Chapter 5: Machine Learning and Model Search

In the last chapter we discussed the strengths of computers: speed, accuracy and not getting bored. We then discussed algorithmic prediction, and how computers were uniquely able to handle prediction by simulation.

In this chapter we also leverage the strengths of computers applied to algorithmic learning. Computers are uniquely equipped to learn by exhaustive search.

We will introduce some fundamentals of Artificial Intelligence and Machine Learning before discussing additional concerns of Computational Scientific Discovery.

5.1 An Artificial Intelligence Researcher’s World-view

Artificial Intelligence (A.I.) researchers divide computational problems into two sets: those that are solvable by “algorithms” and those that are not. When an A.I. researcher say “algorithm” he or she often means something (1) runnable in polynomial time, and (2) guaranteed to give the optimal (or just correct) answer. For example your favorite O(n lg n) or O(n^2) sorting algorithms are “algorithms” in the A.I. sense: stuff the data in, wait O(n lg n) or O(n^2) time, and out comes a correctly sorted data structure. Such “algorithms” are, of course, the bread-and-butter of computer science and are used in a variety of applications everyday.

However, there are other problems that are not (currently) solvable by “algorithms”. Consider playing the game of chess. Can you guarantee that you will always win at chess? Well, perhaps if you are a Grand Master then you may indeed be reasonably sure you can beat everyone around you. But what if you go against another Grand Master? The outcome might be less certain. Well, if you cannot guarantee you will always win, what about the weaker goal of never losing? That might be too difficult to guarantee accomplishing too.

Thus, chess is a good candidate for A.I. Compare chess with tic-tac-toe, where one can guarantee that one will never lose. Chess is an interesting game precisely because of this uncertainty of outcome. Tic-tac-toe, however, is generally only of interest to children. They may not have refined their algorithm for never losing.

How do we proceed when we cannot guarantee we will win (i.e. there is no “algorithm”)? We search for a solution. The A.I. paradigm gives us the infrastructure to attempt to solve “interesting” (i.e. non-“algorithm” solvable) problems by search.

Assume we are writing a program to play chess. We start the game with the board in its initial configuration:
This configuration of our world is called a **state**. This one in particular is the **initial state**.

From this or any other state there are a set of **operators** that we may apply to go to a (in general, different) state. In our chess example the operators are, of course, the legal moves of our pieces. For example, if we were White, from the initial state we may move the king's knight from G1 to F3, or the queen's bishop's pawn from C2 to C4.

Our opponent moves to counter our moves, and (at least initially) has a lot of freedom about which of their operators to execute. Thus, in general, we are only **partially** in control of the state that results for a given operator.

The set of states reachable by the operators from the initial state and its possible successor states defines the **state space** or **search space**.

We traverse the states of the state space with the goal of arriving at a **goal state**. In chess
the goal is checkmate, of course. Hopefully there is at least one goal state in the state space. In general there is more than one (e.g. more than one configuration of the board is corresponds to checkmate). It is up to our search algorithm to try to get us from the initial state to a goal state.

Humans play chess; computers play chess; the best humans are comparable to the best computers, but humans and computers play completely differently. A.I. pioneer Alan Newell considered how all the above could be simultaneously true. He proposed that a given level of competence at problem solving might be achievable by having a lot of knowledge about a problem. That knowledge would facilitate intelligent search, and thus require less search. This is how relatively slow (at the symbol level) humans play good chess. The same level of competence, however, might be achievable with less intelligence if one had the capacity for more search. This is how a computer would play chess.

Because human chess players probably cannot be trained to think much faster the natural way for humans to improve is by playing more. With more experience comes more knowledge about good and bad situations, and increased knowledge would increase human competence. The natural way for a fixed chess algorithm to be more competent would be through more search. We could rely on Moore's Law and obtain faster hardware in the future.
Of course, increasing knowledge and increasing search are not mutually-exclusive. Doing both should dramatically improve the competence of our programs.

Intelligent search requires that we use the best searching algorithm for the circumstance. Recall that all search algorithms should begin at an initial state, apply operators to traverse the state space, and aim for an initial state. All search techniques have to keep track of a list of states that are candidates for visiting next. In a large search space this list can become big quickly. All search techniques have a policy for choosing which state from that list to visit next. This choice effects not only the optimality of the found solution but also the whether or not the algorithm is guaranteed to find a solution if one exists.

Search algorithms are generally divided into two classes. Informed search techniques make educated guesses about where to go next based on domain knowledge. Such techniques require a heuristic function to tell them how preferable various states are so it knows which to visit first. If no such heuristic function exists one can apply uninformed search techniques to search the space in a systematic fashion regardless.

