Week 8

Data Mining and Knowledge Discovery

Dr John Evans
j.evans8@herts.ac.uk

University of Hertfordshire UH

Plan for today

Model Overfitting

Model Selection

Model Evaluation

Recap

- We discussed Hunt's algorithm.
- We saw measures of impurity:

$$
\begin{aligned}
Entropy = & \quad -\sum_{i=0}^{c-1} p_i(t) \log_2(p_i(t)), \\
Gini\ index = & \quad 1 - \sum_{i=0}^{c-1} (p_i(t))^2, \\
Classification\ error = & \quad 1 - max_i[p_i(t)],
\end{aligned}
$$

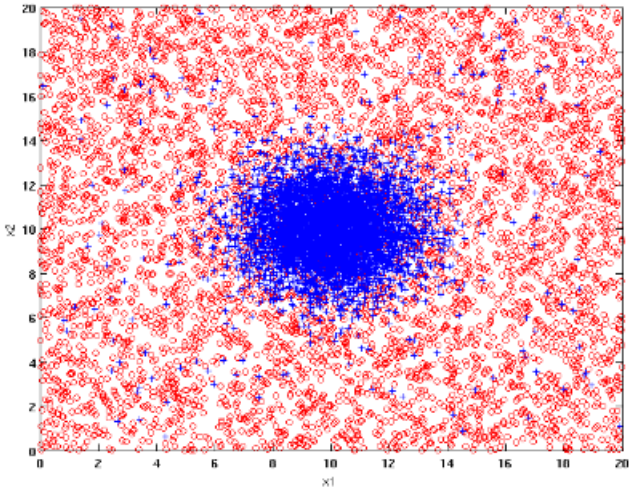
Model overfitting

- Last week, our attempts were to show the lowest error on the training set. However, in many cases a model may be an excellent fit for the training set, yet still show poor generalisation performance. We call this phenomenon, *model overfitting*.

Model overfitting

- Last week, our attempts were to show the lowest error on the training set. However, in many cases a model may be an excellent fit for the training set, yet still show poor generalisation performance. We call this phenomenon, *model overfitting*.
- We demonstrate this with an example. Consider the two-dimensional data set on the following slide in which we have two separate classes, represented as $+$ and $\circ$, each of which has 5400 instances
- All instances belonging to the $\circ$ class were generated from a uniform distribution, while the $+$ class is split. A Gaussian distribution centred at $(10,\ 10)$ with unit variance was used to generate 5000 instances, with the remaining 400 instances sampled from the same uniform distribution as the $\circ$ class.

Model overfitting example



Model overfitting example continued

- It is clear that the $+$ class can be mostly distinguished from the $\circ$ class by drawing a circle centred at $(10, 10)$.

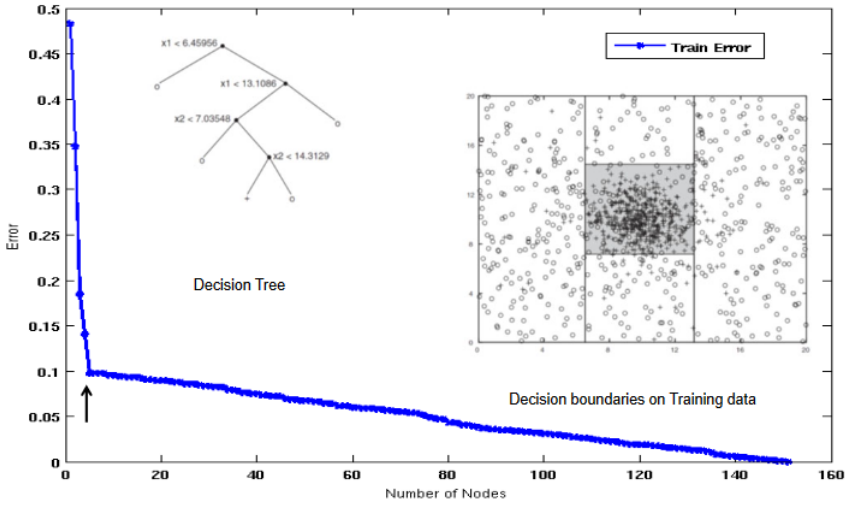
[1] In the pictures, the shaded rectangles represent regions assigned to the $+$ class.

Model overfitting example continued

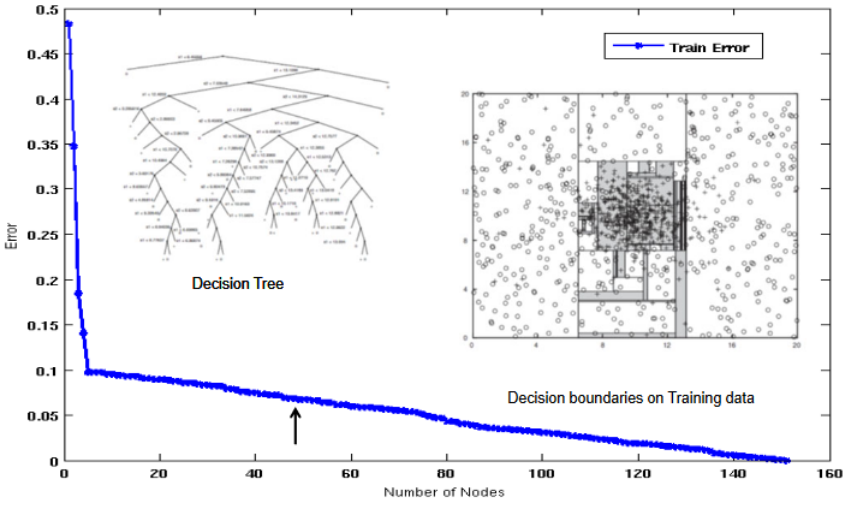
- It is clear that the $+$ class can be mostly distinguished from the $\circ$ class by drawing a circle centred at $(10, 10)$.
- To produce a classifer using this two-dimensional data set, we randomly sampled 10% of the data for training and used the remaining 90% to test.
- This training set can look quite representative of the whole data set and still cause problems.
- We use the Gini index as the impurity measure and construct decision trees of increasing sizes (number of leaf nodes), by recursively expanding a node into child nodes till every leaf node is pure. We can demonstrate this in the following pictures which show the case with 4 nodes and 50 nodes, respectively[1].

