Day 3: Introduction to Machine Learning

Kenneth Benoit

TCD HT 2017

20 February 2017

Outline

- Model evaluation in machine learning
  - Fitting v. overfitting
  - Precision, recall, and accuracy
- Supervised v. unsupervised learning
- Naive Bayes
- *k*-Nearest Neighbour
- Scaling distance
- Clustering
  - *k*-means clustering
  - Hierarchical clustering
- Cross-validation
  - Validation-set approach
  - K-fold Cross-validation

Model evaluation in machine learning

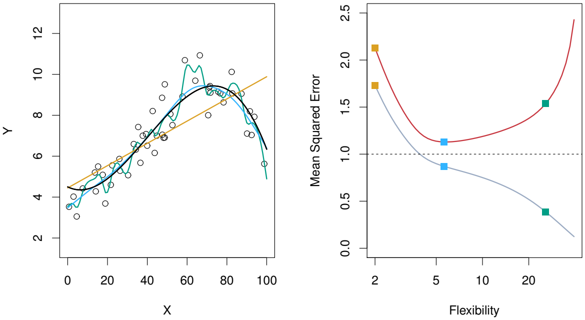
Assessing Model Accuracy

- Suppose we fit a model $\hat{f}(x)$ to some training data $Tr = \{x_i, y_i\}_1^N$, and we wish to see how well it performs.
- We could compute the average squared prediction error over Tr:

  $$MSE_{Tr} = Ave_{i \in Tr}[y_i - \hat{f}(x_i)]^2$$

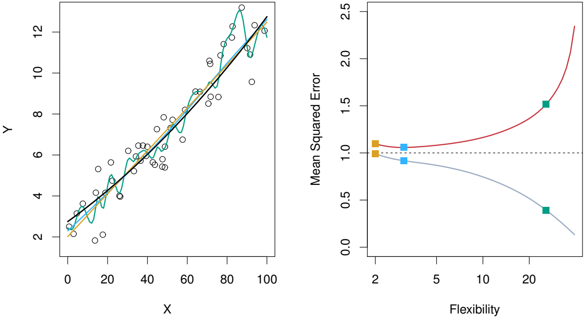
  This may be biased toward more overfit models.
- Instead we should, if possible, compute it using fresh test data $Te = \{x_i, y_i\}_1^M$:

  $$MSE_{Te} = Ave_{i \in Te}[y_i - \hat{f}(x_i)]^2$$

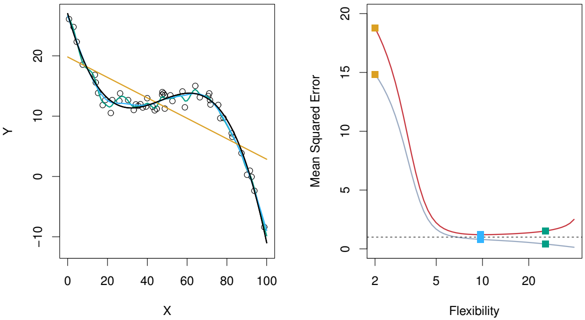


Data simulated from f, shown in black. Three estimates of f are shown: the linear regression line (orange curve), and two smoothing splines.

- Black curve is truth.
- Red curve on right is MSE_{Te}, grey curve is MSE_{Tr}.
- Orange, blue and green curves/squares correspond to fits of different flexibility.



- The setup as before, using a different true f that is much closer to linear. In this setting, linear regression provides a very good fit to the data.
- Here the truth is smoother, so the smoother fit and linear model do really well.



- Setup as above, using a different f that is far from linear.
- In this setting, linear regression provides a very poor fit to the data.
- Here the truth is wiggly and the noise is low, so the more flexible fits do the best.

