Summary of Introduction to Statistical Learning

Contents

1 Basics

Definition 1.1. The *training data* is a set of n vectors called *observations*,

$$\{(x_{i,1}, x_{i,2}, \dots, x_{i,p}, y_i) \mid i = 1, \dots, n\}.$$

$(x_{i,1}, \dots, x_{i,p})$ are observed values for the p input variables (or *predictors*) $X_1, \dots, X_p$ and y_i is the observed value for output variable (or *response*) Y, at the i-th observation. So, the training data can be represented by an $n \times p$ matrix.

Supervised vs unsupervised learning In *unsupervised learning* problems, the observations have no output variable. The aim is finding relations between the observations, for example dividing them into distinct groups (*cluster analysis*). There is also *semi-supervised* learning, if we only have output measurements for some of the observations.

Regression vs classification Regression problems have quantitative (i.e. numerical) response variables. Classification problems have qualitative response variables (aka discrete variables, or classes, or categories, or levels). Inputs can also be quantitative, qualitative or ordered categorical. Usually the type of method is chosen depending on whether the output is quantitative or qualitative and can be used regardless of the input type.

Estimating f We suppose that input and output variables are related by a formula

$$Y = f(X) + \varepsilon$$

where ε is an error term. Then the aim is estimating f in order to predict the output for new inputs (**prediction** problems) or to understand the relation between input and output (**inference** problems). That is, we want to find an $\widehat{f}$ such that

$$f(X) \approx \widehat{f}(X).$$

The error term ε in the first formula is called *irreducible error*, because it is intrinsic in the measurement; the error $|f - \widehat{f}|$ is called *reducible error* because it can be potentially improved by finding a better $\widehat{f}$.

Parametric vs non-parametric methods This can be done with parametric or non-parametric methods (think: linear regression vs nearest-neighbors).

With *parametric methods*, estimating f comes down to estimating a set of parameters. One should first choose a model. For example, choosing a *linear model* means assuming that

$$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

Then *fitting (or training) the model* means estimating the $p+1$ parameters $\beta_0, \dots, \beta_p$; the most common method to achieve this is the least squares. *Overfitting* means that the model follows the error noise too closely (it happens for example when the the number of predictors is large compared to the number of training observations).

A *non-parametric method* makes no assumptions on the form of f and tries to follow the data points as much as possible under some smoothness requirement. These methods assure that the model always matches the data well, but they require a large number of observations. There is danger of overfitting, if the amount of smoothness required is not correct.

Flexibility vs interpretability *Flexibility* of a model means how many kinds of functions it can produce to estimate f (e.g. linear models produce only linear functions). On the other hand, *interpretability* of a model means how easy it is to understand the relation between input and output variables. For inference problems, one prefers interpretable models.

1.1 Measuring the fit

To measure the quality of fit, we need *test observations* (x_0, y_0), i.e. observations outside the training data; then we can compute the *test mean squared error (test MSE)*

$$E_{x_0}(\widehat{f}(x_0) - y_0)^2$$

and choose the model for which it is minimal.

The MSE over the training data (*training MSE*) is not enough: as the flexibility (degrees of freedom) of the model increases, the training MSE decreases monotonically, but the test MSE may start growing again from a certain point (U-shaped curve) (think KNN, training error is always zero for $k = 1$). The training MSE is not a good estimate of the test MSE. When the test MSE is larger than it would be with a less flexible model, we say we are *overfitting* the data.

How do we estimate the test MSE without data? One method will be *cross-validation* 4.1.

1.2 Bias-variance trade-off

The *expected test MSE* at some input x_0 is the test MSE averaged on different estimations of f using a large number of training sets (we are only able to estimate it since f is unknown). It satisfies:

$$e(x_0) \coloneqq E_{\widehat{f}}(y_0 - \widehat{f}(x_0))^2 = \mathbb{V}(\widehat{f}(x_0)) + \mathrm{Bias}(\widehat{f}(x_0))^2 + \mathbb{V}(\varepsilon)$$

(Averaging this now over the possible x_0 in the test set, yields the overall expected MSE).

This shows we must select a method with simultaneously low *variance* (how much the estimate changes when we change the training data set; more flexible methods have higher variance) and low bias (the error we introduce by approximating the phenomenon, e.g. by using a linear model to describe something non-linear; more flexible methods have lower bias). We want to have less variance when we have little data.

More interpretable	More flexible
Less variance	More variance
More bias	Less bias
Suited for inference problems	Danger of overfitting
Require less training data	Require more training data

In real life, how do we compute the test MSE, since f is unobserved? We will estimate it with *cross-validation*.

Enhancements Kernel methods improve KNN using weights decreasing to 0 with the distance from the point, rather than 0/1 weights of KNN. Furthermore, we may emphasize some variables more than others.

Linear models with basis expansion of the inputs.

Local regression.

1.3 Classification problems

When working with classification problems, we can quantify the accuracy of the estimate of $\widehat{f}$ using the following quantity (average over test observations (x_0, y_0) as before):

$$E(I(y_0 \neq \widehat{y}_0)) \quad \text{where } I(y_0 \neq \widehat{y}_0) = \begin{cases} 1 & y_0 \neq \widehat{y}_0 \\ 0 & y_0 = \widehat{y}_0 \end{cases}$$

and $\widehat{y}_0$ is the class predicted according to $\widehat{f}$. It can be shown that this is minimized, on average, by assigning each test input x_0 to the most likely class, i.e. the class j for which the probability

$$P(Y = j \mid X = x_0)$$

is largest. This is called a *Bayes classifier* (it is impossible to compute for real data because one should know the probability distributions).

—-

If $\mathbf{x} \in R_k$, we assign it to class C_k, i.e. we predict $y = k$. The misclassification rate is minimized if we maximize the posterior probability $p(y = k|\mathbf{x})$: indeed, we should maximize

$$\sum_{k=1}^{K} P(\mathbf{x} \in R_k | y = k) = \sum_{k=1}^{K} \int_{R_k} p(\mathbf{x}, y = k) \mathrm{d}\mathbf{x};$$

using relations $p(\mathbf{x}, y) = p(\mathbf{x}|y = k)p(y = k) = p(y = k|\mathbf{x})p(\mathbf{x})$ and noting that the factor $p(\mathbf{x})$ is common to all terms in the sum, we conclude.

Loss/utility If some misclassifications are to be considered worse than others, a loss function is used to account for that. Suppose $\mathbf{x}$ belonging to class C_k is incorrectly assigned to C_j (i.e. $\mathbf{x} \in R_j$, $y = k$): we define the loss as a number L_{kj}, an element of a loss matrix L. Now we seek to minimize the average loss

$$\mathbb{E}(L) = \sum_k \sum_j \int_{R_j} L_{kj} p(\mathbf{x}, y = k) \mathrm{d}\mathbf{x}$$

which is achieved, reasoning as before, by minimizing $\sum_k L_{kj} p(y = k, \mathbf{x})$. The converse concept of loss is utility, which should then be maximized.

Reject option: we avoid classifying inputs $\mathbf{x}$ with $p(y = k|\mathbf{x})$ less than some threshold θ (the more ambiguous cases).

Approaches for inference and decision

- Generative models: those that model the distribution of the inputs too, so that we can generate new data points in the input space: model the joint distribution $p(\mathbf{x}, y)$, or equivalently determine $p(\mathbf{x}|y)$ (need a large enough training set) and $p(y)$ (often just the fraction of training data in each class), then deduce $p(\mathbf{x}) = \sum p(\mathbf{x}|y)p(y)$ and hence $p(y|\mathbf{x}) = p(\mathbf{x}|y)p(y)/p(\mathbf{x})$. Computationally demanding if we just need the posterior probabilities (the class-conditional densities may contain a lot of unneeded structure).
- Discriminative models: model $p(y|\mathbf{x})$
- Find a discriminant function $f(\mathbf{x})$ mapping directly $\mathbf{x}$ onto a class, without using probabilities.

2 Regression

2.1 Simple linear regression with least squares

We say regression of Y onto X. This is the case of a single input variable X. We assume Y is approximately a linear function of X. More precisely, note the different levels: we write

$$Y = f(X) + \varepsilon_0 = (\beta_0 + \beta_1 X + \varepsilon_1) + \varepsilon_0 = \beta_0 + \beta_1 X + \varepsilon \approx \widehat{\beta}_0 + \widehat{\beta}_1 X.$$

ε_0 is the irreducible error; $f(X)$ is the "real" relation, which might be non-linear, hence the error term ε_1 (*model bias*): the line β is its best linear approximation, unknown. The line $\widehat{\beta}$ will be our computed estimation of this line. We may simply assume that f is exactly the line β and call $\varepsilon = \varepsilon_0 + \varepsilon_1$. Here are the different levels:

f
β
predictions $\widehat{\beta}$
observations

We use the least squares method to compute coefficients $\widehat{\beta}_0, \widehat{\beta}_1$ which estimate the coefficients β_0, β_1. First we require that the training data (x_i, y_i) $(i = 1, \dots, n)$ satisfies this approximation; we then look for $\widehat{\beta}_0$, $\widehat{\beta}_1$ minimizing the *residual sum of squares*:

$$\text{RSS} \coloneqq \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2 \coloneqq \sum_{i=1}^{n} (y_i - \widehat{\beta}_0 - \widehat{\beta}_1 x_i)^2$$

The minimizers are found to be (overline denotes the mean over the training data):

$$\widehat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \qquad \widehat{\beta}_0 = \overline{y} - \widehat{\beta}_1 \overline{x}$$

Definition 2.1. The line $Y = \beta_0 + \beta_1 X$ is the *population regression line*, which is the best linear approximation of f; the line $Y = \widehat{\beta}_0 + \widehat{\beta}_1 X$ is the *least squares* line.

This is similar to the difference between the population mean and the sample mean. This analogy is further justified by the following fact: the estimators $\widehat{\beta}_i$ are *unbiased*: their averages tend to β_i as the number of data sets increases.