[1] In the pictures, the shaded rectangles represent regions assigned to the $+$ class.

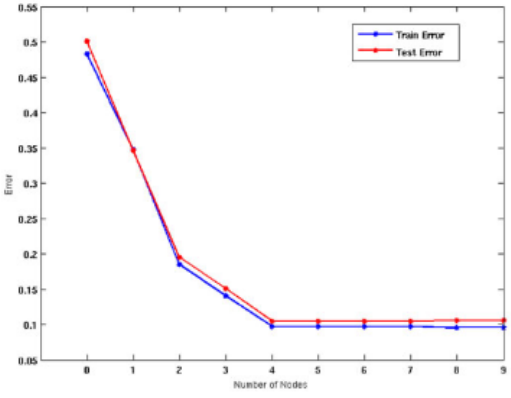
Model overfitting (4 nodes)

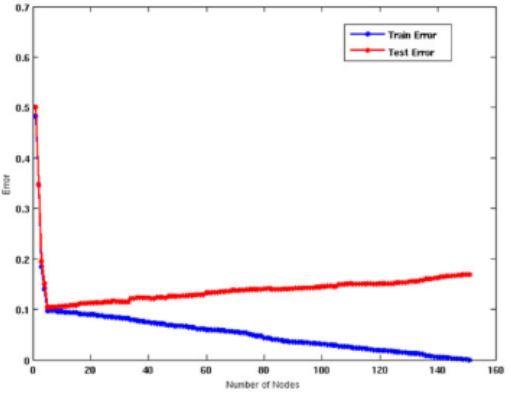


Model overfitting (50 nodes)



Which is better?





What do we notice?

- At first, both error rates are large. This is the case when the tree has only one or two leaf nodes and so the decision tree is too simplistic to fully represent the true relationship between the attributes and the class labels. We call this situation *model underfitting*.

What do we notice?

- At first, both error rates are large. This is the case when the tree has only one or two leaf nodes and so the decision tree is too simplistic to fully represent the true relationship between the attributes and the class labels. We call this situation *model underfitting*.
- As the tree size increases from 1 to 8 nodes, we observe two things:
  1. **Both error rates decrease.** This reflects the idea that larger trees are able to represent more complex decision boundaries.
  2. **The training and test error rates are quite close to each other.** This indicates that the performance on the training set is fairly representative of the generalisation performance.

What do we notice?

- At first, both error rates are large. This is the case when the tree has only one or two leaf nodes and so the decision tree is too simplistic to fully represent the true relationship between the attributes and the class labels. We call this situation *model underfitting*.
- As the tree size increases from 1 to 8 nodes, we observe two things:
  1. **Both error rates decrease.** This reflects the idea that larger trees are able to represent more complex decision boundaries.
  2. **The training and test error rates are quite close to each other.** This indicates that the performance on the training set is fairly representative of the generalisation performance.
- However, as we increase the size of the tree from 8 to 150, something unusual happens. The training error continues to steadily decrease until it eventually reaches zero, but now the test error rate ceases to decrease any further beyond a certain tree size. Indeed, it starts to increase.

Issues with training error and test error

- In this way, the training error rate greatly underestimates the test error rate once the tree becomes too large.
- This behaviour, which may seem counter-intuitive at first, can be attributed to the phenomena of *model overfitting*.

Issues with training error and test error

- In this way, the training error rate greatly underestimates the test error rate once the tree becomes too large.
- This behaviour, which may seem counter-intuitive at first, can be attributed to the phenomena of *model overfitting*.
- In the pursuit of minimising training error rate, we create an overly complex model that captures specific patterns in the training data but fails to learn the true nature of relationships between attributes and class labels in the overall data.

Applying these observations to our example

- In the 4 node tree, this fairly simple tree includes decision boundaries that provide a reasonable approximation to the ideal decision boundary, i.e. a circle centred around the Gaussian distribution at $(10,\ 10)$.
- The training and test error rates may still be non-zero, but they are very close to each other. This indicates that the patterns learned in the training set should generalise well over the test set.

Applying these observations to our example

- In the 4 node tree, this fairly simple tree includes decision boundaries that provide a reasonable approximation to the ideal decision boundary, i.e. a circle centred around the Gaussian distribution at $(10,\ 10)$.
- The training and test error rates may still be non-zero, but they are very close to each other. This indicates that the patterns learned in the training set should generalise well over the test set.
- In contrast, the decision tree of size 50 is much more complex and has complicated decision boundaries. In trying to cover one or true $+$ training instances, it has attempted to cover narrow regions in the input space. The $+$ instances in such regions are highly specific to the training set, as these regions are mostly dominated by $\circ$ instances in the overall data.
- In attempting to perfectly fit the training data, the decision tree starts fine tuning itself to specific patterns in the training data, thereby leading to poor performance on an independently chosen test set.

Factors influencing model overfitting (Limited training size)

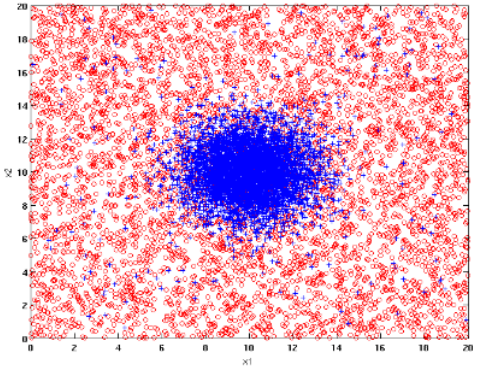
- A training set consisting of a finite number of instances can only ever provide a limited representation of the overall data.
- It is therefore possible that any patterns learned from a training set do not fully represent the true patterns in the overall data. This leads to model overfitting.

