Machine Learning for Political Science: Day 12

Kenneth Benoit

ANU Masterclass 2020

30 October 2020

Outline

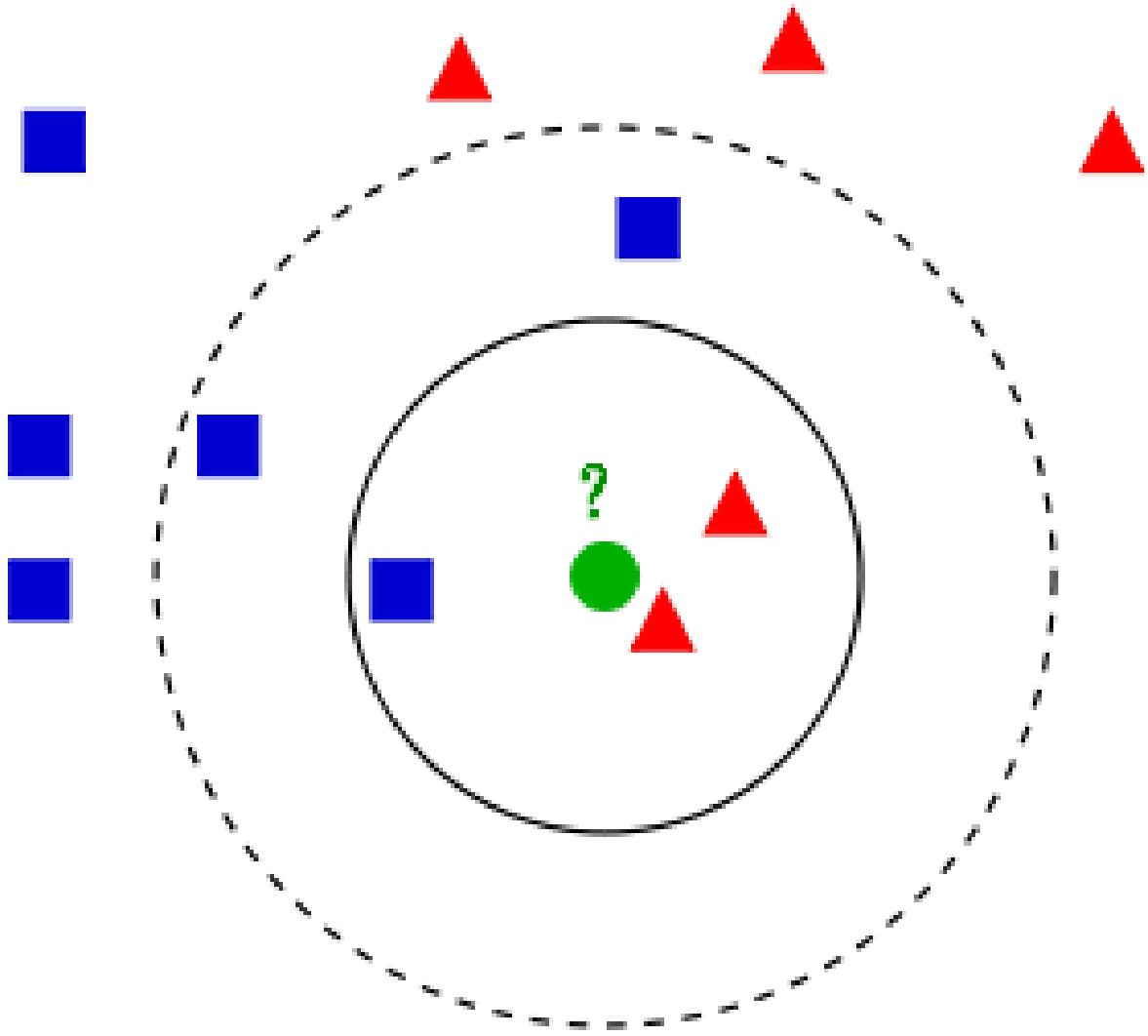
- k-Nearest Neighbour
- Scaling distance
- Clustering
  - k-means clustering
  - Hierarchical clustering
- Cross-validation
  - Validation-set approach
  - K-fold Cross-validation

k-Nearest Neighbour

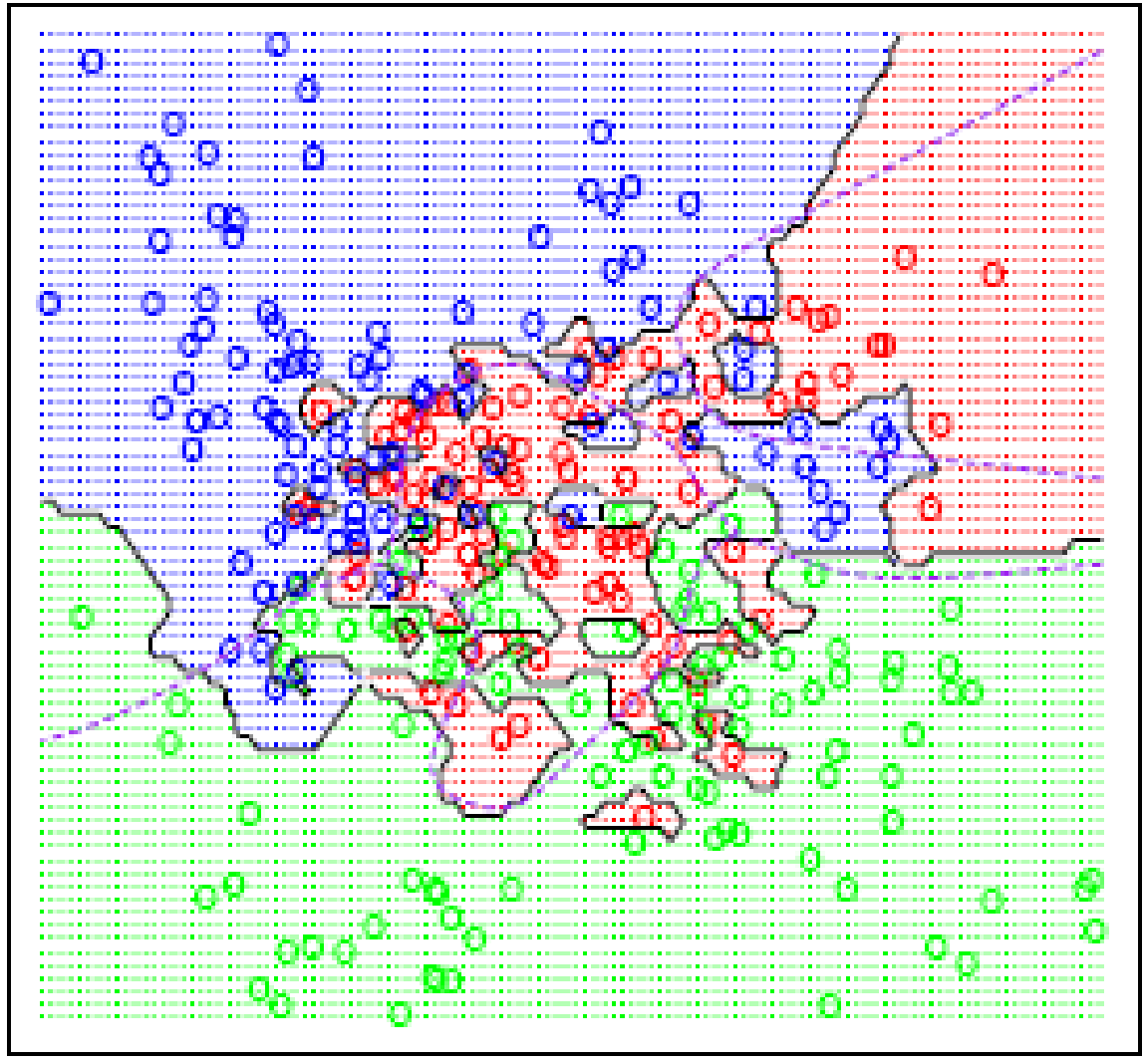
k-nearest neighbour

- A non-parametric method for classifying objects based on the training examples taht are *closest* in the feature space
- A type of instance-based learning, or "lazy learning" where the function is only approximated locally and all computation is deferred until classification
- An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors (where k is a positive integer, usually small)
- Extremely *simple*: the only parameter that adjusts is k (number of neighbors to be used) - increasing k *smooths* the decision boundary

