Statistics and Machine Learning - Cranmer

Ali Al Kadhim

November 14, 2019

1 Cranmer 1/4

For theoretical physicists We have a prediction as an arrow from a theory to prediction or data Theory $\rightarrow$ Data. Whatever theory we have has some parameters θ specifying a particular theory like masses, cosmological constant, etc. Then the data could be generically referred to as X. As experimentalists we collect data, but the data should be relevant in testing a particular theory, so we want to examine several theories and see which one fits best with the theories. One we have the data, we have the opposite direction. We have the data and we want to say something about the parameters, and this $X \rightarrow \theta$ is called inference. But this inversion or inference there could be several different theoretical scenarios, or values of the parameters that are consistent with the data. So What we do to connect these together is a statistical model $P(x|\theta)$.

Given the parameter θ takes on a particular value, what is the probability to have a certain distribution of the data. At the LHC we build a very complicated model for the LHC data that is parameterized in terms of the, for instance, effective field theory coupling, etc. and it is also the model that is used in neural networks. In neural nets, it might not necessarily describe a probability, but maybe some function, etc. The idea is that these X will be treated as random variables, so even if you did the experiment again , the value of x will be random in some way. Most of the time, in particle physics, we look at plots that look like a histogram, for some random variable , like invariant mass, and the y axis it is the number of events,(or counts) per bin. So we want t a statistical model based that works for that, and based on that we want to make some inference. Often times in particle physics, the histogram (distribution) falls of dramatically, for example, the mass of two jets m_{jj} so that it's sometimes on a log scale.

In terms of a discrete distributions, where you talk about a probability mass functions, which refers to when the data are discrete things, then $x \in \{\}$ a discrete set of numbers, then the probability of all the possibilities is 1 $\sum_i p(x_i) = 1$ and if it's continuous $\int p(x)dx = 1$ the units of $p(x)$ in the continuous case is a probability density function, so that the density at some point it's okay for $p(x) > 1$. Instead of a histogram, I could have a smooth distribution... In any

case, we can think of the data as a set of $Data = \{x_i\}_{i=1}^n$. Where the data is a discrete number of events, and for each collision we have a continuous number, like the invariant mass, which is x. We have a number of collisions, which is discrete in nature, and for each collision we have a continuous distribution, like the mass of something, which is what these x are. for any given collision (i=1, i=3,...) we have associated probability density $p(x_1), p(x_2), ..$ then the probability that I got x_1 and x_2, if these collisions are independent from each other, I can multiply these probabilities together, so $\prod_{i=1}^n p(x_i)$ (iid: independent and identically distributed). The continuous things in particle physics $p(x) = \frac{1}{\sigma}\frac{d\sigma}{dx}$ There is a differential cross section, and we normalize it so when we integrate it we get 1.

The cross section, from quantum mechanics is we have an initial state to final state, $\frac{|<i|f>|^2}{<i|i><f|f>} = p_{i\to f}$ This is the thing we calculate from first principles in QM as scattering probability, but we realize this is inconvenient to work with, so we factorize it to a flux component and a cross section. So this is not some statistical overlay in top of our beautiful physics, this is physics we're doing. How do we predict the total number of collisions n? $L\sigma$ where L is time integrated luminosity (of units 1/area) and σ has units of area. In plots we see $L = \int \mathcal{L}dt = 30fb^{-1}$ and often times there is an acceptance factor A did the particles hit my detector, and an efficiency factor ϵ - how often did these what I have found those collisions in the detector. In addition, this could be the cross section for a particular type of process, like the process you're interested in like the Higgs particle, but typically we also have some background events, which we can do the same calculation for and sum them all up, or I can just generically write $n = L\sigma A\epsilon + b$ as the predicted number of events (the number of events is a random variable). Say we do a number counting experiment where we know everything but σ and we want to measure σ. So $\sigma = \frac{n-b}{L\epsilon A}$. This is some type of inversion or inference, because we have the data, and we get some parameter of theory σ. However the background b is some prediction. The other thing is say there is no signal present, then $n = b$ but the number of events is going to fluctuate, could be less than b. Also all these values have uncertainties associated with them, so you want to propagate these uncertainties. So maybe when we measure σ we want to think of it as best guess $\hat{\sigma}$ so $\sigma = \hat{\sigma} \pm \delta\sigma$ This is the goal.

