Generalization and overfitting

- If there are two or more examples with the same description (in terms of attributes) but different classifications $\Rightarrow$ no consistent decision tree exists
- The solution is to have each leaf node report either
  - The majority classification for its set of examples, if a deterministic hypothesis is required, or
  - the estimated probabilities of each classification using the relative frequencies
- It is quite possible, and in fact likely, that even when vital information is missing, the learning algorithm will find a consistent decision tree
- This is because the algorithm can use irrelevant attributes, if any, to make spurious distinctions among the examples

Consider trying to predict the roll of a die on the basis of
- The day and
- The month in which the die was rolled, and
- Which is color of the die,
then as long as no two examples have identical descriptions, the learning algorithm will find an exact hypothesis
- Such a hypothesis will be totally spurious
- The more attributes there are, the more likely it is that an exact hypothesis will be found
- The correct tree to return would be a single leaf node with probabilities close to 1/6 for each roll
- This problem is an example of overfitting, a very general phenomenon afflicting every kind of learning algorithm and target function, not only random concepts
Decision tree pruning

- A simple approach to deal with overfitting is to prune the decision tree.
- Pruning works by preventing recursive splitting on attributes that are not clearly relevant.

Suppose we split a set of examples using an irrelevant attribute:
- Generally, we would expect the resulting subsets to have roughly the same proportions of each class as the original set.
- In this case, the information gain will be close to zero.
- How large a gain should we require in order to split on a particular attribute?

A statistical significance test begins by assuming that there is no underlying pattern (the so-called null hypothesis) and then analyzes the actual data to calculate the extent to which they deviate from a perfect absence of pattern.
- If the degree of deviation is statistically unlikely (usually taken to mean a 5% probability or less), then that is considered to be good evidence for the presence of a significant pattern in the data.
- The probabilities are calculated from standard distributions of the amount of deviation one would expect to see in random sampling.
- Null hypothesis: the attribute at hand is irrelevant and, hence, its information gain for an infinitely large sample is zero.
- We need to calculate the probability that, under the null hypothesis, a sample of size $v = n + p$ would exhibit the observed deviation from the expected distribution of examples.
Let the numbers positive and negative examples in each subset be $p_i$ and $n_i$, respectively.

Their expected values, assuming true irrelevance, are

$$p'_i = \frac{p (p_i + n_i)}{(p + n)}$$
$$n'_i = \frac{n (p_i + n_i)}{(p + n)}$$

where $p$ and $n$ are the total numbers of positive and negative examples in the training set.

A convenient measure for the total deviation is given by

$$D = \sum_{i=1, \ldots, v} \frac{(p_i - p'_i)^2}{p'_i} + \frac{(n_i - n'_i)^2}{n'_i}$$

Under the null hypothesis, the value of $D$ is distributed according to the $\chi^2$ (chi-squared) distribution with $(v-1)$ degrees of freedom.

The probability that the attribute is really irrelevant can be calculated with the help of standard $\chi^2$ tables.

The above method is known as $\chi^2$ (pre-)pruning.

Pruning allows the training examples to contain noise and it also reduces the size of the decision trees and makes them more comprehensible.

More common than pre-pruning are post-pruning methods in which

- One first constructs a decision tree that is as consistent as possible with the training data and
- Then removes those subtrees that have likely been added due to the noise

In cross-validation the known data is divided in $k$ parts, each of which is used as a test set in its turn for a decision tree that has been grown on the other $k-1$ subsets.

Thus one can approximate how well each hypothesis will predict unseen data.
Broadening the applicability of decision trees

- In practice decision tree learning has to answer also the following questions
  - Missing attribute values: while learning and in classifying instances
  - Multivalued discrete attributes: value subsetting or penalizing against too many values
  - Numerical attributes: split point selection for interval division
  - Continuous-valued output attributes
- Decision trees are used widely and many good implementations are available (for free)
- Decision trees fulfill understandability, contrary to neural networks, which is a legal requirement for financial decisions

18.4 Evaluating and Choosing the Best Hypothesis

- We assume that there is a probability distribution over examples that remains stationary over time
  - Each observed value is sampled from that distribution and is independent of previous examples and
  - Each example has identical prior probability distribution
- Examples that satisfy these assumptions are called independent and identically distributed (i.i.d.)
- The error rate of a hypothesis \( h \) is the proportion of mistakes it makes
  - The proportion of times that \( h(x) \neq y \) for an \((x, y)\) example
- Just because a hypothesis \( h \) has low error rate on the training set does not mean that it will generalize well
Model selection: Complexity vs. goodness of fit

