IAML - Week 3

Antonio León Villares

October 2021

Contents

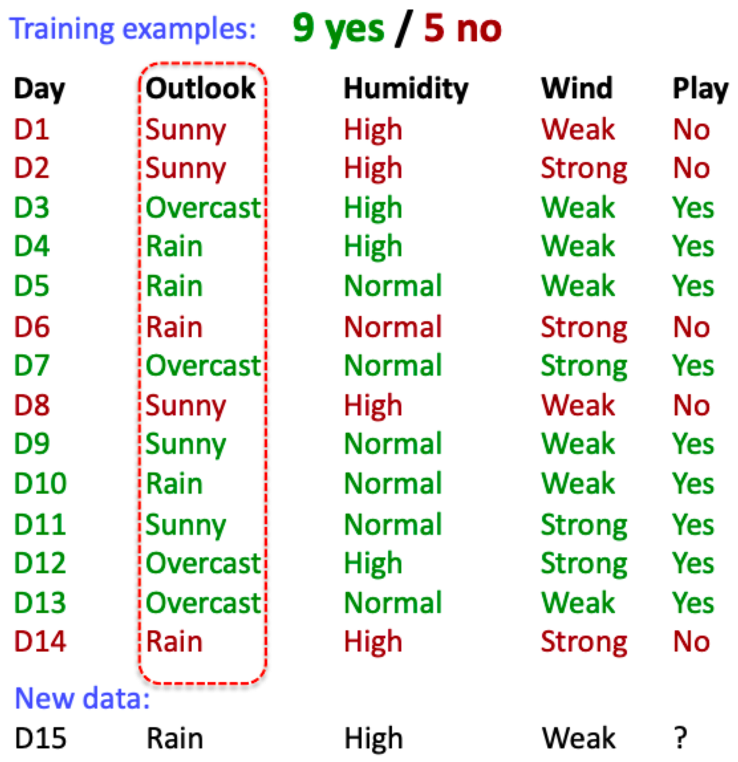
Here is a great set of videos about Decision Trees.

1 Decision Trees

- Decision trees are classifiers built by splitting data based on their features
- The aim is to obtain subsets of very similar data, leading to a tree that can do classification
- Entropy (more exactly Information Gain) is used to split data based on attributes to ensure high purity splits
- Decision trees can be used on continuous attributes by setting bounds ($\geq x$)

1.1 Decision Trees as Classifiers

- **What is a decision tree?**
  - a decision tree is a **classifier**
  - it takes data, which is organised by **attributes**
  - it generates a tree, in which each node splits into branches, based on the different values of an attribute
- **How is a decision tree a classifier?**
  - by training the decision tree, we are capable of splitting the data based on a unique set of attribute values
  - the aim is to have as the leaves the instances of the data which all have the same label. If this is the case we have a **pure** subset
  - when a new instance of the data is given, if we traverse the decision tree, we should arrive at a leaf node, which will give us the classification
  - ultimately, a decision tree is attempting to **understand** what combination of attributes lead to certain outcomes
  - ***Example***:



Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?

Figure 1: This is data representing whether I play tennis. It includes attributes, such as the day in which I played, the outlook, how humid it was or whether it was windy. These are all the **attributes**. Each **attribute** has a set of possible values. For example, the attribute *Outlook* can be either sunny, overcast or rain.

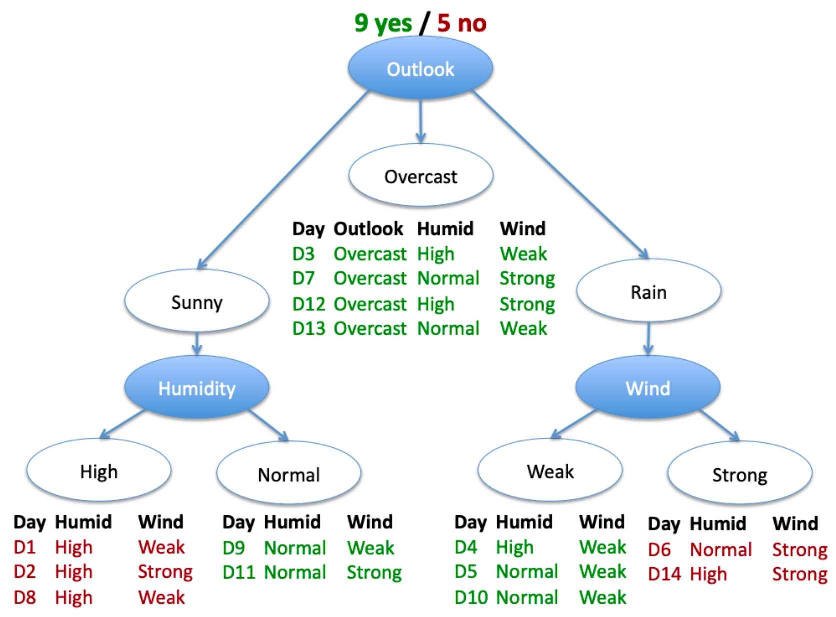


Figure 2: This is *one* possible decision tree (many are possible, depending on the attribute that we choose to split on) for the data above. Notice, there is a subset of data associated with each node. For example, the bottom left node corresponds to the subset of the data in which the outlook was sunny, and the humidity was high. With the current split, we are capable of obtaining **pure** subsets: each subset contains data in which I (exclusively) either played or didn't.

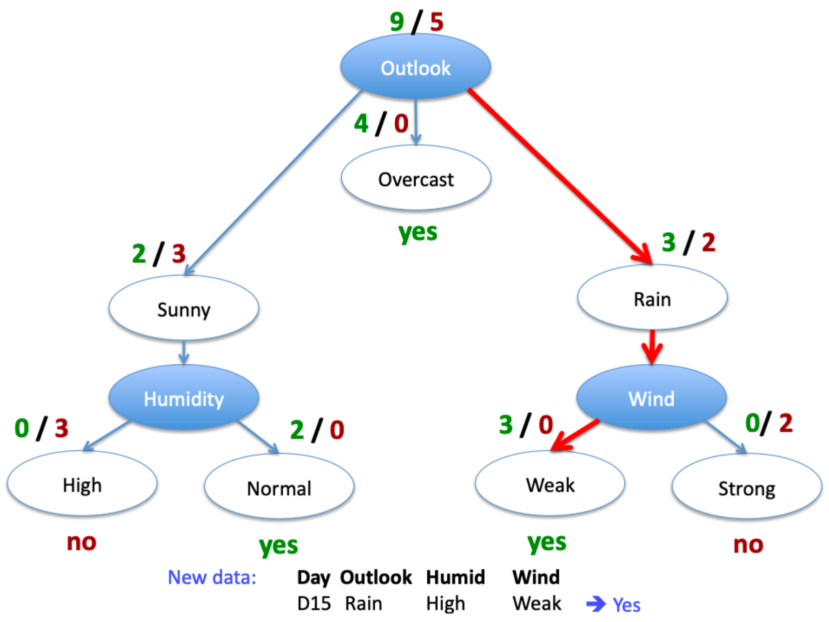


Figure 3: Based on the above, the following will be our decision tree. Once we obtain a pure subset, we can stop adding nodes, as all our test data has the same outcome at the given node. We label the leaf node with "yes" or "no", based on whether we played. If we then give it some new data, we just need to traverse the tree, and see what the outcome is. Thus, if the outlook was rain, the humidity was high, and the wind was weak, the decision tree predicts that I do play.