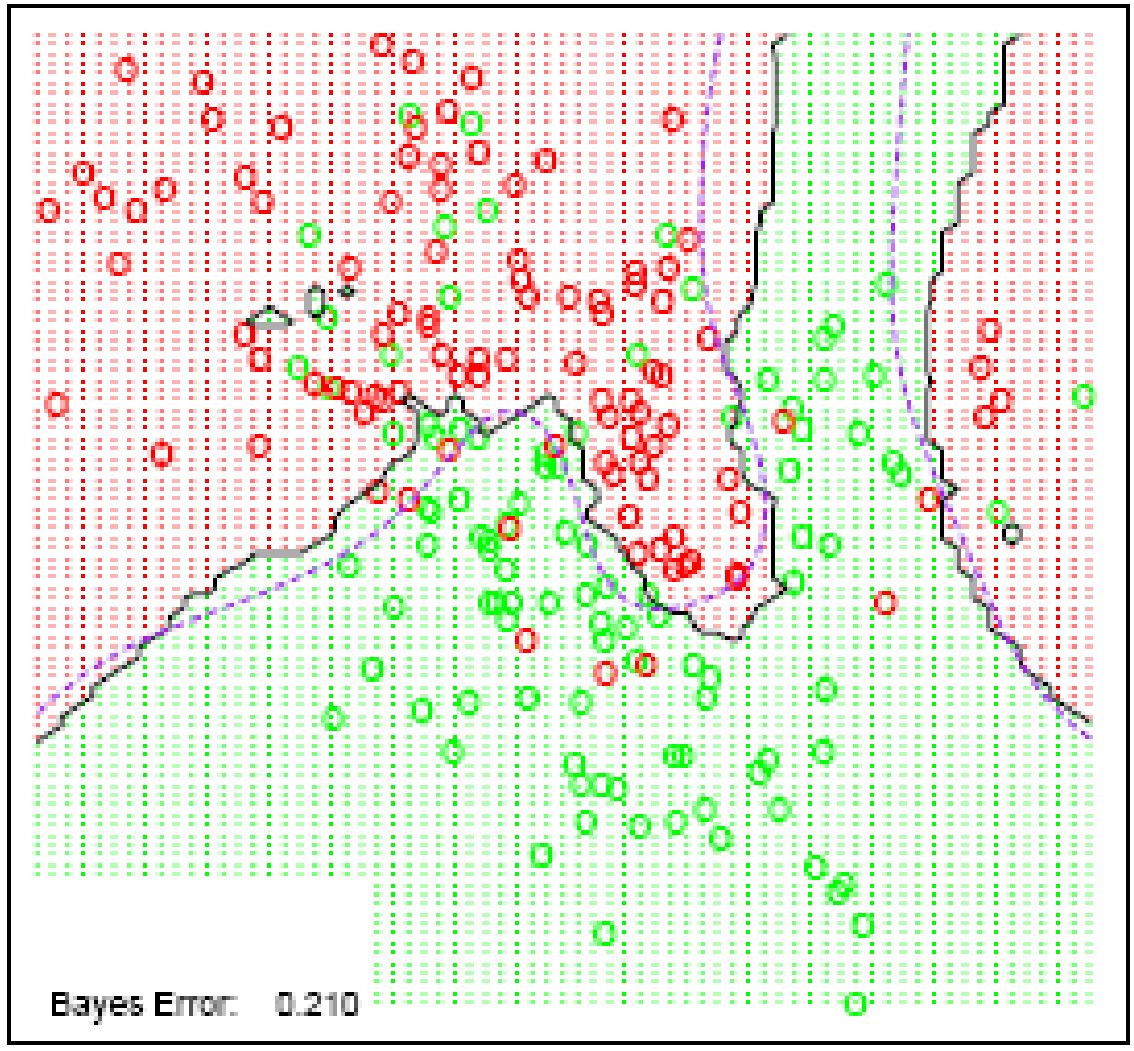
k-NN Example: Red or Blue?



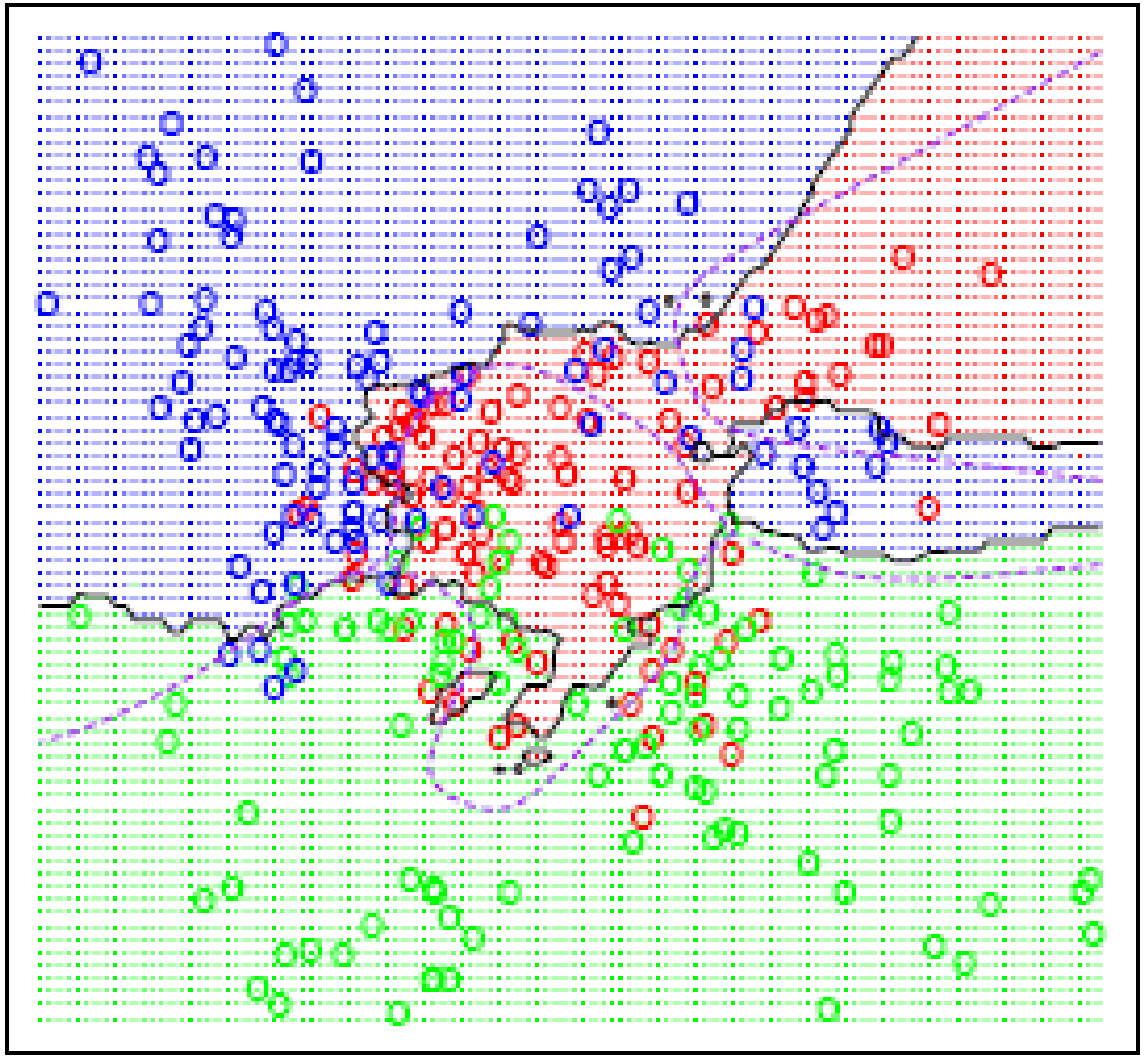
$k = 1$



$k = 7$



$k = 15$



Classifying amicus curiae briefs (Evans et al 2007)

```
## kNN classification
require(class)

## Loading required package:  class

require(quanteda, warn.conflicts = FALSE, quietly = TRUE)

## Package version:  2.1.2
## Parallel computing:  12 of 12 threads used.
## See https://quanteda.io for tutorials and examples.

data(data_corpus_amicus, package = "quanteda.corpora")
# create a matrix of documents and features
amicusDfm <- dfm(data_corpus_amicus, remove = stopwords("english"), stem = TRUE
# threshold-based feature selection
amicusDfm <- dfm_trim(amicusDfm, min_termfreq = 10, min_docfreq = 20)
```

