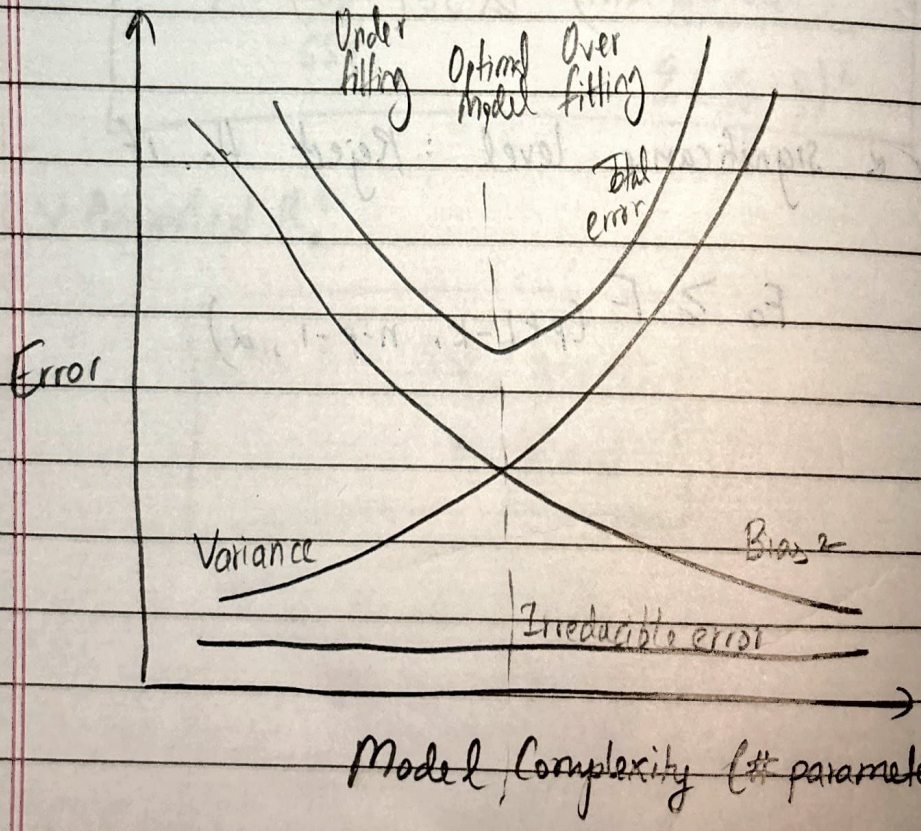
WEEK-7

⇒ ~~Cross Validation:~~

⇒ Bias-Variance trade off on test data set:



Training and Validation data sets

↳ for large data sets divide data set into training dataset (~70%) and remaining validation / test data.

- Training set: $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
- Test set: $(x_{0,i}, y_{0i})$, $i = 1, \ldots n_t$ observations

→ training error rate:

$$MSE_{training} = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \hat{\beta})^2$$

→ Test error rates

$$MSE_{Test} = \frac{1}{n_t} \sum_{i=1}^{n} (y_{0,i} - x_{0,i}^T \hat{\beta})^2$$

↳ If there is enough data:

(1) training set
(2) validation set
(3) test set.

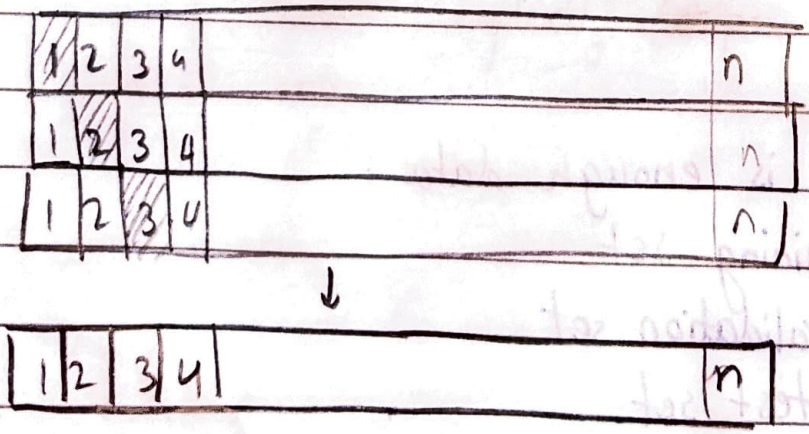
↳ use training set to fit the model
↳ use validation set, to predict validation set errors.

⇒ **Sampling for small data sets:**

↳ validation of models by repeatedly drawing random samples from a training set:
- Validation Set (random Sampling)
- K-fold cross validation
- Bootstrap.

⇒ **Leave one-out-cross-validation (LOOCV):**

↳ Build model using (n-1) samples and predict the response (y_i) for the remaining sample.



$$CV_1 = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \hat{\beta}^{(1)})^2$$

→ Advantages:

↳ far less bias comparison to the validation set approach.
↳ Training set contains (n-1) observations each iteration.
↳ Yield the same results.
↳ Does not overestimate the test error rate as much as the validation set approach.

→ Diadvantages:

↳ Expensive to implement due to fitting happens n times.
↳ It may select a ~~se~~ model of excessive size than the optimal model.

⇒ K-Fold Cross Validation:

↳ training data into K disjoint samples of equal size,

$Z_1, Z_2, \ldots Z_K$

↳ For each validation sample Z_i
- Use remaining data to fit the model
- Predict the response for the validation sample Z_i and compute mean square error (MSE_i)
- Repeat for all K samples
- The K-fold CV

$$CV_{(K)} = \frac{1}{K} \sum_{i=1}^{K} MSE_i$$

* For $K=n$, Leave-one-out-cross-validation (LOOCV)
* In practice $K=5$ or 10 is taken.
* less computation cost.
* For computationally intensive learning ~~models~~ methods
  - LOO CV fits the model n times
  - K-fold CV fits the model K times

MULTIPLE REGRESSION MODELLING BUILDING & SELECTION:

Loading Data:

- Dataset 'nyc' is given in ".csv" format

↳ load data from the file using function read.csv()

reads data and creates data frame.

read.csv(file, row.names = 1)

nyc ← read.csv(file, row.name=1)

- View(nyc) will display the dataframe in a tabular format
- head(nyc) and tail(nyc) will display the first and last six rows from the data frame.

y_i = Price of dinner
x_1 = Customer rating of food
x_2 = decor
x_3 = service
x_4 = east or west

→ Pair wise scatter plot.

```
plot(nyc, main = "Pairwise Scatter plot")
round(cor(nyc), 3)
```

↳ correlation

↳ round to decimal points.

⇒ Model Building:

↳ Dependent variable (y) depends on p independent variables x_i, $i = 1, 2 \dots p$

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots \hat{\beta}_p x_p + \varepsilon$$

for ith observation,

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1,i} + \hat{\beta}_2 x_{2,i} + \cdots \hat{\beta}_p x_{p,i} + \varepsilon_i$$

↳ building the model using lm function:

lm(formula, data)

```
lm(dependent var ~ indep.var1 + indep.var2)
```

```
nyc mod_1 <- lm(Price ~ Food + Decor + Service + East, data = nyc)

or

nyc mod_1 <- lm(Price ~ . , data = nyc)

summary(nyc mod_1)
```

→ New model Dropping service

↳ service has less impact and is not needed in the model.

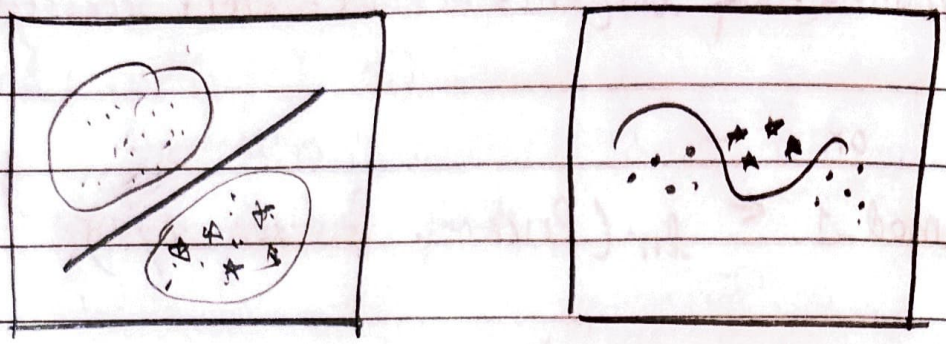
```
nyc mod_2 <- lm(Price ~ Food + Decor + East, data = nyc)
```

Classification:

↳ assigning classes to data points

- Binary
- Multiclass.

- linearly separable problems
- Linearly non-separable problems
- Kernel model methods.



LOGISTIC REGRESSION:

↳ classification technique.
↳ decision boundary (generally linear) derived based on probability interpretation
  • Results in a non-linear optimization problem for parameter estimation

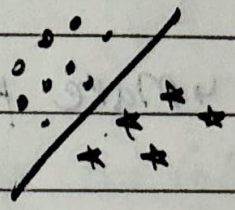
⟹ Binary Classification problem:

↳ task of identifying a category that a new observation belongs to based on the data with known categories (2 categories for binary classification)
↙
simple yes or no problem.

Input Features:

- can be both quantitative or qualitative
- if the ~~iput~~ inputs are quantitative, then there has to be a systemic way of converting them into quantities.
- some data & analytics approach can handle qualitative variables directly.

Linear Classifier:



- decision function is linear
- binary classification can be performed depending on the side of the half-plane that the data falls in.
- Estimating the binary outputs from the probabilities is straightforward through simple thresholding.

Linear and log Models:

Make $p(x)$ a linear function of x

$$p(x) = \beta_0 + \beta_1 X$$

• this makes $p(x)$ unbounded below 0 and above 1

↳ Make $\log(p(x))$ a linear function of x

$$\log(p(x)) = \beta_0 + \beta_1 x$$

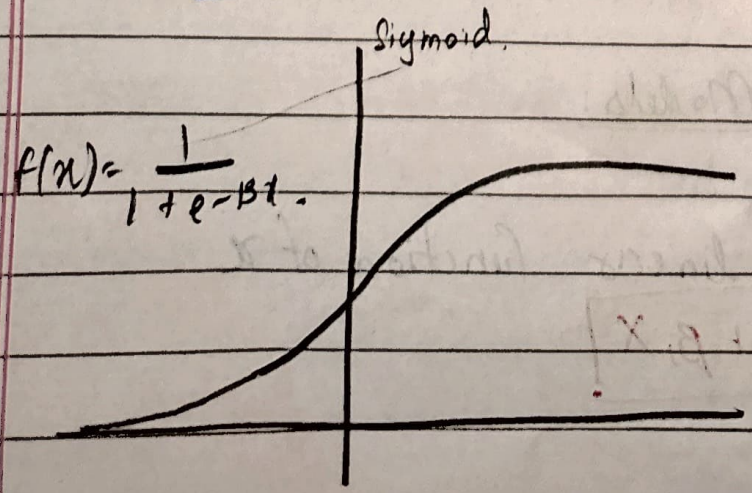
↳ Bounded only on one side.

⇒ Sigmoid Function:

↳ Make $p(x)$ a sigmoid function of x.

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

or $$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x$$



- $p(x)$ bounded above by 1 and below by 0
- good modeling choice for real life scenarios
- the LHS can be interpreted as the log of odds-ratio in the second eqn.

Estimation of the parameters:

- we find the parameters in such a way that plugging these in the model equation should give the best possible classification for the inputs from both the ~~side~~ classes.

- this can be formalized by maximizing the following likelihood function.

$$L(\beta_0, \beta_i) = \prod_{i=1}^{n} (p(x_i))^{y_i} (1-p(x_i))^{(1-y_i)}$$

when x_i belongs to class 0, $y_i = 0$

when x_i belongs to class 1, $y_i = 1$

⇒ Log-likelihood Function:

$$l(\beta_0, \beta_1) = \sum_{i=1}^{n} y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))$$

↳ Now the parameters can be estimated by maximizing the above expression using any non-linear optimization solver.

⇒ Logit model

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

- If $\beta_0 + \beta_1 x$ is non negative then we get $p > 0.5$ and $y = 1$ otherwise $p < 0.5$ and $y = 0$.
- Decision boundary is the eqn $\beta_0 + \beta x$.

Regularization:

general objective:

$$\min_{\theta} -L(\theta)$$

where $L(\theta) = \sum_{i=1}^{n} y_i \log(p(x_i)) + (1-y_i) \log(1-p(x_i))$

- when large number of independent variables are present, logistic regression tends to overfit
- to prevent over-fitting, we need to penalize the coeff.
- this is known as regularization.

- regularization helps in building non-complex models that avoids capturing noise in model due to over-fitting

- the objective now becomes

$$\min_{\theta} -L(\theta) + \lambda * h(\theta)$$

where λ is regularization parameter and $h(\theta)$ is regularization function

↳ Depending on $h(\theta)$, the regularization can be classified as L_1 or L_2 type.

↳ $h(\theta) = \theta^T \theta$ for L2 type regularization.

↳ larger the value of λ, more is the regularization strength.

↳ Regularization helps the model work better on test data to the fact that over-fitting is minimised on training data.

PERFORMANCE ~~MATRIX~~ METRICS.

① Confusion Matrix:

		True Condition.	
	Total population	Condition tve	Condition -ve
Predicted condition	Predicted condition positive	True tve	False tve Type I error
	Predicted condition -ve	False -ve Type II error.	True -ve.

TP - true +ve - Correct identification of +ve labels
TN - true -ve - correct identification of -ve labels
FP - false +ve - Incorrect identification of +ve labels
FN - false -ve - Incorrect identification of -ve labels

$$N = TP + TN + FP + FN$$

→ Accuracy

$$A = \frac{TP + TN}{N}$$

* Total no. of true ~~pos~~ +ve labels = TP + FN

* Total no. of true -ve labels = TN + FP

→ Sensitivity

effectives of a classifier to identify true +ve labels.

$$Se = \frac{TP}{TP + FN}$$

Specificity

↳ Effectiveness of a classifier to identify -ve labels.

$$Sp = \frac{TN}{FP+TN}$$

↳ Both Se and Sp lie b/w 0 and 1, 1 is an ideal value for each of them

↳ Balanced accuracy

$$BA = \frac{(Sensitivity + specificity)}{2}$$

↳ Prevalence

↳ How often does the yes condition actually occur in our sample.

$$P = \frac{TP + FN}{N}$$

↳ Positive predictive value.

↳ Proportion of correct results in labels identified as +ve.

$$PPV = \frac{(\text{sensitivity} * \text{prevalance})}{((\text{sensitivity} * \text{prevalence}) + ((1 - \text{specificity}) * (1 - \text{prevalence}))}$$

↳ -ve prediction value:

↳ proposition of correct results in labels identified as -ve.

$$NPV = \frac{\text{sepecificity} * (1 - \text{prevalence})}{((1 - \text{sensitivity}) * \text{prevalence}) + ((\text{specificity}) * (1 - \text{prevalence}))}$$

Detection Rate:

$$DR = \frac{TP}{N}$$

Detection prevalence

↳ prevalence of predicted ~~values~~ events

$$DP = \frac{TP + FP}{N}$$

↳ the Kappa statistic (or value) is a metric that compares an observed accuracy with an expected accuracy (random chance)

$$Kappa = \frac{\text{observed accuracy} - \text{expected accuracy}}{1 - \text{expected accuracy}}$$

ROC:

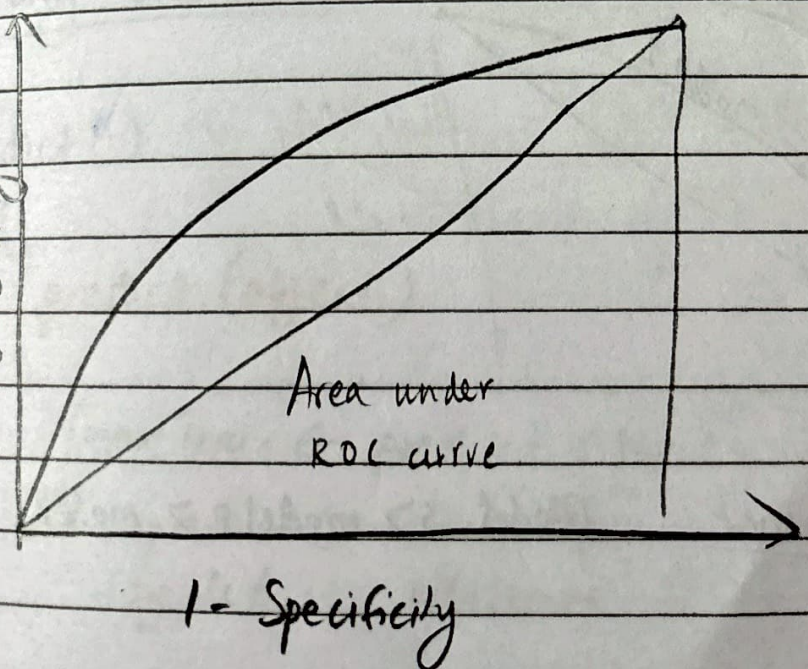
↳ Receiver Operating Characteristics

↳ originally developed and used in signal detection theory.

↳ ROC graph

- Sensitivity as a function of specificity

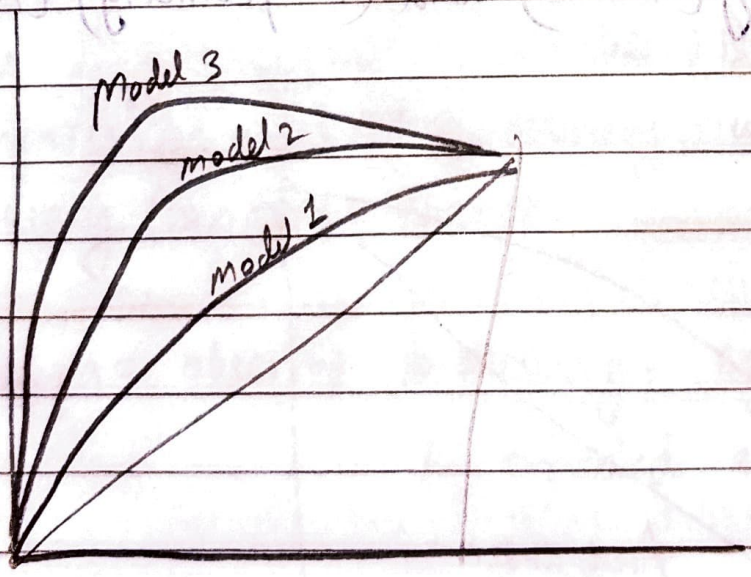
↳ sensitivity (Y-axis) and (1-specificity) (X-axis)



↳ ROC can be used to see the classifier performance at different threshold levels (0 to 1)

↳ AUC – Area under the ROC

- Area = 1 – perfect test
- Area = 0.5 – worthless model
- 0.90 – 1 → excellent
- 0.80 – 0.90 → good
- 0.70 – 0.80 = fair
- 0.60 – 0.70 = poor



Performance: Model 3 > model 2 > model 1

Logistic Regression Implementation in R:

rm(list = ls()) → clear the environment

library(caret) → for confusion matrix

generalized linear model

glm(formula, data, family)

↳ a description of the error distribution and link function to be used in the model

→ Finding the odds:

predict()

↳ predict(object)

logistTrain ← predict(logistfit, type='response')

- predict() with type='response' gives probabilities by default otherwise it returns log(odds)

plot(logistrain

> tapply(logsTrain, crashTest_1$(ar type, mean)

→ for test data:

logisPred ← predict(logsfit, newdata = crashTest_1_TEST, type = 'response')

plot(logsPred)

→ Results:

↳ classify whether the test point is Hatchback / SUV by setting a threshold.

crashtest_1_TEST[logisPred <= 0.5, "LogisPred"] ← "Hatchback"
crashTest_1_TEST[logisPred > 0.5, "LogisPred"] ← "SUV"

→ Confusion Matrix:

confusion Matrix (table (CrashTest_TEST [,7], CrashTest_TEST [,6], positive = 'Hatchback'))