Course Summary: *An Introduction to Statistical Learning with Applications in R*

Yan Zeng

Version 1.0, last revised on 2016-05-14.

Abstract

Digest of course slides of [1], based on James et al. [2].

Contents

1 Introduction

An interesting reference is Silver [3].

Supervised learning problem: we have training data $(x_i, y_i)_{i=1}^N$ and we would like to 1) accurately predict unseen test cases, 2) understand which inputs affect the outcome and how, and 3) assess the quality of our predictions and inferences.

Unsupervised learning: 1) no outcome variable, just a set of predictors (features) measured on a set of samples, 2) objective is more fuzzy, 3) difficulty to know how well you are doing, and 4) can be useful as a pre-processing step for supervised learning.

Statistical Learning versus Machine Learning: Machine learning has a greater emphasis on *large scale* applications and *prediction accuracy*; statistical learning emphasizes *models* and their interpretability, and *precision* and *uncertainty*. Much overlap, much cross-fertilization.

2 Statistical Learning

We write our model as $Y = f(X) + \varepsilon$, where ε captures measurement errors and other discrepancies. Is there an ideal $f(X)$?

A good value is the **regression function**: $f(x) = E[Y|X = x]$, which is the optimal predictor of Y with regard to mean-squared prediction error. In order to estimate f, note we typically have few if any data points for $X = x$ exactly. So we cannot compute $E[Y|X = x]$. Therefore we relax the definition and let

$$\hat{f}(x) = Ave(Y|X \in \mathcal{N}(x)) = \frac{\sum_i y_i 1_{\{x_i \in \mathcal{N}(x)\}}}{\sum_i 1_{\{x_i \in \mathcal{N}(x)\}}}$$

where $\mathcal{N}(x)$ is some neighborhood of x. Then Pythagorean theorem says

$$E[(Y - \hat{f}(X))^2|X = x] = \underbrace{[f(x) - \hat{f}(x)]^2}_{reducible} + \underbrace{Var(\varepsilon|X = x)}_{irreducible}.$$

Nearest neighbor averaging can be pretty good for small dimension and large number of observations – i.e. $p \leq 4$ and large-ish N. Nearest neighbor methods can be lousy when p is large. Reason: the *curse of dimensionality*. Nearest neighbors tend to be far away in high dimensions, so we lose the spirit of estimating $E[Y|X = x]$ by local averaging.

Although it is almost never correct, a *linear model* $f_L(X)$ often serves as a good and interpretable approximation to the unknown true function $f(X)$. More flexible regression models include *thin-plate spline* $\hat{f}_S$.

Some trade-offs:
- Prediction accuracy versus interpretability.
- Good fit versus over-fit or under-fit: how do we know when the fit is just right?
- Parsimony versus black-box.

Increasing in flexibility and decreasing in interpretability: Subset Selection, Lasso $\longrightarrow$ Least Squares $\longrightarrow$ Generalized Additive Models, Trees $\longrightarrow$ Bagging, Boosting, Support Vector Machines.

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 $Te = \{x_i, y_i\}_1^M$:

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

In practice, one can usually compute the training MSE with relative ease, but estimating test MSE is considerably more difficult because usually no test data are available. There are a variety of approaches that can be used in practice to estimate the point where the minimum test MSE is achieved. One important method is *cross-validation*, which is a method for estimating test MSE using the training data.

Bias-variance trade-off: Suppose we have fit a model $\hat{f}(x)$ to some training data *Tr* and let (x_0, y_0) be a test observation drawn from the population. If the true model is $Y = f(X) + \varepsilon$ (with $f(x) = E[Y|X = x]$), then

$$E[y_0 - \hat{f}(x_0)]^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\varepsilon).$$

Here the notation $E[y_0 - \hat{f}(x_0)]^2$ defines the *expected test MSE*, and refers to the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0. The overall expected test MSE can be computed by averaging $E[y_0 - \hat{f}(x_0)]^2$ over all possible values of x_0 in the test set. Note that $\text{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$. Typically as the flexibility of $\hat{f}$ increases, its variance increases, and its bias decreases. So choosing the flexibility based on average test error amounts to a bias-variance trade-off.

