Statistical Learning Notes

June 19, 2020

Contents

Chapter 1

Linear Regression

Linear Regression is a parametric model where we assume a linear relationship between the dependent and independent variables.

$$X = (X_1, X_2, \dots, X_p)$$
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$$

Where X represents a p dimensional input and β are the coefficients, and ϵ is the error which is assumed to have $\mathcal{N}(0, \sigma^2$ distribution. Errors are assumed to be independent. We will usually not know the error or it's variance, and hence our estimate is denoted by $\hat{Y}$. Further, the estimted coefficients will also be denoted with a hat since we can never know the true model, but only get estimates of these parameters

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \beta_2 X_2 + \dots + \hat{\beta}_p X_p + \epsilon$$

1.1 Simple Linear Regression

Here, the independent variable is assumed to have a single dimension, and thus we are only looking towards estimating two coefficients. Let x_i be the i^{th} observation and Y_i be the associated response, then

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

$$\text{Minimize error to estimate coefficients} \quad \underset{\beta_0,\beta_1}{\text{minimize}} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(Y_i - \overline{Y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \quad \hat{\beta}_0 = \overline{Y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\theta}_1 \sim \mathcal{N}\left(\theta_1, \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)$$

$$\hat{\theta}_0 \sim \mathcal{N}\left(\theta_1, \sigma^2 \frac{\sum_{i=1}^{n} x_i^2}{n\left((\sum_{i=1}^{n} x_i^2) - n\bar{x}^2\right)}\right)$$

The mean squared error is used here as it is the natural error function that emerges when we try to obtain MLE estimates of the coefficients. As is visible from the fomulae for point estimates of coefficients, they are linear combinations of normal variables (Y) and thus are normally distributed.

Simple linear regression is discussed in detail in the probability notes. It also discusses confidence intervals for the coefficients, the prediction intervals for the response and hypothesis testing for relation between input and response.

1.1.1 Coefficient of Determination

R^2 is often used as a metric for checking how good the regression model is. It's definition is invariant to the number of independent variables

$$R^2 = \frac{S_{YY} - RSS}{S_{YY}} = 1 - \frac{RSS}{S_{YY}}$$
$$S_{YY} = \text{total variance in Y}, \quad RSS = \text{sum of squares of residuals}$$
$$S_{YY} - RSS = \text{total variance explained by inputs}$$

A good model will explain most of the variance in Y usig the input variables. Hence, R^2 close to 1 is a good model and vice versa. R^2 **usually increases as more and more variables are added to the model**.

1.2 Multiple Linear Regression

By minimizing the sum squared error in a fashion similar to simple linear regression case, we can obtain the regression coefficients as follows (X is $n \times p+1$ size with the first column containing all $1s$)

$$\begin{aligned}
\hat{Y} &= X\hat{\beta} \\
MSE &= (Y - \hat{Y})^T(Y - \hat{Y}) \\
\frac{\partial}{\partial\hat{\beta}} MSE &= 0 \\
\text{or,} \quad 0 &= \frac{\partial}{\partial\hat{\beta}}(Y^TY - Y^TX\hat{\beta} - \hat{\beta}^TX^TY + \hat{\beta}^TX^TX\hat{\beta}) \\
0 &= -Y^TX - Y^TX + \hat{\beta}^TX^TX + \hat{\beta}^TX^TX \\
(X^TX)\hat{\beta} &= X^TY \\
\hat{\beta} &= (X^TX)^{-1}X^TY \qquad (1.1)
\end{aligned}$$

We note that the coefficients are linear combinations of Y and thus normally distributed themselves.

$$\begin{aligned}
E[\hat{\beta}] &= E[(X^TX)^{-1}X^TY] = E[(X^TX)^{-1}X^T(X\beta + \epsilon)] \\
&= E[\beta] + (X^TX)^{-1}X^TE[\epsilon] \\
&= \beta \\
\text{Since} \quad Cov(\mathbf{x}) &= E[(\mathbf{x} - E[\mathbf{x}])(\mathbf{x} - E[\mathbf{x}])^T], \\
\text{and} \quad Cov(\epsilon) &= E[(\epsilon - 0)(\epsilon - 0)^T] = E[\epsilon\epsilon^T] = \sigma^2 \\
Cov(\hat{\beta}) &= Cov((X^TX)^{-1}X^TY) \\
&= E[((X^TX)^{-1}X^T(X\beta + \epsilon) - \beta)((X^TX)^{-1}X^T(X\beta + \epsilon) - \beta)^T] \\
&= (X^TX)^{-1}X^TX(X^TX)^{-T}E[\epsilon\epsilon^T] \\
&= (X^TX)^{-1}X^T\sigma^2 \\
\hat{\beta} &\sim \mathcal{N}(\beta, (X^TX)^{-1}\sigma^2)
\end{aligned}$$

$\hat{\beta}$ is a linear combination of independent Y_is and is normally distributed. Also, note that X^TX is symmetric and so will be it's inverse.

An unbiased estimator of σ^2 is

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad E[\hat{\sigma}^2] = \sigma^2$$

where the denominator is chosen to make the estimator unbiased. it can be shown that

$$(n-p-1)\frac{\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p-1}$$

with $\hat{\beta}$ and $\hat{\sigma}^2$ independent random variables.

1.2.1 Confidence Intervals

The mean response for any new input is also a random variable with the distributions

$$\begin{aligned}
E[x^T\hat{\beta}] &= x^T\beta \\
Var(x^T\hat{\beta}) &= E[(\hat{\beta}-\beta)(\hat{\beta}-\beta)^T] \\
&= x^T(E[\hat{\beta}\hat{\beta}^T] - \beta\beta^T)x = x^T E[(\hat{\beta}-\beta)(\hat{\beta}-\beta)^T]x \\
&= x^T(X^TX)^{-1}x\sigma^2 \\
x^T\hat{\beta} &\sim \mathcal{N}(x^T\beta, x^T(X^TX)^{-1}x\sigma^2)
\end{aligned}$$

Utilizing the unbiased estimate of σ^2 defined above, we can get the confidence and prediction intervals as follows (using t-distribution after dividing the normal distribution of response with the estimator of σ^2) for $1-\alpha$ confidence

$$\begin{aligned}
\text{confidence interval for mean response} \quad x^T\beta &\in x^T\hat{\beta} \pm t_{\alpha/2,n-p}\sqrt{x^T(X^TX)^{-1}x\sigma^2} \\
\text{prediction interval for response} \quad Y &\in x^T\hat{\beta} \pm t_{\alpha/2,n-p}\sqrt{1 + x^T(X^TX)^{-1}x\sigma^2}
\end{aligned}$$

1.2.2 Hypopthesis Testing

Single Coefficient

To test

$$H_0 : \beta_j = 0 \quad \text{versus} \quad H_1 : beta_j \neq 0$$

we utilize some of the above defined distributions

$$\begin{gathered}
\hat{\beta}_j \sim \mathcal{N}(\beta_j, \sigma^2 diag((X^TX)^{-1})_j) \\
(n-p-1)\frac{\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-p-1} \\
\frac{\hat{\beta}_j - \beta_j}{\sigma\sqrt{diag((X^TX)^{-1})_j}} \div \sqrt{(n-p-1)\frac{\hat{\sigma}^2}{\sigma^2(n-p-1)}} \sim t_{n-p-1} \\
\text{and under null hypothesis,} \quad \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{diag((X^TX)^{-1})_j}} \sim t_{n-p-1} \\
\text{where} \quad \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{diag((X^TX)^{-1})_j}} \quad \text{is often called } \textit{Z-score}
\end{gathered}$$

where variance utilizes the diagonal entry of variance of β. If we know the actual variance σ^2, we replace it in the above equation to get a normal distribution instead. A large value of the *Z-score* will lead to the elimination of the null hypothesis meaning the coefficient is not zero. The $1-\alpha$ confidence intervals for β_j then become

$$\beta_j \in \hat{\beta} \pm \hat{\sigma} t_{\alpha/2,n-p-1}\sqrt{diag((X^TX)^{-1})_j}$$

Group of Coefficients, F-test

Suppose we have a set of k coefficients for a categorical variable and we wish to test

$$H_0 : \beta_i = \beta_{i+1} = \cdots = \beta_k = 0 \quad \text{versus} \quad H_1 : \text{at least one of} \quad \beta_j \neq 0, j \in (i, \ldots, k)$$

Then, we use the **F-test** assuming H_0 is true

$$F = \frac{(RSS_0 - RSS_1)/k}{RSS_1/(n-p-1)}$$
$$\text{where} \quad RSS_0 = RSS \quad \text{of model without the } k \text{ coefficients}$$
$$\text{and} \quad RSS_1 = RSS \quad \text{of model with all the coefficients}$$
$$F \sim F_{k,n-p-1}$$

1.2.3 Multiple Outputs

We want to predict multiple outputs $Y_1, Y_2, \ldots, Y_k$ from the same set of variables. The RSS then becomes

$$RSS = \sum_{i=1}^{n}\sum_{j=1}^{k}(y_{i,j} - \hat{y}_{i,j})^2 = \sum_{i=1}^{n}\sum_{j=1}^{k}(y_{i,j} - \beta_{k,0} - \sum_{u=1}^{p}\beta_{k,u}x_{i,u})^2$$
$$\text{Minimizing,} \quad \hat{\beta} = (X_T X)^{-1}X^T Y \quad \text{where } Y \text{ is a } n \times k \text{ matrix}$$

Thus, the problem is similar to doing the linear regression independently on each of the $Y_j s$. One important assumption here is that the errors between different Y_j are not correlated with each other.

1.2.4 Coefficient Interpretation

The interpretation of coefficients for quantitative variables is straightforward. Further, suppose the equation has the form

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

Then, β_1 denotes the change in y that will be caused by changing the value of X_1 by 1 unit, provided all other inputs are constant. If we change two variables simultaneously, we can see their interaction together keeping the remaining variables constant.

However, this procedure is not straightforward for qualitative/categorical variables. Suppose we have one numeric variable, say age, and a variable denoting gender which we will constraint to have two values, 0 denoting male and 1 denoting female. The regression equation becomes

$$y = \beta_{age}x_{age} + \beta_{gender}x_{gender} + \epsilon = \begin{cases} \beta_{age}x_{age} + \beta_{gender} + \epsilon & \text{if male} \\ \beta_{age}x_{age} + \epsilon & \text{otherwise} \end{cases}$$

Thus, $\beta_{gender}(< 0)$ signifies how much y is less for males compared to females. The coefficient in itself has no meaning unless compared with respect to a base value.

Suppose we were to change the convention to -1 for females and 1 for males, then

$$y = \beta_{age}x_{age} + \beta_{gender}x_{gender} + \epsilon = \begin{cases} \beta_{age}x_{age} + \beta_{gender} + \epsilon & \text{if male} \\ \beta_{age}x_{age} - \beta_{gender} + \epsilon & \text{otherwise} \end{cases}$$

Now, $2\beta_{gender}$ gives us the difference between the value of y between males and females, and β_{gender} denotes the change on either side from the base value of $\beta_{age}x_{age} + \epsilon$. This new β should be have of the original coefficient because the real relation has stayed the same. Thus, the interpretation of coefficients changes based on how the variable gets defined.

In case of n levels, we will create $n-1$ variables (since the last variable is perfectly correlated with all the remaining ones). Then, the coefficient of the i^{th} level is simply the increment over the base (last) level. Note that there is no coefficient for the last level, and all the other coefficients are relative to this level.

Hierarchical Principle If we include an interaction in the model, we should include all the main effects, even if their *p-values* of those coefficients are insignificant. Simply put, if X_1X_2 appears in the model, X_1 and X_2 should also be there. Similarly, for a categorical variable, either all the categories are present in the model, or none of them are present. It is alright to just use a single category, but then the remaining categories together constitute the base cateogory and in essence, all categories are still present in the model, albiet in a different form.

Importance of interactions Suppose we build a bank balance model dependent on income, and if a person is a student or not. Basic model

$$y = \beta_{income} + x_{income} + \beta_{student}x_{student} + \epsilon$$

The problem with this model is that for both student and non student, the effect of income is same, which is not what we want. Including interaction terms,

$$y = \beta_{income} + x_{income} + \beta_{student}x_{student} + \beta_{income\times student}x_{income}x_{student} + \epsilon$$

which gives different dependence on income (slope) for student ($\beta_{income} + \beta_{income\times student}$) and non student ($\beta_{income}$).

1.3 Problems when using Linear Regression

Non linearity of data will make linear regression perform poorly as the basic assumption is that the data has linear relation with response. In case the data is non-linear, we expect to see distinct patterns in the residual vs variable plots. To induce linearity, transformations like log, *exp*, *sqrt* etc. can be checked.

Correlation of errors We do not expect ϵ_i to give any information regarding ϵ_{i+1} as that will cause the standard errors to be underestimated giving wider confidence intervals. This can be checked by plotting the residuals vs a shifted version, which should not give any discernable patterns.

Heteroscedasticity or non-constant variance in residuals also violates the assumption that the response/errors have constant variance. This can be checked by plotting the residuals vs a variable. In case the variance is not constant, we expect to see the residuals to get further away from each other as we progress along the variable x.

Outliers can cause trouble with the model as mean square error will give more weightage to larger errors. This can be checked for by plotting either the histogram of the variable to ensure thin tails, or by plotting the residual/standard error estimate, which should have no discernable pattern.

Multicollinearity occurs when one of the variable is expressible as a linear combination of a set of the remaining variables. This can cause large variance in the estimation of the coefficients. It can be checked for using *Variance Inflation Factor*

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

where $R^2_{X_j|X_{-j}}$ is the R^2 score obtained by regressing the j^{th} variable on all the remaining variables. Typically, VIF should be $\leq 3 - 5$. Higher values indicate significant correlation of this variable with the others.

Chapter 2

Classification

For more than two classes, it is hard to maintain the ordering between them using linear regression. For two classes however, linear regression can be used to prepare an ordering of the data (although difficult to interpret as probability themselves).
Classification using linear regression to predict binary reponse will be same as Linear Discriminant Analysis (LDA).

2.1 Logistic Regression

Modelling binary response with linear regression might produce values outside the range $[0, 1]$ (and possibly negative as well). Hence we use a logistic function to compress the outputs to $[0, 1]$ range.

$$p(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$
$$odds = \frac{p(Y = 1|X)}{1 - p(Y = 1|X)} = e^{\beta_0 + \beta_1 X}$$

- The solution to this model is obtained via **Maximum Likelihood Estimation**.
- Odds are also used to interpret probability. A low value of odds (close to 0) indicates a low probability while a high value (close to inf) indicates a high probability.
- One unit change in X will cause β_1 change in the *log odds*.

2.1.1 Loss Function

Maximum Likelihood is used to determine the coefficients. Basic intuition is to choose such a pair of $\beta' s$ that will make the predicted probability as close to the correct binary response (0 or 1) as possible.

$$\begin{aligned}
\text{Denoting} \quad p(x_i) &= P(Y=1|X=x_i) = 1/(1+exp(-\beta^T x_i)) \\
\text{likelihood function} = l(\beta_0,\beta_1) &= \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1-p(x_{i'})) \\
\text{Taking logarithm, } logloss &= \sum_{i:y_i=1} \log p(x_i) + \sum_{i':y_{i'}=0} \log(1-p(x_{i'})) \\
&= \sum_{i=1}^{n} y\log p(x_i) + (1-y)\log(1+p(x_i)) \qquad \text{since } y = 0 \text{ or } 1 \\
&= \sum_{i=1}^{n} y\beta^T x_i - log(1+exp(\beta^T x_i))
\end{aligned}$$

All the formulae listed here and above extend for the case of multiple variables, wherein we simply replace the sum $\beta_0 + \beta_1 X$ with $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$.

For the below derivations, refer to section 9.1 for complete matrix calculus To solve for the βs, we consider the derivate of the *log likelihood*

$$\frac{\partial l}{\partial \beta} = \sum_{i=1}^{n} x_i(y_i - p_i) = 0$$

which are $p-1$ non linear equations. We use the Newton-Raphson method and Hessian matrix for solving them

$$\begin{aligned}
\frac{\partial^2 l}{\partial\beta\partial\beta^T} &= -\sum_{i=1}^{n} x_i x_i^T p(x_i)(1-p(x_i)) \\
\beta^{new} &= \beta^{old} - \left(\frac{\partial^2 l}{\partial\beta\partial\beta^T}\right)^{-1} \frac{\partial l}{\partial \beta}
\end{aligned}$$

Converting the above equations to function form for ease of obtaining solution

$$\begin{aligned}
\frac{\partial l}{\partial \beta} &= X^T(y-\mathbf{p}) \\
\frac{\partial^2 l}{\partial\beta\partial\beta^T} &= -X^T W X \\
\text{where} \quad W &= diag([p(x_1)(1-p(x_1)),\ldots,p(x_n)(1-p(x_n))])
\end{aligned}$$

and the Newton-Raphson update becomes

$$\begin{aligned}
\beta^{new} &= \beta^{old} + (X^T W X)^{-1} X^T (y-\mathbf{p}) \\
&= (X^T W X)^{-1} X^T W (X\beta^{old} + W^{-1}(y-\mathbf{p})) \\
&= (X^T W X)^{-1} X^T W z
\end{aligned}$$

where z is called the adjusted response (target in regression). The equation is same as equation (1.1) with an added weight term of W. Hence, this equation solves the weighted least squares problem

$$\beta^{new} = \underset{\beta}{\operatorname{argmin}} (z - X\beta)^T W (z - X\beta)$$

and is known as *iteratively reweighted least squares* (IRLS). The equation converges since *log likelihood* is concave. In case of non convergence/huge jumps, reducing the step size helps.

Further, the above formulation allows us to get the distribution of β

$$\hat{\beta} \sim \mathcal{N}(\beta, (X^T W X)^{-1})$$

Wald test, rao score test

In case of multiple classes, y becomes a matrix of shape $n \times K$ and W is non-diagonal, but the solution to IRLS is not simple. Methods like co-ordinate wise gradient descent works better.