Classifying amicus curiae briefs (Evans et al 2007)

```
# tf-idf weighting
amicusDfm <- dfm_tfidf(amicusDfm)
# partition the training and test sets
train <- amicusDfm[!is.na(docvars(data_corpus_amicus, "trainclass")), ]
test  <- amicusDfm[!is.na(docvars(data_corpus_amicus, "testclass")), ]
trainclass <- docvars(data_corpus_amicus, "trainclass")[1:4]
```

Classifying amicus curiae briefs (Evans et al 2007)

```
# classifier with k=1
classified <- knn(train, test, trainclass, k = 1)
table(classified, docvars(data_corpus_amicus, "testclass")[-c(1:4)])

##
## classified AP AR
##          P  0  0
##          R 19 79
```

Classifying amicus curiae briefs (Evans et al 2007)

```
# classifier with k=2
classified <- knn(train, test, trainclass, k=2)
table(classified, docvars(data_corpus_amicus, "testclass")[-c(1:4)])

##
## classified AP AR
##          P  9 43
##          R 10 36
```

k-nearest neighbour issues: Dimensionality

- Distance usually relates to all the attributes and assumes all of them have the same effects on distance
- Misclassification may results from attributes not confirming to this assumption (sometimes called the "curse of dimensionality") – solution is to reduce the dimensions
- There are (many!) different *metrics* of distance

Scaling distance

Unsupervised "learning": scaling distance

- Features are treated as a quantitative matrix of variable values features
  - often normalized or standardized to allow similar computations of distance
- Many possible definitions of *distance* exist
  - see for instance `summary(pr_DB)` from `proxy` library
- Works on any quantitative matrix of features

Distance measures

```
library(proxy, warn.conflicts = FALSE, quietly = TRUE)
summary(pr_DB)

## * Similarity measures:
## Braun-Blanquet, Chi-squared, correlation, cosine, Cramer, Dice, eDice,
## eJaccard, Fager, Faith, Gower, Hamman, Jaccard, Kulczynski1,
## Kulczynski2, Michael, Mountford, Mozley, Ochiai, Pearson, Phi,
## Phi-squared, Russel, simple matching, Simpson, Stiles, Tanimoto,
## Tschuprow, Yule, Yule2
##
## * Distance measures:
## Bhjattacharyya, Bray, Canberra, Chord, divergence, Euclidean, fJaccard,
## Geodesic, Hellinger, Kullback, Levenshtein, Mahalanobis, Manhattan,
## Minkowski, Podani, Soergel, supremum, Wave, Whittaker
```

Example: text, representing documents as vectors

- The idea is that (weighted) features form a vector for each document, and that these vectors can be judged using metrics of similarity
- A document's vector for us is simply (for us) the row of the document-feature matrix

What a distance matrix looks like

$$\begin{bmatrix} & 0.3 & 0.4 & 0.7 \\ 0.3 & & 0.5 & 0.8 \\ 0.4 & 0.5 & & 0.45 \\ 0.7 & 0.8 & 0.45 & \end{bmatrix}$$

For instance, the dissimilarity between the first and second observations is 0.3, and the dissimilarity between the second and fourth observations is 0.8.

USArrests dataset example

```
head(USArrests, 10)
```

```
##             Murder Assault UrbanPop Rape
## Alabama       13.2     236       58 21.2
## Alaska        10.0     263       48 44.5
## Arizona        8.1     294       80 31.0
## Arkansas       8.8     190       50 19.5
## California     9.0     276       91 40.6
## Colorado       7.9     204       78 38.7
## Connecticut    3.3     110       77 11.1
## Delaware       5.9     238       72 15.8
## Florida       15.4     335       80 31.9
## Georgia       17.4     211       60 25.8
```

USArrests dataset example

```r
as.matrix(dist(USArrests))[1:5, 1:5]
```

```
##             Alabama   Alaska   Arizona  Arkansas California
## Alabama     0.00000 37.17701  63.00833  46.92814   55.52477
## Alaska     37.17701  0.00000  46.59249  77.19741   45.10222
## Arizona    63.00833 46.59249   0.00000 108.85192   23.19418
## Arkansas   46.92814 77.19741 108.85192   0.00000   97.58202
## California 55.52477 45.10222  23.19418  97.58202    0.00000
```

```r
as.matrix(dist(USArrests, method = "manhattan"))[1:5, 1:5]
```

```
##             Alabama Alaska Arizona Arkansas California
## Alabama         0.0   63.5    94.9     60.1       96.6
## Alaska         63.5    0.0    78.4    101.2       60.9
## Arizona        94.9   78.4     0.0    146.2       39.5
## Arkansas       60.1  101.2   146.2      0.0      148.3
## California     96.6   60.9    39.5    148.3        0.0
```

Euclidean distance

Between document A and B where j indexes their features, where y_{ij} is the value for feature j of document i

- Euclidean distance is based on the Pythagorean theorem
- Formula

$$\sqrt{\sum_{j=1}^{j}(y_{Aj} - y_{Bj})^2} \tag{1}$$

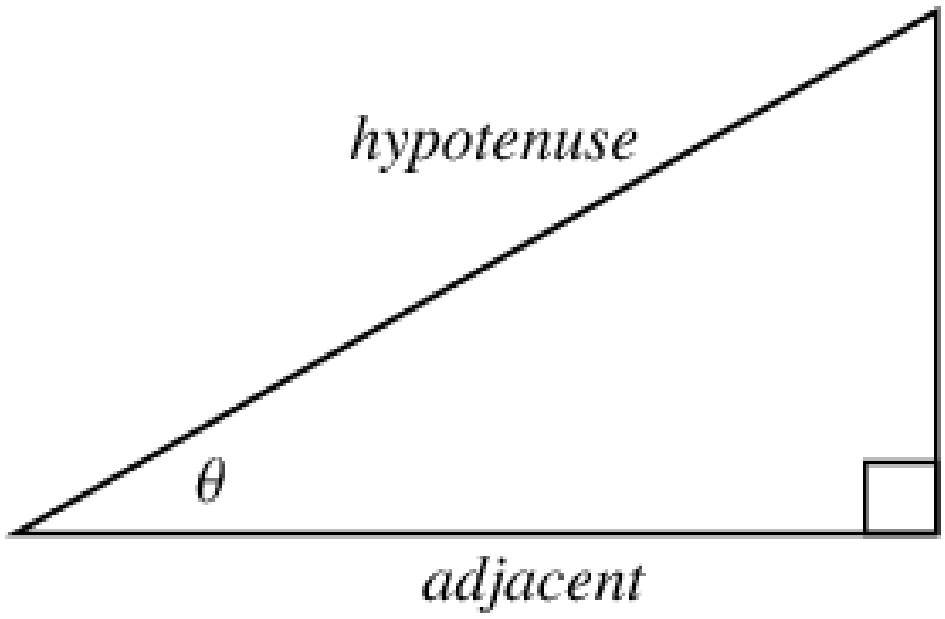
- In vector notation:

$$\|\mathbf{y}_A - \mathbf{y}_B\| \tag{2}$$

- Can be performed for any number of features J (or V as the vocabulary size is sometimes called – the number of columns in of the dfm, same as the number of feature types in the corpus)

A geometric interpretation of "distance"

In a right angled triangle, the cosine of an angle θ or $\cos(\theta)$ is the length of the adjacent side divided by the length of the hypotenuse



We can use the vectors to represent the text location in a V-dimensional vector space and compute the angles between them

Clustering

The idea of "clusters"

- Essentially: groups of items such that inside a cluster they are very similar to each other, but very different from those outside the cluster
- "unsupervised classification": cluster is not to relate features to classes or latent traits, but rather to estimate membership of distinct groups
- groups are given labels through post-estimation interpretation of their elements
- typically used when we do not and never will know the "true" class labels
- issues: how to weight distance is arbitrary
  - which dimensionality? (determined by which features are selected)
  - how to weight distance is arbitrary
  - different metrics for distance