Factors influencing model overfitting (Limited training size)

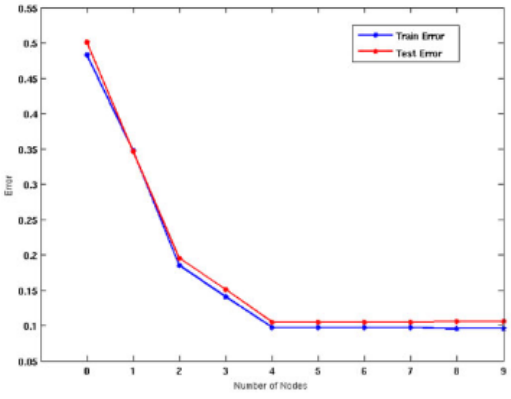
- A training set consisting of a finite number of instances can only ever provide a limited representation of the overall data.
- It is therefore possible that any patterns learned from a training set do not fully represent the true patterns in the overall data. This leads to model overfitting.
- In general, increasing the size of a training set (number of training instances), increases the chance that the patterns learned from the training set resemble true patterns in the overall data.
- Hence, the effect of overfitting can be reduced by increasing the training data size.

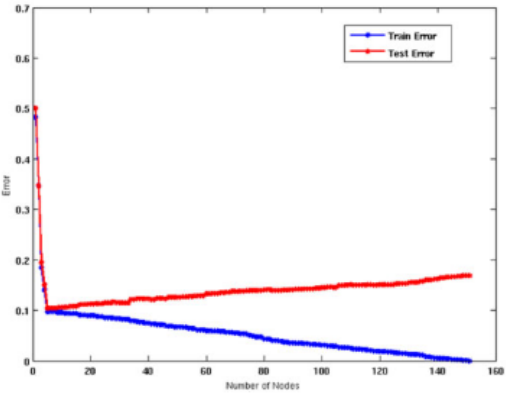
Example

Recall our data set from last time. We generated 5400 $\circ$ from a uniform distribution, 5000 $+$ from a Gaussian distribution centred at $(10,\ 10)$ with unit variance, and 400 $+$ from the same uniform distribution that generated the $\circ$ class.



Example (From last time)



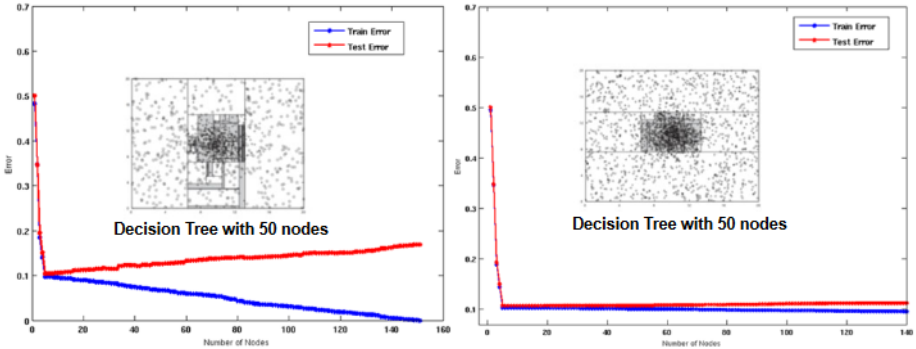


Example continued

Suppose now we use twice the number of training instance, i.e. we use 20% of the data for training (instead of just 10%). The rest is used for testing. Then we get the following:

Example continued

Suppose now we use twice the number of training instance, i.e. we use 20% of the data for training (instead of just 10%). The rest is used for testing. Then we get the following:



Using twice the number of data instances

Example (What does this show?)

- There are two major differences in the trends shown above from those seen earlier:
  1. Even though the training error rate decreases with increasing tree size in both cases, its rate of decrease is much smaller when using twice the training size.
  2. For a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size.

Example (What does this show?)

- There are two major differences in the trends shown above from those seen earlier:
  1. Even though the training error rate decreases with increasing tree size in both cases, its rate of decrease is much smaller when using twice the training size.
  2. For a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size.
- These training differences suggest that the patterns learned using 20% of the data for training are more generalisable than those learned using 10%.

Example (What does this show?)

- There are two major differences in the trends shown above from those seen earlier:
  1. Even though the training error rate decreases with increasing tree size in both cases, its rate of decrease is much smaller when using twice the training size.
  2. For a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size.
- These training differences suggest that the patterns learned using 20% of the data for training are more generalisable than those learned using 10%.
- The above image also shows the decision boundaries for a decision tree with 50 nodes, learned using 20% of data for training. In contrast to the tree of the same size learned using 10% data for training, we can see that the decision tree is not capturing specific patterns of noisy $+$ instances in the training set. Instead, the high model complexity of 50 leaf nodes is being effectively used to learn the boundaries of the $+$ instances centred at $(10, 10)$.

Factors influencing model overfitting (High model complexity)

- Generally, a more complex model has a greater ability to represent complex patterns in the data.
- Decision trees with a greater number of leaf nodes will in general be able to represent more complex decision boundaries, when compared to decision trees with fewer leaf nodes.

Factors influencing model overfitting (High model complexity)

- Generally, a more complex model has a greater ability to represent complex patterns in the data.
- Decision trees with a greater number of leaf nodes will in general be able to represent more complex decision boundaries, when compared to decision trees with fewer leaf nodes.
- However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalise well over unseen instances.
- Models with high complexity should therefore be judiciously used to avoid overfitting.

Factors influencing model overfitting (High model complexity)

- Generally, a more complex model has a greater ability to represent complex patterns in the data.
- Decision trees with a greater number of leaf nodes will in general be able to represent more complex decision boundaries, when compared to decision trees with fewer leaf nodes.
- However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalise well over unseen instances.
- Models with high complexity should therefore be judiciously used to avoid overfitting.

Question: How do we measure complexity?

Measuring complexity

- One way is to look at the number of ‘parameters’ that need to be inferred from the training set.
- The attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set. A decision tree with a larger number of attribute test conditions (and consequently more leaf nodes) thus involves more ‘parameters’ and hence is more complex.