It's very common that when you're trying to estimate a parameter θ, often times people write the parameter with a hat on it, but this is some function of the data, so this is a random variable as well. We need to think of, what is the probabilistic version of $n = L\sigma A\epsilon + b$?

There is some number of tries or trials (N), and there is some number of collisions (n). Then there is a probability for collision p. Then there is a maximum number of collisions, (if every one of them hit) $n = N$. And somewhere in the middle is the expected number of collisions Np. So what's the distribution, or probability of success? Say n are successes which have probability p,

then this is a binomial distribution $\binom{N}{n}\ p^n\ (1-p)^{N-n} = Bin(n|N,p)$. So the Poisson in front is just the probability of success, of finding the Higgs, for large N. If we are thinking about Higgs boson collisions, how many collisions do I get that produce the Higgs n? not that many. N is very huge number, the total number of collisions. So there is a limit where $Np = \mu$ and you take the limit $\lim_{N\to\infty} N(n|N,p) = Pois(n|\mu) = \frac{\mu^n\ e^{-\mu}}{n!}$ a poisson distribution centered at μ. So what we have is $p(data) = Pois(n|\mu)\ \prod_{i=1}^{n} p(x_i)$ Also $< n >= \mu$.
When we have a binned picture, we can have the product of poissons, and another view is to look at the height of these bins turns into μ of that particular bin, over the sum of μ of bins so that it is a normalized histogram $\frac{\mu_b}{\sum \mu_b}$ Then when you add it up it equals one. So that the x axis is x, and $p(x) = \frac{\mu_b}{\mu}$ for each bin. This picture is a probability density function that is piecewise constant. We have Poisson for the total number of events (where b is for each bin, and i is for each event within the bins) $Pois(n = \sum_b n_b|\mu = \sum_b \mu_b) \prod_i p(x_i)$, which is equivalent to a product of poissons.

We can think of it also as a summation of different Poissons for each event. When you add different distributions we have convolutions, and convolutions of poissons is also a poisson. Now when we talk about the expected number of events in a particular bin μ_b that number could be coming from several different background processes.

We can think of the raw data that is attained from the detector as X_{raw}, if we want to look like what this distribution looks like. I want to model $P(X_{raw}|\theta)$ Well these simulation chain is referred to as Monte Carlo. FOr example for the internal piece, MadGraph, we have this cross section, at the phase space of parton level, $\frac{d\sigma(\phi)}{d\phi}$ can be calculated, but the total cross section is integreted numerically through monte carlo integration. It works like this: if we have a complicated distribution $p(x)$ then $\int p(x)f(x)dx \approx \frac{1}{N}\sum_{i=1} Nf(x_i)$ if x_i is distributed as $p(x)$.

We have to think of the process at the detector, we have very complicated process. The first part is the parton-level process. The next part is the parton shower, where a particular gluon that splits into two quarks or something like that, and then you have the hadronization process, where quarks get together to get you individual hadrons. Then you have stable particles that hit the detector, so now you have a charged pion that hits the detector and it ionizes matter through the way, maybe it produces cherenkov light or some nuclear reaction. So you have the lower level partons modeled with madgraph, then hadronization by pythia, and the energy deposits by Geant. Then we apply code that groups the particles back as reconstruction algorithms, that gives estimates of the particles energies, and then maybe something like the masses of these particles. Then if the want to calculate the probability (the hisogram),

we have to calculated

$$p(X_{raw}|\theta) = \int p(x|z_{Geant})p(z_{pshower}|\phi)p(\phi|\theta)dz_{Geant}dz_{pshower}d\phi$$

This gigantic integral is what this simulation chain is doing. This simulation process is symbolically doing this gigantic integral, which gives at the end this raw detector readout. This ends in $p(x)$, what's the chance of getting in this bin (histogram)? The point here is that this integral is totally impossible to do: we can do sample from it using Monte Carlo techniques, but it is intractable. What's happened very recently is that this integral is intractable, so we use these Monte carlo techniques and make histograms which approximate $p(x_{raw}|\theta)$. But it is not the actual thing, because if nothing else, you have a finite number of MC samples, so if you do the whole thing again, you have a different number of MC samples per bin so it fluctuates. And also you approximate whatever is not constant across those bins. This histogram approach has been working well for us, but the other problem is that it works well if x is a functional, but what if I try to describe the probability distribution for many different variables at the same time? For example if x includes 10 different angles and energies. The amount of data you want to fill increases exponentially with the number of bins. The way we've been approaching data with histograms is with very low dimensional data that describe the collisions, but uf we choose just one of these variables we might be losing a lot of information. So with ML part of the idea is we want to use a higher dimensional version of the data, where we can describe the events using all of the particles energies and momenta, getting us closer to the real thing $p(x_{raw}|\theta)$ which may be a fairly high dimensional function. This is Likelihood-free Inference. So the idea is that we still want to do inference on the parameter θ but $p(x_{raw}|\theta)$ when you fix the data and think about the parameter is called the likelihood, but because the likelhood involved this intractable integral, we don't know what this likelihood is.

