Lesson 5

Studying Learning

5.1 Ways to Study Learning

The mainstream approach to studying learning is task-based. The idea is we obtain better learning algorithms if we can show a learning algorithm exhibits measurable performance improvement on data sets. This approach emphasizes running algorithms on specific data sets and measuring performance results.

The other approach is interested in understanding the *general behavior* of the learning algorithm. In this line of research, learning algorithms are studied with respect to different kinds of criteria. This approach emphasizes mathematical analysis of the behavior of algorithms in different kinds of settings.

Both methods are important; however, the former is mostly taught nowadays. This is one reason why this course focuses on the second method.

Niyogi (2006) comments on these two methods are instructive.

> Another aspect of the book is its focus on mathematical models where the relationship between various objects may be formally (provably) studied. A complementary approach is to consider the larger class of computational models where one resorts to simulations. Mathematical models with their equations and proofs and computational models with their programs and simulations provide different and important windows of insight into the phenomena at hand. In the first, one constructs idealized and simplified models but one can now reason precisely about the behaviour of such models and therefore be sure of one's conclusions. In the second, one constructs more realistic models but because of the complexity, one will need to resort to heuristic arguments and simulations. In summary, for mathematical models the assumptions are more questionable but the conclusions are more reliable — for computational models, the assumptions are more believable but the conclusions more suspect.

If our route to understanding learning is solely through measuring performance, then we neglect understanding the general behavior of our learning strategies. When we observe that a learning program A achieves some performance measure m on a test set T after being exposed to a set of training data D, it is not clear what we can conclude about A or the relationship between A, D and T because A might give very different results on other testing data T' and other training data D'. As Heinz and Riggle (2011) put it "The goal of determining which properties of the data critically underlie learnability ...is precisely why learning theory focuses mainly on the *properties of classes of languages* or the *general behaviour of specific algorithms*, as opposed to the specific behaviour of specific algorithms [emphasis in original]."

I agree with Niyogi that both are important and that they are complimentary approaches. However, given the propensity of the task-based performance approaches in conferences, articles, and classrooms, this view that Niyogi and I share appears to be very much in the minority. In my view, this course serves as a small but important counterweight.

5.2 Performance on Tasks

Figure 5.1 shows the general workflow of implementing and running a ML system on a corpus of data split into a training and testing sets, and evaluating the result. An important principle at work is that the testing and training sets should come from the same source, but they should not overlap. If we do well on a test set with this training set, what confidence do we have we will do well on some other corpus?

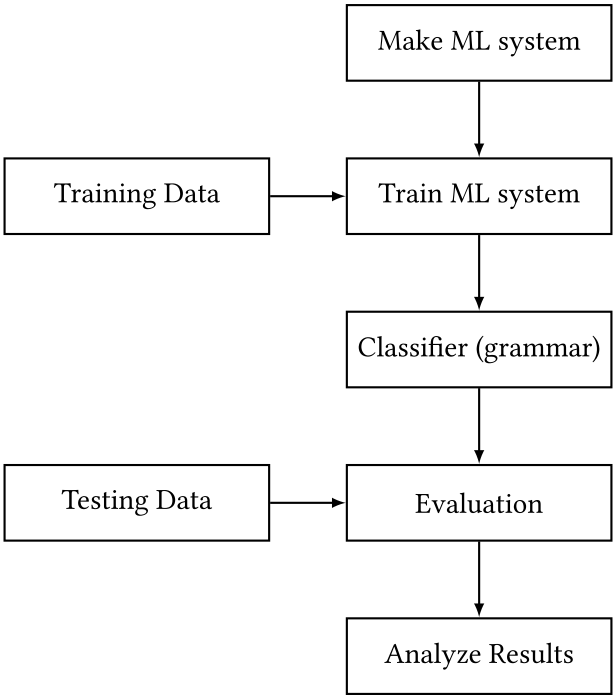

Figure 5.1: Empirical investigation of ML system performance on a corpus.

One method to increase our confidence without increasing our data is to use cross-fold validation. The idea is that we can take a given corpus and turn it into k-corpora by dividing it into k disjoint blocks. Then we can train on $k-1$ blocks and test on the leftover block. Figure 5.2 illustrates with $k = 10$. If we get comparable performance with cross-validation (for example the standard deviation is small) then we have more confidence that the initial result is not due to very particular aspects of the training corpus.

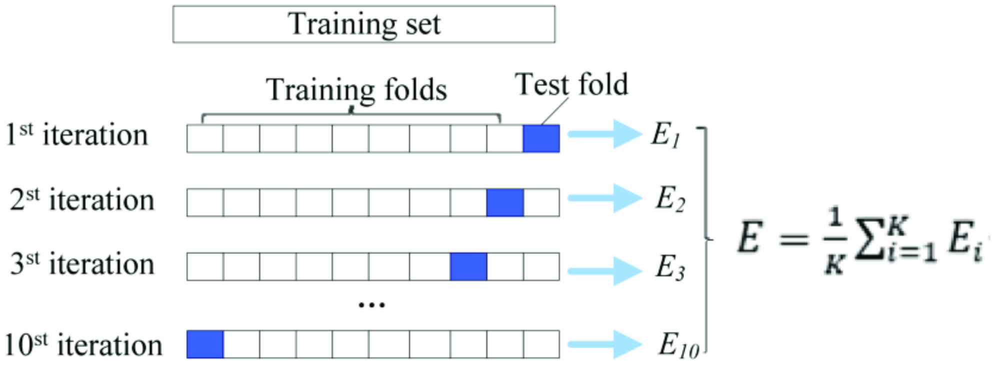

Figure 5.2: Illustration of cross-fold validation with 10 folds. (Source: DOI:10.3390/ijms19072071)

Of course the results may be due to aspects of the particular corpus that preserved across the different training folds. For this reason we may also want to repeat this entire procedure on different corpora. For ML systems designed to learn natural languages, for example, we may run the same ML algorithm on data sets from corpora collected from different languages.

One complication is that many ML systems have hyperparameters. A simple example is a n-gram language model. Which n will get the best result? We could train on several n values and report their results on the test set. However, this has the downside that we are using the test set for a dual purpose. We are using it to set the hyperparameters and we are using it to evaluate our best model. The test set should only be used to measure model performance, once it has been set.

So how do we set the hyperparameters? Well the idea is to split the corpus into three sets: training, validation, and testing. The validation set is sometimes called the development set. When finding the best hyperparameters, train the model and test it on the validation set. Repeat with different hyperparameter settings until you are satisfied. The process of finding good hyperparameter values is often called "tuning the model." Once satisfied, the model is evaluated against the testing set. Figure 5.3 shows a workflow incorporating the validations set. Often cross-fold validation is used in the model tuning phase (see https://www.statology.org/validation-set-vs-test-set/ for example.)

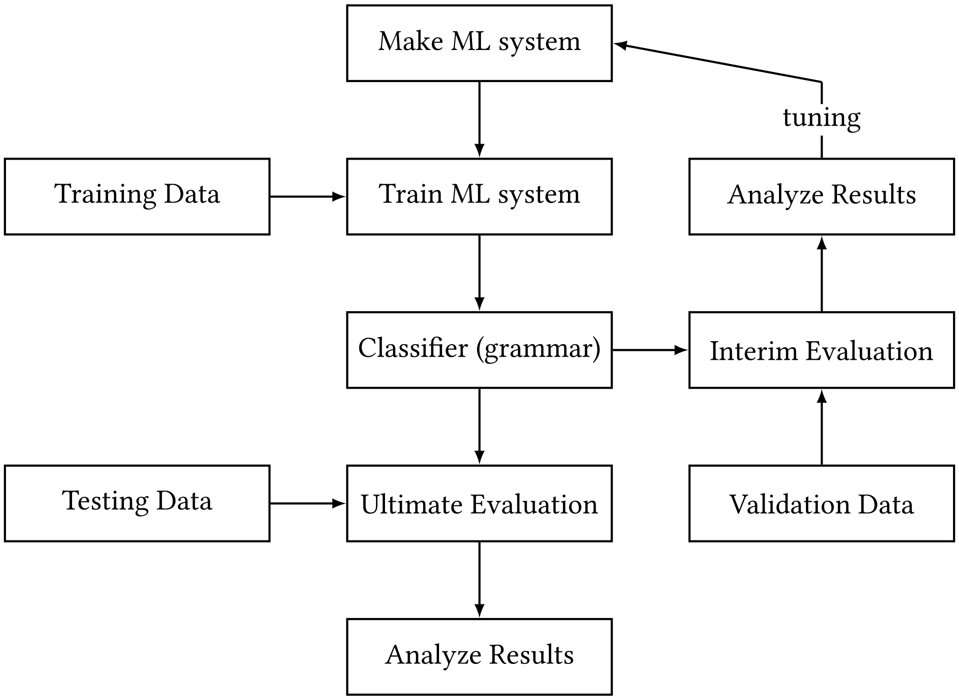

Figure 5.3: Empirical investigation of ML system performance on a corpus.

These procedures – use of validation set, cross-validation, testing on multiple corpora – do improve our confidence that the ML system is doing what we want. However, what about the next corpus? Are there any guarantees?

5.3 Behavioral Analysis

I take analysis of algorithms to be important precisely because it give us guarantees regarding the outcome of following some procedure. If there were no guarantees, there would be no field of computer science. Instead it would look a lot like experimental psychology. Let's see what happens when we run this experiment! This may be unfair to experimental psychology, and I don't doubt there are productive experimental paradigms in the psychological sciences. But that is not the same as knowing something *in advance* about the outcome of running some procedure.

Multiple criteria for understanding the general behavior of a ML system can be understood by observing its behavior over time, as training data is fed to it one piece at a time, as shown in Figure 5.4. There the notation $t[n]$ refers to the sequence $t(1), t(2), \dots t(n)$.

Time t	1	2	3	4	...	n	...
Evidence at time t	$t(1)$	$t(2)$	$t(3)$	$t(4)$	...	$t(n)$	...
Input to ML system at time t	$t[1]$	$t[2]$	$t[3]$	$t[4]$	...	$t[n]$	...
Output of ML system at time t	G_1	G_2	G_3	G_4	...	G_n	...
Language of G at time t	L_1	L_2	L_3	L_4	...	L_n	...

Figure 5.4: Analyzing a ML algorithm's behavior over time.

- A ML system **identifies a class of languages in the limit from positive data** iff for each language L in the class, for each positive presentation t of L, there is some n such that for all $m > n$, we have $G_m = G_n$ and $L_n = L$.
- A ML system **probably approximately identifies a class of concepts** iff for each concept C in the class, for each distribution D over the instance space, there is some n such that for all $m > n$, we have $\Pr[\texttt{error}_D(C, L_m) < \epsilon] > 1 - \delta$.
- A ML system **is a consistent estimator for a parametric model** iff for each parameter Θ in the model, if the data is randomly presented i.i.d. according to Θ, the ML system's estimate $\hat{\Theta}$ approaches the true value of Θ. (formally: $\forall \epsilon > 0, \exists n$ such that $\forall m > n$ $[\hat{\Theta}_m - \Theta < \epsilon]$.

Incidentally, historically consistent estimators are the oldest. I suspect Gold was influenced by the notion of consistent estimators in his development of the identification in the limit paradigm. The infinite sequence is an idealization, and it is not novel to Gold. Its true origin is statistics. Valiant developed the PAC framework to address perceived weaknesses in the identification in the limit paradigm.

These learning criteria are important, and while I would argue they are necessary to understanding the general behavior of a learning algorithm, I would also argue they are *not* sufficient. The reason they are not sufficient, is that they do not tell you about the behavior of the algorithm in the *short* term, only in the long term. How is the learning algorithm behaving before convergence? This is also an important aspect of the algorithm's general behavior.

Computational learning theory (and statistical learning theory) addresses short term general behavior as well. In the context of the identification in the limit paradigm, we have already seen some properties that can be attributed to a learning algorithms ϕ such as the following. For any finite set of strings D, let $G = \phi(D)$ and let $L = L(G)$. We assume a class of languages C.

- For any finite set D, we have $D \subseteq L$. (consistency)
- Whenever $D \subseteq D'$, we have $L \subseteq L'$. (monotonicity)
- For any finite set D, L is the smallest (closest?) language in C consistent with D. (formally: for all $L' \in C$ such that $D \subseteq L'$ then $L' \not\subseteq L$).

Many other similar kinds of statements have been studied (Jain *et al.*, 1999) and it can be useful to show your learning algorithms exhibit these kinds of behaviors.

As another example, consider algorithms ϕ that aim to learn the parameters Θ of a parametric model M which establishes a probability distribution over which the data is assumed to be generated in an i.i.d. fashion. Here we let $\phi(D) = \hat{\Theta}$.

- For any finite set of data D, we have, for all $\Theta' \in M$, the following: $\mathrm{Pr}_M(D \mid \hat{\Theta}) \geq \mathrm{Pr}_M(D \mid \Theta')$ (Maximum Likelihood Estimate; MLE).

Parametric models that maximize entropy also maximize likelihood (Murphy, 2012). Incidentally, estimators that maximize the likelihood are consistent estimators.

The important point I want to make here is that there are two aspects to understanding the general behavior of an algorithm: short term and long-term. The long-term behavior is a statement you can make about what will eventually happen with enough data. The short term statement says something about what happens given any data set. Often these two things go hand-in-hand. If you have an algorithm that gives you the MLE estimate, you have a consistent estimator. If you have an algorithm that gives you the smallest language in the class covering the data (and the class has Angluin's tell-tale property) then you have an algorithm that identifies the language in the limit from positive data.

Not all learning algorithms' short term behavior is understood. For example, we understand the long-term behavior of RPNI, but not it's short term behavior. If you haven't see the characteristic sample yet, we don't know what we can say about the quality of the DFA that is returned. All bets are off. The same goes (I believe) for its descendants which can learn probabilistic deterministic finite-state acceptors (ALERGIA), deterministic string-to-string transducers (OSTIA), and other such algorithms (e.g. SOSFIA). See de la Higuera (2010) for details on RPNI, ALERGIA, and OSTIA and Jardine *et al.* (2014) for SOSFIA.

Backing up a bit, a striking difference between all of these criteria (and these are not the only ones) and the performance-based approaches is the presence of the *class* of concepts, languages, parameter values that could be source of the data we are learning from, as well as the *range* of data-generating processes potentially underlying the data. These are necessary to analyze the general behavior of a learning algorithm because they are necessary to define the instance space of the computational problem.

- Which concepts to do you wish to learn?
- What kinds of data do you wish to learn them from?
- What counts as successful learning?

5.4 An Analogy with Maze Solving

I often use the analogy of sorting algorithms. Every library has algorithms that sorts lists. These algorithms were developed and their general behavior is well-understood. We can prove that the quicksort algorithm, for example, can sort *any* list and there are bounds on how much "work" it takes in terms of its time and space complexity.

Suppose instead we only had only tested sorting programs on some lists. Suppose we had a library of 10,000 lists and we had a program that could sort some of the at least mostly correctly.

It helps that sorting is a well-defined problem, and an algorithmic solution is not hard to develop and prove that it succeeds on any list.

Another analogy I want to try to draw here maze solving, which is a more complicated problem than sorting lists.

How can we solve a maze? We have to define the problem we are trying to solve.

- Which mazes to do you wish to solve?
- What kind of information do you have about the maze?
- What counts as success?

The analogy I want to draw is along these lines.

- formal language $\sim$ maze
- classes of languages $\sim$ classes of mazes
- learning strategy $\sim$ maze-solving strategy

Bibliography

Heinz, Jeffrey, and Jason Riggle. 2011. Learnability. In *Blackwell Companion to Phonology*, edited by Marc van Oostendorp, Colin Ewen, Beth Hume, and Keren Rice. Wiley-Blackwell.

de la Higuera, Colin. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press.

Jain, Sanjay, Daniel Osherson, James S. Royer, and Arun Sharma. 1999. *Systems That Learn: An Introduction to Learning Theory (Learning, Development and Conceptual Change).* 2nd ed. The MIT Press.

Jardine, Adam, Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Very efficient learning of structured classes of subsequential functions from positive data. In *Proceedings of the Twelfth International Conference on Grammatical Inference (ICGI 2014)*, edited by Alexander Clark, Makoto Kanazawa, and Ryo Yoshinaka, vol. 34, 94–108. JMLR: Workshop and Conference Proceedings.

Murphy, Kevin. 2012. *Machine Learning: A probabilistic perspective.* MIT Press.

Niyogi, Partha. 2006. *The Computational Nature of Language Learning and Evolution.* Cambridge, MA: MIT Press.