Measuring the fit *Standard errors* tell us how much these estimations differ from β_0, β_1:

$$\mathrm{SE}(\widehat{\beta}_0)^2 = \mathbb{V}(\varepsilon)\left(\frac{1}{n} + \frac{\overline{x}^2}{\sum_{i=1}^n (x_i - \overline{x})^2}\right) \qquad \mathrm{SE}(\widehat{\beta}_1)^2 = \frac{\mathbb{V}(\varepsilon)}{\sum_{i=1}^n (x_i - \overline{x})^2}$$

and $\sigma = \sqrt{\mathbb{V}(\varepsilon)}$ (unknown) can be estimated with the *residual standard error*, the "average amount" that the output deviates from the regression line:

$$\mathrm{RSE} = \sqrt{\mathrm{RSS}/(n-2)}.$$

($n-2$ is the degree of freedom). One should relate this to the context (e.g. the mean of the output values) to get a percentage error. This is one of the ways to measure the fit of the model.

If we want to measure the fit with a number between 0 and 1, we can use the R^2-statistic:

$$R^2 := \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} := 1 - \frac{\sum_{i=1}^n (y_i - \widehat{y}_i)^2}{\sum_{i=1}^n (y_i - \overline{y})^2}$$

which can be shown to equal $\mathrm{Cor}(X,Y)^2$ in the single input variable case. It measures the percentage of variability of Y that is explained using the regression, that is, explained by the predictors. Values close to 1 indicate a strong association. In many typical applications where the linear model is a rough simplification, it's realistic to get values below 0.1.

Hypothesis test We can use standard errors to compute *confidence intervals* for β_0, β_1, or to perform *hypothesis tests* on them (is there a relationship between X and Y? I.e., is $\beta_1 \neq 0$?).

How far from zero should $\widehat{\beta}_1$ be to conclude that $\beta_1 \neq 0$? That depends on $\mathrm{SE}(\widehat{\beta}_1)$, more precisely on the *t-statistic*

$$t = \frac{\widehat{\beta}_1 - 0}{\mathrm{SE}(\widehat{\beta}_1)}.$$

If $\beta_1 = 0$ we expect that it has a t-distribution with $n-2$ degrees of freedom, similar to a normal distribution, so we can compute the the probability (*p-value*) of observing values $\geqslant |t|$. Small p-values indicate that our hypothesis $\beta_1 = 0$ is unlikely, and we reject the null-hypothesis. Typical p-values cutoffs are 5% or 1%.

2.2 Multiple linear regression

We can do multiple linear regression with p coefficients as long as $p \leqslant n$ (otherwise see *forward selection* below).

It may happen that a simple regression coefficient is non-zero while the corresponding coefficient in a multiple regression model is zero: this happens if the relation observed in the simple model is not really due to that input variable, but to some other variable related to that which is considered among the variables in the multiple model (*confounding*: a correlation which does not imply causation; one of the variables influences both the outcome and another variable).

Hypothesis test

Definition 2.2 (F-statistic)**.** To see if at least one of the predictors is related to the response, we need to test the null-hypothesis $\beta_1 = \cdots = \beta_p = 0$. We do it with the F-statistic

$$F = \frac{(\mathrm{TSS} - \mathrm{RSS})/p}{\mathrm{RSS}/(n-p-1)}:$$

one can show that the denominator has expectation $\sigma^2 = \mathbb{V}(\varepsilon)$ and, if the null hypothesis holds, the numerator too, otherwise it is strictly bigger. Hence a value of F around 1 indicates the null-hypothesis and bigger values suggest that at least one of the input variables is related to the output. How large should it be? The smaller n, the larger the values of F should be to deduce a correlation. If H_0 is true and the ε_i have a normal distribution (or anyway if n is large), it follows an F-distribution of which we may compute the p-value.

More in general, to test the null-hypothesis $\beta_1 = \cdots = \beta_q = 0$ for some subset of q variables, we use fit a new model which *excludes* those q variables and check the corresponding F-statistic

$$F = \frac{(\mathrm{RSS}_q - \mathrm{RSS})/q}{\mathrm{RSS}/(n-p-1)}$$

where RSS_q is the RSS of such model (note that in the case $q = p$, it degenerates to TSS) .

In the tests for one-variable $q = 1$ these F-statistics are equivalent to (are the square of) the corresponding t-statistics. (Note: these single t- or F-statistics are *not* the ones from simple regression, but still multiple: they are still influenced by the whole set of predictors. However, now we explain why it's still not enough to look at these single statistics and we should look at the overall ones instead).

Remark 2.3. The overall F-statistic is useful because one cannot always use the individual p-values to conclude that some variable is correlated to the output: if p is large, then some p-values will happen to be small by chance alone, even if H_0 is true. In this case, about 5% of the individual p-values will be below 0.05, whereas there is only a 5% probability that the p-value of the overall F-statistic will be below 0.05.

Which predictors are related to the response?

Remark 2.4 (Variable selection)**.** Once we conclude, using the F-statistic, that some predictors are associated to the response, we must find out which ones. Looking at the individual p-values could be misleading if p is large, as explained above. We could fit a different model for each subset of variables, but that is clearly infeasible.

- Forward selection: we add, one at a time, the variable which produces the lowest RSS models (1-variable, 2-variable, etc). We continue until some stopping rule is satisfied (e.g. all variables have a p-value below some threshold). This method can be used even if $p > n$.
- Backward selection: we start with the whole model and remove the variable with the largest p-value, then fit the model again and repeat until some stopping rule is satisfied.
- Mixed selection: start with the forward selection. If, at a certain point, some p-values become too large, we remove that variable from the model. We continue until some stopping rule is satisfied and all variables have a sufficiently low p-value.

Model fit Model fit is measured with RSE or R^2. The R^2 always increases when more variables are added to the model, but increases very little if the association is weak.

Confidence/Prediction intervals We use confidence intervals to account for the reducible error $|f(X) - \widehat{Y}|$. A 95% confidence interval means that, if we construct a different confidence interval for a huge number of data sets, 95% of them will contain the true value of $f(X)$. The 95% confidence interval is roughly constructed as $\widehat{\beta}_i \pm 2\mathrm{SE}(\widehat{\beta}_i)$.

Prediction intervals are wider than confidence intervals because they also account for the irreducible error. A 95% prediction interval is defined analogously.

Qualitative predictors We use one or more dummy variables (one less than the number of levels of the qualitative predictor) that take values in e.g. $\{0,1\}$, $\{-1,1\}$ etc. There will be a level (arbitrary choice) with no dummy variables known as the *baseline.*

Non-linear extensions of the model

Definition 2.5. An *interaction term* is an additional predictor such as X_1X_2 in the model $Y = \beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1X_2$. The p-value for β_3 indicates whether the true relationship should include the mixed term or not. On the other hand, it may happen that the mixed term has a low p-value but the *main effects* (X_1 and X_2) do not; the *hierarchical principle* states that, whenever we add an interaction term, we should also include its main effect even if the associated p-values are not significant.

We may further extend the model to polynomial regression by introducing new variables $X_2 = X_1^2$, $X_3 = X_1^3$, ... (as many as is optimal) interpreting the equation as linear with added variables. How many is optimal? Look at the coefficients and p-values or alternatively use the approach described in Cross Validation below.

2.3 Possible problems

Identifying non-linearity We plot the graph predictor-residuals (x_i, $e_i = y_i - \widehat{y}_i$) for a simple regression or fitted values-residuals ($\widehat{y}_i$, e_i) for a multiple regression, and hope to see no discernible pattern in the graph. If there seem to be non-linear relations, a simple approach is to try using non-linear transformations of the predictors like $\log X$, $\sqrt{X}$, X^2 etc.

Correlated error terms Happens for example in time series; plotting the graph (time, residuals) should show no discernible pattern if the errors are uncorrelated, otherwise we will see *tracking* (adjacent residuals having similar values). Another case of correlated errors: predicting height from weight if some people belong to the same family or eat the same diet etc.

Non-constant variance of error terms This is called *heteroscedasticity*; the frequent case is that the variances increase with the value of the response (e.g. expenditure for meals becomes more variable for wealthier people, measurements become less precise at longer distances, ...). It can be seen from the residual plot as a funnel shape. A possible solution is to transform the response with a concave function like $\log Y$, $\sqrt{Y}$.

Outliers are points y_i far from the predicted values $\widehat{y}_i$. The presence of an outlier may influence the RSE or the R^2 a lot even if it doesn't influence the regression line too much. We identify them using again residual plots, especially *studentized residuals* (each residual is divided by its estimated standard error), which, if greater than 3, indicate a suspect outlier. An outlier might occur due to errors in data collection (and therefore could simply be removed) but also due to a deficiency with the model such as a missing predictor.

High leverage points , on the other hand, are points with unusual predictor values. They may influence the regression line more than outliers. The *leverage statistic* for simple linear regression is

$$h_i := \frac{1}{n} + \frac{(x_i - \overline{x})^2}{\sum_{j=1}^{n}(x_j - \overline{x})^2}$$

which is between $1/n$ and 1, and the average of all the h_i is $(p+1)/n$, so if some h_i is much greater we may suspect the point has high leverage.

Collinearity happens when two predictors are correlated, e.g. they decrease or increase together, making it hard to tell apart their effects on the response. It reduces the accuracy of the estimates of the coefficients: the curve $(\beta_1, \beta_2) \mapsto \text{RSS}$ shows that there is a broad range of coefficient values resulting in similar RSS. The standard error for $\widehat{\beta}_j$ grows, hence the t-statistic declines, so the *power* of the hypothesis test is reduced and p-value grows (e.g. the p-value of one of the two variables may increase more than the other meaning that the importance of the former variable is masked by the latter).

We may try detecting collinearity by looking at the correlation matrix of the predictors, but there could still be a problem of *multicollinearity*, that is, a collinearity between three or more variables with no pair of these having a particularly high correlation. This is why we compute the *variance inflation factor (VIF)*

$$\frac{1}{1 - R^2_{X_j|X_{-j}}}$$

(values above 5 or 10 are suspect). The solution is either dropping one of the collinear predictors or combining the collinear predictors into a single one.