Generalization and overfitting

- Generalization: A classifier or a regression algorithm learns to correctly predict output from given inputs not only in previously seen samples but also in previously unseen samples
- Overfitting: A classifier or a regression algorithm learns to correctly predict output from given inputs in previously seen samples but fails to do so in previously unseen samples. This causes poor prediction/generalization

How model fit is evaluated

- For discretely-valued outcomes (class prediction): Goal is to maximize the frontier of precise identification of true condition with accurate recall, defined in terms of false positives and false negatives
  - will define formally later
- For continuously-valued outcomes: minimize Root Mean Squared Error (RMSE)

Precision and recall

- Illustration framework

		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Precision and recall and related statistics

- Precision: $\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$
- Recall: $\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$
- Accuracy: $\frac{\text{Correctly classified}}{\text{Total number of cases}}$
- $F1 = 2\,\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
  (the harmonic mean of precision and recall)

Example: Computing precision/recall

Assume:

- We have a sample in which 80 outcomes are really positive (as opposed to negative, as in sentiment)
- Our method declares that 60 are positive
- Of the 60 declared positive, 45 are actually positive

Solution:

$$\begin{aligned} \text{Precision} &= (45/(45+15)) = 45/60 = 0.75 \\ \text{Recall} &= (45/(45+35)) = 45/80 = 0.56 \end{aligned}$$

Accuracy?

		True condition		
		Positive	Negative	
Prediction	Positive	🟩 45	🟥	60
	Negative	🟥	🟩	
		80		

add in the cells we can compute

		True condition		
		Positive	Negative	
Prediction	Positive	45	*15*	60
	Negative	*35*		
		80		

How do we get ”true” condition?

- In some domains: through more expensive or extensive tests
- In social sciences: typically by expert annotation or coding
- A scheme should be tested and reported for its reliability

Supervised v. unsupervised learning

From fitting predictive models to "machine learning"

- classical statistical analysis: estimate marginal effects
- predictive models: forecast the (unknown) value of a (new or future) observation
- machine learning: make predictions on data using a more broadly defined combination of statistical models and computational algorithms

Supervised v. unsupervised learning

- *Supervised methods* require a training set that exmplify constrasting classes, identified by the researcher
  - regression models belong to this category
- *Unsupervised methods* identify patterns without requiring an explicit training step
  - often involves calibrating some critical input parameter, such as the number of categories into which items will be clustered
  - more post-hoc interpretation is required

Supervised v. unsupervised methods: examples

- Supervised: Naive Bayes, k-Nearest Neighbor, Support Vector Machines (SVM)
- Unsupervised: correspondence analysis, IRT models, factor analytic approaches

Naive Bayes

Naive Bayes classification

- The following examples refer to “words” and “documents” but can be thought of as generic “features” and “cases”
- We will being with a discrete case, and then cover continuous feature values
- Objective is typically MAP: identification of the *maximum a posteriori* class probability

Multinomial Bayes model of Class given a Word

Consider J word types distributed across I documents, each assigned one of K classes.

At the word level, Bayes Theorem tells us that:

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

For two classes, this can be expressed as

$$= \frac{P(w_j|c_k)P(c_k)}{P(w_j|c_k)P(c_k) + P(w_j|c_{\neg k})P(c_{\neg k})} \tag{1}$$

Multinomial Bayes model of Class given a Word

Class-conditional word likelihoods

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j|c_k)P(c_k) + P(w_j|c_{\neg k})P(c_{\neg k})}$$

- The word likelihood within class
- The maximum likelihood estimate is simply the proportion of times that word j occurs in class k, but it is more common to use Laplace smoothing by adding 1 to each oberved count within class

Multinomial Bayes model of Class given a Word

Word probabilities

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j)}$$

- This represents the word probability from the training corpus
- Usually uninteresting, since it is constant for the training data, but needed to compute posteriors on a probability scale

Multinomial Bayes model of Class given a Word

Class prior probabilities

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j|c_k)P(c_k) + P(w_j|c_{\neg k})P(c_{\neg k})}$$

- This represents the class prior probability
- Machine learning typically takes this as the document frequency in the training set
- This approach is flawed for scaling, however, since we are scaling the latent class-ness of an unknown document, not predicting class – uniform priors are more appropriate

Multinomial Bayes model of Class given a Word

Class posterior probabilities

$$P(c_k|w_j) = \frac{P(w_j|c_k)P(c_k)}{P(w_j|c_k)P(c_k) + P(w_j|c_{\neg k})P(c_{\neg k})}$$

- This represents the posterior probability of membership in class k for word j

Moving to the document level

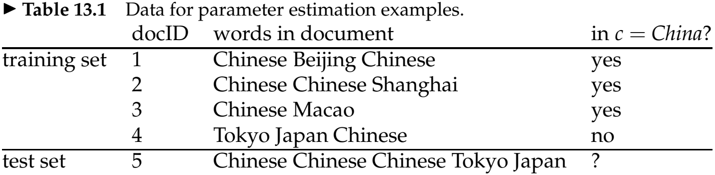
- The “Naive” Bayes model of a joint document-level class posterior assumes conditional independence, to multiply the word likelihoods from a “test” document, to produce:

$$P(c|d) = P(c) \prod_{j} \frac{P(w_j|c)}{P(w_j)}$$

- This is why we call it “naive”: because it (wrongly) assumes:
  - *conditional independence* of word counts
  - *positional independence* of word counts

Naive Bayes Classification Example

(From Manning, Raghavan and Schütze, *Introduction to Information Retrieval*)



▶ **Table 13.1** Data for parameter estimation examples.

