Math in Computer Science

Haley Beavers Chris Daw Evan Smith

May 21, 2019

Contents

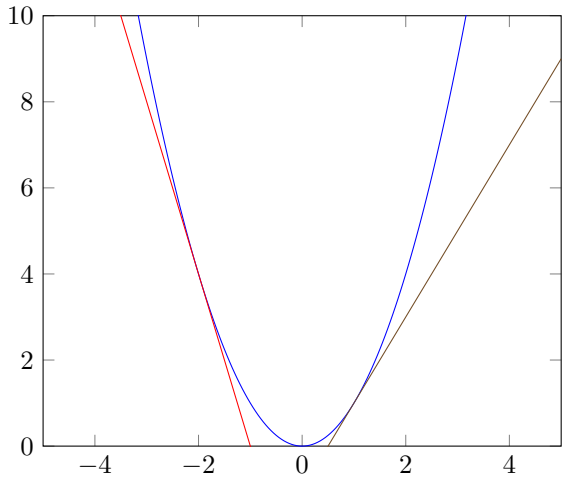
1 Calculus

Calculus should be viewed as a language of change. It allows us to model our world in a language that is not only static and linear, but dynamic and changing. We can then manipulate these models to take advantage of calculus to solve dynamic problems in the real world such as optimization problems and machine learning.

1.1 Basics

If you have not yet taken MATH 124, the basic idea behind derivative calculus is to find the rate of change of the value of a function over small change of the dependent variable, Δx. If we have a linear function like $y = 2x$, for example, the rate of change of the function is constant; at every x value the slope of the function is 2 so any small change Δx will change y by $2 \cdot \Delta x$.

Now consider a non-linear function $y = x^2$, for example. The slope of this function is not constant so how could we find its rate of change? Using the small change Δx will essentially approximate the exact slope around that change, but we want to find the exact, **instantaneous rate of change** at a single point. With derivative calculus, we can find that specific point! We say that as Δx approaches an infinitesimally small value, it becomes the same as looking at just one point on the graph. We call this infinitesimally small change dx. On the graphic below, the blue line is $y = x^2$ while the red and green line show the slope at exactly $x = -2$ and $x = 1$.



At any point, for example $x = 1$, the instantaneous rate of change describes the amount the y value would change if we were to move a very small amount in the x direction, dx, thus the instantaneous rate of change is the slope at that point. We can find this slope with the derivative of the graph $\frac{d}{dx}(x^2) = 2x$ so at $x = 1$, the slope is $2(1) = 2$ which is what we see on the graph in green.

1.2 Gradients

Given any differentiable function $F(x)$, the gradient of that function, $\nabla F(x)$, is a vector pointing in the direction of greatest increase from that point. We will not dive into the nitty-gritty details about how this is calculated, but it consists of computing partial derivatives with respect to each individual dependent variable. The magnitude of the gradient, $|\nabla F(x)|$, gives us the value of the maximal rate of change. Gradients are very useful in optimization as we can use them to locate local minima and maxima of a function.

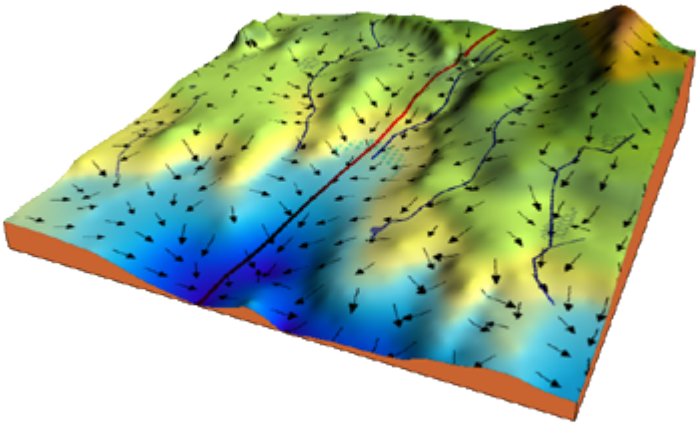
For example, if we are standing on a hill whose shape can be modeled by a differentiable function $F(x)$, then $\nabla F(x)$ will point whichever direction will increase your elevation the most with a small step in that direction.

1.3 Uses

1.3.1 Gradient Descent

Gradient descent is an optimization algorithm used heavily in machine learning for minimizing the loss of the model. As its name suggests, gradient descent moves in the direction opposite of the gradient, $-\nabla F(x)$, such that it moves in the direction of steepest descent on the function $F(x)$. In the gradient descent algorithm, we iteratively take steps in the direction of steepest descent until we find a local minimum.