2.4 K-nearest neighbors (KNN) regression

A non-parametric method. Fix an integer value for K, take a prediction point x_0; let N_0 be the set of the K training observations closest to x_0; then we estimate $f(x_0)$ as the average of their responses:

$$\widehat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$$

The result is a step function (there are regions of constant predictions) which gets smoother for larger choices of K (the optimal value depends on the bias-variance tradeoff: smaller values are the most flexible with high variance since the prediction depends on just one observation; larger values may cause bias by masking some of the structure of f). $K = 1$ just interpolates (passes through) the data. Later we introduce approaches for estimating error rates.

Which method is best? The parametric method is better if the chosen parametric form is close to the true form of f, unless the number of observations per predictor is small. By comparing test MSE, we see that the less linear f is, the better KNN is (with a quite large K). But in higher dimensions, KNN often performs worse than linear regression even in highly non-linear cases: spreading a certain number of observations over many dimensions results in the *curse of dimensionality* (a given observation having no nearby neighbors).

3 Classification

3.1 K-nearest neighbors (KNN)

In real life we can't compute a Bayes classifier, so we must estimate the probabilities. One (non-parametric) method is the *K-nearest neighbors* classifier (KNN): x is assigned to the most common class among the training observations closest to x (for example, if we must assign a color blue/orange and we have a single blue point in the middle of an orange cluster, it will be, incorrectly, assigned color orange). Precisely, given a test input x_0, consider the set N_0 of the K points in the *training* data which are closest to x_0 and estimate $P(Y = j \mid X = x_0)$ as

$$\frac{1}{K} \sum_{i \in N_0} I(y_i = j).$$

The choice of K: small K corresponds to an overly flexible method with low bias but high variance; large K corresponds to a low-variance high-bias method, close to linear.

For the relation between training error rate and test error rate, the same considerations as in the regression case apply (as $1/K$ increases, the training error decreases while the test error has an U-shape). It is thus critical to choose the right level of flexibility.

3.2 Least squares for classification?

Why not encode the outcome as a quantitative variable where integers $0, 1, \dots$ represent the classes? For a binary response, this can be done and gives a linear decision boundary. We interpret a least squares regression line as an estimate of the probability $P(Y = 1 \mid X)$ (although some values could be outside $[0, 1]$). We must convert the fitted values back into classes by a rule assigning to 1 if $\widehat{Y} > 0.5$...

However, for responses with more than 2 classes, this approach could only work if the outcomes can be ordered and we may think of the difference between consecutive classes as more or less the same; otherwise, changing the order of the encoding will produce substantially different predictions. So we must develop different methods.

3.3 Logistic regression

Suppose a binary response; instead of modeling Y, we model the probability $P(Y = 1 \mid X) =: p(X)$. Then we decide to assign X to class 1 if $p(X) > 0.5$, or according to some other threshold.

We want to keep $p(X)$ between 0 and 1 to avoid the problem arising with the linear regression approach, so we decide to use the *logistic function*

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}.$$

We must estimate the coefficients β_0, β_1. From the relation $\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$ we might, passing to the log (this quantity is called the *log-odds*), fit the resulting model with least squares. However we prefer the general *maximum likelihood* method: we choose β_0, β_1 so as to maximize the likelihood function

$$l(\beta_0, \beta_1) := \prod_{i\colon y_i=1} p(x_i) \prod_{j\colon y_j=0} (1 - p(x_j)). \tag{1}$$

The null hypothesis is $\beta_1 = 0$.

For multiple-class classification an extension of logistic regression is possible, however we mainly use the following method.

3.4 Linear discriminant analysis (LDA)

Bayes' theorem implies

$$p_k(X) := P(Y = k \mid X = x) = \frac{P(Y = k)P(X = x \mid Y = k)}{\sum_l P(Y = l)P(X = x \mid Y = l)} =: \frac{\pi_k f_k(x)}{\sum_l \pi_l f_l(x)}.$$

π_k can be estimated as the fraction of training observations belonging to class k over all training observations; the problem now is estimating the densities $f_k(x)$ (once we know them, we can use the Bayes classifier, which sends an observation to the class for which $p_k(X)$ is largest). How to estimate this quantity?

Case $p = 1$ For LDA we must suppose f_k are all Gaussian with μ_k and $\sigma_k^2 = \sigma^2$ for all k, that is

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right).$$

Plug it into $p_k(x)$ and take the log; then the Bayes classifier must assign the observation to the class maximizing the *discriminant function*

$$\delta_k(x) := x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log \pi_k.$$

In real life we have to estimate π_k, μ_k and σ in the natural way:

$$\begin{aligned}\widehat{\mu}_k &:= \frac{1}{n_k}\sum_{i:y_i=k} x_i\\ \widehat{\sigma}^2 &:= \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\widehat{\mu}_k)^2\\ \widehat{\pi}_k &:= n_k/n\end{aligned}$$

and plugging these estimates into $\delta_k(x)$ gives us $\widehat{\delta}_k(x)$ to be maximized.

Case $p > 1$ Now an observation $x = (x_1, \dots, x_p)$ in class k is drawn from a multivariate Gaussian distribution $N(\mu_k, \Sigma)$ where μ_k is a mean vector and Σ is the covariance matrix, common among all classes. $f(x)$ takes the multivariate Gaussian density form; plug it into $p_k(X)$ and get the matrix form of $\delta_k(x)$:

$$\delta_k(x) := x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k.$$

The parameters are estimated similarly as before. There are as many decision boundaries (the sets of x such that $\delta_k(x) = \delta_l(x)$ for $k \neq l$) as pairs of classes, $\binom{K}{2}$.

Remark 3.1. The *confusion matrix* for a binary classifier is

		True status		
		$+$	$-$	Total
Prediction	$-$	T_-	F_+	N
	$+$	F_-	T_+	P
	Total	N^*	P^*	Total

$-$ may be thought as the null hypothesis.

- The false positive rate F_+/N is the *type 1 error.*
- The false negative rate is F_-/P is the *type 2 error.*
- The true positive rate T_+/P is the *sensitivity* of the classifier.
- The true negative rate T_-/N is the *specificity* ($1-$ false pos. rate).
- The positive predictive value T_+/P^* is the *precision.*
- The negative predictive value is T_-/N^*.
- true neg. rate $+$ false pos. rate $= 1 =$ false neg. rate $+$ true pos. rate.

Even if the overall error rate is low, one of the two might still be very high; the Bayes classifier, approximated by LDA, has the lowest total error rate, but we may want to modify LDA if we need prefer a reduction in one of the two error types at the cost of a slightly higher total error rate; we do that by changing the threshold to a value other than 50%. We may look at the 3 (threshold, error rate)-graphs: the overall and one for the incorrectly classified data of each category.

The *ROC curve* is the curve parametrized by (false positive rate, true positive rate). The area under the curve measures the overall performance of the classifier, the closer to 1 the better. They are useful for comparing different classifiers, because they take all possible thresholds into account.

3.5 Quadratic discriminant analysis (QDA)

Finally we may assume an observation $x = (x_1, \dots, x_p)$ in class k is drawn from a multivariate Gaussian distribution $N(\mu_k, \Sigma_k)$ where μ_k is a mean vector and Σ_k is a covariance matrix, for each k. $f(x)$ takes the multivariate Gaussian density form; plug it into $p_k(X)$ and get

$$\delta_k(x) := -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k) + \log \pi_k$$

which is now quadratic in x.

Estimating a single Σ requires estimating $p(p+1)/2$ parameters; with k covariance matrices we must estimate $Kp(p+1)/2$ parameters. The LDA becomes linear in x so there are only Kp linear coefficients to estimate. Therefore, the LDA is much less flexible than QDA, hence may have more bias; so, we want to use QDA when we are not concerned about high variance because we have a large training set, or when we can't assume a common variance among classes.

3.6 Comparison of methods

KNN is non-parametric, unlike logistic regression and LDA and QDA, so it can be better in the case of highly non-linear decision boundaries (as long as we choose the level of smoothness, i.e. choose K, correctly). However it doesn't tell us what predictors are important, whereas the other methods give us coefficients and p-values.

Logistic regression and LDA both produce linear decision boundaries: ($p = 1$) using the LDA formula for $p_1(x)$ we get the log-odds $\log\left(\frac{p_1(x)}{1-p_1(x)}\right) = c_0 + c_1 x$ where c_0, c_1 are functions of μ_1, μ_2, σ^2). So the log-odds has a linear form like in logistic regression, with the difference that β_0, β_1 in LR are estimated using maximum likelihood, whereas c_0, c_1 are computed with estimated mean and variance from a normal distribution. The same connection between LR and LDA holds for $p > 1$.

However, LDA can be better if the assumptions about Gaussian distribution with common covariance matrix among classes holds, otherwise LR can be better.

QDA is a compromise between KNN and LR/LDA.

3.7 Extensions

We can also use transformations of the predictors to work with non-linear relations, like in the regression problems. For instance, we could include X^2, $X^3, \dots$ as predictors. This may improve things if the added flexibility reduces the bias enough. Adding all quadratic terms in LDA would give a QDA model but we would get different parameter estimates.

4 Resampling methods

Repeatedly drawing samples from a training set and refitting a model on each sample in order to get information. *Model assessment* means evaluating a model's performance. *Model selection* means selecting the proper level of flexibility for a model.

4.1 Cross-Validation (CV)

We estimate the test error rate by holding out a subset of training observations from the fitting process and then using it to test the model (though our predictions will be made on the model fit with the whole set?).

Validation set this approach involves randomly dividing the set into two parts, a training set and a validation set. Possible problems to consider: high variability between the test errors of different validation sets; fewer observations used to fit the model (because a part is used for testing) so the test error tends to be overestimated. We address these them in the following methods. We start with the regression setting.

