15-388/688 - Practical Data Science: Debugging data science

J. Zico Kolter
School of Computer Science
Fall 2019

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Data science debugging

Imagine this: you go about defining a data science problem, define input/output pairs for a prediction task, only to find that when you run some machine learning algorithm, it doesn't work

This happens to everyone

"What differentiates experts in data science from others is not what you do first, it's what you do second when that first thing doesn't work"

– Kotter's Law

Traditional debugging

Traditional debugging of programs is relatively straightforward

You have some desired input/output pairs

You have a mental model (or maybe something more formal) of how each step in the algorithm “should” work

You trace through the execution of the program (either through a debugger or with print statement), to see where the state diverges from your mental model (or to discover your mental model is wrong)

Data science debugging

You have some desired input/output pairs

Your mental model is that an ML algorithm should work because … math? ... magic?

What can you trace through to see why it may not be working? Not very useful to step through an implementation of logistic regression…

Debugging data science vs. machine learning

Many of the topics here overlap with material on “debugging machine learning”

We are indeed going to focus largely on debugging data science prediction tasks (debugging web scraping, etc, is much more like traditional debugging)

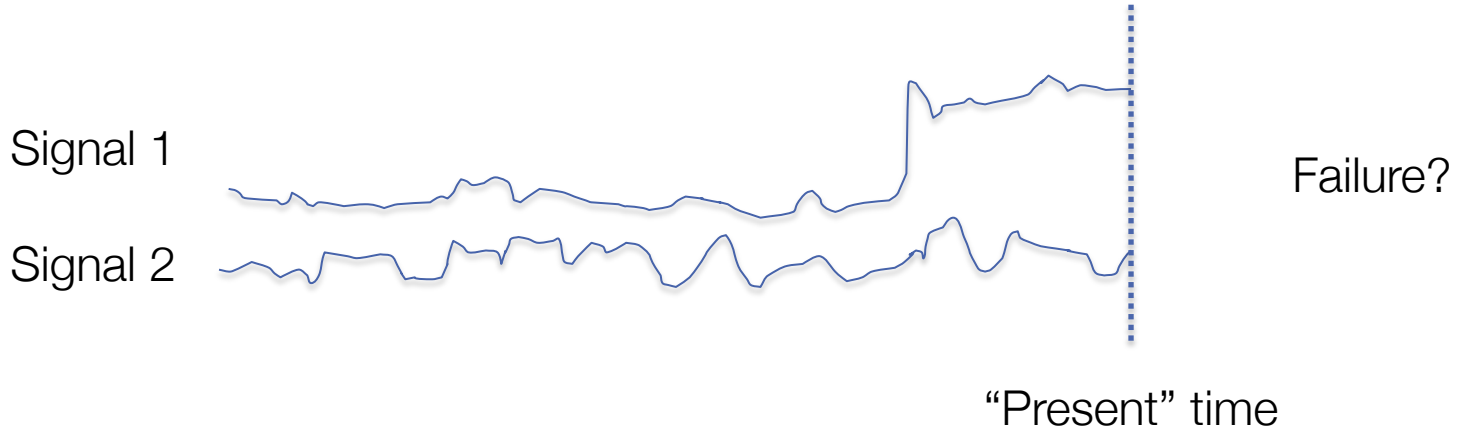
But, there is also a more abstract concept here of debugging the *problem* instead of debugging the *algorithm*

An example: predictive maintenance

An example task: you run a large factory and what to predict whether any given machine will fail within the next 90 days

You're given signals monitoring the state of this device

You want to predict binary response of whether the machine will fail



Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

The first step of data science debugging

Step 1: determine if your problem is impossible

There are plenty of tasks that would be really nice to be able to predict, and absolutely no evidence that there the necessary signals to predict them (see e.g., predicting stock market from Twitter)

But, hope springs eternal, and it's hard to prove a negative…

A good proxy for impossibility

Step 1: ~~determine if your problem is impossible~~ see if *you* can solve your problem manually

Create an interface where you play the role of the prediction algorithm, you need to make the predictions of the outputs given the available inputs

To do this, you'll need to provide some intuitive way of visualizing what a complete set of input features looks like: tabular data for a few features, raw images, raw text, etc

Just like a machine learning algorithm, you can refer to training data (where you know the labels), but you can't peak at the answer on your test/validation set