2.1.2 Multiple Classes

Note that the log of odds formula above is the ratio of probability of $Y = 1$ to probability of $Y = 0$. Here, $Y = 0$ can be seen as a reference class. Similarly, in the case of K classes, we will have $K-1$ classifiers, each classifying with respect to the last class (since the decision boundary has to be between two classes) and will take the form

$$\begin{aligned}
log(\frac{P(Y=1|X=x)}{P(Y=K|X=x)}) &= \beta_{1,0} + \beta_1^T x \\
log(\frac{P(Y=2|X=x)}{P(Y=K|X=x)}) &= \beta_{2,0} + \beta_2^T x \\
&\vdots \\
log(\frac{P(Y=K-1|X=x)}{P(Y=K|X=x)}) &= \beta_{K-1,0} + \beta_{K-1}^T x \\
\text{and} \quad \sum_{k=1}^{K} P(Y=k|X=x) &= 1 \\
\text{Giving} \quad P(Y=k|X=x) &= \frac{exp(\beta_{k,0} + \beta_k^T x)}{1 + \sum_{j=1}^{K-1} exp(\beta_{j,0} + \beta_j^T x)} \quad \text{for } j < K \\
P(Y=K|X=x) &= \frac{1}{1 + \sum_{j=1}^{K-1} exp(\beta_{j,0} + \beta_j^T x)} \\
\text{Parameter Set} \quad \theta &= \{\beta_{1,0}, \beta_1, \ldots, \beta_{K-1,0}, \beta_{K-1}\}
\end{aligned}$$

2.2 Linear Discriminant Analysis (LDA)

Why use LDA ?

- When the **classes are well separated**, the parameter estimates for the **logistic regression** model are surprisingly **unstable**. **LDA** does not suffer from this problem and is relatively **stable**.
- if n **is small** and the distribution of X **is approximately normal** in each of the classes, **LDA** is again **more stable** than logistic regression.
- LDA is popular when we have **more than two classes**.

LDA first models the distribution of X in each class, and then uses Bayes' rule to flip this and get $p(Y|X)$. When these distributions of X are normal, the model is very similar in form to logistic regression.

2.2.1 Model Derivation

Let the total number of classes be K and the prior probability that a randomly chosen observation comes from the k^{th} class be $\pi_k = P(Y = k)$. Also, let $f_k(x) = P(X = x|Y = k)$ denote the probability distribution function of X for the data points belonging to the class k. By Bayes' Rule

$$
\begin{aligned}
\pi_k &= \frac{\text{Observations in class k}}{\text{Total observations}} \\
p(Y = k|X = x) &= \frac{P(Y = k)P(X = x|Y = k)}{P(X = x)} \\
&= \frac{P(Y = k)P(X = x|Y = k)}{\sum_{l=1}^{K} P(Y = l)P(X = x|Y = l)} \\
&= \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}
\end{aligned}
$$

2.2.2 Gaussian Model with one Predictor

We assume the predictor to have a Gaussian distribution. For simplicity, also assume that the variances of X for all the K classes are also the same (fundamental assumption for linearity of decision boundary). Then,

$$
\begin{aligned}
f_k(x) &= \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu_k)^2}{2\sigma^2}} \\
p_k(x) &= \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu_k)^2}{2\sigma^2}}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu_l)^2}{2\sigma^2}}}
\end{aligned}
$$

For any given x, we notice that all $f_k(x)$'s have the same denominator. To assign a class, we just need to find the maximum value. Taking log, removing the denominator and removing the parts corresponding to x from numerator (since they are same across all classses),

$$\log p_k(x) \propto \log \pi_k + \frac{\mu_k^2}{2\sigma^2} - \frac{x\mu_k}{\sigma^2}$$

In the case of two classes, the decision boundary can be found by equating the two log *probabilities* (assume the priors to be same for simplicity)

$$
\begin{aligned}
x\frac{\mu_1}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} &= x\frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2} \\
\text{or, } x &= \frac{\mu_1 + \mu_2}{2}
\end{aligned}
$$

μ_k and σ^2 need to be estimated from the data, which can be done through the following formulae (n is total training examples and n_k is total training examples from class k)

$$
\begin{aligned}
\hat{\mu_k} &= \frac{1}{n_k} \sum_{i:y_i=k} x_i \\
\hat{\sigma^2} &= \frac{1}{N-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x - \hat{\mu_k})^2
\end{aligned}
$$

2.2.3 Multivariate Gaussian

A Multivariate Gaussian is an extension of the 1-D gaussian to multiple dimensions. Here, we assume that each of the individual dimensions is itself a Gaussian, with the different dimensions having correlation with each other, which are all specified in the correlation matrix.

$$X \sim \mathcal{N}(\mu, \Sigma)$$
$$f(x) = \frac{1}{(2\pi)^{p/2} \mid \Sigma \mid^{1/2}} \exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu))$$

Here, μ is the mean vector Σ is the covariance matrix (symmetric).
Assume μ_k represents the mean vector for individual classes and we have a common covariance matrix across all classes. Plugging this into the LDA equation and removing the common part across all classes, the discriminant becomes

$$\log p_k(x) \propto \log \pi_k + x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k$$

To calculate the decision boundary, we simply do a pairwise equality between the discriminants of the individual classes and get the pairwise decision boundaries which are all linear.

2.2.4 Quadratic Discriminant Analysis (QDA)

The assumption of same covariance matrix Σ across all classes is fundamental to LDA in order to create the linear decision boundaries.
However, in QDA, we relax this condition to allow class specific covariance matrix Σ_k. Thus, for the k^{th} class, X comes from $X \sim \mathcal{N}(\mu_k, \Sigma_k$.
Plugging this into the classification rule to get the discriminants (removing denominators as they are common for all classes)

$$\delta_k(x) = \log \pi_k - \frac{1}{2}\log \mid \Sigma \mid - \frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)$$
$$\text{Note that, } x^T \Sigma_k^{-1} \mu_k = \mu_k^T \Sigma_k^{-1} x \text{ since } \Sigma \text{ is symmetric and } x^T \Sigma_k^{-1} \mu_k \text{ is scalar}$$
$$\delta_k(x) = \log \pi_k - \frac{1}{2}\log \mid \Sigma \mid - \frac{1}{2} x^T \Sigma_k^{-1} x + x^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k$$

Notice the term $x^T \Sigma_k^{-1} x$ that gives the classifier it's quadratic form.
However, since we are calculating individual covariance matrices for all the classes, we need to calculate more parameters than before which requires more data.
The following points about QDA vs LDA must be noted

- QDA requires evaluation of substantially more parameters than LDA which subsequently means that more training data points must be available.
- QDA will be superior if the decision boundaries are not linear, i.e., LDA's assumption of equal variances for all classes will not hold true which will cause LDA to have a higher bias.
- QDA is more flexible than LDA which can reduce bias. However, bias-variance trade-off implies that variance can be relatively higher for QDA if training examples are not sufficient.

2.3 Comparison of Classifiers

Logistic Regression and LDA are similar in the sense that they produce linear decision boundaries.

- Logistic Regression estimates coefficients using Maximum Likelihood Estimate
- LDA estimates parameters using the sample mean and variance

For both the models, $\log odds$ takes a linear form. LDA adds a strong assumption of normal distribution of the predictor variables.

Comparison of models

- Logistic Regression is the simplest classifier one can build. It assumes linearly separated decision boundaries. It is usually used as a binary classifier. The decision boundary can be made non linear by adding transformed version of the predictors like second powers, interaction terms etc.
- LDA is also a linear classifier, but works under the assumptions that the decision boundaries are linear and all the classes share the same covariance matrix. It works well with multiple classes. The performance can be quite bad if the underlying variables are not normally distributed.
- QDA is a natural extension of LDA that relaxes the assumption of shared covariance matrix and allows each class to have a separate covariance matrix. This causes QDA to work well when decision boundaries have non linearity
- KNN is a non parametric model that is the most flexible. However, we can get no indication of which predictor is important, and the model can suffer from high variance.

2.4 Classfication Metrics

Several classification metrics are available for binary classifiers which are used based on the problem setting.

2.4.1 Confusion Matrix

This matrix tabulates the number of cases we are classifying and misclassifying.

Confusion Matrix	**Actual Positive**	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

Based on the above table, we define the following terms

- Accuracy $= \frac{TP+TN}{P+N}$
- Sensitivity or True Positive Rate (TPR) or Recall $= \frac{TP}{P} = \frac{TP}{TP+FN}$
- Specificity or True Negative Rate (TNR) $= \frac{TN}{N} = \frac{TN}{FP+TN}$
- Precision or Positive Predicted Value (PPV) $= \frac{TP}{FP+TP}$
- False Positive Rate $= \frac{FP}{N} = \frac{FP}{FP+TN}$
- F_1 Score $= \frac{2*precision*recall}{precision+recall}$

2.4.2 Receiver Operating Characteristics (ROC Curve)

ROC curve is plot between **True Positive Rate** and **False Positive Rate**, or equivalently, between **sensitivity/recall** and $1-$ **specificity**. The area under the plotted curve is know as AUC score.
The curve is plot by repeatedly constructing the confusion matrix at different probability thresholds (i.e. changing the decision boundary to see how the confusion matrix changes).
ROC Curve is agnostic of the class balancing in the data set, and is thus used frequently in case of class imbalance to judge a classifier. A random classifier will have AUC of 0.5 as at any threshold, the number of correctly and incorrectly classified points will roughly be the same. A perfect classifier will be able to segregate the population perfectly and will have the value of AUC as 1.0.

Chapter 3

Linear Model Selection and Regularization

Linear models are often simple and easy to interpret at the cost of having high bias if the relationship in the data is not linear. Some considerations about linear models

- If $n >> p$, least square estimates often have less variance. If n is larger than p, then least square estimates can have some variance. While if $n < p$, we are looking at non unique solutions which can cause lot of variation in the test predictions.
- It is often the case that many of the predictors do not have a relationship with the response. Hence, it is a good idea to remove those and make the model more interpretable at the cost of some bias. Least square estimates almost never give zero coefficients.

There are major ways in which the number of variables in the model can be reduced

- Selecting a **subset of variables** that go well with the response. This itself can be done by forward selection, backward elimination etc.
- **Shrinking** some of the **coefficients** to zero. This is a great help in reducing the variance of the predictions.
- **Dimension Reduction** helps in projecting the p predictors onto a M dimensional space where $M < p$. This utilizes linear combinations to create a set of new features.

3.1 Subset Selection

3.1.1 Best Subset Selection

This is a naive approach that essentially tries to find the best model among 2^p models that are trained on all possible subsets of the p variables. As we increase the subset of variables, the training error will monotonically decrease whereas the same cannot be said for the test error. A number of criteria like test MSE, R^2, AIC etc can be used to pick the models.

In case of classification models, similar argument holds and a more general error metric *deviance* can be used. *Deviance* is defined as $-2 * \log \textit{likelihood}$ of the data. The smaller the *deviance*, the better the model fit.

The huge search space presented by this approach easily overfits as the search space presents more opportunities to find better fits. However, this causes a higher variance in the predictions on future data and can possibly also have higher test error.

Note regarding dummy variables Whenever we have a group of variables that actually represent the same quantity, like a dummy variable for gender/income, it is important to keep either all of them, or none of them in the model. This is in accordance with keeping the degrees of freedom consistent. Hence, in any subset selection algorithm, all variables of the set are either considered together or not considered at all.

3.1.2 Forward Stepwise Selection

This is a greedy approach that significantly shrinks the search space being checked (in comparison to the best subset selection approach).
Forward Stepwise Selection Algorithm

1. Let M_0 denote the null model, i.e., the model with no predictors
2. For $k = 0, 1, \dots, p-1$
   (a) Consider all $p-k$ models formed by adding a single predictor to the model M_k
   (b) Select the best model M_{k+1} among the $p-k$ models on the basis of the error metric
3. From the models $M_0, M_1, \dots, M_p$, select the one with the lowest cross validation error on the evaluation choosing the appropriate error metric

This approach effectively has reduced the search space from 2^p to $1 + p(p+1)/2$. Although, now it is not guaranteed that the model selected will be the best one among 2^p.

Note that in linear regression, the forward stepwise method will start with first adding the intercept, and then the independent variables one by one.

3.1.3 Backward Stepwise Selection

This approach is the opposite of forward stepwise selection. We recursively reduce the number of variables in our model.

1. Let M_p denote the complete model, i.e., the model with all p predictors
2. For $k = p, p-1, \dots, 1$
   (a) Consider all $k-1$ models that keep all but one predictors in the current model M_k
   (b) Among these, select the best model M_{k-1} with the lowest error
3. From the models $M_0, M_1, \dots, M_p$, select the one with the lowest cross validation error on the evaluation choosing the appropriate error metric

The number of models explored is same as the forward stepwise method.
A hybrid approach is usually selected where we start with the usual forward selection method, but while adding variables, we do not add a variable if it does not give significant improvement. Another approach can be to remove redundant variables using p-test at every step of forward selection.

3.2 Metrics for evaluating Subset Models

In linear models, as we add more variables, the training error usually monotonically decreases. Test error may not behave in the same way. When training a model, the coefficients obtained are specific for minimizing the training error and hence will have less bias in comparison to the test error.
Hence, subset evaluation using training error will usually favour models with more number of variables. To overcome this

- Correct the training error estimate to correctly calculate test error
- Use a validation test or k-fold validation for better estimate of test error

3.2.1 C_p Estimate

For a least square fitted model,

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

p is the number of predictors and $\hat{\sigma}^2$ is the estimate of the error associated with each observation. This is typically evaluated using the model built on all p predictors.
Clearly, as p increases, we are penalizing the model more to compensate for the decrease in the training RSS. When $\hat{\sigma}$ is an unbiased estimate of σ, we can show that this is infact an unbiased estimate of the test MSE.

3.2.2 Akaike Information Criterion (AIC)

AIC is defined for a large class of models fit by the maximum likelihood estimate.
For least squares fit in linear models, the errors are assumed to be gaussian and thus, AIC and least squares mean the same thing. For this case

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2p\hat{\sigma}^2)$$

where we have omitted an additive constant for the sake of simplicity.
For least squares models, C_p and AIC are proportional to each other.

3.2.3 Bayesian Information Criterion (BIC)

BIC is derived from a Bayesian point of view, but ends up looking similar to the above defined errors.
For least squares error without constants, the BIC is

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

Note that the $\log(n)$ term will put a heavier weight on the error term for large p. Hence, BIC will tend to select models with lower number of variables in comparison to say C_p.

3.2.4 Adjusted R^2

Recall that R^2 is defined as $1 - RSS/TSS$. Adjusted R^2 is

$$Adjusted\ R^2 = 1 - \frac{\frac{RSS}{n-p-1}}{\frac{TSS}{n-1}}$$

This adjusted R^2 might increase or decrease when adding variables due to the terms corresponding to p. The intuition is that, after the correct number of variables have been identified, the decrease in RSS is less in comparison to the decrease in $n-p-1$ which will slightly increase the Adjusted R^2.

3.3 Shrinkage Methods

Instead of using a subset of predictors, we can also use all of the predictors and shrink the coefficients towards zero. This approach significantly reduces the variance in the model estimates as the subset selection methods often suffer from high variance. The famous ones here are *Ridge Regression* and *Lasso Regression.*

3.3.1 Ridge Regression

Ridge Regression is very similar to the least square estimate for linear regression except that we add a term corresponding to the squared sum of the regression coefficients in the error.

$$\begin{aligned} error &= RSS + \lambda \sum_{j=1} p\beta_j^2 \\ &= (Y - X\beta)^T(Y - X\beta) + \lambda\beta^T\beta \end{aligned}$$

λ is a tuning parameter that needs to be chosen separately. It acts as a weight between the error in the data and how large are the regression coefficients. It is also known as the shrinkage penalty.
Note that we will not include the intercept term in shrinkage because it is simply the mean estimate of the model when all the predictors are zero and may not necessarily zero. Thus the above model includes p terms in β. When the model inputs are all centered (which is always preferred), the intercept can be calculated in the end as simply $\sum_{i=1}^n y_i/n$

Using least squares estimate,

$$\begin{aligned} error &= (Y - X\beta)^T(Y - X\beta) + \lambda\beta^T\beta \\ \frac{\partial error}{\partial \beta} &= 0 \\ \implies 0 &= -Y^TX - Y^TX + \beta^TX^TX + \beta^TX^TX + \lambda\beta^T + \lambda\beta^T \\ \beta^T(X^TX + \lambda I) &= Y^TX \\ (X^TX + \lambda I)^T\beta &= X^TY \\ (X^TX + \lambda I)\beta &= X^TY \\ &\boxed{\beta = (X^TX + \lambda I)^{-1}X^TY} \end{aligned}$$

$\lambda = 0$ will result in the simple least squares regression while $\lambda \to \inf$ will force the coefficients to go towards zero.

As is clear from the formula, the error term is sensitive to the actual scale of the coefficients which is ultimately dependent on the predictors themselves. In a simple least squares regression, the coefficients will scale up and down depending on how the data is scaled. The same is not true for Ridge Regression.
Hence when using **Ridge Regression, it is always advisable to *standardize* the predic-**

tors before training the model using

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

The success of Ridge Regression is based in the **bias variance tradeoff**. If the data is linear, simple linear regression will have a very low bias but high variance, making it sensitive to the training data. As λ is introduced, it forces the model to have less flexibility by reducing the coefficiets value and subsequently their power on the prediction. This causes a reduction in the variance at expense of slight increase in bias. However, this trend is not monotonic with increasing λ and the appropriate value must be chosen based on the errors observed.
Ridge regression will tend to give similar coefficient values for correlated variables.

3.3.2 Lasso Regression

Notice that Ridge Regression will try to reduce the value of some of the coefficients, but it will never set them to exactly zero. Hence, we will end up using all the p predictors in the model which may not be interpretable if the value of p is large.

Lasso Regression comes over this disadvantage by defining the error as

$$error = RSS + \lambda \sum_{j=1}^{p} \mid \lambda_j \mid$$

which does not include the intercept. When λ **is sufficiently large, Lasso Regression forces some of the variables to be exactly zero**. This is very useful in reducing the subset of variables that we use in the model, thereby increasing model interpretability.
Lasso Regression can give different coefficient values to correlated variables.
Similar to Ridge Regression, if we have standardized the input variables, the intercept is simply the average of the y's and can be computed in the end after obtaining the other coefficients.

3.3.3 Alternative Formulation to Ridge and Lasso Regression

These regressions can also be considered as solving a constrained optimisation problem

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

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

for Ridge and Lasso regression respectively. This holds true because these regressions are effectively trying to limit the size of the coefficients themselves. For $\lambda = 0$, we have no bound on the size and s in equations above is close to inf. As λ increases, s will start to decrease and be 0 for $\lambda \rightarrow$ inf.