- We can think of finding the best hypothesis as two tasks:
  - **Model selection** defines the hypothesis space and
  - **Optimization** finds the best hypothesis within that space
- How to select among models that are parameterized by size
  - With polynomials we have \( \text{size} = 1 \) for linear functions,
    \( \text{size} = 2 \) for quadratics, and so on
  - For decision trees, the size could be the number of nodes in
    the tree
- We want to find the value of the size parameter that best
  balances underfitting and overfitting to give the best test set
  accuracy

- A wrapper takes a learning algorithm as an argument (DT
  learning for example)
- The wrapper enumerates models according to the size parameter
- For each size, it uses cross validation (say) on the learner to
  compute the average error rate on training and test sets
- We start with the smallest, simplest models (which probably
  underfit the data), and iterate, considering more complex models
  at each step, until the models start to overfit
- The cross validation picks the value of \( \text{size} \) with the lowest
  validation set error
- We then generate a hypothesis of that \( \text{size} \) using all the data
  (without holding out any of it; eventually we should evaluate the
  returned hypothesis on a separate test set)
From error rates to loss

- Consider the problem of classifying emails as spam or non-spam
- It is worse to classify non-spam as spam than to classify spam as non-spam
- So a classifier with a 1% error rate, where almost all errors were classifying spam as non-spam, would be better than a classifier with only a 0.5% error rate, if most of those errors were classifying non-spam as spam
- Utility is what learners – like decision makers – should maximize
- In machine learning it is traditional to express utilities by means of loss functions
- The loss function $L(x, y, \hat{y})$ is defined as the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$:
  $$L(x, y, \hat{y}) = U(result \ of \ using \ y \ given \ an \ input \ x) - U(result \ of \ using \ \hat{y} \ given \ an \ input \ x)$$

Often a simplified version of the loss function is used: It is 10 times worse to classify non-spam as spam than vice-versa:

- $L(spam, nonspam) = 1$, $L(nonspam, spam) = 10$
- Note that $L(y, y)$ is always zero
- In general for real-valued data small errors are better than large ones
- Two functions that implement that idea are the absolute value of the difference (called the $L_1$ loss), and the square of the difference (called the $L_2$ loss)
- Minimizing error rate is formulated in the $L_{0/1}$ loss function

### $L_1$ loss:
$$L_1(y, \hat{y}) = |y - \hat{y}|$$

### $L_2$ loss:
$$L_2(y, \hat{y}) = (y - \hat{y})^2$$

### $L_{0/1}$ loss:
$$L_{0/1}(y, \hat{y}) = 0 \text{ if } y = \hat{y}, \text{ else } 1$$
Let $P(X, Y)$ be a prior probability distribution over examples. Let $E$ be the set of all possible input-output examples. Then the expected **generalization loss** for a hypothesis $h$ (w.r.t. loss function $L$) is:

$$\text{GenLoss}_L(h) = \sum_{(x, y) \in E} L(y, h(x)) P(x, y)$$

The best hypothesis $h^*$ is the one with the minimum expected generalization loss:

$$h^* = \arg \min_{h \in H} \text{GenLoss}_L(h)$$

Because $P(x, y)$ is not known, the learning agent can only estimate generalization loss with **empirical loss** on the set of examples $E$:

$$\text{EmpLoss}_{L,E}(h) = \frac{1}{n} \sum_{(x, y) \in E} L(y, h(x))$$

The estimated best hypothesis $\hat{h}^*$ is then the one with minimum empirical loss:

$$\hat{h}^* = \arg \min_{h \in H} \text{EmpLoss}_{L,E}(h)$$

**Regularization**

Earlier on we did model selection with cross-validation on model size. An alternative approach is to search for a hypothesis that directly minimizes the weighted sum of empirical loss and the complexity of the hypothesis, which we call the total cost:

$$\text{Cost}(h) = \text{EmpLoss}(h) + \Lambda \text{Complexity}(h)$$

$$\hat{h}^* = \arg \min_{h \in H} \text{Cost}(h)$$

Here $\Lambda$ is a parameter, a positive number that serves as a conversion rate between loss and hypothesis complexity. We still need to do cross-validation search to find the hypothesis that generalizes best, but this time with different values of $\Lambda$. This process of explicitly penalizing complex hypotheses is called **regularization**.
18.5 The Theory of Learning

- Computational Learning Theory (COLT) [Valiant 1984]
- Any hypothesis that is seriously wrong will almost certainly be “found out” with high probability after a small number of examples, because it will make an incorrect prediction
- Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: that is, it must be probably approximately correct (PAC)
- PAC-learning is based on the stationarity assumption: Future examples are going to be drawn from the same fixed distribution \( P(E) = P(X, Y) \) as past examples
- We do not necessarily know the distribution
- Without this connection between the past and the future, learning would be practically impossible

