Harvard University
CS 186

Lecture 3

Lecture Notes by Jeremy Nixon

February 4, 2015

1 Overview

New Room: SC B

The Practical will go up as soon as it's approved by Kaggle.

In CS, there are generally only videos if the courses are taught through the extension school simultaneously.

On the book: Don't conflate the quality of the book with difficulty with the material. This course combines a number of topics in linear algebra and statistics nad so it can be difficult to be comfortable with the material. If you're on the weaker end of the math backgound, you may need to do a lot of work to hang in there. Also, the notation - the new vocabulary - can be difficult the first time you see it. What I hope to give you is not something that makes this easy but something that makes this rewarding.

There's this site by Robert Grosse and Reeve called meta academy that goes over the graph of knowledge required to do understand a concept in machine learning. They link to useful resources on the web, and indentify the relevant chapter and section in Kevin Murphy or Chris Bishop's book.

2 Basis Functions

Basis functions are another intimidating, jargony things. A lot of supervised learning in general is about turning your inpot object into some label with a mapping. In your practical, you're going to have a giant set of molecular candidtates and you're going to have to predict the efficiency of molecules that we don't know the answer to already - an efficenty of a solar cell. But a representation of a molecule doesn't make sense as an input.

So you want to think of these basis functions as features that let us turn raw input data into inputs to a regression. We want to represent our data as a vector of real numbers, and we'll be given some features to work with in the beginning but the point is that the basis functions are one way to go from an abstract molecule object to a vector of some length that we can turn into an input to a machine learning algorithm.

Basis Function = Features

We'll go from some molecular graph and you'll come up with a collection of these ϕ functions that take in these objects and produce a real value representations.

Molecule $-> \phi_j(molecule) -> R^J$

Say you want to get good at sentiment analysis - figure out how positive or negative a person's opinion is based on the text. What are some good features for sentiment analysis? Capitalization, punctuation, etcetera. So we'll take these features out of the document and turn them into a vector of numbers using a Basis function. Each basis function produces a scalar.

This seems like preprocessing, I thought that basis functions were about the functional form. So sometimes the features that you want to find are functions - relatively simple polynomials, or whatever.
Let's imagine that we have one-dimensional data. We have squares and circles on the x axis, and we want to find some partition of this space that will separate the data. But sometimes there isn't one. So what if we add a ϕ function $\phi_1(x) = xand\phi_2(x) = x^2$. By adding a dimension, suddenly it's possible to draw a really simple line that will properly separate the data. Now ther's a simple classifier that separates the two classes. This is a feature - this doesn't feel like thinking hard about sentiment, but it's a feature that we added to the data that reveals some structure in the data that we can exploit.

Question: In real life things are never this simple, so part of the process has to be discovering these ϕs, and in many ways this is the whole game. Someone applying machine learning in industry needs to do *Feature Engineering*.

People Doing Feature Engineering need to

1. Think Real Hard - This is one version of feature engineering
2. Learn Them - (Artificial Neural Networks) Another version is to learn them from the data. This is what deep learning is all about - give theϕs themselves parameter. The early neurtons are trying to adapt to find newϕs and the higher leveles look a lot like regression. You can remind people that neural networks are just adapted basis function regression.

3. Go Infinite (Kernels) - You can go throw the kitchen sink at things - you can try to come up with every possible basis function and use a cleve trick to allow you to impilcitly map your data into an infinite dimensional feature space.

4. Unsupervised Learning - you learn these features in an unsupervised way, those features will be great ϕs for some representation of the data.

In Practice, you probalby want to do a number of these things. Comming up with a good representation for your data is an extremely important part of the problem.
Some people view machine learning as being entirely classification.
The ϕ functions create a vector.

For example, a common non-parametric technique is to use localized bumps to smooth things out after the model gets weighted. Teh width of the bumps can be made narrower or thicker. Adding feature functions that let us map through this augmented representation is the name of the game.
In this case, the basis funciton will map between dimensions, from R^D to R_J.

So this big Φ matrix is NxJ where N is the number of data and J is the number of features. If we drew this big matrix out, the height would be N Nd the width would be J, and the 1,1 element would be $\phi_1(x_1)$, then $\phi_2(x_1)$, and so on, all the way to $\phi_J(x_1)$. The matrix x values also go down to N.

So we can represent the entire matrix as $\phi : X -> R^J$

The Nerual network story is about back propegation, and so when we want to learn the ϕs and they have their own parameter this is more complicated, for now we're going to assume that our ϕs are fixed.

The fact that this is called a basis function does not mean that these have to be orthogonal.

3 Baysean Inference

Using the Gaussian Distribution: Let's contextualize why Baysian Reasoning is so important. Maximum likelihood, and least squares, take data and a model of how the world works and they make a point estimate - your maximum likelihood gets you one number that is your best guess of whoat your estimate should be given the data.

The idea with being Baysian is that we're going to use a distribution to represent our estimate instead of a single point. It is often the case that there is a lot of ambiguitiy about what the data are telling you, and it's useful to capture some of that ambiguity with a distribution.

The way that the maximum likelihood works is that the mean is what is returned to you. This is out point estimate - our best estimate of where the mean is. What's the maximumm likelyhood mean? The sample mean - the mean of the data. Taking the average over the size of your data washes out a lot of the noise.

As we have fewer points, the sample mean gives us a poorer estimate of our mean. We're getting a single number with MLE, and sometimes that single number is near the truth. So mapped point estimates don't allow us to represent the distribution that we get with Baysian reasoning.

Baysian reasoning has been controversial in other fields, but in AI it has been an obvious improvement. With Beysianism we have a prior - an initial guess. Unfortunately, beysian methods do not have the same guarantees that you get with the frequentist point estimate. The main difference is between the probability is something that really exists, or whether they're a disctiption of uncertainty in your head. We're going to focus on Beysian theory here. These methods do not have to follow the normal distribution, but it is often quite likely.

Broadly speaking, we have a mean here, and we have an example of a set of possible parameters that we ca have in a model. We have this model for the data, and n datapoints in some space.

$X_n \in R$

Data: $x-_{n}{}_{n=1}^{N}$

$x_n \ N(\mu, \sigma^2$

$\mu_{MLE} = \frac{1}{N} \sum_{n=1}^{N} x_n$

$=$ argmax$\Pi n = 1^N(x_n|\mu, \sigma^2$

$p(\mu|x_{n}{}_{n=1}^{N} = \frac{p(\mu)p(x_{n}{}_{n=1}^{N}|\mu)}{\int p(\mu)p(x_{n}{}_{n-1}^{N}|\mu)}$

The willingness to treat μ as a random variable is appealing for AI problems because uncertainty is what we deal with all the time.

Bayes rule is really boing because it is just the product and sum rules of probability. It's controversial because of our willingness to commit to μ being a random variable.

One way to think about it is as a data processing system. It's a way to go from some belief about the world - seeing some data, seeing something happen. Ryan encountered this problem where he wanted to build a map of the room to get a robot to not run into things. So he'd run around and get some data, and then the robot would do actions, and he had to decide whether he should trust the new data or the old data more. Keven Murphy said that this was a Baysean reasoning problem.

Here's what beysian reasoning is about. I have some belief about the world. And then I go and I gather osme data, and I see some observation. So if my prior distribution was a set

of a hypotheses for μ, we multiply it point wise by a likelihood function. We re-weight this guy against some likelihood problem. We have some new evidence and need to integrate it into the new distribution, so we integrate it and normalize.

We take the volume of the distribution that is the evidence and the volume of the prior, and we combine them to get our new belief.

$P(A, B) = p(A)p(B|A) = p(B)p(A|B)$