To the Evil Genuises Team

Joe Datz

January 2019

Hey guys, just though I'd recap with what we covered on monday:

Firstly, the address for Pitt's Research Computing is this. Secondly, Here are the resources I brought up monday:

- Coursera - Machine Learning, with Andrew Ng
  - Link
  - This is more of a computer scientists or mathematicians perspective onto machine learning. It will talk about the subject through the lens of calculus/linear algebra than it will probability theory in most of the course. Although it says its an 8 week course, it could be done in 2-3 weeks since you guys are Math/CS majors and don't need nearly as much prep work. The requirements are some linear algebra and calculus 3.
- An Introduction to Statistical Learning, with Applications in R
  - Link
  - This is a statistician's perspective onto machine learning – it will emphasize probability theory and clear interpretations of what the model is telling us, and stay far away from any direct calculus or linear algebra (though it will definitely be happening in the background). The free pdf version can be found on the linked webpage. The only requirements are a familiarity with probability theory.

I think the best thing that can be done for your team right now is to try and tackle these resources before the semester picks up speed. This is to give everyone a baseline of shared understanding about the basics of the project, and make sure everyone is speaking the same language to each other. This is *also* to help you make a decision about whether or not you want to use machine learning in your project, because learning how useful it will be to you now will save you a lot of time in the future.

When I did my project, I both was the only person on my team to do Andrew Ng's course or pick up a textbook, and only finished it during spring break in March. This caused a lot of lag time to producing something substantial for the project, and a lot of communication issues because of the lack of shared knowledge. I'd hate to see such a BS-y obstacle get in the ways of you guys being able to create something by the end of the semester.

Here are some other handy resources:

- Machine Learning for Absolute Beginners, by Oliver Theobald
  - This is NOT a book that will necessarily help you with developing the project. This book will simply outlay the landscape of Machine Learning and introduce you to many techniques for the project. It is very straightforward and could be read in an afternoon.
- Hands-On Machine Learning with Scikit and Tensorflow, by Aurelion Geron
  - This book will not do a great deal to teach you about Machine Learning, but what it will do nicely is introduce you to the different tools that are available in Python to do machine learning projects. Use this as a reference for if you decide to move your project over to Python.
- Python for Data Analysis, by Wes McKinnon
  - The advice above also applies for this book.
- Stanford's CS229
  - Link
  - This webpage is for Stanford's Introductory Machine Learning Class for graduate students and contains notes, homework assignments, and now lecture videos. This is a great supplement to whatever other materials you may desire to read.
- Elements of Statistical Learning
  - This is the graduate-level version of An Introduction to Statistical Learning and is the book I am reading from now. It goes much more deeply into the probability theory aspects of machine learning than ISL will and won't be (as) afraid to talk about Linear Algebra or Calculus. What's also nice about this book is that after the first 4 chapters, every other chapter can be read in whatever order you'd like to. I only recommend this if someone in your group has a very solid foundation in Probability already or is willing to put the time into doing so.

- Pattern Recognition and Machine Learning, by Christopher Bishop
  - This is one of the first major texts on the subject. I will not be reading this book until later this year (I personally struggled with it when I tried to read it for my project and tried something else), so if anyone wants to borrow my copy, they are more than welcome to. You'll need to know your Calc 3 and Linear Algebra backwards, forwards, and sideways to get through this book. This book will give an introduction to probability theory for those who haven't done it before, but I'd still supplement that with another text – it's an entire 1.5 semesters of probability theory crammed into two chapters.
- Convex Optimization, by Stephen Boyd
  - Link
  - This book was recommended to me by the professor running the Machine Learning graduate course in the math department. It will mainly focus on the algorithms that are used to solve ML problems and the theory underpinning them.

Finally, some of those subtleties (and some mistakes I made when I talked) that I was talking about earlier:

1 Supervised Learning.

Yesterday I stated I believe this is a supervised learning problem. Theoretically, this is the problem statement of a supervised machine learning problem:

Given a dataset matrix X, with a target vector (or matrix) Y, can we find a function $H(X) \to Y$ with minimal error?

I should have brought up that I presumed a Y vector which is clearly what you'd want exists in your data. For my project it was straightforward, but that may or may not be the case in yours. You may have to transform, edit, or use other techniques to get a Y vector that you want for outputs.

