# A quick(ish) reference for some concepts whose intuitions/simple methods of understanding sometimes escape me.

David Khachatrian

Ongoing.

# Contents


**1 Information Theory** **2**
- 1.1 Entropy and Cross-Entropy . . . 2
- 1.2 KL divergence and mutual information. . . . 3
- 1.3 Entropy of variables vs. entropy of distributions . . . 4

**2 Statistics** **6**
- 2.1 Bayes' Theorem . . . 6

**3 Constrained Optimization Problems** **8**
- 3.1 Equality constraints only: The Method of Lagrange Multipliers/The Lagrangian . . . 8
  - 3.1.1 Intuition with just one constraint . . . 8
  - 3.1.2 Doin' me some gradients . . . 9
  - 3.1.3 The Lagrangian: a packaged function . . . 10
  - 3.1.4 Extension to more than one constraint . . . 11
  - 3.1.5 Explaining the convenience, and a more generalizable intuition. . . . 12
- 3.2 Constraints with Inequalities: the Karush-Kuhn-Tucker (KKT) Conditions . 14
  - 3.2.1 Not nearly as scary as they make it out to be. . . . 14
  - 3.2.2 If it's not that bad, why did you talk so much? . . . 17

**4 Machine learning methods.** **18**
- 4.1 Boosting. . . . 18
  - 4.1.1 An analogy to Taylor expansions. . . . 18
  - 4.1.2 Meta-algorithm vs. algorithm . . . 19
  - 4.1.3 Where the analogy is left wanting. . . . 19
  - 4.1.4 Gradient boosting. . . . 20

**5 Drafts and WIPs** **22**

**Index** **23**

# 1. Information Theory

## 1.1 Entropy and Cross-Entropy

There are a number of ways to think about/get to the Shannon entropy ($H$). One simple way is to say what we want it to mean qualitatively and impose some desired characteristics to determine its mathematical formulation. In this case, it may make more sense to start with cross-entropy first.

Say you're sending me messages from an alphabet of symbols $A$. Some are more likely than others, and we'll call the probability distribution associated with the actual generator of symbols $p$. I'm over here on the other side thinking like I'm a pair of smartypants that has figured out how likely you are to send each symbol to me. We'll call my expected distribution (which may or may not be the correct distribution) $q$.

Now we want to measure how surprised I am by any given message you send me -- let's call it $\tau$.[1] We would imagine the following properties would be useful for $\tau$ to have:

1. The more surprised I am, the larger $\tau$ gets (otherwise, it wouldn't exactly be doing a good job measuring my surprise)
2. If symbols are independently generated, we can construct the total surprise of my message by adding the total surprise of each symbol in the message, i.e., $\tau_M = \sum_{i=1}^{m} \tau_{M[i]}$ where $M \in A^m$ is a string of symbols, $m$ is the cardinality of $M$, and $M[i]$ denotes the $i$'th element of $M$.

Using these two criteria, a reasonable mapping is

$$\tau_{a\sim q} = \log\left(\frac{1}{q(a)}\right), a \in A$$

where $q(a)$ is the probability I think you'll send the symbol $a$.

I mean, that's great and all, but that works for individual instances of strings or symbols. How much should I *expect* to be surprised by any given symbol? Well, that'd depend on how often you *actually* send me a symbol, alongside how often I expect to receive that particular symbol.

Hmm, this smells of expectation! And indeed, that's all we do -- take the expectation over $A$ (using the *actual* probability of each symbol occurring, i.e., using $p$) of my surprise per symbol:

$$E_{a\sim p}[\tau_{a\sim q}] = \sum_{a\in A} p_a \log\left(\frac{1}{q(a)}\right) = H(p,q)$$

[1] Rhymes with "wow". Alas, not standard notation for a measure of surprise.

We denote this metric $H(p, q)$ (where $p$ and $q$ are the actual and presumed distributions, respectively) and call it the **cross-entropy** (because that sounds pretty cyberpunk to me. I like to think they throw in "cross" because it measures the surprise caused by "crossing" the distributions $p$ and $q$ together.)