Classification problems: Here the response variable Y is qualitative with discrete values in $\mathcal{C}$. Our goals are to build a classifier $C(X)$ that assigns a class label from $\mathcal{C}$ to a future unlabeled observation X. Is there an ideal $C(X)$? Suppose the K elements in $\mathcal{C}$ are numbered $1, 2, \cdots, K$. Let

$$p_k(x) = Pr(Y = k|X = x), \ k = 1, 2, \cdots, K.$$

Then the *Bayes optimal classifier at x* is

$$C(x) = j \text{ if } p_j(x) = \max\{p_1(x), p_2(x), \cdots, p_K(x)\}.$$

Nearest-neighbor averaging can be used as before, which also breaks down as dimension grows. However, the impact on $\hat{C}(x)$ is less than on $\hat{p}_k(x)$, $k = 1, \cdots, K$. Typically we measure the performance of $\hat{C}(x)$ using the misclassification error rate:

$$Err_{Te} = Ave_{i \in Te} I_{\{y_i \neq \hat{C}(x_i)\}}.$$

The Bayes classifier (using the true $p_k(x)$) has smallest error (in the population). Support-vector machines build structured models for $C(x)$; we will also build structured models for representing the $p_k(x)$, e.g. logistic regression, generalized additive models.

3 Linear Regression

For basics, we refer to Zeng [4], Section 7.

Some important questions of linear regression:

• *Is at least one of the predictors X_1, X_2, $\cdots$, X_p useful in predicting the response?*

• *Do all the predictors help to explain Y, or is only a subset of the predictors useful?*

• *How well does the model fit the data?*

• *Given a set of predictor values, what response value should we predict, and how accurate is our prediction?*

Is at least one predictor useful? Use the F-statistic

$$F = \frac{(TSS - RSS)/p}{RSS/(n - p - 1)} \sim F_{p,n-p-1}.$$

Deciding on the important variables.

• *All subsets* or *best subsets* regression: compute the least squares fit for all possible subsets and then choose between them based on some criterion that balances training error with model size.

• *Forward selection.* Begin with the *null model* – a model that contains an intercept but no predictors. Fit p simple linear regressions and add to the null model the variable that results in the lowest RSS. Add to that model the variable that results in the lowest RSS amongst all two-variable models. Continue until some stopping rule is satisfied, for example when all remaining variables have a p-value above some threshold.

• *Backward selection.* Start with al variables in the model. Remove the variable with the largest p-value – that is, the variable that is the least statistically significant. The new $(p-1)$-variable model is fit, and the variable with the largest p-value is removed. Continue until a stopping rule is reached. For instance, we may stop when all remaining variables have a significant p-value defined by some significance threshold.

• More systematic criteria: *Mallow's C_p*, *Akaike information criterion (AIC)*, *Bayesian information criterion (BIC)*, *adjusted R^2*, and *Cross-validation (CV).*

Interactions. The *hierarchy principle*: *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.* The rationale for this principle is that interactions are hard to interpret in a model without main effects – their meaning is changed.

4 Classification

Given a feature vector X and a qualitative response Y taking values in the set $\mathcal{C}$, the classification task is to build a function $C(X)$ that takes as input the feature vector X and predicts its value for Y; i.e. $C(X) \in \mathcal{C}$. Often we are more interested in estimating the *probabilities* that X belongs to each category in $\mathcal{C}$.

Logistic regression. Let's write $p(X) = Pr(Y = 1|X)$ for short. Logistic regression uses the form

$$p(X) = \frac{e^{\beta_0 + \sum_{i=1}^p \beta_i X_i}}{1 + e^{\beta_0 + \sum_{i=1}^p \beta_i X_i}}.$$

With the *log odds* or *logit* transformation of $p(X)$, the equations becomes

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \sum_{i=1}^p \beta_i X_i.$$

We use maximum likelihood to estimate the parameters:

$$\arg\max_{\beta_0,\beta} l(\beta_0, \beta) = \arg\max_{\beta_0,\beta} \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i)).$$

Logistic regression with more than two classes. One version has the symmetric form

$$Pr(Y = k|X) = \frac{e^{\beta_{0k} + \sum_{i=1}^p \beta_{ik} X_i}}{\sum_{l=1}^K e^{\beta_{0l} + \sum_{i=1}^p \beta_{il} X_i}}$$

Here there is a linear function for *each* class. Multiclass logistic regression is also referred to as *multinomial regression.*

Discriminant analysis.

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^K \pi_l f_l(x)}$$

where $f(x) = Pr(X = x|Y = k)$ is the density for X in class k and $\pi_k = Pr(Y = k)$ is the marginal or *prior* probability for class k. We classify a new point according to which density is highest. 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. However, estimating $f_k(X)$ tends to be more challenging, unless we assume some simple forms for these densities.

Why discriminant analysis?

• When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.

• If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.

• Linear discriminant analysis is popular when we have more than two response classes, because it also provides low-dimensional views of the data.

Summary:

• Logistic regression is very popular for classification, especially when $K = 2$.

• Linear discriminant analysis is useful when n is small, or the classes are well separated, and Gaussian assumptions are reasonable. Also when $K > 2$.