Popular uninformed search algorithms include depth first search, breadth first search, uniform cost search, and iterative deepening depth first search. Depth first search starts at the initial state, pushes all states it could go to next on a stack, and pops a state to visit. If that state is not the goal state it then pushes all states it can do to next from that state on to the stack and then pops a new one. When it reaches a state with no successor states it can backtrack by popping the stack again to get the next successor.

Depth first search has relatively good space requirements for the depth that it goes. It does this by bypassing many other states it could go to (at least temporarily) in favor of following just one. Thus, if it is unlucky then it may pass up the goal state. Further, if the state space is infinite (or very large relative to your space and time constraints) then it might miss a goal state altogether.

Breadth first search starts at the initial state and puts successor states in a first-in-first-out queue. This has the effect of making the algorithm search all states at rank one away from the initial state before going on to the rank two states, and of searching all of those before going on to rank three, etc.

Breadth first search's systematic exploration of all rank \( n \) states before going to rank \((n+1)\) states guarantees it will find a goal state (if one exists). Further, it will find the shallowest such goal state, and when the goal is to minimize the number of steps from the initial state this one will be the optimal one. Breadth first search achieves this at the cost of memory. As it explores the states of rank \( n \) it must keep those states of rank \((n+1)\) in its queue. If there is a relatively consistent number of successor states of each given state then the queue will have
to store exponentially more states as the search progresses to deeper ranks.

**Uniform cost search** is a generalization of breadth first search where there is a cost associated with each operator. Uniform cost search uses a priority queue to always choose the cheapest next state to visit rather than the strick first-in-first-out queue employed by breadth first search. Because it visits all states from “cheapest” to increasingly more “expensive” it will first the best state first. Also like breadth first search its queue is expected to grow exponentially.

**Iterative deepening depth first search** combines the space efficiency of depth first search with the shortest-ranked goal finding ability of breadth first search. It does this by doing a depth first search only to the first rank. If a goal state has not been found yet then it does a depth first search to the second rank. If a goal state still has not been found yet then it does a depth first search to the third rank, etc.

Iterative deepening depth first search is space efficient and optimal goal finding (where “optimal” means “shortest path from the initial state”) but obtaining this was not free. This search algorithm wastes time revisiting states it already has visited. For example, assume that we finally found a goal state at rank \( n \). To get there we had to gradually work our way to that rank from ranks 2, 3, 4 . . . \((n-1)\). All of that time was wasted.

The informed search methods have the potential for doing better because they use heuristic function to guess how good successor states are. Like uniform cost search, informed search algorithms use a priority queue to arrange the set of potential next states to explore.

**Greedy best first search** enqueues the states by some estimate of how close that state is to the goal state. If this estimated cost was the true cost then it would find the optimal path. But if we were that lucky to have that accurate a heuristic function then we could solve the problem without search!

\(A^*\) heuristic function sums the known cost of getting to a particular state used by uniform cost search with the estimated cost of getting from that state to the goal state used by greedy best first search. If the heuristic is good\(^1\) then this is an “optimal” search in the sense that, like uniform cost search it always finds the optimal goal, and like greedy best first it homes in on the goal state efficiently.

In the discussion above we have considered searching from the initial state headed toward a goal state. One could, however, do the opposite: start at a goal state and try to work backwards to the initial state. In chess this would correspond to visualizing your opponent in checkmate and then working backward to see how you could maneuver him or her into that position. Additionally, one can search simultaneously from initial state to goal state, and from goal state to initial state, to try to link in the middle. This technique has the potential to reduce the explored portion of the search space.

### 5.2 An Overview of Machine Learning

Machine learning (ML) is the area of A.I. that deals with an algorithm's ability to improve its

---

\(^1\) A “good” \(A^*\) heuristic never overestimates the cost to the goal.
performance at some set of tasks given more experience. It uses A.I.'s search paradigm in conduction with other goals, like being accurate in performance and concise in notation.

“Improving performance” is a very general goal. One could argue that numerical methods take a bad initial guess at coefficients and turn them into good guesses, thus broadening ML's roots beyond AI. This is not quite the mentality of ML, though. Recall the A.I. definition of “algorithm”; if some technique exists such that one can stuff in data, turn the crank, wait $O(n^3)$ time and receive the correct (or “optimal”) answer then it is considered a “boring” problem.