Leave-One-Out Cross-Validation (LOOCV) If $\{x_1, \dots, x_n\}$ is the set of observations, we use the singleton $\{x_i\}$ as validation set and the remaining $n-1$ observations are used for training; for each $i = 1, \dots, n$ we fit the model and we predict $\widehat{y}_i$. Then the estimation for the MSE will be the average of the single MSE's:

$$\mathrm{CV}_{(n)} \coloneqq \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2.$$

Problem: can be computationally expensive if n is large. For least squares linear/polynomial regression, however, $\mathrm{CV}_{(n)}$ can be computed by a single (the original) model fit and there is a magic equality $\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i-\widehat{y}_i}{1-h_i}\right)^2$ where the predictions this time are from the original fit and h_i is the leverage.

k-fold Cross-Validation More generally, we divide the set of observations into k approximately equal groups and fit k models using each time a different group for validation and the remaining $k-1$ sets for fitting. For the i-th group we compute MSE_i and the overall MSE estimate is their average $\mathrm{CV}_{(k)} \coloneqq \frac{1}{k}\sum \mathrm{MSE}_i$.

We typically use $k = 5$ or $k = 10$. Since the split of the data is random, we get slightly different results every time we run the algorithm, but only slightly.

If we use simulated data to compare the true test MSE to the CV estimates, we see that the two CV methods perform very similarly; they sometimes underestimate the true MSE, however the shape is quite accurate so it is not a problem if we just want to find the minimum point in the test MSE curve (as we sometimes want when comparing different methods).

k-fold CV often gives more accurate estimates because of the bias-variance trade-off. LOOCV gives approximately an unbiased estimate because it contains almost all of the observations, but with higher variance because it is the mean of correlated quantities, since the n models are trained on almost identical sets.

Cross-Validation (CV) for classification Works the same by averaging the $I(y_i \neq \widehat{y}_i)$.

To compare different classification methods we plot the graphs (flexibility, error rate) using a CV method.

More...

- Ensemble learning: methods that aggregate different prediction models.
- Voting classifier:
  - Hard voting classifier: aggregates the prediction and selects the mode (the prediction with the most votes). Can have much better accuracy than each single predictor.
  - Soft voting: weighted probabilities.
- Bagging and pasting

  - bagging = bootstrap aggregating: same algorithm trained on different subsets of the training set, sampled with replacement.
  - pasting: like above but without replacement.

- Aggregation function is usually the statistical mode.

4.2 The bootstrap

In order to compute the variability of an estimator, with simulated data we can generate a large number of independent data sets. With a real data set of n observations, we can emulate this by simply generating sets of n random observations sampled with replacement from the original set.

5 Improvements of least squares

Subset selection To improve interpretability we may set some coefficients to zero if the corresponding predictors are irrelevant (referred to as *noise*, as opposed to the *signal*). We will describe some approaches for this variable selection.

Shrinkage (or regularization) Least squares has high variance if n and p are comparable, and infinite variance if $p > n$ because the coefficients are not unique anymore. Here we add a constraint to the estimate of coefficients in order to reduce the variance.

Dimension reduction We project the p predictors on an m-dimensional subspace, $m < p$, and use these m projections as predictors to fit the model with least squares.

5.1 Subset selection

Best subset selection We fit all possible 2^p models (one for each subset of predictors) and choose the best, as follows.

Let M_0 be the null model, containing no predictors and simply predicting the sample mean for each observation. For $k = 1, \ldots, p$ fit all $\binom{p}{k}$ models containing exactly k predictors and call M_k the one minimizing training error (having the smallest RSS or equivalently the largest R^2). Then choose the best among $M_0, \ldots, M_p$ using CV or other methods described below (now we can't use RSS or R^2 because the training error decreases monotonically as we add more predictors; we must use test error here).

Problems: this can only be done for small p for computational reasons and also because with a large search space we have higher chances of finding models that look good on training data despite having no predictive power on future data, leading to overfitting and high variance of coefficient estimates. In the following stepwise methods the set of models considered is far more restricted.

Forward stepwise selection M_0 null model. Call M_1 the best-trained (smallest RSS or highest R^2) model among the p models with 1 predictor. Call M_2 the best-trained model among the $p-1$ models obtained adding one more predictor to M_1, and so on. Then choose the best among $M_0, \ldots, M_p$ in the same way as before (CV etc.)

Here we must only fit $1 + \sum_{k=0}^{p-1}(p-k) = 1 + p(p+1)/2$ models. Of course it's not guaranteed to find the best model out of the 2^p.

This is the only method that we can use if $n < p$, in which case, however, we can only construct models $M_0, \ldots, M_{n-1}$ because fitting with least squares will not yield a unique solution if $p \geqslant n$.

Backward stepwise selection We start with the model with all p predictors and remove them one at a time with the same logic as before.

Mixed selection Start as in forward selection, but after each new variable is added we may also remove another one which is no longer improving model fit.

Choosing the optimal model To estimate the test error, we may either make an adjustment to the training error or estimate it directly using one of the validation approaches already discussed.

Examples of adjustments. For a least squares model with k predictors, the C_p estimate of test MSE is

$$C_p := \frac{1}{n}(\mathrm{RSS} + 2k\widehat{\sigma}^2) = \mathrm{MSE} + \frac{2k\widehat{\sigma}^2}{n}$$

where $\widehat{\sigma}^2$ is an estimate of the variance of ε (it can be shown that if it's unbiased then C_p is an unbiased estimate of the test MSE).

The adjusted R^2 is

$$R^2 = 1 - \frac{\mathrm{RSS}/(n-k-1)}{\mathrm{TSS}/(n-1)}$$

which does not increase monotonically anymore as the number of variables grows, but may increase of decrease due to the presence of k.

We may estimate test MSE using the above methods as well as validation methods, then compare the curves (#predictors, error estimation) between the different methods. However, there is some uncertainty (e.g. repeating CV with different splits of the data into sets would give different results) and if some of these models appear to be more or less equally good for different numbers of parameters, then it's common to choose the smallest one (the one with the least parameters); to do this we use the *one-standard-error rule*: calculate the SE of the estimated MSE for each model size, then select the smallest model for which this SE is within one standard error of the lowest point on the curve.

5.2 Shrinkage methods

Ridge regression Instead of minimizing the RSS, we minimize

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

where the *tuning parameter* λ controls how much the coefficients will be shrunk (the coefficients tend to 0 as $\lambda \to \infty$). We call $\widehat{\beta}_\lambda^R$ the vector of these coefficients. As λ increases, the flexibility of the ridge regression fit decreases as well as variance, whereas bias grows, so we can choose the optimal λ as the one minimizing test MSE (we do this by CV).

Replacing X_j by cX_j yields a scaling $\widehat{\beta}_j/c$ of the coefficient, hence $X_j\widehat{\beta}_j$ remains the same, whereas the ridge regression coefficients may change substantially instead. Hence it is better to apply ridge regression after standardizing the predictors, i.e. using

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

This method has great computational advantages because it requires fitting only one model, and actually the computations required to solve the regression equation simultaneously for all values of λ are almost identical to those for fitting a least squares model.

The disadvantage: it will always include all p predictors in the final model, bad for interpretability if p is large. This is addressed in the following method.

Lasso (least absolute shrinkage and selection operator) The quantity to minimize to get the Lasso coefficients $\widehat{\beta}_\lambda^L$ is the same as above but replacing β_j^2 by $|\beta_j|$ in the penalty term (we say it's an ℓ_1 penalty instead of an ℓ_2 penalty). This has the effect of forcing some of the coefficient estimates to be exactly zero when λ is sufficiently large (the reason can be seen geometrically: the solution is the intersection between the RSE ellipses and a diamond which has corners on the axes), so the lasso performs variable selection (we say it generates *sparse models*). Depending on the value of λ, the Lasso can generate a model involving any number of parameters. The optimal value for λ is the one with minimal CV error and this value may do a good job at identifying the signal and setting noise coefficients to 0.

Ridge vs Lasso Equivalently, ridge (resp. lasso) regression coefficients solve the problem of minimizing RSS subject to the constraint $\sum \beta_j^2 \leqslant s$ (resp. $\sum |\beta_j| \leqslant s$); so, the coefficients will be the points with minimal RSS among the points that lie within the circle (resp. diamond) defined by the constraint equation.

There is a close connection to best subset selection because the problem of finding coefficients minimizing RSS and such that no more than s coefficients are non-zero is equivalent to the constraint $\sum I(\beta_j \neq 0) \leqslant s$.

Ridge regression will be better when all the predictors are useful, otherwise the lasso will be better.

Intuition: ridge regression shrinks the coefficient by multiplying by a scalar, whereas lasso reduces them by subtracting a fixed amount and sends to zero those whose absolute value is less than that scalar.

5.3 Dimension reduction

Let $Z_1, \dots, Z_m$ be $m < p$ linear combinations $Z_h = \sum_{j=1}^p \varphi_{jh} X_j$, then we can fit the linear regression model

$$y_i = \theta_0 + \sum_{h=1}^m \theta_h z_{ih} + \varepsilon_i, \quad i = 1, \dots, n$$

so we have reduced the problem to estimating $m+1$ coefficients (instead of $p+1$). This approach can outperform least squares regression if we choose the φ_{jh} wisely. Substituting the definition of Z_h inside the above formula we see that this is like the original model with β_j subject to the constraint $\beta_j = \sum_{h=1}^m \theta_h \varphi_{jh}$. When $m \ll p$ this reduces the variance. We describe two approaches for defining the φ_{jh}.

Principal components analysis (PCA) / regression (PCR) Let X be an $n \times p$ data matrix; the *first principal component* is the direction along which, if we project the observations, there is the largest possible variance; alternatively, the line that is as close as possible to the data, minimizing the squared (perpendicular!) distance between the points and the line (compare to residual sum of squares: these are not residuals and here we only work with predictors). It is given by

$$Z_1 = \varphi_{11}(X_1 - \overline{x}_1) + \dots + \varphi_{p1}(X_p - \overline{x}_p)$$

where the overlines are the means of the all the n j-th components. This is also an n-vector, $X_j = (X_{1j}, \dots, X_{nj})$ is an n-vector, $j = 1, \dots, p$, the subtraction of a scalar is defined in the natural way, $\sum_{i=1}^p \varphi_{i1}^2 = 1$ so we don't blow up the variance, and we have

$$z_{i1} = \varphi_{11}(X_{i1} - \overline{x}_1) + \dots + \varphi_{p1}(X_{ip} - \overline{x}_p).$$

called the *first principal component score* for the i-th observation (in the 2-dimensional case it's the distance from 0 of the projection of the point on the line).