• Naive Bayes is useful when p is very large.

5 Resampling Methods

Cross-validation provides estimates of test-set prediction error, and *bootstrap* provides estimates of the standard deviation and bias of our parameter estimates.

Validation-set approach. Here 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 in the case of a qualitative (discrete) response.

This approach has several drawbacks:

• the validation estimate of the test error can be highly variable.

• 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.

• the validation set error may tend to *overestimate* the test error for the model fit on the entire data set.

K-fold cross-validation.

• *Widely used 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 $k = 1, 2, \cdots, K$, and then the results are combined:

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

where n_k is the number of observations in part k, n is the total number of observations, and

$$MSE_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k.$$

Here C_k denotes the indices of the observations in part k and $\hat{y}_i$ is the fit for observation i, obtained from the data with part k removed.

• 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 $K = n$, but this estimate has high variance.

• $K = 5$ or 10 provides a good compromise for this bias-variance tradeoff.

Cross-validation for classification problems.

•

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

where $Err_k = \sum_{i \in C_k} I_{\{y_i \neq \hat{y}_i\}} / n_k$.

• The estimated standard deviation of CV_K is

$$\widehat{SE}(CV_K) = \sqrt{\sum_{k=1}^{K} (Err_k - \overline{Err}_k)^2/(K-1)}$$

• This is a useful estimate, but strictly speaking, not quite valid: typical classification procedures consist of two steps – 1. select the relevant predictors; 2. apply a classifier such as logistic regression, using only the selected predictors. We should apply cross-validation to both step 1 and step 2.

The bootstrap. It is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method. For example, it can provide an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.

• 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 *with replacement*.

• Each of these "bootstrap data sets" is created by sampling *with replacement*, and is the *same size* as our original data set. As a result some observation may appear more than once in a given bootstrap data set and some not at all.

• Denoting the first bootstrap data set by Z^{*1}, we use Z^{*1} to produce a new bootstrap estimate for α, which we call $\hat{\alpha}^{*1}$. This procedure is repeated B times for some large value of B (say 100 or 1000), in order to produce B different bootstrap data sets and B corresponding α estimates.

• We estimate the standard error of these bootstrap estimates using the formula

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1} \sum_{r=1}^{B} (\hat{\alpha}^{*r} - \bar{\hat{\alpha}}^{*})^2}.$$

• In more complex data situations, figuring out the appropriate way to generate bootstrap samples can require some thought. For example, if the data is a time series, we can't simply sample the observations with replacement. We can instead create blocks of consecutive observations, and sample those with replacements. Then we paste together sampled blocks to obtain a bootstrap data set.

Other uses of the bootstrap.

• Primarily used to obtain standard errors of an estimate.

• Also provides approximate confidence intervals for a population parameter.

• The above confidence interval is called a *Bootstrap Percentile* confidence interval.

Can the bootstrap estimate prediction error?

• In cross-validation, there is no overlap among the K folds data sets. This is crucial for its success.

• To estimate prediction error using the bootstrap, we could think about using each bootstrap dataset as our training sample, and the original sample as our validation sample. But each bootstrap sample has significant overlap with the original data. This will cause the bootstrap to seriously underestimate the true prediction error.

Pre-validation.

The bootstrap versus permutation tests.

6 Linear Model Selection and Regularization

Despite its simplicity, the linear model has distinct advantages in terms of its *interpretability* and often shows good *predictive performance*. Hence we discuss in this lecture some ways in which the simple linear model can be improved, by replacing ordinary least squares (OLS) fitting with some alternative fitting procedures. There are three alternative classes of methods:

• *Subset Selection*. We identify a subset of the p predictors that we believe to be related to the response. We then fit a model using least squares on the reduced set of variables.

• *Shrinkage.* We fit a model involving all p predictors, but the estimated coefficients are shrunken towards zero relative to the least squares estimates. This shrinkage (also known as *regularization*) has the effect of reducing variance and can also perform variable selection.

• *Dimension Reduction.* We project the p predictors into a M-dimensional subspace, where $M < p$. This is achieved by computing M different *linear combinations*, or *projections*, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.

6.1 Estimating Test Error

We can *indirectly* estimate test error by making an adjustment to the training error to account for the bias due to overfitting. The indirect estimates include C_p, AIC, BIC, and adjusted R^2. We can also *directly* estimate the test error, using either a validation set approach or a cross-validation approach.

Indirect Method: C_p, AIC, BIC, and Adjusted R^2

These techniques adjust the training error for the model size, and can be used to select among a set of models with different numbers of variables.