[2]This is similar to the *multiple testing problem* in statistics.

Measuring complexity

- One way is to look at the number of ‘parameters’ that need to be inferred from the training set.
- The attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set. A decision tree with a larger number of attribute test conditions (and consequently more leaf nodes) thus involves more ‘parameters’ and hence is more complex.
- Suppose we have a class of models with a certain number of parameters. A learning algorithm tries to select the best combination of parameter values than maximises some evaluation metric (e.g. accuracy) over the training set.
- If the number of parameter value combinations (and hence the complexity) is large, then the algorithm has to select the best combination from a larger number of possibilities, using a limited training set.
- This leads to a greater chance of the algorithm picking a spurious combination. The combination maximises the evaluation but does so by random chance.[2]

[2] This is similar to the *multiple testing problem* in statistics.

Example (Multiple testing problem)

- Consider the task of predicting whether the stock market will rise or fall in the next ten trading days.

Example (Multiple testing problem)

- Consider the task of predicting whether the stock market will rise or fall in the next ten trading days.
- If a stock analyst makes random guesses the probability that their prediction is correct on any trading day is 0.5.
- However, the probability that they correctly predict at least nine out of ten times is extremely low:

$$\frac{\binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0107.$$

Example (Multiple testing problem)

- Consider the task of predicting whether the stock market will rise or fall in the next ten trading days.
- If a stock analyst makes random guesses the probability that their prediction is correct on any trading day is 0.5.
- However, the probability that they correctly predict at least nine out of ten times is extremely low:

$$\frac{\binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0107.$$

- Now suppose we wish to choose an investment advisor from a pool of 200 stock analysis.
- Our strategy is to select the analyst who makes the most number of correct predictions in the next ten trading days.

Example continued

- The flaw in our thinking is that even if all the analysts make their predictions in a random fashion, the probability that at least one of them makes at least nine correct predictions is very high:

$$1 - (1 - 0.0107)^{200} = 0.8847.$$

Example continued

- The flaw in our thinking is that even if all the analysts make their predictions in a random fashion, the probability that at least one of them makes at least nine correct predictions is very high:

$$1 - (1 - 0.0107)^{200} = 0.8847.$$

- Although each analyst has a low probability of predicting at least nine times correctly, considered together we have a high probability of finding at least one analyst who can do so.
- Of course, there is no guarantee in the future that such an analyst will continue to make accurate predictions by random guessing.

Back to model overfitting

- In the context of learning a classification model each combination of parameter values corresponds to an analyst, while the number of training instances corresponds to the number of days.
- Analogous to the task of selecting the best analyst who makes the most accurate predictions on consecutive days, the task of a learning algorithm is to select the best combination of parameters that results in the highest accuracy on the training set.
- If the number of parameter combinations is large but the training size is small, it is highly likely for the learning algorithm to choose a spurious parameter combination that provides high training accuracy by simply random chance.

Factors influencing model overfitting (Presence of noise)

Consider the following training and test sets for a mammal classification problem. Note that two of the ten training records are mislabelled (bats and whales).

Factors influencing model overfitting (Presence of noise)

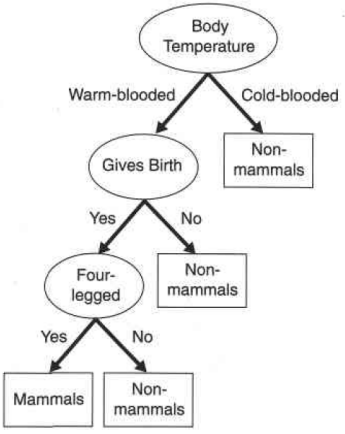
Consider the following training and test sets for a mammal classification problem. Note that two of the ten training records are mislabelled (bats and whales).

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no∗
whale	warm-blooded	yes	no	no	no∗
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

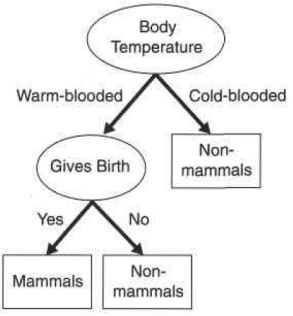
Example (Test set)

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	warm-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

Example (Decision tree)



(a) Model M1



(b) Model M2

Example (Concluded)

- M1 is a decision tree that perfectly fits the training data.
- Although the training error for the tree is zero, its error rate on the test is 30%. Both humans and dolphins were misclassified as non-mammals because their `Body Temperature`, `Gives Birth` and `Four-legged` are identical to the mislabelled records in the training set.

Example (Concluded)

- M1 is a decision tree that perfectly fits the training data.
- Although the training error for the tree is zero, its error rate on the test is 30%. Both humans and dolphins were misclassified as non-mammals because their `Body Temperature`, `Gives Birth` and `Four-legged` are identical to the mislabelled records in the training set.
- Spiny anteaters, on the other hand, represent an exceptional case in which the class label of a test record contradicts the class labels of other similar records.
- Errors due to exceptional cases are often unavoidable and establish the minimum error rate achievable by any classifier.

Example (Concluded)

- M1 is a decision tree that perfectly fits the training data.
- Although the training error for the tree is zero, its error rate on the test is 30%. Both humans and dolphins were misclassified as non-mammals because their `Body Temperature`, `Gives Birth` and `Four-legged` are identical to the mislabelled records in the training set.
- Spiny anteaters, on the other hand, represent an exceptional case in which the class label of a test record contradicts the class labels of other similar records.
- Errors due to exceptional cases are often unavoidable and establish the minimum error rate achievable by any classifier.
- In contrast, the decision tree M2 has a lower test error rate (10%) even though its training error rate is higher (20%).

Model selection

- Often, we have several possible classification models to choose from, many of which have varying levels of complexity.
- We want to choose the model that shows the lowest generalisation error rate.