ML algorithms are often given data from which to learn some predictive data structures. This data is generally in the form of vectors of attribute values. There are $N$ attributes $\text{attr}_1, \text{attr}_2, \ldots, \text{attr}_N$. Each one of these attributes has some legal set of values. Vectors of values detail instances (objects or events) in the domain about which the user wants predictions. Many learning algorithms create data structures that predict the values of one attribute given the values of the others. Consider the data below with attributes Outlook, Temp, Humidity, Windy, and PlayTennis. Each vector details an instance of weather conditions and whether or not tennis was played.

| Outlook  | Temp(F) | Humidity(%) | Windy | PlayTennis?
|----------|---------|-------------|-------|--------------
| sunny    | 75      | 70          | true  | yes          |
| sunny    | 80      | 90          | true  | no           |
| sunny    | 85      | 85          | false | no           |
| sunny    | 72      | 95          | false | no           |
| sunny    | 69      | 70          | true  | yes          |
| overcast | 72      | 90          | true  | yes          |
| overcast | 83      | 78          | false | yes          |
| overcast | 64      | 65          | true  | yes          |
| overcast | 81      | 75          | false | yes          |
| rain     | 71      | 80          | true  | no           |
| rain     | 65      | 70          | true  | no           |
| rain     | 75      | 80          | false | yes          |
| rain     | 68      | 80          | false | yes          |
| rain     | 70      | 96          | false | yes          |

A good example of a proper ML problem is finding the most concise decision tree consistent with some data. A decision tree is a data structure for predicting some learning attribute’s value given the values of several test attributes. For example, with the data above it is natural to try to predict the values of attribute PlayTennis (the learning attribute) given the values for the weather conditions Outlook, Temp, Humidity and Windy (the test attributes). One starts at the tree’s root and follows the labeled arcs down towards the leaves. At each non-leaf node, one computes an expression given at that node. (Often this is just the value of a test attribute). One then takes the arc that matches the answer. This process stops when on reaches a leaf. Leaves are labeled with the same value for the query attribute.
The problem of finding the “optimal” (meaning “smallest”) tree has been proved NP-complete (see Hyafil and Rivest). Rather than an exponential search for the best tree, many algorithms employ a heuristic that generally finds good trees. Information theory is employed to see which test attribute best splits a set of training instances into distinct learning attribute values. A greedy algorithm chooses that attribute and splits the instances according to its values. The results more, smaller sets of instances, each of which is homogeneous in the value of the attribute that was picked to split.

Our goal is to create a (hopefully) small tree that can predict the learning attribute’s values. If all (or most) instances in a set have the same value then we need not proceed any further. We may make a leaf labeled with that learning attribute value.

However, if there is sufficient disagreement among the instances about the learning attribute’s value then the attribute choice and splitting process should be recursively applied to this smaller set. This procedure continues until either all instances are sufficiently homogeneous, or until we have no more test attributes with which to split the instances. If we run out of instances first then we make a leaf and label it with the majority learning attribute value.

This whole procedure requires an information-theoretic heuristic to attribute choice. Let $S$ be a set of training instances. Let $C_1, C_2, \ldots, C_k$ be subsets of $S$ with some common property (e.g. all have the same value of the learning attribute). Let $\text{sizeof}(S)$ denote how many instances are in $S$, and $\text{freq}(C_i, S)$ denote how many instances of $S$ are in $C_i$ (i.e. $\text{sizeof}(C_i)$).

Let us gain an intuition before plowing into the equations. We would like to know the expected amount information needed to specify a sequence of instances that occurs in $S$. This sequence can be composed of instances $C_i$ according to their given frequencies $\text{freq}(C_i, S)$. Consider some how much information it takes to encode one particular $C_i$:

- If there are few $C_i$’s (close to 0) then this will take on average few bits. Each instance when it does occur will require more than one bit to note. However, because each event is rare the expected information will be less than one bit. The rarer it is, the less the expected information.
If there are many $C_i$'s (close to $\text{sizeof}(S)$) then this will also take few bits. Because instances are so common they are a good default guess. Depending on how common they are one bit can represent the occurrence of multiple $C_i$ instances. Thus each occurrence will take less than one bit.

If about half the instances are $C_i$'s (close to $\text{sizeof}(S)/2$) then this will take about one bit. Whether or not $C_i$ occurs is about a 50-50 random event. Because of this no compression is possible and we will need to invest one bit to record the occurrence (or non-occurrence) of $C_i$.

More formally, the probability of one subset $C_i$ is:

$$p(C_i|S) = \frac{\text{freq}(C_i,S)}{\text{sizeof}(S)}$$

therefore the information needed to specify one occurrence of $C_i$ is:

$$\text{info}_{\text{single}}(C_i) = -\log\left( \frac{\text{freq}(C_i,S)}{\text{sizeof}(S)} \right) \text{ bits}$$

We compute the expected amount of information by multiplying the information needed to specify one occurrence by its expected probability. The information for all of $S$ can then by obtained by summing over all subsets.

$$\text{info}(S) = \sum \left( \frac{\text{freq}(C_i,S)}{\text{sizeof}(S)} \right) \times -\log\left( \frac{\text{freq}(C_i,S)}{\text{sizeof}(S)} \right)$$

Here the variable $i$ ranges from 1 to $k$, the number of subsets.