Imagine a 3d terrain with rolling hills and a lake. We start in a random location that happens to land us halfway up the hill and our goal is to find the bottom of the lake. How can we achieve our goal? Gradient descent! We calculate the negative gradient of our current location and take a small step in that direction. At this new location we can recalculate our negative gradient and once again take a small step in that direction. This process repeats until we find a local minimum. We will not necessarily find the bottom of the lake using this method, but we will definitely find a local optima that is better than where we started. There are many variants of gradient descent which attempt to skip past local optima in search for a better alternative.



On this image (source), there are arrows at various points along the terrain showing the direction opposite the gradient at each point (the steepest downhill direction). Image

1.3.2 Gradient Descent for Machine Learning

In machine learning we have parameters in our models that learn to find a local optimum based on a cost function. The cost function defines how well our model performs its specified task so we want to minimize the cost to find the model which performs the best. This can be achieved through gradient descent where for every iteration of the algorithm we take a step in the direction opposite $\nabla F(x)$ where $F(x)$ is our cost function and x are our model's parameters.

We call the size of the step we take the "learning rate". With a larger learning rate, we will find the minimum faster (with fewer iterations), but we risk overshooting an optimal minimum. Conversely, with very small learning rates, we will take longer to converge to a minimum and we may not break out of a local minimum which is not optimal.

2 Statistics

2.1 Basics

Essentially, statistics is the study of making inferences from data. This data can be in many forms, from numbers to sounds to images. There are so many questions that can be answered confidently using statistical analysis.

2.2 N-Gram Language Modelling

The applications of statistics are broad. One such application is language modeling. A language model is a probability distribution over sequences of words

that can be used for tasks like sentence generation, machine translation, and text classification. Today we're going to learn about n-gram language modelling. The basic idea is that by analyzing the frequency at which particular words follow each other we can make feasible inferences about which words make sense together. So a language model is essentially made by going through a text and counting the frequency at which a word a follows another word b. Let's examine the following set of sentences with the symbols $< s >$ and $< /s >$ denoting the start and end of a sentence.

$$< s > I \textit{ live in Washington} < /s >$$
$$< s > I \textit{ am a student} < /s >$$
$$< s > I \textit{ go to school in Bellingham} < /s >$$

Using these sample sentences, let's calculate the most likely candidate to follow the word *in* using a bi-gram language model. Because this is a small sample, let's look only at all unique words that follow the word *in*. In a purely programmatic situation, we would iterate through the entire list of unique words in our text sample called a vocabulary and would consider all of those words to be suitable candidates for a word following *in* even if they may never have come after *in*. But again, because this is a small sample, let's just look at the words we do know follow *in* which are $Bellingham$ and $Washington$.

$$\begin{aligned} P(\textbf{Bellingham}|\textbf{in}) &= \frac{\textit{number of times } \textbf{\textit{in Bellingham}} \textit{ appears in text}}{\textit{number of times } \textbf{\textit{in}} \textit{ appears in text}} \\ &= \frac{1}{2} \\ &= 0.5 \end{aligned} \tag{1}$$

$$\begin{aligned} P(\textbf{Washington}|\textbf{in}) &= \frac{\textit{number of times } \textbf{\textit{in Washington}} \textit{ appears in text}}{\textit{number of times } \textbf{\textit{in}} \textit{ appears in text}} \\ &= \frac{1}{2} \\ &= 0.5 \end{aligned} \tag{2}$$

If we were to use a unigram model, we would just predict that the most likely word is just the word that appears in the given text sample the most $P(w_i)$. If we were using a bigram model we would examine the probability that a given word follows the previous word. If we were using a trigram model we would examine the probability that a given word follows the previous two words etc. Below is the basic probability equation for unigrams, bigrams, and trigrams respectively.

$$\begin{aligned} P(w_i|w_0...w_{i-1}) &\approx P(w_i) \\ &\approx P(w_i|w_{i-1}) \\ &\approx P(w_i|w_{i-1}w_{i-2}) \end{aligned} \tag{3}$$

In an n-gram language model we can look at how many times an word a appears after a particular sequence of "n" grams. As you go up in grams, or words, you'll get more accuracy for a bit but then be left with a very rigid model when looking at the probability that a word a appears after a specific 10 word sequence.

3 Linear Regression

3.1 What is Linear Regression?

Linear Regression is the modeling of a linear relationship between some response Scalar, and sets of one or more predictor variables. When we only have one predictor variable it is called a Simple Linear Regression. If we are using multiple variables to predict some response it is called Multiple Linear Regression.