The equations above can be interpreted as finding the minimum RSS among the points that lie inside the geometric shapes defined by the constraints. For $p = 2$, Ridge defines a circle $\beta_1^2 + \beta_2^2 \leq s$ and Lasso defines a diamond $\mid \beta_1 \mid + \mid \beta_2 \mid \leq s$.

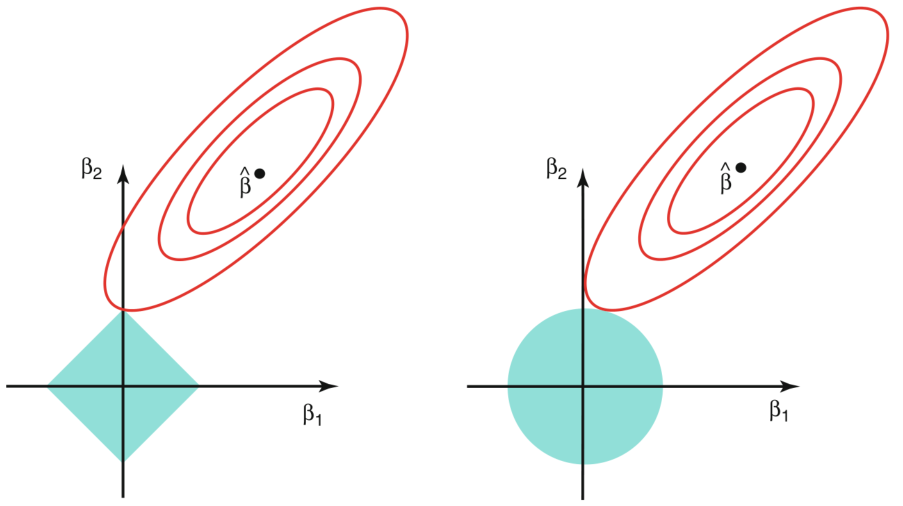

Figure 3.1: For $p = 2$, the left and right images correspond to Lasso and Ridge Regression.

3.3.4 Variable Selection Property of Lasso Regression

In the figure 3.1, $\hat{\beta}$ corresponds to the least squares estimate of β and the contours in red colour show the same value of RSS. The blue coloured regions correspond to the constraints defined above (diamond for lasso and circle for ridge).
Clearly, circle being a smooth shape, the lowest RSS contour will not usually touch it at one of the axis points. However, for the sharp diamond shape, the least RSS value is likely to be encountered along the axis. The shapes of the constraint regions can be controlled through λ and for lasso, more constrained models (small s or higher λ) will cause more of the coefficients to be zero. The argmuents are very well valid in higher dimensions as well.

3.3.5 Bayesian Interpretation

Lasso and Ridge Regression are also natural solutions when the the coefficients are assumed to have certain specific priors.

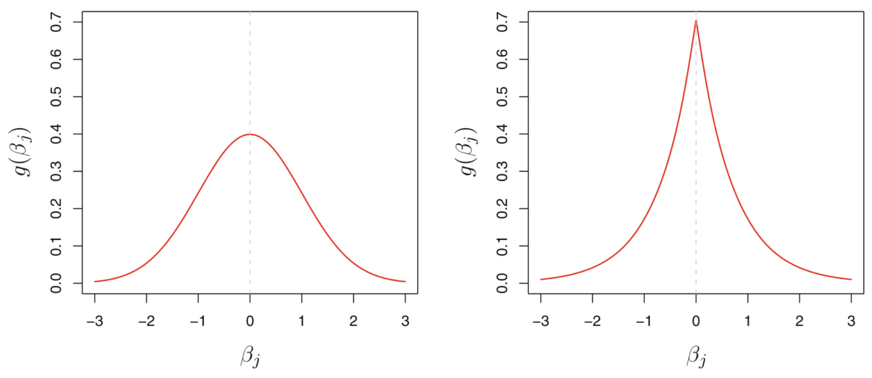

Figure 3.2: Gaussian prior on left and double exponential on the right

$$p(\beta|X,Y) \propto f(Y|X,\beta)p(\beta|X) = f(Y|X,\beta)p(\beta) \qquad \text{assuming X is constant}$$
$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$
$$\text{Assume, } p(\beta) = \prod_{j=1}^{p} \beta_p \qquad \text{for some density function } g$$

The following are observed for different priors on β

- When the density function of β_j is assumed to be a standard normal, the posterior is same as solving the ridge regression error function
- When the density function is assumed to be a double exponential, the posterior is the same as solving lasso error function

From the visuals of the priors in figure 3.2, double exponential is steeply peaked at zero, which clearly implies that the prior itself assumes some of the coefficients are likely zero. On the other hand, the gaussian priod is flatter and does not necessarily require the coefficients to be zero.

3.3.6 Elastic Net

Both Lasso and Ridge regression offer their own set of advantages. Ridge helps shrink down coefficients and variance, while lasso helps in variable selection. Elastic Net aims to combine both of these together to do both shrinkage and variable selection. The loss function for Elastic Net is

$$error = (Y - X\beta)^T(Y - X\beta) + \lambda(\alpha\beta^T\beta + (1-\alpha)\sum_{j=1}^{p} \mid \beta_j \mid)$$
$$\text{or,} \quad = RSS + \lambda \sum_{i=1}^{p} (\alpha\beta_j^2 + (1-\alpha) \mid \beta_j \mid)$$

Based on the Bayesian interpretation, one can expect that Elastic Net has the priors distributed in the form $\mid \beta \mid^q, q \in (1,2)$. However, this formulation gives us rounded edges along the axis (as opposed to corners in case of lasso or $q = 1$). Elastic Net slightly modifies this to have similar shape to $\mid \beta \mid^q, q \in (1,2)$ but with sharp corners which allows enforcing some coefficients to zero.

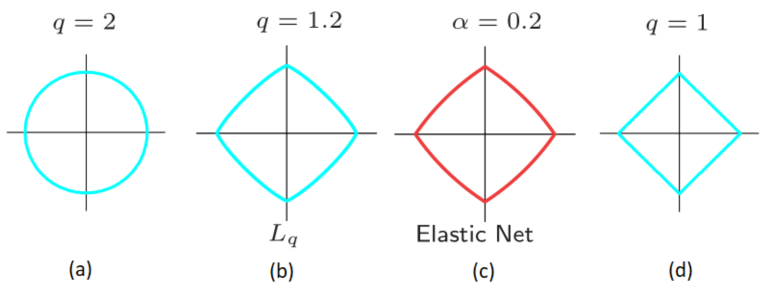

Figure 3.3: Contours of $\sum_p \mid \beta_j \mid^q$ in case of (a) Ridge ($q = 2$) (b) $q = 1.2$ (c) Elastic Net ($\alpha = 0.2$) (d) Lasso $q = 1$

3.4 Dimension Reduction Methods

The methods seen above achieve reduction in variance by either reducing the number of variables being used for estimation, or reduce the value of coefficients of those variables.
Dimensionality reduction uses a **linear map** to convert the original p variables to M variables where $M < p$.

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j \qquad \text{where } \phi_{jm} \text{ are constants}$$

$$RSS = \sum_{i=1}^{n} (y_i - (\theta_0 + \sum_{m=1}^{M} \theta_m Z_m))^2 \qquad \text{where } \theta_m \text{ are linear regression coefficients for } Z_m$$

This new formulation restructures the original least squares formula. The whole process now is

- Obtain the linear map using techniques like PCA
- Use least squares on the transformed predictors to obtain regression estimates

3.4.1 Principal Components Analysis (PCA)

PCA is one of the most common methods used for dimensionality reduction. It is based on the principle of finding those directions that **maximize the variance of data**. The intuition behind finding this direction is that this separates the data best given the high variance. Hence, the performance of a classifier/regressor would be better if the training was done using the transformed predictors.

Assume that we want to find M components for PCA, where $M <= p$. Then,

1. Find the direction/projection of the data that gives the maximum variance under the constraint $\sum_{j=1}^{p} \phi_{jm}^2 = 1$ (normalized vector) for obtaining the m^{th} transformed predictor
2. If the total components found is $< M$, repeat step 1 with the added constraint that the next component has zero correlation with the previous component

This constraint of finding components with zero correlation essentially means that in the multidimensional space, we are finding a set of **orthogonal vectors**. The order of choosing the new predictors is in decreasing order of the information they contain. Thus, the first predictor will now contain the most information.

Derivation of PCA It is a good idea to **standardize/scale the variables before running PCA**. Not only is it easier to derive the components, but they are also unaffected by the scale of the individual variables (since we know that scaling a variable multiplies the variance by square of the scaler and PCA is all about finding maximum variances !). If the variables represent same/similar quantities with similar scales, scaling is not mandadtory. However, we usually have data consisting of several quantities measured on different scales and thus scaling comes in handy. Scaling can be done to get the data between $0to1$ or by divinding by σ to get similar scale.

After setting the the mean of all predictors to zero and standard deviation to one, the components can be derived by considering the following optimization problem

$$\underset{\phi_{11},\dots,\phi_{p1}}{\text{maximize}}\{\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p}\phi_{j1}x_{ij})^2\} \text{ subject to } \sum_{j=1}^{p}\phi_{j1}^2 = 1$$

$$\text{Or, } \underset{\Phi_1}{\text{maximize}}\,\frac{1}{n}Z_1^T Z_1 \text{ subject to } \sum_{j=1}^{p}\phi_{j1}^2 = 1$$

$$\text{Where } Z_1 = X\Phi_1$$

$$\text{Define the Lagrangian } L(\Phi_1, \lambda) = \frac{1}{n}(\Phi_1^T X^T X\Phi_1) - \lambda(\Phi_1^T\Phi_1 - 1)$$

$$\frac{\partial L(\Phi_1,\lambda)}{\partial\lambda} = 0, \text{ and } \frac{\partial L(\Phi_1,\lambda)}{\partial\Phi_1} = 0$$

$$\text{Giving, } 2\frac{1}{n}(\Phi_1^T X^T X) - 2\lambda\Phi_1^T = 0 \text{ and } \Phi_1^T\Phi_1 - 1 = 0$$

$$\frac{X^T X}{n}\Phi_1 = \lambda\Phi_1$$

$$\text{Since all } X_i \text{ are centered, } \frac{X^T X}{n} \text{ is the covariance matrix } \Sigma_X$$

$$(= E[(X - E[X])^T(X - E[X])] \text{since we want } p \times p \text{ dimension})$$

$$\Sigma_X\Phi_1 = \lambda\Phi_1 \text{ with } \Phi_1^T\Phi_1 = 1$$

Which is nothing but the **Eigenvectors of the Covariance Matrix of X**. Notice that the maximization is achieved through the eigenvector with the highest eigenvalue. Since Σ_X is positive semi-definite, all the eigenvalues will be ≥ 0.
Furthermore, we know that all eigenvectors are orthogonal to each other, and hence they will also satisfy the condition that all the linear mapping vectors are not correlated to each other.
If we try to find the coefficients for Z_2, we are looking at the exact same optimization, but with the additional constraint that Φ_2 is not correlated to Φ_1. This will yield the second eigenvector of the covariance matrix.
Hence, **the linear maps for obtaining the PCA transformations are nothing but the eigenvectors of the covariance matrix Σ_X of the original data X**. We have a total of p eigenvectors and thus, the maximum components obtainable is also p.

Explained Variance The variance explained by the m^{th} component is nothing but the ratio of variance of Z_m and total variance of the data. Mathematically this is

$$\begin{aligned}\text{Explained variance of component } m &= \frac{\frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p}\phi_{jm}x_{ij})^2}{\frac{1}{n}\sum_{i=1} n\sum_{j=1} p x_{ij}^2} \\ &= \frac{\frac{1}{n}Z_m^T Z_m}{tr(\frac{X^T X}{n})} \\ &= \frac{\Phi_m^T\frac{X^T X}{n}\Phi_m}{tr(\Sigma_X)}\end{aligned}$$

However, note that in the derivation, Φ_m is the eigenvector and thus, $(X^T X)\Phi_m = \lambda_m \Phi_m$ and $\Phi_m^T \Phi_m = 1$. Substituiting in the above equation,

$$\text{Explained variance } = \frac{\lambda_m}{tr(\Sigma_X)}$$

$$\text{or, } \boxed{\text{Explained variance } = \frac{\lambda_m}{\lambda_1 + \cdots + \lambda_p}}$$

Where the last equation comes from the fact that the trace of a matrix is simply the sum of it's eigenvalues.
A **Scree Plot** is a plot between the explained variance and the index of the prinicple components. It's cumulative version can be used to determine the number of components to keep on the basis of how much of the total variance we want to explain. The elbow point of the Scree Plot can also help determine the components at which the explained variance drops significantly.

Principal Components Regression (PCR) is simply using some m out of M components to perform linear regression. This approach will typically work best when a few components of the PCA sufficiently explain the whole data. This way we reduce the variance at the cost of slight change in bias. PCR can also be viewed as a continuous version of Ridge Regression.

3.4.2 Partial Least Squares

This method is closely related to PCA/PCR. In the previous methods, an unsupervised approach was used to project the matrix M onto a lower dimensional space. This transformation did not take Y, the response, into consideration. PLS will incorporate Y as well for finding the transformations.
The algorithm is as follows

1. Calculate the coefficients for the first component as the coefficient obtained by regressing Y on X_j
2. Using this component, regress X_j's on Z_1 and get the residuals. These are the unexplained components of the variables.
3. Calculate Z_2 using Y and these residuals
4. Repeate the process above till M components are obtained

Though this procedure looks slightly more involved, it performs not better than lasso regression and PCR in practice. Although this method reduces bias, it also leads to quite an increase in variance as well, making the method not very useful.

3.5 Curse of Dimensionality

Most of the methods discuss here **work well when** $n >> p$. There can be many reasons why the model may not perform well in higher dimensions, but the major one will be the fact that as more and more dimensions are added to the model, chances of overfitting increase and so do the chances that the additional variables are simply noise.
We refer to the problem of training a model a high dimensional problem when $p > n$, or we have more data than number of predictors. Note that it is easy to obtain $R^2 = 1$ in such a setting which consequently leads to $\hat{\sigma}^2 \approx 0$. Metrics like C_p, AIC, BIC become useless.
Hence, in higher dimensional settings, it is important to obtain model performance on unseen data as there is a good chance of obtaining perfect results on the training set. Metrics associated with the training set can thus prove to be misleading.

Intuition for the curse of dimensionality can be easily obtained in the context of KNN. Suppose the data is uniformly distributed along any dimension considered. Let the model be built in such a way that when making predictions, it uses 10% of the data along all dimensions. In the case of a single dimension, we need 10% of the data. In the case of 10 dimensions, we will need to check $10\%^{10} = 10^{-8}\%$ of data. Clearly this number grows as we consider more and more dimensions.
Thus, as the number of dimensions grow, we require substantially more data as the data observable in the neighborhood of the point is extremely small. This illustrates the fact that in higher dimensions, having less data will lead to a poor model.

3.6 Exercises

3.6.1 Questions

1. We perform the best subset, forward stepwise and backward stepwise on a single data set and obtain p+1 models containing $0, 1, \ldots, p$ predictors.
   (a) Which of the three models with k predictors has the smalles training RSS ?
   (b) Which of the three models with k predictors has the smallest test RSS ?
   (c) True or False
      i. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the $(k + 1)$-variable model identified by forward stepwise selection.
      ii. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the $(k + 1)$-variable model identified by backward stepwise selection.
      iii. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by forward stepwise selection.
      iv. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the $(k + 1)$-variable model identified by backward stepwise selection.
      v. The predictors in the k-variable model identified by best subset are a subset of the predictors in the $(k + 1)$-variable model identified by best subset selection.
2. For parts (a) through (c), indicate the correct choice and explain the answer.
   (a) The lasso, relative to least square is
      i. More flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrese in variance.
      ii. More flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.
      iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.
      iv. Less flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.
   (b) Repeat (a) for ridge regression relative to least squares.
   (c) Repeat (b) for non-linear methods relative to least squares.

3. Suppose we estimate regression coefficients in a linear model by minimizing

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

for a particular value of s. For the following parts, indicate which of the options is correct and explain.

   (a) As we increase s from 0, the training RSS will

      i. Increase initially, and then eventually start decreasing in an inverted U shape.

      ii. Decrease initially, and then eventually start increasing in a U shape.

      iii. Steadily increase.

      iv. Steadily decrease.

      v. Remain constant.

   (b) Repeat (a) for test RSS.

   (c) Repeat (a) for variance.

   (d) Repeat (a) for (squared) bias.

   (e) Repeat (a) for the irreducible error.

4. Suppose we estimate regression coefficients in a linear model by minimizing

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

for a particular value of λ. For the following parts, indicate which of the options is correct and explain.

   (a) As we increase s from 0, the training RSS will

      i. Increase initially, and then eventually start decreasing in an inverted U shape.

      ii. Decrease initially, and then eventually start increasing in a U shape.

      iii. Steadily increase.

      iv. Steadily decrease.

      v. Remain constant.

   (b) Repeat (a) for test RSS.

   (c) Repeat (a) for variance.

   (d) Repeat (a) for (squared) bias.

   (e) Repeat (a) for the irreducible error.

5. It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will now explore this property in a very simple setting.
Suppose that $n = 2$, $p = 2$, $x_{11} = x_{12}$, $x_{21} = x_{22}$. Furthermore, suppose that $y_1 + y_2 = 0$ and $x_{11} + x_{21} = 0$ and $x_{12} + x_{22} = 0$, so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero: $\hat{\beta}_0 = 0$.

   (a) Write out the ridge regression optimization problem in this setting.

   (b) Argue that in this setting, the ridge coefficient estimates satisfy $\hat{\beta}_1 = \hat{\beta}_2$.

   (c) Write out the lasso optimization problem in this setting.

(d) Argue that in this setting, the lasso coefficients $\hat{\beta_1}$ and $\hat{\beta_1}$ are not unique—in other words, there are many possible solutions to the optimization problem in (c). Describe these solutions.