Let us pause again to regain an intuition. Consider a case where $k = 2$, and that in 10 instances every one is in $C_1$ and none in $C_2$. That is, $\text{sizeof}(S) = 10$, $\text{freq}(C_1,S) = 10$, $\text{freq}(C_2,S) = 0$. Therefore:

$$\text{info}(S) = \frac{\text{freq}(C_1,S)}{\text{sizeof}(S)} \times -\log\left( \frac{\text{freq}(C_1,S)}{\text{sizeof}(S)} \right) + \frac{\text{freq}(C_2,S)}{\text{sizeof}(S)} \times -\log\left( \frac{\text{freq}(C_2,S)}{\text{sizeof}(S)} \right)$$

$$= \frac{(10/10) \times -\log(10/10)}{\text{sizeof}(S)} + \frac{(0/10) \times -\log(0/10)}{\text{sizeof}(S)}$$

$$= \frac{0 \times -\log(10/10)}{\text{sizeof}(S)} + \frac{0 \times -\log(0/10)}{\text{sizeof}(S)}$$

$$= 0 + 0$$

$$= 0 \text{ bits}$$

The intuition behind this value is “If we know that we’re dealing with $S$, then we know that all of it’s members are in $C_1$. There is no need to specify which is $C_1$ and which is $C_2$”

Consider another case again with $k = 2$ and 10 instances but with half of the members in $C_1$ and half in $C_2$. That is, $\text{sizeof}(S) = 10$, $\text{freq}(C_1,S) = 5$, $\text{freq}(C_2,S) = 5$. Therefore:

$$\text{info}(S) = \frac{\text{freq}(C_1,S)}{\text{sizeof}(S)} \times -\log\left( \frac{\text{freq}(C_1,S)}{\text{sizeof}(S)} \right) + \frac{\text{freq}(C_2,S)}{\text{sizeof}(S)} \times -\log\left( \frac{\text{freq}(C_2,S)}{\text{sizeof}(S)} \right)$$

$$= \frac{(5/10) \times -\log(5/10)}{\text{sizeof}(S)} + \frac{(5/10) \times -\log(5/10)}{\text{sizeof}(S)}$$

Copyright (c) 2008 by Joseph Phillips. All Rights Reserved.
The intuition behind this value is “If we know that we’re dealing with \( S \), then it’s a 50-50 guess which members belong to \( C_1 \) and which to \( C_2 \). We need to specify to which class each instance belongs. No compression is possible.”

Now we may apply our information theory understanding to pick the most informative attribute. Recall we want to best separate the instances according to their learning attribute’s values. There already is some distribution of learning attribute values, and associated expected amount of information to identify a random instance’s learning attribute value. We want to choose find the test attribute that divides the instances into sets, each of which needs less expected information (i.e. is more homogeneous) to identify a random instance’s learning attribute value.

We define information gain as:

\[
\text{gain}(\text{attr}) = (\text{ave info to specify a learning val}) - (\text{ave info to specify a learning val after partition by attr}) = \text{info}(T) - \text{info}(T,\text{attr})
\]

where:

- \( n \) = number of distinct values that attribute \( \text{attr} \) has
- \( T_i \) = set where all members have same \( \text{attr} \) value \( v_i \)
- \( \text{info}(S,\text{attr}) = \sum (\frac{\text{sizeof}(T_i)}{\text{sizeof}(T)}) \times \text{info}(T_i) \)

where \( i \) goes from 1 to \( n \). When \( \text{info}(T,\text{attr}) \) is small the classes well separated, and \( \text{gain}(\text{attr}) \) is large.

We may use the gain equation to choose the (locally) best attribute. Revisiting the data on weather and tennis we can compute the gain for \( \text{Outlook} \):

\[
\text{info}(\text{PlayTennis}) = -\frac{9}{14} \times \lg(\frac{9}{14}) - \frac{5}{14} \times \lg(\frac{5}{14}) = 0.940 \text{ bits}
\]

\[
\text{info}(\text{PlayTennis,Outlook}) = \frac{5}{14} \times (-\frac{2}{5} \times \lg(\frac{2}{5}) - \frac{3}{5} \times \lg(\frac{3}{5})) + \frac{4}{14} \times (-\frac{4}{4} \times \lg(\frac{4}{4}) - 0/4 \times \lg(0/4)) + \frac{5}{14} \times (-\frac{3}{5} \times \lg(\frac{3}{5}) - \frac{2}{5} \times \lg(\frac{2}{5}))
= 0.694 \text{ bits}
\]

\[
\text{gain}(\text{Outlook}) = 0.940 - 0.694 = 0.246 \text{ bits}
\]

We can compare that with the gain for \( \text{Windy} \):

\[
\text{info}(\text{PlayTennis}) = -\frac{9}{14} \times \lg(\frac{9}{14}) - \frac{5}{14} \times \lg(\frac{5}{14}) = 0.940 \text{ bits}
\]

Copyright (c) 2008 by Joseph Phillips. All Rights Reserved.
info(PlayTennis, Windy) = \frac{6}{14} \times (-\frac{3}{6} \times \log_2(\frac{3}{6}) - \frac{3}{6} \times \log_2(\frac{3}{6})) + \frac{8}{14} \times (-\frac{6}{8} \times \log_2(\frac{6}{8}) - \frac{2}{8} \times \log_2(\frac{2}{8}))
= 0.892 \text{ bits}

gain(Windy) = 0.940 - 0.892 = 0.048 \text{ bits}

Because $gain(\text{Outlook}) > gain(\text{Windy})$ we prefer to split on Outlook.