Model selection

- Often, we have several possible classification models to choose from, many of which have varying levels of complexity.
- We want to choose the model that shows the lowest generalisation error rate.
- As we have seen, looking at test error rate alone is not enough to reliably choose a model. As such, we need to other means to influence our selection.
- We will see two generic approaches to estimate the generalisation performance of a model. We will then present specific strategies for using these approaches in the context of decision tree induction.

Using a validation set

- A popular strategy is to use a *validation set* which contains data not used for training the model.
- As this data is unseen by the model, error rates on a validation set are usually a better indicator of generalisation performance than the training error rate alone.
- We want to find *validation errors* and use these to influence our model selection.

Using a validation set

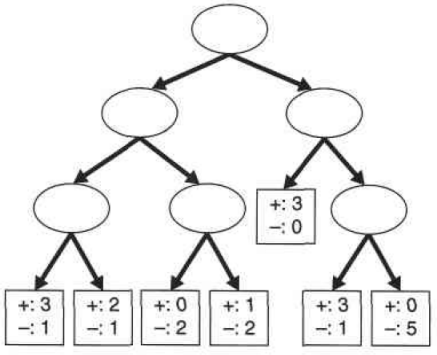
- A popular strategy is to use a *validation set* which contains data not used for training the model.
- As this data is unseen by the model, error rates on a validation set are usually a better indicator of generalisation performance than the training error rate alone.
- We want to find *validation errors* and use these to influence our model selection.
- To do so, we take our training set (call it $D.train$) and partition it into two smaller subsets (call them $D.tr$ and $D.val$). We then use $D.tr$ to train out model, and $D.val$ for the validation set.
- How we partition $D.train$ is up to us and is largely situation specific, e.g. we could use two-thirds for $D.tr$ and one-third for $D.val$.
- For any choice of classification model m that is trained on $D.tr$, we can then estimate its validation error rate on $D.val$, labelled $err_{val}(m)$.
- The model that shows the lowest value of $err_{val}(m)$ can then be selected.

A limitation of validation sets

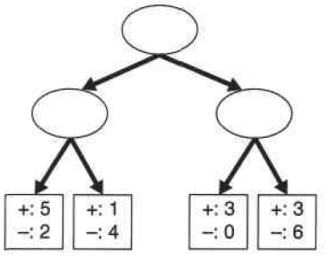
- One limitation of this approach is its sensitivity to the sizes chosen from the partition.
- If we make $D.tr$ too small then this may result in the learning of a poor classification model since a smaller training set is likely to be less representative of the overall data.
- Likewise, if $D.val$ is too small, the validation error rate might not be representative of the generalisation error rate, thereby nullifying the point of this step.

Example (Training data)

Consider the following two decision trees generated from the same training data:



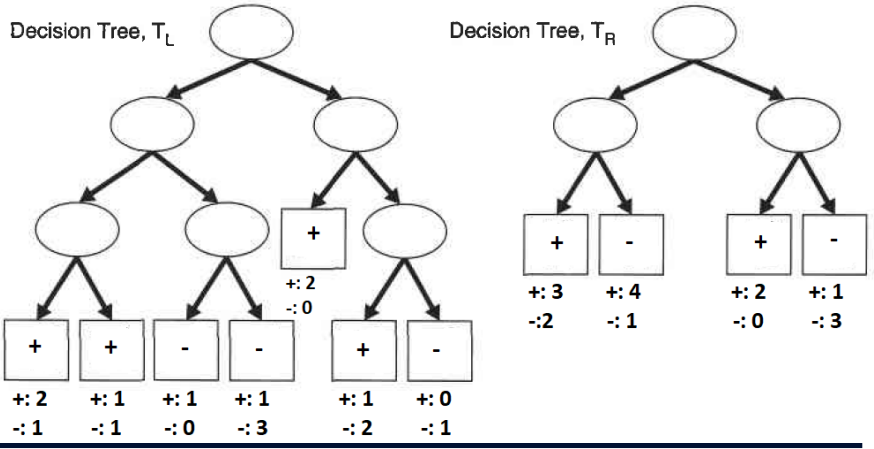
Decision Tree, T_L



Decision Tree, T_R

Example (Validation set)

Now we use the validation set. The counts beneath the leaf nodes represent the proportion of data instances in the validation set that reach each of the nodes.



Computing validation error

Using our earlier notation, for T_L, we have:

$$f_{11} = 6 \ , \ f_{00} = 4 \ , \ f_{10} = 4 \ , \ f_{01} = 2.$$

Thus, the validation error rate for the left tree is $err_{val}(T_L) = \frac{6}{16} = 0.375$.

Computing validation error

Using our earlier notation, for T_L, we have:

$$f_{11} = 6 \ , \ f_{00} = 4 \ , \ f_{10} = 4 \ , \ f_{01} = 2.$$

Thus, the validation error rate for the left tree is $err_{val}(T_L) = \frac{6}{16} = 0.375$.
Likewise, for T_R we have the following:

$$f_{11} = 5 \ , \ f_{00} = 4 \ , \ f_{10} = 2 \ , \ f_{01} = 5.$$

Thus, the validation error rate for the right tree is $err_{val}(T_R) = \frac{7}{16} = 0.4375$.

Computing validation error

Using our earlier notation, for T_L, we have:

$$f_{11} = 6 \ , \ f_{00} = 4 \ , \ f_{10} = 4 \ , \ f_{01} = 2.$$

Thus, the validation error rate for the left tree is $err_{val}(T_L) = \frac{6}{16} = 0.375$.
Likewise, for T_R we have the following:

$$f_{11} = 5 \ , \ f_{00} = 4 \ , \ f_{10} = 2 \ , \ f_{01} = 5.$$

Thus, the validation error rate for the right tree is $err_{val}(T_R) = \frac{7}{16} = 0.4375$.

Based on this, the left tree is preferred over the right one.

Incorporating model complexity

- As discussed previously, the chance for model overfitting increases as the model becomes more complex. As such, we need not only consider training error rate, but also model complexity.