The second principal component would be a linear combination of the variables that is uncorrelated with Z_1 (equivalently, it is orthogonal to Z_1!) and has largest variance subject to this constraint, and so on. One can construct up to p distinct principal components.

The first one is the one capturing the most information: it is a single number summary of the predictors. If they have a linear relationship, this can work well. If the second principal component scores are closer to zero, this tells us that it's not as important. The idea is that most of the variability in the data can be explained by a small number of directions (principal components) and our assumption is that these are also the ones that can explain the relationship with Y (however this is not guaranteed, since the directions are identified in an *unsupervised* way, i.e. without using Y). This motivates the following approach of using only the first $m < p$ principal components as predictors. If our assumption is true, this will be better than the original model, because estimating less coefficients will reduce overfitting.

We construct $Z_1, \ldots, Z_m$ and use these as predictors in a linear regression model fit using least squares. It is recommended to standardize the predictors before generating the principal components, otherwise high-variance variables will play a larger role. The number m is chosen by CV.

How well it performs depends on how much the data depends on the principal components. Note that this is not a feature selection methods because the Z_i are linear combinations of all of the original predictors. PCR is related to ridge regression and can be thought as a continuous version of it.

Partial Least Squares (PLS) This is a supervised alternative to PCR: it identifies a new set of features $Z_1, \ldots, Z_m$ in a supervised way (finding directions that explain both the predictors and the response) and then fits the linear model with least squares.

Standardize the predictors, then compute Z_1 as above by setting φ_{j1} equal to the coefficient from the simple linear regression of Y onto X_j, which can be shown to be proportional to $\mathrm{Cor}(X_j, Y)$. Hence the highest weight is on the variables with the strongest correlation to the response. Then we regress each variable on Z_1 and take residuals, interpreted as the information not explained by the first PLS direction, and we compute Z_2 using this orthogonalized data in the same way as Z_1 was computed from the original data.

5.4 High-dimensional setting

The *high-dimensional* setting is when $p \not\ll n$, that means, either when $p > n$ or p is slightly smaller than n. This is often the case today, when we may have extremely large p (e.g. single chemical particles, a list of terms entered into a search engine) but n (e.g. a list of patients, a list of users) might be limited by cost, sample availability, etc.

In high dimension one needs special methods. For example, if we performed linear regression with n not much bigger than p, we have overfitting of the data: think of the case of 1 predictor and 2 observations, then we have a perfect fit regardless of the values of those observations (therefore, we should never measure model fit on training data, but always on validation sets). We can't use the adjustments methods (e.g. C_p) because here we have $\widehat{\sigma}^2 = 0$ (estimation with RSS?).

We see that, counter-intuitively, as the number of features p grows, the test MSE also grows (unless those features are useful), because of overfitting: this is called *curse of dimensionality.* We fix this by choosing less flexible models using selection/shrinking/PCR methods. We need to use larger values of λ to keep the test MSE low (whereas in low dimension the best results are obtained for smaller values of λ).

Interpreting results: in high dimension, there is an extreme problem of multicollinearity because any variable can be written as a linear combination of all the others. So we can never know exactly which variables are truly predictive of the outcome, and we can hope at best to assign large coefficients to the variables which are correlated to those. The model can still be very effective but there can be many different other ones.

6 Non-linear models

6.1 Polynomial regression

Already discussed: we fit the model

$$Y_i = \beta_0 + \beta_1 X_i + \cdots + \beta_d X_i^d + \varepsilon_i \qquad i = 1, \ldots n;$$

we rarely go beyond the third or fourth degree because the curve can become overly flexible, especially near the boundary.

Least squares gives us the covariance matrix $\widehat{C}$ for the $\widehat{\beta}_j$ and then, for a fixed x_0, we have the estimated variance

$$\mathbb{V}(\widehat{f}(x_0)) = (1, x_0, \ldots, x_0^d)\widehat{C}(1, x_0, \ldots, x_0^d)^T,$$

the estimated pointwise standard error $\mathrm{SE}(\widehat{f}(x_0))$ which is its square root, and an approximate 95% confidence interval given by $\widehat{f}(x_0) \pm 2\mathrm{SE}(\widehat{f}(x_0))$.

6.2 Step functions

They're useful to avoid imposing a global structure. Divide the range of X into k *bins*, by choosing cutpoints $c_1, \ldots, c_k$; create $k+1$ new dummy variables $C_0(X) = I(X < c_1)$, $C_i = I(c_i \leqslant X < c_{i+1})$, then fit

$$Y_i = \beta_0 + \beta_1 C_1(X_i) + \cdots + \beta_k C_k(X_i) + \varepsilon_i.$$

The constant pieces can of course miss an increasing trend, etc.

6.3 Basis function

This approach generalizes the previous two. We fit the model

$$Y_i = \beta_0 + \beta_1 b_1(X_i) + \cdots + \beta_k b_k(X_i) + \varepsilon_i$$

for fixed known functions $b_j(X_i)$, which are seen as the predictors of a linear model which we fit with least squares.

6.4 Regression splines

Piecewise polynomial regression The points where the coefficients β change are called *knots*; the model for a single knot at c takes a form like

$$Y_i = \begin{cases} \beta_{01} + \beta_{11} X_i + \cdots + \varepsilon_i & X_i < c \\ \beta_{02} + \beta_{12} X_i + \cdots + \varepsilon_i & X_i \geqslant c \end{cases}$$

so we fit two different polynomial functions, one for observations with $X_i < c$ and one for $X_i \geqslant c$. More knots lead to more flexible models. Placing K different knots leads to fitting $K+1$ polynomials.

Constraints and splines If the curve becomes too flexible, we may place continuity constraints on the curve and its derivatives. A *degree-d spline* is a piecewise polynomial of degree d with continuity up to degree $d-1$ at each knot.

Each constraint takes away one degree of freedom, e.g.:
piecewise cubic with 1 knot: 8 degrees of freedom;
cubic spline with 1 knot: 5 degrees of freedom $= 8 - (3$ constraints). A cubic spline with k knots has $4+k$ degrees of freedom.

Spline basis representation To fit a degree-d spline with k knots we can use the basis functions model

$$Y_i = \beta_0 + \beta_1 b_1(X_i) + \cdots + \beta_{k+3} b_{k+3}(X_i) + \varepsilon_i$$

for an appropriate choice (not unique) of b functions. A way to do it for a cubic spline is to use

$$X, X^2, X^3, h(X,\xi_1), \ldots, h(X,\xi_k)$$

where, for each knot ξ, $h(X,\xi)$ is a *truncated power basis* function

$$h(X,\xi) := \begin{cases} 0 & X \leqslant \xi \\ (X-\xi)^3 & X > \xi \end{cases}$$

which amounts to estimating $k+4$ regression coefficients, so fitting a cubic spline with k knots uses $k+4$ degrees of freedom.

Natural spline Splines can have high variance near the boundary of X, so we may add boundary constraints requiring that the function be linear at the boundary, where X is smaller than the first knot or larger than the last knot.

Choosing number and location of the knots Usually knots are placed at uniform quantiles, specifying the desired numbers of degrees of freedom. To choose the number we try different options and then we either see which produces the best looking curve or use CV.

Regression splines are better than polynomial regression because we can keep lower degrees while increasing flexibility with the number of knots; we can also choose to place more knots where f seems to be changing more rapidly.

Double descent is when, after the first U shape, the test error starts decreasing again as we increase flexibility. If we consider a spline with a number of degrees of freedom such that it is interpolating the data, the test error may be very high, but if we add more degrees of freedom, then we have an infinite number of least squares coefficient estimates and we can select the *minimum-norm* solution, minimizing $\sum_{j=1}^{d} \widehat{\beta}_j^2$: this gives the smoothest curve among the possible choices, and this may be much better than the previous (less-flexible) model. This especially works well with high *signal-to-noise ratio* $\mathbb{V}(f(X))/\sigma^2$, which measures how close to the true curve the data points are.

6.5 Smoothing splines

We want to find a function $g(x)$ minimizing the RSS $= \sum_{i=1}^{n}(y_i - g(x_i))^2$ but satisfying some smoothness requirements to avoid overfitting the data (which would happen, for example, by just interpolating the y_i). A way to do that is to find g minimizing

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

where λ is a tuning parameter controlling bias-variance tradeoff. The second term is a penalty term penalizing the variability in g. The integral of g'' measures how much in total g' varies (we don't want the slope of g to vary too much, i.e., don't want g to wiggle too much).

One can show $g(x)$ is then a natural cubic spline with knots at $x_1, \dots, x_n$, but shrunken depending on λ. We refer to the *effective degrees of freedom* df_λ because usually degrees of freedom refers to the number of free parameters such as the number of coefficients fit in a polynomial spline, n in this case, but some of them are constrained and df_λ takes it into account; it decreases from n to 2 as λ goes from 0 to ∞.

To find the best λ we use CV. There is a formula to compute the the LOOCV RSS error of smoothing splines with essentially the same cost of computing a single fit.