Now, we'd imagine that I'd be the least surprised if my presumed distribution of symbols were in fact the actual distribution, i.e. when $p = q$. In fact, this is true!

$$H(p, p) = H(p) = \sum_{a \in A} p_a \log\left(\frac{1}{p_a}\right)$$

is called the **entropy** (or **Shannon entropy**) of the probability distribution and written $H$.[2] Usually, the *log*s written above are base-2. This permits a way of thinking of the value of the Shannon entropy: if I'm only allowed to ask the same series of questions to you to figure out which symbol you want to send me and I know the actual probability distribution $p$ of symbols, how many questions should I expect to ask (i.e. mean/average) before I figure out the answer? The cross-entropy is the same thing, expect I don't necessarily know the actual probability distribution $p$ of symbols, I just think I do (and I think it's $q$) and base my series of questions based on $q$.

## 1.2 KL divergence and mutual information.

Now, hopefully that makes it clear that $p \neq q \implies H(p, q) > H(p)$ — if I don't know the actual distribution, I'm not going to be able to answer the most optimal series of questions. This suggestions to us a notion of "distance" (i.e. a metric) between the probability distributions $p$ and $q$:

$$KL(p \mid\mid q) = H(p, q) - H(p)$$

This metric is called the **Kullback–Leibler divergence** (because names) or the **KL divergence** (because initialisms), or seemingly most rarely but probably most clearly the **relative entropy**. Out loud, you'd say $KL(p \mid\mid q)$ is "the [blah] of p with respect to q". The closer the KL divergence is to 0, the closer $q$ is to being $p$, and $KL(p \mid\mid q) = 0 \implies p = q$ at every point in the domain of $q$ (which is the same as the domain of $p$). (From the above formula, it should be clear that the KL divergence is not symmetric. The first argument is the "correct" distribution and we're measuring how suboptimal the second argument/distribution is at replicating the first one.)

This makes describing the **mutual information between two random variables $X$ and $Y$** in terms of a KL divergence fairly intuitive. Just running off the name, if X and Y were independent, you'd expect no mutual information between them — observing one variable wouldn't tell you anything about the other variable, you don't gain a lot of information about one from the other. In such a case, we know something about their probability distributions, namely that they're independent, i.e., $P(X, Y) = P(X)P(Y)$, which implies

[2] You know, it'd arguably make more sense for the ***S**hannon* entropy to be denoted $S$, which would also be a happy notational coincidence with the symbol used for entropy in most other fields. Instead, we have a notational collision with *enthalpy*. Ah well, such is the arbitrariness of a symbol's meaning. (I suppose it's fitting.)

$KL\left(P(X,Y) \mid\mid P(X)P(Y)\right) = 0$. And in fact, one way of writing the mutual information between random variables X and Y is exactly

$$I(X;Y) = KL\left(P(X,Y) \mid\mid P(X)P(Y)\right)$$

which I think is pretty neat.

- DK (4/24/18)

## 1.3 Entropy of variables vs. entropy of distributions

Now, there is another way of approaching mutual information by defining a conditional entropy between two random variables (itself requiring a defintion of joint entropy in order to put the equation in a "plug n' chug" form). This raises some questions, the first and most pressing of which being "*Variable?* We've been talking about distributions this whole time!"[3] , and the answer to which is "Yes, variable." Since we've been focusing on distributions this whole time, the switch to a random variable $X$ is actually fairly straightforward – we use the distribution from which $X$ is drawn.

More explicitly, say $X$ is drawn from a probability distribution $p$. Then the **entropy of** $X$ is

$$H(X) = \sum_{x \in X} p(x) \log\left(\frac{1}{p(x)}\right)$$

This may be reminiscent of our $\tau_{a \sim q}$ from earlier, except that now we're using the actual distribution of $X$ (which is $p$), so it'd be $\tau_{x \sim p}$.

We can sort of think of it like this: the information entropy only really makes sense for probability distributions. So, if we want to figure out how hard it is to encode a random variable $X$, we kind of "pull out" the probability distribution that $X$ is drawn from and use that in our formula.