Incorporating model complexity

- As discussed previously, the chance for model overfitting increases as the model becomes more complex. As such, we need not only consider training error rate, but also model complexity.
- A generic approach to account for model complexity while estimated generalisation performance is formally described as follows.
  - Suppose we have a training set $D.train$ and a class of models $\mathcal{M}$ (for example, $\mathcal{M}$ could represent the set of all possible decision trees).

Incorporating model complexity

- As discussed previously, the chance for model overfitting increases as the model becomes more complex. As such, we need not only consider training error rate, but also model complexity.
- A generic approach to account for model complexity while estimated generalisation performance is formally described as follows.
  - Suppose we have a training set $D.train$ and a class of models $\mathcal{M}$ (for example, $\mathcal{M}$ could represent the set of all possible decision trees).
  - We consider a learning a classification model m that is in $\mathcal{M}$, and are interested in estimating the generalisation error rate of m, denoted by $gen.error(m)$.

Incorporating model complexity

- As discussed previously, the chance for model overfitting increases as the model becomes more complex. As such, we need not only consider training error rate, but also model complexity.
- A generic approach to account for model complexity while estimated generalisation performance is formally described as follows.
  - Suppose we have a training set $D.train$ and a class of models $\mathcal{M}$ (for example, $\mathcal{M}$ could represent the set of all possible decision trees).
  - We consider a learning a classification model m that is in $\mathcal{M}$, and are interested in estimating the generalisation error rate of m, denoted by $gen.error(m)$.
  - The training error rate of m ($train.error(m, D.train)$) can underestimate $gen.error(m)$ when the model complexity is high. We therefore represent $gen.error(m)$ as a function of the training error rate *and* model complexity of $\mathcal{M}$, denoted by $complexity(\mathcal{M})$.

The formula

- We define the function by,

$$gen.error(m) = train.error(m,\, D.train) + \alpha \times complexity(\mathcal{M}), \qquad (1)$$

where α is a hyper-parameter that strikes a balance between minimising training error and reducing model complexity.

The formula

- We define the function by,

$$gen.error(m) = train.error(m,\, D.train) + \alpha \times complexity(\mathcal{M}), \qquad (1)$$

  where α is a hyper-parameter that strikes a balance between minimising training error and reducing model complexity.
- A higher value of α gives more emphasis to the model complexity in the estimation of generalisation performance.
- To choose the right value of α we can make use of a validation set. For instance, we can iterate through a range of values of α and for every possible value we learn a model on a subset $D.tr$ of $D.train$. We then compute the validation error rate on a separate subset $D.val$ and select the value of α that provides the lowest validation error rate.

The context of our formula

- Equation 1 provides one possible approach for incorporating model complexity into the estimate of generalisation performance.
- This approach is at the heart of a number of techniques for estimating generalisation performance, such as:
  - the structural risk minimisation principle;
  - the Akaike's Information Criterion (AIC);
  - the Bayesian Information Criterion (BIC).
- In particular, the structural risk minimisation principle serves as the building block for learning support vector machines.

Bringing this back to decision trees

- In the context of decision trees, the complexity of a decision tree can be estimated as the ratio of the number of leaf nodes to the number of training instances.
- Let k be the number of leaf nodes and N_{train} be the number of training instances. The complexity of a decision tree can then be described as $\frac{k}{N_{train}}$.

Intuition: For a larger training size, we can learn a decision tree with a larger number of leaf nodes without it becoming overly complex.

The generalisation error rate

- The generalisation error rate of a decision tree T can then be computed using Equation 1 as follows:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}},$$

where

- $err(T)$ is the training error of the decision tree, and
- Ω is a hyper-parameter that makes a trade-off between reducing training error and minimising model complexity, similar to α before.[3]

[3] We view Ω as the relative cost of adding a leaf node relative to incurring a training error.

The generalisation error rate

- The generalisation error rate of a decision tree T can then be computed using Equation 1 as follows:

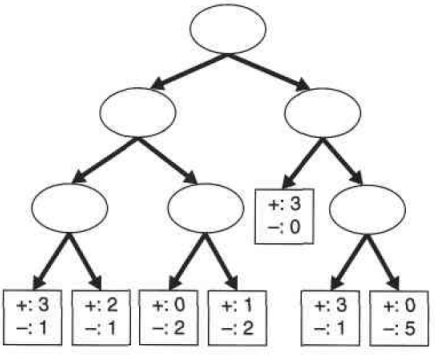
$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}},$$

  where
  - $err(T)$ is the training error of the decision tree, and
  - Ω is a hyper-parameter that makes a trade-off between reducing training error and minimising model complexity, similar to α before.[3]
- In the literature, this is often referred to as the *pessimistic error estimate*, reflecting the idea that it assumes the generalisation error rate to be worse than the training error rate (by adding a penalty term for model complexity).
- In contrast, the *optimistic error estimate* (or *resubstitution estimate*) simply uses the training error as an estimate of the generalisation error rate.

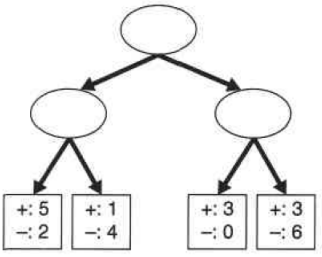
[3] We view Ω as the relative cost of adding a leaf node relative to incurring a training error.

Example

Return to the decision trees from earlier:



Decision Tree, T_L



Decision Tree, T_R

Example

- If each leaf node is labelled according to the majority class of training instances that reach the node, the training error rate for the left tree will be $err(T_L) = \frac{4}{24} = 0.167$, while the training error rate for the right tree will be $err(T_R) = \frac{6}{24} = 0.25$.
- Based on their error training rates alone, T_L would be preferred over T_R, even though T_L is more complex than T_R.

