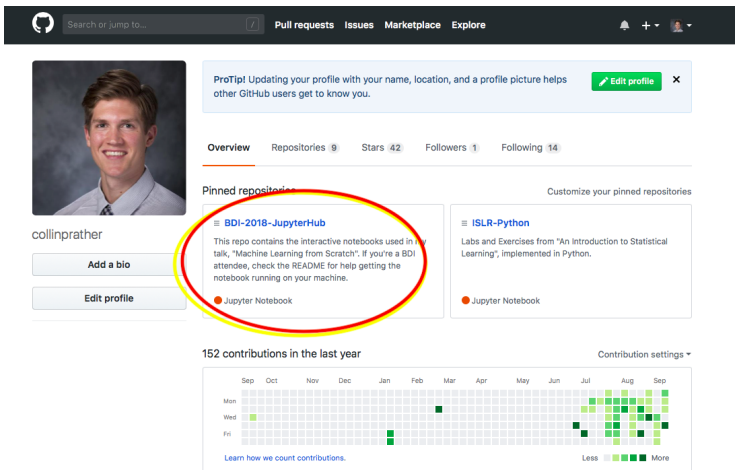
Welcome!

While you're waiting... I've prepared a Jupyter notebook that we will use to explore our data and build a machine learning algorithm from scratch. In order to get the notebook up and running on your computer:

1.) Head to *https://github.com/collinprather*
2.) Click on the "BDI-2018-JupyterHub" repository
3.) Scroll down and follow the step-by-step instructions in the readme.md



Machine Learning From Scratch

└Welcome!

Welcome!

While you're waiting... I've prepared a Jupyter notebook that we will use to explore our data and build a machine learning algorithm from scratch. In order to get the notebook up and running on your computer:

1.) Head to *https://github.com/collinprather*
2.) Click on the "BDI-2018-JupyterHub" repository
3.) Scroll down and follow the step-by-step instructions in the readme.md

Before we get going here, I'd like to direct you to colab.research.google.com to get things set up for the live coding demo that will take place in the second half of our time today. Follow the instructions on my github at...

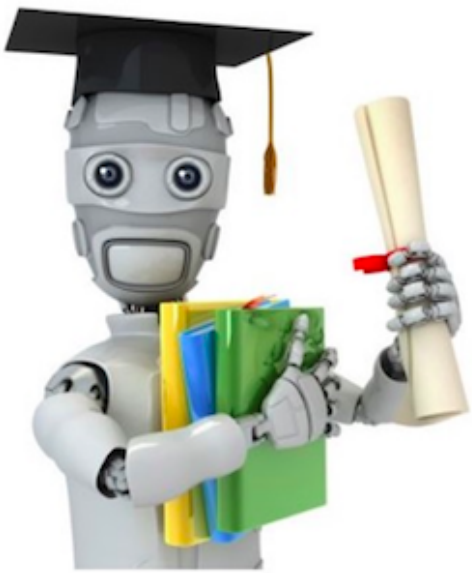
Machine Learning From Scratch

Collin Prather

September 21st, 2018

Good morning, and thank you all for making it out this early! These conference days are long, and to be here bright and early on day 3 is no joke, so I'm glad you're here.
We are going to be tackling machine learning from a ground-up approach today. Can I get a quick show of hands, how many of us here have experience applying machine learning to solving problems?
Before we really dive in, I want to tell you a bit about my talk today, and to do so, we're going to start with two things that help tremendously in successfully applying machine learning... (1) Understanding the math at play: a great way to build that intuitive understanding is to implement a machine learning algorithm from scratch! (2) Knowing how to process/manipulate the data to be ML ready. Now, the skills required for these two steps are admittedly a bit disparate, however, the skills needed to successfully apply ML is inherently interdisciplinary - so I think speaking on these two topics together is rather fitting. And we will, after a brief intro to ml, We will start with talking a lot about the data and do a bit of a case study on the steps in the ML process, and then we're

What is Machine Learning?