Mallow's C_p:

$$C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2),$$

where d is he total number of parameters used and $\hat{\sigma}^2$ is an estimate of the variance of the error ϵ associated with each response measurement. The smaller is C_p, the better.

AIC: The AIC criterion is defined for a large class of models fit by maximum likelihood:

$$AIC = -2\log L + 2 \cdot d$$

where L is the maximized value of the likelihood function for the estimated model. The smaller is AIC, the better.

In the case of the linear model with Gaussian errors, maximum likelihood and least squares are the same thing, and C_p and AIC are equivalent.

BIC: compared with C_p, since $\log n > 2$ for any $n > 7$, the BIC statistic generally places a heavier penalty on models with many variables, and hence results in the selection of smaller models than C_p.

$$BIC = \frac{1}{n}(RSS + \log(n)d\hat{\sigma}^2).$$

Adjusted R^2:

$$\text{Adjusted } R^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$$

where TSS is the total sum of squares. Unlike C_p, AIC and BIC, a large value of adjusted R^2 indicates a model with a small test error. Maximizing the adjusted R^2 is equivalent to minimizing $\frac{RSS}{n-d-1}$.

Direct Method: Validation and Cross-Validation

These procedures have an advantage relative to AIC, BIC, C_p, and adjusted R^2, in that they provide a direct estimate of the test error, and *doesn't require an estimate of the error variance σ^2*. They can also be used in a wide range of model selection tasks, even in cases where it is hard to pinpoint the model degrees of freedom or hard to estimate the error variance σ^2.

In this setting, we can select a model using the *one-standard-error rule.* We first calculate the standard error of the estimated test MSE for each model size, and then select the smallest model for which the estimated test error is within one standard error of the lowest point on the curve.

6.2 Subset Selection

Best Subset Selection: beside linear regression, the same ideas also apply to other types of models, such as logistic regression.

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.

2. For $k = 1, 2, \cdots, p$:

(a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.

(b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.

3. Select a single best model from among $\mathcal{M}_0$, $\cdots$, $\mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

For computational reasons, best subset selection cannot be applied with very large p (with 2^p subsets to screen). Best subset selection may also suffer from *overfitting* and high variance of the coefficient estimates when the search space is large. For both of these reasons, *stepwise* methods are attractive alternatives.

Forward Stepwise Selection:

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.

2. For $k = 0, \cdots, p-1$:

(a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.

(b) Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having the smallest RSS or highest R^2.

3. Select a single best model from among $\mathcal{M}_0$, $\cdots$, $\mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Backward Stepwise Selection: different from forward stepwise selection, backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit).

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.

2. For $k = p, p-1, \cdots, 1$:

(a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.

(b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.

3. Select a single best model from among $\mathcal{M}_0, \cdots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

6.3 Shrinkage

Ridge Regression. Recall that the least squares fitting procedure estimates β_0, β_1, $\cdots$, β_p using the values that minimize

$$RSS = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2.$$

In contrast, the ridge regression coefficient estimates $\hat{\beta}^R$ are the values that minimize

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2,$$

where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately. The intuition is that the second term, $\lambda\sum_j \beta_j^2$, called a *shrinkage penalty*, is small when β_1, $\cdots$, β_p are close to zero, and so it has the effect of *shrinking* the estimates of β_j towards zero. The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates. Selecting a good value for λ is critical; cross-validation is used for this.

The standard least squares coefficient estimates are *scale equivariant*. In other words, regardless of how the jth predictor is scaled, $X_j\hat{\beta}_j$ will remain the same. In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function. Therefore, it is best to apply ridge regression after *standardizing the predictors*, using the formula

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij}-\bar{x}_j)^2}}.$$

Lasso. Ridge regression does have one obvious disadvantage: unlike subset selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model.

The *Lasso* is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients, $\hat{\beta}_\lambda^L$, minimize the quantity

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|.$$

As with ridge regression, the lasso shrinks the coefficient estimates towards zero. However, in the case of the lasso, the l_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large. Hence, much like best subset selection, the lasso performs *variable selection.* As in ridge regression, selecting a good value of λ for the lasso is critical; cross-validation is again the method of choice.

Why is it that the lasso, unlike ridge regression, results in coefficient estimates that are exactly equal to zero? One can show that the lasso and ridge regression coefficient estimates solve the problems

$$\min_{\beta}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \text{ subject to } \sum_{j=1}^{p}|\beta_j| \le s$$

and

$$\min_{\beta}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \text{ subject to } \sum_{j=1}^{p}\beta_j^2 \le s$$

respectively. As a general result of convex optimization, the minimal values occur on extreme points, and the extreme points of the set $\{(\beta_1,\cdots,\beta_p) : \sum_{j=1}^{p}|\beta_j| \le s\}$ are those points on axis.