Let \( X \) be the instance space from which the examples are drawn from with a stationary distribution
- \( H \) is the hypothesis class and \( N \) denotes the number of training examples
- We assume that the true function \( f \) is a member of \( H \)
- The error rate of a hypothesis \( h \) is defined as the expected generalization error for examples drawn from the stationary distribution

\[
\text{error}(h) = \text{GenLoss}_{0/1}(h) = \sum_{(x,y)} L_{0/1}(y, h(x)) P(x, y)
\]

A hypothesis \( h \) is called approximately correct if \( \text{error}(h) \leq \varepsilon \), where \( \varepsilon \) is a small constant
Let us examine a hypothesis $h_b$, which is not an $\varepsilon$-approximation of the true function.

Then $error(h_b) > \varepsilon$ and the probability that it agrees with a given example is at most $1 - \varepsilon$.

Since the examples are independent, the bound for $N$ examples is

$$P(h_b \text{ agrees with } N \text{ examples}) \leq (1 - \varepsilon)^N$$

The probability that the hypothesis class $H$ contains at least one such hypothesis is at most $|H|(1 - \varepsilon)^N$.

We would like to reduce the probability of this event below some small number $\delta$.

$$|H|(1 - \varepsilon)^N \leq \delta$$

Then

$$N \geq \frac{1}{-\ln(1 - \varepsilon)}(\ln|H| + \ln(1/\delta))$$

Because $\ln(1 + a) < a$, we can achieve this if we allow the algorithm to see

$$N \geq \frac{1}{\varepsilon}(\ln|H| + \ln(1/\delta))$$

examples.

Thus, if a learning algorithm returns a hypothesis that is consistent with this many examples, then with probability at least $1 - \delta$, it has error at most $\varepsilon$.

In other words, it is probably approximately correct.

The number of required examples, as a function of $\varepsilon$ and $\delta$, is called the sample complexity of the hypothesis space.
Learning decision lists

- A decision list consists of a series of tests, each of which is a conjunction of literals
- The tests (rules) are attempted in order
- If a test succeeds, the decision list specifies the value to be returned
- If a test fails, processing continues with the next test in the list

![Decision List Diagram]

- Decision lists resemble decision trees, but their overall structure is simpler: they branch only in one direction
- In contrast, the individual tests are more complex
- Decision lists generalize decision trees
- If we allow tests of arbitrary size, then decision lists can represent any Boolean function
- On the other hand, if we restrict the size of each test to at most \( k \) literals (\( k \)-DL), then it is possible for the learning algorithm to generalize successfully from a small number of examples
- Language \( k \)-DL includes as a subset the language \( k \)-DT, the set of all decision trees of depth at most \( k \)
- Let \( k \)-DL(\( n \)) denote a \( k \)-DL language using \( n \) Boolean attributes
Let $\text{Conj}(n, k)$ denote the conjunctions of at most $k$ literals using $n$ attributes.

The number of conjunctions of $k$ literals from $n$ attributes is given by

$$|\text{Conj}(n, k)| = \sum_{i=1}^{k} \binom{2n}{i} = O(n^k)$$

In a decision list each conjunction can be attached to either a Yes or No outcome or can be absent from the decision list.

Hence, there are at most $3^{|\text{Conj}(n, k)|}$ distinct sets of component tests.

Each of these sets of tests can be in any order, so

$$|k\text{-DL}(n)| \leq 3^{|\text{Conj}(n, k)|} |\text{Conj}(n, k)|!$$

Combining the above estimates yields

$$|k\text{-DL}(n)| \leq 2^{O(n^k \log_2(n^k))}$$

We plug this into the formula for sample complexity to show that the number of examples needed for PAC-learning is

$$N \geq \frac{1}{\epsilon} \left( |O(n^k \log_2(n^k))| + \ln(1/\delta) \right)$$

Hence, a $k$-DL function is PAC-learnable from a number of examples polynomial in $n$.

Therefore, any algorithm that returns a consistent decision list will PAC-learn a $k$-DL function in a reasonable number of examples, for small $k$.

The following greedy algorithm will output a consistent decision list (if one exists).
Algorithm DL-learner(S)

Input: S training examples
Output: a decision list or failure

1. if S = Ø then return the trivial decision list No;
2. Choose a test t that matches a subset of examples S_i ≠ Ø such that the members are all positive or negative;
3. if there is no such t then fail;
4. if all examples in S_i are positive
   then o ← Yes
   else o ← No;
5. L ← DL-learner(S \ S_i);
6. return a decision list with initial test t and outcome o and remaining tests given by L;