6. Using the bayesian formulation given below, derive the ridge and lasso formulations

$$p(\beta|X,Y) \propto f(Y|\beta,X)p(\beta|X) = f(Y|\beta,X)p(\beta)$$

(a) Suppose $y_i = \beta_0 + x_i^T\beta + \epsilon_i$ where ϵ_i is normal with mean 0 and variance σ^2. Calculate the likelihood of the data.

(b) Assume the following prior for β: $\beta_1, \ldots, \beta_p$ are independent identically distributed with the distribution $p(\beta) = \frac{1}{2b}\exp(- \mid \beta \mid /b)$. Write the posterior for b in this setting.

(c) Argue that the lasso estimate is the *mode* of the posterior on β.

(d) Now assume that the β are independently and identically distributed with the mean 0 and variance c. Find the posterior of the β in this case.

(e) Argue that the ridge regression estimate is both the mode and mean for β under this distribution.

3.6.2 Answers

1. (a) Best subset model will have the lowest training RSS as it looks over all possible subsets of models.

   (b) Best subset model will typically overfit and thus forward or backward stepwise models will have lower test RSS.

   (c) i. True as we add variables to the last model.

   ii. True as we remove variables from the last model.

   iii. False as this is not necessary. The paths chosen are independent.

   iv. False.

   v. False as the set of variables can be completely independent.

2. (a) iii as lasso is less flexible (some coefficients are zero) which decreases variance with an increase in bias.

   (b) iii as it is similar to lasso in terms of purpose of use.

   (c) ii as non-linear models make less assumptions and are more flexible. This reduces bias at expense of variance.

3. (a) Refer to 3.1. As we increase s, we are expanding the region of allowed β. Hence, we are moving closer to the least squares fit which gives minimum training RSS.

   (b) We have high bias with a constant model. As s increases, the test RSS initially decreases and then begins into increase as we keep relaxing the model, giving a U shape.

   (c) Constant model has zero variance. It increases as we relax the model (increase s).

   (d) By bias variance tradeoff, bias reduces as s increases.

   (e) It is always constant.

4. (a) As λ increases, the model becomes more constrained and training RSS increases.

   (b) At λ the model has minimum training RSS and high test RSS. As λ increases, test RSS initially decreases until it is high for large λ.

(c) As the model becomes less and less flexible to a constant model, the variance keeps decreasing.

(d) By bias variance tradeoff, bias will keep increasing (it's minimum for least squares fit).

(e) By definition, it remains constant.

5. (a) The data points after substituitions become $((x_{11}, x_{11}), y_1)$ and $((-x_{11}, -x_{11}), -y_1)$. The optimization functhion then becomes

$$\begin{aligned} RSS &= \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \\ &= 2(y_1 - x_{11}(\beta_1 + \beta_2))^2 + \lambda(\beta_1^2 + \beta_1^2) \end{aligned}$$

(b) Taking partial derivates with β_1 and β_2,

$$\begin{aligned} 4(y_1 - x_{11}(\beta_1 + \beta_2))(-x_{11}) + 2\lambda\beta_1 = 0 \\ 4(y_1 - x_{11}(\beta_1 + \beta_2))(-x_{11}) + 2\lambda\beta_2 = 0 \end{aligned}$$

which gives $\beta_1 = \beta_2$.

(c) In a similar fashion to above, the final optimization is

$$RSS = 2(y_1 - x_{11}(\beta_1 + \beta_2))^2 + \lambda(\mid \beta_1 \mid + \mid \beta_1 \mid)$$

(d) Taking partial derivates with β_1 and β_2,

$$\begin{aligned} 4(y_1 - x_{11}(\beta_1 + \beta_2))(-x_{11}) + 2\lambda\frac{\mid \beta_1 \mid}{\beta_1} = 0 \\ 4(y_1 - x_{11}(\beta_1 + \beta_2))(-x_{11}) + 2\lambda\frac{\mid \beta_2 \mid}{\beta_2} = 0 \end{aligned}$$

which gives $\frac{|\beta_1|}{\beta_1} = \frac{|\beta_2|}{\beta_2}$, clearly not a unique solution.

6. (a) For any y_i, the distribution given X and β is normal since y_i = constant + Normal. The constant is $\beta_0 + x_i\beta$ and the variance is σ^2.

$$f(Y|X,\beta) = \prod_{i=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(y_i - \beta_0 - x_i\beta)^2}{2\sigma^2})$$

(b) The prior for β will simply be the product of all the β_i. The posterior thus becomes

$$\begin{aligned} p(\beta|X,Y) &= f(Y|X,\beta)\prod_{j=1}^{p}\frac{1}{2b}exp-\frac{\mid \beta_j \mid}{b} \\ \ln(p(\beta|X,Y)) &= \ln((\frac{1}{\sqrt{2\pi\sigma^2}})^n(\frac{1}{2b})^p) + \sum_{i=1}^{n}-\frac{(y_i - \beta_0 - x_i\beta)^2}{2\sigma^2} + \sum_{j=1}^{p}-\frac{\mid \beta_j \mid}{b} \end{aligned}$$

Removing constants and adjusting (includling -1), maximizing above equation is same as minimizing

$$\sum_{i=1}^{n}(y_i - \beta_0 - x_i\beta)^2 + \frac{2\sigma^2}{b}\sum_{j=1}^{p}\mid \beta_j \mid$$

which is the Lasso regression optimization function with $\lambda = 2\sigma^2/b$.

(c) β obtained as a result of maximizing the posterior $p(\beta|X, Y)$ is the *mode* of that distribution and also happens to be the solution of the lasso regressin optimization function.

(d) The posterior is similar in form to the one derived above with minor changes

$$p(\beta|X, Y) = f(Y|X, \beta) \prod_{j=1}^{p} \frac{1}{2c^2} exp - \frac{\beta_j^2}{2c^2}$$

$$\ln(p(\beta|X, Y)) = \ln((\frac{1}{\sqrt{2\pi\sigma^2}})^n (\frac{1}{2c^2})^p) + \sum_{i=1}^{n} - \frac{(y_i - \beta_0 - x_i\beta)^2}{2\sigma^2} + \sum_{j=1}^{p} - \frac{\beta_j^2}{2c^2}$$

(e) Maximizing the above posterior is same as minimizing

$$\sum_{i=1}^{n} (y_i - \beta_0 - x_i\beta)^2 + \frac{2\sigma^2}{b} \sum_{j=1}^{p} \beta_j^2$$

which is a quadratic. The log of posterior is itself a parabola. The β that maximizes the posterior is not only the mode of that distribution, but also the mean (due to the symmetry). And coincidentally, it will also minimize the ridge regression optimization function given above.

Chapter 4

Moving Beyond Linearity

In this section, we explore some modifications to the linear regression model in order to incorportate some non linearity as well for reducing bias/

4.1 Polynomial Regression

This is a simply modification to the original linear regression setting where we incorporate higher powers of the predictor as well. This helps incorporate non linear trends in the data as well while retaining an additive relationship between the variables.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x \qquad \text{Linear Regression}$$
$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \cdots + \hat{\beta}_d x^d \qquad \text{Polynomial Regression}$$

Generally, is is **unusual to have** d **to be greater than** 3 **or** 4 because higher degree of polynomial makes the model highly flexible with very curvy shapes. This may reduce bias to a large extent but the variance obtained thus is quite high, expecially near the end values of the range.

$$\begin{aligned} \hat{C} &= Cov(\hat{\beta}, \hat{\beta}) \\ l_i &= (1, x_i, x_i^2, \ldots, x_i^d)^T \\ Var[\hat{y}_i] &= l_i^T \hat{C} l_i \end{aligned}$$

The same form of polynomial regression terms works in the context of classification as well for Logistic Regression.

4.2 Step Function

Polynomial Regression, like Linear Regression, still imposes a global structure on the data. The functional form of the model is same throughout. Step Function on the other hand has a local structure and divided the range into k **intervals such that each interval is fitted with a**

different constant.

$$\begin{aligned}
C_0(X) &= I(X < c_1) \\
C_1(X) &= I(c_1 \leq X < c_2) \\
C_2(X) &= I(c_2 \leq X < c_3) \\
&\dots \\
C_{k-1}(X) &= I(c_{k-1} \leq X < c_k) \\
C_k(X) &= I(c_k \geq X)
\end{aligned}$$

$$\hat{y} = \beta_0 + \beta_1 C_1(x) + \cdots + \beta_k C_k(x)$$

Note that, β_0 and $C_0(x)$ are equivalent since both will act as the intercept and the mean value of y in the range $x < c_1$. Similarly, each of the β_i captures the average of the response in the corresponding interval defined by the indicator function.
This approach works well when there are natural breakpoints in the data and if there are indeed constant trends in those intervals. As soon as a non constant trend emerges locally, the approach fails.

4.3 Basis Function

Basis function is the generalized form of the above two approaches. Basis are functions on X that transform it. Thus, when doing regression with basis functions, we simply regress y with a family of transformations of x.
For polynomial regression, the basis are powers of x and for step function, the basis are constant values in different ranges.

4.4 Regression splines (Polynomials)

This is a class of methods that extends upon the family of polynomial and step regressions.

Instead of fitting a global polynomial to X, we fit **local piecewise polynomials** to X such that they are continuous at the breakpoints. The points wherer the polynomials change are called **knots**.

As we increase the number of knots, we can get a more flexible predictor. In general, if there are K knots, we have $K + 1$ polynomials.

Having this many splines creates a lot of degrees of freedom in the data. To solve these many equations, we need to impose constraints of **continuity and continuity of derivatives at knots**. In general, for a d degree spline, we will impose the function to be continuous at knots and also have the $d-1$ derivatives to be equal at the knots. The reduced degrees of freedom are essential for obtaining unique solutions to the coefficients such that to overall curve still looks continuous.

Instead of fitting multiple splines, we can also defined the function in a single format as follows. A truncated power basis function is defined as

$$h(x, \xi) = (x - \xi)_+^d = \begin{cases} (x - \xi)^d & \text{if } x > \xi \\ 0 & 0 \text{ otherwise} \end{cases}$$

$$\hat{y} = \beta_0 + \beta_1 b_1(x) + \beta_2 b_2(x) + \cdots + \beta_{K+d} b_{K+d}(x)$$

where the terms $b_1, \ldots, b_d$ are the usual terms $x, x^2, \ldots, x^d$ and the remaining terms correspond to a truncated power function per knot. This ensures that the polynomials are continuous at the knots upto the $d-1$ derivates.

A **natural spline** is a regression spline that has the additional constraint of being linear at the boundaries, i.e., outside the extreme knots. This reduced flexibility allows for stable estimates at the knots and also makes the confidence bands narrower.

4.4.1 Choosing the degrees of freedom

While it may make sense to put knots at the points where the function seems to change rapidly (and not at the points where the function seems to be relatively stable), in practice the knots are usually placed at equal quantiles of the data, or generally are placed in an equidistant fashion. For deciding the degree of freedoms, another popular choice is cross validation. We always use say 10% of the data as test and gauge the RSS on this set across different degrees of freedom. A plot of RSS vs. the degrees of freedom (similar to elbow curve) can aid choosing the desired point where the RSS significantly drops.
In practice, cubic splines are popular as higher order polynomials tend to give high variance, and the plots with cubic splines also look relatively stable.

4.5 Smoothing Splines

The idea is to fit a smooth function that predicts the reponse well. We minimize the following

$$Loss = \sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt \qquad \lambda > 0$$

which can be decomposed into two parts, the first part encourages the function g to fit the data well and the second part encourages the function to be smooth throughout.
If the function is constant or straight line, the double derivative is zero while on the other extreme, if the function changes rapidly, double derivative will also take large values.

When λ is zero, the function simply takes the values of y and thus becomes very jumpy (introducing high variance). On the other extreme, if $\lambda \to \inf$, the function is forced to be linear (which makes the double derivative zero). This is exactly the least square fit that we have seen earlier.

The function that minimizes the above loss is nothing but a cubic spline with knots at $x_1, x_2, \ldots, x_n$ with the additional constraint that the first and second derivatives are smooth at the knots. (The function is cubic in between the points). Outside the extreme knots, the function simply takes a linear form.
In some sense, this is similar to the natural splines but is a shrinked version of the same, λ controlling the level of shrinkage (or how large the roughness can be).

$$\hat{g}_\lambda = S_\lambda y$$
$$df_\lambda = \sum_{i=1}^{n}\{S_\lambda\}_{ii}$$

where $\hat{g}_\lambda$ is a n-vector containing the values at the points x_is as a solution to a particular value of λ. S_λ is a $n \times n$ matrix obtained as the solution of the above error function.

4.5.1 Choosing Reguularization Parameter

It is possible to show that as λ increases from $0 \to \inf$, the effective degrees of freedom (df_λ) go from $n \to 2$.

Cross validation can also be a useful approach to determine the value of λ. However, similar to linear regression, there exists a formula for computing the loss for LOOCV by just fitting a single model to the entire data set.

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

where $\hat{g}_\lambda^{(-i)}(x_i)$ denotes the estimate at any x_i without using the point i for calculating the loss (leave one out), while $\hat{g}_\lambda(x_i)$ denotes the estimate at x_i using all of the data.

4.6 Local Regression

As the name suggests, we fit regression models in the neighborhood of the point for which we want to make a prediction. But the difference from a normal linear regression is that we give weights to the points such that the points closest get higher weights and points further away get zero.

Suppose we want to make the predicton at the point x_0. The algorithm is as follows

1. gather the fraction $s = k/n$ points closest to x_0 where s is also called the span an acts like a regularizer. Higher s will make a global fit and vice versa.
2. Assign a weight $K_{i0} = K(x_i, x_0)$ to all the points in the neighborhood of x_0 such that the point closest gets the highest weight while the one furthest away gets the weight zero. All the other points in the data get the weight zero.
3. Minimize the error given by

   $$RSS = \sum_{i=1}^{n} K(x_i, x_0)(y_i - \beta_0 - \beta_1 x_i)^2$$

   to estimate the values of β_0 and β_1. Note that K is a function known beforehand and thus constant.
4. Make the prediction as $\hat{y}_0 = \beta_0 + \beta_1 x_0$

These models are very useful for adapting to changes in the neighborhood of a given point. The distance function can take several forms like constant, linear, quadratic or even normal distribution.
The approach can be extended to multiple dimensions as well by defining corresponding distances. However, the approach performs **poorly in dimensions higher than** 3 **or** 4 as it becomes difficult to find neighbors in the surrounding area (dimensionality curse).

4.7 Generalized Additive Models

The approches discussed above are extensions of the linear regression model for a single predictor by introducing more flexbility into the models. This idea can be extended for p predictors in the framework of *Generalized Additive Models*. These are applicable for both classification and regression.

$$\hat{y}_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \qquad \text{Linear Rgerssion}$$

$$\hat{y}_i = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) \qquad \text{Generalized Additive Models}$$

where $f_i(x)$ are non-linear smooth functions applied to each of the predictors separately.

Each of the functions above can be fit using all of the sections defined above. For example, some of the predictors can be smooth splines, some can be just dummy variables, and some can be polynomials. Thus, we expand from p predictors to a multitude of them, where we can choose a different expansion method for each of them individually. We have combined individual simple linear regression models into a general framework for p predictors.

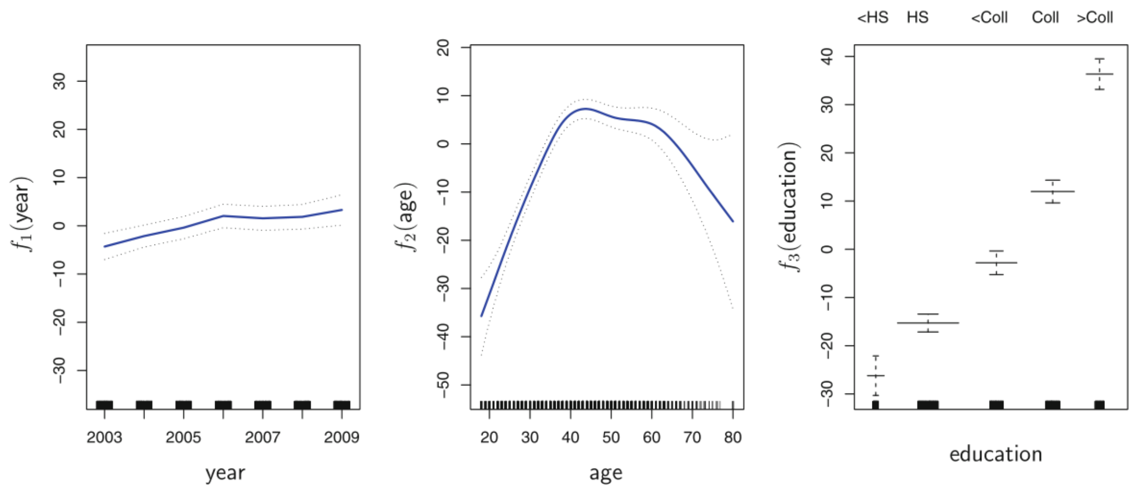

Figure 4.1: Separate non-linear functions for three different variables. y-axis is the response. Left two plots are amoothing splines with different degrees of freedom. Right plot is dummy variables.

Fitting smoothing splines is not trivial as the loss function is not simple least squares. However, softwares can still fit using *partial residuals*. We repeatedly update the fit for a single predictor, keeping the others constant.
Summarizing,

- GAMs allow us to fit a non-linear f_j to each X_j, so that we can automatically model non-linear relationships that standard linear regression will miss. This means that we do not need to manually try out many different transformations on each variable individually.
- The non-linear fits can potentially make more accurate predictions for the response Y .
- Because the model is additive, we can still examine the effect of each X_j on Y individually while holding all of the other variables fixed. Hence if we are interested in inference, GAMs provide a useful representation.
- The smoothness of the function fj for the variable Xj can be summarized via degrees of freedom.
- The main limitation of GAMs is that the model is restricted to be additive. With many variables, important interactions can be missed. However, as with linear regression, we

can manually add interaction terms to the GAM model by including additional predictors of the form $X_j \times X_k$. In addition we can add low-dimensional interaction functions of the form $f_{jk}(X_j, X_k)$ into the model; such terms can be fit using two-dimensional smoothers such as local regression, or two-dimensional splines.

4.8 Exercises

1. Suppose that a curve $\hat{g}$ is icomputed to smoothly fit a set of n points using the following formula

$$\hat{g} = \underset{g}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int \left[g^{(m)}\right]^2 dx \right)$$

where $g^{(m)}$ represents the mth derivtive of g (and $g^{(0)} = g$). Provide the functional form of g in the following scenarios.