Conclusion. Neither ridge regression nor the lasso will universally dominate the other. In general, one might expect the lasso to perform better when the response is a function of only a relatively small number of predictors. However, the number of predictors that is related to the response is never known *a priori* for real data sets. A technique such as cross-validation can be used in order to determine which approach is better on a particular data set.

To select for ridge regression and lasso the tuning parameter λ or equivalently, the value of the constraint s, we can use cross-validation. We choose a grid of λ values, and compute the cross-validation error rate for each value of λ. We then select the tuning parameter value for which the cross-validation error is smallest. Finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter.

6.4 Dimension Reduction

Let Z_1, Z_2, $\cdots$, Z_M represent $M < p$ linear combinations of our original p predictors:

$$Z_m = \sum_{j=1}^{p}\phi_{mj}X_j$$

for some constants $\phi_{m1}, \cdots, \phi_{mp}$. We can then fit the linear regression model using OLS:

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i,\ i = 1, \cdots, n.$$

If the constants $\phi_{m1}, \cdots, \phi_{mp}$ are chosen wisely, then such dimension reduction approaches can often outperform OLS regression.

Principal Components Regression (PCR) applies principal components analysis (PCA) to define the linear combinations of the predictors, for use in regression. The first principal component is that (normalized) linear combination of the variables with the largest variance. The second principal component has largest variance, subject to being uncorrelated with the first. And so on. Hence with many correlated original variables, we replace them with a small set of principal components that capture their joint variation.

Partial Least Squares. PCR identifies linear combinations, or *directions*, that best represent the predictors $X_1, \cdots, X_p$. These directions are identified in an *unsupervised* way, since the response Y is not used to help determine the principal component directions. Consequently, PCR suffers from a potentially serious drawback: there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.

Like PCR, Partial Least Squares (PLS) is a dimension reduction method, which first identifies a new set of features $Z_1, \cdots, Z_M$ that are linear combinations of the original features, and then fits a linear model via OLS using these M new features. But unlike PCR, PLS identifies these new features in a supervised way. Roughly speaking, the PLS approach attempts to find directions that help explain both the response and the predictors.

More precisely, after standardizing the p predictors, PLS computes the first direction Z_1 by setting each ϕ_{1j} in the equation

$$Z_1 = \sum_{j=1}^{p} \phi_{1j} X_j$$

to the coefficient from the simple linear regression of Y onto X_j. One can show that this coefficient is proportional to the correlation between Y and X_j in the multiple linear regression. Intuitively, in computing $Z_1 = \sum_{j=1}^{p} \phi_{1j} X_j$, PLS places the highest weight on the variables that are most strongly related to the response. Subsequent directions are found by taking residuals and then repeating the above prescription.

7 Moving Beyond Linearity

For nonlinear models, we are not really interested in the coefficients; we are more interested in the fitted function values at any value x_0.

Polynomial regression. We either fix the degree d at some reasonably low value, or use cross-validation to choose d. One caveat is that polynomials have notorious tail behavior – very bad for extrapolation. Example in R: $y \sim \texttt{poly}(x, \texttt{degree} = 3)$.

Step functions. This method creates a series of dummy variables representing each group; it's also a useful way of creating interactions that are easy to interpret.

Regression splines. Splines have the "maximum" amount of continuity. Fitting splines in R is easy: $\texttt{bs}(x, \cdots)$ for any degree splines, and $\texttt{ns}(x, ...)$ for natural cubic splines, in package `splines`. To place knots, one strategy is to decide K, the number of knots, and then place them at appropriate quantiles of the observed X.

Smoothing splines. Consider the criterion for fitting a smooth function $g(x)$ to some data:

$$\min_{g \in S} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt.$$

The first term is RSS, and tries to make $g(x)$ match the data at each x_i. The second term is a *roughness penalty* and controls how wiggly $g(x)$ is. It is modulated by the *tuning parameter* $\lambda \geq 0$: the smaller λ, the more wiggly the function, eventually interpolating y_i when $\lambda = 0$; as $\lambda \to \infty$, the function $g(x)$ becomes linear.

The solution is a natural cubic spline (a cubic spline that extrapolates linearly beyond the boundary knots), with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.

• Smoothing splines avoid the knot-selection issue, leaving a single λ to be chosen.

• The algorithmic details are too complex to describe here. In R, the function `smooth.spline()` will fit a smoothing spline.

• The vector of n fitted values can be written as $\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y}$, where $\mathbf{S}_\lambda$ is a $n \times n$ matrix (determined by the x_i and λ).