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Naive Bayes Classification Example

Example 13.1: For the example in Table 13.1, the multinomial parameters we need to classify the test document are the priors $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$ and the following conditional probabilities:

$$\begin{aligned}
\hat{P}(\textsf{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\textsf{Tokyo}|c) = \hat{P}(\textsf{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\textsf{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\textsf{Tokyo}|\overline{c}) = \hat{P}(\textsf{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B in Equation (13.7) is 6 as the vocabulary consists of six terms.

We then get:

$$\begin{aligned}
\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003. \\
\hat{P}(\overline{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001.
\end{aligned}$$

Thus, the classifier assigns the test document to c = *China*. The reason for this classification decision is that the three occurrences of the positive indicator Chinese in d_5 outweigh the occurrences of the two negative indicators Japan and Tokyo.

Naive Bayes with continuous covariates

```
library(e1071)   # has a normal distribution Naive Bayes

# Congressional Voting Records of 1984 (abstentions treated as missing)
data(HouseVotes84, package = "mlbench")

## Error in find.package(package, lib.loc, verbose = verbose):  there is
no package called ’mlbench’

model <- naiveBayes(Class ~ ., data = HouseVotes84)

## Error in is.data.frame(data):  object ’HouseVotes84’ not found

# predict the first 10 Congresspeople
data.frame(Predicted = predict(model, HouseVotes84[1:10,-1]),
           Actual = HouseVotes84[1:10,1],
           postPr = predict(model, HouseVotes84[1:10, -1], type = "raw"))

## Error in predict(model, HouseVotes84[1:10, -1]):  object ’model’ not
found
```

Overall prediction performance

```
# now all of them: this is the resubstitution error
(mytable <- table(predict(model, HouseVotes84[,-1]), HouseVotes84$Class))

## Error in predict(model, HouseVotes84[, -1]):  object ’model’ not
found

prop.table(mytable, margin=1)

## Error in sweep(x, margin, margin.table(x, margin), "/", check.margin
= FALSE): object ’mytable’ not found
```

With Laplace smoothing

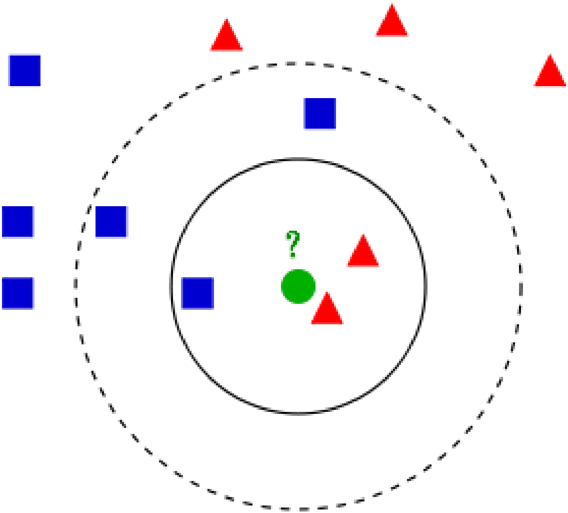
```
model <- naiveBayes(Class ~ ., data = HouseVotes84, laplace = 3)

## Error in is.data.frame(data):  object ’HouseVotes84’ not found

(mytable <- table(predict(model, HouseVotes84[,-1]), HouseVotes84$Class))

## Error in predict(model, HouseVotes84[, -1]):  object ’model’ not
found

prop.table(mytable, margin=1)

## Error in sweep(x, margin, margin.table(x, margin), "/", check.margin
= FALSE): object ’mytable’ not found
```

k-Nearest Neighbour

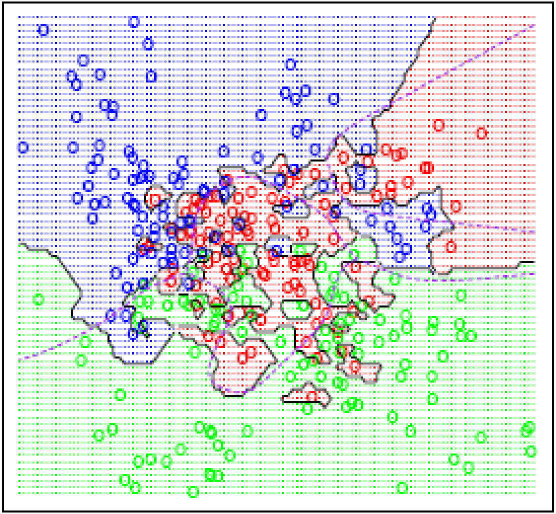
k-nearest neighbour

- 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

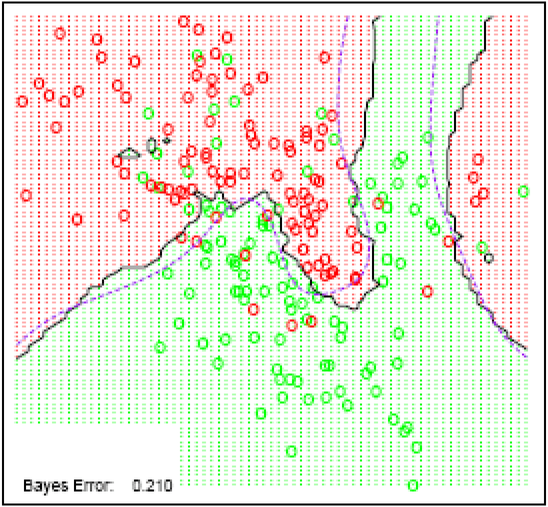
k-NN Example: Red or Blue?



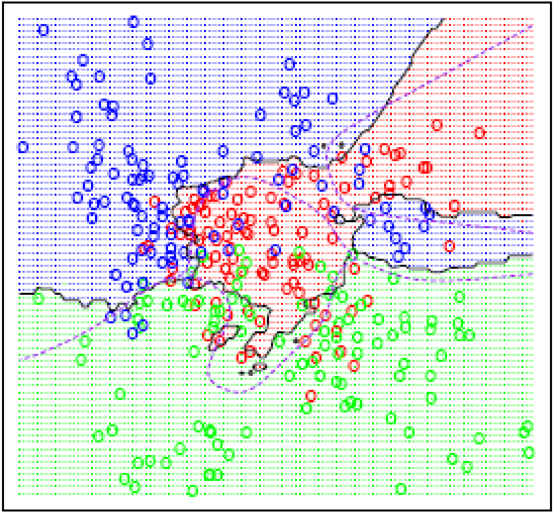
$k = 1$



$k = 7$



$k = 15$



Classifying amicus curiae briefs (Evans et al 2007)

```
## kNN classification
require(class)

## Loading required package:  class

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

## quanteda version 0.9.9.24
## Using 3 of 4 cores for parallel computing

data(amicusCorpus, package = "quantedaData")

## Warning in data(amicusCorpus, package = "quantedaData"):  data set
'amicusCorpus' not found

# create a matrix of documents and features
amicusDfm <- dfm(amicusCorpus, remove = stopwords("english"),
                 stem = TRUE, verbose = FALSE)

## Error in dfm(amicusCorpus, remove = stopwords("english"), stem =
TRUE, :  object 'amicusCorpus' not found

# threshold-based feature selection
amicusDfm <- trim(amicusDfm, min_count = 10, min_docfreq = 20)

## Error in trim(amicusDfm, min_count = 10, min_docfreq = 20):  object
```

Classifying amicus curiae briefs (Evans et al 2007)

```
# tf-idf weighting
amicusDfm <- dfm_weight(amicusDfm, "tfidf")

## Error in is(x, "dfm"):  object ’amicusDfm’ not found

# partition the training and test sets
train <- amicusDfm[!is.na(docvars(amicusCorpus, "trainclass")), ]

## Error in eval(expr, envir, enclos):  object ’amicusDfm’ not found

test  <- amicusDfm[!is.na(docvars(amicusCorpus, "testclass")), ]

## Error in eval(expr, envir, enclos):  object ’amicusDfm’ not found

trainclass <- docvars(amicusCorpus, "trainclass")[1:4]

## Error in docvars(amicusCorpus, "trainclass"):  object ’amicusCorpus’
not found
```

Classifying amicus curiae briefs (Evans et al 2007)

```
# classifier with k=1
classified <- knn(train, test, trainclass, k=1)

## Error in as.matrix(train):  object 'train' not found

table(classified, docvars(amicusCorpus, "testclass")[-c(1:4)])

## Error in table(classified, docvars(amicusCorpus,
"testclass")[-c(1:4)]):  object 'classified' not found
```

Classifying amicus curiae briefs (Evans et al 2007)

```
# classifier with k=2
classified <- knn(train, test, trainclass, k=2)

## Error in as.matrix(train):  object 'train' not found

table(classified, docvars(amicusCorpus, "testclass")[-c(1:4)])

## Error in table(classified, docvars(amicusCorpus,
"testclass")[-c(1:4)]):  object 'classified' not found
```

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

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

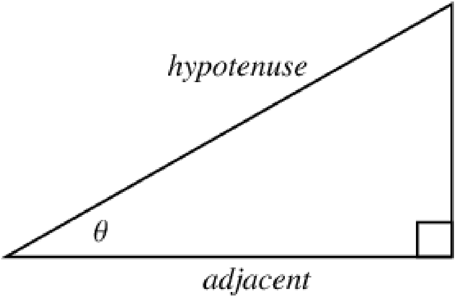
- In vector notation:

$$\| \mathbf{y}_A - \mathbf{y}_B \| \tag{3}$$

- 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(θ) 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

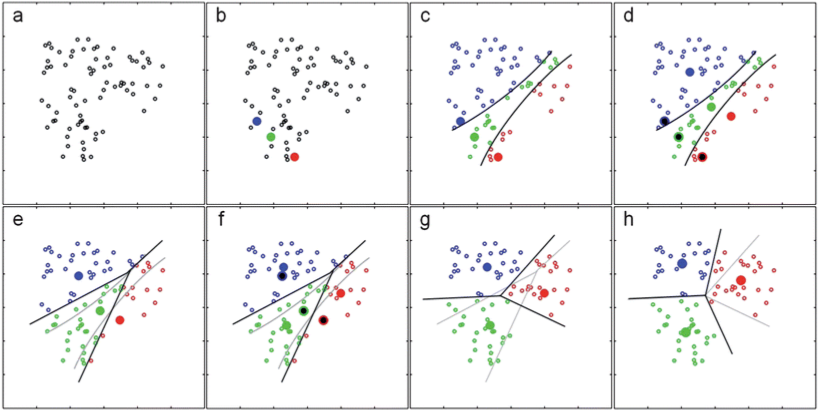
k-means clustering

- 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

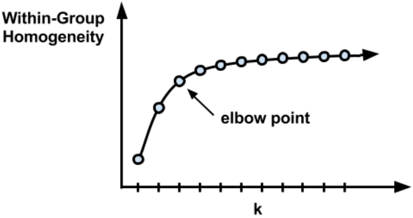


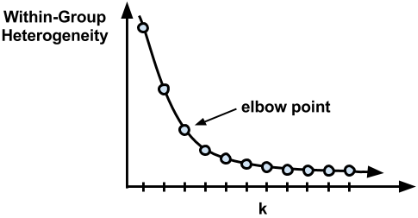
Choosing the appropriate number of clusters

- 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)

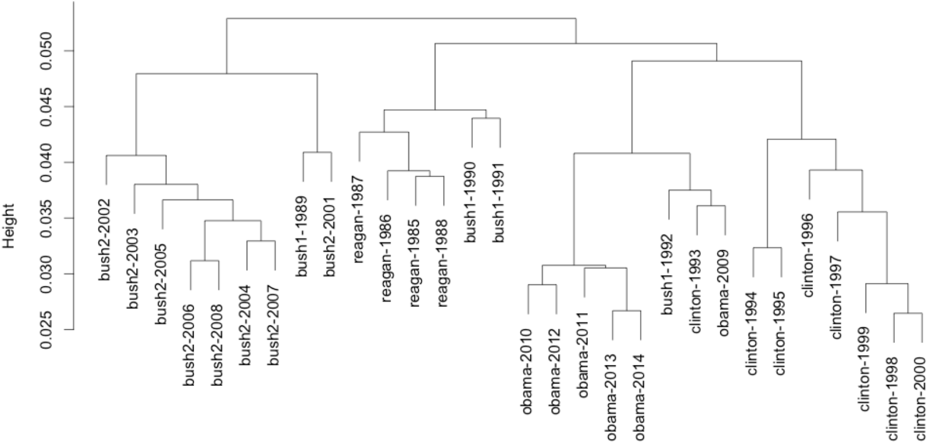
Hierarchical clustering algorithm

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="quantedaData")
presDfm <- dfm(corpus_subset(data_corpus_SOTU, lubridate::year(Date)>1960),
               verbose = FALSE, stem = TRUE,
               remove = stopwords("english"))
presDfm <- dfm_trim(presDfm, min_count=5, min_docfreq=3)
# hierarchical clustering - get distances on normalized dfm
presDistMat <- textstat_dist(dfm_weight(presDfm, "relFreq"))
# hiarchical clustering the distance object
presCluster <- hclust(presDistMat)
# label with document names
presCluster$labels <- docnames(presDfm)
# plot as a dendrogram
plot(presCluster)
```

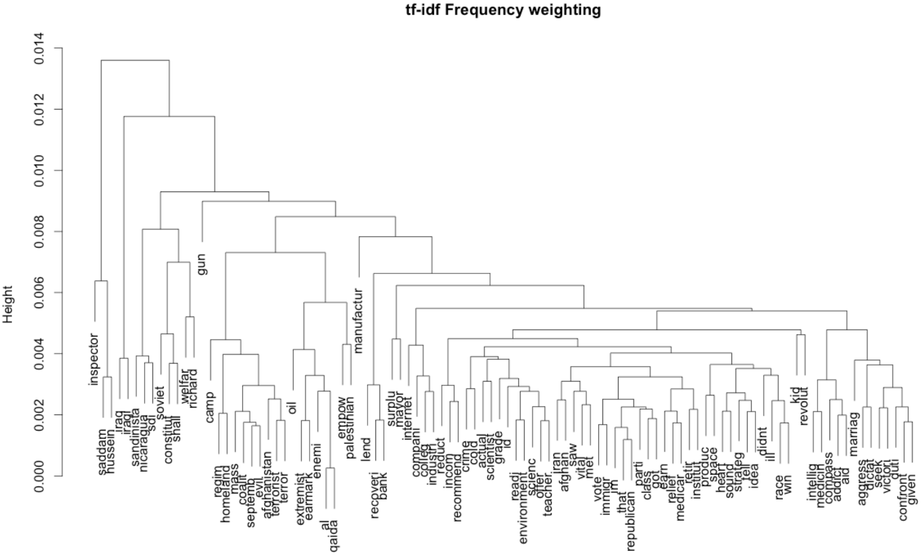
Dendrogram: Presidential State of the Union addresses



Dendrogram: Presidential State of the Union addresses