6.6 Local regression

We estimate the response at x_0 by regression using only the observations near x_0, weighted depending on their distance. We choose the number k of local points; we assign weights $w_i = w(x_i, x_0)$ (the function to do this is a choice to be made), then we find $\widehat{\beta}_0$ and $\widehat{\beta}_1$ minimizing

$$\sum_{i=1}^{n} w_i(y_i - \beta_0 - \beta_1 x_1)^2$$

and the fitted value at x_0 is $\widehat{f}(x_0) = \widehat{\beta}_0 + \widehat{\beta}_1 x_0$. The fraction $s = k/n$ of local observation over the total is the *span*, which controls flexibility like a tuning parameter.The smaller the value of s, the more local and flexible the fit; it can be chosen by CV or specified directly. We may also choose which type of regression to perform, like constant, linear or quadratic. When working with multiple predictors we may even use *varying coefficient models*: we fit a multiple linear regression model that is global in some variables and local in another, such as time. This can be generalized to models which are local in p variables by using p-dimensional neighborhoods (but if $p \gg 3$ or 4 there will usually not be enough observations close to x_0).

6.7 Generalized additive models (GAM)

Now we study non-linear (additive) models in more variables.

GAM for regression problems We replace the linear terms in $y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$ with smooth functions f_j by writing

$$y_i = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \varepsilon_i.$$

This model is additive because of the summation of the f_j.

Using the previous methods, we fit each f_j and get graphs $(X_j, f_j(X_j))$. (Using smoothing splines takes a bit more work and an approach known as *backfitting*). Additivity implies that we can still examine the effect of each X_j on Y individually while holding the other variables fixed. The main pro of GAM's is that we don't need to manually try out different transformations of the variables in order to fit a non-linear model.

Although the model is additive we may still manually include additional interaction terms $f_{jk}(X_j, X_k)$ and fit them using two-dimensional methods.

GAM for classification problems For simplicity we assume $Y \in \{0, 1\}$. In logistic regression we assumed the logit was a linear function of the predictors; now we assume

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \sum_{j=1}^{p} f_j(X_j)$$

7 Tree-based methods

7.1 Regression trees

To make a prediction for $(x_1, \dots, x_p)$, we split the range of X_1 into two parts e.g. $X_1 < c$, $X_1 \geqslant c$; the region containing x_1 is split further along the X_2 axis, analogously, and we go on in this fashion. Then the prediction is the mean (or the mode) response for the region containing $(x_1, \dots, x_p)$. The regions created are called *terminal nodes* or *leaves* of the tree. The nodes of the tree where the predictor space is split are called *internal nodes.* Segments connecting nodes are *branches.*

The regions into which we partition the space could have any shape, but we require them to be rectangles for simplicity. Now we want to find boxes R_j that minimize the RSS

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

where $\widehat{y}_{R_j}$ is the mean response for the training observations in R_j. To do this we use a top-down (begins at the top of the tree where all observations belong to a single region) greedy (at each step the best split is made without looking ahead) algorithm called recursive binary splitting. We choose j and s such that splitting into

$$R_1(j, s) := \{X \mid X_j < s\} \quad R_2(j, s) := \{X \mid X_j \geqslant s\}$$

leads to the lowest RSS, i.e. find j and s minimizing

$$\sum_{i : x_i \in R_1(j,s)} (y_i - \widehat{y}_{R_1(j,s)})^2 + \sum_{i : x_i \in R_2(j,s)} (y_i - \widehat{y}_{R_2(j,s)})^2.$$

Then we repeat the process, this time splitting one of these two regions. The process continues until we reach a stopping criterion, e.g. no region containing more than five observations.

This process however may lead to overfitting the data; a tree with fewer splits may lead to lower variance at the cost of some bias. Idea: only split if the decrease in RSS due to that split is worth it. Problem: a split that seemed worthless could have been followed by a very useful split! Solution: grow a large tree T_0 and then *prune* it to obtain a subtree. How do we do it? We can't CV all possible subtrees! So we have *tree pruning* methods.

7.2 Classification trees

In this case the predicted response will be the most common class of training observations in that region. To grow the tree we can't look at RSS; the natural alternative would be *classification error rate* (fraction of training observations in the region that don't belong to the most common class, computed as $1 - \max_k \widehat{p}_{mk}$ where $\widehat{p}_{mk}$ is the proportion of training observations in the mth region belonging to the kth class). In practice we prefer two other measures which are more sensitive. The *Gini index* $G = \sum_{k=1}^{K} \widehat{p}_{mk}(1 - \widehat{p}_{mk})$ takes small values if the mth node is *pure*, i.e. all the $\widehat{p}_{mk}$ are close to 0 or 1, i.e. the node contains predominantly observations from a single class. The *entropy* $D = -\sum_{k=1}^{K} \widehat{p}_{mk} \log \widehat{p}_{mk} \geqslant 0$ also measures purity. Purity is important to be confident that our prediction is correct.

Characteristics of trees and improvements If the relation is highly non-linear, we will be better off using trees rather than linear models. They are easily interpretable. However they can be very non-robust: a small change in the data can cause a large change in the final tree. They suffer from high variance: the results vary a lot on different training sets. The following methods can be used to reduce the variance and improve predictive accuracy.

7.3 Bagging

This also works for regression methods other than trees. We reduce the variance of a method using *bootstrap aggregation* or *bagging.* Averaging a set of observation reduces variance to σ^2/n; so, we could take many training sets from the population, build a model for each one and then average the predictions. Since we don't have many training sets, we use the bootstrap 4.2.

In case of classification trees, instead of the average we take the most common class.

How many models (B) should we build with the bootstrap? The more the better, until the error is low enough.

Out-of-bag (OOB) error To estimate the error of a bagged model with large data sets, it is convenient to use OOB error estimation (virtually equivalent to LOOCV for large B). It can be shown that on average each bagged tree makes use of around 2/3 of the observations; the remaining third are the out-of-bag (OOB) observations, so we have around $B/3$ predictions for each observation to average (or take the majority vote from) returning a single prediction, and we can compute the overall OOB MSE/classification error.

Interpretability We have improved prediction accuracy at the expense of interpretability. How do we interpret results? We can use the total amount that the RSS/Gini index is decreased due to splits over a given predictor, averaged over all trees. A large value indicates an important predictor.

7.4 Random forests

Again we build a number of trees using bootstrap, but at each split, a random sample of m predictors (typically $m \approx \sqrt{p}$) is chosen and the split is allowed to use only one of those predictors. Why? Because if there is a very strong predictor, then it would be used in the top split by most of the bagged trees which would then look quite similar, and averaging many highly correlated quantities does not lead to a large reduction of variance as averaging many uncorrelated quantities. This problem is overcome with random forests: on average, $(p-m)/p$ splits will not even consider the strong predictor. This is a way of *decorrelating* the trees. The case $m = p$ is bagging. Using small m will be helpful in case of many correlated predictors.

7.5 Boosting

This also works for regression methods other than trees. We describe it for regression trees only (more complex for classification trees).

We construct B trees as follows: start with $\widehat{f}(x) = 0$ and $r_i := y_i$ for $i = 1, \dots, n$; for $b = 1, \dots B$ we fit a tree $\widehat{f}_b$ to the data (X, r) with d splits, we update $\widehat{f}(x) \leftarrow \widehat{f}(x) + \lambda \widehat{f}_b(x)$ and update the residuals $r_i \leftarrow r_i - \lambda \widehat{f}_b(x_i)$, and iterate. The boosted model is $\widehat{f}(x) = \sum_{b=1}^{B} \lambda \widehat{f}_b(x)$.

Note we are fitting the trees to the residuals. Here B too large leads to overfitting so we will choose it with CV. This approach *learns slowly* because we are slowly improving $\widehat{f}$, and the parameter λ controls the rate at which boosting learns; typical values are 0.01 or 0.001. d controls the complexity of the boosted ensemble.

8 Classification with support vector machines (SVM)

This is a method for classification with two classes. Consider an $n \times p$ matrix X consisting of n training inputs, i.e. p-vectors $x_i = (x_{i1}, \dots, x_{ip})$ with $y_i \in \{-1, 1\}$, and consider a test observation $x^* = (x_1^*, \dots, x_p^*)$.

We want to construct a separating hyperplane, separating the two classes, i.e. such that: if the hyperplane has equation $f(X) = \sum_{i=1}^p \beta_i X_i$, then $f(x_i) > 0$ if $y_i = 1$ and $f(x_i) < 0$ if $y_i = -1$. Then x^* will be classified according to the sign of $f(x^*)$ and moreover, if $f(x^*)$ is far from zero, then we can be sure that our assignment is right.

8.1 Maximal margin classifiers

If our data can be perfectly separated by a hyperplane, then there are infinite such hyperplanes (by just shifting or rotating it a bit). The natural choice to use is the maximal margin hyperplane (a.k.a. optimal separating hyperplane) which is the one farthest from the observations. The training observations with minimum distance from the hyperplane are called *support vectors* (because if they were moved then the hyperplane would move as well). The maximal margin hyperplane depends only on the support vectors! The *margin* is the minimum distance between the observations and the mmh, and margin also refers to the two hyperplanes parallel to the mmh and passing through the support vectors. We hope that a classifier with large margin on the training data will also have large margin on the test data.

How do we construct it? We must maximize M (the width of the margin) subject to

$$\sum_{i=1}^p \beta_i^2 = 1$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geqslant M \text{ for } i = 1, \dots, n$$

The second condition ensures that each observation lies on the correct side: it would be enough to require > 0, but this way we also require that there is enough distance. Indeed, the first condition can be shown to imply that the left side of the inequality is the distance of the i-th observation from the hyperplane.

What do we do if a separating hyperplane does not exist?

8.2 Support vector classifiers

In the previous case, adding a single observation may change the mmh dramatically; this suggests the mmh may overfit the training data. We use a support vector classifier a.k.a. soft margin classifier: instead of seeking the largest margin, we allow some observations to be past the margin (whence the name soft), or even on the wrong side of the hyperplane, which is why the method can be used in case a separating hyperplane does not exist.

