best* k-predictor model for $k = 0, \ldots, p$.
  - Can use AIC or BIC to select best model size
  - Best subset selection is computationally inefficient
  - Forward selection is OK, but Lasso generally works better if p *isn't* small

- Regularized regression (Lasso)
  - Automatically performs variable selection
  - Reduces model variance by shrinking estimated coefficients $\hat{\beta}_j$ towards 0

Bayes Methods

Bayes' theorem gives us a way of combining prior probabilities π_k, feature densities $f_k(x)$ and observed data to get a posterior probability that an observation with inputs x_0 is in class $y_0 = k$.

$$\hat{y}_0 = \underset{k=1,\ldots,K}{\operatorname{argmax}} \underbrace{\frac{\pi_k f_k(x_0)}{\sum_{\ell=1}^{K} \pi_\ell f_\ell(x_0)}}_{\text{posterior prob.}} = \underset{k=1,\ldots,K}{\operatorname{argmax}} \; \pi_k f_k(x_0)$$

- Linear discriminant analysis (LDA) assumes all the $f_k(x)$ are Multivariate Normal(μ_k, Σ)
  - Different means, *same* covariance matrix
- Quadratic discriminant analysis (QDA) assumes all the $f_k(x)$ are Multivariante Normal(μ_k, Σ_k)
  - Different means, different variances
- Naive Bayes assumes that within each class, all of the inputs are independent: $f_k(x) = \prod_{j=1}^{p} f_{k,j}(x_j)$

LDA vs. QDA

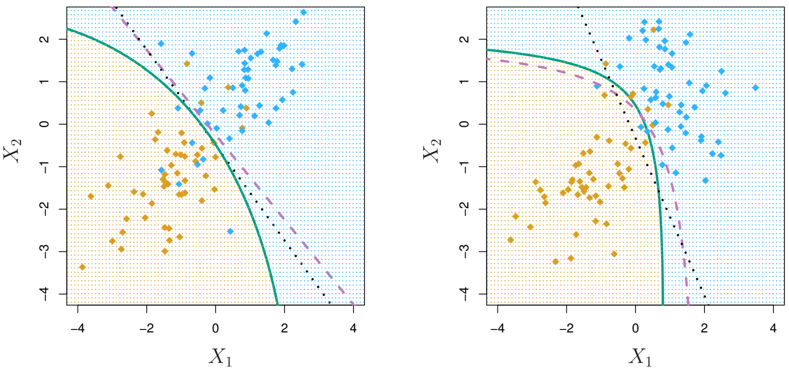


Figure 4.9 from ISL. Dashed purple curve is the Bayes classifier decision boundary. Solid green curve is QDA, dotted black line is LDA. Left: True boundary is linear. Right: True boundary is quadratic.

Naive Bayes vs. LDA vs. QDA

- Naive Bayes scales well to problems where p is large
    - If you have enough data to estimate the univariate density of each predictor (i.e., enough to form a *nice* histogram), you can apply Naive Bayes

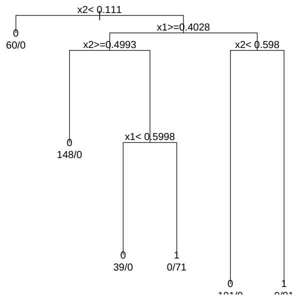
- In LDA, we have to estimate $K \times p$ parameters to get $\hat{\mu}_k$'s and another $\frac{1}{2}p(p+1)$ parameters to estimate the $p \times p$ covariance matrix $\hat{\Sigma}$.

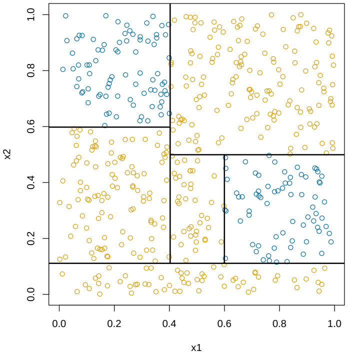
- In QDA, we have to estimate the means and K $p \times p$ covariance matrices. That's $\frac{1}{2}Kp(p+1)$ parameters!