   (a) $\lambda = \inf, m = 0$

   (b) $\lambda = \inf, m = 1$

   (c) $\lambda = \inf, m = 2$

   (d) $\lambda = \inf, m = 3$

   (e) $\lambda = 0, m = 3$

2. Suppose we fit a curve with basis functions $b_1(X) = I(0 \leq X \leq 2) - (X-1)I(1 \leq X \leq 2)$, $b_2(X) = (X-3)I(3 \leq X \leq 4) + I(4 \leq X \leq 5)$. We fit the regression model

$$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon$$

and obtain the coefficient estimates as $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1$ and $\hat{\beta}^2 = 3$. Plot the estimated curve between $X = -2$ and $X = 2$.

3. Consider the two curves $\hat{g}_1$ and $\hat{g}_2$ defined by

$$\hat{g}_1 = \underset{g}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int \left[g^{(3)}(x)\right]^2 dx \right) \hat{g}_2 = \underset{g}{\operatorname{argmin}} \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int \left[g^{(4)}(x)\right]^2 dx \right)$$

where $g^{(m)}$ represents the mth derivative of g.

   (a) as $\lambda \to \inf$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training RSS ?

   (b) as $\lambda \to \inf$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller test RSS ?

   (c) as $\lambda = 0$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training and test RSS ?

4.9 Solutions

1. Higher derivatives will allow for a higher degree polynomial to be fit to the data

   (a) Since λ is inf, we only need to worry about the second term. Minimizing area under $g(x)^2$ is same as taking $g(x) = 0$.

   (b) Similar to above, now area under $g^{(1)}(x)$ must be minimized which means the second derivative is zero and g is a constant. To minimize the residuals, $g = \sum_{i=1}^{n} y_i$.

   (c) Second derivative is zero means g is a linear function. To minimize residuals, this is same as linear regression least squares.

(d) thrid derivative is zero means that g is a quadratic. Hence, we fit a quadratic equation over the data by minimizing the least squares.

(e) We only need to bother with the residuals term now. Now g can take many forms depending on how smooth we wish the function to be.

2. For the range $[-2, 2]$ the curve is simply defined as below

$$\hat{Y} = \begin{cases} 1 & -2 \leq x < 0 \\ 2 & 0 \leq x < 1 \\ 3 - x & 1 \leq x \leq 2 \end{cases}$$

3. For $\lambda \to \inf$, the second part of the loss will dominate. As the derivative will increase, the solution can be a higher degree polynomial (as discussed in ans 1) which means a more flexible model and hence lower error on training data.

   (a) $\hat{g}_2$ is the more flexible model and thus should have lower training RSS.

   (b) Test RSS is not trivial but generally, the more flexbile the model, the higher variance it has and the lower test RSS. Thus, $\hat{g}_1$ should have the lower test RSS.

   (c) $\lambda = 0$ implies that both the error terms are similar and thus, both the functions are same.

Chapter 5

Resampling Methods

Sampling methods are a class of methods that serve a twofold purpose

- Provide a subset of data to be used for evaluating the test error rate
- Use different samples of data to assess the variability in the model parameters to choose the level of flexibility

5.1 Cross Validation

Validation set approach forms the bedrock for cross validation. We randomly split our training data into two sets, a training set and a validation set. We fit our models on the training set, and the validation set will serve as the unseen data. We can evaluate different models on the validation test to judge which of them performs the best.
A small problem with this approach is that the validation error will depend on the split of the data, i.e., we can expect slighlty different validation errors based on which subset of data we train. Hence, it is better to do this sampling multiple times in order to confidently select models and report test errors.
Furthermore, by preparing a validation set, we are reducing the size of our training set. Larger training data will usually result in better models. Hence, we might be overestimating the test errors in this case. A simple way to avoid this problem is to choose small sizes of the validation set and do the tests multiple times.

5.1.1 Leave One Out CV

An extension of the validation approach, here we train the data on all but on example. This way, if the data has n examples, we build n models (each trained on $n-1$ examples) and the error is

$$\text{test error } = \frac{1}{n}\sum_{i=1}^{n} error_i$$

where $error_i$ is the error on i^{th} observation from the model trained on remaining $n-1$ observations

Computing this can be extremely expensive when n is large. For linear regression, there exists a trick by which the time taken to get *test error* is exactly the same as the time to fit a single model on entire data set !

$$\begin{aligned}\hat{\beta} &= (X^TX)^{-1}X^TY \\ \text{Hat matrix } H &= X(X^TX)^{-1}X^T\end{aligned}$$

Let X_i denote the matrix X but with the i^{th} row removed, and similarly Y_i. Let x_i^T denote the i^{th} row of X and h_i be the diagonal entry of H. Then we have the following

$$\begin{aligned}X_i^TX_i &= X^TX - x_ix_i^T \\ X_i^TY &= X^TY - x_iy_i \\ h_i &= x_i^T(X^TX)^{-1}x \\ \hat{\beta}_i &= (X_i^TX_i)^{-1}X_i^TY_i \\ e_i &= y_i - x_i^T\hat{\beta}_i\end{aligned}$$

Also, we have the Sherman–Morrison formula for calculating the inverse of a perturbated matrix using the original matrix. Let A be the original invertible square matrix and u, v be column vectors. Then,

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

The formula can be verified by evaluating $LHS * RHS = RHS * LHS = I$.

Substituiting $A = X^TX$ and $-u = v = x_i$,

$$\begin{aligned}(X^TX - x_ix_i^T)^{-1} &= (X^TX)^{-1} + \frac{(X^TX)^{-1}x_ix_i^T(X^TX)^{-1}}{1 - x_i^T(X^TX)^{-1}x_i} \\ (X_i^TX_i)^{-1} &= (X^TX)^{-1} + \frac{(X^TX)^{-1}x_ix_i^T(X^TX)^{-1}}{1 - x_i^T(X^TX)^{-1}x_i} \\ \hat{\beta}_i(X_i^TY_i)^{-1} &= (X^TX)^{-1} + \frac{(X^TX)^{-1}x_ix_i^T(X^TX)^{-1}}{1 - x_i^T(X^TX)^{-1}x_i} \\ \hat{\beta}_i &= [(X^TX)^{-1} + \frac{(X^TX)^{-1}x_ix_i^T(X^TX)^{-1}}{1 - x_i^T(X^TX)^{-1}x_i}](X_i^TY_i) \\ \hat{\beta}_i &= [(X^TX)^{-1} + \frac{(X^TX)^{-1}x_ix_i^T(X^TX)^{-1}}{1 - h_i}](X^TY - x_iy_i) \\ \hat{\beta}_i &= \hat{\beta} - [\frac{(X^TX)^{-1}x_i}{1 - h_i}](y_i(1 - h_i) - x_i^T\beta_i +\hat{}h_iy_i) \\ &= \hat{\beta} - [\frac{(X^TX)^{-1}x_i(y_i - x_i^T\hat{\beta})}{1 - h_i}] \\ \text{Subsequently, } e_i &= y_i - x_i^T\hat{\beta}_i \\ &= y_i - x_i^T(\hat{\beta} - [\frac{(X^TX)^{-1}x_i(y_i - x_i^T\hat{\beta})}{1 - h_i}]) \\ &= (y_i - x_i^T\hat{\beta}) + \frac{h_i(y_i - x_i^T\hat{\beta})}{1 - h_i} \\ \text{or, } e_i &= \frac{y_i - x_i^T\hat{\beta}}{1 - h_i} \\ &= \frac{y_i - \hat{y}_i}{1 - h_i}\end{aligned}$$

Applying this formula for all the errors across the n models,

$$\begin{aligned}\text{test error } &= \frac{1}{n}\sum_{i=1}^{n} error_i \\ &= \frac{1}{n}\sum_{i=1}^{n}(\frac{y_i - \hat{y_i}}{1-h_i})^2\end{aligned}$$

which can be computed by simply building a single model on all the n data points.

5.1.2 k-Fold Cross Validation

Here, we divide the data set into k groups of roughly the equal size, and build k models. In each model, one of the folds is chosen as the validation set while the remaining $k-1$ folds are used for training. Let $MSE_1, MSE_2, \ldots, MSE_k$ denote the individual errors on the k subsets (when they are used for validation), then the estimate of test MSE is

$$MSE = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

Typically, $k = 5, 10$. Leave one out CV is a special case of k-fold CV. The MSE calculated above can help determine the best flexibility to chose for a given model (using the minima of MSE vs fliexibility) or help compare different types of models using the value of MSE.

5.1.3 k-Fold CV bias-variance

k-fold CV is preferable over LOOCV not just because of computational issues, but also due to a bias variance tradeoff. Note that the more data we use, it is expected that we will have less bias. By this logic, we expect LOOCV to have a low bias compared to k-fld CV. However, variance is also of interest in a statistical model. The n models that we fit in LOOCV are correlated with one another, since we have almost identical data sets being used for training. Hence, the average of the MSE's of such correlated models will tend to have a higher variance. On the other hand, k-fold CV effectively creates less correlated models which have quite lower variance in comparison. Hence k-fold CV is preferable in most of the cases.

$$Var(X+Y) = Var(X) + Var(Y) + 2Cov(X,Y)$$

Hence, higher the correlation, higher the variance of sum of two random variables.

5.2 Bootstrap

A widely applicable method that helps quantify uncertainty in the estimates of a statistical model. This is especially useful when formulae don't exist to quantify this uncertainty.
To estimate the uncertainty in estimates, we repeatedly create new data sets called bootstrap samples. Here, we simply select n observations from the original data set, but with replacement. This creates a slightly different version of the original data on which we train our model and get the estimates. From several bootsrapped data sets, we obtain a distribution of our estimate (a collection of estimates) whose mean and variance can help quantify the uncertainty.

5.2.1 Probability of selection

Since we are sampling with replacement, notice that the chance that the i^{th} observation is in the sampled data set is given by

$$\begin{aligned} P(not\ selection) &= (1-\frac{1}{n})^n \\ p(selection) &= 1-(1-\frac{1}{n})^n \end{aligned}$$

This comes using the product rule. The chance that any observation of the bootstrap sample is not equal to i^{th} observation is simply $1-\frac{1}{n}$ since we can choose any of the other $n-1$ examples.

$$\begin{aligned} \text{Note, } y &= \lim_{n\to\inf}(1-\frac{1}{n})^n \\ \ln(y) &= \lim_{n\to\inf}\frac{1}{n}\ln(1-\frac{1}{n}) \\ &= \lim_{n\to\inf}\frac{\ln(1-\frac{1}{n})}{\frac{1}{n}} \\ &= \lim_{n\to\inf}\frac{\frac{1}{1-\frac{1}{n}}\frac{1}{n^2}}{-\frac{1}{n^2}} && \text{Using L'Hospital's Rule} \\ &= \lim_{n\to\inf}-\frac{1}{1-\frac{1}{n}} \\ &= -1 \\ y &= \frac{1}{e} \\ \lim_{n\to\inf}P(selection) &= 1-\frac{1}{e}\approx 0.632 \end{aligned}$$

Hence, the probability that an observation from the original data set falls into a bootstrap sample is 63.2% which suggests that the sampled data sets are sufficiently variable !

Chapter 6

Tree Based Models

The idea behind this family of methods is to segment the region into subspaces and the prediction for any subspace is made by taking the *mean* or *mode* of the response in that region.
A simple decision tree may not give performance at par with linear regression or generalized additive models, but when combined with techniques like bagging and boosting, multiple trees can yield significant reduction in bias.
A simple decision tree finds its utility in being relatively simpler in interpretation than it's derived non-linear family.

6.1 Regression Trees

The algorithm behind building trees consists of two basic steps

1. Divide the predictors $X_1, X_2, \ldots, X_n$ into J **distinct** and **non-overlapping** regions $R_1, R_2, \ldots, R_J$.

2. For making a prediction in the region R_j, simply take the **response mean** of all the data points following in that region.

With the above definition, the loss becomes

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_J} (y_i - \bar{y}_{R_J})^2$$

But minimizing this loss is infeasible in practice as the total number of possible partitions are too large !

To overcome this, we build the trees in a greedy top down approach called **recursive binary splitting**. This method aims to find the best possible split at each level in a greedy manner.

6.1.1 The Algorithm

To start off, the alogithm will try to split the data into two regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ using the values of predictor X_j and the search is performed across all the predictors to choose the one which minimizes the RSS. Mathematically

$$R_1(j,s) = \{X|X_j < s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\}$$
$$\underset{j,s}{\text{minimize}} \sum_{i:x_i \in R_1} (y_i - \bar{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \bar{y}_{R_2})^2$$

where $\bar{y}_{R_1}$ and $\bar{y}_{R_2}$ are the response mean in the region R_1 and R_2 respectively.

The same algorithm is run recursively. The only change that is made that after the first split, we now wolr independently on two separate regions. For each region, we will not be able to consider all the possible values of s since the region has been restricted. Otherwise, the RSS term will remain the same.

The process is stopped when a pre-determined stopping criteria is reached. For making a prediction, we will first determine the region in which the test observation falls and then make the prediction based on the mean of the training samples in this region.

6.1.2 Tree Pruning

The tree build using above strategy can have good results on the training set, but will yield poor performance on the test set due to the high complexity. A better strategy is to build **short trees which are less complex and produce lower variance at cost of little bias** and are more interpretable.

A simple strategy can be to build only when there is a decrease in RSS above a threshold. This approach can be short-sighted as some future good branches can be missed.

Tree pruning is to **build a large tree and then shrink it back to obtain a subtree**. The RSS of a subtree can be found via cross validation. This approach is expensive for all possible subtrees.

Cost complexity Pruning or **weakest link pruning** is an approach to overcome this problem. For a non-negative parameter α, there exists a subtree T of the large tree T_0 such that the following is minimized

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

where $|T|$ is the number of terminal nodes in the tree. The formulation is similar to lasso and here we are controlling the complexity of the tree via a parameter α. Larger the α, the less number of terminal nodes there would be in the tree.

The algorithm can then be summarized as follows

1. Use *binary recursive splitting* to obtain a large tree on the data set. Stop only when at a terminal node, the number of observations is less than a minimum.
2. Apply *cost complexity pruning* in order to obtain a small tree T as a function of the cost parameter α (fix a set of α and get a set of subtrees).
3. Use K-fold cross validation to choose α. For each of the folds $k = 1, 2, \ldots, K$
   (a) Repeat steps 1 and 2 excluding the data from the kth fold to obtain a family of subtrees as a function of α.
   (b) Evaluate the mean predicted validation error as a function of α.
   Choose the value of α that minimizes the average error across all the folds.
4. Select the substree from step 2 that corresponds to the chosen value of α.

6.2 Classification Trees

Here we predict a qualitative variable and will report the **most occurring class in the region**. Since we are almost always also interested in the probabilistic aspects of the prediction, we also consider **the proportion of the classes occurring in a region**.

6.2.1 The Algorithm

The algorithm is almost similar to the regression trees but instead of RSS, we need to use something that is related to class proportions and frequencies. A very straightforward metric is **classification error rate**

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

where $\hat{p}_{mk}$ is the proportion of the kth class in the mth region. However, in practice, this metric is not sufficient to grow the trees.

6.2.2 Gini Index

Gini Index is defined as

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

for a given region. Clearly, G is close to zero when the node is pure, i.e. most of the observations come from the same class. Gini index is also know as a measure of **purity** since it's low value implies predominance of a single class in the node.

6.2.3 Entropy

Entropy is defined as

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

for a region. Entropy takes value close to zero if the class probabilities are close to 0 or 1. Gini index and Entropy are somewhat similar numerically.

6.3 Bagging

Decision tree suffer from a high variance. When the data is divided into smaller parts, it is quite likely to expect variance when the data set itself is changed. Bootstrap Aggregation of Bagging is a method to try to overcome this problem.

Averaging a set of independent random variables reduces variance. Consider n random variables with the same variance σ^2, then the variance of their average is σ^2/n. Naturally, we can extend this idea to build several prediction models on different training data sets and average out their results to reduce the variance in the response.

We train b different models, make predictions $\hat{f}_i(x)$ using these and then average them out to get a low variance

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x)$$

This reduction is obvious when the trees are completely independent. This is not true in majority of the cases. Since the trees are deep, we expect them to have very low bias, and similar expectations/means. The noise is introduced due to their variance. Suppose the trees come from the same distribution, but have some pairwise correlation ρ

$$E[\hat{f}_b(x)] = \mu, \quad Var(\hat{f}_b(x)) = \sigma^2 = E[\hat{f}_b(x)^2] - \mu^2$$
$$\rho = \frac{E[(\hat{f}_i(x) - \mu)(\hat{f}_j(x) - \mu)]}{\sqrt{Var(\hat{f}_i(x))Var(\hat{f}_j(x))}} = \frac{E[\hat{f}_i(x)\hat{f}_j(x) - \mu^2]}{\sigma^2}$$

Then, variance of the average of trees

$$\begin{aligned} Var(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)) &= E[\big(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)\big)^2] - E[\big(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)\big)]^2 \\ &= \frac{1}{B^2}\big(E[\sum_{b=1}^{B}\hat{f}_b(x)^2] + \sum_{i<j}E[\hat{f}_i(x)\hat{f}_j(x)]\big) - \mu^2 \\ &= \frac{1}{B^2}\big(B\mu^2 + B(B-1)(\sigma^2+\mu^2)\big) - \mu^2 \\ &= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \end{aligned} \tag{6.1}$$

When the number of trees is large enough, ρ will decide how much the variance shrinks.

Since we do not have access to different training datasets, we will get the average using bootstraps of the original data set. Note that **trees grown on bootstrapped data sets are deep and not pruned** so that they have low bias. Even though they may have high variance, averaging will reduce it out.

Averaging bootstrapped predictors works for regression while we **take the majority vote in case of classification**. The overall prediction is the most commonly occurring class across the B trees. Note that **using a large value of B will not lead to overfitting**, it should just be sufficiently large to ensure the error has come down compared to a single tree (building too many trees can take up significant time).