We may apply this algorithm to our data to create a predictive decision tree, but how good is it? In the example above we had only 14 examples. Just as we expect humans to gain more knowledge with more experience we could expect our ML algorithms to do better with more experience. This is detailed with the curve labeled “performance on training data” in figure below.

![Performance on test data](image)

(The accuracy may not get to 0 with an infinite number of examples because of noise in the data. The data might have inconsistencies like two instances of identical weather conditions with one case where we do play tennis and another when we do not.)

The result is slightly different if we use our tree to predict different data. Accuracy on this “test” data also improves as the training set grows, but then we find that it starts doing worse the larger the training data set.

This problem is called **overfitting**: after a while the learning algorithm learns non-generalized details unique to the training data.

We would like to learn the general patterns in the data without overfitting the peculiarities in the data. Further we would like to use all of our instances to train.

**Cross validation** is a procedure that facilitates these goals. We divide the data into $L$ sets. For each set we use all of the other data as training data from which to learn the model. We then use the isolated set to test it.

We can use the example and discussion above to reify the main tenets of machine learning.

1. **Gathering training data.** In the example above we used the 14 instances of weather conditions and PlayTennis values. There is an art to picking the right training data.
One issue is setting some aside to test the resulting model. For classification tasks like above it is also a good idea to gather data that both covers the full space of expected value vectors and that represents each class about an equal number of times. This latter condition might differ significantly from the data vectors that you see in real life. For example, consider wanting to distinguish between situations labeled safe and dangerous. In real life 99.9% of the data might be labeled safe. ML algorithms, however, tend to do best when about half of the training data is labeled safe and about half dangerous.

2. Choosing a target function. Decision trees are popular for learning classifications. Sometimes, however, we need more information like a number. Such situations could call for a neural network but only if we do not care for a deeper understanding of the structure of the solution.

3. Choosing a representation. The choice of representation is an important consideration in both science and ML. Ideally, we want the “right” representation that lets us identify what is important and ignore other details. In the tennis example above, instead of the simple variable Windy we could have given a WindVelocity attribute that also encodes the wind’s direction: positive values for “blowing to the east” and negative values for “blowing to the west”. This extra information might confuse our learning algorithm, however, or require it to have more instances to learn a structure of equivalent predictive ability.

4. Choosing an algorithm. In the example above we used a greedy algorithm and outline how it could be tested with cross validation. Other options exists though. We could have allowed for backtracking, or used a genetic algorithm to find a good tree. The choice of algorithm is intertwined with the choice for representation. Backpropogation is a common learning algorithm for neural networks.

5.3 Machine Learning and Computational Scientific Discovery

Machine learning (ML) is an established discipline from which computational scientific discovery (CSD) heavily borrows, but CSD other unique concerns. There are at least three such concerns:

1. In CSD we are not trying to supplant scientists but to work with them. What the computer brings to the relationship is speed, accuracy and the inability to get bored. For applying ML to CSD this means exhaustive search.

2. The process of ML is iterative, but the process of CSD is iterative and ongoing.

3. CSD uses background knowledge. Unlike ML accuracy is not necessarily the primary goal. CSD should also be guided by the dominant paradigm, at least when doing Normal Science.

Let us discuss these in turn.

Computers are naturally useful to scientists because of their speed, accuracy and tenacity. Thus, computers are best employed for repetitive computations. Last chapter we considered doing simulations as a special type of prediction; simulations generally require many computations.
In this chapter we consider exhaustive search as a machine learning technique. Like simulations, exhaustive search requires repetitive computations. Search should be exhaustive so as not to miss anything, but we can do it intelligently. We can search from most probable to least probable. If we make the Occam's Razor assumption that “simpler” means “more probable” then we can search from simplest model to increasingly more complex.

How “exhaustive” should exhaustive search be? Consider a model with at least one floating point parameter \( x \), to which we can add some small non-zero change \( dx \). Varying this parameter, \( x, x + dx, x + 2\times dx, x + 3\times dx, etc., \) could be considered making a new model. Obviously searching all such “models” is prohibitively expensive and unnecessary.

We should settle then for searching all qualitatively different models. Within each qualitatively unique model state we can use calculus and other numerical methods techniques to approximate the best fitting model parameters.