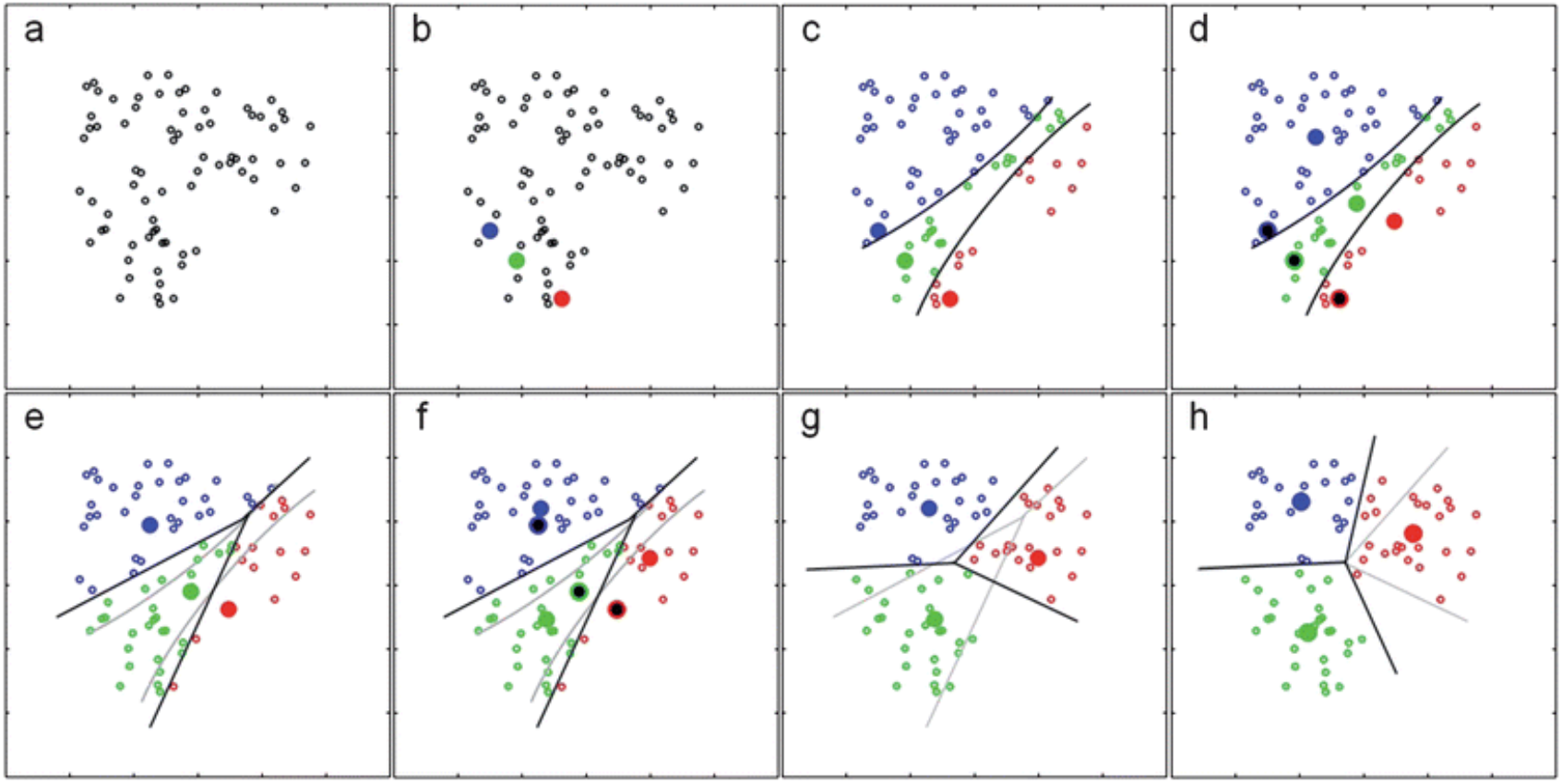
k-means clustering

- Essence: assign each item to one of k clusters, where the goal is to minimized within-cluster difference and maximize between-cluster differences
- Uses random starting positions and iterates until stable
- as with *kNN*, k-means clustering treats feature values as coordinates in a multi-dimensional space
- Advantages
  - simplicity
  - highly flexible
  - efficient
- Disadvantages
  - no fixed rules for determining k
  - uses an element of randomness for starting values

Algorithm details

1. Choose starting values
   - assign random positions to k starting values that will serve as the "cluster centres", known as "centroids" ; or,
   - assign each feature randomly to one of k classes
2. assign each item to the class of the centroid that is "closest"
   - Euclidean distance is most common
   - any others may also be used (Manhattan, Mikowski, Mahalanobis, etc.)
   - (assumes feature vectors have been normalized within item)
3. update: recompute the cluster centroids as the mean value of the points assigned to that cluster
4. repeat reassignment of points and updating centroids
5. repeat 2–4 until some stopping condition is satisfied
   - e.g. when no items are reclassified following update of centroids

k-means clustering illustrated

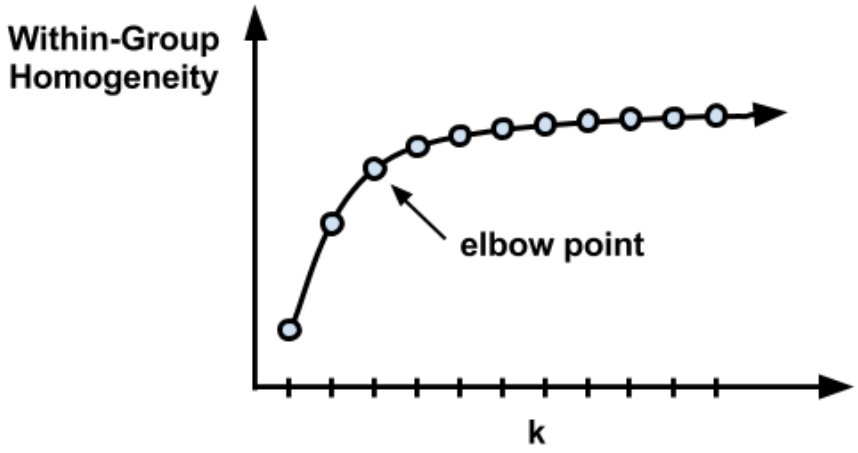


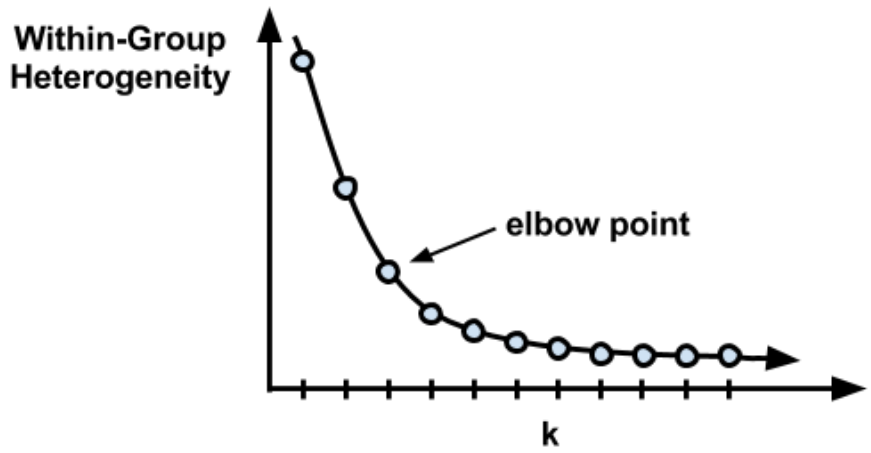
Choosing the appropriate number of clusters

- very often based on prior information about the number of categories sought
  - for example, you need to cluster people in a class into a fixed number of (like-minded) tutorial groups
- a (rough!) guideline: set $k = \sqrt{N/2}$ where N is the number of items to be classified
  - usually too big: setting k to large values will improve within-cluster similarity, but risks *overfitting*

Choosing the appropriate number of clusters

- "elbow plots": fit multiple clusters with different k values, and choose k beyond which are diminishing gains





Choosing the appropriate number of clusters

- "fit" statistics to measure homogeneity within clusters and heterogeneity in between
  - numerous examples exist
- "iterative heuristic fitting"* (IHF) (trying different values and looking at what seems most plausible)

* Warning: This is my (slightly facetious) term only!