For the discovery of Higgs, the two primary channels were 2 photons, and 4 leptons. The 4l channel used a histogram from "Monte Carlo" simulation. In the case of Higgs to 3 photons, they did not use a histogram, because it was a smooth function, so they fit a falling exponential or a polynomial. The statistical model using this polynomial is very loosely connected to the standard model. When you do statistical model of the data, you have a lot of freedom of how you do it, but if you see some discrepancy, if you use like a polynomial your claim is much weaker. As a working physicist, by far the most important thing is about building a statistical model of the data. The true scattering process does not look like a simple description of like a falling exponential.

2 Louppe 1/3 - Fundamental of Machine Learning

We will build a deep neural network from the bottom up, and how to implement them. If we see the first picture as one object, we infer the other objects as the same. This is automatic extraction of semantic information, we do it in object recognition, speech processing, language processing, planning, etc.). We want a program which inputs, say images, as array of pixel numbers that would tell us what it is. We can write a program that learns the task of extracting semantic information. Definiting a parametric model with many parameters, and then optimizing all the parameters, by "making it work" on the training data. The agent (neural network) is bad at playing the game, but as it trains more and more, it becomes better and better. You can label also every pixel in the screen and identify regions in the screen that represent categories or classes.

Supervised Learning is the task where we are given data $(x_i, y_i) \sim p(x, y)$ are drawn from a joint probability distribution., for example they could be a model from physics. x_i is input sample, is a vector of real values of size p $\in R^p$ and y is the category (what you're trying to predict) (say 0 or 1). if x is some event, y could also be some property in the event you want to predict. Then we have two main tasks.

Classification: you want to compute $P(Y = y|X = x)$ we are given some input sample x (vector) and we want to predict what is the value of y that is most likely given x. The other task is Regression: We want to determine the expected y given some particular input. $E_{y\sim P(Y=y|X=x)}[Y = y|X = x]$. The goal of supervised learning is starting from x we want to determine y and this conditional probability of y given x $P(Y = y|X = x)$. Regression aims to estimate the function given the data.

We do that using Empirical Risk Minimization. We want to find function $f \in \mathcal{F}$ in some output space. For example could be a family of function of polynomials of some degree. And f maps the input space to the output space $f : x \mapsto y$. We need to quantify the goodness of any function. We need to define a loss function, that takes some output value and some other output value (like the prediction by our function) and produces a real number $l : y \times y \mapsto \mathcal{R}$. So for classification, the loss function we could use takes some value y and input prediction $f(x)$, and we want to quantify if this prediction f is close to y. $l(y, f(x)) = 1_{y\neq f(x)}$ where 1 is the identity function which would be 0 if y=f(x) and 1 if the prediction is incorrect. For regression, we have a function and a common one takes the distance $l(y, f(x)) = (y - f(x))^2$. Now we can define according to this criteria the expected risk $R(f) = E_{x,y,\sim P(x,y)}[l(y, f(x))]$. Our goal is to find some f_* which minimize the expected risk $f_* = argmin_{f\in\mathcal{F}}R(f)$.

But we have is finite number of samples. We can define the expected risk of

our function over finite number of samples (say given a finite set D) $\hat{R}(f, D) = \frac{1}{N}\sum_{(x_i,y_i)\in D} l(y :, f(x_i))$. It is how good the risk function given a dataset. So we find a function that mininmizes this quantity instead. $f_*^D = argmin_f \hat{R}(f, D)$. With a lot of data $\lim_{N\to\infty} f_*^D = f_*$.

