Twitter Thread by Santiago



Santiago
@svpino

I'll blow your mind with a technique you aren't using yet.

Sometimes, you want your system to do exactly the opposite of what your machine learning model thinks you should do.

Let me convince you. ↓

I'm going to start with a nice problem:

Imagine a model that looks at a picture of an electrical transformer and predicts whether it's about to break or not.

Don't worry about how the model does this. We are going to focus on the results instead.

There are 4 possible results for this model:

1. It predicts a bad unit as bad.
2. It predicts a bad unit as good.
3. It predicts a good unit as bad.
4. It predicts a good unit as good.

#2 and #3 are the mistakes the model makes.

Assuming we run 100 units through the model, we can organize the results in a matrix:

• The rows represent the "actual" condition of the transformer.

• The columns represent the "prediction" of the model.

We call this a "Confusion Matrix."

		predicted	
		bad	good
actual	bad	60	3
	good	7	30

This is how we can read this confusion matrix:

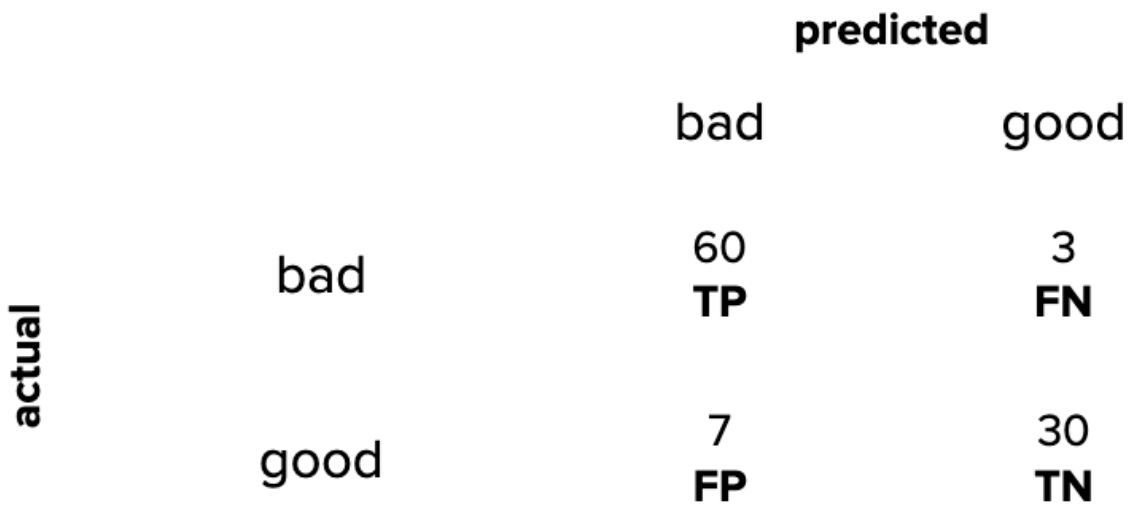
- 60 bad units were predicted as bad.
- 3 bad units were predicted as good.
- 7 good units were predicted as bad.
- 30 good units were predicted as good.

		predicted	
		bad	good
actual	bad	60	3
	good	7	30

We have a name for each one of these values:

1. True positives (TP) = 60
2. False negatives (FN) = 3
3. False positives (FP) = 7
4. False negatives (FN) = 30

In this example, "POSITIVE" corresponds to a bad unit.



Our model made 10 mistakes:

- 7 false positives
- 3 false negatives

At first glance, it may seem that we have a problem with the false positives.

But here is where things start getting interesting.

What happens if we need to send a technician to inspect every transformer that our model thinks is about to break?

Let's say that it takes 2 hours to inspect the transformer, and the technician charges $100/hr.

Every false positive will cost us $200 (2 hours x $100/hr)!

We have 7 false positives.

$200 x 7 = $1,400.

Out of the 100 samples we ran, we'll incur $1,400 in false positives if we follow the model's recommendation.

Let's now take a look at the false negatives.

Imagine that if we miss a bad transformer, the unit breaks, so there will be an outage, and we'll need to scramble to restore service to that area.

The average cost of fixing this mess is $1,000.

We got 3 false negatives.

$1,000 x 3 = $3,000.

Following the model's recommendations, our total cost would be:

• False-positive costs: $1,400
• False-negative costs: $3,000

$1,400 + $3,000 = $4,400.

Now it's time to do some magic and optimize this.

Before we get to the fun part, keep this in mind:

In this example, false negatives are more expensive than false positives.

We want our model to minimize false negatives.

How can we do this?

Here is a really cool approach.

Let's assume that our model assigns a probability to each prediction.

So whenever it says "this unit is bad...", it also returns "...with 65% percent probability."

We can use that!

Let's illustrate this with an example.

Our model returns:

• Prediction: Good.
• Probability: 55%.

Assume we can compute the opposite probability as 1 - 0.55. Therefore:

• Prediction: Bad
• Probability: 45%

Let's combine this with the costs.

We only care about the mistakes because if the model gets it right, the cost is $0.

Before trusting the result, we will compute what's the potential cost if it makes a mistake:

• Model predicts "Good," but the unit is "Bad."
• Model predicts "Bad," but the unit is "Good."

Potential Mistake 1: The model predicts "Good," but the unit is "Bad."

This would be a False Negative.

Probability: 55%
Cost: $1,000

Potential cost of returning "Good": 0.55 * $1,000 = $550.

Potential Mistake 2: The model predicts "Bad," but the unit is "Good."

This would be a False Positive.

Probability: 45%
Cost: $200

Potential cost of returning "Bad": 0.45 * $200 = $90.

Think about this:

• The model predicted that the unit is "Good."

• If we trust it, and we are wrong, our cost will be $550.

• If we do the opposite, and we are wrong, our cost will be $90.

Our best bet, in this case, is to do the opposite of what the model says!

This technique has a name:

"Cost Sensitivity."

Adding a cost-sensitive layer on top of the mistakes of a model is a great way to analyze and optimize your predictions.

Isn't this beautiful?

Every week, I post 2 or 3 threads like this, breaking down machine learning concepts and giving you ideas on applying them in real-life situations.

If you find this helpful, follow me @svpino so we can do this thing together!

Details in the thread:

- I missed “True negatives” when describing the values in the confusion matrix. It should be item #4.

- The percentages (55% and 45%) are swapped when computing the estimated costs.