Solve the optimization problem of maximizing M (the width of the margin) subject to

$$\sum_{i=1}^p \beta_i^2 = 1$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \geqslant M(1 - \varepsilon_i)$$
$$\text{for } i = 1, \dots, n, \;\; \varepsilon_i \geqslant 0, \;\; \sum_{i=1}^n \varepsilon_i \leqslant C$$

where $C \geqslant 0$ is a tuning parameter and ε_i are *slack variables* that allow individual observations to be on the wrong side of the margin or the hyperplane: if $\varepsilon_i = 0$ then x_i is on the correct side of the margin; if $\varepsilon_i > 0$ it has violated the margin; if $\varepsilon_i > 1$ it is on the wrong side of the hyperplane. C determines how much the observations can violate the margin: no more than C observations can be on the wrong side of the hyperplane. The larger C gets, the more observations can violate the margin, hence the wider the margin gets (and the lower the variance, the higher the bias).

The hyperplane depends only on observations that lie on the margin or that violate it, known as support vectors. This is in contrast to LDA whose classification rule depends on all of the observations in each class, and it is similar to logistic regression which has low sensitivity to observations far from the decision boundary (LR and SVC are closely related).

The actual computation of the svc hyperplane is technical, but it can be shown that the it can be represented as

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

where the brackets denote inner product. To estimate the α's and β_0 all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_j \rangle$ between pairs of training observations. Moreover, α_i is nonzero only for training observations which are support vectors, so there are far fewer coefficients to consider.

8.3 Support vector machines (SVM)

What if the decision boundary is not linear? We could fit a SVC with $2p$ features X_1, X_1^2, X_2, X_2^2 etc., so the decision boundary will have equation $q(X) = 0$ with $q(X)$ quadratic in the X_i's. One could enlarge the feature space even more with higher-order monomials or interaction terms, and even non-polynomial functions of the X_i's, but this might enlarge it too much leading to unmanageable computations. SVM's allow us to enlarge it in an efficient way: we replace the above form of the hyperplane $f(x)$ with

$$f(x) = \beta_0 + \sum_{x_i \text{ supp.vec.}} \alpha_i K(x, x_i)$$

where the inner product is replaced with a more general *kernel*. This means we must only compute $\binom{n}{2}$ terms $K(x_i, x_j)$.

Examples of kernels: the inner products are $K(x_i, x_j) = \sum_{h=1}^{p} x_{ij} x_{hj}$, it quantifies the similarity of a pair of observations using Pearson (standard) correlation. A polynomial kernel of degree d is $K(x_i, x_j) = \left(1 + \sum_{h=1}^{p} x_{ij} x_{hj}\right)^d$.

8.4 Extensions to more than two classes

One-versus-one (or all-pairs) classification Suppose we have k classes; we construct all the $\binom{k}{2}$ svm's, we check our test observation on all of them and we assign it to the class to which it was assigned most frequently.

One-versus-all classification Suppose we have k classes; we construct the k svm's comparing one of the classes to the remaining $k-1$. If $f_i(X)$ is the equation of the classifier comparing class i coded as 1 to the others coded as -1, we assign x^* to the class for which $f_i(x^*)$ is largest.

8.5 Relationship to logistic regression

The svc optimization problem can be rewritten as

$$\text{minimize}\left\{\sum_{i=1}^{n}\max[0, 1-y_i f(x_i)] + \lambda\sum_{j=1}^{p}\beta_j^2\right\}$$

where larger values of tuning parameter λ allow more violations of the margins. This has the usual form minimize$\{L(\mathbf{X}, \mathbf{y}, \beta) + \lambda P(\beta)\}$ where L is a loss function quantifying the fit and P is a penalty.

We use LR if classes tend to overlap and use SVM's if they are well separated. We could also use non-linear kernels in other methods such as LR, although they are mainly used with SVM's. SVM can be extended to regression problems too.

9 Unsupervised learning

The problem of unsupervised learning is that there is no objective goal to achieve (such as the prediction of a response) or criterion to assess the quality of results (like the estimation of an error), rather it is a part of exploratory data analysis.

Both PCA and clustering seek to simplify the data via a small number of summaries, but PCA seeks a low-dimensional representation of the data explaining most of the variance, whereas clustering looks for homogeneous subgroups among the observations.

9.1 Principal components analysis (PCA)

Not only used in supervised learning as already seen, but also used as a tool for data visualization: it is unfeasible to examine all the $\binom{p}{2}$ two-dimensional scatterplots and they wouldn't be informative since they just contain a fraction of the total information, so PCA gives us a low-dimensional representation of the data that captures as much information as possible. (In many statistical techniques we may get less noisy results by replacing the $n \times p$ data matrix with the $n \times M$ data matrix whose columns are the first $M \ll p$ principal components).

The first principal component is the linear combination of the features

$$Z_1 = \varphi_{11}X_1 + \cdots + \varphi_{p1}X_p$$

which has the largest variance and is normalized, i.e. the *loading vector*

$$\varphi_1 = (\varphi_{11}, \ldots, \varphi_{p1})^T$$

(the direction of the first principal component) satisfies $\sum_{j=1}^{p}\varphi_{j1}^2 = 1$ (arbitrarily large loadings would result in arbitrarily large variance). We assume each variable has mean 0 (i.e. the column means of X are 0), since we only care about the variance. We want to maximize the variance of the *scores*

$$z_{i1} = \sum_{j=1}^{p}\varphi_{j1}x_{ij} = (x_{i1}, \ldots, x_{ip})\varphi_1 \qquad i = 1, \ldots, n$$

(they are the projections of the x_i's onto φ_1) subject to the normalization constraint, i.e. solve the following problem:

$$\text{maximize } \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\varphi_{j1}x_{ij}\right)^2 \qquad \text{subject to the above constraint.}$$

(the quantity on the left is the sample variance of the scores since those also have mean 0). This problem is solved by eigen decompositions.

Then Z_2 is found similarly as Z_1 (replacing φ_1 by φ_2), plus the constraint that it be orthogonal to Z_1 (see the first discussion of PCA).

Note: we have uniqueness up to sign flip of the loading vectors (because they just represent a direction) and score vectors (because the variance of Z and $-Z$ is the same).

Interpretation of the principal components A *biplot* is a square containing the n observations plotted according to the bottom and left axes which are respectively 1st and 2nd pc scores (one reads z_{i1} on the bottom axis and z_{i2} on the left axis, for $i = 1, \dots, n$), and containing also the p components of the 1st and 2nd loading vectors where one reads φ_{i1} on the top axis and φ_{i2} on the right axis for $i = 1, \dots, p$.

Among these p components, those which have larger values on one of the two loading vectors tell us roughly which information that loading vectors represent (the loading vector places more weight on those features). The components which are closer together are more closely correlated. Observations with high scores on one of the two components are those for which the feature represented by that component is more prominent.

Interpretation 2 φ_1 is the line in p-dimensional space that is closest to the n observations (using average squared Euclidean distance). φ_1 and φ_2 span a plane that is closest to the n observations, and so on. So, the first m score vectors and loading vectors provide the best m-dimensional approximation to the i-th observation: $x_{ij} \approx \sum_{h=1}^{m} z_{ih}\varphi_{jh}$ assuming the data matrix is column-centered. When $m = \min(n-1, p)$ (the maximum possible number of principal components) then this representation is an exact equality.

Scaling the variables As well as centering the variables, the results of PCA also depend on whether the variables have been scaled to have $\sigma = 1$ (unlike linear regression for example); if we don't, the first pc loading will be larger for variables that have a larger variance for mere reasons of measurement scales. If, however, the variables are measured in the same units, we may choose not to scale them.

Proportion of variance explained (PVE) How much information is lost by projecting the observations onto the first few principal components? The *total variance* of a data set (assuming mean 0) is

$$\sum_{j=1}^{p} \mathbb{V}(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2;$$

the variance explained by the m-th component is

$$\frac{1}{n} \sum_{i=1}^{n} z_{im}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \varphi_{jm} x_{ij} \right)^2$$

and therefore the PVE of the m-th principal component is

$$\frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{p} \varphi_{jm} x_{ij} \right)^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2} \geqslant 0$$

and the cumulative PVE for the first M components is the sum of these quantities.

How many principal components should we use? We examine the *scree plot* (principal component, PVE) and look where the PVE drops off (e.g. explains less than 10% of the variance or so). In practice, if the first pc's don't show interesting patterns then it's useless to use more. On the other hand, if we use pc's for a supervised analysis regression problem as done earlier on, we may use CV or a similar approach.

9.2 Clustering methods

We may cluster observations on the basis of the features or conversely.

K-means clustering We want to partition the (indices of the) observations into K sets $C_1, \ldots, C_k$. A good clustering is one minimizing the *within-cluster variation* for each cluster, defined for example as

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

We want to minimize $\sum_{k=1}^{K} W(C_k)$. How? There are almost K^n ways to partition n observations into K clusters! However the following algorithm easily gives a local optimum :

1. Randomly assign a number from 1 to K to each observation, creating random clusters.
2. For each cluster, compute its *centroid*, the p-vector whose components are the feature means. Assign each observation to the cluster whose centroid is closest.
3. Iterate the previous point until the assignments stop changing.

It works because of the equality $W(C_k) = 2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \overline{x}_{kj})^2$ where the overline term is the mean for the j-th feature in the k-th cluster. As it runs, the algorithm keeps improving until it no longer changes, but the results depend on the initial random cluster assignments, so we run the algorithm multiple times and then select the best solution, that with the smallest objective.

Problem: how do we choose K? We try different choices and look for the one with the most interpretable solution. To avoid pre-specifying K, we resort to the following method.

Hierarchical clustering It results in a *dendrogram* which lets us view at once the clustering obtained for each possible number of clusters. The leaves represent the observations and the similarity of two observations is given by the point on the vertical axis where the branches containing them first fuse (proximity along the horizontal axis is irrelevant).

To identify clusters, draw a horizontal line, the clusters are represented by the branches which are cut through by that line. We can look at the dendrogram and decide by eye how many clusters we want (decide where to cut) on the basis of the height of the fusions.

Limitation: the name is hierarchical is because the clusters obtained contain the smaller clusters of a finer clustering. This is not always good: in a group of people, the best 2-clustering could divide them by gender and the best 3-clustering could divide them by nationality, so we would have no nesting. In this case we would be better off with K-means clustering.