• The *effective degrees of freedom* are given by

$$df_\lambda = \sum_{i=1}^{n} \{\mathbf{S}_\lambda\}_{ii}.$$

• We can specify *df* rather than λ! In R: `smooth.spline(age, wage, df = 10)`.

• The leave-one-out (LOO) cross-validated error is given by

$$RSS_{cv}(\lambda) = \sum_{i=1}^{n} (y_i - \hat{g}_\lambda^{(-i)}(x_i))^2 = \sum_{i=1}^{n} \left[\frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}} \right]^2.$$

In R: `smooth.spline(age, wage)`.

Local regression. With a sliding weight function, we fit separate linear fits over the range of X by weighted least squares. See [2] for more details, and `loess()` function in R.

Generalized additive models (GAM). This method allows for flexible nonlinearities in several variables, but retains the additive structure of linear models:

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i.$$

• Can fit a GAM simply using, e.g. natural splines.

• Coefficients not that interesting; fitted functions are.

• Can mix terms – some linear, some nonlinear – and use `anova()` to compare models.

• Can use smoothing splines or local regression as well.

• GAMs are additive, although low-order interactions can be included in a natural way using, e.g. bivariate smoothers or interactions of the form `ns(age,df=5):ns(year,df=5)`.

• GAMs for classification

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$

8 Tree-Based Methods

Tree-based methods are simple and useful for interpretation. However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy. Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then combined to yield a single consensus prediction. Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

The basics of regression decision trees. We divide the predictor space – that is, the set of possible values for $X_1, X_2, \cdots, X_p$ into J distinct and non-overlapping regions, $R_1, R_2, \cdots, R_J$. 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.

In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes $R_1, \cdots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, we take a *top-down*, *greedy* approach that is known as recursive binary splitting. The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree. It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

• 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.

• Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.

• However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.

• Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

• We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Pruning a tree. The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. A smaller tree with fewer splits (that is, fewer regions $R_1, \cdots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias. One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold. This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split – that is, a split that leads to a large reduction in RSS later on. A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree. *Cost complexity pruning* – also known as *weakest link pruning* – is used to do this.

• We consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

• The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.

• We select an optimal value $\hat{\alpha}$ using cross-validation.

• We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: regression tree algorithm.

• 1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

• 2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

• 3. Use K-fold cross-validation to choose α. For each $k = 1, \cdots, K$:
3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
Average the results, and pick α to minimize the average error.
• 4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Classification trees. Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one. For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

• Just as in the regression setting, we use recursive binary splitting to grow a classification tree.

• In the classification setting, RSS cannot be used as a criterion for making the binary splits. A natural alternative to RSS is the *classification error rate*. This is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

• The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.

• For this reason the Gini index is referred to as a measure of node *purity* – a small value indicates that a node contains predominantly observations from a single class.

• An alternative to the Gini index is *cross-entropy*, given by

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

It turns out that the Gini index and the cross-entropy are very similar numerically.

Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book. However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Bagging. *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

• Recall that given a set of n independent observations $Z_1, \cdots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

• In other words, *averaging a set of observations reduces variance.* Of course, this is not practical because we generally do not have access to multiple training sets. Instead, we can bootstrap, by taking repeated samples from the (single) training data set.

• In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x).$$

This is called *bagging.*

• The above prescription applied to regression trees. For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the B predictions.

Out-of-Bag Error Estimation. Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations. We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average. This estimate is essentially the LOO cross-validation error for bagging, if B is large.

Random forests. Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees. As in bagging, we build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{pp}$ – that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Boosting. Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.

Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Notably, each tree is built on a bootstrap data set, independent of the other trees. Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

```
• 1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.
• 2. For $b = 1, 2, \cdots, B$, repeat:
    2.1 Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training data $(X, r)$.
    2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree: $\hat{f}(x) \longleftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.
    2.3 Update the residuals: $r_i \longleftarrow r_i - \lambda \hat{f}^b(x_i)$.
• 3. Output the boosted model: $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.
```

The idea behind this procedure. Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly. Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals. Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm. 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.

Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. Students can learn about the details in *Elements of Statistical Learning*, chapter 10. The R package `gbm` (gradient boosted models) handles a variety of regression and classification problems.

Tuning parameters for boosting.

• The *number of trees B*. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

• The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

• The *number of splits d* in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split and resulting in an additive model. More generally d is the *interaction depth*, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Variable importance measure. For bagged/RF regression trees, we 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. Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.

9 Support Vector Machines

Maximal margin classifier.