Example

- If each leaf node is labelled according to the majority class of training instances that reach the node, the training error rate for the left tree will be $err(T_L) = \frac{4}{24} = 0.167$, while the training error rate for the right tree will be $err(T_R) = \frac{6}{24} = 0.25$.
- Based on their error training rates alone, T_L would be preferred over T_R, even though T_L is more complex than T_R.
- Next, we assume that the cost associated with each leaf node is $\Omega = 0.5$. Then the generalisation error estimate for T_L will be,

$$err_{gen}(T_L) = \frac{4}{24} + 0.5 \times \frac{7}{24} = 0.3125$$

Example

- If each leaf node is labelled according to the majority class of training instances that reach the node, the training error rate for the left tree will be $err(T_L) = \frac{4}{24} = 0.167$, while the training error rate for the right tree will be $err(T_R) = \frac{6}{24} = 0.25$.
- Based on their error training rates alone, T_L would be preferred over T_R, even though T_L is more complex than T_R.
- Next, we assume that the cost associated with each leaf node is $\Omega = 0.5$. Then the generalisation error estimate for T_L will be,

$$err_{gen}(T_L) = \frac{4}{24} + 0.5 \times \frac{7}{24} = 0.3125$$

and the generalisation error estimate for T_R will be

$$err_{gen}(T_R) = \frac{6}{24} + 0.5 \times \frac{4}{24} = \frac{1}{3}.$$

Example (Choosing Ω)

- Since T_L has a lower generalisation error rate, it will still be preferred over T_R.
- Note that $\Omega = 0.5$ implies that a node should always be expanded into its two child nodes if it improves the prediction of at least one training instance, since expanding a node is less costly than misclassifying a training instance.

Example (Choosing Ω)

- Since T_L has a lower generalisation error rate, it will still be preferred over T_R.
- Note that $\Omega = 0.5$ implies that a node should always be expanded into its two child nodes if it improves the prediction of at least one training instance, since expanding a node is less costly than misclassifying a training instance.
- Alternatively, if $\Omega = 1$, then the generalisation error rate for T_L is $err_{gen}(T_L) = \frac{11}{24} = 0.458$ and for T_R is $err_{gen}(T_R) = \frac{10}{24} = 0.417$.
- In this case, T_R is the preferred option because it has a lower generalisation error rate.

Example (Choosing Ω)

- Since T_L has a lower generalisation error rate, it will still be preferred over T_R.
- Note that $\Omega = 0.5$ implies that a node should always be expanded into its two child nodes if it improves the prediction of at least one training instance, since expanding a node is less costly than misclassifying a training instance.
- Alternatively, if $\Omega = 1$, then the generalisation error rate for T_L is $err_{gen}(T_L) = \frac{11}{24} = 0.458$ and for T_R is $err_{gen}(T_R) = \frac{10}{24} = 0.417$.
- In this case, T_R is the preferred option because it has a lower generalisation error rate.
- This demonstrates the impact of choosing different values for Ω. As with α, the value of Ω can be selected with the help of a validation set.

Model Evaluation

Exercise: Read pp. 106-108 of the notes (Estimating Statistical Bounds)

Model Evaluation

Exercise: Read pp. 106-108 of the notes (Estimating Statistical Bounds)

- We conclude by discussing ways to estimate a classification model's generalisation performance, i.e. its performance on unseen instances outside of $D.train$.
- If we use the validation error rate for model section, the resulting model would be deliberately chosen to minimise the errors on the validation set. The validation error rate may thus under-estimate the true generalisation error rate, and hence cannot be reliably used for model evaluation.

Model Evaluation

Exercise: Read pp. 106-108 of the notes (Estimating Statistical Bounds)

- We conclude by discussing ways to estimate a classification model's generalisation performance, i.e. its performance on unseen instances outside of $D.train$.
- If we use the validation error rate for model section, the resulting model would be deliberately chosen to minimise the errors on the validation set. The validation error rate may thus under-estimate the true generalisation error rate, and hence cannot be reliably used for model evaluation.
- A correct approach would therefore be to assess the performance of a learned model on a labelled test set that has not been used at any stage of model selection. This can be achieved by partitioning the entire set of labelled instanced D into two disjoint subsets, $D.train$ (used for model selection), and $D.test$ (used for computing the test error rate, err_{test}.

The Holdout Method

- The most basic technique for partitioning a labelled data set is the *holdout method*.
- Here, the labelled set D is randomly partitioned into two disjoint sets, called the training set $D.train$ and the test set $D.test$.
- A classification model is then induced from $D.train$ using the model selection approaches already discussed, and its error rate on $D.test$, err_{test} is used as an estimate of the generalisation error rate.

The Holdout Method

- The most basic technique for partitioning a labelled data set is the *holdout method*.
- Here, the labelled set D is randomly partitioned into two disjoint sets, called the training set $D.train$ and the test set $D.test$.
- A classification model is then induced from $D.train$ using the model selection approaches already discussed, and its error rate on $D.test$, err_{test} is used as an estimate of the generalisation error rate.
- The proportion of data reserved for training and for testing is generally situation specific, e.g. we could split them two-thirds and one-third, respectively.
- As with the trade-off faced when partitioning $D.train$ into $D.tr$ and $D.val$, choosing the right fraction of labelled data to be used for training/test is crucial and generally non-trivial.

The Holdout Method continued

- If the size of $D.train$ is small, the learned classification model may be improperly learned using an insufficient number of training instances. This results in a biased estimated of generalisation performance.

The Holdout Method continued

- If the size of $D.train$ is small, the learned classification model may be improperly learned using an insufficient number of training instances. This results in a biased estimated of generalisation performance.
- Likewise, if $D.test$ is small, err_{test} may be less reliable as it would be computed over a small number of test instances.
- Moreover, err_{test} can have a high variance as we change the random partitioning of D into $D.train$ and $D.test$.

The Holdout Method continued

- If the size of $D.train$ is small, the learned classification model may be improperly learned using an insufficient number of training instances. This results in a biased estimated of generalisation performance.
- Likewise, if $D.test$ is small, err_{test} may be less reliable as it would be computed over a small number of test instances.
- Moreover, err_{test} can have a high variance as we change the random partitioning of D into $D.train$ and $D.test$.
- We can of course repeat the holdout method several times to obtain a distribution of the test error rates. This approach is known as *random subsampling* and produces a distribution of the error rates that can be used to understand the variance of err_{test}.