Machine Learning

Arthur Samuel:

Machine learning is "Field of study that gives computers the ability to learn without being explicitly programmed".

What is Machine Learning?

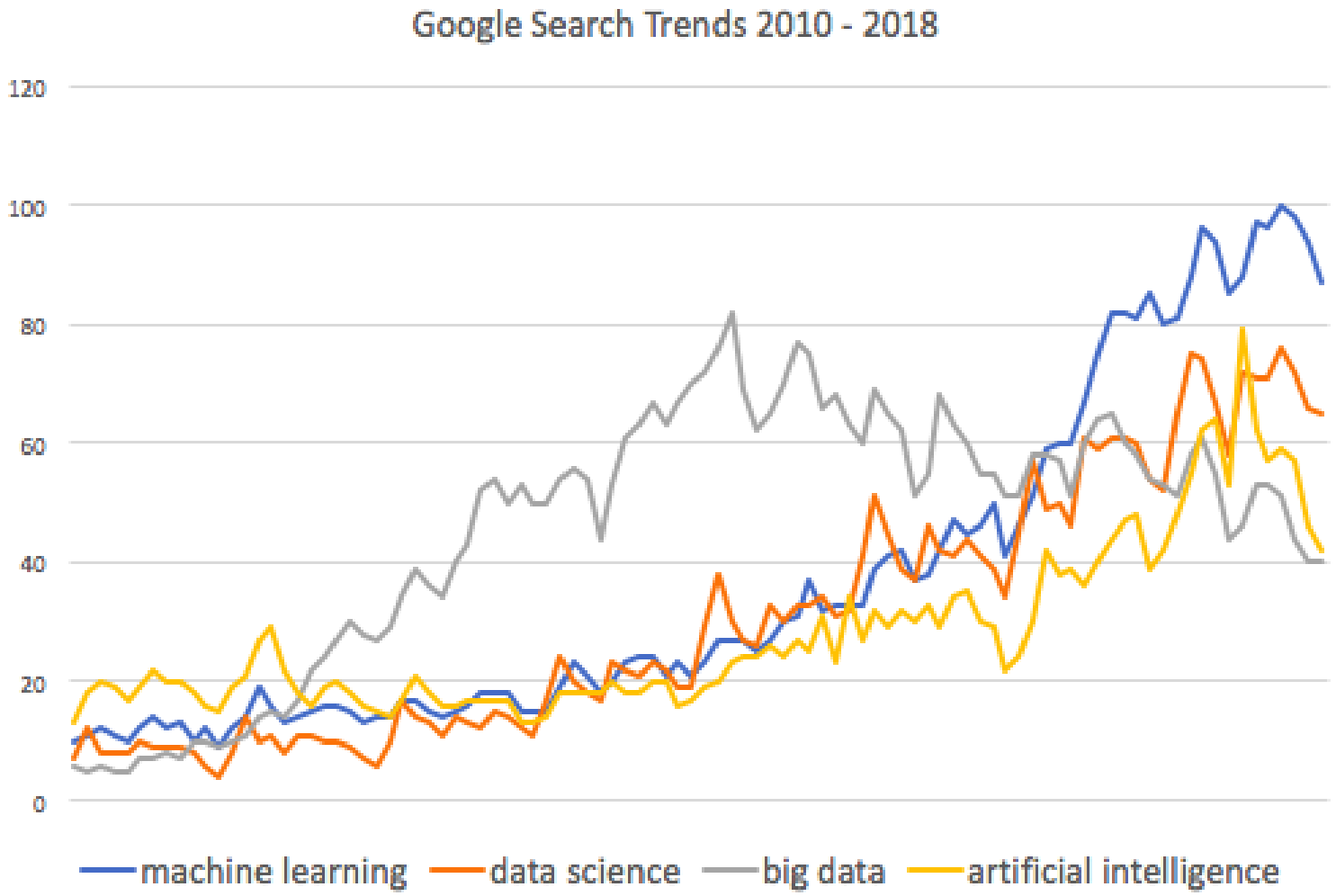
Machine Learning

Arthur Samuel:
Machine learning is "Field of study that gives computers the ability to learn without being explicitly programmed".