2 Unsupervised Learning.

Unsupervised learning doesn't have a clear theory statement because it represents a hodgepodge of many different tools with many different end goals. One such end goal is *dimensionality reduction* algorithms, where we seek to take our matrix vector X and reduce its size by eliminating unnecessary variables or compression of the size of the matrix by replacing variables with a new one that approximates multiple others. Given the size of your dataset, I would recommend looking into a few to make sure you can work with data on your personal computers.

3 Support Vector Machines.

On Monday someone suggested that a Support Vector Machine problem could be solved with least squares and I didn't give a proper answer to the question. I think Support Vectors *could* be solved with least-squares technically but usually not practically. Here is the setup for a Support Vector Machine:

We have a dataset of points that we would like to classify as Type 1 or Type -1. One way we could start to do this is by creating a plane that divides the two:

$$\vec{w}^T\vec{x} - \vec{b} = 0$$

And then say that everything above this plane ($\vec{w}^T\vec{x} - \vec{b} > 0$) is of Type 1 and everything below this plane ($\vec{x}^T\vec{x} - \vec{b} < 0$) is of Type -1. If the two data types can be separated this way, there are actually *infinite* ways to draw a plan between the points, so we want to find a way that *best* separates the two types.

The way a Support Vector Machine chooses to define "best" is by having something called a "margin" built into this particular plane. What this amounts mathematically is having to additional parallel planes

$$\vec{w}^T\vec{x} - \vec{b} = 1, \;\; \vec{w}^T\vec{x} - \vec{b} = -1$$

that exist on both sides of the first plane. The two goals of having these planes is that there are no data points in the space between them, and that the gap between them is *as large as possible*. Making a statement like *as large as possible* turns this question into a calculus/optimization problem.

To start framing how we'll solve this problem, we'll find the distance between the two planes. Let x_0 be a point on the plane $\vec{w}^T\vec{x} - \vec{b} = -1$. That is, $\vec{w}^T\vec{x}_0 - \vec{b} = -1$. Note that $\frac{\vec{w}}{||w||}$ is the unit normal vector for this plane. Since these two planes we've decided previously are parallel, there exists some $c \in \mathbb{R}$ for which $x_0 + c\frac{\vec{w}}{||\vec{w}||}$ is translated onto the secondary plane $\vec{w}^T\vec{x} - \vec{b} = 1$. This can be solved as:

$$\vec{w}^T(x_0 + c\frac{\vec{w}}{||w||}) - \vec{b} = 1$$

$$\Leftrightarrow \vec{w}^T x_0 - \vec{b} = 1 - c\frac{\vec{w}^T\vec{w}}{||\vec{w}||}$$

$$\Leftrightarrow -1 = 1 - c||\vec{w}||$$

$$\Leftrightarrow c = \frac{2}{||\vec{w}||}$$

So that we have the distance between the two is $c = \frac{2}{||\vec{w}||}$. To maximize this gap that we want, we want to maximize this equation, and the best way to go about this is to minimize $||\vec{w}||$ with the constraint that there are no data points in the gap between the two planes. The constraints can be written as $y_i(\vec{w}^T x_i - \vec{b}) \geq 1$ where y is each data-point's known class number 1 or -1, and i is each data point.

Having gone through this set up, I think the only case least-squares can be used is if the data can already be linearly separated. If it can, this means the inequality constraints above can be written as equalities instead, and then least squares could be applied. If not, or if the matrix setup is singluar (i.e, a repeated data point), then this does not apply.

It is also possible to create a nonlinear plane as well using something called a *Kernel*. Think of it as a generalization of the dot product from Calc 3. In these kinds of circumstances, the SMO algorithm is employed to solve this problem. For further details, take a look at CS229's information on Support Vector machines.

4 Random Forests.

A Random Forest starts with an individual *Decision Tree*, a flowchart that can represent as a binary tree to make a decision on what a data point in a particular point in space should evaluated as.

To make a decision path, the tree creates a series of horizontal and vertical splits in the data's graph. Everything to the left or right (or top and bottom) of a split is given a category of Type 1 or Type -1 based on what the majority of data is in a split.