6.3.1 Out of Bag Error

It can be shown the probability of an observation being present in a bootstrapped dataset is $1 - 1/e$ (this can be shown by considering that the probability that an observation is not present in the data set is $((1 - 1/n)^n$ and taking the limit $n \to \inf$). Thus, on an average, a data will only contain about 2/3rd (0.63 to be exact) of the total observations in it. We can make use of the observations not used for training to estimate the error.

Any observation not part of the bootstrapped set is called **Out of Bag** and the error we are about to calculate is out of bag error. Now, we can take the ith observation, and get it's prediction from the sets where it was not used for training. This will be around 1/3rd of the predictors. We can combine these results using average in case of regression or majority vote in case of classification. Thus, we have a "test error" for all the observations in the data.

6.3.2 Variable Importance

We have been able to reduce the variance using the bagging approach, but the model is no longer interpretable due to the presence of multiple trees. We need to somehow aggregate all the trees to get this measure.

A simple workaround is to calculate the total amount the RSS (regression) or Gini Index (classification) has decreased across all the trees due to split on a particular variable. To make the values comparable, we simply take the average across B trees. A large value means that the variable has high importance.

6.4 Random Forests

Random forests overcome the problem of **decorrelation** in bagging. The trees built using bagging are similar because if there is a strong predictor, we expect it to dominate the splits in each of the trees and give similar trees.

From equation (6.1), with large number of trees B, the aim is to reduce ρ to get significant reduction in σ^2. **Random Forests achieve decorrelation by using only a subset of variables for each split (typically $\sqrt{p}$)**. This subset is chosen randomly at each split to ensure that the trees differ enough in structure to give less correlated predictions. **The performance improvement through random forests is solely the result of variance reduction** since each individual tree is achieving low bias.

Similar to bagging, **random forests will not overfit if we increase the number of trees B**. Hence, the choice can be made based on whether the reduction in test error is sufficient. Although, too many trees may start to make the model too rich, causing the development of correlated trees which increases variance. Thus, the main difference between bagging and random forest is the random choice of variables at split points in random forest. Reducing m will reduce the correlation between the trees. However, too much reduction in m may cause an increase in bias.

As the ratio of relevant variables to noise variables in the data decreases, the gap between random forests and boosting grows because at each split, the probability of selection of random variables becomes low. However, as this probability goes up, random forest catches up to boosting quickly.

In practical use, the concepts of random forest/bagging benefit non linear estimators the most due to the low bias and high variance of individual trees.

6.4.1 Algorithm

1. For $b = 1$ to B
   (a) Select a bootstrap sample Z of size N from the data
   (b) Recursively build the decision tree T_b using the following rules until any of the tree constraints is broken
      i. Select a subet of m variables out of p
      ii. Choose the best variable for split using predefined criteria
      iii. split the node into two child nodes
2. Output the ensemble of trees $\{T_b\}_1^B$
3. Make predictions according to the following

- Regression: $\hat{f}^B_{rf}(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$
  By the inventors, default $p = \lfloor p/3 \rfloor$ and minimum node size of 5
- Classification: $\hat{C}^B_{rf}(x) = majoirty\ vote\ \{\hat{C}_b(x)\}_1^B$
  where $\hat{C}_b(x)$ is the class prediction by a single tree and $\hat{C}^B_{rf}(x)$ is the class prediction by the random forest.
  By the inventors, default $p = \lfloor \sqrt{p} \rfloor$ and minimum node size of 1

But note that m and node size are both hyperparameters and the best values will vary between datasets.

Similar to section 6.3.1, we can use the samples not used for training a tree to calculate the out of bag error for that tree and consequently the forest. This can be used as an indicator to decide when to stop training.

6.4.2 Variable Importance

The relative variable importance is calculated using the total improvement in split criteria across all the trees

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 I(node(t) = l) \quad \text{the sum only includes the splits}$$

$$\mathcal{I}_l^2 = \frac{1}{B}\sum_{b=1}^{B} I_l^2(T_b)$$

where $\hat{i}_t^2$ is the square of improvement criteria at node t of the T^{th} tree, and $I_l^2(T)$ is the importance at the T^{th} tree. Since these are relative importance values, usually the variable with maximum importance is assigned a score of 100 and the remaining values of importances are scaled accordingly.

Another importance criteria uses the out of bag samples. Whenever a tree is grown, the oob sample is passed down the tree and the accuracy is recorded. Then, the j^{th} variable is randomly permuted in this sample and the accuracy is again recorded. This decrease in accuracy is averaged across all trees to get the accuracy value for that variable. The results are finally shown as a percent of the maximum. This gives more uniform importances. Note that we are not setting a variable to zero, but randomly permuting it.

6.5 Boosting

Boosting is a general tecnique that can be applied to multiple statistical learning techniques and not just decision trees. Like Bagging, boosting is also trained on multiple data sets derived from the training data, but instead of building independent trees on different data sets, **boosting is a sequential process that builds trees on modified versions of the original data set**.

Given a model, boosting will try to fit a tree on the residuals obtained by the model instead of the response Y. The model is then updated by adding this tree and the residuals are calculated again. **Trees in boosting are typically small, just a few terminal nodes** controlled by the parameter d and we allow the tree to fit the areas where improvement is needed in a slow manner. Shrinkage parameter λ allows to further slow down this process allowing for a variety of trees.

Boosting has three parameters

1. B, the number of trees. Unlike bagging, large B can overfit although slowly. Right value of B can be determined through cross validation.

2. Shrinkage parameter λ which controls how slowly the model is learnt and is a small positive number in the range 0.1 to 0.001 typically. Very smal λ can require a very large B to achieve a good performance.

3. The number of splits d in a single tree. Typically $d = 1$ is used and referred to a stump. Higer value signifies higher interaction between the variables. When using stumps, the model is simply additive since we are using a single variable in each of the trees.

Boosting algorithm is as follows

1. Set the residuals $r_i = y_i$ and the model $\hat{f}(x) = 0$.

2. For $b = 1, \cdots, B$ repeat

   (a) Fit the model $\hat{f}^b(x)$ with d splits on the data (X, r)

   (b) Update the model

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$

   (c) Update the residuals

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x)$$

3. Return the boosted model

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

6.6 Adaboost.M1

The convention is N is the number of training examples and M is the number of trees.
We consider the two class problem for simpler analysis, and denote them by $Y \in \{-1, 1\}$. The error rate is defined as

$$\overline{err} = \sum_{i=1}^{N} I(y_i \neq G(x_i))$$

and if the error rate is near 0.5, then the classifier is no better than a random guess.

Boosting builds trees in a sequential manner, and outputs of all the trees are weighted to get the final output from the classifier

$$y = \sum_{m=1}^{M} \alpha_m G_m(x)$$

where we build a total of M trees and G_m is a weak clasifier.

At each step, data weights are recalculated. Observations misclassified at the last step receive higher weight and vice versa. To start off, all observations receive the same weight $1/N$.

6.6.1 Basic Algorithm

1. Initialize the weights of all observations as $w_i = \cdots = w_n = 1/N$
2. For $m = 1$ to M
   (a) Fit a classifier $G_m(x)$ to the training data with weights w_i
   (b) Compute weighted error
   $$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i} \tag{6.2}$$
   (c) Compute tree weight
   $$\alpha_m = log\left(\frac{1 - err_m}{err_m}\right) \tag{6.3}$$
   (d) Update the observation weights as
   $$w_i \leftarrow w_i \cdot exp(\alpha_m \cdot I(y_i \neq G_m(x)), i = 1, \ldots, n \tag{6.4}$$
3. Output the final classifier output $\sum_{m=1}^{M} \alpha_m G_m(x)$

Note that the algorithm here returns discrete classes and is called *Discrete Adaboost.*

6.6.2 Forward Stagewise Additive Modelling

The boosting algorithm is one solution to a more general set of problems

$$\underset{\beta_1,\ldots,\beta_m,\gamma_1,\ldots,\gamma_m}{\text{minimize}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right)$$

where $L(y, f(x))$ is a loss function (log likelihood or squared error) averaged over the data and $b(x; \gamma_m)$ is a basis function that maps the input vector to a scalar. γ_m is a set of parameters, which can be parameters of a decision tree like depth, number of nodes and samples in a node.

The above loss function is difficult to directly minimize for a set of basis functions. Instead, the simpler problem to solve is

$$\underset{\beta,\gamma}{\text{minimize}}\, L\left(y, \beta \cdot b(x; \gamma)\right)$$

The general algorith then is

1. Initialize $f_0(x) = 0$
2. for $m = 1$ to M
   (a) Compute the parameters
   $$\beta_m, \gamma_m = \underset{\beta,\gamma}{\text{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$
   (b) Update $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

In the case of regression, the above formulation with least squares loss becomes

$$L = \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) + \beta b(x_i; \gamma))^2 = \sum_{i=1}^{N} (r_{i,m} + \beta b(x_i; \gamma))^2$$

which means we are fitting the new basis function on the residuals of the previous formulation. Though this loss is good for regression, we need different loss function for a classification problem.

6.6.3 Exponential Loss

Using the class indicators as $Y \in \{-1, 1\}$, we show that AdaBoost.M1 uses exponential loss to build the stagewise additive model

$$\begin{aligned}
L(y, f(x)) &= exp(-y \cdot f(x)) \\
\beta_m, G_m &= \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} exp(-y_i(f_{m-1}(x_i) + \beta G(x_i))) \\
&= \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} exp(-\beta y_i G(x_i)) \\
\text{with} \quad w_i^{(m)} &= exp(-y_i f_{m-1}(x_i)) \quad \text{independent of } \beta \text{ and } G(x)
\end{aligned} \tag{6.5}$$

The weights keep changing with each iteration. We can rewrite the last equation as

$$\begin{aligned}
\beta_m, G_m &= \underset{\beta,G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} exp(-\beta y_i G(x_i)) \\
&= \underset{\beta,G}{\operatorname{argmin}}\, e^{-\beta} \cdot \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \cdot \sum_{y_i \neq G(x_i)} w_i^{(m)} \\
&= \underset{\beta,G}{\operatorname{argmin}}\, e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} (1 - I(y_i \neq G(x_i))) + e^{\beta} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) \\
&= \underset{\beta,G}{\operatorname{argmin}} (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}
\end{aligned}$$

$$\begin{aligned}
\text{Additionally,} \quad G_m(x) &= \underset{G(x)}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i)) \\
\text{Substituiting,} \quad \beta_m &= \underset{\beta}{\operatorname{argmin}} (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} G_m(x_i) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} \\
\text{Differentiating,} \quad \beta_m &= \frac{1}{2} \log \left(\frac{\sum_{i=1}^{N} w_i}{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))} - 1 \right) \\
&= \frac{1}{2} \log \left(\frac{1 - err_m}{err_m} \right) \quad \text{from (6.2)}
\end{aligned}$$

Hence, the recursive $f(x)$ and $w(x)$ update becomes

$$\begin{aligned}
f_m(x) &= f_{m-1}(x) + \beta_m G_m(x) \\
w_i^{(m+1)} &= exp(-y_i f_m(x_i)) = exp(-y_i(f_{m-1}(x) + \beta_m G_m(x_i))) \\
&= w_i exp(-y_i \beta_m G_m(x_i)) && \text{from (6.5)} \\
\text{Also,} \quad \alpha_m &= 2\beta_m && \text{from (6.3) and (6.2)} \\
\text{and} \quad I(y_i \neq G(x_i)) &= \frac{1 - y_i G(x_i)}{2} \\
w_i^{(m+1)} &= w_i e^{-\beta_m} e^{\alpha I(y_i \neq G_m(x_i))}
\end{aligned}$$

which is similar to the form obtained in equation (6.4) with an added constant $exp(-\beta_m)$ same across all the data points and hence makes no difference. The probability of prediction of the classes then becomes

$$P(Y=1|X=x) = \frac{exp(f(x))}{exp(-f(x))+exp(f(x))} = \frac{1}{1+exp(-2f(x))}$$
$$\text{and} \quad sign(f(x)) = sign\left(\frac{1}{2}\log\left(\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}\right)\right)$$

meaning the output that is the sign of the function is sign of log likelihood and thus justified.

6.7 Boosting Trees

The following section defines N as the total number of training data points and M as the total number of trees/boosting rounds.
A tree can be defined and prepared in the following manner

$$f(x_i) = \gamma_j \text{ if } x_i \in R_j, \text{ where } R^j \text{ are partitions of input space}$$
$$T(x_i; R, \gamma) = \sum_{j=1}^{J} \gamma_j I(x_i \in R_j)$$
$$R, \gamma = \underset{R,\gamma}{\text{argmin}} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$
$$\text{A more simpler form} \quad R, \gamma = \underset{R,\gamma}{\text{argmin}} \sum_{i=1}^{N} L(y_i, T(x_i; R, \gamma))$$

γ_j for R_j is simply the quantity that minimizes the error averaged over all $x_i \in R_j$. For regression, this is simply the mean of the node (to minimize the mean squared loss) and for classifictation, the majority class (to minimize the misclassificaion rate). To build the tree, we recursively partition the data into two regions such that the partitioning results in a decrease in the above error function.

The stagewise additive model then becomes

$$f_M(x) = \sum_{j=1}^{J} T(x; R_m, \gamma_m)$$
$$R_m, \gamma_m = \underset{R_m,\gamma_m}{\text{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; R_m, \gamma_m)) \tag{6.6}$$

If the regions are already known, the above equation simplifies to

$$\gamma_{m,j} = \underset{\gamma_{m,j}}{\text{argmin}} \sum_{x_i \in R_j} L(x_i, f_{m-1}(x_i) + \gamma_{m,j}) \tag{6.7}$$

For regression problems, equation (6.6) is easily solvable since the minimzer of a given region is simply the average of y in that region. Hence, we can scan over the variables to find the optimum split that minimizes this loss. For absolute loss, the minimizer is the median.

For classification problems with the exponential loss, this equation is same as the adaboost algorithm discussed in section 6.6 with the restriction that the output of any tree is $\in \{-1, 1\}$.

6.7.1 Gradient Descent

For any loss function for a given output function,

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

$$\mathbf{f}(x) = \underset{f(x_1), f(x_2), \ldots, f(x_N)}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f(x_i))$$

where we have denoted $\mathbf{f}(x)$ as a vector in $\mathbb{R}^N$ space. Thus, minimizing the loss function is finding the minima in this $\mathbb{R}^N$ space. We can do this using gradient descent. Gradient descent will take multiple steps until it is able to reach the minima

$$\mathbf{f}_M(x) = \sum_{m=1}^{M} \mathbf{h}_m(x) \quad \text{where} \quad \mathbf{h}_m(x) \in \mathbb{R}^N$$

and $\mathbf{h}_m(x)$ are steps taken in such a way to reduce the loss. Thus, the final $\mathbf{f}(x)$ is simply a sum of these small steps. We usually start with $\mathbf{f}(x) = \mathbf{h}_0(x)$ as an initial guess.

At any point m, the gradient can be chosen as $\mathbf{h}_m = -\rho_m \eth_m$ where ρ_m is a scalar and the gradient $\mathbf{g}$ is calculated across all i data points

$$g_{i,m} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

The solution to ρ_m and the update of $\mathbf{f}$ then become

$$\rho_m = \underset{\rho}{\operatorname{argmin}}\, L(y, \mathbf{f_m} - \rho \mathbf{g}_m)$$

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m g_m$$

But, the problem is that for any new data point, the gradient will not be known since the target value is not available. Hence, we try to predict this gradient using a tree and then calculate $f_m(x)$ to make the desired prediction. Predicting gradient is a regression problem

$$R, \gamma = \underset{r,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} (-g_{i,m} - T(x_i; R, \gamma))^2$$

which has fast algorithms using regression trees. Note that the tree is built using gradient as the target, but γ is calculated from equation (6.7).

6.7.2 Gradient Boosting Algorithm

1. Initialize $f_0(x) = \operatorname{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M
   (a) For all points $i = (1, 2, \ldots, N)$, calculate gradient

$$g_{i,m} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

(b) Prepare a regression tree on the negative of gradients

$$R_m, \gamma_m = \underset{r,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} (-g_{i,m} - T(x_i; R, \gamma))^2$$

to get the terminal regions/nodes of the tree $R_j, j = 1, 2, \ldots, J$

(c) Calculate the γ on the above regions

$$\gamma_{m,j} = \underset{\gamma_{m,j}}{\operatorname{argmin}} \sum_{x_i \in R_{m,j}} L(x_i, f_{m-1}(x_i) + \gamma_{m,j})$$

(d) Update $f(x)$

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} \gamma_{m,j} I(x \in R_{m,j})$$

3. Return $f_M(x)$ (and the regions if required)

The procedure is called Gradient Boosting since the procedure of boosting is applied on the gradients obtained at each step (since we are effectively building trees to correctly predict the gradient).

Further note that γ calculated above is essentially ρg_m from previous gradient descent section. Thus, we first use least squares to get approximation of gradient as well as the regions of the tree, and then minimize the actual loss function to assign values to those regions/approximate correct gradient for the loss function in those regions.

6.7.3 Boosting Parameters

Tree Size

Ideally, we would build the tree as large as possible, and then prune it. This gives trees of varying sizes along the boosting iterations and also makes the whole process computationally inefficient. To avoid this, we simply restrict the size of all trees to be the same as J from the start.

The size of the tree captures the level of interactions among variables we are allowing to be represented. A decision stump only allows the direct effect of the variable to be captured. Tree of size 3 will allow second order interactions to also be captured. Thus, J should reflect the degree of interaction we permit in the model. Typically, $J > 10$ is too large and rarely useful. $4 \leq J \leq 8$ should suffice in most conditions, with $J = 6$ being a good starting point in such cases.

Number of Trees M

Clearly, increasing M will keep on reducing the error on the training data. Thus, a good idea is to use a validation set and monitor the changes in validation error to determine the best value for M. This is analogous to early stopping in neural networks.

Learning Rate

Apart from controlling M to prevent overfitting, we can also control a shrinkage parameter

$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J} \gamma_{m,j} I(x \in R_{m,j})$$

where ν is the amount by which we are scaling the contribution of each successive tree into the final model. Typically, $\nu < 0.1$ performs really well in practice. Note that the lower we make the contribution of individual trees into the model, the longer we need to train the model. Thus, ν and M have an opposite effect on each other. Typically boosting will involve small trees and large M should not pose much difficulty.