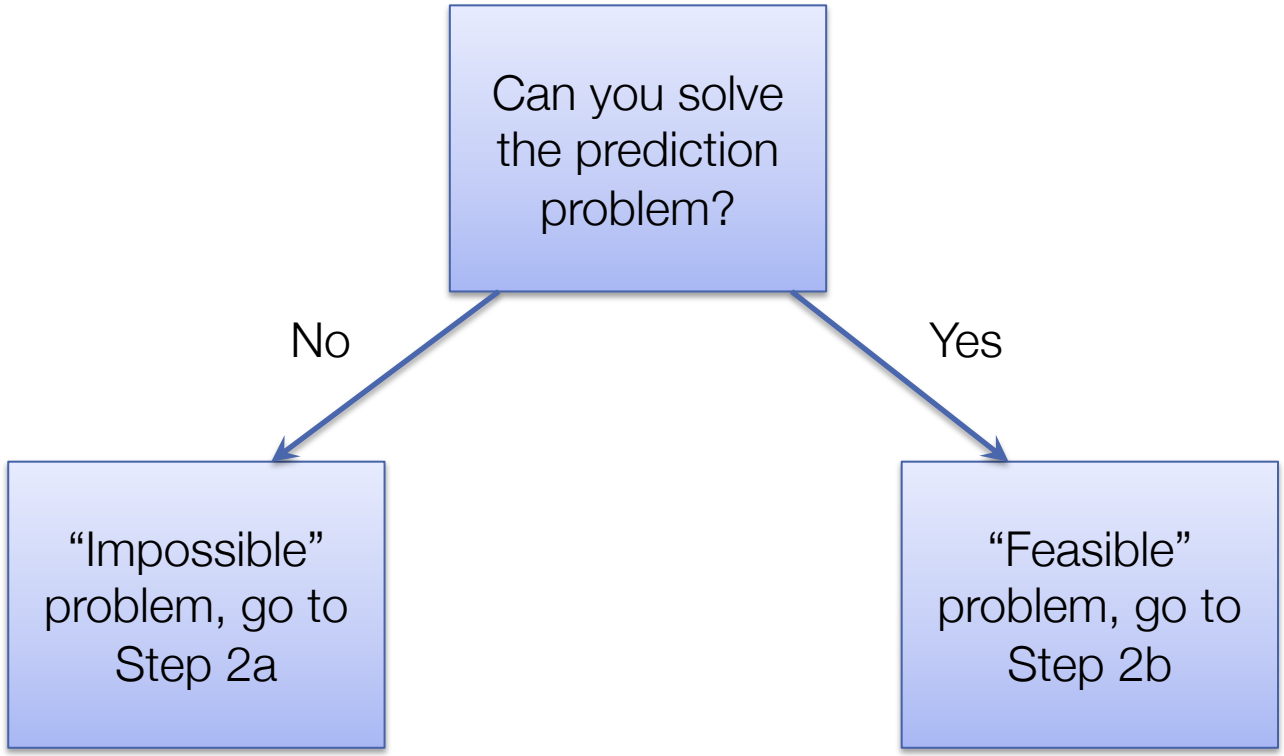
What about “superhuman” machine learning

It’s a common misconception that machine learning will *outperform* human experts on most tasks

In reality, the benefit from machine learning often doesn’t come from superhuman performance in most cases, it comes from the ability to scale up expert-level performance extremely quickly

If you can’t make good predictions, neither will a machine learning algorithm (at least the first time through, and probably always)

Decision diagram



Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Dealing with “impossible” problems

So you’ve built a tool to manually classify examples, run through many cases (or had a domain expert run through them), and you get poor performance

What do you do?

You do *not* try to throw more, bigger, badder, machine learning algorithms at the problem

Instead you need to change the problem by: 1) changing the input (i.e., the features), 2) changing the output (i.e., the problem definition)

Changing the input (i.e., adding features)

The fact that we can always add more features is what makes these problems "impossible" (with quotes) instead of impossible (no quotes)

You can always hold out hope that you just one data source away from finding the "magical" feature that will make your problem easy

But you probably aren't… adding more data is good, but:

1. Do spot checks (visually) to see if this new features can help *you* differentiate between what you were previously unable to predict

2. Get advice from domain experts, see what sorts of data source they use in practice (if people are already solving the problem)

Changing the output (i.e., changing the problem)

Just make the problem easier! (well, still need to preserve the character of the data science problem)

A very useful procedure: instead of trying to predict the future, try to predict what an expert would predict given the features you have available

E.g., for predictive maintenance this shifts the question from: “would this machine fail?” to “would an expert choose to do maintenance on this machine?”