Cross-validation

- This widely-used model evaluation method aims to make effective use of all labelled instances in D for both training and testing.
- To demonstrate, suppose we are given a labelled set that we have randomly partitioned into three equal-sized subsets, S_1, S_2, S_3 (see Figure on next slide).
- For the first run, we train a model using subsets S_2, S_3 (shown as empty blocks) and test the model on subset S_1. The test error rate on S_1, denoted by $err(S_1)$, is then computed in the first run.

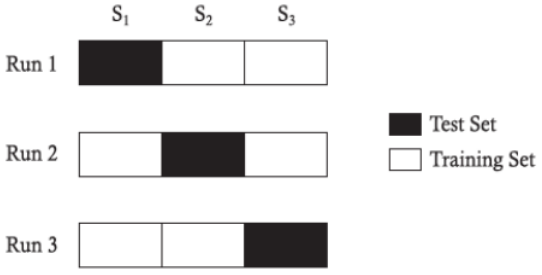
Cross-validation

- This widely-used model evaluation method aims to make effective use of all labelled instances in D for both training and testing.
- To demonstrate, suppose we are given a labelled set that we have randomly partitioned into three equal-sized subsets, $S_1,\ S_2,\ S_3$ (see Figure on next slide).
- For the first run, we train a model using subsets $S_2,\ S_3$ (shown as empty blocks) and test the model on subset S_1. The test error rate on S_1, denoted by $err(S_1)$, is then computed in the first run.
- Likewise, for the second run we use $S_1,\ S_3$ as the training set and S_2 as the test set, computing the test error rate $err(S_2)$.
- Finally, we use $S_1,\ S_2$ as the training sets, and S_3 as the test set for which we compute $err(S_3)$.

Cross-validation

- This widely-used model evaluation method aims to make effective use of all labelled instances in D for both training and testing.
- To demonstrate, suppose we are given a labelled set that we have randomly partitioned into three equal-sized subsets, $S_1,\ S_2,\ S_3$ (see Figure on next slide).
- For the first run, we train a model using subsets $S_2,\ S_3$ (shown as empty blocks) and test the model on subset S_1. The test error rate on S_1, denoted by $err(S_1)$, is then computed in the first run.
- Likewise, for the second run we use $S_1,\ S_3$ as the training set and S_2 as the test set, computing the test error rate $err(S_2)$.
- Finally, we use $S_1,\ S_2$ as the training sets, and S_3 as the test set for which we compute $err(S_3)$.
- The overall test error rate is obtained by summing up the number of errors committed in each test subset across all runs and divided by the total number of instances. This approach is called *three-fold validation*.

Cross-validation example



More generally...

- k-fold cross-validation partitions our labelled data set D (of size N) into k equal-sized[4] partitions (or 'folds') $S_1, \ldots, S_k$.
- During the i^{th} run, partition S_i is reserves as the test set and the remaining partitions are collectively used to train the model.
- A model $m(i)$ is learned and applied on $D.test(i)$ to obtain the sum of test errors $err_{sum}(i)$.
- This procedure is then repeated k times and the total test error rate is given by,

$$err_{test} = \frac{\sum_i^k err_{sum}(i)}{N}.$$

[4] If the set does not partition exactly, then each subset is approximately of equal size.

More generally...

- k-fold cross-validation partitions our labelled data set D (of size N) into k equal-sized[4] partitions (or 'folds') $S_1, \ldots, S_k$.
- During the i^{th} run, partition S_i is reserves as the test set and the remaining partitions are collectively used to train the model.
- A model $m(i)$ is learned and applied on $D.test(i)$ to obtain the sum of test errors $err_{sum}(i)$.
- This procedure is then repeated k times and the total test error rate is given by,

$$err_{test} = \frac{\sum_i^k err_{sum}(i)}{N}.$$

- Unlike the holdout and random subsampling methods, here each sample is used the same number of times $(k-1)$ for training and once for testing.
- Every run uses $\frac{k-1}{k}$ fraction of the data for training and $\frac{1}{k}$ fraction for testing.

[4] If the set does not partition exactly, then each subset is approximately of equal size.

How do we choose k?

- A small value of k will result in a smaller training set at every run, which will result in a larger estimate of generalisation error rate than what is expected of a model trained over the entire labelled set.

How do we choose k?

- A small value of k will result in a smaller training set at every run, which will result in a larger estimate of generalisation error rate than what is expected of a model trained over the entire labelled set.
- At the other end, a higher value of k results in a larger training set at every run which in turn reduces the bias in the estimate of generalisation error rate.
- In the extreme case when $k = N$, every run uses exactly one data instance for testing and the remainder of the data for training. This special case is called the *leave-one-out* approach.
- This approach has the advantage of utilising as much data as possible for training but is computationally expensive for large data sets (cross-validation needs to be repeated N times).

How do we choose k?

- A small value of k will result in a smaller training set at every run, which will result in a larger estimate of generalisation error rate than what is expected of a model trained over the entire labelled set.
- At the other end, a higher value of k results in a larger training set at every run which in turn reduces the bias in the estimate of generalisation error rate.
- In the extreme case when $k = N$, every run uses exactly one data instance for testing and the remainder of the data for training. This special case is called the *leave-one-out* approach.
- This approach has the advantage of utilising as much data as possible for training but is computationally expensive for large data sets (cross-validation needs to be repeated N times).
- For most practical applications, the choice of k is between 5 and 10. This provides a reasonable approach for estimating the generalisation error rate because each fold is able to make use of 80% to 90% of the labelled data for training.

Summary

- We have discussed factors influencing model overfitting:
  - Limited training size.
  - High model complexity.
  - Presence of noise.
- We have seen ways to select different models.
- We have seen how to use a validation set and how to compute validation error.
- We have also seen how to incorporate model complexity:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}.$$

- We have seen how to evaluate our model. In particular, we saw:
  - The Holdout Method.
  - Cross-validation.