Low ν values give good improvements in mean squared loss and classification loss. However, not much improvement in seen for misclassification error loss. This stagewise shrinkage for boosting trees is similar to the $L1$ penalty in some respects. The best strategy to choose ν and M is to first determine ν for a fixed M and then choose M by early stopping. ν is also useful in stabilizing the model for training.

Subsampling

Stochastic Gradient Descent is the other name for this modified technique where at every stage when building a tree, we sample a fraction of the population ($\eta = 1/2$ or lower if large data set) with replacement (similar to bagging). This helps in reducing some variance due to the noisy trees, reduces training time, and usually gives superior results.

Usually, subsampling will work best with some shrinkage as well. Subsampling without shrinkage gives poor results.

6.7.4 Interpretation

A decision tree assigns the same value to all the data points that fall in the same region. Thus, when partitioning a given region into two regions using a variable, the two new regions have two different value as outputs for data points falling in those regions. Thus, the change in the loss function using a variable as a partition is given by

$$\hat{i}_j = \Big(\sum_{x_i \in R_{j,1}} \gamma_{j,1} + \sum_{x_i \in R_{j,2}} \gamma_{j,2}\Big) - \sum_{x_i \in R_j} \gamma_j$$
$$R_j = R_{j,1} \cup R_{j,2} \quad \text{is obtained by paritioning on variable } l$$

where γ are obtained by minimizing the loss function across the points $x_i \in R_j$. Thus, this quantity can be seen as a proxy for importance, by checking how much reduction in loss function is this variable able to bring. The partitioning is done by choosing the variable which gives the maximum reduction.

This reduction in loss is used as an indicator for importance

$$\mathcal{I}_l^2(T_m) = \sum_{j=1}^{J-1} \hat{i}_j I(v(j) = l)$$

where $v(j)$ is the variable selected for split at the j^{th} node, and the sum is only calculated over the $J-1$ internal nodes of the tree. The total importance across the M trees becomes

$$\mathcal{I}_l^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_l^2(T_m)$$

The actual importance is the square root of this value. Since the importance is relative, by convention, the most important variable is given the importance of 100, and the remaining variables are appropriately scaled with respect to this.

In case of multiple classes, we first sum the importance of a variable in a given class across all the trees, and then average out this value across the classes to get a single value.

Chapter 7

Support Vector Machines

Maximal margin classifier is the first idea for the development towards SVMs. Maximal margin classifier utilizes hyperplanes and requires that the data is linearly separable. The extension of this is Support Vector Classifier that can be applied to an even broader class of problems where classes may not be perfectly separable. SVMs are a further extension of SVCs to account for non linear separation boundaries.

Hyperplane in a p-dimensional space is a surface that is an affine subspace (means it need not pass through origin) in the $p-1$ dimensional space. For a 2D space it will be a 1D line, for a 3D space it will be a 2D plane, and so on. The equation is of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

Any point in the vector space can fall into one of the three regions

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \begin{cases} > 0 \\ = 0 \\ > 0 \end{cases}$$

and thus a hyperplane can be a useful demarcation between two regions or classes.

From now on, let $\beta^T = (\beta_1, \ldots, \beta_p)$, $x^T = (x_1, \ldots, x_p)$, β_0 be a scalar, and N denote the total number of training data points. Although not represented by a bold font, β and x are both vectors. Since $\beta^T x$ is a scalar, $\beta^T x = x^T \beta$.

7.1 Maximal Margin Classifier

Maimum Margin Classifier uses hyper planes to find a separable boundary between linearly separable data points.

Suppose we have a set of data points with p predictors and they belong to two classes given by $y_i = \{-1, 1\}$. Suppose the points are perfectly separable through a hyperplane. Then the following hold

$$\begin{aligned} & \beta_0 + \beta^T x_i && > 0 \quad \text{when} \quad y_i = -1 \\ \text{and} \quad & \beta_0 + \beta^T x_i && < 0 \quad \text{when} \quad y_i = 1 \\ \implies & y_i(\beta_0 + \beta^T x_i) > 0 \end{aligned}$$

Thus classification can be made into the positive or negative class simply based on the sign of the quantity $\beta^T x$. The further a point is, the more confident we will be in the classification.

Note that there can be infinite such hyperplanes that perfectly separate the data, and each can be obtained by slightly perturbing the given plane. Define margin as the minimum perpendicular distance from all training observations to this plane. The maximum margin classifier will be the one for which this margin is maximum.

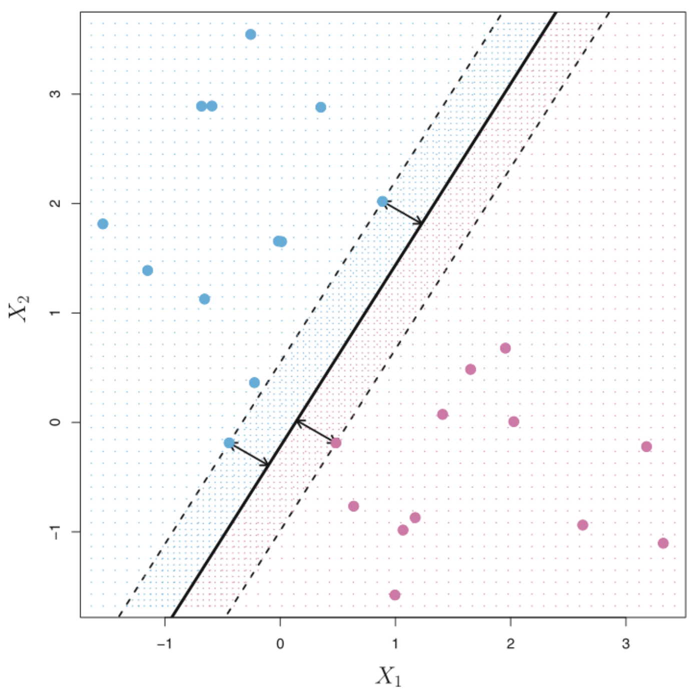

Figure 7.1: The Maximal Margin Classifier with the Support Vectors. Dotted lines represent the margin.

Note that the location of the maximal margin is determined only by the points closest to the boundary. If a point farther away would slightly move, the boundary would still be the same. Whereas if the point closer to the boundary would shift, the boundary itself would change as can be seen in figure 7.1. These set of observations are know as **support vectors**. And by symmetry, the perpendicular distances of these closest points from the plane are same.

7.1.1 Algorithm

Finding the boundary is same as solving for the following optimization problem (M is the margin)

$$\begin{aligned} &\underset{\beta_0,\beta}{\text{maximize}}\, M \\ &\text{subject to} \quad \sum_{i=1}^{p} \beta_i^2 = 1, \\ &\text{and} \quad y_i(\beta_0 + \beta^T x_i) \geq M \quad \forall \quad i = 1, 2, \ldots, N \end{aligned}$$

The constraint $\sum_{i=1}^{p} \beta_i^2 = 1$ gives rise to the unique property that $\beta_0 + \beta^T x$ is the perpendicular distance of the point x from the hyperplane, making the last constraint equation valid. The proof of the perpendicular distance equation is given in section 9.3.

By using the perpendicular distance using the equation from section 9.3, we can replace the constraint on the norm and rewrite as

$$\begin{gathered}\underset{\beta_0,\beta}{\text{maximize}}\, M \\ \text{subject to} \quad y_i(\beta_0 + \beta^T x_i) \geq M\|\beta\| \quad \forall \quad i = 1, 2, \ldots, N\end{gathered}$$

Note that the last equation remains same when we multiply by a positive constant. Hence, we can choose $\|\beta\| = 1/M$ for simplicity and the maximization problem becomes a minimization one (a factor of 1/2 is introduced to simplify the derivative of the square term)

$$\begin{gathered}\underset{\beta_0,\beta}{\text{minimize}}\, \frac{1}{2}\|\beta\|^2 \\ \text{subject to} 1 - \quad y_i(\beta_0 + \beta^T x_i) \leq 0 \quad \forall \quad i = 1, 2, \ldots, N\end{gathered}$$

For the following derivations involving linear optimization, refer to appendix (section 9.4). Invoking the Lagrangian multipliers, the new optimization problem is

$$\underset{\beta_0,\beta}{\text{minimize}}\, \frac{1}{2}\|\beta\|^2 - \sum_{j=1}^{N} \lambda_j (y_j(\beta_0 + \beta^T x_j) - 1)$$

where $\lambda = (\lambda_1, \ldots \lambda_N)^T$. Using the Wolfe Dual, the following is the dual problem

$$\begin{gathered}\underset{\beta,\beta_0,\lambda}{\text{maximize}}\, L(\beta, \beta_0, \lambda) = \frac{1}{2}\|\beta\|^2 - \sum_{j=1}^{N} \lambda_j (y_j(\beta_0 + \beta^T x_j) - 1) \\ \text{subject to} \quad \lambda > 0, \quad \frac{\partial L}{\partial \beta} = 0 \quad \text{and} \quad \frac{\partial L}{\partial \beta_0} = 0\end{gathered}$$

The partial derivatives give

$$\beta = \sum_{j=1}^{N} \lambda_j y_j x_j \quad \text{and} \quad 0 = \sum_{j=1}^{N} \lambda_j y_j$$

Substituiting in the dual,

$$\begin{aligned} L(\lambda) &= \frac{1}{2}\beta^T\beta - \sum_{j=1}^{N} \lambda_j (y_j(\beta_0 + \beta^T x_j) - 1) \\ &= \frac{1}{2}\beta^T\beta - \beta_0(\sum_{j=1}^{N} \lambda_j y_j) - \beta^T(\sum_{j=1}^{N} \lambda_j y_j x_j) + \sum_{j=1}^{N} \lambda_j \\ &= \frac{1}{2}\beta^T\beta - \beta_0(0) - \beta^T\beta + \sum_{j=1}^{N} \lambda_j \\ &= \sum_{j=1}^{N} \lambda_j - \frac{1}{2}\beta^T\beta = \sum_{j=1}^{N} \lambda_j - \frac{1}{2}(\sum_{i=1}^{N} \lambda_i y_i x_i^T)(\sum_{j=1}^{N} \lambda_j y_j x_j) \\ &= \sum_{j=1}^{N} \lambda_j - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j \end{aligned}$$

along with the constraints,

$$\underset{\lambda}{\text{maximize}} \sum_{j=1}^{N} \lambda_j - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

$$\text{subject to} \quad \lambda > 0 \quad \text{and} \quad 0 = \sum_{j=1}^{N} \lambda_j y_j$$

which is a quadratic optimization problem with linear constraints, and is solvable through linear optimization softwares.

KKT conditions also need to be satisfied for the optimal solution, which gives

$$\begin{aligned} \lambda_j^*(y_j(\beta_0^* + \beta^{*T} x_j) - 1 = 0 \quad \forall\, j = 1, 2, \dots N \\ \implies y_j(\beta_0^* + \beta^{*T} x_j) - 1 = 0 \quad \text{for points on margin} \\ \lambda_j^* = 0 \quad \text{for points away from the margin} \end{aligned}$$

which is expected based on the definition of the problem as only the points on margin decide the separating hyperplane. The predictions for new data points are simply made on the basis of the sign of $\beta_0^* + \beta^{*T} x$

7.2 Support Vector Classifier

The previous section was the best case scenario when all observations are perfectly separable. But in real data, this is seldom the case and we encounter the scenario that some observations will be misclassified. To take this into account, we maintain the same optimization as the previous section, but introduce new slack variables to account for the misclassified points

$$\begin{aligned} &\underset{\beta_0,\dots,\beta_p,M}{\text{maximize}}\, M \\ &\text{subject to} \quad \sum_{i=1}^{p} \beta_i^2 = 1, \\ &\text{and} \quad y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad \forall \quad i = 1, 2, \dots, n \\ &\epsilon_i > 0, \sum_{i=1}^{n} \epsilon_i \leq C \end{aligned}$$

which simply means that if

1. $\epsilon_i = 0$, we have correctly classified the observation
2. $\epsilon_i < 1$, the observation is correctly classified, but lies between the hyperplane and the margin
3. $\epsilon_i > 1$, the observation is misclassified

Chapter 8

Clustering

Clustering is an unsupervised learning techniques that aims to finds clusters or groups in the data such that observations in the same group are similar to each other while observations in different groups are different from each other.

Clustering is useful for an exploratory analysis of the data and also useful in problems like customer segmentation.

8.1 K-means Clustering

A very powerful technique that organizes the data into K distinct groups such that each observation will fall into exactly one group and when all the groups are combined, they cover the entire data set. K is determined beforehand and is the number of clusters we are going to make.

The fundamental idea of clustering is to reduce the within cluster variation. Let C_k denote the set containing the indices of the points falling in the cluster k and $W(C_k)$ be the within cluster variation for cluster k. Then,

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \sum_{k=1}^{K} W(C_k)$$

Using Euclidean distance as a measure of the intercluster distance between two points, we can redefine the optimization summing across all the dimensions of the data as

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \sum_{k=1}^{K} \frac{1}{|C_k|} \left\{ \sum_{i_1,i_2 \in C_k} \sum_{j=1}^{p} (x_{i_1 j} - x_{i_2 j})^2 \right\}$$

where $|C_k|$ is the number of observations in the cluster C_k. Considering all possible partitions is impossible for large n and we use the following algorithm to obtain the local optimum.

8.1.1 Algorithm

We repeat the following until some predefined convergence criteria

1. Random assign a cluster in $1, \ldots, K$ to each observation in the data.
2. Repeat the following till convergence
   (a) Calculate the centroid for each cluster where centroid is a p dimensional vector whose each component is the average of the components of all the points that fall in the considered cluster.

(b) Assign each observation the cluster index of the centroid that is closest to the given observation (using Euclidean distance).

It is usually a good idea to center and standardize the variables first so that the individual magnitudes and variances don't affect the Euclidean distances drastically.
To see why using distance from centroid is a good replacement for the pairwise distance, consider the following equation

$$\sum_{i_1,i_2\in C_k}(x_{i_1}-x_{i_2})^T(x_{i_1}-x_{i_2})=\frac{1}{2}\sum_{i\in C_k}\sum_{j\in C_k}(x_i-x_j)^T(x_i-x_j)$$

where the right side allows for all possible pairs including the ones where the indices might repeat. Continuing to expand the right hand side,

$$\begin{aligned}\sum_{i\in C_k}\sum_{j\in C_k}(x_i-x_j)^T(x_i-x_j) &= \sum_{i\in C_k}\left(|C_k|(x_i^Tx_i)-2x_i^T\sum_{j\in C_k}x_j+\sum_{j\in C_k}x_j^Tx_j\right)\\ &= \sum_{i\in C_k}\left(|C_k|(x_i^Tx_i)-2|C_k|x_i^T\bar{x}+\sum_{j\in C_k}x_j^Tx_j\right)\\ &= \sum_{i\in C_k}\left(|C_k|(x_i^Tx_i)-2|C_k|x_i^T\bar{x}\right)+\sum_{i\in C_k}|C_k|x_i^Tx_i\\ &= \sum_{i\in C_k}\left(2|C_k|(x_i^Tx_i)-2|C_k|x_i^T\bar{x}\right)\\ &= 2|C_k|\left\{\left(\sum_{i\in C_k}(x_i^Tx_i)\right)-|C_k|\bar{x}^T\bar{x}\right\}\\ &= 2|C_k|\left\{\left(\sum_{i\in C_k}(x_i^Tx_i)\right)-2|C_k|\bar{x}^T\bar{x}+|C_k|\bar{x}^T\bar{x}\right\}\\ &= 2|C_k|\left\{\left(\sum_{i\in C_k}(x_i^Tx_i)\right)-2(\sum_{i\in C_k}x_i^T\bar{x})+|C_k|\bar{x}^T\bar{x}\right\}\\ &= 2|C_k|\left\{\sum_{i\in C_k}x_i^Tx_i-2x_i^T\bar{x}+\bar{x}^T\bar{x}\right\}\\ &= 2|C_k|\left\{\sum_{i\in C_k}(x_i-\bar{x})^T(x_i-\bar{x})\right\}\end{aligned}$$

Thus, $\frac{1}{|C_k|}\left\{\sum_{i_1,i_2\in C_k}\sum_{j=1}^{p}(x_{i_1j}-x_{i_2j})^2\right\}=\left\{\sum_{i\in C_k}(x_i-\bar{x})^T(x_i-\bar{x})\right\}$

Thus, the quantity we set out to minimize for each cluster is indeed the sum of distance of each point from the centroid of the cluster, which means the cluster is to be chosen based on the closest centroid to minimize the overall intra cluster distance.

The optimum found by K-means clustering is local which makes it important to run the algorithm with different random initializations to get the minima.

Elbow Curve is a plot between the total intra cluster distance vs the number of cluster K and is a visual method to obtain the optimum number of clusters to use. The elbow shape refers to the fact that if there is indeed clusters present in the data, the plot will see a sharp decline in the intra cluster distance for some k.
Note that the curve is going to always keep decreasing as in the limiting case when we have n

points and n clusters, the total distance will be zero. Hence, the number of clusters must be carefully chosen.

8.2 Hierarchical Clustering

K-means suffers from the disadvantage that the number of clusters needs to be specified beforehand. Hierarchical does not require such a consideration beforehand. here we dicsuss the **bottom-up** or **agglomerative clustering** approach. Hierarchical clustering is visualized using a **dendogram** which is a tree like diagram draw upside down. Starting from the bottom, branches are originate from the individual data points and slowly start merging as we move upward. The earlier the branches merge, the similar the data points are and vice versa. (Be careful to not judge the similarity from the proximity on the horizontal axis)

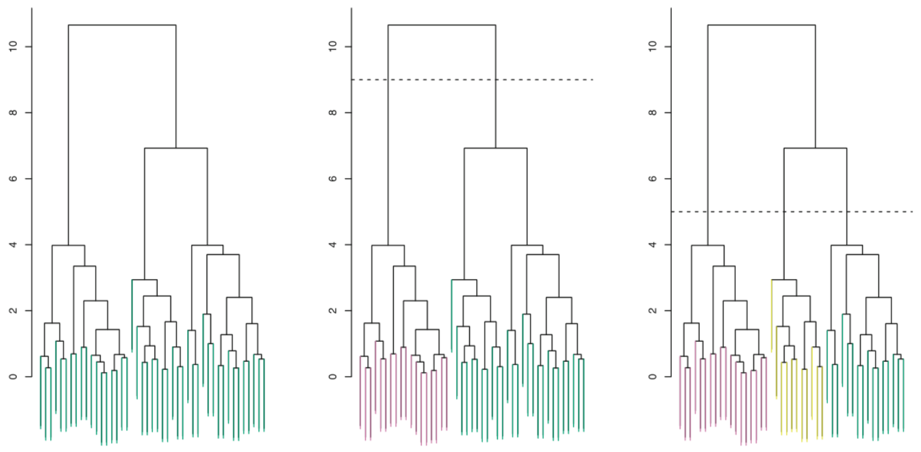

Figure 8.1: Visualization of dendogram. The two curves on the right colour the different clusters obtained based on the height at which we decide to cut.

