Lesson 01 - Introduction

What does 'learning' mean?

Learning: Data $\rightarrow$ Grammar

1. What is Data?
2. What is Grammar?
3. What does the arrow do and How does it do it?
4. Does the transformation the arrow represents have any properties we can identify?
5. What does it mean to say that a learning algorithm 'works' or 'is successful'?

Computational Problems

Problem: Question $\rightarrow$ Answer

Mathematically, one can think of a problem as a function that maps questions (instances of problems) to correct answers. Here are some examples.

Example: Computing the area of a rectangle

Question	Answer
$\ell = 3m, w = 4m$	$12m^2$
$\ell = 10m, w = 5m$	$50m^2$
...	...

Example: Sorting Lists

Question	Answer
[3,4,6,2,1]	[1,2,3,4,6]
[100,-99,98,101]	[-99,98,100,101]
...	...

Algorithms

An algorithm is a mechanical, step-by-step procedure that correctly solves a problem. That is, it can take any instance of the problem (question) as its input and, after going through the step-by-step procedure, it produces an output which is the correct answer to the question.

Well known algorithms exist for solving the above examples. For finding the area of the rectangle, one can multiply the two numbers together, and multiplication of two numbers is essentially a solved problem. For list sorting, there are all kinds of algorithms which can do this (bubble sort, merge sort, heap sort, quicksort and many others).

Algorithms for Learning

Question	Answer
?	?

Data	Grammar
?	?

1. What is the data?
2. What is the grammar?

In the above examples, the problems were *parameterized*. For finding the area of the rectangle, there was a range of lengths and widths to be considered. For list sorting, we implied that any list of integers was a valid 'question'. For language learning, we might ask What range of Data ought we consider?

Note that even for something as easy as computing the area of a rectangle, there are some not-so-innocent assumptions. For example, we may think that the inputs could be any pair of *real* numbers instead of any pair of integers. Doing so, however, runs into a real problem: most real numbers aren't real in the sense that they cannot be written down or effectively represented (Chaitin 2004). What we really have is multiplication of *rational* numbers, which allows us to approximate real numbers to any desired degree of precision.

Example: Maze Solving

Question	Answer
maze 1	a path through maze 1
maze 2	a path through maze 2
…	…

What kinds of mazes do we want to consider?

- Mazes with one entrance and one exit?
- Mazes with more than one entrance and more than one exit?
- Mazes with one entrance and zero or more exits?
- Mazes where you have a view from above versus mazes where you have 1st person view?
- Mazes where you can leave breadcrumbs?
- Mazes with minotaurs which must be avoided?
- Mazes on the surface of toruses, or mazes in normal Euclidean space?
- Mazes with wormholes or teleporting portals?
- Mazes whose walls change over time?

There are lots of ways to parameterize mazes! The same is true for Data for learning problems.

We can also think of what kinds of answers we want. For example, if no path exists, do we want the algorithm to tell us no path exists? Or is it OK if it just crashes or gives a wrong answer here?

Finally suppose we have identified a class of mazes and an algorithm that solves this class. That is, for any maze in the class, it produces a correct solution for how to escape the maze. We can also ask some other questions of this algorithm. 1. Given a maze M, can we say anything about how much time it takes to get out of it? 2. Given a maze M, can we say anything about how far we are from the exit as we move along the path? For instance, are we always getting closer, or sometimes are we taken farther away?

Answers to these sorts of questions tell us something about the behavior of our maze-escaping procedure.

Example: Estimating the probability of an unfair coin

We begin with the simplest parametric model I know of: the unfair coin. A fair coin has equal probability of landing heads or tails when flipped. An unfair coin has probability θ of landing heads and probability $1 - \theta$ of landing tails. So a fair coin is the special case of an unfair coin when $\theta = 0.5$. This probability θ is the sole parameter in our model of unfair coins. The value is fixed and does not change with time.

1. What is Data? One idea is that the data is a sequence of coin flips. For instance, our data may consist something like: HTHHTTH or TTTHHTHT.
2. What is the Grammar? One idea is that the grammar is just the parameter θ, whose true (computable) value we want to estimate. Thus, we know $0 \leq \theta \leq 1$.

Now we ask is there a procedure which can take Data and give us back correct values of θ. A commonly invoked procedure, which I will call the "OT ratio", estimates θ to be the ratio obtained by dividing the number of **Observed** head flips by the **Total** number of flips.

Questions 1. What are some issues with this approach? 2. How can these be resolved?

Limiting behavior of estimators

One way statisticians resolved the issues discussed above is to consider an *unending* sequence of coin flips. Importantly, each flip of the coin is **independent and identically distributed** (i.i.d), which means no one flip affects the probability of another (independence) and each flip has an identical distribution (so the probability of head is θ and that never changes). Then one can investigate the behavior of the estimator "in the limit".

Imagine a sequence of time steps, one after another. At each moment, the coin is flipped. At each moment a new estimate is formed based on *the sequence of flips up to the current moment*.

time	1	2	...	n	$n+1$	...
flip	flip_1	flip_2	...	flip_n	flip_{n+1}	...
estimate	S_1	S_2	...	S_n	S_{n+1}	...

In other words, S_n is not just based on flip_n but on the sequence shown below.

$$\text{flip}_1, \text{flip}_2, \ldots \text{flip}_n$$

Now one can ask about the behavior of S_n as n gets larger.

Definition: weak consistency

An estimator S for θ is said to be "weakly consistent" if it converges in probability to the true value of θ. Formally,

$$\text{for all } \epsilon > 0, \lim_{n\to\infty} P(|S_n - \epsilon|) < \epsilon$$

Definition: strong consistency

An estimator S for θ is said to be "strongly consistent" if it converges 'almost surely' to the true value of θ. Formally,

$$P(\lim_{n\to\infty} S_n = \theta) = 1$$

Theorems

1. Strong consistency implies weak consistency but not vice versa.
2. The estimator by OT ratio is weakly consistent.

3. The estimator by OT ratio is not strongly consistent.

Question: Suppose we flip the a coin 10 times. Is there any guarantee that S_n is close to the true value of θ?

Likelihood

The likelihood function L is the probability of the data given a value θ. For example, $L(HTH \mid \theta = 1/3) = 1/3 \times 2/3 \times 1/3 = 2/27$.

Theorem: For any data sequence, the estimate S_n produced by the OT ratio maximizes the likelihood of the data. In other words, for any data sequence $D = \text{flip}_1, \text{flip}_2, \ldots \text{flip}_n$, and for any $\hat{\theta}$ between 0 and 1 not equal to S_n, we have $L(D \mid S_n) \geq L(D \mid \hat{\theta})$.

For this reason, the estimator by OT ratio is commonly called the Maximum Likelihood Estimator (MLE).

General Lessons

1. It is useful to think about learning behavior over time, as data continually arrives.
2. "Long-term" aspects of behavior are important (what can we say about the limiting behavior of the learning algorithm).
3. "Short-term" aspects of behavior are important too (given finitely many data points, what can we say about the quality of the learner's output).
4. We want both kinds of theoretical results. In future classes we will see how past researchers have thought about these issues with respect to natural languages.