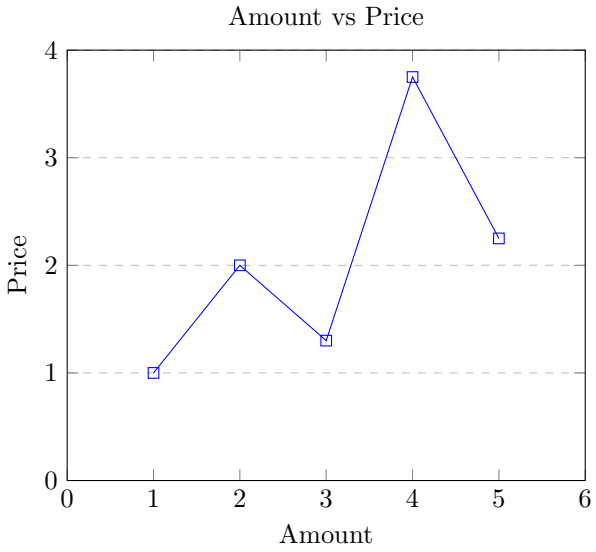
3.2 What is Linear Regression Used For?

Linear Algebra is a great tool used to model linear relationships. It's main uses are prediction and Estimation. Linear Regression is a grate predictive model to start with as it is one of the easier ones to understand, and very powerful, as many things have linear relationships.

3.3 Example

I believe the best way to learn about linear regression is through examples.

Price(in Dollar's)	Amount
1	1
2	2
3	1.5
4	3.7
5	2.25



The Goal is to find the line that is "the most correct" to do this we need to minimize the squared error. Most Lines are written in the form $y = mx + b$ but when we're doing linear regression we write it as $Y_i' = bX_i + A$, where Y' is the prediction variable which is in this case price and b is the slope of the line with the least squared error and A is the intercept. To find the least squared error we must first define error.To do so we first define some imaginary line that we don't know called the True Regression Line. Which we define as $Y = bX + A$. then the error is how off we were from that true line or $e = Y_i - Y_i$ and we can define Q as the sum of these errors squared:

$$Q = \sum_{i=1}^{n} (Y_i - Y_i')^2$$

which now since we know $Y_i' = bX + A$ we can replace Y_i' giving us:

$$Q = \sum_{i=1}^{n} (Y_i - bX + A)^2$$

Now all that's left to do is find the b and A which give us the smallest Q. (take the derivative with respect to b and A, set to 0, and solve for A and b). With just a little bit of calculus, and a lot of number manipulation we can get.

$$A = \overline{\mathbb{Y}} - b\overline{\mathbb{X}}$$

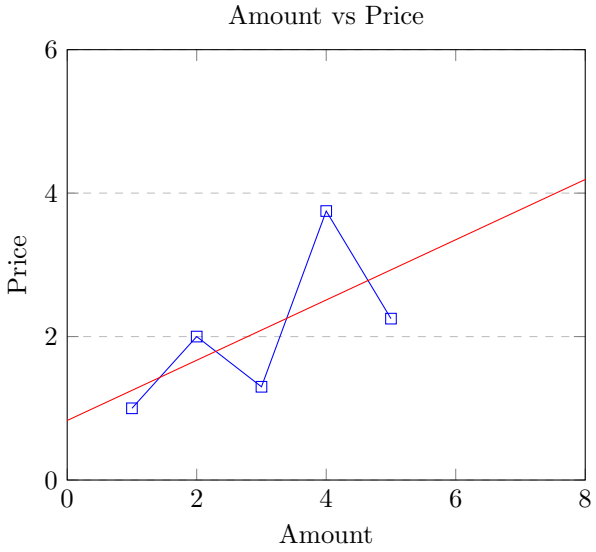
$$b = \frac{\sum_{i=1}^{n}(x_i - \overline{\mathbb{X}})(Y_i - \overline{\mathbb{Y}})}{\sum_{i=1}^{n}(x_i - \overline{\mathbb{X}})^2} \tag{4}$$

Using the formulas above we can find that:

$$A = .83$$

$$b = .420$$

$$Y' = .420X + .83$$



What does this have to do with computer science? I believe Linear Regression is a great first step towards machine learning. Many things are surprisingly Linear, and don't need any of the more fancy machine learning algorithms. Another advantage is getting use to modeling the world in the language of math, and finding things that have correlation with each other.