- So why do even bother with methods like LDA or QDA?
    - They can capture meaningful interactions. Naive Bayes cannot.

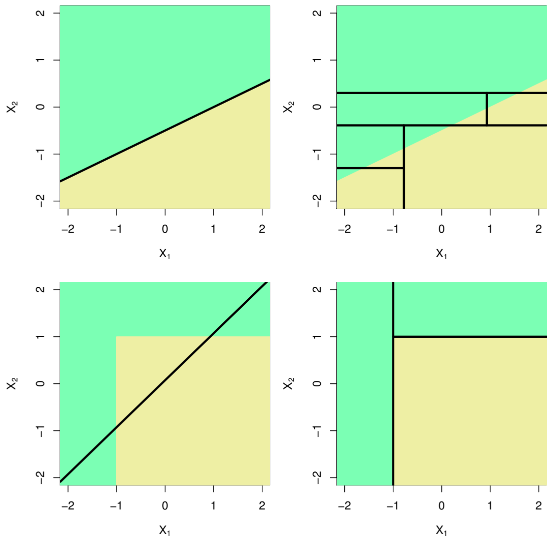
Decision trees (CART)

- CART builds a deep tree via the recursive binary partitioning algorithm; the tree is then pruned to reduce variance
- At each step, one of the existing rectangles is split into two parts
- The splitting variable and splitting location is chosen to maximize the decrease in error (RSS for regression, Gini index for classification)
- Output: A tree representation, and a partition of the space into rectangles





Trees vs. Linear models



ISL Figure 8.7. Trees are bad when the boundary is linear, but very good when the boundary is well-described by a simple rectangular partition.

Classification trees vs. Other methods

Method	Interpretable	Flexible	Makes assumptions?
Logistic regression	Yes	Extensible	Yes
k-NN	No	Highly	No
LDA/QDA	Sometimes	No	Yes
Trees	Yes[1]	Somewhat	No

- Trees don't assume any particular relationship between the response Y and the inputs X_j, and large trees are quite flexible
- They tend to be poor predictors/classifiers because they are highly variable
  - Small changes in the training data can produce big changes in the resulting tree
- We can built ensembles of trees via Bagging, Random forests, and Boosting…but we lose interpretability

[1]Assuming the tree isn't too large

Bagging: Classification trees

- Given a training data (x_i, y_i), $i = 1, \ldots n$, bagging averages the predictions from classification trees over a collection of bootstrap samples.
    - If we average over a bunch of high-variance, low-bias methods, we decrease the variance, while retaining low bias

- Here we'll describe how to apply Bagging to Classification Trees
    1. For $b = 1, \ldots, B$, get a bootstrap sample of size n from the training data: (x_i^{*b}, y_i^{*b}), $i = 1, \ldots n$
    2. Fit a classification tree $\hat{f}^{\text{tree},b}$ on each sample
    3. Classify a new point x_0 by aggregating the votes (or probability estimates) across all B trees

- In Step (3), we can use the Consensus approach (classify to the class with the most votes), or the Probability approach (average the class probability estimates across all trees, classify to highest average probability)

- In Step (2) the trees are grown very large, with no pruning.