2.1 Polynomial regression

We want to find a polynomial that minimizes this empirical risk. Say x is drawn from a uniform $x \sim U[-10, 10]$ and $y \sim N(g(x), \sigma^2)$ where $g(x)$ is a poly of degree 3. So now $\mathcal{F} = 3 = \mathcal{R}^4$ is a poly of degree 3, which can be summarized by 4 numbers. Then the prediction of our model $\hat{y} = f(x; w)$ and is parameterized by w where w is a vector from this family of polynomials R^4, so $\hat{y} = \sum_{d=0}^{3} w_d x^d$ and my goal is to find these w's.

$$w_*^D = argmin_{w\in\mathcal{R}^4} \hat{R}(w, D) = argmin_w \frac{1}{N} \sum_{(x_i,y_i)} l(y_i, f(x_i, w))$$

which for regression is distance $= \frac{1}{N}\sum_i (y_i - f(x_i, w))^2$. Where f is given by our model (a polynomial)

$$w_*^D = \frac{1}{N} \left[\begin{pmatrix} y_i \\ y_N \end{pmatrix} - \begin{pmatrix} x_1^0 & ... & x_1^3 \\ x_N^0 & ... & x_N^3 \end{pmatrix} \begin{pmatrix} w_0 \\ w_N \end{pmatrix} \right]^2$$

This is ordinary least squares. This can be solved such that the optimal value that would minimize this quantity is $w_*^D = (X^T X)^{-1} X^T y$ The X is the training data matrix and this vector w is the one that would minimize the empirical risk.

2.2 Neural Network

The first model of a deep neural network was derived in the 50's and was called the perceptron. Our model $\sigma(x) = 1$ if $x \geq 0$, and $= 0$ otherwise. This is an activation function and the function of the perceptron $f(x, w, b) - \sigma(w^T x + b)$ which gives you a linear combination, and produces the function, so we want to find w and b. To go around this problem we introduce another model called Linear Discriminant Analysis (LDA). So we have $(x, y) \sim P(x, y)$ where again x is in R^n and $y = \{0, 1\}$. And Assume $P(x|y) \sim N(\mu_y, \Sigma)$, so now we want to find μ_y, Σ. So now we want to ask what is the probability that $y = 1$,

$$P(y = 1|x) = \frac{P(x|y = 1)P(y = 1)}{P(x)} = \frac{P(x|y = 1)P(y = 1)}{P(x|y = 0)P(y = 0) + P(x|y = 1)P(y = 1)} = \frac{1}{1 + \frac{P(x|y=0)P(y=0)}{P(x|y=1)P(y=1)}}$$

$\sigma(x) = \frac{1}{1+exp(-x)}$ so that this expression can be written as $\sigma(log\frac{P(x|y=1)}{P(x|y=0)} + log\frac{P(y=1)}{P(y=0)})$ and defining $log\frac{P(y=1)}{P(y=0)}) = a$ we can simplify

$$= \sigma(logP(x|y = 1) - logP(x|y = 0) + a)$$

and $\sigma(logP(x|y=1) \propto \frac{1}{2}(x0\mu)^T\Sigma^{-1}(x-\mu)$ And simplify again

$$= \sigma((\mu_1-\mu_0)^T\Sigma^{-1}x + \frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1) + a)$$

I call $\sigma((\mu_1-\mu_0)^T\Sigma^{-1}x = w^T$ and $\frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0-\mu_1^T\Sigma^{-1}\mu_1) = b$ So $= \sigma(w^Tx+b)$ so my prediction according to this LDA is some activation of some linear combination of the data. So I can estimate parameters that by these estimates w^T and b. So x are some data in 2-D, just some coordinates of points in a plane, and you want to determine whether these coordinates correspond to blue or red dots. So you fit the parameters of this model, so you fit the parameters of this Gaussian and from the training data you have the mean of the first and second classes.

2.3 Logistic Regression

$P(y=1|x) = \sigma(w^Tx+b)$ where σ is the same sigmoid function as above. Here we no longer make the assumption that $P(x|y)$ is normally distributed. We want to find values w, b such that the likelihood of our data is maximum and for each of them we evaluate the likelihood under my model $armax_{w,b}P(D|w,b) = \prod_{(x_i,y_i)\in D} P(y_i|x_i;w,b) =$. We have pairs for which y=1, and for y=0, so

$$= \prod_i \sigma(w^Tx_i+b)^{y_i}(1-\sigma(w^Tx_i+b))^{1-y_i} = argmin_{w,b} = \sum_i [-y_i log\sigma(w^Tx+b) - (1-y_i)log(1-\sigma(w^Tx_i+b))]$$

Because the argmax is the negative of log argmin. This expression is "loss entropy", it is some $l(y_i, \hat{y_i})$ which corresponds to cross entropy $H(p,q) = E_p[-logq]$ which compares two distributions, and in our case, the two distributions we are comparing are $p = y|x_i$ and q is our prediction $q = \hat{y}|x_i$. $= argmin_{w,b}\mathcal{L}(w,b) = \hat{R}((w,b),D)$.