Other clustering methods: hierarchical clustering

- *agglomerative*: works from the bottom up to create clusters
- like *k*-means, usually involves *projection*: reducing the features through either selection or projection to a lower-dimensional representation
  1. local projection: reducing features within document
  2. global projection: reducting features across all documents (Schütze and Silverstein, 1997)
  3. SVD methods, such PCA on a normalized feature matrix
  4. usually simple threshold-based truncation is used (keep all but 100 highest frequency or tf-idf terms)
- frequently/always involves weighting (normalizing term frequency, tf-idf)

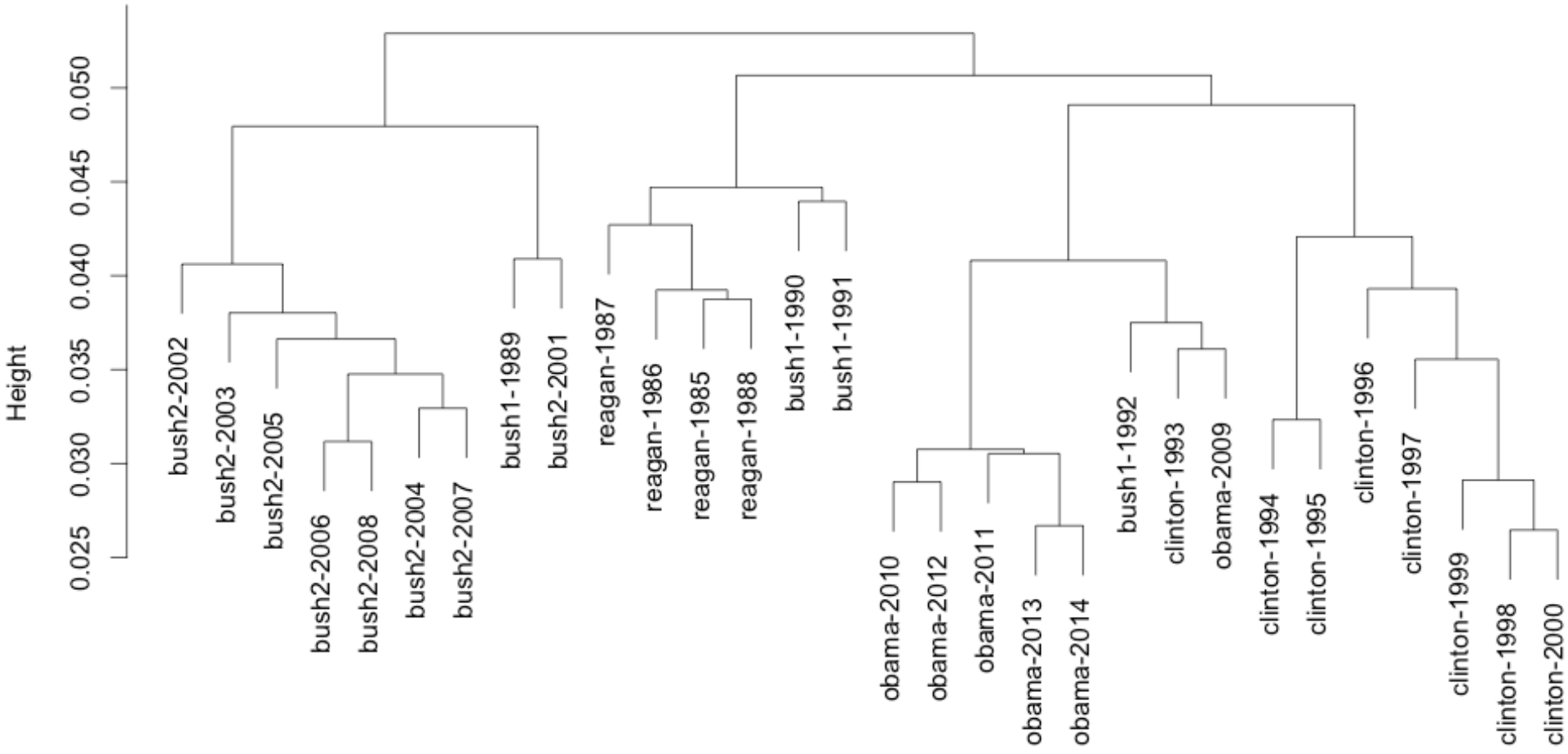
Hierarchical clustering algorithm

1. start by considering each item as its own cluster, for n clusters
2. calculate the $N(N-1)/2$ pairwise distances between each of the n clusters, store in a matrix D_0
3. find smallest (off-diagonal) distance in D_0, and merge the items corresponding to the i, j indexes in D_0 into a new "cluster"
4. recalculate distance matrix D_1 with new cluster(s). options for determining the location of a cluster include:
   - centroids (mean)
   - most dissimilar objects
   - Ward's measure(s) based on minimizing variance
5. repeat 3–4 until a stopping condition is reached
   - e.g. all items have been merged into a single cluster
6. to plot the *dendrograms*, need decisions on ordering, since there are $2^{(N-1)}$ possible orderings

Dendrogram: Presidential State of the Union addresses

```
data(data_corpus_sotu, package = "quanteda.corpora")
presDfm <- dfm(corpus_subset(data_corpus_sotu, Date > "1960-01-01"),
               stem = TRUE,
               remove = stopwords("english"))
presDfm <- dfm_trim(presDfm, min_termfreq = 5, min_docfreq = 3)
# hierarchical clustering - get distances on normalized dfm
presDistMat <- textstat_dist(dfm_weight(presDfm, scheme = "prop")) %>%
  as.dist()
# hiarchical clustering the distance object
presCluster <- hclust(presDistMat)
# label with document names
presCluster$labels <- docnames(presDfm)
# plot as a dendrogram
plot(presCluster)
```

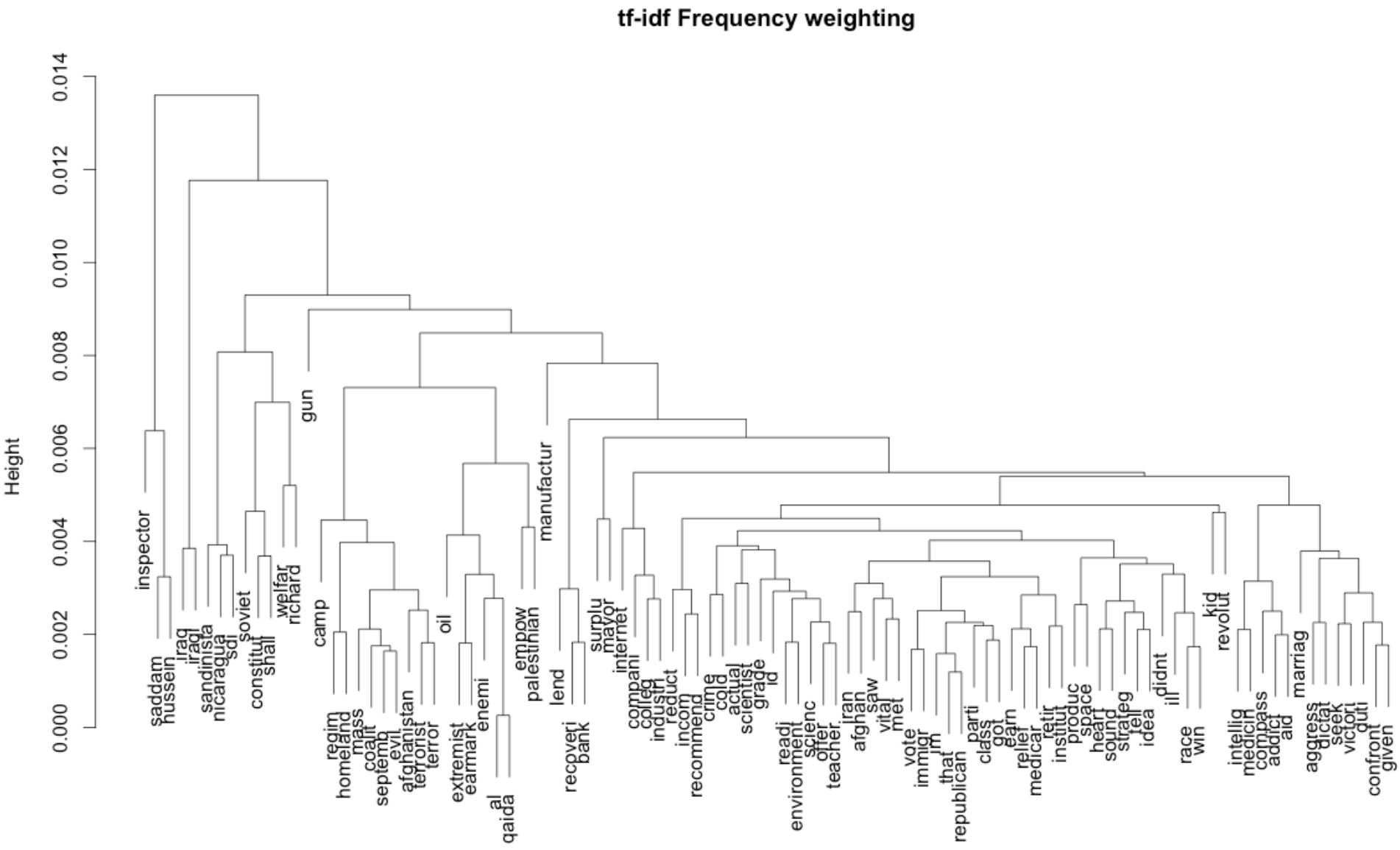
Dendrogram: Presidential State of the Union addresses