Random Forests

- Random forests provide an improvement over bagged trees by incorporating a small tweak that decorrelates the individual trees
  - This further reduces variance when we average the trees

- We still build each tree on a bootstrapped training sample

- But now, each time a split in a tree is considered, the tree may only split on a predictor from a randomly selected subset of m predictors

- A fresh selection of m randomly selected predictors is presented at each split... not for each tree, but for each split of each tree

- $m \approx \sqrt{p}$ turns out to be a good choice[2]
  - E.g., if we have 100 predictors, each split will be allowed to choose from among 10 randomly selected predictors

[2] $m = p/3$ is R's `randomForest` default for regression, $m = \sqrt{p}$ is the default for classification

Boosting algorithm: Regression trees

1. Set $\hat{f}(x) = 0$ and *residuals* $r_i = y_i$ for all i in the training set
2. For $b = 1, 2, \ldots, B$, repeat:
    1. Fit tree $\hat{f}^b$ with d splits to the training data (X, r)[3]
    2. Update $\hat{f}$ by adding *shrunken* version of the new tree:
    $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$
    3. Update the residuals
    $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

- λ is called the shrinkage parameter. Typically, $\lambda \ll 1$

[3] We're treating the residual vector r as our outcome at each step.

What's boosting trying to do?

- Boosting works best if d (the size of each tree) is small
- Given the current model, we fit a decision tree to the *residuals* from the model
- This new tree helps us perform just a little bit better in places where the current model wasn't doing well
- Unlike bagging, where each tree is large and tries to model the entire (bootstrapped) data well, each tree in boosting tries to incrementally improve the existing model
- Think of boosting as learning slowly: We use small trees, try to make incremental improvements, and further slow down the learning process by incorporating the *shrinkage parameter* λ

Random forests

- Uses bagging: Builds B trees from bootstrap samples
- Trees can be grown in parallel
- Each tree tries to model the full outcome y
- Trees are grown large, left unpruned
- Tuning parameters:
  - B - number of trees (large B is good, helps reduce variance)
  - m - number of predictors considered at each split (small m is good, helps de-correlate the trees)

Boosted trees

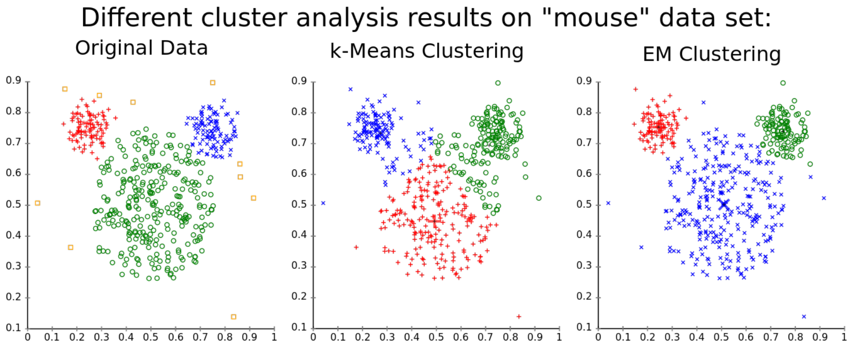
- Uses B trees, each fit to the entire training data
- Trees are grown sequentially
- Instead of modeling full outcome y, at each step a tree is built to model r, the latest residual
- Trees are grown very small, “stumps” (d = 1) are popular
- Tuning parameters:
  - B - number of trees (if B is too large, big risk of overfitting)
  - d - Tree depth (“interaction depth”)
  - λ - Shrinkage parameter (“learning rate”)

- Both are examples of *ensemble methods*, which are models that form estimates by combining the results of a bunch of other models

Unsupervised learning methods

- K-means clustering
- Hierarchical clustering
  - Single linkage
  - Complete linkage
  - Average linkage
- Gaussian Mixture Models (“EM” clustering)
- Association rule mining
- Principal components analysis

Gaussian Mixture Modeling vs. K-means



- GMM's do better on this example because they essentially allow for a data-adaptive notion of *distance* when assigning points to centroids
- i.e., In the original data, we have 2 clumps with small variance, and one clump with large variance
- K-means can't capture this added information
- GMM's say: An observation belongs to C_k if its *variance-adjusted* (i.e., Σ_k-adjusted) distance to μ_k is small

K-means

- Assumes spherical clusters
- Minimizes within-cluster sum of squared distances
- Hard assignment of points to clusters