With this strategy we already have an existence proof that it’s feasible

Changing the output #2

Move from a question of getting “good” prediction to a question of characterizing the uncertainty of your predictions

Seems like a cop-out, but many tasks are *inherently* stochastic, the best you can do is try to quantify the likely uncertainty in output given the input

E.g.: if 10% of all machines fail within 90 days, it can still be really valuable to predict if whether a machine will fail with 30% probability

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Dealing with feasible problems

Good news! Your prediction problem seems to be solvable (because you can solve it)

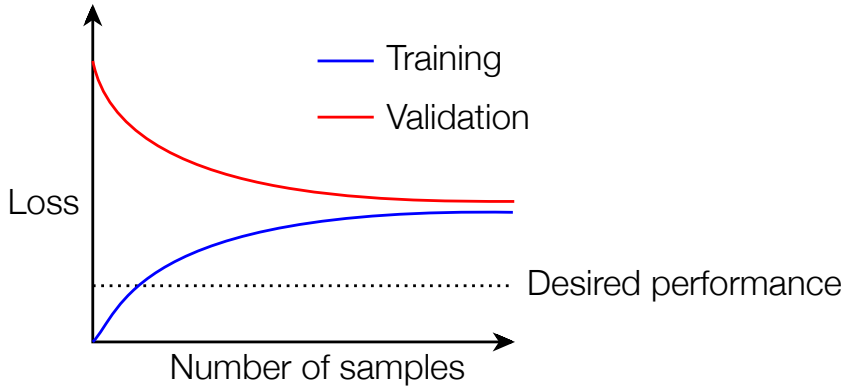
You run your machine learning algorithm, and find that it doesn't work (performs worse than you do)

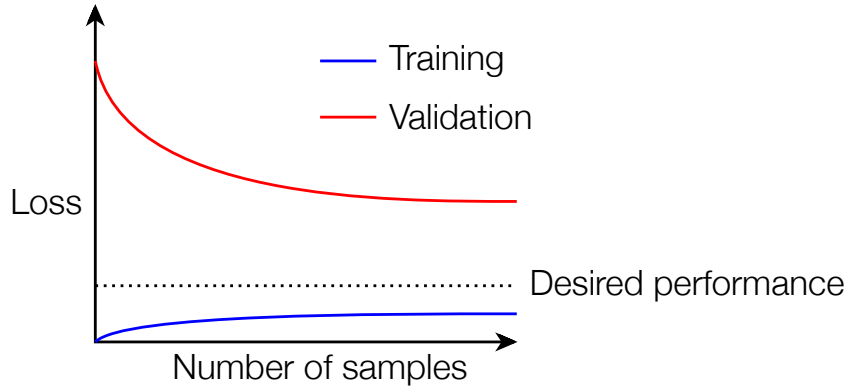
Again, you can try just throwing more algorithms, data, features, etc, at the problem, but this is unlikely to succeed

Instead you want to build diagnostics that can check what the problem may be

Characterizing bias vs. variance

Consider the training and testing loss of your algorithm (often plotting over different numbers of samples), to determine if you problem is one of high bias or high variance



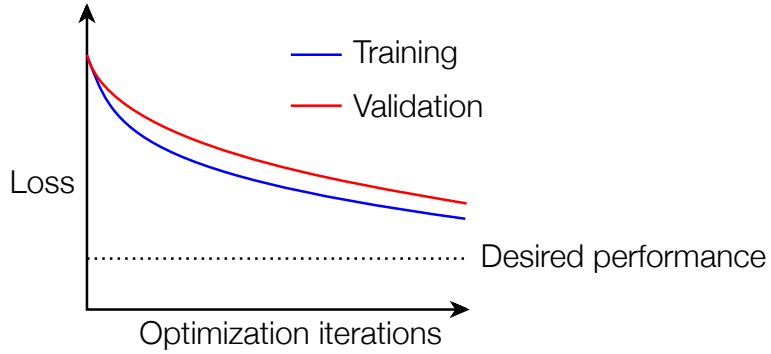


For high bias, add features based upon your own intuition of how you solved the problem

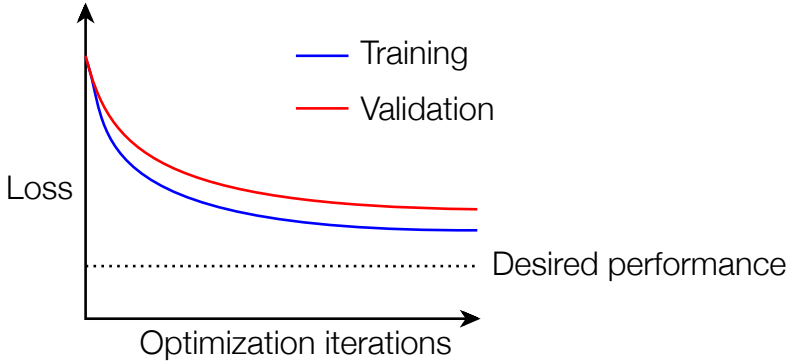
For high variance, add data or remove features (keeping features based upon your intuition)