Dendrogram: Presidential State of the Union addresses

```
# word dendrogram with tf-idf weighting
wordDfm <- presDfm %>%
  dfm_remove("\\p{P}", valuetype = "regex") %>%
  dfm_trim(min_termfreq = 50, termfreq_type = "rank")
wordDistMat <- textstat_dist(wordDfm, margin = "feature") %>%
  as.dist()
wordCluster <- hclust(wordDistMat)
plot(wordCluster, xlab="", main="Top 50 features")
```

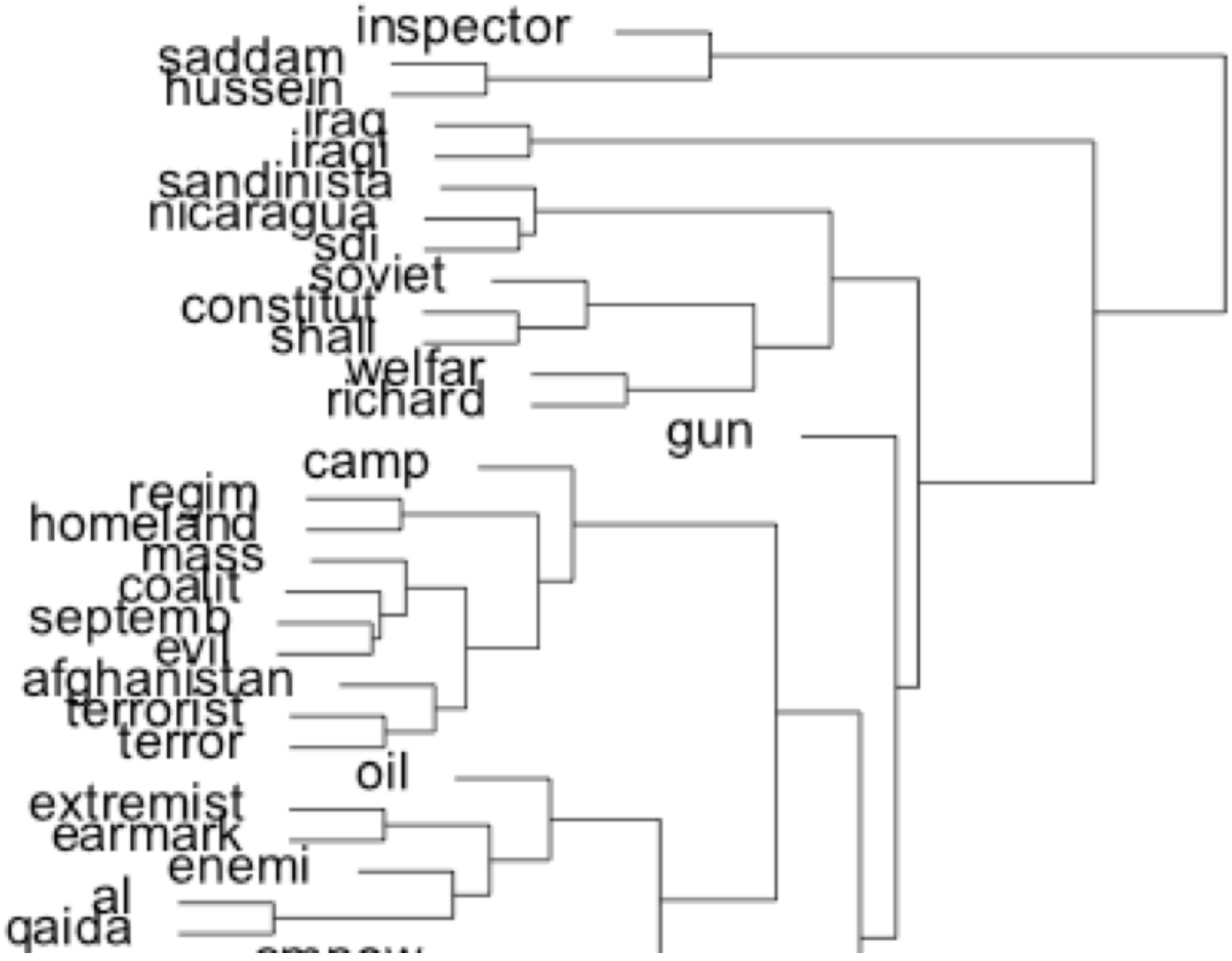
Dendrogram: Presidential State of the Union addresses



Pros and cons of hierarchical clustering

- advantages
  - deterministic, unlike k-means
  - no need to decide on k in advance (although can specify as a stopping condition)
  - allows hierarchical relations to be examined (usually through *dendrograms*)
- disadvantages
  - more complex to compute: quadratic in complexity: $O(n^2)$ – whereas k-means has complexity that is $O(n)$
  - the decision about where to create branches and in what order can be somewhat arbitrary, determined by method of declaring the "distance" to already formed clusters
  - for words, tends to identify collocations as base-level clusters (e.g. "saddam" and "hussein")

Dendrogram: Presidential State of the Union addresses



Resampling

- Today we discuss two resampling methods: cross-validation and the bootstrap.
- These methods refit a model of interest to samples formed from the training set, in order to obtain additional information about the fitted model.
- E.g., they provide estimates of test-set prediction error, and the standard deviation and bias of our parameter estimates.

Training Error versus Test error

- The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training.
- Training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

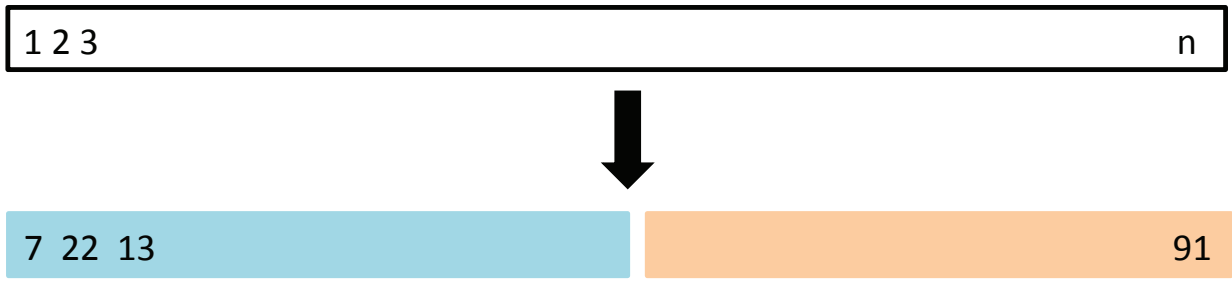
Error rate estimates

- Ideally you would have a large designated test set.
- Some methods make a mathematical adjustment to the training error rate in order to estimate the test error rate. These include the Cp statistic, AIC and BIC.
- Alternatively you can estimate the test error by holding out a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations. That's our focus here.

Validation-set approach

- We randomly divide the available set of samples into two parts: a training set and a validation (or hold-out) set.
- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.
- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response and misclassification rate for qualitative response models.