Gaussian Mixture Models

- Explicitly allows for non-spherical clusters (via Σ_k)
- Minimizes a slightly different distance, which accounts for priors $\hat{\pi}_k$ and covariances $\hat{\Sigma}_k$
- Soft (probabilistic) assignment of points to clusters

- Both require K to be specified in advance, results can change a lot across different choices of K
- Both are fit with very similar looking iterative algorithms
- Algorithms converge to something, but not necessarily the global optimum
- Results depend on random initialization
  - Try different random initializations, and keep the best result

Hierarchical clustering

- We discussed agglomermerative ("bottom-up") hierarchical clustering
- Hierarchical clustering is performed using a dissimilarity metric $d(x_i, x_{i'})$ telling us how dissimilar two points are, and a linkage $d(G, H)$ telling us how dissimilar two clusters are

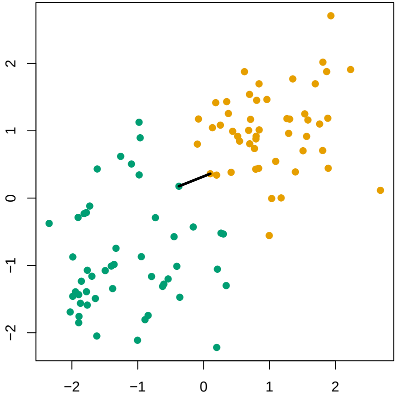
Linkage	*Description*
Complete	Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *largest* of these dissimilarities.
Single	Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *smallest* of these dissimilarities.
Average	Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *average* of these dissimilarities.

Single linkage

In single linkage (i.e., nearest-neighbor linkage), the dissimilarity between G, H is the smallest dissimilarity between two points in different groups:

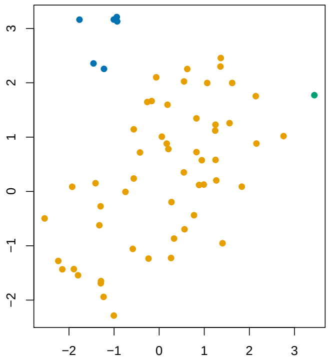
$$d_{\text{single}}(G,H) = \min_{i \in G,\, j \in H} d(x_i, x_j)$$

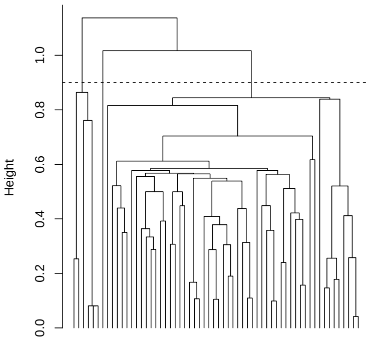
Example (dissimilarities d_{ij} are distances, groups are marked by colors): single linkage score $d_{\text{single}}(G,H)$ is the distance of the closest pair



Single linkage example

Here $n = 60$, $x_i \in \mathbb{R}^2$, $d_{ij} = \|x_i - x_j\|_2$. Cutting the tree at $h = 0.9$ gives the clustering assignments marked by colors





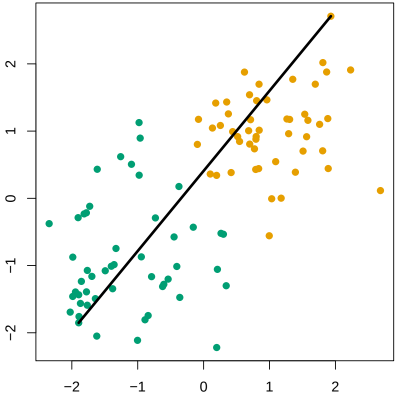
Cut interpretation: for each point x_i, there is another point x_j in its cluster such that $d(x_i, x_j) \leq 0.9$

Complete linkage

In complete linkage (i.e., furthest-neighbor linkage), dissimilarity between G, H is the largest dissimilarity between two points in different groups:

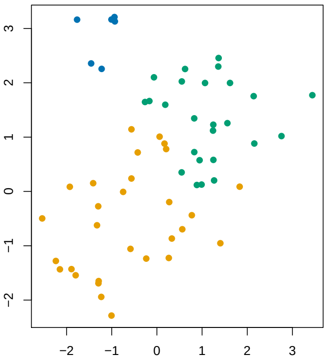
$$d_{\text{complete}}(G, H) = \max_{i \in G,\, j \in H} d(x_i, x_j)$$

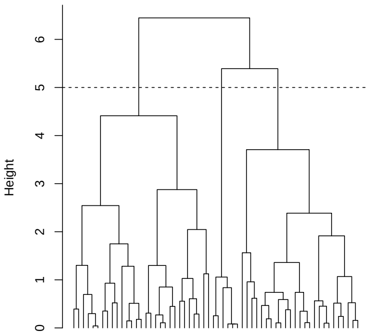
Example (dissimilarities d_{ij} are distances, groups are marked by colors): complete linkage score $d_{\text{complete}}(G, H)$ is the distance of the furthest pair



Complete linkage example

Same data as before. Cutting the tree at $h = 5$ gives the clustering assignments marked by colors





Cut interpretation: for each point x_i, every other point x_j in its cluster satisfies $d(x_i, x_j) \leq 5$

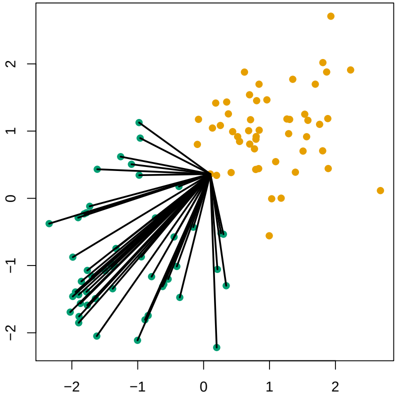
Average linkage

In average linkage, the dissimilarity between G, H is the average dissimilarity over all points in opposite groups:

$$d_{\text{average}}(G,H) = \frac{1}{|G|\cdot|H|}\sum_{i\in G,\, j\in H} d(x_i, x_j)$$

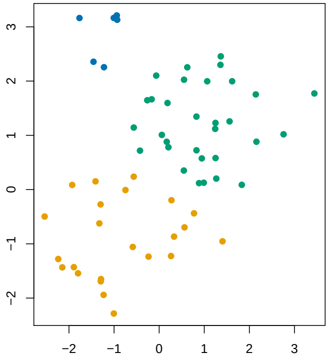
Example (dissimilarities d_{ij} are distances, groups are marked by colors): average linkage score $d_{\text{average}}(G,H)$ is the average distance across all pairs

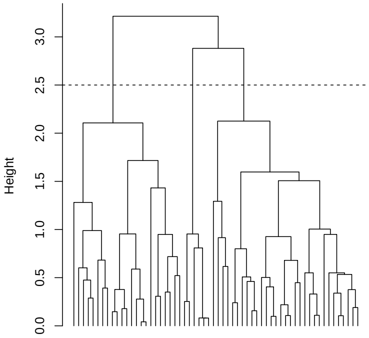
(Plot here only shows distances between the alert points and one orange point)



Average linkage example

Same data as before. Cutting the tree at $h = 2.5$ gives clustering assignments marked by the colors





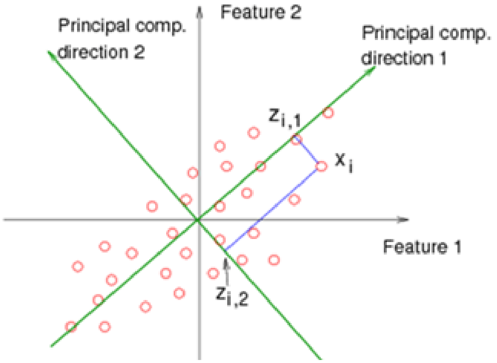
Cut interpretation: there really isn't a good one! ☹

Shortcomings of Single and Complete linkage

Single and complete linkage have some practical problems:

- Single linkage suffers from chaining.
  - In order to merge two groups, only need one pair of points to be close, irrespective of all others. Therefore *clusters can be too spread out*, and not compact enough.
- Complete linkage avoids chaining, but suffers from crowding.
  - Because its score is based on the worst-case dissimilarity between pairs, *a point can be closer to points in other clusters than to points in its own cluster*. Clusters are compact, but not far enough apart.
- Average linkage tries to strike a balance. It uses average pairwise dissimilarity, so clusters tend to be relatively compact and relatively far apart
  - But, average linkage doesn't give us a nice interpretation when we cut the dendrogram at height h, *and* the results change if we apply even a *monote increasing transformation* to the dissimilarity: E.g., $d \to d^2$ or $d \to \frac{e^d}{1+e^d}$

Dimensionality reduction via PCA: A picture

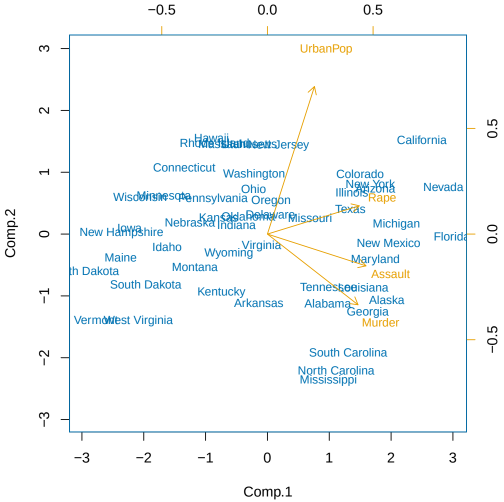


[source: https://onlinecourses.science.psu.edu/stat857]

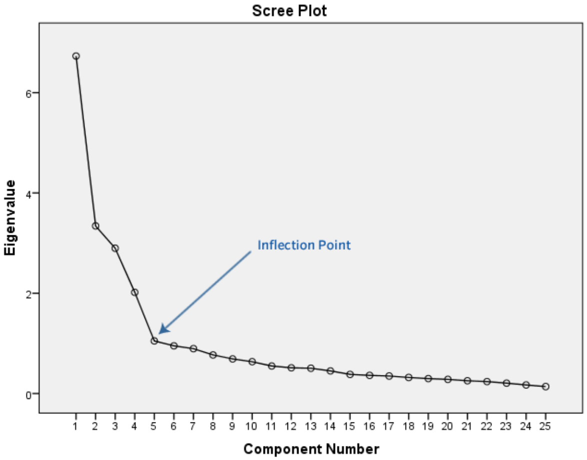
- This Figure shows an observation $x_i = (x_{i1}, x_{i2})$ along with z_{i1}, its projection onto the first principal component direction, and z_{i2}, its projection onto the second principal component direction

Biplot of US Arrests data

```
arrests.pca$loadings = -arrests.pca$loadings # Flip loadings (phi's)
arrests.pca$scores = -arrests.pca$scores # Flip scores (z's)
biplot(arrests.pca, scale = 0) # Construct biplot
```



Selecting number of PCA components: Finding elbows in scree plots



[source: https://gugginotes.wordpress.com/]

- *Eigenvalue* y-axis label should be interpreted as PVE
- Rule-of-thumb: Stop at the *elbow* in the Scree plot. ($k = 5$ here)

Other things you should know

- Using principal components as variables in regression is called Principal Components Regress (PCR)
- PCR can fail if the top principal components aren't associated with Y
- Correspondence Analysis extends PCA to handle categorical variables
- Multidimensional Scaling (MDS) produces low-dimensional projections of data based on dissimilarity matrices