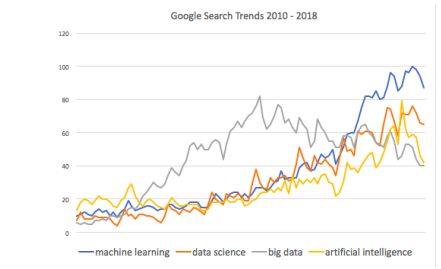
All this talk begs the question, what is machine learning? Even according to the experts, the exact definifion of the field of machine learning is a bit fuzzy, but As early as 1959, Arthur Samuel was famously quoted as defining machine learning as...
ML techniqes can be applied to a wide range of problems in diverse industries. In fact, ML has become ubiquitous in our everyday lives
* Siri/ Amazon Alexa
* Recommendation systems (amazon, netflix)
* Fraud Detection
* Disease diagnosis
* Supply Chain Optimization

The list goes on and on. ML algorithms are used to learn from historical data in order to make predictions about novel data.

According to Google...



According to Google...



This may not be a surprise to you, but the world's google search history reflects a steady increase in interest in machine learning and other related terms like Data Science, Big Data, and Artificial Intelligence.
This graph was pulled pretty simply from trends.google.com, which is actually fascinating, they make it very simple to look up search history trends. It's worth checking out.

The rise of the big data era has given us access to astounding amounts of data. That phenomenon paired with with the exponential growth we've experienced in computational advances, has created the perfect storm for the emergence of the field of machine learning.

Choosing a Model/Representation

	Classification	**Regression**
Supervised	• Logistic Regression • Naive-Bayes • KNN • SVM	• Linear Regression • Decision Trees • Random Forests
Unsupervised	• Apriori • Hidden Markov Model	• PCA • K-means • SVD

	Classification	Regression
Supervised	• Logistic Regression • Naive-Bayes • KNN • SVM	• Linear Regression • Decision Trees • Random Forests
Unsupervised	• Apriori • Hidden Markov Model	• PCA • K-means • SVD

Now, we mentioned that machine learning is used to learn from historical data in order to make predictions about novel data, but as I've alluded to, given the diverse use-cases, not all ML looks alike. There are hundreds of models to choose from, and trade-offs associated with each. The good news is, with each type of machine learning problem comes its own little subset of algorithms – so this narrows down our search considerably.
In our case, we're working on a supervised classification problem, so the upper left hand corner displays some common algorithms for problems like ours. It's very common to test a handful of them and choose the one that performs best on your dataset (or even combine some of them into an ensemble model.)
Remeber, the ultimate objective to choose a model that learns a predictive rule (which comes in the form of a equation) that can be generalized to new observations (both for classification and regression)

Find examples of all 4 different types of ML

Now that we have some motivation for what machine learning is and what it can do, let's move into the steps of the machine learning process.

Identify the Problem

Setp 0 is to identify a problem that can be framed in such a way that it can be solved using machine learning.
Let's say that you work for the city of Grand Rapids, and you've observed an influx in hit and run car crashes in recent years. This is a very concerning thing for the city and you decide that it is time to really start cracking down on the perpetrators. Now the investigative team is obviously interested in gaining any info they can on these fleeing drivers. One characteristic that may heavily inform their investigation is whether or not the driver was drunk. For example, If they knew that the driver at fault was drunk, maybe they could start at local bars near the crash site, trace some steps back. It may not be certain, but it surely be helpful, at least could point them in the right direction. Is it possible to know if a driver was drunk after the fact? Do we have data that help us answer this question?