Secondly, both ML and CSD are iterative, but CSD is on going. Like most model building efforts, ML modelers often work in close conjunction with the intended users. A tentative model is built, the user comments on it, the model is then refined. This cycle may go on for several iterations before an adequate model is created.

In CSD models are also iteratively refined in close conjunction with their users (scientists). The models, however, are not necessarily just artifacts to hand off to scientists before finding a new project, but starting points for the next revision of the model. Scientists may revise models until the paradigm finally changes.

Finally, both CSD and ML rely on two things: the data and an inductive bias. Inductive bias is necessary because after seeing \( N \) training cases the \( N+1^{th} \) test case can truly be anything. The inductive bias tells the ML algorithm what can be assumed about the target function. This allows the ML algorithm to focus on the aspects of the training cases that will (hopefully) generalize to the test cases.

A variety of inductive biases are used in ML. We have discussed minimal cross-validation
error. Additionally there is minimum description length (a quantification of Occam’s Razor), maximal conditional independence (for Bayesian nets), maximal boundary size between classes (for support vector machines), minimal feature usage (the ignoring of extraneous data), and locality (for k-nearest neighbor algorithm).

However, in addition to some accuracy-rewarding inductive bias like minimal cross-validation error and some conciseness quantification inductive bias like minimum description length, CSD also needs to use the paradigm. The current paradigm becomes an input into the inductive bias, biasing models to conform to it.

The current paradigm can bias model choice in a variety of ways. It can prefer models that
- are expressible in the same language as the paradigm,
- re-use paradigm elements instead of positing something new,
- re-use the types of reasoning used in the paradigm instead of using a novel form.

A last point is that because the model fits into the paradigm and because Kuhnian “normal science” is iterative, the model can subtly change the paradigm. For example, a scientist may find some new representation, method of calculation or some new type of object to study that fits within the current paradigm. This may impress his or her peers so much that they abandon their prior way of promoting the paradigm in favor of this new way.

5.4 Penalties for Novel Objects

The preference implicit in a paradigm for “object conservatism”, the re-use of existing objects in scientific models, is broad across the sciences and across types of objects. Object conservatism does not mean that new concepts and terms never introduced. Rather,
combinations of believed objects should be convincingly exhausted before one resorts to inventing new objects.

In what follows below I list four examples of “object conservatism” for three different types of objects: physical objects, attribute and processes.

### 5.4.1 Penalty for New Objects 1: Polywater

**Background**

A *polymer* is a long, repetitive chain of atoms. For example, polyethylene is a chain of carbon atoms, where each carbon atom is attached to another before it, another behind it, and to hydrogen atoms of its own.

**The Story**

In 1962, Soviet scientist Nikolai Fedyakin condensed water in and forced water through narrow quartz capillary tubes. He measured properties of the water after this procedure. Its boiling point had increased beyond 100 C. Its freezing point had decreased below 0 C. Also, it was more viscous than ordinary water. Its properties were more similar to syrup than water.

Fedyakin said that he took precautions to keep the water clean. Thus, this had to be hither to unknown form of water. Water is bland stuff, what could have happened to it? Increased boiling point and viscosity (but *increased* freezing point rather than *decreased* freezing point) are all hallmarks of a larger molecule. This new form or water was given several names including “anomalous water”, “orthowater” and *polywater*.

Another Soviet scientist, Boris Derjaguin, met Nikolai Fedyakin and took the results to Moscow and then to the West, to the United Kingdom in 1966.

This finding caused a stir in the West. Some could replicate findings. Others could not.

Eventually Denis Rousseau of Bell Labs tried to replicate the experiment with his post-handball sweat. He found it had similar properties as the supposed “polywater”. Soon it became accepted that the “polywater” phenomenon was due to trace organic impurities in the water.

### 5.4.2 Penalty for New Objects 2: Piltdown Man

**Background**

There was a great interest in finding bones of ancient people in Europe around 1900. When bones of people who came to be known as *Neanderthals* were found in Germany in 1863 people did not know *what* they were. Certainly no living German had such a large nose, and prominent supraorbital ridge (bony ridge above the eyes). Who were these people? Were they Cossacks wounded during the Napoleonic Wars who crawled in to a cave to die? Anatomically, Cossacks are like Germans, and like all modern humans.

Eventually it dawned on scientists that this must be a form of “ancient European” when
bones were compared with other findings: skulls from Engis Belgium (1829) and Gibraltar (1848).

In 1868 in France more cave-dwelling “ancient Europeans” were found. The Cro-Magnon were closer to modern human and lived (as determined by their surviving artifacts) a culturally-richer life. Among other things they painted the macro fauna of their environment.

“Ancient Europeans” had been found in Germany. “Ancient Europeans” had been found in France. What about Great Britain!?