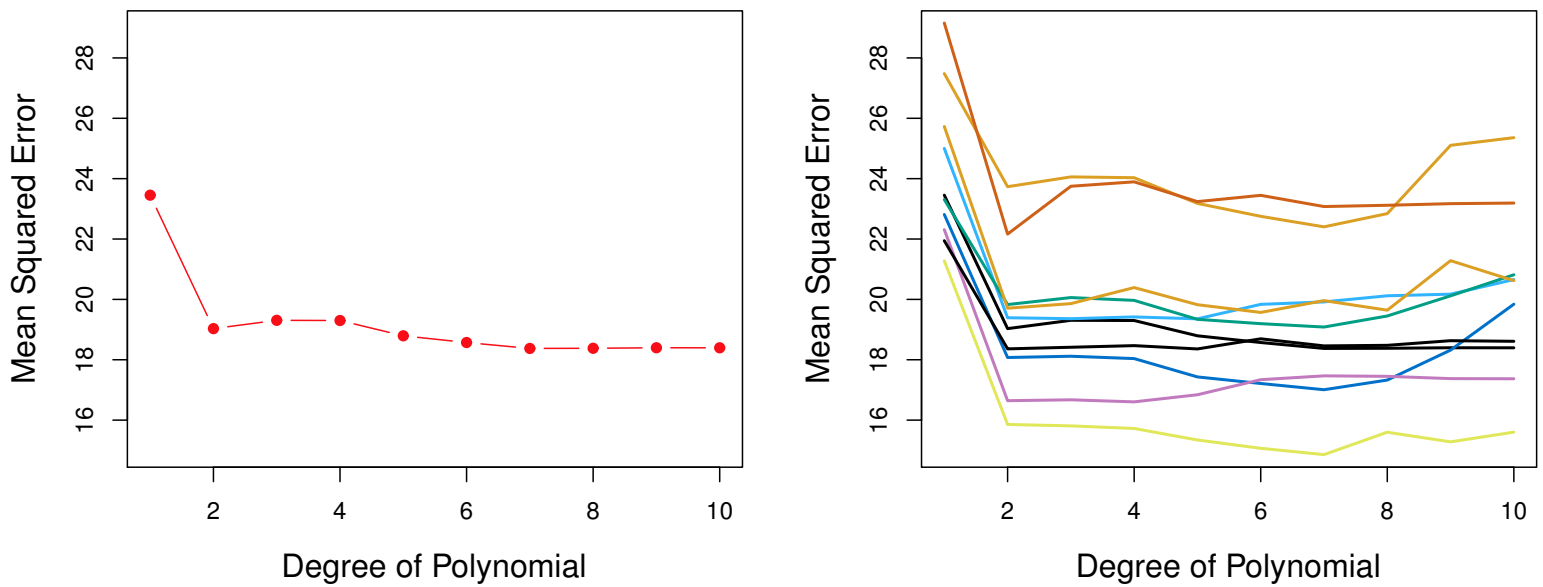
The Validation process



A random splitting into two halves: left part is training set, right part is validation set.

Example

- Want to compare linear vs higher-order polynomial terms in a linear regression with our Auto dataset.
- We randomly split the 392 observations into two sets, a training set containing 196 of the data points, and a validation set containing the remaining 196 observations.



Left panel shows single split; right panel shows multiple splits.

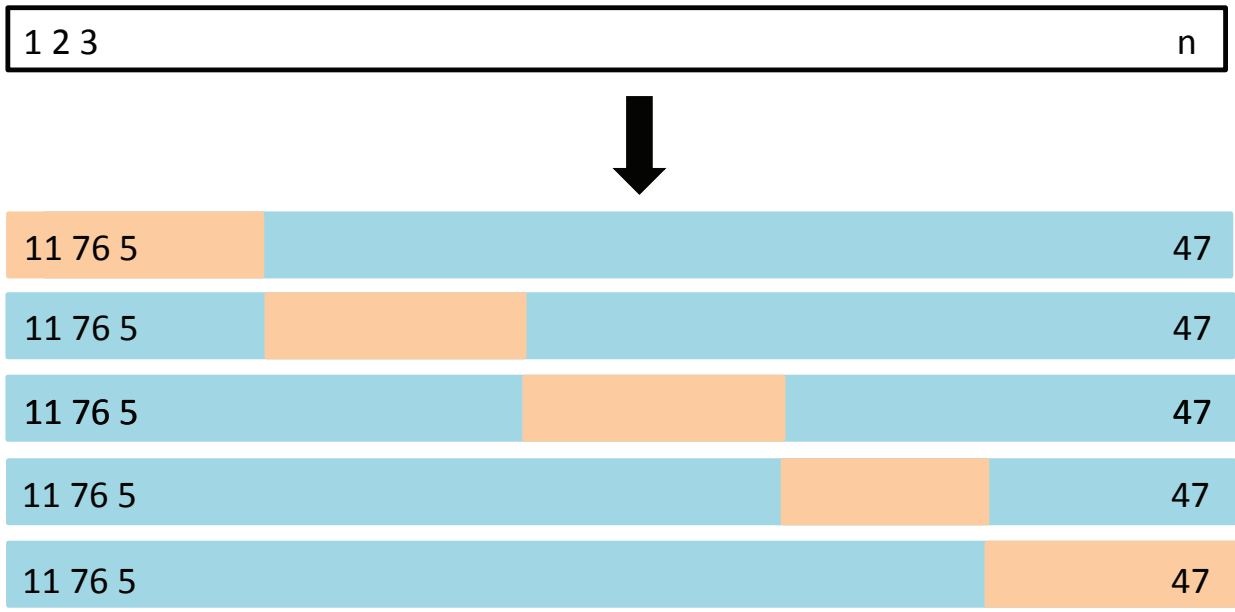
Issues with validation set approach

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset of the observations – those that are included in the training set rather than in the validation set – are used to fit the model.
- This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set. Why?

K-fold Cross-validation

- Very popular approach for estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Idea is to randomly divide the data into K equal-sized parts. We leave out part k, fit the model to the other $K-1$ parts (combined), and then obtain predictions for the left-out kth part.
- This is done in turn for each part $k = 1, 2, \ldots, K$, and then the results are combined.

5-fold CV



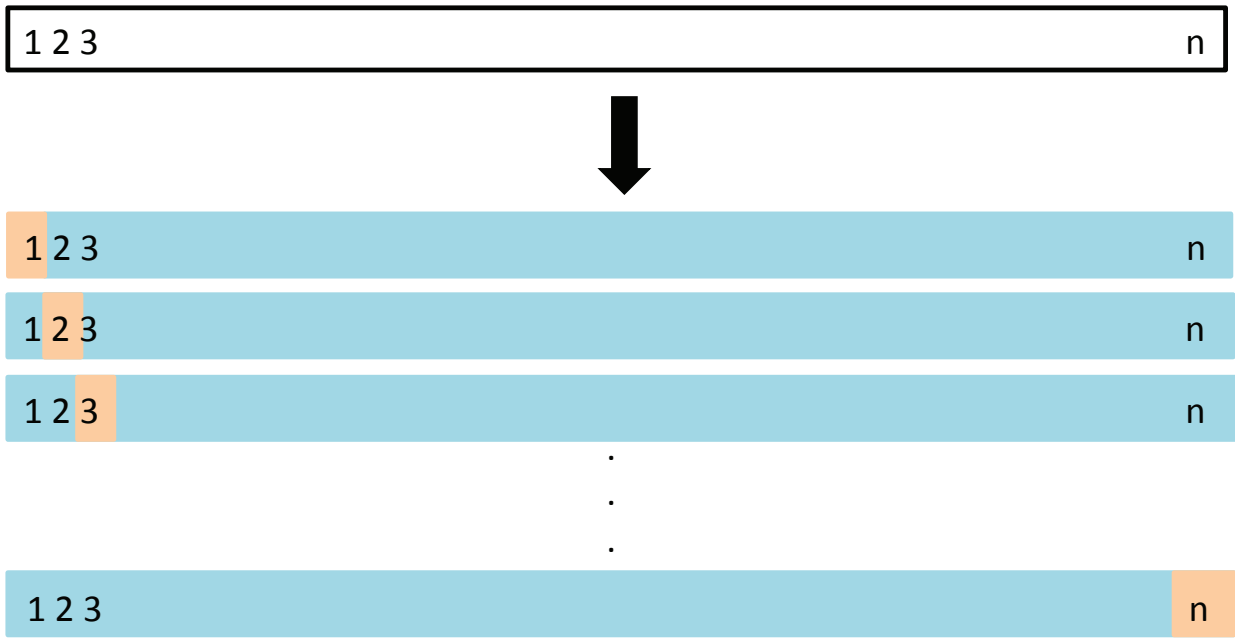
Mechanism

- Let the K parts be $C_1, C_2, \ldots, CK$, where CK denotes the indices of the observations in part k. There are n_k observations in part k: if N is a multiple of K, then $n_k = n/K$.
- Compute

$$CV_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{MSE}_k$$

  where $\mathrm{MSE}_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$, and $\hat{y}_i$) is the fit for observation i, obtained from the data with part k removed.
- Setting $K = n$ yields n-fold or leave-one out cross-validation (LOOCV).