This is a perliminary step of any machine learning project, asking the questions, can this problem be solved with data? Can we learn from

Get the Data

In our case, we'll head to GRData

Step 1! Obtaining the data you'll need looks very different depending on what domain you're working in. In some instances, it can be fairly simple and straightforward, for example, In a business context, most often it will require querying some sort of internal database. Could also be downloading a csv file. In other instances, it may require a bit more creativity

In our case, we're lucky enough to have access to a meticulously maintained public database on the city of GR: GRData. Scroll through, show them the site and data, etc.

Machine Learning From Scratch

Building a Support Vector Machine from Scratch

Get the Data

Get the Data

In our case, we'll head to GRData

Here we'll first be going through a general example of building an svm from scratch on a toy dataset, then apply some of Python's ml tools to our crash data set to predict whether or not a car crash was caused by a drunk driver!

Here, we will begin by importing the necessary libraries – (audience should just run these). So here is what our plotted data looks like.
The first time you run code you may get this pop-up warning you that the notebook wasn't authored by google... that's because it was authored by me, so you can click right through that
And here is a snapshot of data set. Note that for each data point (each row/each dot), we have an x_1 value, and X_2 value, and a y value (which represents the target variable, which we are trying to predict). In this case, we've rather defined the red data points to be labeled as -1 annd the blue data points to be labeled as 1. It may seem weird to call them x_1 and x_2, but we'll address why we do that in a little bit.

Representation: Support Vector Machine

The first step in building any sort of predictive model is choosing a representation. For our example today, we choose a Support Vector Machine. Big picture - The goal of a SVM is to To find the optimal separating hyperplane which maximizes the margin of the training data. Now, a hyperplane is kind of like a fancy word for a line, or maybe better put, it's a general way to talk about a line. The black line on this graph of our dataset is a hyperplane that separate the two classes (red/blue). Since our data is two dimensional (we have an x, and a y (1, 2)) the hyperplane is a line

The margin of the training data refers to how much is between the hyperplane and the two classes of data on each side.

How do we find the optimal separating hyperplane?:
Now, as we mentioned, the goal of the SVM is to find the optimal separating hyperplane, so how do we do that? It appears that there can be many different separating hyperplanes (often there is infinite). We're going to choose the hyperplane that is as far away as possible from each class of datapoints.
We want to maximize the margin because It generalizes better to classifying unseen observations -¿ (meaning that it makes better predictions)
If we have our usual equation for a line: y=mx+b, then all we need to do is find that optimal *m* and *b* that characterize the optimal hyperplane!

Understanding the Definition of a hyperplane:

As we've seen previously, the separating hyperplane that we keep referring to (in 2-d) is just a line. I'm confident that you're familiar with the way we define lines:

Now, a quick refresher that b is just a constant that represents the y-intercept, m (usually defines the slope) is a coefficient to the variable x (also a constant) and the variable y even has a coefficient as well, in this case it's 1. if we try to expand this equation y=mx+b to more and more dimensions(variables), it's not a real great form to express it in. So instead, we choose to set the whole equation equal to zero. With some algebraic manipulation, we get that:

So, that looks a bit unnatural..so its common convention to represent all coefficients/constants with β, and all variables with X_i, so that we don't run out of variables to use.

In our Crash data, we use 14 variables...

Evaluation

Quick refresher on the dot product:

Now that we have established a representation for our SVM, we'll move on to defining how we'll evaluate its accuracy. Let's start with talking for a minute about the dot product. The dot product has many different names across fields of mathematics: inner product, and linear combination are also common. Regardless of its name, it is a useful operation to use between vectors. To put it simply, if you take the dot product of two vectors, you multiply all their corresponding elements and take the sum. (note that the vectors must have the same dimensions). The output of the dot product is not another vector, but just a scalar value.

Dot product as distance from hyperplane:

Now you'll notice that this dot product indeed looks very similar to the way that we've defined the equation of a hyperplane. It turns out that, when we take the dot product of our weights vector $\overrightarrow{w}$ and a single datapoint $\overrightarrow{x^{(i)}}$, the resulting number that we get can be thought of as the distance from the data point to the hyperplane (how far away it is). And since we're trying to maximize the width of our margin, having an efficient way to calculate how far away each datapoint is from the hyperplane proves to be cruical.

Now, if we're being mathematically rigorous, we may want to frame that a bit differently, but at least for all intensive purposes, we can think of it this way!

The fact that we've defined the equation of a hyperplane to be the dot product of our weights and features equal to zero implies that the hyperplane and the weights vectors are perpendicular (aka orthogonal). And since they are orthogonal, we're able to exploit some linear algebra

The Hinge Loss Function is defined as:

Now, armed with this ability to calculate how far away each data pointis from the hyperplane, we can now use the Hinge Loss Function to quantify how wrong each of our predictions are!

One more important thing to note about the dot product is that it is signed, meaning that it can be either positive or negative, namely, data points below the hyperplane will have a negative distance from the hyperplane, and datapoints above the hyperplane, will have a positive distance from hyperplane.

Talk about how we will predict which class each data point comes from, then use the hinge loss to tell us how wrong we are.

The Hinge Loss Function is defined as (Part 2): Breaking it down

1 minus y times the dot product of the weights vector and the datapoint itself.

Let's break this down piece by piece. First, we'll look at the inputs. $\overrightarrow{w}$ is a vector of weights (coefficients) in the linear equation. $\overrightarrow{x^{(i)}}$ is a vector representing a single datapoint (the i^{th} datapoint), or a single row in our dataset. $y^{(i)}$ is the label associated with the datapoint (which is either 1 or -1). The ouput is a scalar value (a number) representing a penalty for how wrong our prediction was. The greater the penatly, the worse our estimated weights were in classifying the data.

The Hinge Loss Function is defined as (Part 3): little subscript plus sign

One thing I've neglected to mention thus far is the little subscript plus sign at the end of the loss function. That little plus sign denotes that our loss function only looks at positive values, meaning that the output of function is a negative value, it's just going to change it to zero. This intuitively makes sense! If we get a negative penalty for our algorithm... that means that the prediction must be very correct, so instead, we'll just assign a penalty of 0, basically no penalty at all.

The Hinge Loss Function is defined as (Part 4): working it out

Now, if you actually try to work this out with the datapoints, it can be a bit confusing, as there are lots of negatives and the sign of the function is constantly changing, so I've included this little table that works out small exmaples for all 4 cases, when the data point is correctly and incorrectly classified from the negative class, and when the data point was correctly and incorrectly classified from the postive class. For these little examples, I've assumed that the datapoint is 3 units away from the hyperplane. You'll notice if you skip to the bottom that the penalty for both incorrectly classified datapoints is 4, while the penalty for the correctly classified datapoints is 0.

The Hinge Loss Function is defined as (Part 5): essence of ml

This is really worth emphasizing, because it doesn't just apply to SVM's but is the basis of machine learning. Machines learn by associating a quantitative penalty to incorrect predictions. The machine then sets out to minimize the penalty, which ultimately will result in making the maximium amount of correct predictions.

This is the essence of the current state of artificial intelligence. All AI is based on this principle, everything from autonomous vehicles, to amazon's recommendation system.

Hinge Loss with regularization term:

Now, with the hinge loss function we've defined, when we try to minimize the loss function, we will (if possible) find a separating hyperplane that perfectly classifies the data i.e. no misclaassifications. Which sounds great! The only problem with that is that is there are any outliers in our dataset, it can seriously skew the hyperplane. Below we have two nearly identical datasets...the one on the right has one outlier (circle it with mouse) and it has completely changed our hyperplane.

Again, we need to remember that we our goal is to find a hyperplane that will best classify new datapoints...so we don't want it to be affected by outliers.

In order to defend against the outlier, we will introduce a regularization term to the end of our hinge loss function, parameterized by the coefficient λ.

Regularization term as key to bias-variance tradeoff

The regularization term does more than just protect against outliers though. This provides a great opportunity to take a brief detour through the bias-variance tradeoff:

The bias-variance tradeoff is a consideration that permeates all types of ML and is a critical aspect to take into consideration when choosing a model. These dartboards create a great analogy for understanding the bias-variance trade-off. Bias and Variance have some specific definitions in statistics, but in this context, they are words that we use to describe the behavior of a machine learning model.

For example, If I were to say that our model has high variance, then that means that its predictions would vary significantly if we were to train it on a different sample of our dataset.

If I were to say that our model was highly biased, then that would mean that it's predictions are very consistent, regardless of the sample of data that we trained it on. (that does not necessarily mean accurate, as we see in the bottom left corner.)

Optimization

Optimization

Now, we've been talking quite a bit about minimizing the penalty to our algorithm for mis-classifying data points, but it is not immediately obvious how to minimize the penalty.

We're going to use an algorithm called Stochastic Gradient Descent that will iteratively find the weights that minimize the total "loss" or "cost" to our Support Vector Machines, resulting in the very best possible predictions. **Look at this visually in two ways! 1. With stanford web demo, 2. with gradient descent drawing**

Stanford Web App

For one of its course, Stanford has created this awesome web demo that actively animates gradient descent. Here, they have three different classes (red, blue, green), while we only have two (red, blue), yet the principles are exactly the same. Gradient Descent begins with totally random separating boundaries, this means that we begin with random weights, $\overrightarrow{w}$, random coefficients in the equation of the separating hyperplane. (click randomize a few times)

The algorithm then iteratively updates the weights little by little until the separating hyperplane is correctly classifying the data points.(start repeated update with softmax)

So this is how gradient descent works.

Gradient Descent Drawing

I want to show you another angle we can look at this from. Take a look at this drawing. Imagine that this red line is the hinge loss function (preface: it's not. But it is a representative of the general concept of gradient descent), which, remember, outputs how wrong our predictions are, so we want to minimize this function, by finding the weights that correspond to the smallest possible output. All we have to do is move towards the bottom of this function, which is exactly what gradient descent does.

Transition to Math of GD

Pragmatically speaking, the algorithm is going to loop through each observation (row) in our dataset, and check if our current weights would have correctly or incorrectly classified it. If it would have incorrectly classified the data point, then the algorithm updates the weights accordingly.

A natural question to ask then is, how are we going to update the weights? How do we move from the top of the function to the bottom? We will do so by using partial derivatives of the hinge loss function. We do this because:

- The derivative of the hinge loss function gives us the slope of our hinge loss function at our current values for $\overrightarrow{w}$. Given the slope, we want to "move" or update our weights in the direction that the slope is decreasing i.e heading towards the bottom of the function.

Calculating the partial derivatives:

Okay, so let's calculate the partial derivatives. Using a rule from calculus, we can separately calculate the partial derivative of the first and second terms of the hinge loss function (loss term and regularization term) separately.

As we can see here, the partial derivative of the loss term depends on whether the datapoint was correctly or incorrectly classified. (0 if correctly classified, or -y times x if incorrectly classified). The partial derivative of the regularization term is 2 times lambda times w.

Update Rules:

Conclusion Now, Admittedly, I found and explored this dataset for the purpose of presenting here, (look up notes from michael to better explain this?)

But truly, the challenge in ML is not the math (the machine will do it for you), and often it's not the data itself (as aforementioned, it's becoming increasingly more accessible). The challenge is often building a ML model that is stable, has "business" value, and is able to be used in production.

I thought that maybe I can leave you with this challenge: If you knew, as we've shown today, that given data from the Police's U10 reports, you could accurately predict if causes were caused by drunk drivers, how would that be helpful? What could that be used for? How could we apply it/put it into use?

These are the questions that should drive us as data scientists.

 https://www.sisense.com/glossary/data-exploration/

 https://towardsdatascience.com/understanding-feature-engineering-part-2-categorical-data-f54324193e63

 http://scott.fortmann-roe.com/docs/BiasVariance.html