2.4 Gradient Discent

So now we want to minimize our loss using an algorithm called gradient descent. We posit some approximate of this loss $\mathcal{L}(\theta)$. So we posit some model that minimized it in the neighborhood of θ_0,

$$\hat{\mathcal{L}}(\theta_0+\epsilon) = \mathcal{L}(\theta_0) + \epsilon^T\nabla_\theta\mathcal{L}(\theta_0) + \frac{1}{2\gamma}|\epsilon|^2$$

. We say that the gradient is zero at the min $\nabla_\epsilon\hat{\mathcal{L}} = 0 = \nabla_\theta\mathcal{L}(\theta_0) + \frac{1}{\gamma}\epsilon$ The step you should do to minimize this model is $\epsilon = -\gamma\nabla_\theta\mathcal{L}(\theta_0)$ So iteratively we have

$$\theta_{t+1} = \theta_t - \gamma\nabla_\theta\mathcal{L}(\theta_t)$$

3 Cranmer 2/4

Notation, an expectation where the data is drawn from some distribution $E_{x \sim p(x)}[f(x)] = \int p(x)f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ if $x_i \approx p(x_i)$. (last step was MC approx)
You need to find the loss function such that the expected risk can be approximated from MC. $R[f] = E_{x,y\sim p(x,y)}[l(y, f(x))]$ this is a distibution of, say, y being a simulated event being higgs (label), and x is the data of the event. Also we have $f_* = argmin_{f\in\mathcal{F}} R[f]$ (argmin just means the argument (f) is what you minimize). Calculus of variations is very useful for these functions.

$P(A|B)P(B) = P(AandB) = P(B|A)P(A)$ For continuous variables, $P(x, y) = P(x|y)P(y) = P(y|x)P(x)$ so that $P(y) = \int p(x, y)dx)$ When you integrate one of the variables you're marginalizing that variable.
If the loss function is $l(y, f(x)) = (y - f(x))^2$ and we do calculus of variations $\frac{\delta R[f]}{\delta f} = 0$ implies that (for any function that minimizes the square loss) $f_*(x) = E_{y\sim p(y|x)}[y]$.
$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$ let's think of binary classification where $y = 1$ or $y = 0$. Usually a capital letter is referring to it as a random variable, and lower case is it takes this specific value. So $P(y = 1|x) = \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1)+p(x|y=0)p(y=0)}$.
When we search for a new particle, we set it up as a hypothesis test. We have a null hypothesis : a standard model background with no Higgs, and an alternate hypothesis which is the background + the Higgs, and we are trying to choose between them based on the data. The sentence for the discovery was "the probability of the null hypothesis (background only) is small given the data".
$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ so $P(theor|Data) = \frac{P(Data|theor)P(theor)}{P(Data)}$. We talked about $Pois(n|\mu = L\sigma\epsilon A)$ from the theory side I am able to predict cross sections like this, and that predicts and expected number of events, and from these I can make arguments of how many collisions I expect to see. So $Pois(n|\mu = L\sigma\epsilon A) = P(Data|theor)$. So $P(Data|Theor)$ is the likelihood, the part I know how to calculate from theory. And $P(theor)$ is the prior. $P(Data)$ can be thought of as a normalization constant.
When we make Neural net classifier, we just train a classifier with some signal events and some background events. With ML, should be more comfortable with being Bayesian, because prior is part of the training data. You have two hypothesis $H_0(y = 0)$ and $H_1(y = 1)$. Then you make a decision to accept or reject the null hypothesis. Neyman-Pearson Hypothesis Testing is: fix α which is rejecting H_0 and then minimize β which is accepting H_1, equivalently, maximizing $1 - \beta$.
The Neyman-Pearson lemma: The most powerful (simple) hypothesis test is given by a contour of the likelihood ratio.

$$w = \{x : \frac{P(x|H_1)}{P(x|H_0)} > K_\alpha\}$$

where K_α is some contour level.