Cross-validation: Right and Wrong

- Suppose we're in a classification setting with $Y \in \{0, 1\}$
- Getting data points is really *expensive*, so we wind up with only $n = 120$ observations
- However, we have *a lot* of features collected on each observation: $p = 5000$
- Fitting a model with $n = 120$ and $p = 5000$ is hopeless, so we take the following approach:
  1. For each of the $p = 5000$ features, calculate the feature's correlation with y. Identify the $q = 100$ predictors that have the highest correlation with the outcome y.
  2. Apply a classifier using just these 100 predictors
- Question: How do we estimate the test error of our classifier?
- Can we apply Cross-validation to just Step 2?

Cross-validation: The wrong way

1. For each of the $p = 5000$ features, calculate the feature's correlation with y. Identify the $q = 100$ predictors that have the highest correlation with the outcome y.
2. Apply a classifier using just these 100 predictors

- It is very wrong to estimate test error by applying Cross-validation just on Step 2
- Doing so would ignore the fact that we *used the y values of the training data in Step 1*
- To get valid error estimates, each step of Cross-validation must go through every part of the model-fitting process that in any way makes use of the outcome variable y
- If we just Cross-validate Step 2, we can easily wind up with a CV error estimate of 0% when the true test error is 50%.

WRONG approach

Step 1

Identify the 100 predictors with the highest value of

$\mathrm{corr}(X_j, y)$

Cross-validation

Split data into K folds

```
For $k$ = 1, 2, …, K:
    -Fit classifier using all the 100 selected predictors
        on all observations except those in fold $k$
    -Use classifier to predict labels for data in fold $k$
    -Calculate test error on fold $k$
Return: average test error across all of the folds
```

RIGHT approach

Cross-validation

Split data into K folds

```
For $k$ = 1, 2, …, K:
  -Using observations not in fold $k$, identify the 100 predictors with the highest value of corr($X_j$, y)
  -Fit classifier using these 100 selected predictors
     on all observations except those in fold $k$
  -Use classifier to predict labels for data in fold $k$
  -Calculate test error on fold $k$
Return: average test error across all of the folds
```

Classification with imbalanced data

- Suppose that we're trying to identify *fraudulent transactions*
- We build a random forest classifier, and find that it gets 97% accuracy
- Looking back at the data, we see that just 3% of our observations were from fraudulent transactions
- Our random forest got 97% accuracy just by classifying $\hat{y}_i =$ `not fraudulent` for every observation
- This is an example of imbalanced data
- *Most* classification problems you'll encounter in the real world are imbalanced in this way

Strategies for dealing with imbalanced data

- Stop using *classification accuracy* as your performance metric
    - Instead, assess how well your classifier is doing using more task-relevant metrics: Precision, Recall (Sensitivity), Specificity, PPV, profit

- Take steps to artificially improve balance:
    - Randomly remove instances of your over-represented class (under-sample)

    - Randomly add copies of your under-represented class (over-sample, sampling *with* replacement)

    - Cluster your over-represented observations into a large number of clusters, and select just one representative observation from each class.

    - Create synthetic samples from your under-represented class using a method like SMOTE (Synthetic Minority Over-sampling Technique)

- If using trees or forests: Try growing very deep trees and applying a cost-sensitive pruning technique

How do we approach a particular data analytic task?

In the next few slides I'm going to discuss general “rules of thumb” for approaching a data analytic tasks.

!!Disclaimer!!: There are many reasonable ways to approach problems. The suggestions made in the subsequent slides are at best a helpful over-simplification.

How do we approach a particular data analytic task?

Here's a sequence of questions that you should ask yourself when trying to decide what method(s) to use.