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes. Recall in an orthonormal system, the distance between a point $P = (x_1, x_2, \cdots, x_p)$ and a plane $\pi : \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0$ is

$$\frac{|\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p|}{\sqrt{\sum_{j=1}^p \beta_j^2}}.$$

So finding the maximal margin classifier can be formulated as a constrained optimization problem:

$$\text{maximize}_{\beta_0,\beta_1,\cdots,\beta_p} M \text{ subject to } \sum_{j=1}^p \beta_j^2 = 1 \text{ and } |\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}| \geq M \text{ for all } i = 1, \cdots, N.$$

This can be rephrased as a convex quadratic program, and solved efficiently. The function `svm()` in package e1071 solves this problem efficiently.

Support vector classifier. Sometimes the data are not separable by a linear boundary. This is often the case, unless $N < p$. Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier. The *support vector classifier* maximizes a *soft* margin:

$$\text{maximize}_{\beta_0,\beta_1,\cdots,\beta_p} M \text{ subject to } \sum_{j=1}^p \beta_j^2 = 1 \text{ and } |\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}| \geq M(1 - \varepsilon_i) \text{ with } \varepsilon_i \geq 0, \sum_{i=1}^n \varepsilon_i \leq C,$$

where C is a regularization parameter.

Sometime a linear boundary simply won't work, no matter what value of C. In this case, we can enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$, $\cdots$. Hence we go from a p-dimensional space to an $M > p$ dimensional space. This will fit a support-vector classifier in the enlarged space and results in non-linear decision boundaries in the original space.

Support vector machines. Polynomials (especially high-dimensional ones) get wild rather fast. There is a more elegant and controlled way to introduce nonlinearities in support-vector classifiers – through the use of kernels. It can be shown that

• The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^n \alpha_i \langle x, x_i \rangle,$$

where there are n parameters α_i, $i = 1, \cdots, n$, one per training observation.

• To estimate the parameters $\alpha_1, \cdots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_i' \rangle$ between all pairs of training observations.

It turns out that most of the $\hat{\alpha}_i$ can be zero:

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle$$

where $\mathcal{S}$ is the *support set* of indices i such that $\hat{\alpha}_i > 0$.

If we can compute inner-products between observations, we can fit a SV classifier. Some special kernel functions can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

computes the inner product needed for d dimensional polynomials – $\binom{p+d}{d}$ basis functions. The solution has the form

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i).$$

SVMs with more than two classes. The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

• One versus All (OVA). Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \cdots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}^k(x^*)$ is largest.

• One versus One (OVO). Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{kl}(x)$. Classify x^* to the class that wins the most pairwise competitions.

Which to choose? If K is not too large, use OVO.

Relationship to logistic regression. With $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$ can rephrase support-vector classifier optimization as

$$\min_{\beta_0, \beta_1, \cdots, \beta_p} \left\{ \sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}.$$

This has the form of *loss plus penalty.* The loss is known as the *hinge loss.* Very similar to “loss” in logistic regression (negative log-likelihood). Which to use: SVM or Logistic Regression?

• When classes are (nearly) separable, SVM does better than LR. So does LDA.

• When not, LR (with ridge penalty) and SVM very similar.

• If you wish to estimate probabilities, LR is the choice.

• For nonlinear boundaries, kernel SVMs are popular. Can use kernels with LR and LDA as well, but computations are more expensive.

10 Unsupervised Learning

In *supervised learning methods* such as regression and classification, we observe both a set of features X_1, X_2, $\cdots$, X_p for each object, as well as a response or outcome variable Y. The goal is then to predict Y using X_1, X_2, $\cdots$, X_p.

In *unsupervised learning*, we observe only the features X_1, X_2, $\cdots$, X_p. We are not interested in prediction, because we do not have an associated response variable Y. The goal is to discover interesting things about the measurements: is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations? We discuss two methods:

• *principal components analysis*, a tool used for data visualization or data pre-processing before supervised techniques are applied, and

• *clustering*, a broad class of methods for discovering unknown subgroups in data.

Principal components analysis (PCA). PCA produces a low-dimensional representation of a dataset. It finds a sequence of linear combinations of the variables that have maximal variance, and are mutually uncorrelated. Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data visualization.

• The first principal component of a set of features X_1, X_2, $\cdots$, X_p is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$$

that has the largest variance. By *normalized*, we mean that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. We refer to the elements ϕ_{11}, ϕ_{21}, $\cdots$, ϕ_{p1} as the loadings of the first principal component; together, the loadings make up the principal component loading vector, $\phi_1 = (\phi_{11}, \phi_{21}, \cdots, \phi_{p1})^T$.