The Story

In 1912 Charles Dawson claimed at the Geological Society of London to be given a skull by men in at Piltdown gravel pit in Piltdown, East Sussex, England. Dawson found some further fragments, reconstructed them, and presented them. His reconstruction had a relatively small brain size, so it appeared to be somewhat between apes and modern man.

Arthur Smith Woodward (center) and Charles Dawson (right) excavating the Piltdown gravels in 1911.

The skepticism began relatively soon. The Royal College of Surgeons made their own reconstruction and declared that the cranial case looked like that of modern man. In 1915 French paleontologist Marcellin Boule said that the jaw was from some ape. The American zoologist Gerrit Smith Miller's findings were similar, that it was from a fossil ape. In 1923 after the First World War German anatomist Franz Weidenreich stated that it was a modern human cranium and orangutan jaw with filed teeth.

Thereafter, at least in the English speaking world, Piltdown Man was part of the pantheon of ancient humans, but detached from the others. How did he fit in? No one knew.

Finally, in 1953 Oxford anthropologist Kenneth Page Oakley conclusively determined that the skull was from a medieval human, the lower jaw from a Sarawak orangutan, and the teeth from a fossil chimpanzee.

Piltdown Man had been a fraud. Since that time people have speculated that Arthur Conan Doyle, creator of the Sherlock Holmes detective story series, invented the whole thing to let Dawson make a fool of himself.
5.4.3 Penalty for New Attributes: Inertial Versus Gravitational Mass

Background and Story

We can conceptualize at least three different types of mass. First is inertial mass: the resistance bodies have to move. This is given by the m in the equation \( F = ma \). Rewriting it to \( a = F/m \) we see that for fixed force \( F \), if \( m \) increases then the resulting acceleration \( a \) on the object will decrease.

We can also conceptualize two different forms of gravitational mass. Passive gravitational mass is the ability of a body to be attracted to another body. Consider a person Suspending a chair above the floor. The gravitational field of the Earth pulls on the passive mass of the chair, and thus a person must exert a counter force to keep the chair in the air. The little \( m \) in the gravitational equation \( F = GMm/r^2 \) (for \( M \gg m \)) denotes passive gravitational mass.

Lastly, active gravitational mass is the ability a body has to attract another body. In the Earth-pulling-on-chair example it is the Earth’s active gravitational mass that attracts the chair. The large \( M \) in the gravitational equation \( F = GMm/r^2 \) (for \( M \gg m \)) denotes active gravitational mass.

We can conceptual three different forms of mass, but does Nature distinguish among them? From Issac Newton’s formulation of mass, to Albert Einstein’s explanation of gravity as mass-warped space-time, people have speculated about this and have done experiments to see if there are any discrepancies.

So far it appears that all conceptualizations of mass are indistinguishable. Thus, physicists since Newton just call it one thing: “mass”.

5.4.4 Penalty for New Processes: Cold Fusion

Background

“Cold Fusion” is probably a poor name for the supposed phenomenon identified by Pons and Fleischmann. Perhaps a better term is “catalyzed fusion”. Let us briefly go over “catalysis” and “fusion”.

Catalysis

Catalysis is the process of some substance acting to hasten a reaction without itself being consumed by that reaction. The theory among chemists about how it works is the following.

Supposed there is some reaction where \( A \) and \( B \) combine to form \( D \). It does go, but only slowly:

\[
A + B \rightarrow D
\]
If we introduce a catalyst C then chemists posit that it speeds the reaction by binding (strongly or weakly) with A and B, and by constraining them into conformations that favor their reaction. The finished product D then detaches from C, leaving it available to catalyze another A and B pair.

\[
\begin{align*}
A + C & \rightarrow A-C & \text{(activated catalyst)} \\
B + A-C & \rightarrow A-C-B & \text{(ready to go)} \\
A-C-B & \rightarrow C-D & \text{(easier reaction)} \\
C-D & \rightarrow C + D & \text{(C ready to use again)}
\end{align*}
\]

From an energetic viewpoint the reaction speeds up because it lowers the energy that the reagents need to complete the reaction. Normally molecules and atoms have a range of energies characterized by the **Boltzmann Distribution**. The higher the temperature the more the energetic molecules or atoms, but at any temperature there will always be a mix of some fast-moving (i.e. hot) particles with some slow-moving (i.e. slow) particles.

Molecules have to reconfigure their electrons and atoms to complete a reaction. This requires some minimum threshold of energy (i.e. speed, or temperature). Only those molecules moving fast enough and at the correct orientation relative to each other can complete the reaction.

Chemists believe that catalysts reduce the energy that the reagents need to react, or hold them in the correct orientation relative to each other, or both. In so doing they increasing the chances of the successful completion of reactions. The macroscopic manifestation of this is an increased reaction rate.