We use a bottom-up approach (start from the leaves of the tree). Define a *dissimilarity* measure between each pair of observations, e.g. by Euclidean distance or correlation-based distance (two observations are similar if their features are highly correlated). Start considering each observation as a single cluster, then iteratively fuse the two clusters which are most similar to each other until all observations belong to a single cluster and the dendrogram is complete. To do all of this, we must only define a notion of dissimilarity between clusters (*linkage*), which can be of the following kinds:

- complete: maximal intercluster dissimilarity, i.e. the largest dissimilarity out of all pairs with one observation in each of the two clusters.
- single: minimal intercluster dissimilarity.
- average: mean intercluster dissimilarity, i.e. the average of all pairwise dissimilarities.

- centroid: dissimilarity between the centroids of the two clusters. (This can result in *inversions*): two clusters are fused at a height below either of the individual clusters).

Considerations Should the variables be standardized when using clustering methods? Yes if we want the variables to have equal importance or if they are measured on different scales.

There are methods (but no consensus on the best one) to assign a p-value to a cluster to assess whether there is more evidence for the cluster than one would expect due to chance, or if we're just *clustering the noise.*

Both K-means and hierarchical clustering assign each observation to a cluster, but sometimes we may want to discard some outliers which would distort the clusters; this is done using so-called mixture models.

Clustering methods are not very robust to perturbations of the data: removing a random subset of observations and repeating the clustering might radically change it.

10 Deep learning and neural networks

Deep learning is a good choice when the sample size of the training set is very large and interpretability is not a high priority.

A neural network takes an input $X = (X_1, \dots, X_p)$ and predicts Y by building a non-linear function

$$\begin{aligned} f(X) &:= \beta_0 + \sum_{k=1}^{K} \beta_k A_k \\ &:= \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) \\ &:= \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j) \end{aligned}$$

- The nonlinear *activation function* g and the number K are chosen in advance;
- the h_k are learned during the training of the network;
- the *activations* A_k (the *units* of the *hidden layer*) are transformations of the original features, analogous to basis functions.

The result $f(X)$ is a linear regression model in the K activations. The β's and w's need to be estimated from the data, usually done by minimizing $\sum_{i=1}^{n}(y_i - f(x_i))^2$.

The *sigmoid* activation function is $g(z) = \frac{e^z}{1+e^z}$, the same used in logistic regression to convert a linear function to probabilities. The preferred choice today is the *ReLU (rectified linear unit)* function

$$g(z) = (z)_+ = \begin{cases} 0 & z < 0 \\ z & z \geqslant 0 \end{cases}$$

If g was linear we'd just have a linear model; g nonlinear can capture interaction terms etc; however we wouldn't use a function like $g(z) = z^2$ because we would always get second-degree polynomial coordinates.

Multilayer neural networks Usually there are multiple hidden layers, of sizes K_1, $K_2, \dots$; instead of using a single hidden layer with many units we use more layers of smaller size.

The k-th activation of layer ℓ is

$$A_k^{(\ell)} := h_k^{(\ell)}(X) = g\left(w_{k0}^{(\ell)} + \sum_{j=1}^{K_{\ell-1}} w_{kj}^{(\ell)} A_j^{(\ell-1)} \right) \qquad k = 1, \dots K_\ell$$

Note $K_0 := p$ and $A_j^{(0)} := X_j$.

Qualitative responses Neural networks are widely used for image classification. For greyscale images we use $p =$ width $\times$ length $=$ #pixels and each component of $X = (X_i)_i$ is the brightness value 0-255. If we have 10 classes (e.g. recognition of a digit), the output layer will have 10 linear models:

$$Z_m = \beta_{m0} + \sum_{j=1}^{K_L} \beta_{mj} A_j^{(L)}$$

where L is the number of layers. Then we use the *softmax* activation function:

$$f_m(X) = \Pr(Y = m | X) = \frac{e^{Z_m}}{\sum_{j=0}^{9} e^{Z_j}}$$

so that the ten $f_m(X)$ behave like probabilities and sum to one. To train the network we look for coefficients minimizing the *cross-entropy* (negative multinomial log-likelihood, generalizing 1):

$$-\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} \log(f_m(x_i)).$$

How to fit this? Minimizing that is not straightforward. The problem is non-convex so there are multiple solutions. We use *gradient descent*: consider a vector θ containing all the parameters and write the objective as a function of the parameters, $R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$; we start with a guess θ_0 and find a vector δ such that $R(\theta_0 + \delta) < R(\theta_0)$, and iterate this until the objective fails to decrease. The direction δ is found as $-\rho \nabla R(\theta_0)$, for a small enough *learning rate* ρ.

To avoid overfitting, when we have a huge number of weights in multilayer models, we should use regularization on the objective function, $R(\theta, \lambda)$ where we added a term $\lambda \sum_j \theta_j^2$.

10.1 Image classification with convolutional neural networks (CNN)

Each image has a resolution of width$\times$length with three 8-bit numbers per pixel representing red, green and blue, organized in a 3-dimensional array called *feature map*, whose axes are: width, length and *channel* (the one for colors). CNNs mimic the way humans classify images by recognizing specific features. First low-level features are identified (patches of color etc.), then combined to form higher-level features (parts of the ears, eyes etc.).

The architecture is the following. The image is first sent to a:

Convolution layer It is made of *convolution filters*, which are $\ell_1 \times \ell_2$ matrices (one for each channel), small compared to the image resolution. Each submatrix of the image gets multiplied by the convolution filter, and this picks out the characteristic of the convolution filter returning a convoluted image, which are the units of a hidden layer.

In image processing the filters are predefined; with CNN the filters are learned for the specific classification task. (*Weight freezing*: one can use pretrained hidden layers when working with small training sets, then train only the last few layers).

The convoluted image (black and white?) is then sent to a:

Pooling layer Condense large images into smaller summary images: e.g., the *max pooling* operation summarizes each non-overlapping 2×2 block of pixels using the maximum value in the block.

This process is iterated, then in the end the last pooling layer units are flattened, which means that the pixels are treated as separate units and fed into one or more layers before reaching the output layer, which is a softmax activation.

Data augmentation A trick: each training image is replicated with a slight distortion (shift, rotation etc.) not affecting human recognition; this is a way of increasing the training set protecting against overfitting.

10.2 Document classification

E.g. the document will be a given film review and the response will be *positive* or *negative.* The simplest way to featurize a document is the *bag-of-words* model: we take a language dictionary (or some subset) of M words, and assign to the document an M-vector of 1 and 0's depending on whether a word is present or not in the document (but we could also choose to record the number of occurrences). E.g., $n = 25000$ reviews and $M = 10000$ words, and the $n \times M$ training feature matrix X is sparse (mostly 0's).

Note that a 2-class neural network is a nonlinear logistic regression model: by substituting the above formulas for $f_m(X)$ and Z_m, we see that for K classes we only need to estimate $K - 1$ sets of coeficients:

$$\log\left(\frac{\Pr(Y=1|X)}{\Pr(Y=0|X)}\right) = Z_1 - Z_0 = (\beta_{10} - \beta_{00}) + \sum_{j=1}^{K_2} (\beta_{1j} - \beta_{0j}) A_j^{(2)}.$$

Rather than classification error, in machine learning it's more common to consider accuracy (fraction of correct assignments).

To take the context of each words into account, a method is the *bag-of-n-grams*: we record the consecutive co-occurence of every distinct consecutive sequence of n words.

10.3 Recurrent neural network (RNN)

In a RNN, the input X is a sequence and the RNN takes advantage of the sequential aspect just as the CNN with spatial structure. E.g., a text document is a sequence of L words $X = (X_1, \ldots, X_L)$ where each X_i is a word (it could be represented with one-hot encoding as a vector $X_i = (X_{i1}, \ldots, X_{ip})$ whose components correspond the words in a language dictionary, with only one 1 and all remaining 0's).

We also have a sequence $A = (A_1, \ldots, A_L)$ of hidden-layers, each $A_i = (A_{i1}, \ldots, A_{iK})$ consisting of K units. As the sequence is processed one X_i at a time, the network updates the activations A_i in the hidden layer taking as input X_i and A_{i-1}; each A_i feeds into the output layer and produces a prediction O_i for Y; the last one, O_L, is the most relevant.

- W: the $K \times (p + 1)$ matrix for the weights for the input layer;
- U: the $K \times K$ matrix for weights from hidden layers to hidden layers;
- B: the $K + 1$ vector of weights for the output layer.

$$A_{ik} = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{ij} + \sum_{s=1}^{K} u_{ks} A_{i-1,s}\right)$$

where g is an activation function such as ReLU; note W, U, B are not functions of i. The output O_i, for a quantitative response, is computed as

$$O_i = \beta_0 + \sum_{k=1}^{K} \beta_k A_{ik}$$

The loss function for an observation (X, Y) is $(Y - O_L)^2$ and therefore the parameters W, U, B are learned by minimizing $\sum_{i=1}^{n}(y_i - o_{iL})^2$.

Time series forecasting

- Time series exhibit *auto-correlation*: values close in time tend to be similar to each other. We can look at the graph (lag, autocorrelation) showing the correlation coefficient for observation i time units apart, as a function of i.
- Another characteristic of time series forecasting: the response variable is also a predictor (we use the past values to predict the next ones).
- We only have one series of data, not a big number n! This makes the structure of the problem very different.

We have three daily time series v_t, r_t, z_t, $t = 1, \ldots, T$ and we want to predict v_t from past values of all three series.

Consider $X = (X_1, \ldots, X_L)$ where $X_1 = (v_{t-L}, r_{t-L}, z_{t-L})$, $\ldots$, $X_L = (v_{t-1}, r_{t-1}, z_{t-1})$ (we want to use the past L values for our prediction, and L should be chosen with validation methods), and $Y = v_t$. We can create $T - L$ pairs (X, Y) for $t = L + 1, \ldots, T$.

We choose K and fit the model splitting the time series so that up to a certain point we use it as training data and afterwards we use it as test data; to make predictions in this test period, we will need to use the test data itself.