- **Why do we keep track of the counts at each node?**
  - notice, at each node we are keeping track of the number of data points in the subset which was labelled as either "yes" or "no"
  - for example, the subset of the data in which the outlook was sunny is comprised of 5 data points: 2 in which I played tennis, and 3 in which I didn't
  - we do this to allow **pruning** of the tree
  - for example, if we decide to prune the humidity node, and don't split on it, then any new data in which the outlook was sunny would be classified as a "no", as in 3 out of the 5 data instances I didn't play tennis when it was sunny

1.2 The ID3 Algorithm

- **What is the ID3 Algorithm?**

  - a recursive algorithm used to build decision trees

- **What are other Decision Tree building algorithms?**
  - beyond ID3, we also have:
    - *CaRT*, developed independently of ID3, uses different metrics
    - *C4.5*, developed by the same person who developed ID3, it allows non-categorical data for classification

- **What are the step in the ID3 Algorithm?**
  - the pseudo-code for the algorithm is:

Algorithm 1 ID3

```
procedure SPLIT(node, {examples})
    $A \leftarrow$ best attribute to split the $\{examples\}$
    $childNodes \leftarrow []$
    $childSubsets \leftarrow []$
    for value v in A do
        $childNodes.insert(v)$
        $splitExamples \leftarrow$ get $\{examples\}$ in which $A = v$
        $childSubsets.insert(splitExamples)$
    end for
    for i in range(len(childNodes) do
        if childSubsets[i] is pure then
            STOP
        else
            $Split(childNodes[i], childSubsets[i])$
        end if
    end for
end procedure
```

  - to start it off `node` will be the root node, and {`examples`} will be the whole data.
  - at each step, we pick the best attribute; then over all possible values, we create a new child node, and create a subset of the data in which the data had a specific value. Then, we recursively call split if the subset of data is **not** pure
  - for example if `A` were *Outlook*, the values of `v` would be Sunny, Overcast and Rain

1.3 Entropy, Information Gain and Information Gain Ratio

- **In ID3, how do pick the “best” attribute to split on?**

  - we can define "best" attribute as that attribute which, when split on, produces "purer" subsets

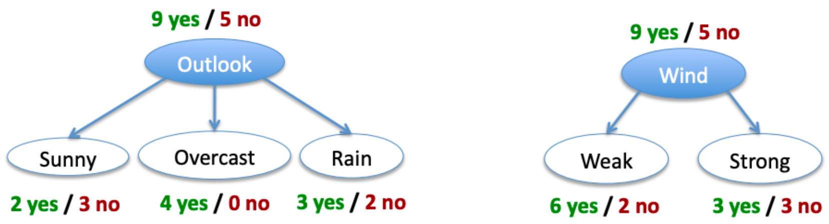


Figure 4: If we need to choose to split between *Outlook* and *Wind*, we can think that *Outlook* is better in 2 ways: firstly, it leads to an immediate pure subset; secondly, *Wind* leads to a subset with a 50-50 split; this is not helpful, as it doesn't lead to effective decision-making.

  - ultimately, we seek some way of evaluating whether some split provides us with information to guide our decision
  - a pure subset gives us 100% certainty, whilst a 50-50 split leaves us completely uncertain
  - we also a require a measure which is **symmetric**: 4 yes, 0 no is equally pure as 0 yes, 4 no

- **What is entropy?**
  - **entropy** is a measure that we can use to evaluate the "purity" of a subset
  - intuitively, from physics, entropy measures "disorder", so a higher entropy corresponds to more disorder (impurity)
  - it is a value which ranges from 0 to 1, with 0 indicating a pure subset, and 1 indicating a 50-50 subset
  - $$H(S) = -p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)}$$

    where:
    * **S**: the subset on which we calculate the entropy
    * $\boldsymbol{p_{(+)}}$: proportion of positives ("yes") in S. More generally, one of the possible outcomes of our prediction.
    * $\boldsymbol{p_{(-)}}$: proportion of negatives ("no") in S. More generally, the other of the possible outcomes of our prediction.

- if we have a 50-50 subset, $p_{(+)} = p_{(-)} = 0.5$, so:

$$\begin{aligned} H(S) &= -p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)} \\ &= -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} \\ &= \frac{1}{2} + \frac{1}{2} \\ &= 1 \end{aligned}$$

- if we have a pure subset, $p_{(+)} = 1, p_{(-)} = 0$ (or viceversa), so:

$$\begin{aligned} H(S) &= -p_{(+)} \log_2 p_{(+)} - p_{(-)} \log_2 p_{(-)} \\ &= -1 \log_2 1 - 0 \log_2 0 \\ &= 0 \end{aligned}$$

  (Please do ignore the fact that we actually find it acceptable to assume $0 \times \log_2 0 = 0$)

- **What does entropy have to do with computer bits?**
  - in lectures, they described entropy as the number of bits required to be certain about a classification.
    - if you have a 50-50 split, you need 1 bit to be certain (the bit telling you the outcome of the classification)
    - if you have a pure subset, you need 0 bits to be certain, as the subset is pure
    - entropy thus gives you fractional bits to measure uncertainty: the less bits you need to be certain, the purer the split

  This idea of bits can be generalised for multiclass systems: if you have 4 possible classes, you might need at most 2 bits if you get a 50-50 split. For more on information entropy, see this video by Khan Academy

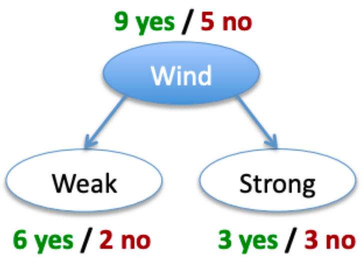
- **What is information gain?**
  - **entropy** allows us to discern the purity of a subset
  - when deciding on which attribute to split on, we need to consider the entropy of all the subsets which it produces
  - the **information gain** is a measure which allows us to pick the attribute that, when split, leads to the purest subsets - this is precisely our "best" attribute
  - informally, the higher the information gain, the more our uncertainty decreases - that is, it measures how much entropy decreases by splitting on an attribute