LOOCV



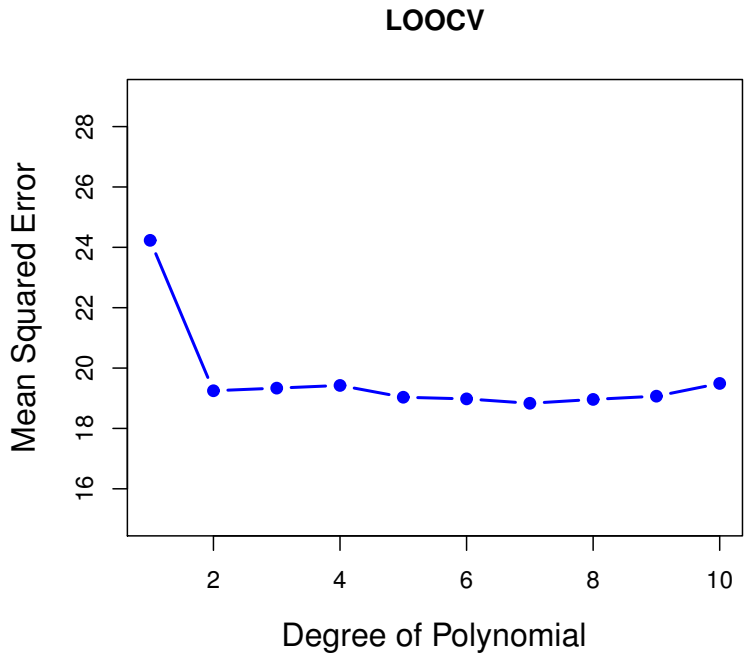
Special case of linear regression

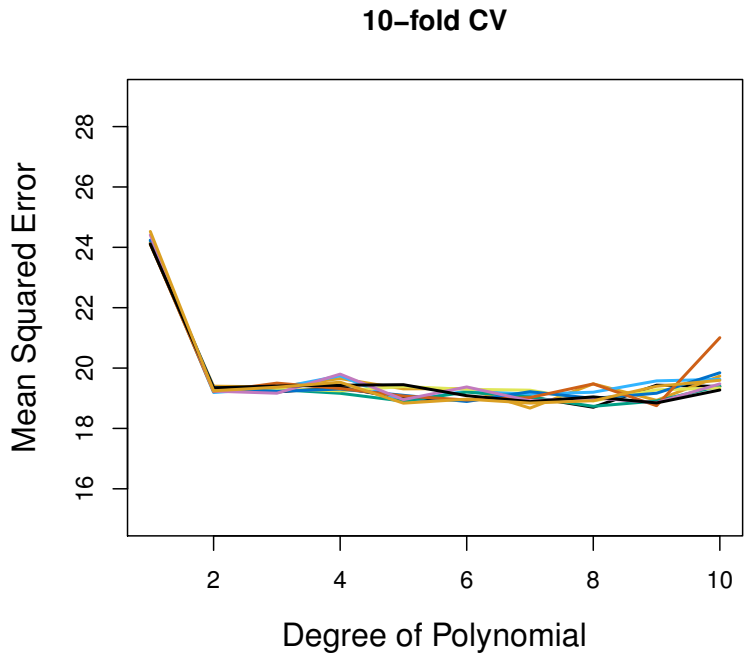
- With least-squares linear or polynomial regression, there is a shortcut making the cost of LOOCV the same as that of a single model fit.
- The following formula holds:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2$$

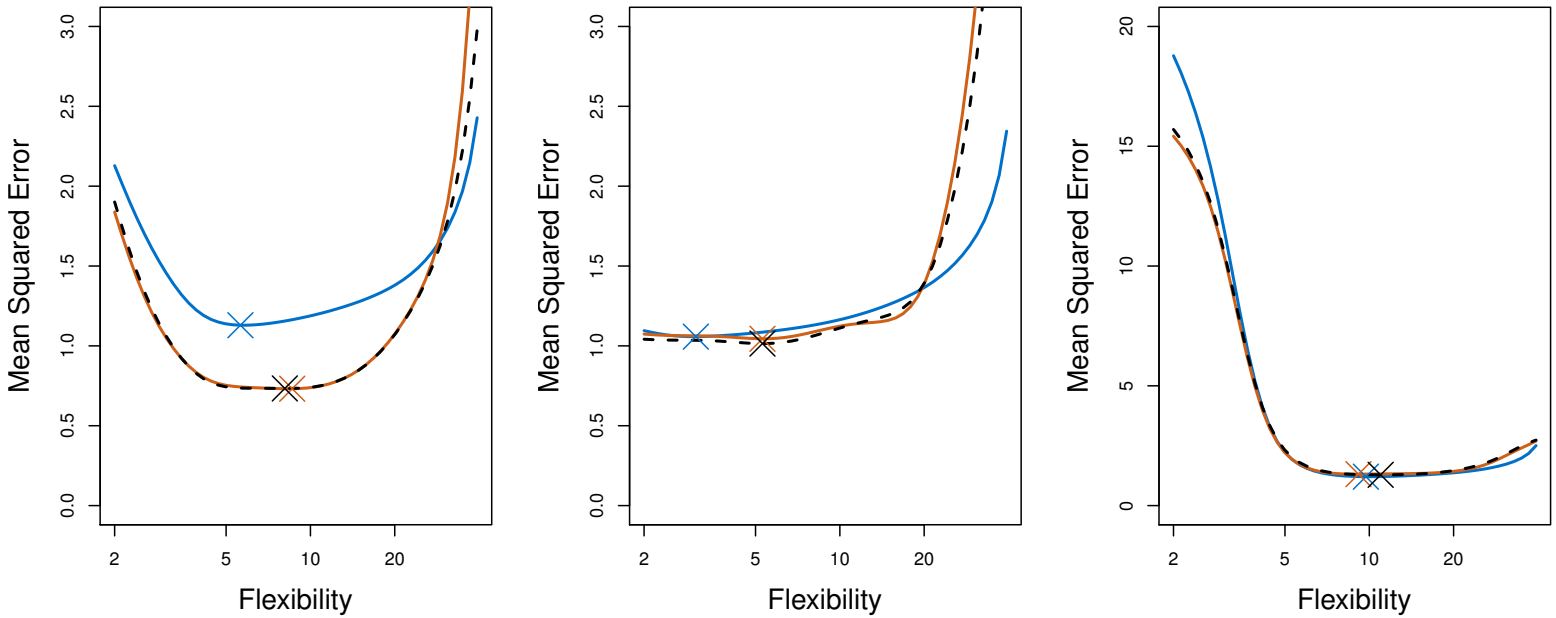
  where $\hat{y}_i$ is the ith fitted value from the original least squares fit, and h_i is the leverage (diagonal of the "hat" matrix). This is similar to the ordinary MSE, except the ith residual is divided by $(1 - h_i)$.
- The estimates from each fold are highly correlated and hence their average can have high variance.
- A better choice is $K = 5$ or 10.

Auto data example





True and estimated test MSE for the simulated data



Additional issues with CV

- Since each training set is only $(K-1)/K$ as big as the original training set, the estimates of prediction error will typically be biased upward.
- This bias is minimized when $K = n$ (LOOCV), but this estimate has high variance, as noted earlier.
- $K = 5$ or 10 provides a good balance for this bias-variance tradeoff.

CV for classification

- We divide the data into K roughly equal-sized parts $C_1, C_2, \ldots, C_K$. C_k denotes the indices of the observations in part k. There are n_k observations in part k: if n is a multiple of K, then $n_k = n/K$.
- Compute

$$CV_K = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{Err_k}$$

where $\mathrm{Err_k} = \sum_{i \in C_k} I(y_i \neq \hat{y}_i)/n_k$.

CV application

- Consider a simple classifier applied to some two-class data:
  1. Starting with 5000 predictors and 50 samples, find the 100 predictors having the largest correlation with the class labels.
  2. We then apply a classifier such as logistic regression, using only these 100 predictors.
- How do we estimate the test set performance of this classifier?
- Can we apply cross-validation in step 2, ignoring step 1?

CV application

- This would ignore the fact that in Step 1, the procedure has already seen the labels of the training data, and made use of them. This is a form of training and must be included in the validation process.
- It is easy to simulate realistic data with the class labels independent of the outcome, so that true test error $=50\%$, but the CV error estimate that ignores Step 1 is zero. (You can try doing this in class later today.

CV application

- Incorrect: Apply cross-validation in step 2.
- Correct: Apply cross-validation to steps 1 and 2.