- Is it a supervised or unsupervised learning problem?
- **Unsupervised**:
  - Are trying to group together the observations (cluster), group together the features (cluster) or construct a low-dimensional representation of our data (dimensionality reduction)?
  - If it's a clustering problem:
    - Is the data set very large?
    - > K-means computation scales as $O(n)$, hierarchical clustering is generally $O(n^3)$, so hierclust may not work on large data sets
    - How many clusters do we expect? What shape do we expect/want them to take?
    - > Hierarchical clustering allows us to consider different choices of K in a manner that produces easily comparable clusterings from one K to the next
    - > K-means and GMM's can give very different clusterings for different choices of K

How do we approach a particular data analytic task?

- **Unsupervised**:
  - Do we want to scale the points in any way?
  - If it's a hierarchical clustering problem:
    - What kinds of similarity do we want our clustering to capture?
    - > Use a dissimilarity metric that reflects this.
    - Do we need a guarantee on the "connectedness" of our clusters when we cut the dendrogram at level h? If yes, choose linkage accordingly.s
    - > Single linkage: Ensures that for every cluster C_k and every $x_i \in C_k$, there's at least one other point $x_{i'} \in C_k$ with $d(x_i, x_{i'}) \leq h$
    - > Complete linkage: Ensures that for every cluster C_k, and every pair of points $x_i, x_{i'} \in C_k$, we have $d(x_i, x_{i'}) \leq h$
    - Other good/bad aspects of the 3 linkages we discussed are covered in earlier slides
  - If it could be a K-means or GMM problem:
    - Is n sufficiently large and p sufficiently small to be able to estimate the covariance matrices Σ_k for GMM's?
    - Might we have ellipsoidal clusters or spherical clusters where some are more diffuse than others? (GMM's can work better here)
    - Do we want a *hard clustering* (each point gets assigned a cluster), or a *soft clustering* (get an estimated probability that point i is in cluster C_k)

How do we approach a particular data analytic task?

- **Supervised**:
  - Should you use Classification or Regression? Might both work?
  - How many observations are there?
  - How many features are there?
  - Is the sample size much larger than the number of features?
  - Do I need a very interpretable model?
  - Are there many highly correlated predictors?
  - Are most of the features likely to be relevant?
  - For Classification in particular:
    - What's more costly: False Positives or False Negatives?
    - What is the prevalence of the interesting class? Is the data imbalanced?
    - Do we need good probability estimates, or do we just care about getting the right class labels?

Problem: Small n, large p

- If we have a lot of features and not very many observations, we can try:
  - Variable selection
  - Regularized regression
  - Cluster features or run PCA to reduce dimension
  - Feature screening
    - E.g., Calculate correlations $\mathrm{cor}(X_j, y)$, and keep only the features that have high marginal correlation with y
  - Naive Bayes
- We should avoid methods that tend to have high variability (HV), or which require a lot of parameters to be estimated (LP). E.g., avoid:
  - (HV) Large decision trees (Only use pruned trees!)
  - (HV) Unregularized regression
  - (HV) k-Nearest Neighbours
  - (LP) QDA
- Bagging, Random forests and Boosting may still give good results. Proceed with caution, and carefully apply Cross-validation to avoid over-fitting.

Problem: Complex relationship between y and inputs

- Assuming you have the sample size to reliably fit flexible models, try:
  - Decision trees
  - Bagging, Random forests, Boosting
  - LDA, QDA (for classification)
  - Additive models
  - k-Nearest Neighbours
    - k-NN can work well if p is quite small, or if you carefully construct a distance metric that measures distance in just the *relevant directions*

- If you believe there may be important interaction effects among the inputs, avoid:
  - Additive models
  - Naive Bayes
  - Logistic regression (without interactions)

Acknowledgements

All of the lectures notes for this class feature content borrowed with or without modification from the following sources:

- 36-462/36-662 Lecture notes (Prof. Tibshirani, Prof. G'Sell, Prof. Shalizi)
- 95-791 Lecture notes (Prof. Dubrawski)
- *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani
- *Applied Predictive Modeling*, (Springer, 2013), Max Kuhn and Kjell Johnson