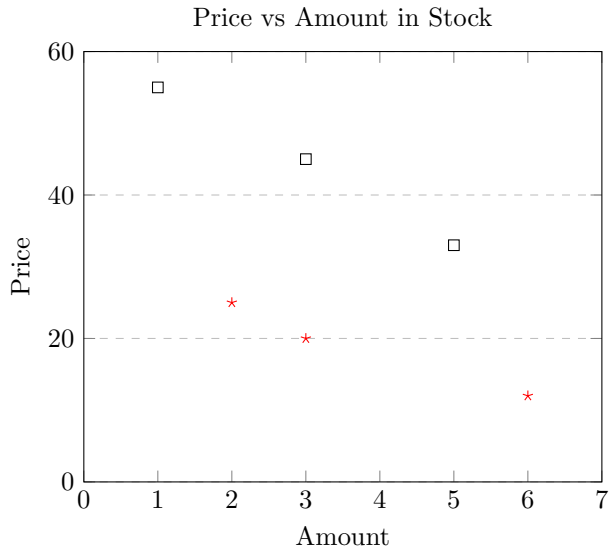
4 Support-Vector-Machine or SVM

4.1 What is an SVM

A support vector machine in very simple terms is just another fancy line(Or Hyper Plane in higher dimensions). The purpose of this line is two divide two sets of data into that are plotted as N-Dimensional points into distinct classifications. First you give it training data to help it draw the line(or Hyper Plane), and then with that line you can classify new things based on which side of the line they fall on(which half space they are in). It is a simple but very effective model at classifying data.

4.2 Example

Price(in Dollar's)	Amount In Stock	Model
45	3	1
55	1	1
33	5	1
12	6	0
20	3	0
25	2	0



Lets add some context. say you have some machine learning hw due the next day and you were supposed to collect a lot of data for your fancy classifier. what you were supposed to do was drive from store to store and record the price amount in stock and model of some product and then train a model on it. but you're super lazy and only went to 6 stores. This is where a support vector machine really shines. The main place to use them is exactly when you need to classify or regress small amounts of data.

Where do we start? well we need to draw a line that divides these 2 sets of data the most. But there are way to many lines we could pick(infinitely many). Lets define what it means to say divides the 2 sets of data the most. To do this we need some definitions.

Input Vectors: the things we put in like (45,3) or (25,2)

Support Vectors: Input vectors that just touch the boundary of the margin

Margin: This distance from the decision surface to the closest data point.

decision surface: the hyper plane or in this case line that splits the data.

So like in most things we need to maximize something. in this case the distance between the margins. In other words we want this line to be as far away from any data point as we can get it.

Now lets throw math at this.

A decision hyperplane can be defined by an intercept b and a decision hyperplane(or in this case line) normal vector $\vec{w}$ which is perpendicular to the hyperplane.

If you know its intercept and the vector its perpendicular too there is only one hyper plane that satisfies that constraint

$\vec{w}$ can also be called the Weight Vector.

remember: In geometry, a normal is an object such as a line or vector that is perpendicular to a given object.

now there are a lot of hyper planes or lines to choose from if b isn't specified so b needs to be specified.

We can define b as...

$$W^T * X = -b$$

Now we have our training set which can be defined as an input vector and a label we don't use 1 and use for labels but 1 and -1 so from now on model 1 has the label 1 and model 0 has the label -1. In more math terms we can say

$$D = X_i, y_i$$

where each X_i has a corresponding y_i

Our descision function is now

$$f(x) = Sign(W^T X + b)$$

Now we have to look at our data and finish up the model. lets define r to be the distance from a data point to our decision boundary. The shortest distance will always be parallel to the hyper plane and therefore parallel to X and can be re written using the unit vector $r * X|X|$ We label the point closest to our decision boundary(hyper plane) as X^1 Now

$$X^1 = X - yr * X|X|$$

Where all y does is flip the sign to get to either side of the decision boundary. Since X^1 is right on the decision boundary it also follows that

$$W^T * X + b = 0$$

Which when we plug in the last equation gives

$$W^T(X - yrW/|W|) + b = 0$$

which when solved for r gives

$$r = y(W^T * X + b)/|W|$$

The points closest to the decision boundary(the hyper plane) are the support vectors

The geometric margin of the classifier: the maximum width of the band that can be drawn separating the support vectors of the two classes. twice the minimum value over data points for r

Now using some math we don't have time to go over we can scale our whole problem so that the geometric margin p can be defined as $p = |W|/2$

now all we need to do is find the minimum p over our whole data set where $y_i(W^T X + b) >= 1$

This is a standard optimization problem now and can be solved using some quadratic programming

and with some high level math magic we get

$$f(X) = sign(\sum a_i y_i X_i^T X + b)$$

Back to the actual Data we had... modified a bit

Price(in Dollar's)	Amount In Stock	Model
45	3	1
55	1	1
33	5	1
12	6	-1
20	3	-1
25	2	-1

This is not really meant to be done by hand so a lot of this is estimation...

the first step is (we change the minimize function for convinience) minimize $1/2|W|^2$ given $y_i(W^T X + b) >= 1$

This happens when this constraint is satisfied with equality by the two support vectors

The real first step is to find the smallest distance between 2 of the points from the different classes. which in this case is the points(2,25)(5,33). this will give us our weight vector $W = (3, 8)$ and with some messing around we get $-\frac{-8}{3}x + 40$


Price vs Amount in Stock
60
40
20
0
Price
0
1
2
3
4
5
6
7
Amount