**Nuclear Fusion**

Nuclear fusion is the process where light but neutron-rich nuclei combine to form a heavier nucleus. When “light” (i.e. small, low atomic number, close nuclearly with hydrogen) nuclei do this large amounts of energy are released. This process is most energetic for neutron-rich forms of hydrogen. It becomes less energetic for heavier atoms on up to Iron (atomic number 26). (Beyond Iron one obtains more energy from the decomposition of nuclei via fission, with the heavy atoms Uranium and Plutonium being notable extremes.)
One gets a lot of energy out (for small nuclei) but first one has to put a lot of energy in. This is because of the nuclei themselves. Nuclear forces keep nuclei together and (with sufficient number of neutrons around to help) overcome the electrostatic repulsion of the nucleus' protons. These nuclear forces, however, only are strong over very short distances. At longer distances the electrostatic forces (mutual repulsion among protons) prevail. Protons, very dense concentrations of positive charge, do not want to approach each other.

Thus nuclear fusion only occurs if you really beat on and squeeze the nuclei in a forced marriage. It occurs naturally in stars where their gravitational pressure in their center generated by their mass overcomes the electrostatic repulsion. And, in the early 1950s, both the United States and the Soviet Union figured out how to accomplish man-made fusion at about the same time: by confining the Hydrogen in a very small room with a very big bomb. Hydrogen bombs are generally initiated with fission bombs: the fission bomb is needed to generate the pressure and temperature necessary to force the fusion fuel to compress it sufficiently to achieve fusion.

2 After Deuterium and Tritium are finally tortured into getting married they find they love each other after all.
3 Your author is a pacifist-leaning vegetarian and does not advocate the design and manufacture of nuclear weapons. The design of a Hydrogen bomb is included, however, to make a point.
Fusion weapons, like fission weapons, have been around for half a century but there is no equivalent fusion-powered power plant. Fusion is more powerful than fission so a fusion power plant could make more energy than a modern nuclear power plant. Hydrogen is easier to obtain than Uranium so its fuel would be cheaper. The by-product Helium-4 is used in balloons and is not toxic and non-radioactive, so running a fusion power plant would be mean less waste than a modern nuclear power plant.

The United States, Japan, the Europeans and the Soviet Union all have tried to tame and harness fusion. So far all have failed. The reaction is generally initiated with a bank of lasers that uses the energy of a small city. The extremely hot plasma is generally attempted to contain with very strong magnetic fields. The fusion plasma, however, is hard to contain. It squeezes out, cools off, and stops reacting within a fraction of a second.

**The Story**

By 1989, when Americans Martin Fleischmann and Stanley Pons announced on the front page of the *New York Times* that they had achieved fusion *at room temperature* using standard laboratory equipment, several governments had already spent the equivalent of billions of dollars in big fancy labs with big fancy lasers and big fancy plasma-containment setups.

*At room temperature!*

*No initiating atomic bomb or big laser bank!*

*Controlled, without some huge-ass magnetic field to keep the plasma in place!*

Had they done all of this they would have solved the problems of global warming, fossil fuel depletion, nuclear waste buildup, and the cash outflow industrial countries have to energy-producing countries in one blow.

What they did was to electrolyze (electrically split) heavy water (Deuterium oxide or D₂O, Deuterium is a heavier form of Hydrogen with a neutron added). When they did this they
observed more energy coming out than energy they placed in. They used Palladium electrodes and hypothesized that this metal was somehow catalyzing the reaction.

After the story ran in the *New York Times* it became and worldwide sensation. Scientists around the world tried to replicate the result. Some said they could. Others said they could not. Some sited cold fusion as the reason why Jupiter appeared to be emitting an anomalouos amount of radio energy. Others wondered how Palladium could overcome the large electrostatic repulsion between Deuterium atoms to get them to fuse.

If the reaction did go then there should be two products: more energy and Helium-4. Some claimed that they also got excess energy. Others claimed that those who said they got excess energy really had not accounted for all the energy they themselves were putting in. Some claimed that they got Helium. Others claimed that those who said they obtained Helium did not account for the minute amount in the air due to radioactive decay.

After a year or so with no definitive results the controversy died down, and it appeared to be a case of over-eager scientists being insufficiently cautious about their claims. Research in this area has continued with still claiming more energy or more Helium than expected, but the hype and strong claims have gone.

### 5.5 Summary

Science is conservative. It prefers to re-use old elements of the paradigm rather than posit brand new objects, attributes and processes. Of course, scientists do posit new objects, but other scientists hold them to account and ask if existing objects could be arranged to account for the same data.

Thus, scientists are not *necessarily* slaves to fitting the data. They must also fit the paradigm as well. This is done by using the paradigm's objects and attributes in combinations allowed by the paradigm to explain data the paradigm specifies as worthy of prediction.

### References: