James Le Follow
Software Engineering || Product Management || Data Science (https://jameskle.com/)
Jan 20 · 16 min read

12 Useful Things to Know about Machine Learning

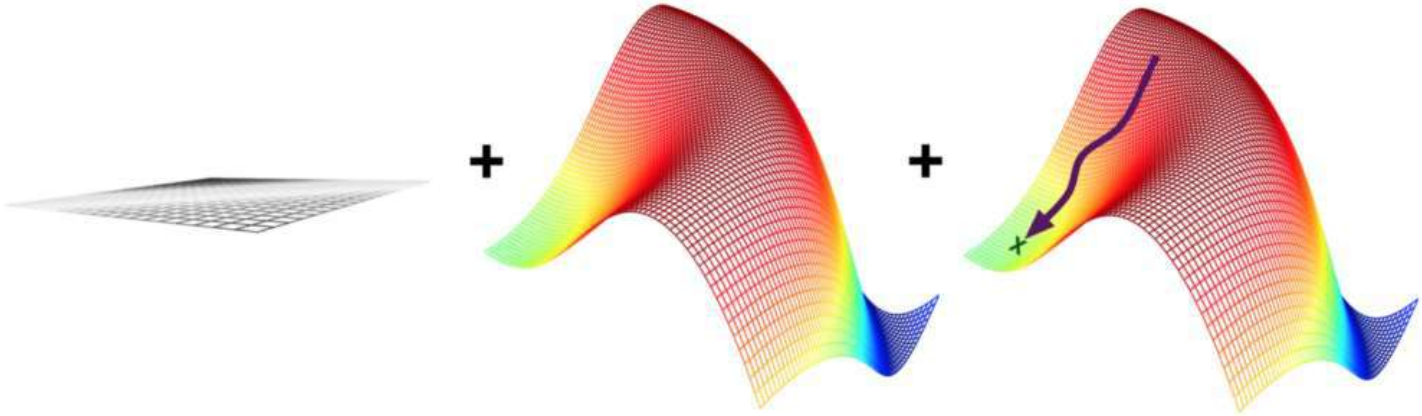
Machine learning algorithms can figure out how to perform important tasks by generalizing from examples. This is often feasible and cost-effective where manual programming is not. As more data becomes available, more ambitious problems can be tackled. As a result, machine learning is widely used in computer sincere and other fields. However, developing successful machine learning applications requires a substantial amount of "black art" that is hard to find in textbooks.

I recently read an amazing technical paper by Professor Pedro Domingos of University of Washington titled "**A Few Useful Things to Know about Machine Learning.**" It summarizes 12 key lessons that machine learning researchers and practitioners have learned include pitfalls to avoid, important issues to focus on, and answers to common questions. I'd like to share these lessons in this article because they are extremely useful when thinking about tackling your next machine learning problems.

1—Learning = Representation + Evaluation + Optimization

All machine learning algorithms generally consist of combinations of just 3 components:

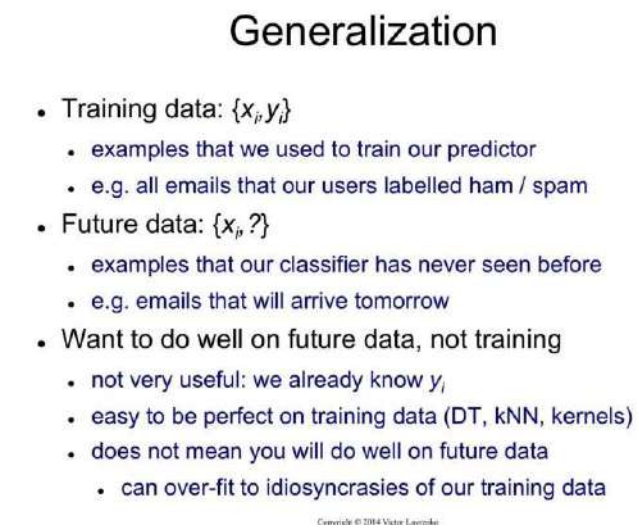
- **Representation:** A classifier must be represented in some formal language that the computer can handle. Conversely, choosing a representation for a learner is tantamount to choosing the set of classifiers that it can possibly learn. This set is called the *hypothesis space* of the learner. If a classifier is not in the hypothesis space, it cannot be learned. A related question is how to represent the input, i.e., what features to use.
- **Evaluation:** An evaluation function is needed to distinguish good classifiers from bad ones. The evaluation function used internally by the algorithm may differ from the external one that we want the classifier to optimize, for ease of optimization and due to the issues discussed in the next section.
- **Optimization:** Finally, we need a method to search among the classifiers in the language for the highest-scoring one. The choice of optimization technique is key to the efficiency of the learner, and also helps determine the classifier produced if the evaluation function has more than one optimum. It is common for new learners to start out using off-the-shelf optimizers, which are later replaced by custom-designed ones.



2—It's Generalization that Counts

The fundamental goal of machine learning is to *generalize* beyond the examples in the training set. This is because, no matter how much data we have, it is very unlikely that we will see those exact examples again at test time. Doing well on the training set is easy. The most common mistake among machine learning beginners is to test on the training data and have the illusion of success. If the chosen classifier is then tested on new data, it is often no better than random guessing. So, if you hire someone to build a classifier, be sure to keep some of the data

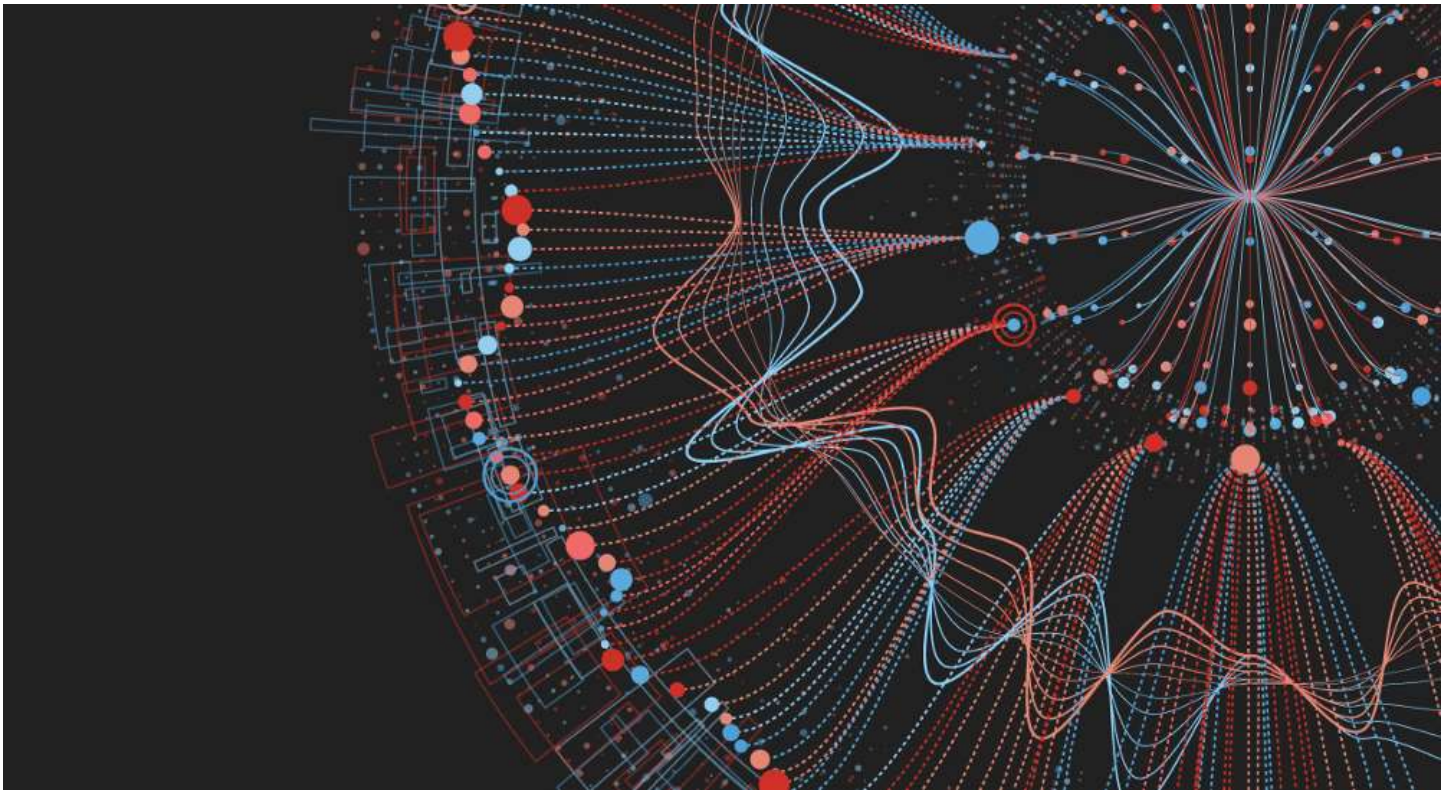
to yourself and test the classifier they give you on it. Conversely, if you've been hired to build a classifier, set some of the data aside from the beginning, and only use it to test your chosen classifier at the very end, followed by learning your final classifier on the whole data.



3—Data Alone is Not Enough

Generalization being the goal has another major consequence: data alone is not enough, no matter how much of it you have.

This seems like rather depressing news. How then can we ever hope to learn anything? Luckily, the functions we want to learn in the real world are not drawn uniformly from the set of all mathematically possible functions! In fact, very general assumptions—like smoothness, similar examples having similar classes, limited dependencies, or limited complexity—are often enough to do very well, and this is a large part of why machine learning has been so successful. Like deduction, induction (what learners do) is a knowledge lever: it turns a small amount of input knowledge into a large amount of output knowledge. Induction is a vastly more powerful lever than deduction, requiring much less input knowledge to produce useful results, but it still needs more than zero input knowledge to work. And, as with any lever, the more we put in, the more we can get out.



In retrospect, the need for knowledge in learning should not be surprising. Machine learning is not magic; it can't get something from nothing. What it does is get more from less. Programming, like all engineering, is a lot of work: we have to build everything from scratch. Learning is a more like farming, which lets nature do most of the work. Farmers combine seeds with nutrients to grow crops. Learners combine knowledge with data to grow programs.

4—Overfitting Has Many Faces

What if the knowledge and data we have are not sufficient to completely determine the correct classifier? Then we run the risk of just hallucinating a classifier (or parts of it) that is not grounded in reality, and is simply encoding random quirks in the data. This problem is called *overfitting,* and is the bugbear of machine learning. When your learner outputs a classifier that is 100% accurate on the training data but only 50% accurate on test data, when in fact it could have output one that is 75% accurate on both, it has overfit.

Everyone in machine learning knows about overfitting, but it comes in many forms that are not immediately obvious. One way to understand overfitting is by decomposing generalization error into *bias* and *variance.* Bias is a learner's tendency to consistently learn the same wrong thing. Variance is the tendency to learn random things irrespective of the real signal. A linear learner has high bias, because when the frontier between two classes is not a hyperplane the learner is unable to induce it. Decision trees don't have this problem because they can represent any Boolean function, but on the other hand they can suffer from high variance: decision trees learned on different training

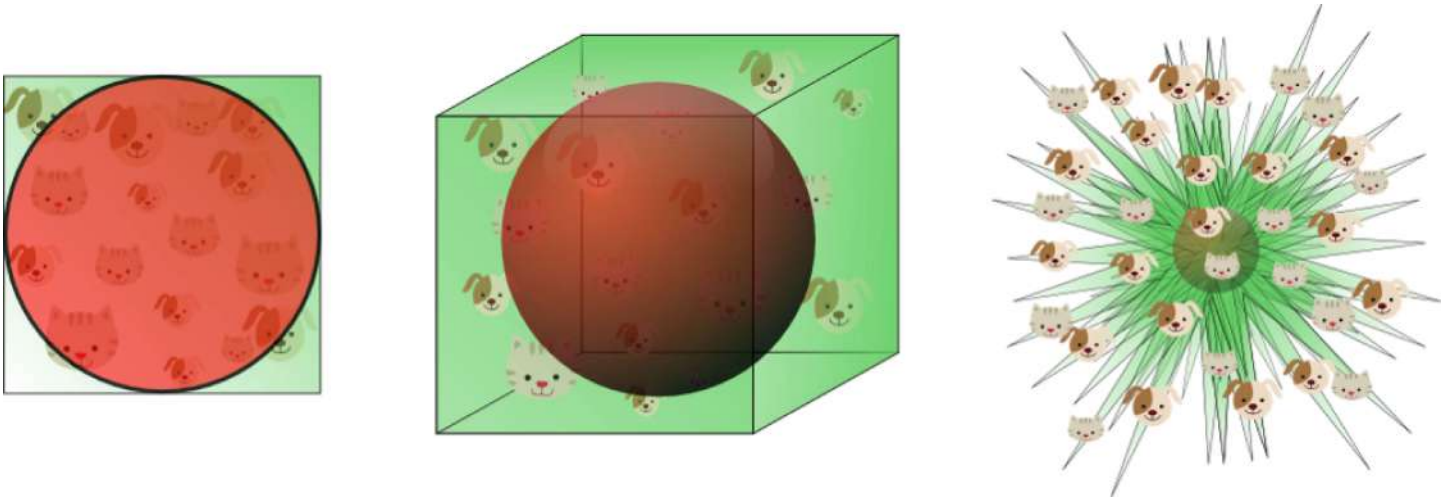
sets generated by the same phenomenon are often very different, when in fact they should be the same.

Cross-validation can help to combat overfitting, for example by using it to choose the best size of decision tree to learn. But it's no panacea, since if we use it to make too many parameter choices it can itself start to overfit.

Besides cross-validation, there are many methods to combat overfitting. The most popular one is adding a *regularization term* to the evaluation function. This can, for example, penalize classifiers with more structure, thereby favoring smaller ones with less room to overfit. Another option is to perform a statistical significance test like chi-square before adding new structure, to decide whether the distribution of the class really is different with and without this structure. These techniques are particularly useful when data is very scarce. Nevertheless, you should be skeptical of claims that a particular technique "solves" the overfitting problem. It's easy to avoid overfitting (variance) by falling into the opposite error of underfitting (bias). Simultaneously avoiding both requires learning a perfect classifier, and short of knowing it in advance there is no single technique that will always do best (no free lunch).

5—Intuition Fails in High Dimensions

After overfitting, the biggest problem in machine learning is the *curse of dimensionality*. This expression was coined by Bellman in 1961 to refer to the fact that many algorithms that work fine in low dimensions become intractable when the input is high-dimensional. But in machine learning it refers to much more. Generalizing correctly becomes exponentially harder as the dimensionality (number of features) of the examples grow, because a fixed-size training set covers a dwindling fraction of the input space.



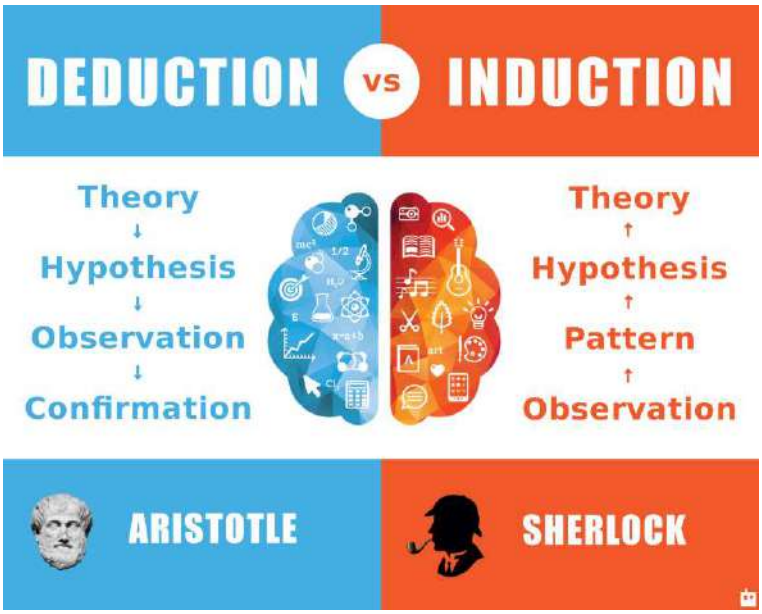
The general problem with high dimensions is that our intuitions, which come from a 3-dimensional world, often do not apply in high-dimensional ones. In high dimensions, most of the mass of a multivariate Gaussian distribution is not near the mean, but in an increasingly distant "shell" around it; and most of the volume of a high-dimensional orange is in the skin, not the pulp. If a constant number of examples is distributed uniformly in a high-dimensional hypercube, beyond some dimensionality most examples are closer to a face of the hypercube than to their nearest neighbor. And if we approximate a hypersphere by inscribing it in a hypercube, in high dimensions almost all the volume of the hypercube is outside the hypersphere. This is bad news for machine learning, where shapes of one type are often approximated by shapes of another.

Building a classifier in 2 or 3 dimensions is easy; we can find a reasonable frontier between examples of different classes just by visual inspection. But in high dimensions it's hard to understand what is happening. This in turn makes it difficult to design a good classifier. Naively, one might think that gathering more features never hurts, since at worst they provide no new information about the class. But in fact, their benefits may be outweighed by the curse of dimensionality.

6—Theoretical Guarantees are Not What They Seem

Machine learning papers are full of theoretical guarantees. The most common type is a bound on the number of examples needed to ensure good generalization. What should you make of these guarantees? First of all, it's remarkable that they are even possible. Induction is traditionally contrasted with deduction: in deduction you can guarantee that the conclusions are correct; in induction all bets are off. Or such was the conventional wisdom for many centuries. One of the major developments of recent decades has been the realization that in fact we can have guarantees on the results of induction, particularly if we're willing to settle for probabilistic guarantees.

We have to be careful about what a bound like this means. For instance, it does not say that, if your learner returned a hypothesis consistent with a particular training set, then this hypothesis probably generalizes well. What is says is that, given a large enough training set, with high probability your learner will either return a hypothesis that generalizes well or be unable to find a consistent hypothesis. The bound also says nothing about how to select a good hypothesis space. It only tells us that, if the hypothesis space contains the true classifier, then the probability that the learner outputs a bad classifier decreases with training set size. If we shrink the hypothesis space, the bound improves, but the chances that it contains the true classifier shrink also.



Another common type of theoretical guarantee is asymptotic: given infinite data, the learner is guaranteed to output the correct classifier. This is reassuring, but it would be rash to choose one learner over another because of its asymptotic guarantees. In practice, we are seldom in the asymptotic regime (also known as "asymptopia"). And, because of the bias-variance tradeoff discussed above, if learner A is better than learner B given infinite data, B is often better than A given finite data.

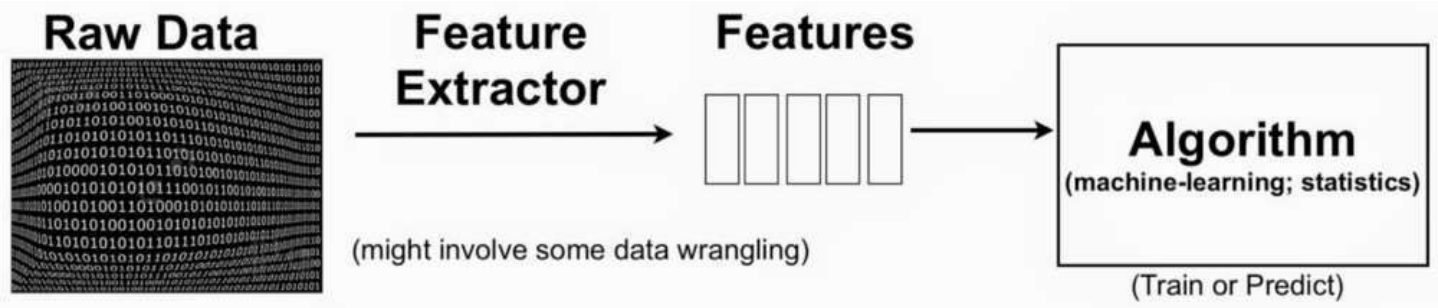
The main role of theoretical guarantees in machine learning is not as a criterion for practical decisions, but as a source of understanding and driving force for algorithm design. In this capacity, they are quite useful; indeed, the close interplay of theory and practice is one of the main reasons machine learning has made so much progress over the years. But *caveat emptor:* learning is a complex phenomenon, and just because a learner has a theoretical justification and works in practice doesn't mean the former is the reason for the latter.

7—Feature Engineering is the Key

At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor

is the features used. If you have many independent features that each correlate well with the class, learning is easy. On the other hand, if the class is a very complex function of the features, you may not be able to learn it. Often, the raw data is not in a form that is amenable to learning, but you can construct features from it that are. This is typically where most of the effort in a machine learning project goes. It is often also one of the most interesting parts, where intuition, creativity and "black art" are as important as the technical stuff.

First-timers are often surprised by how little time in a machine learning project is spent actually doing machine learning. But it makes sense if you consider how time-consuming it is to gather data, integrate it, clean it and pre-process it, and how much trial and error can go into feature design. Also, machine learning is not a one-shot process of building a dataset and running a learner, but rather an iterative process of running the learner, analyzing the results, modifying the data and/or the learner, and repeating. Learning is often the quickest part of this, but that's because we've already mastered it pretty well! Feature engineering is more difficult because it's domain-specific, while learners can be largely general-purpose. However, there is no sharp frontier between the two, and this is another reason the most useful learners are those that facilitate incorporating knowledge.

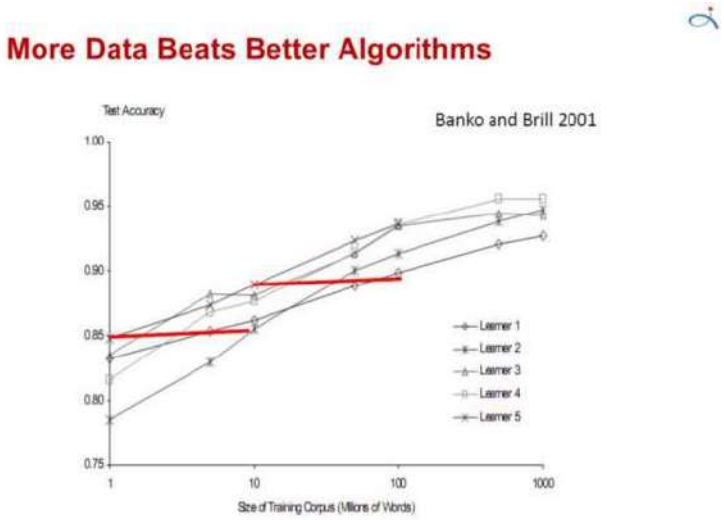


8—More Data Beats a Cleverer Algorithm

In most of computer science, the 2 main limited resources are time and memory. In machine learning, there is a third one: training data. Which one is the bottleneck has changed from decade to decade. In the 1980's, it tended to be data. Today it is often time. Enormous mountains of data are available, but there is not enough time to process it, so it goes unused. This leads to a paradox: even though in principle more data means that more complex classifiers can be learned, in practice simpler classifiers wind up being used, because complex ones take too long to learn. Part of the answer is to come up with fast ways to learn complex classifiers, and indeed there has been remarkable progress in this direction.

Part of the reason using cleverer algorithms has a smaller payoff than you might expect is that, to a first approximation, they all do the same. This is surprising when you consider representations as different as,

say, sets of rules and neural networks. But in fact propositional rules are readily encoded as neural networks, and similar relationships hold between other representations. All learners essentially work by grouping nearby examples into the same class; the key difference is in the meaning of "nearby." With non-uniformly distributed data, learners can produce widely different frontiers while still making the same predictions in the regions that matter (those with a substantial number of training examples, and therefore also where most text examples are likely to appear). This also helps explain why powerful learns can be unstable but still accurate.

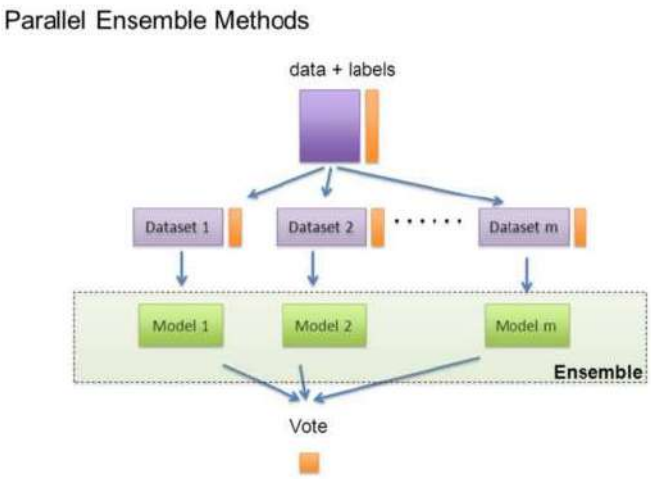


As a rule, it pays to try the simplest learners first (e.g., naive Bayes before logistic regression, k-nearest neighbor before support vector machines). More sophisticated learners are seductive, but they are usually harder to use, because they have more knobs you need to turn to get good results, and because their internals are more opaque).

Learners can be divided into 2 major types: those whose representation has a fixed size, like linear classifiers, and those whose representation can grow with the data, like decision trees. Fixed-size learners can only take advantage of so much data. Variable-size learners can in principle learn any function given sufficient data, but in practice they may not, because of limitations of the algorithm or computational cost. Also, because of the curse of dimensionality, no existing amount of data may be enough. For these reasons, clever algorithms—those that make the most of the data and computing resources available—often pay off in the end, provided you're willing to put in the effort. There is no sharp frontier between designing learners and learning classifiers; rather, any given piece of knowledge could be encoded in the learner or learned from data. So machine learning projects often wind up having a significant component of learner design, and practitioners need to have some expertise in it.

9—Learn Many Models, Not Just One

In the early days of machine learning, everyone had their favorite learner, together with some *a priori* reasons to believe in its superiority. Most effort went into trying many variations of it and selecting the best one. Then systematic empirical comparisons showed that the best learner varies from application to application, and systems containing many different learners started to appear. Effort now went into trying many variations of many learners, and still selecting just the best one. But then researchers noticed that, if instead of selecting the best variation found, we combine many variations, the results are better—often much better—and at little extra effort for the user.

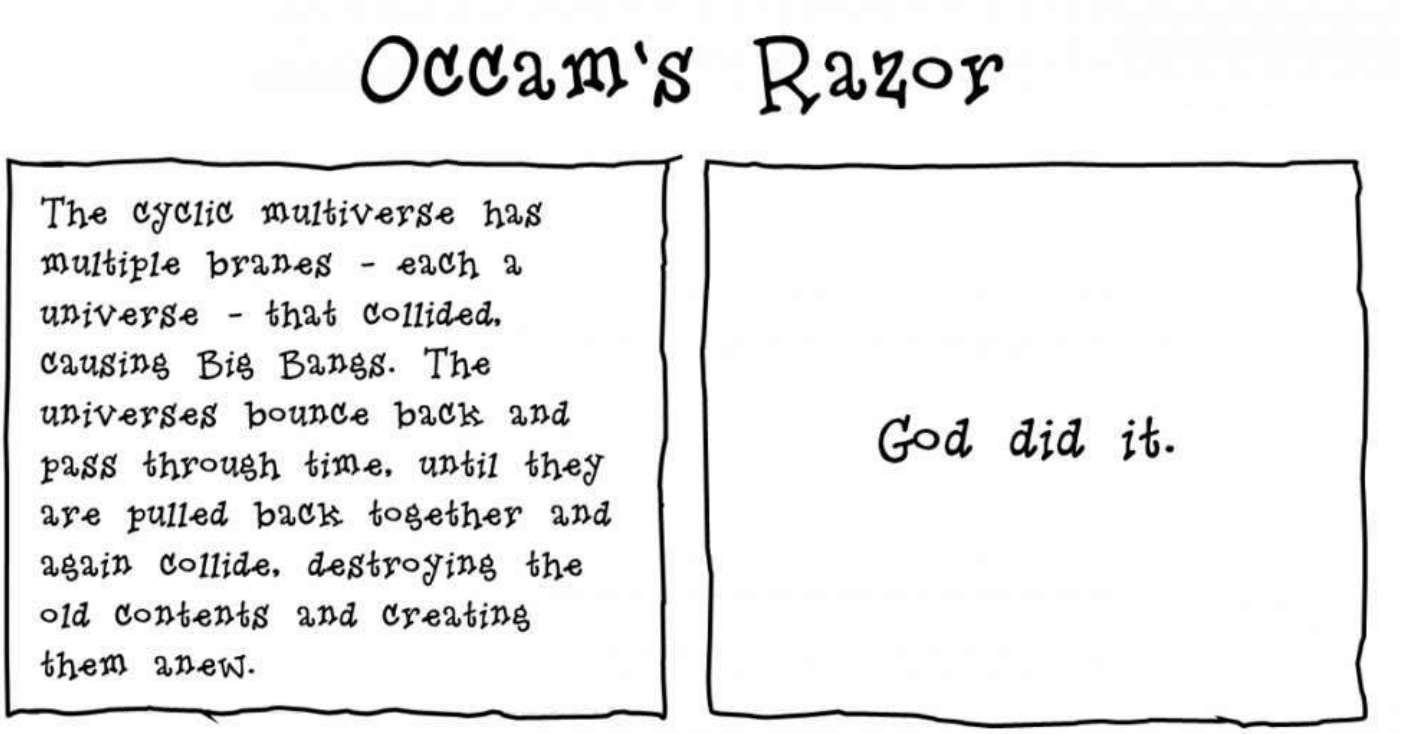


Creating such *model ensembles* is now standard. In the simplest technique, called *bagging,* we simply generate random variations of the training set by resampling, learn a classifier on each, and combine the results by voting. This works because it greatly reduces variance while only slightly increasing bias. In *boosting,* training examples have weights, and these are varied so that each new classifier focuses on the examples the previous ones tended to get wrong. In *stacking,* the outputs of individual classifiers become the inputs of a "higher-level" learner that figures out how best to combine them.

Many other techniques exist, and the trend is toward larger and larger ensembles. In the Netflix prize, teams from all over the world competed to build the best video recommender system. As the competition progressed, teams found that they obtained the best results by combining their learners with other teams', and merged into larger and larger teams. The winner and runner-up were both stacked ensembles of over 100 learners, and combining the two ensembles further improved the results. Doubtless we will see even larger ones in the future.

10—Simplicity Does Not Imply Accuracy

Occam's razor famously states that entities should not be multiplied beyond necessity. In machine learning, this is often taken to mean that, given two classifiers with the same training error, the simpler of the two will likely have the lowest test error. Purported proofs of this claim appear regularly in the literature, but in fact there are many counter-examples to it, and the "no free lunch" theorems imply it cannot be true.



We saw one counterexample in the previous section: model ensembles. The generalization error of a boosted ensemble continues to improve by adding classifiers even after the training error has reached zero. Thus, contrary to intuition, there is no necessary connection between the number of parameters of a model and its tendency to overfit.

A more sophisticated view instead equates complexity with the size of the hypothesis space, on the basis that smaller spaces allow hypotheses to be represented by shorter codes. Bounds like the one in the section on theoretical guarantees above might then be viewed as implying that shorter hypotheses generalize better. This can be further refined by assigning shorter codes to the hypothesis in the space that we have some *a priori* preference for. But viewing this as "proof" of a tradeoff between accuracy and simplicity is circular reasoning: we made the hypotheses we prefer simpler by design, and if they are accurate it's because our preferences are accurate, not because the hypotheses are "simple" in the representation we chose.

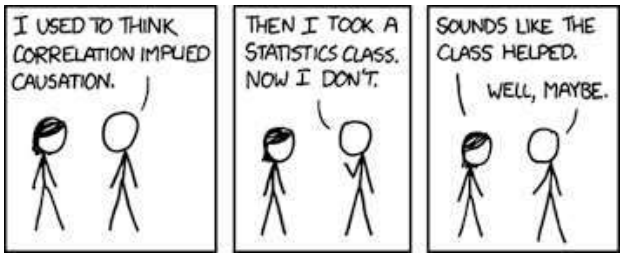
11—Representable Does Not Imply Learnable

Essentially all representations used in variable-size learners have associated theorems of the form “Every function can be represented, or approximated arbitrarily closely, using this representation.” Reassured by this, fans of the representation often proceed to ignore all others. However, just because a function can be represented does not mean it can be learned. For example, standard decision tree learners cannot learn trees with more leaves than there are training examples. In continuous spaces, representing even simple functions using a fixed set of primitives often requires an infinite number of components.

Further, if the hypothesis space has many local optima of the evaluation function, as is often the case, the learner may not find the true function even if it is representable. Given finite data, time and memory, standard learners can learn only a tiny subset of all possible functions, and these subsets are different for learners with different representations. Therefore the key question is not “Can it be represented?”, to which the answer is often trivial, but “Can it be learned?” And it pays to try different learners (and possibly combine them).

12—Correlation Does Not Imply Causation

The point that correlation does not imply causation is made so often that it is perhaps not worth belaboring. But, even though learners of the kind we have been discussing can only learn correlations, their results are often treated as representing causal relations. Isn’t this wrong? if so, then why do people do it?



More often than not, the goal of learning predictive models is to use them as guides to action. If we find that beer and diapers are often bought together at the supermarket, then perhaps putting beer next to the diaper section will increase sales. But short of actually doing the experiment it's difficult to tell. Machine learning is usually applied to *observational* data, where the predictive variables are not under control of the learner, as opposed to *experimental* data, where they are. Some learning algorithms can potentially extract causal information from observational data but their applicability is rather restricted. On the other hand, correlation is a sign of a potential causal connection, and we can use it as a guide to further investigation.

Conclusion

Like any discipline, machine learning has a lot of "folk wisdom" that can be hard to come by, but is crucial for success. Professor Domingos' paper summarized some of the most salient items that you need to know.