- indeed, by splitting we reduce entropy, as we can never actually increase the impurity of a subset

- **How is information gain computed?**
  - for information gain, we need to consider a subset S of the data, and an attribute A, where A will be the attribute used to generate subsets from S
  - to calculate information gain:

$$Gain(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

    where:
    * v is one of the possible values of attribute A
    * S_v is the subset of S in which we observe attribute A have value v
    * $\sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$ is a **weighted average** of entropies of the child nodes
  - ***Example***:



- **How do we use information gain to decided on the attribute to split?**
  - we pick the attribute with the **highest** information gain:

$$\max_{a \in A} Gain(S, a)$$

    The entropy for $S = Wind$ is:

$$H(S_{Wind}) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.94$$

    If we split on $Wind$, then:
    * the entropy resulting from the subset in which the $Wind$ is $Weak$ is:

$$H(S_{Weak}) = -\frac{6}{8} \log_2 \frac{6}{8} - \frac{2}{8} \log_2 \frac{2}{8} = 0.81$$

      This is expected, as 6 vs 2 is much more pure than 9 vs 5

    * the entropy resulting from the subset in which the *Wind* is *Strong* is:

$$H(S_{Strong}) = 1$$

Thus, it follows that:

$$Gain(S_{Wind}, Wind) = 0.94 - \left(\frac{8}{14}0.81 + \frac{6}{14}1\right) = 0.049$$

- **How is information gain used to build the Decision Trees?**
    - it is constantly used in the recursive step to build a node
    - once we split on a certain attribute, we no longer split on it
- **What is the main issue with information gain?**
    - since information gain is designed to detect purity, it will be **biased** towards attributes with many values
    - the more values, the more likely it is for a pure subset to appear (more values = more specific)

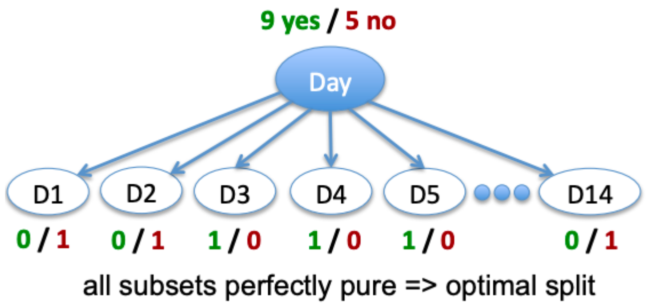


Figure 5: If we split based on the days attribute, then each unique observation corresponds to a unique day, which will be seen as a perfect split. However, this is useless when classifying.

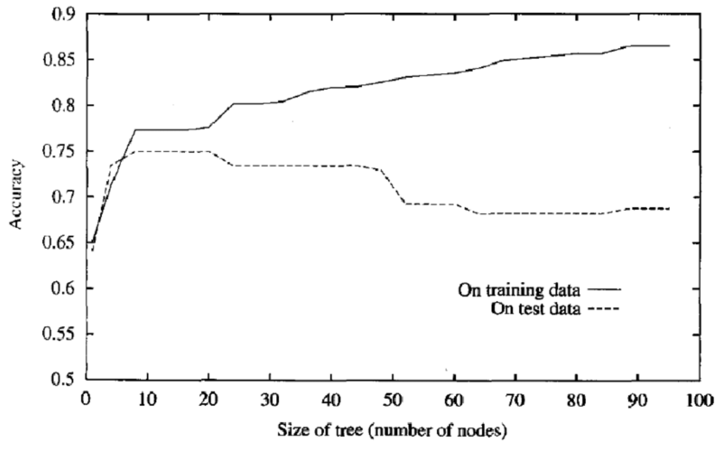
- **How does gain ratio solve the problems of information gain?**
    - the **gain ratio** is a way of penalising those attributes which have many values
    - to compute it:

$$SplitEntropy(S, A) = -\sum_{v \in Values(A)} \frac{|S_v|}{|S|} \log \frac{|S_v|}{|S|}$$

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitEntropy(S, A)}$$

1.4 Dealing With Overfitting

- **How is overfitting an issue for decision trees?**
  - since the ID3 algorithm splits until we reach a pure set, it can be the case that each leaf node eventually contains a singleton subset (as singleton subsets are pure)
  - in other words, it can reach 100% accuracy **for the training dataset**
  - this is clearly an issue for new data which doesn't resemble the training data



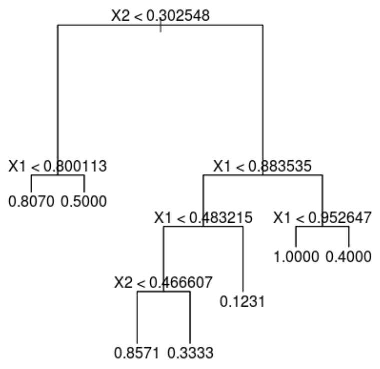
- **How can we fix overfitting for decision trees?**
  - 2 main methods:
    1. **Statistical Significance**: stop splitting when the subsets become "too small" (i.e when the split is no long statistically significant). For example:
       * if a set has 2 items, a non-statistically significant split would be one that splits the 2 items into 2 subsets - this is like a coin toss, it doesn't add "value" to our decision
       * if we have a set of 100 items, and upon the split we get 2 subsets, each of size 50, one positive and one negative, this **is** statistically significant (a lot of the data "repeats" in its label, it indicates underlying structure)

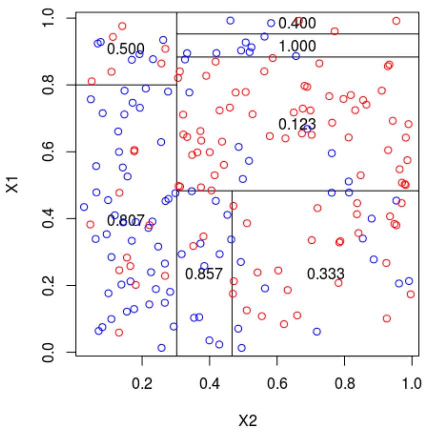
       This is not a particularly effective method nonetheless.
    2. **Validation Set and Pruning**: split your training data into 2: a training set and a validation set. Train a **full** decision tree on the training set. Using the validation set, you can compute the accuracy of the decision tree. For each node n, compute the accuracy of the tree by (pretend) removing n. Prune the node

with the greatest accuracy improvement, and repeat. Stop once the accuracy will no longer increase.

1.5 Decision Trees For Continuous Attributes

- **How can we adapt categorical decision trees to handle continuos attributes?**
  - if we deal with real valued attributes, we can do splits based on some threshold value (for example, split if $temperature > 20°C$)
  - this leads to a decision boundary in which the xy plane is split by straight lines
  - it also allows us to split on one single attribute at different levels of the tree (for example, at level 1 we can split if $temperature > 20°C$, and at level 4 we can split if $temperature \leq 37.8°C$)
- **How do you decide the thresholds when using Decision Trees for continuous attributes?**
  - we can train the thresholds in a similar way as to how we train on the attribute for which we split
  - we compute the threshold which allows us to best split the data. This can also be evaluated with Information Gain.
  - more on this StackOverflow Post





1.6 Decision Trees For Multiclass Classification

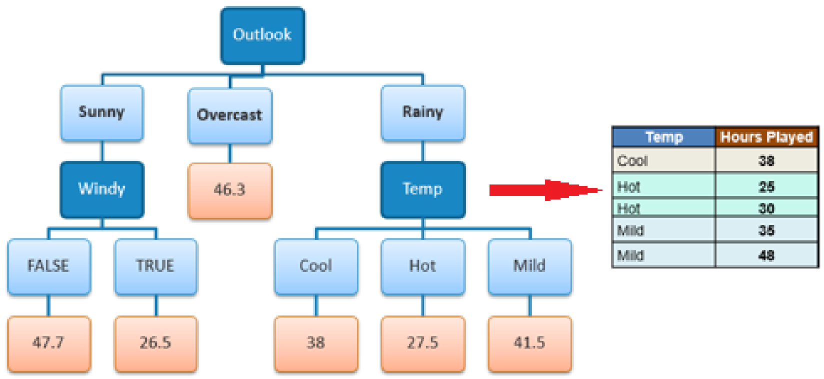
- **How can we adapt decision trees to handle multi-class classification?**

- Decision Trees work in the same way for binary or multi-class classification, the only difference being how we compute the information entropy:
$$H(S) = -\sum_{c \in Classes} p_c \log_2 p_c$$

1.7 Decision Trees For Regression

Here is a more in detail (includes examples, calculations) view of how regression can be adapted for Decision Trees.

- **How do regression and classification differ for Decision Trees?**
  - classification returns classes, whilst regression returns a real number
  - without classes, entropy/gain can't be computed
- **How are splits calculated for Decision Trees when using regression?**
  - instead of entropy, we split based on the **variance** of the subsets
  - the aim is to put data instances together in subsets, such that the variance of the subsets (with respect to the variable we are trying to predict) is minimised
  - thus, we can think that at the leaf nodes, we have the subsets with minimal variance
  - for example, if we are trying to predict stock prices, we can have a leaf node corresponding to all the data instances with low prices, and another leaf node with data instances with high prices
- **How is the output of regression computed using Decision Trees?**
  - each data point has a real number associated to the feature we are trying to predict
  - we can take the average of these for each subset at the leaf nodes, and output that
  - alternatively, we can fit a linear model to the subset data

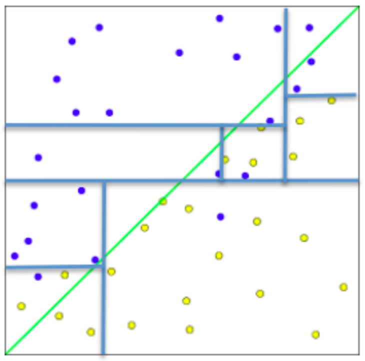


Temp	Hours Played
Cool	38
Hot	25
Hot	30
Mild	35
Mild	48

Figure 6: Here, we try to predict how much time we played. Notice for example, that the subset associated with the *Hot* node has an average of 27.5 which is taken as the output of the regression model.

1.8 Evaluating Decision Trees

- **What are the pros of using Decision Trees?**
  - *highly interpretable*: humans can look at decision trees and immediately understand why a certain classification was made
  - *irrelevant attributes ignored*: they would have gain close to 0, so wouldn't be considered for splitting
  - *handle irrelevant data*
  - *compact*
  - *fast to test*: most of the time taken is in training; once we have a Decision Tree the complexity of getting a classification is $\mathcal{O}(depth)$, and the depth is typically much less than the number of attributes; other classifiers tend to be linear in the number of attributes
- **What are the cons of using Decision Trees?**
  - *non-optimal*: Gain is a greedy heuristic, therefore ID3 only picks locally optimal splits; might lead to non-optimal trees, but searching all possible trees is unfeasible (exponential)
  - *axis aligned splits*: since splits are done in the xy axes, data which is diagonal or non-linear will be harder to classify



1.9 Random Forest Algorithm

- **What is a Random Forest?**
  - consists on training many decision trees, which build the forest
  - a classification is made by passing new data through all the trees, and doing a majority vote on which class was predicted the most
- **How do we build a Random Decision Forest?**
  1. original data set is S, we want K decision trees
  2. pick K mutually exclusive, random subsets of S. Call one such instance S_r
  3. use ID3 to build a tree T_r using S_r. However, when splitting we don't consider all attributes, but rather a random subset of all attributes. Gain is computed using S_r, not S
  4. repeat for $r \in [1, K]$

  This algorithm ensures that each Decision Tree is very decoupled

2 Generalisation and Evaluation

- Training, Testing and Generalisation are the amin types of error
- Testing error is used to estimate generalisation error, via confidence intervals
- Validation sets allow us to gauge the predictive capacity of our model
- Measuring the performance of a classification algorithm can be done via accuracy, recall, precision, miss rate, or false alarm rate
- Measuring the performance of a regression algorithm can be done via root mean squared error, median absolute deviation or correlation coefficients

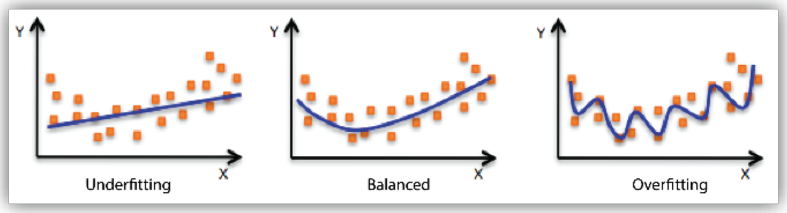
2.1 Generalisation in Machine Learning

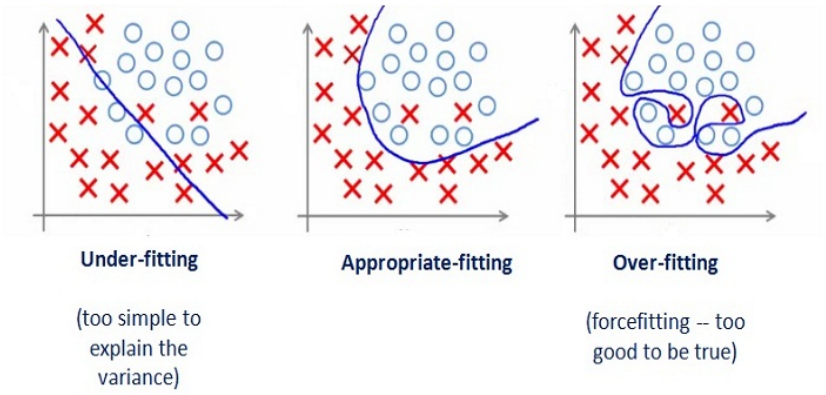
- **Why is generalisation important in Machine Learning?**
  - we build learning algorithms based on training data
  - the idea is for our model to be **general**, so as to obtain good performance on new/future data
  - in particular, being good on training data is not that *useful*: it is easy to be perfect in training, but that doesn't necessarily translate to high performance in future data

2.2 Overfitting and Underfitting

- **When do we say that a predictor overfitting?**
  - when we learn a model that is too **complex/flexible**, and so, is too adapted to the idiosyncrasies of the training data
  - in particular, may take random noise as patterns to learn
- **How do we formally describe when a model is overfitting?**
  - we say a predictor F overfits data, if we can find some other predictor F', and:
    * $E_F(train) < E_{F'}(train)$
    * $E_F(test) > E_{F'}(test)$
  - in other words, there are other predictors which fare better on test data, even if they do worse in train data (compared to F)
- **What is undefitting?**
  - opposite to overfitting, we obtain a model which is too **rigid/simplistic**

- it can't pick up on the underlying patterns of data
- we say a predictor F underfits data, if we can find some other predictor F', and:
  - $E_F(train) > E_{F'}(train)$
  - $E_F(test) > E_{F'}(test)$





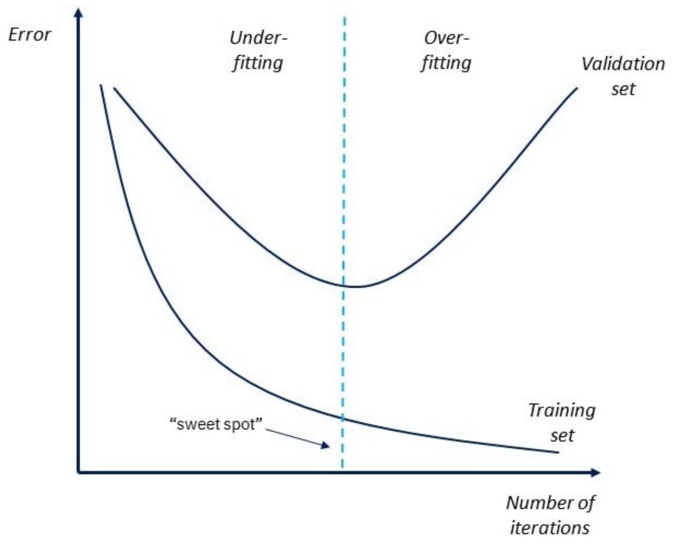


Figure 7: A model which underfits will have very high training/testing errors. A model which overfits will have very low training errors, but very high testing error. The key is to find a "sweet spot".

2.3 Types of Error: Training, Generalisation and Testing

2.3.1 Types of Error

- **What is training error?**
  - the error associated with what is predicted by the model, and the training labels:

$$E_{train} = \frac{1}{n}\sum_{i=1}^{n} ERROR\left(f_D(\underline{x}_i), y_i\right)$$

    where $ERROR$ is the error function specific to the task.

- **What is generalisation error?**
  - the error associated with predictions on future data
  - naturally, we don't know how the new data will look, nor what labels it will have
  - if we know the possible ranges of data and labels (for example, if data are the pixels of a 20x20 image, and labels are a number from

0 to 9), then:

$$E_{generalisation} = \int ERROR\,(f_D(\underline{x}), y)\, p(y, \underline{x}) d\underline{x}$$

with $p(y, \underline{x})$ being the probability of seeing y and $\underline{x}$ on the future data

- the issue is that we can't know p, so at most we can attempt to *estimate* the generalisation error

- **How can we minimise generalisation error?**
  - most learning algorithms have "knobs" which can be fine-tuned, depending on the specifics of the task
  - for example, if data is limited, we prefer a rigid predictor, as the data probably doesn't give an accurate overview of the real data
  - if we have a complex prediction problem, we prefer a more complex predictor, as it will be able to better pick up underlying patterns
  - examples of such knobs are:
    * *Regression*: order of the polynomial used
    * *Naive Bayes*: number of attributes used, σ^2
    * *Decision Tree*: number of nodes, pruning confidence
    * *kNN*: number of nearest neighbours

- **What is testing error?**
  - error associated with predictions on a testing set
  - the testing error is what we use to *estimate* the generalisation error
  - the testing set is taken from the training data, and never seen by the predictor during training

2.3.2 Estimating Generalisation Error

- **How good is testing error in estimating generalisation error?**
  - as the size of the testing data increases, it becomes more representative of the general population
  - if the testing set is an unbiased sample from $p(y, \underline{x})$, then by the law of large numbers:

$$\lim_{n\to\infty} E_{test} = E_{generalisation}$$

- **How can we quantify the "goodness" of testing error as an estimate for generalisation error?**

- we can use **confidence intervals** to give a bound on how much the generalisation error can differ from the testing error
- for example, if we define a 95% confidence interval $E_{test} \pm \Delta E$, we would expect 95% of future data to have an error within this interval

- **How do we compute confidence intervals for generalisation error?**
  - let E is the true error rate (probability of predictor misclassified), and let E_{test} be the estimate for E
  - if we take n instances, the number of misclassified instances can be described by a binomial distribution, with:

$$\mu = nE$$

$$\sigma^2 = nE(1-E)$$

  - the error is then given by $\frac{\#misclassified}{n}$, so by the law of large numbers, the error will follow a Gaussian Distribution:

$$E_{future} \sim N\left(E, \frac{E(1-E)}{n}\right)$$

  - thus, the p% confidence interval will be:

$$E \pm \sqrt{\frac{E(1-E)}{n}} \times \Phi^{-1}\left(\frac{1-p}{2}\right)$$

    where Φ^{-1} represents the inverse of a Normal Distribution

2.4 Validation Methods in Machine Learning

2.4.1 Validation Sets

- **Which 3 sets are used when developing a Machine Learning Model?**
  1. **Training Set**: to train the model
  2. **Validation Set**: pick algorithm (which one is best for the task?) or fine tune the parameters ("knobs") of the model
  3. **Testing Set**: estimate future error rate

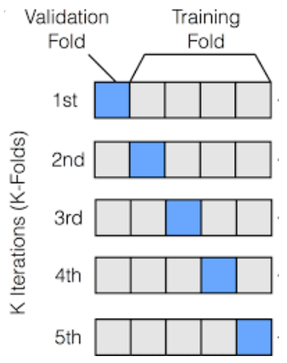
- **What are the issues with using a naive validation set?**
  - naively, from the training data, we split it into 3 subsets, 1 for training, 1 for validating, 1 for testing
  - this is an issue, as dividing the data up comes at a cost of model performance:

    * accurate estimates of generalisation error can only occur if the testing set is large
    * an accurate model can only occur if the training set is large

  However, $n_{train} + n_{test} = constant$, so both can't be large

2.4.2 k-Fold Cross-Validation

- **What does k-Fold Cross-Validation consist on?**
  - split the training data into k parts
  - use $k-1$ of the parts to train a model, and test it on the remaining "fold"
  - repeat, but this time using a different fold for testing
  - the estimate for the generalisation error can then be taken as the average error over all k of the training instances (overall we will have trained k different models)
  - lastly, use 100% of the data to train the final model
- this is "allowed", since we are never using the same subset to both train and test



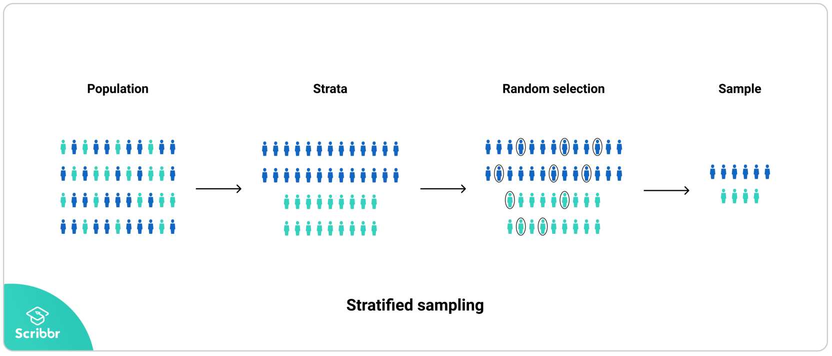
2.4.3 Leave-One-Out Cross-Validation

- if we have n data points, Leave-One-Out Cross-Validation performs n-Fold Cross Validation: create n folds, train on $n-1$ of them, and test on the remaining
- in general, this will train a better classifier (a lot of training data)
- however:
  - it is computer intensive

  - there are issues with certain class distribution. For example, if we have 2 classes, A and B, and data is 50-50 split between them. Imagine we train a majority classifier: it classifies based on which class appeared the most in training. Notice, if we use Leave-One-Out, then we will get 100% error. After all, if a class is most popular in training, it must be because the test instance must be of the opposite class (for example, if we have 100 samples, and the test sample is A, then we will have 50 B, and 49 A). Thus, the classifier will always pick the most popular class, which will never be the class of the training instance.

2.4.4 Stratified Sampling

- **What is an underlying issue of k-Fold Cross-Validation/LOOCV?**
  - in doing the random splits, it is possible that training and testing sets are unbalanced in terms of the classes present in each of the sets
  - classes in different proportions across sets might influence the performance of a model
- **What is stratified sampling?**
  - a method to ensure that class distribution across trianing/testing folds is balanced
  - can be achieved by:
    1. split each class into k parts
    2. the ith fold of the data is done by joining the ith folds of each of the class folds



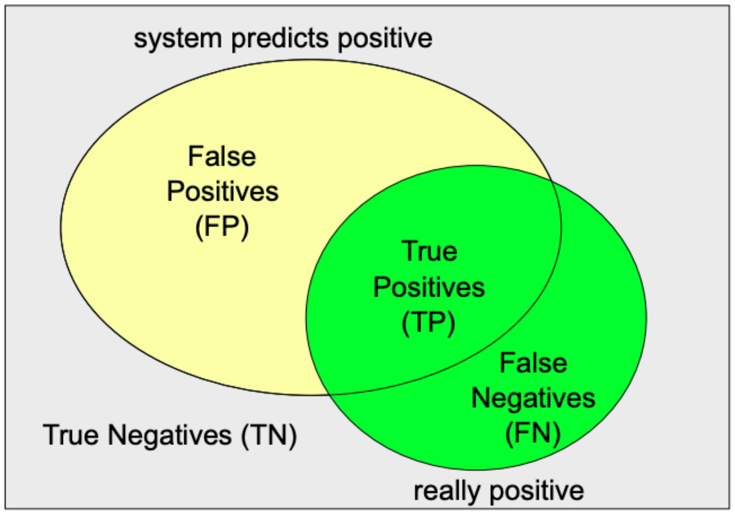
2.5 Evaluating Classification

- **How can we evaluate the "goodness" of a Machine Learning model?**

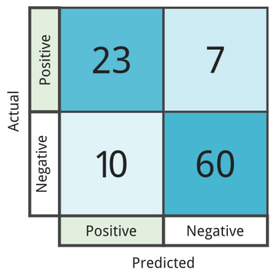
- we need to be able to **evaluate** a model, in order to decide what the best algorithm for a given task is
- depending on the task, we might ask different questions:
  * **Classification**: how often do we classify something right/wrong?
  * **Regression**: how close are we to what we are trying to predict?
  * **Unsupervised**: how well do we describe our data?

2.5.1 False Positives and False Negatives

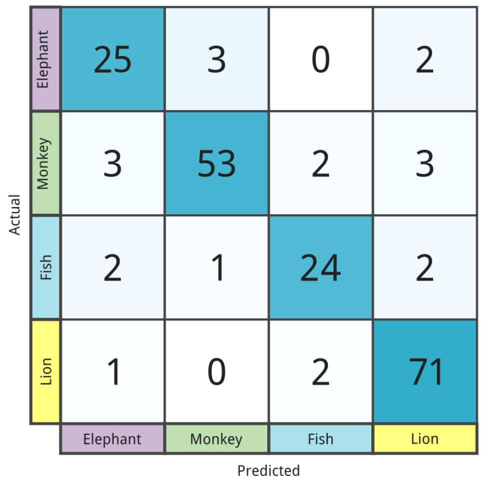
- **What is a false positive?**
  - in binary classification, a data point which is labelled as negative, but is classified as positive
- **What is a false negative?**
  - in binary classification, a data point which is labelled as positive, but is classified as negative