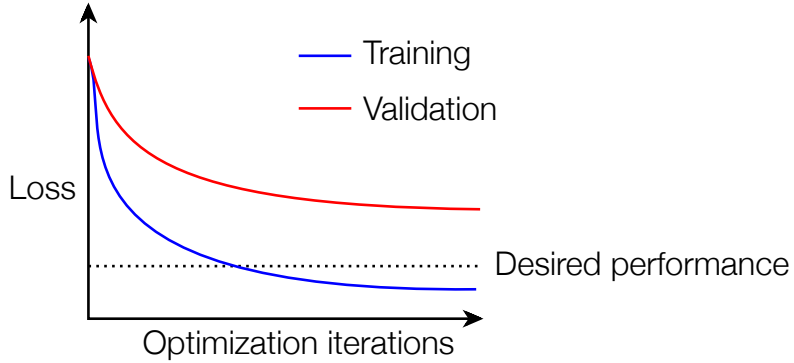
Characterizing optimization performance

It is a much less common problem, but you may want to look at training/testing loss versus algorithm iteration, may look like this:



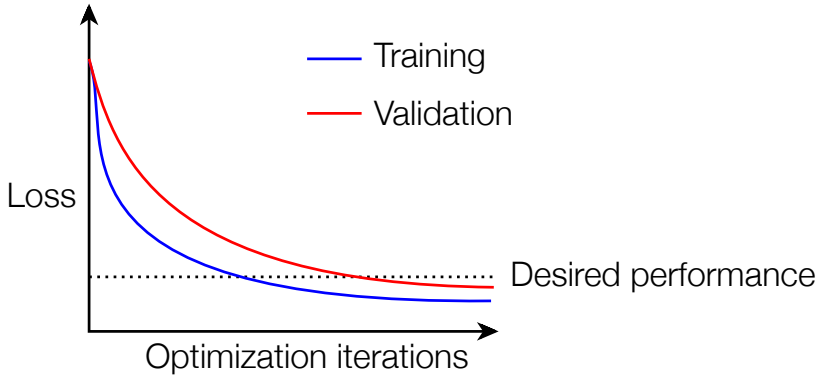
But it probably looks like this:

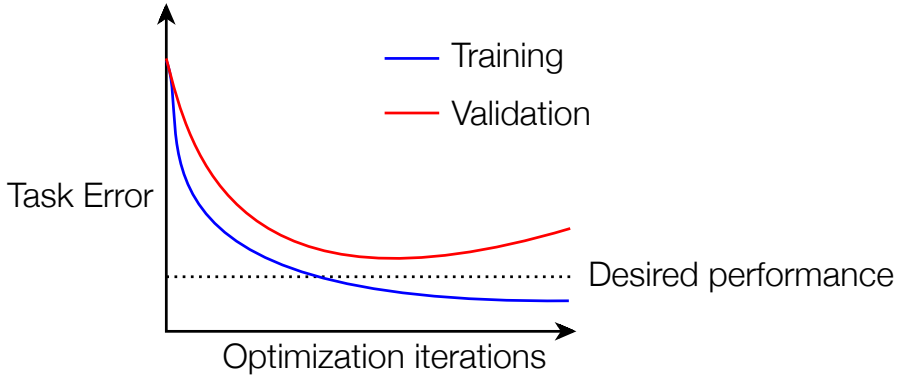




Consider loss vs. task error

Remember that machine learning algorithms try to minimize some loss, which may be different from the task error you actually want to optimize





This is common when dealing e.g. with imbalanced data sets for which cost of different classifications is very different

Get a Ph.D.

Your problem may genuinely be “AI Hard”, you can solve it, but a computer cannot

These are typically the most “interesting” questions from a research standpoint

But, presuming you don’t want to interrupt your career for a life in academics (though I can’t fathom why not), you will want to go back to adjusting the problem itself to be feasible given a more limited set of features

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

One more possibility

You run your machine learning algorithm right off the bat (because I know you won't actually write the diagnostics first), and it works great

Be skeptical, unless it's "obvious" how to solve the problem

Still try to solve the problem manually, and if you can't, be *extremely skeptical*

The reason: it's very easy to accidentally set up a problem that lets the algorithm "cheat", use a feature that is virtually a deterministic one-to-one function of desired output

I have seen companies "fix" their historical data in a way that makes it easy to predict upcoming failure…

Poll: A very bad data science product

(This is based upon, sadly, a true story). You are interviewing with a company that claims to have built a ML-based lie detector test, which can determine whether a person is lying from a video of their conversation. Where would the data science debugging process presumably break down for this problem?

1. Step 1: Determining if your problem is impossible

2. Step 2a: Debugging and correcting impossible problems

3. Step 2b: Debugging machine learning performance

4. Step 1b: Getting impossibly good performance

Summary

	ML predicts well	ML predicts poorly
You predict well	Congrats, but still be skeptical and do a bit of analysis	“Feasible” problem, debug your machine learning algorithm
You predict poorly	Be skeptical, make sure you aren’t cheating	“Impossible” problem, debug the problem