To compute principal components, suppose we have an $n \times p$ data set $\mathbf{X}$. We assume that each of the variables in $\mathbf{X}$ has been centered to have mean zero. We then look for the linear combination of the sample feature values of the form

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \cdots + \phi_{p1}x_{ip}$$

for $i = 1, \cdots, n$ that has largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. Since each of the x_{ij} has mean zero, then so does z_{i1} (for any values of ϕ_{j1}). Hence the sample variance of the z_{i1} can be written as $\frac{1}{n}\sum_{i=1}^{n} z_{i1}^2$.

Plug in the first principal component loading vector solves the optimization problem

$$\max_{\phi_{11},\cdots,\phi_{p1}} \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^2 \text{ subject to } \sum_{j=1}^{p}\phi_{j1}^2 = 1.$$

The problem can be solved via a singular value decomposition of the matrix X. We refer to Z_1 as the first principal component, with realized values z_{11}, $\cdots$, z_{n1}.

The loading vector ϕ_1 with elements ϕ_{11}, ϕ_{21}, $\cdots$, ϕ_{p1} defines a direction in feature space along which the data vary the most. If we project the n data points x_1, $\cdots$, x_n onto this direction, the projected values are the principal component scores z_{11}, $\cdots$, z_{n1} themselves.

• The second principal component is the linear combination of X_1, $\cdots$, X_p that has maximal variance among all linear combinations that are uncorrelated with Z_1. The second principal component scores z_{12}, z_{22}, $\cdots$, z_{n2} take the form

$$z_{i2} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip},$$

where ϕ_2 is the second principal component loading vector, with elements ϕ_{12}, ϕ_{22}, $\cdots$, ϕ_{p2}.

• It turns out that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction ϕ_2 to be orthogonal (perpendicular) to the direction ϕ_1. And so on.

• The principal component directions ϕ_1, ϕ_2, ϕ_3, $\cdots$ are the ordered sequence of right singular vectors of the matrix $\mathbf{X}$, and the variances of the components are $\frac{1}{n}$ times the squares of the singular values. There are at most $\min(n-1, p)$ principal components.

Clustering methods. Clustering refers to a very broad set of techniques for finding subgroups, or clusters, in a data set. We seek a partition of the data into distinct groups so that the observations within each group are quite similar to each other. To make this concrete, we must define what it means for two or more observations to be similar or different. Indeed, this is often a domain-specific consideration that must be made based on knowledge of the data being studied.

There are two clustering methods:

• In *K-means clustering*, we seek to partition the observations into a pre-specified number of clusters.

• In *hierarchical clustering*, we do not know in advance how many clusters we want; in fact, we end up with a tree-like visual representation of the observations, called a *dendrogram*, that allows us to view at once the clusterings obtained for each possible number of clusters, from 1 to n.

K-means clustering. The idea behind K-means clustering is that a good clustering is one for which the within-cluster variation is as small as possible. The within-cluster variation for cluster C_k is a measure $WCV(C_k)$ of the amount by which the observations within a cluster differ from each other, where

$$WCV(C_k) = \frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij} - x_{i'j})^2$$

with $|C_k|$ denoting the number of observations in the kth cluster. Hence we want to solve the problem

$$\min_{C_1,\cdots,C_k}\left\{\sum_{k=1}^{K} WCV(C_k)\right\}.$$

The K-means clustering algorithm is:

• 1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.

• 2. Iterate until the cluster assignments stop changing:

2.1 For each of the K clusters, compute the cluster centroid. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

2.2 Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

This algorithm is guaranteed to decrease the value of the objective at each step, because

$$\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2 = 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2$$

where $\bar{x}_{kj}=\frac{1}{|C_k|}\sum_{i\in C_k} x_{ij}$ is the mean for feature j in cluster C_k.

Hierarchical clustering. K-means clustering requires us to pre-specify the number of clusters K. This can be a disadvantage. Hierarchical clustering is an alternative approach which does not require that we commit to a particular choice of K.In this section, we describe *bottom-up* or *agglomerative* clustering. This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram is built starting from the leaves and combining clusters up to the trunk.

Hierarchical clustering algorithm in words:

• Start with each point in its own cluster.

• Identify the closest two clusters and merge them.

• Repeat.

• Ends when all points are in a single cluster.

References

[1] Data School. "In-depth introduction to machine learning in 15 hours of expert videos". http://www.dataschool.io/15-hours-of-expert-machine-learning-videos/, 2014.9.3. 1

[2] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning : with applications in R.* New York, Springer, 2014. 1, 11

[3] Nate Silver. *The Signal and the Noise: Why So Many Predictions Fail–but Some Don't.* Penguin Books, 2015. 2

[4] Yan Zeng. "Book summary: *Econometrics for dummies*". http://opensrcsolutions.webs.com/, 2015. 3