- **What is a confusion matrix?**
  - if we have n classes, an $n \times n$ grid, in which we include the number of data points which were classified in a certain way
  - the axes are the actual label of the point, and the classified label
  - it allows us to visually evaluate the prediction capability of a classifier
  - we seek large numbers in the main diagonal (indicating points being classified correctly)



Actual \ Predicted	Positive	Negative
Positive	23	7
Negative	10	60



Actual \ Predicted	Elephant	Monkey	Fish	Lion
Elephant	25	3	0	2
Monkey	3	53	2	3
Fish	2	1	24	2
Lion	1	0	2	71

2.5.2 Classification Error and Accuracy

- **How are classification error and accuracy related?**
  - classification error is the proportion of points which we predicted incorrectly:
$$\frac{error}{total} = \frac{FP + FN}{TP + TN + FP + FN}$$
- classification accuracy is the proportion of points which we predicted correctly:
$$1 - classification\ error = \frac{TP + TN}{TP + TN + FP + FN}$$
- **What are the issues associated with classification error/accuracy?**
  - quite useless if we are dealing with **unbalanced classes**
  - **Example**: if a Nobel Prize is awarded to 0.0001% of the population, a very accurate classifier would be one which predicts that no one gets the Nobel Prize, as that is 99.9999% accurate. As Humans, we would prefer a classifier which gets some predictions wrong (False Positives), but is capable of correctly predicting *who* gets the prize

2.5.3 Recall, Precision, Miss and False Alarm

- **What is the False Alarm Rate/False Positive Rate?**
  - the proportion of negative instances which were classified as positive (that is, the proportion of false positives, over all negatives):
$$\frac{FP}{FP + TN}$$

  - think of false alarm as in: we predict an earthquake will happen, but it actually doesn't, it was a false alarm

- **What is the Miss Rate/False Negative Rate?**
  - the proportion of positive instances which were classified as negative (that is, the proportion of false negatives, over all positives):

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

  - think of miss as in: we predict an earthquake won't happen, but it actually does, we missed the classification

- **What is Recall/True Positive Rate?**
  - the proportion of positive instances which were classified correctly as positives (that is, the proportion of true positives, over all positives):

$$\frac{TP}{FN + TP} = 1 - \textit{miss rate}$$

- **What is Precision?**
  - the proportion of true positives, out of all instances which we predicted as positive:

$$\frac{TP}{TP + FP}$$

- **Which of the measures above should we report about?**
  - it is meaningless to put them individually: it is easy to get 100% Recall, or 0% False Alarm
  - together, they provide an overview of the strnegths and weaknesses of the model
  - typical combinations include:
    * recall and precision
    * miss and false alarm
    * true positive and false positive rates

2.5.4 Classification Cost and Utility

- **When is single-value evaluation used?**
  - accuracy is not useful with unbalanced data, but it is useful since it is a (meaninful) single-value, unlike recall, precision, false/miss alarm
  - single values are useful in tasks such as competitive evaluation (is one better than the other?), or automatic learner optimisation

- sometimes there are domain-specific measures

- **What is detection cost?**
  - if we know the cost of a misclassification (i.e material costs derived from a false positive/false negative), we can compute the **detection cost**:
  $$C_{FP} \times \mathit{FP\ rate} + C_{FN} \times \mathit{FN\ rate}$$
  - it is a weighted average of the FP,FN rates
  - useful when evaluating event detection

- **What is F-Measure?**
  - the harmonic mean of recall and precision:
  $$\frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$
  - used in information retrieval (i.e search engines), as it gives a measure similar to accuracy, but without including false negatives

2.5.5 ROC Curve

- **Can you evaluate a system solely on performance?**
  - no, as the performance may be affected by other factors
  - for example, consider 2 predictors:
    * **Predictor 1**: TP = 50%, FP = 20%
    * **Predictor 2**: TP = 100%, FP = 60%

    it can be the case that both these predictors are the exact same model; the only difference being the **threshold** which we use to classify instances

- **How do thresholds affect error rates?**
  - classifiers train a function $f(x)$ and typically use $f(x) > t$ to define whether x is class A or B
  - thus, the threshold t will determine error rates
  - for example, in Naive Bayes, $P(spam|x) > 0.5$ we have a 0.5 threshold. If we have the exact same model, but using $P(spam|x) > 0.2$ we are likely to get more true and false positives

- **What is a ROC curve?**
  - a curve which illustrates the classification ability of a predictor as the threshold varies

    - it plots the true positive rate (proportion of positives correctly classified as positives) vs the false positive rate (proportion of negatives incorrectly classified as positive)

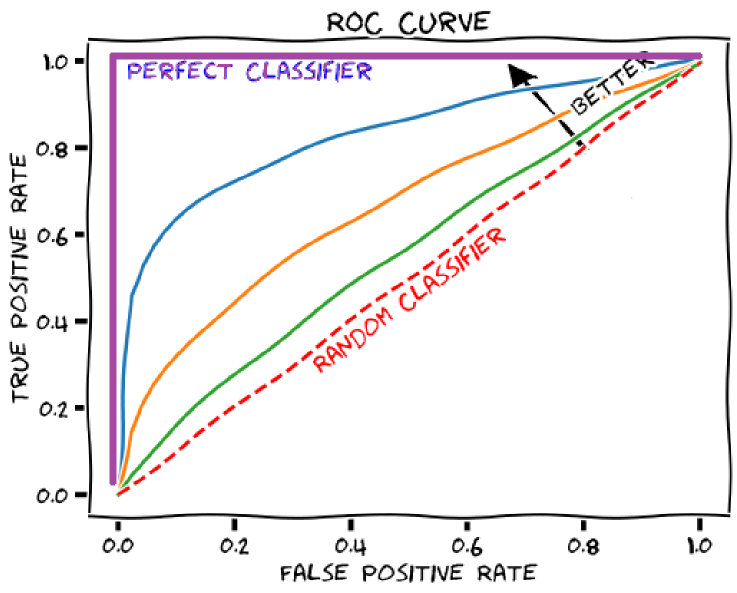


Figure 8: A diagonal line indicates a random classifier; the perfect classifier is thus because there is a point in which its true positive rate is 1 (always correctly classified positive instances as positive) and its false positive rate is 0 (never misclassifies a negative as positive).

2.6 Evaluating Regression

- **How is evaluating regression different from evaluating classification?**
    - in classification, we assign labels to data instances, and we hope that we are right
    - in regression, the output is almost guaranteed to never be the exact value
    - thus, to evaluate regression, instead of considering how often we are wrong, we compute *by how much* are we wrong

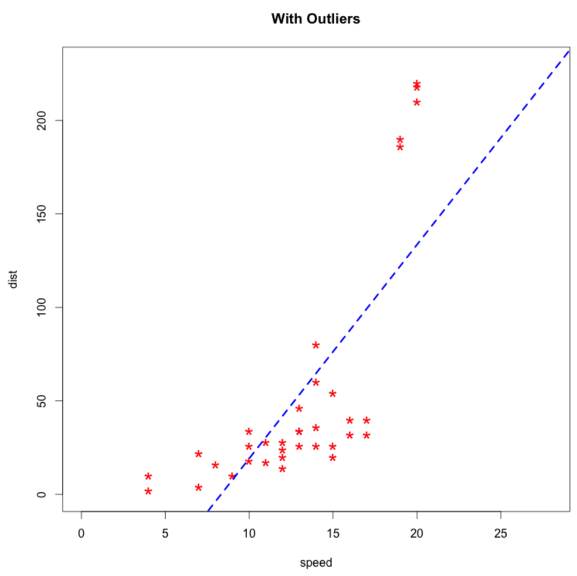
2.6.1 Root Mean Squared Error

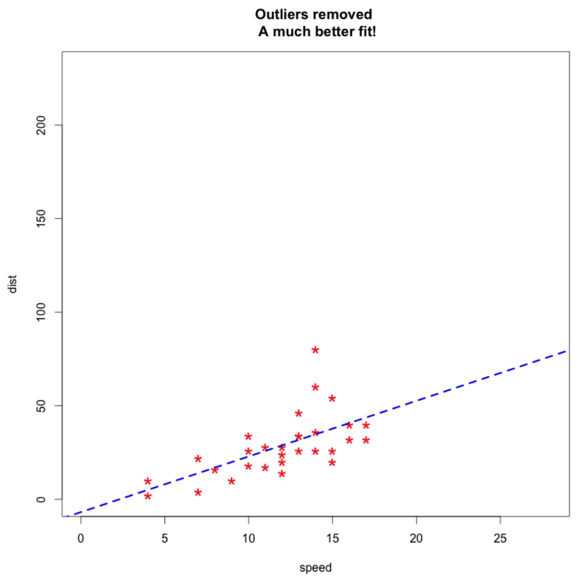
- **What is Root Mean Squared Error?**

  - the average, squared deviation from the true value:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(f(x_i) - y_i)^2}$$

- **How do outliers affect RMSE?**
  - RMSE is extremely sensitive to outliers
  - data with a small number of large errors can have the same MSE as data with many small errors:
    * 99 exact predicitons, 1 prediciton off by 10
    * 100 predictions off by 1
  - adding one outlier to the data can completely change the regression model if we use RMSE





- **How does mean/scale affect RMSE?**
  - mean can greatly affect the prediction
  - we can build a model which captures the pattern in the data, but has a different mean
  - then, this model will have a larger RMSE than a regression model which just predicts the mean of the data

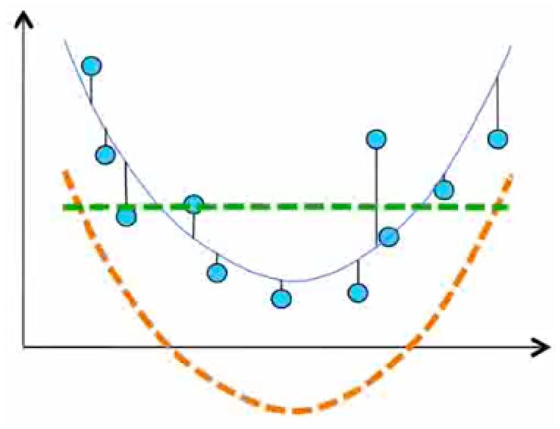


Figure 9: In the above figure, the orange line is good at capturing the essence of the data, but it is considered a worse model than the mean if we use RMSE

- **What is Relative Squared Error?**
  - an alternative to RMSE, which scales the RMSE by the RMSE of using a regression model which just predicts the mean of the data:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n} (f(x_i) - y_i)^2}{\sum_{i=0}^{n} (\mu_y - y_i)^2}}$$

2.6.2 Mean Absolute Error

- **What is Mean Absolute Error?**
  - like RMSE, but instead of squaring the differences, we consider the absolute value:

$$MAE = \frac{1}{n} \sum_{i=0}^{n} |f(x_i) - y_i|$$

  - by removing the squaring, MAE is less sensitive to outliers (one big error won't lead to the same value as many small errors)
  - in the same way as RMSE was sensitive to mean, MAE is sensitive to the median of the data

2.6.3 Median Absolute Deviation

- **What is Median Absolute Deviation?**
  - like MAE, but instead of averaging, we consider the median:

$$MAD = med\{|f(x_i) - y_i|\}$$

- extremely robust, as it ignores outliers
- can also define MAD^2
- however, unlike with the mean, the median is not differentiable, which makes minimising it hard

2.6.4 Correlation Coefficient

- **What is the Correlation Coefficient?**
  - a measure which is completely insensitive to mean and scale:

$$CC = \frac{\sum_{i=1}^{n}(f(x_i) - \mu_f)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(f(x_i) - \mu_f)^2 \sum_{i=1}^{n}(y_i - \mu_y)^2}}$$

  - informally, measures how well our prediction changes with y_i (i.e expect bigger $f(x_i)$ if y_i is larger)
  - the main issue is that completely unrelated data can have the same CC, so they would have the same; **it is always important to visualise data**

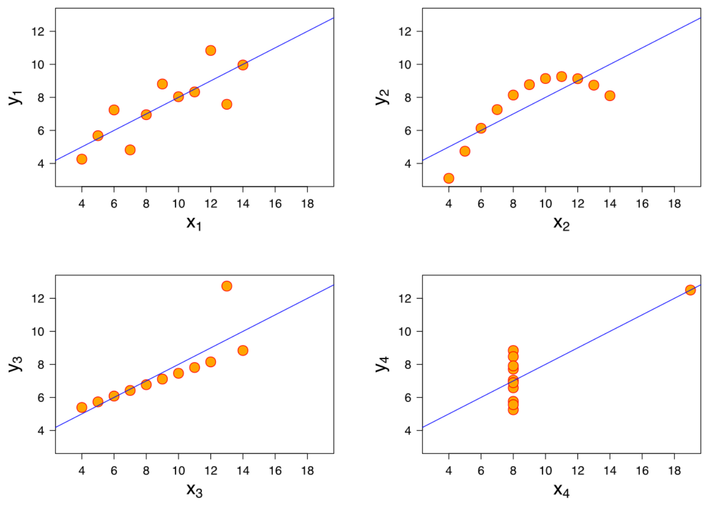


Figure 10: This is known as Anscombe's quartet, and shows the "danger" of relying solely on CC