To calculate what is an appropriate split, a decision tree uses one of two equations:

$$E = -\sum_{i=1}^{C} p_n \log_2(p_n)$$

$$G = \sum_{i=1}^{c} p_n(1 - p_n)$$

The first is called *Entropy* and the second is called the *Gini Criterion*, and the motivations for either require knowing a subset of Probability called *Information Theory*. The sum is from 1 to however many classes there are, and p_n is a probability calculation for the fraction of each class inside a split. At each split in the tree, the algorithm computes where the best location to split is based on one of these two criterions by minimizing either equation. The algorithm stops

when either all data points are in their own individual box, or some stopping criterion about the number of splits possible is reached.

A *Random Forest* is composed of many *Decision Trees*, and in all cases the more individual trees are used the better the final outcome will be. To compute a prediction for a data point, the data point is given to each decision tree to decide what it should be classified as based on its previous splits.

Had we just ran a Random Forest on the original data matrix X though, we would just end up with the same tree 1000 times because they are all determined by the same criterion for deciding a split. To encourage each decision tree to be different, a Random Forest employs two other techniques on the decision tree:

1. Each Decision Tree is only given a random approximately $\frac{2}{3}$rds sampling of the data. This technique is called the *Bootstrap* and can be applied in many different contexts, but goes hand-in-hand with a Random Forest.

2. Each decision tree at every opportunity to split is told it can only split on a random subset of the variables of the matrix X.

Doing this ensure each tree is very different from one another, and makes sure we can minimize a property called *variance* (see the two resources outlined). There are also other techniques that can be experimented with in the *An Introduction to Statistical Learning* textbook.

5 How to approach solving this problem.

I specifically brought up Random Forests and Support Vector Machines in class Monday because they're both choices which perform with high accuracy when you have a very, very high number of observations and relatively low variables. There also exists a lot of existing software in MATLAB, Python, or R which have these problem solving techniques developed, so you don't have to spend time developing your own. In theory after the 2-3 weeks of doing Coursera or ISL, you could pull up the dataset on Python or R and see how well either technique works. From there, you can make a decision on your next step.

There is a problem with doing this though. If you choose to use machine learning to solve the problem, the way I've described the Supervised Machine Learning problem is building the function

$$H(X) \to Y$$

In your case I think this is an oversimplification of the particular problem you have. If you use machine learning, then you will have to develop

H(Pitcher, Hitter) $\to$ A matrix of the 1st, 2nd, and 3rd pitch and a type of throw.

Or three different models for the 1st, 2nd, and 3rd pitch individually. Or, even regression if you so choose. This is a *much* harder problem. I'm suggesting that you start with either an SVM or a Random Forest if you choose to model as a classification problem so that you can move to a prototype stage before tackling the bigger problem. My personal belief is that you should try a Neural Net as a final version for this type of problem. They're very nuanced and throwing them at you before you have other basic knowledge down wouldn't be helpful, so I will not go over them here.

One of you suggested that you start with a simple model before moving to a more complex model. I think that's a completely appropriate approach and was an oversight in my project. If that's what you'd like to do, *An Introduction to Statistical Learning* is the more appropriate resource to start with. It will go over Linear Regression with many different adaptations before moving to an SVM or Random Forest, because the author takes after the same philosophy.

6 Other notes

1. It would definitely help a lot if a few of you sat down to learn baseball statistics and the context behind why sabermetrics began. A good resource for statistics is fan graphs. A good book for context is *Moneyball* by Michael Lewis. (It's also just an awesome book in general.)
2. Whomever has not learned a lot of statistics or probability theory would benefit a lot by picking a book up and studying it over the course of the semester. Unfortunately the *Coursera* web program will avoid going in depth about the subject.
3. The calculation for RAM requirements is (64 bytes per floating point number)*(rows of a matrix)*(columns of a matrix), in bytes - ask google to finish the conversion from bytes to gigabytes. This of course not including whatever compression techniques are included in the language you're working with.
4. I'm being very pro-ML for this project, but I'd like you guys to keep in mind that this (probably) isn't the only avenue for solving this problem. Keep all your options on a list and open for finding a solution, and find other sources of advice as well.
5. If I'm not mistaken, using the CRC requires training. Keep that in mind when planning out how you guys are going to tackle the problem.

That's it guys. Hope all goes well with the project. If you have any more questions, my contact info is joseph.datz@pitt.edu. I'll send my Pirates email once I recieve one as well.

-Joe Datz