The number of clusters is simply determined by the height at which we made the cut. The middle figure in figure 8.1 shows a cut at height 9 which results in two branches and thus two clusters.
Changing the height from the highest value to the lowest value will result in 1 and n clusters respectively. Thus, we do not need to specify the number of clusters beforehand but it is rather to be chosen by us based on the height. This height can be seen similar to K in K-means clustering.

The inherent nesting of the clusters as indicated visually by the dendogram may not be possible in every data set and there will be cases when K-means clustering may be superior.

8.2.1 Algorithm

Hierarchical Clustering is performed in a bottom-up approach. Start with n clusters where each observation is it's own cluster. Define a dissimilarity measure between each pair of observation. This can be Euclidean distance as well. Now, cluster the observations that are least dissimilar into the same group, which will give us $n-1$ clusters. Again use the dissimilarity measure to group two similar observations until the total number of clusters is 1.

Consequently, there will be cases when we need to determine the dissimilarity between a group and an observation or a pair of groups. This is done using **linkage**. Four types of linkages used are **complete, average, single** and **centroid**. Average and complete linkages are preferred over single linkages, and all three are more popular than centroid linkage. Average and complete likages will usually give balanced dendograms.

Following are teh descriptions of individual types of linkages

- Complete
  Maximal intercluster dissimilarity. Take the maximum of the pairwise dissimilarity between observations of cluster A and cluster B.
- Single
  Minimum intercluster dissimilarity. Take the minimum of the pairwise dissimilarity between observations of cluster A and cluster B.
  Single linkage can result in extended trailing clusters in which single observations fuse one at a time.
- Average
  Mean intercluster dissimilarity. Take the average of the pairwise dissimilarity between observations of cluster A and B.
- Centroid
  Take the dissimilarity between the centroid of cluster A and B. Centroid linkages can result in undesirable inversions.

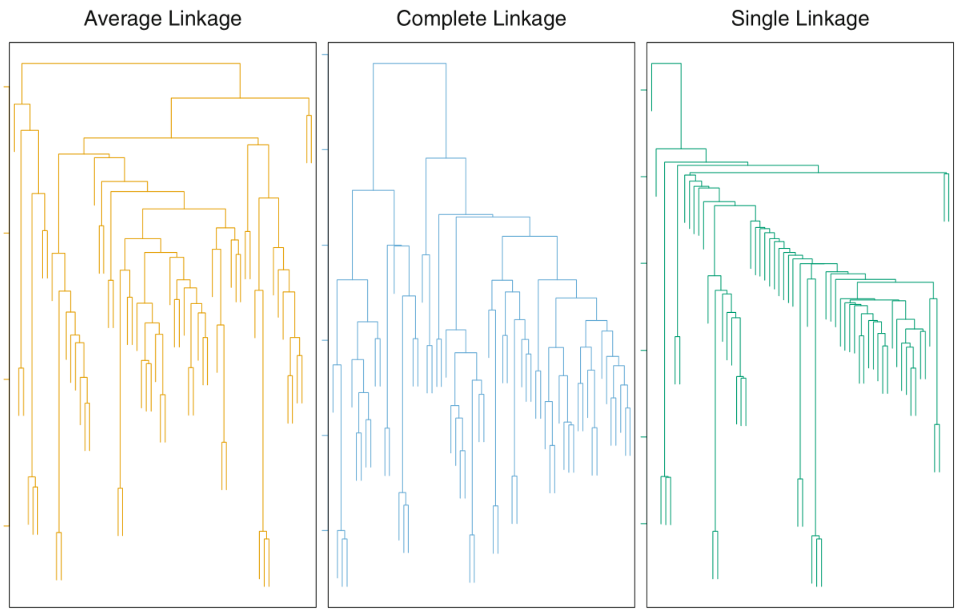

Figure 8.2: Visualization of dendograms obtained for different choice of linkages.

The algorithm can be summarized as follows

1. Start with n clusters where each observation is it's own cluster. Compute $n(n-1)/2$ pairwise dissimilarity measures between all pairs.
2. For $i = n, n-1, \ldots, 2$
   (a) Compute the pairwise dissimilarity between all i clusters and take the two clusters with the least dissimilarity (or highest similarity). Fuse them into a single cluster. The dissimilarity measure is also indicative of the height in the dendogram where the two clusters fuse.
   (b) Recompute the pairwise dissimilarity between the $n-1$ clusters.

8.2.2 Dissimilarity Measures

So far, euclidean distance has been considered as the defacto dissimilarity measure. In some cases, this may not work well if the magnitude of the observations vary significantly between the different predictors. In such cases, correlation based measures can be preferred since they will group observations with similar behaviour together and not focus on magnitude.
This can be useful in retail behaviour when we want to check profiles based on the whether similar products are purchased rather than how many of them are purchased.

8.2.3 Key Considerations

Following are a set of general rules when doing clustering

- centering and bringing the variables to the same scale is useful when measuring Euclidean distance for the obvious reason of not letting magnitudes affect the distances.
- Different types of clustering approaches should be explored to check which performs the best. This is important as in unsupervised learning, the structure of data is not known beforehand and it is important to explore multiple hypothesis.
- Several choices of similarity measures and linkages can be explored for further understanding of data.
- Clustering can be non robust and thus the results should be "validated" by performing clustering on multiple subsets of data to assure stability.
- In come cases, the hard cluster assignment of K-means and hierarchical clustering may not be useful. Probabilistic models like mixture models can be explored in this case.

Chapter 9

Appendix

9.1 Matrix Calculus used in Logistic Regression Derivation

The equations below present the extended version of the matrix calculus in section 2.1.1

Note the derivate of $\beta^T x$ which is a scalar. β and x are $p+1 \times 1$ vectors

$$\frac{\partial}{\partial \beta}\beta^T x = \begin{bmatrix} \frac{\partial}{\partial \beta_0}\sum_{j=0}^{p}\beta_j x_j \\ \frac{\partial}{\partial \beta_1}\sum_{j=0}^{p}\beta_j x_j \\ \vdots \\ \frac{\partial}{\partial \beta_p}\sum_{j=0}^{p}\beta_j x_j \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix} = x \tag{9.1}$$

We solve the single derivate first (y_i and $p(x_i$ are scalars)

$$\begin{aligned} \frac{\partial}{\partial \beta}\sum_{i=1}^{n} y\beta^T x_i + log(1 - exp(\beta^T x_i)) &= \sum_{i=1}^{n} y\frac{\partial}{\partial \beta}y\beta^T x_i - \frac{exp(\beta^T x_i)}{1 - exp(\beta^T x_i)}\frac{\partial}{\partial \beta}y\beta^T x_i \\ &= \sum_{i=1}^{n} x_i(y_i - p(x_i)) \end{aligned}$$

To get the second derivative, which is the Hessian matrix, we take derivative with β^T (to get a matrix)

$$\frac{\partial}{\partial \beta^T}\sum_{i=1}^{n} x_i(y_i - p(x_i)) = -\frac{\partial}{\partial \beta^T}\sum_{i=1}^{n} x_i p(x_i)$$

First, let's take the derivative of the scalar $p(x_i)$ with a scalar β_j

$$\begin{aligned} \frac{\partial}{\partial \beta_j}p(x_i) &= \frac{\partial}{\partial \beta_j}\frac{exp(\beta^T x_i)}{1 + exp(\beta^T x_i)} \\ &= \frac{\partial}{\partial \beta^T x_i}\frac{exp(\beta^T x_i)}{1 + exp(\beta^T x_i)}\frac{\partial}{\partial \beta_j}\beta^T x_i \quad \text{chain rule} \\ &= \frac{exp(\beta^T x_i}{(1 + exp(\beta^T x_i))^2}x_{i,j} \quad \text{from 9.1} \\ &= p(x_i)(1 - p(x_i))x_{i,j} \end{aligned}$$

Hence, the hessian matrix is

$$
\begin{aligned}
\frac{\partial l^2}{\partial\beta\partial\beta^T} &= -\frac{\partial}{\partial\beta^T}\sum_{i=1}^{n} x_i p(x_i) \\
&= \sum_{i=1}^{n}\begin{bmatrix} \frac{\partial}{\partial\beta_0}x_{i,0}p(x_i) & \frac{\partial}{\partial\beta_1}x_{i,0}p(x_i) & \dots & \frac{\partial}{\partial\beta_p}x_{i,0}p(x_i) \\ \frac{\partial}{\partial\beta_0}x_{i,1}p(x_i) & \frac{\partial}{\partial\beta_1}x_{i,1}p(x_i) & \dots & \frac{\partial}{\partial\beta_p}x_{i,1}p(x_i) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial}{\partial\beta_0}x_{i,p}p(x_i) & \frac{\partial}{\partial\beta_1}x_{i,p}p(x_i) & \dots & \frac{\partial}{\partial\beta_p}x_{i,p}p(x_i) \end{bmatrix} \\
&= \sum_{i=1}^{n} p(x_i)(1-p(x_i))\begin{bmatrix} x_{i,0}x_{i,0} & x_{i,0}x_{i,1} & \dots & x_{i,0}x_{i,p} \\ x_{i,1}x_{i,0} & x_{i,1}x_{i,1} & \dots & x_{i,1}x_{i,p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{i,p}x_{i,0} & x_{i,p}x_{i,1} & \dots & x_{i,p}x_{i,p} \end{bmatrix} \\
&= \sum_{i=1}^{n} p(x_i)(1-p(x_i))x_i x_i^T
\end{aligned}
$$

9.2 Comparison of different Classifiers

Some characteristics of different learning methods. Key: ▲*= good,* ◆*=fair, and* ▼*=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Figure 9.1: Comparison of Classifiers

9.3 Perpendicular distance in Maximum Margin Classifier

The equation of the plane is given by $\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p = 0$. Let x be the point whose perpendicular distance from the plane we want to calculate and let x_a a point on the plane.

The distance between x and x_a is

$$d^2 = \sum_{i=1}^{p}(x_i - x_{ai})^2$$

and the minimum of this distance is the required perpendicular distance. Since x_a lies on the plane, we have a constrained optimization

$$\underset{x_{a1},\ldots,x_{ap}}{\text{minimize}} \sum_{i=1}^{p}(x_i - x_{ai})^2 + \lambda(\beta_0 + \beta_1 x_{a1} + \cdots + \beta_p x_{ap})$$

Taking the partial derivatives with respect to all components of x_a and λ,

$$\begin{aligned}
-2(x_i - x_{ai}) + \lambda\beta_i &= 0 \quad \forall \quad i = 1, \ldots, p \\
\beta_0 + \beta_1 x_{a1} + \cdots + \beta_p x_{ap} &= 0 \\
\implies \beta_0 + \sum_{i=1}\beta_i(x_i - \frac{\lambda\beta_i}{2}) &= 0 \\
\implies 2\frac{\beta_0 + \sum_{i=1}^{p}\beta_i x_i}{\sum_{i=1}^{p}\beta_i^2} &= \lambda \\
\implies x_i - x_{ai} &= \beta_i \frac{\beta_0 + \sum_{i=1}^{p}\beta_i x_i}{\sum_{i=1}^{p}\beta_i^2} \quad \forall \quad i = 1, \ldots, p \\
\text{giving} \quad \sum_{i=1}^{p}\beta_i^2 \frac{(\beta_0 + \sum_{i=1}^{p}\beta_i x_i)^2}{(\sum_{i=1}^{p}\beta_i^2)^2} &= d^2 \\
\text{or} \quad \frac{(\beta_0 + \sum_{i=1}^{p}\beta_i x_i)^2}{\sum_{i=1}^{p}\beta_i^2} &= d^2
\end{aligned}$$

Hence, the perpendicular distance is same as putting the point in the equation of the hyperplane and dividing by the norm of the coefficients. In the special case where the norm equals 1, the distance becomes $|\beta_0 + \sum_{i=1}^{p}\beta_i x_i|$.

9.4 Lagrange Multiplier, Primal and Dual

Consider a constrained optimization problem of the form

$$\underset{x}{\text{minimize}}\, f(x) \quad \text{subject to} \quad h(x) = c$$

where $x \in \mathbb{R}^n$ is a vector, c is a constant and $f : \mathbb{R}^n \to \mathbb{R}$. To invoke the concept of Lagrange multipliers, we use gradients.

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix}$$

For any level surface, the gradient will be perpendicular to the function. This arises from the definition of the gradient, which is the direction of greatest change. But for a level surface, this should not happen since the function value is constant. Hence, the gradient vector will lie perpendicular to the level surface (figure 9.2).

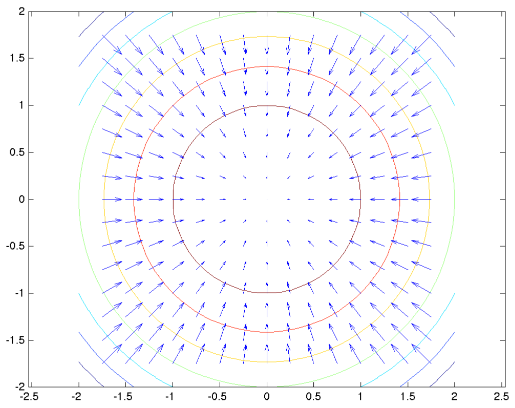

Figure 9.2: Gradients for level surfaces $f(x, y) = x^2 + y^2$

In the context of the solution to the original problem, we will notice that the maximum under constraint occurs when the two surfaces are tangent to each other. This implies that the gradients at the point of maxima are in the same direction (values could be different). We represent this equality using **Lagrange Multiplier** λ (scalar).

$$\nabla f(x) = \lambda \nabla h(x)$$

Thus, now we have $n+1$ equations with us. n equations from the proportionality of the gradients and 1 from the original constraint. This helps solve for n components of x and the value of λ. The problem is often formulated as follows

$$L(x, \lambda) = f(x) - \lambda(h(x) - c)$$
$$\frac{\partial L}{\partial x}(x, \lambda) = 0 \quad \text{and} \quad \frac{\partial L}{\partial \lambda}(x, \lambda) = 0$$

and we can have multiple equality constraints, with a different multiplier for each equation. Using Lagrange multiplier, we converted a constrained optimization problem to an unconstrained one.

Inequality Constraintss require slightly different handling since we do not have level surfaces anymore. In this context, we have the following problem

$$\begin{aligned} \underset{x}{\text{minimize}}\, f(x) &\quad f(x) : \mathbb{R}^n \to \mathbb{R} \\ \text{subject to} \quad h(x) = 0 &\quad h(x) : \mathbb{R}^n \to \mathbb{R}^m \\ \text{and} \quad g(x) \leq 0 &\quad h(x) : \mathbb{R}^n \to \mathbb{R}^p \\ L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x) \end{aligned}$$

where L is the Lagrangian and, $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p$ are thought of as penalties associated with the constraints. This is called the **Primal**. For a given λ and μ we have an unconstrained problem.

The **Dual function** is defined as

$$q(\lambda, \mu) = \min_x L(x, \lambda, \mu) \quad q : \mathbb{R}^{m+p} \to \mathbb{R}$$

Notice that the constraint $g(x)$ is violated when $g(x) > 0$. We want to penalize the violation of constraint, meaning, $\mu^T g(x) \geq 0$ would make the minimum larger, giving the additional constraint that $\mu \geq 0$.

With the definition of the dual function, the **Dual problem** is

$$\begin{gathered}\underset{\lambda,\mu}{\text{maximize}}\, q(\lambda,\mu) \\ \text{with} \quad \mu \geq 0 \quad \text{and} \quad q \text{ is bounded}\end{gathered}$$

The idea is to solve the dual problem to get the lower bound on the minimum of the original **Primal** problem. It is easy to show through the definitions that for the optimal x^*

$$\begin{aligned} q(\lambda,\mu) &= \min_x L(x,\lambda,\mu) \leq L(x^*,\lambda,\mu) \\ &= f(x^*) + \lambda^T h(x^*) + \mu^T g(x^*) \leq f(x^*) \end{aligned}$$

since x^* satisfies the original constraints. Thus, the maximum of the dual gives the lower bound on the optimal solution.

In the case of inequalities, we notice the following at the optimal solution

$$\begin{aligned} &\text{if} \quad g(x^*) = 0 \quad \text{then} \quad \frac{df}{dx}(x^*) + \lambda^T \frac{dh}{dx}(x^*) + \mu^T \frac{dg}{dx}(x^*) = 0 \text{ and } \mu > 0 \\ &\text{if} \quad g(x^*) < 0 \quad \text{then} \quad \frac{df}{dx}(x^*) + \lambda^T \frac{dh}{dx}(x^*) + \mu^T \frac{dg}{dx}(x^*) = 0 \text{ and } \mu = 0 \end{aligned}$$

where the second case occurs because after assuming $g(x) < 0$, we are just looking at equating the derivatives of $f(x) - \lambda^T h(x)$ to zero. The two equations above lead to

$$g(x^*) \leq 0, \quad \mu \geq 0, \quad \mu^T g(x^*) = 0$$

and the last set of conditions are called Karush–Kuhn–Tucker conditions (satisfied only by the optimal point).

Wolfe Dual is a special case of the above equations. Suppose we do not have any equality constraints, and we also know that $f(x)$ is convex and differentiable, and $g(x)$ is concave (linear will also do) and differentiable. Then the dual is

$$\max_\mu q(\mu) = \max_\mu \min_x L(x,\mu)$$

The minimum occurs at the point where gradient of L is zero and the **Wolfe Dual** is

$$\begin{gathered}\underset{x,\mu}{\text{maximize}}\, L(x,\mu) \\ \text{subject to} \quad \mu \geq 0 \quad \text{and} \quad \frac{\partial L}{\partial x} = 0\end{gathered}$$

along with the KKT conditions valid.