```r
# word dendrogram with tf-idf weighting
wordDfm <- dfm_sort(tfidf(presDfm))  # sort in decreasing order of total word fr
wordDfm <- t(wordDfm)[1:100,]    # because transposed
wordDistMat <- textstat_dist(wordDfm)
wordCluster <- hclust(wordDistMat)
suppressWarnings(plot(wordCluster, xlab="", main="tf-idf Frequency weighting"))
```

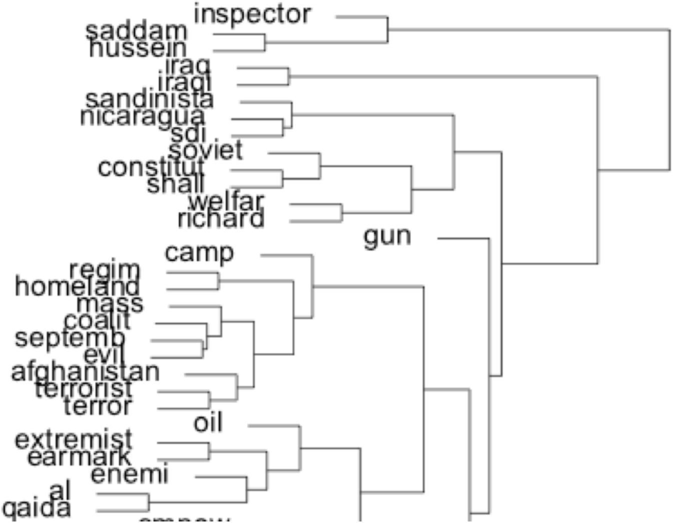
Dendrogram: Presidential State of the Union addresses



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.

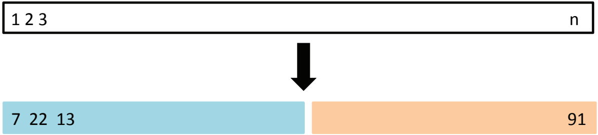
Error rate estimates

- 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.

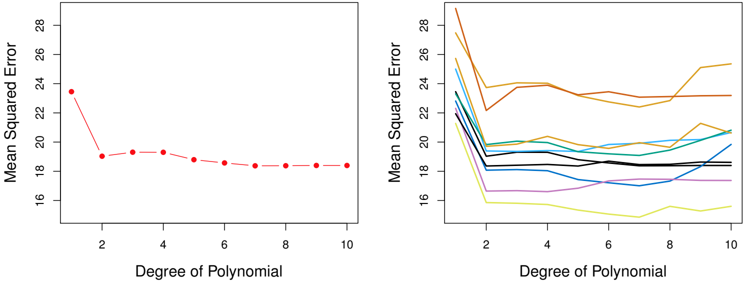
The Validation process



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.

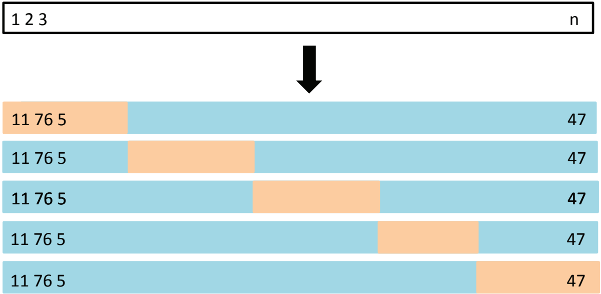
Issues with validation set approach

- 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



1 2 3 n

11 76 5 47

11 76 5 47

11 76 5 47

11 76 5 47

11 76 5 47

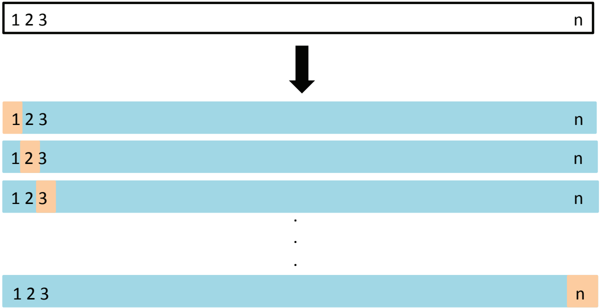
Mechanism

- 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



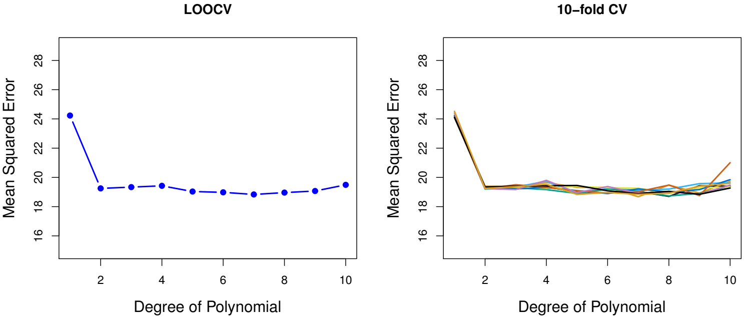
Special case of linear regression

- 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:

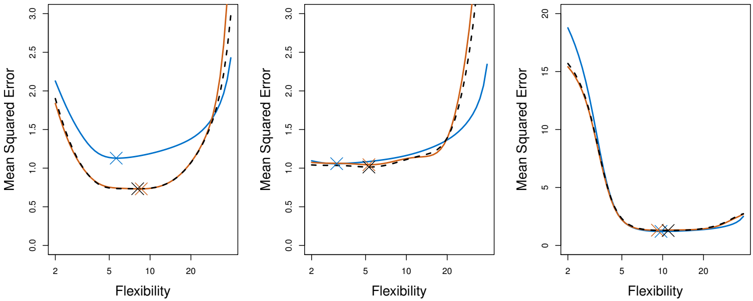
$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2$$

  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.