Now, unfortunately, the notation gets muddy when we allow ourselves to shove in these random variables as arguments. For example, what does $H(X,Y)$ (X and Y being random variables) mean? H is provided two arguments – is it the cross-entropy between the underlying distributions of X and Y? Nope, it's the **joint entropy** of X and Y, i.e.,

$$H(X,Y) = H(p)$$

where $p$ is the joint distribution of the random variables $X$ and $Y$. Our main defense against confusing the two formulae is that random variables are (normally – hopefully!) denoted by capital letters while distributions are usually denoted by lowercase letters.

You may wonder "what's the point?" Well, this begins to allow information theory analogues for intuitions on random variables gleaned from statistics. The main missing piece

[3] More an outcry than anything, but still appropriate.

at this point is the **conditional entropy** of a random variable Y with respect to X. We'd expect that we could relate the joint entropy and conditional entropy of random variables with one another, like how we can do so with the joint and conditional probability distributions of the variables, especially since we've defined the entropy of a random variable in terms of its probability distribution. And since:

1. we applied a logarithm to the probability distribution (and took the expected value) when defining the entropy of a random variable; and
2. the relationship between conditional and joint probability depends on multiplication:

$$P(Y|X) \times P(X) = P(X,Y)$$

we'd want the relationship between conditional and joint entropy to hold via addition:

$$H(Y|X) + H(X) = H(X,Y)$$

In fact, that's exactly the case! You can derive the formula for conditional entropy based on the above relationship. And just to circle back to information gain, we'd expect that we might gain some information about a random variable X when we observe the random variable Y, depending on how the joint distribution P(X,Y) compares with P(X). Earlier, we defined the mutual information in terms of a KL divergence:

$$I(X;Y) = KL\left(P(X,Y) \;||\; P(X)P(Y)\right)$$

but we can also capture the quantity of "how much does knowing Y (on average) help us figure out X (and vice-versa)?" using the language of entropies of random variables:

$$I(X;Y) = H(X) - H(X|Y)$$

This is the expected amount of information gained by knowing the state of the random variable Y. Specific values of Y may give more information about X – may lead to a much tighter conditional distribution – than others. Then, you can look at it for specific cases, i.e., the **information gain** about X from observing a specific state y for Y, IG(X,Y = y), which is related to the mutual information via $E_{y\sim Y}\left[IG(X,Y)\right] = I(X;Y)$). So if we observe that Y took on the value y, the information gain for X would be

$$IG(X, Y = y) = KL\left(P(X,Y) \;||\; P(X|Y = y)\right)$$

Writing the above in terms of information entropy would be

$$IG(X, Y = y) = H(X) - H(X|Y = y)$$

where $H(X|Y = y)$ is an expectation over the conditional distribution $P(X|Y = y)$:

$$H(X|Y = y) = \sum_{x \in X} p(x|y) \times \log\left(\frac{1}{p(x|y)}\right)$$

The main takeaway is that *defining the entropy of a random variable as it is above allows for the migrations of intuitions about random variables from statistics and probability.* A worthy cause![4]

- DK, 5/14/18

[4]The notational confusion is still unfortunate though.

# 2. Statistics

## 2.1 Bayes' Theorem

I can never seem to remember Bayes' Theorem directly, as they write it out in textbooks. It makes so much more sense to me to think about it from the relationships between conditional and joint probabilites/distributions, and one of the common tricks to make Bayes' Theorem useful in practice also comes to me far more easily when explicitly thinking about events as being sampled from a *sample space* of possibilities/outcomes.

Consider two possible events A and B. Let's keep in mind that A is just one possible outcome out of a set of possibilities, as is B; we'll say $\alpha$ is the set of possibilities from which $A$ was drawn and $\beta$ is the set of possibilities from which $B$ was drawn, i.e. $A \sim \alpha$ and $B \sim \beta$ (this will be good to remember later). Now:

$$\text{Pr[A and B both occur]} := P(A, B)$$

Assuming individual events happen separately, there are two ways for both A and B to occur:

1. A happens first, then B happens.
2. B happens first, then A happens.

("Duh", I know.)
Keeping in mind that the first event might affect the probability of the second event occurring (i.e. remembering that conditional probabilities exist), we can write:

$$P(A, B) = P(A) \times P(B \mid A) = P(B) \times P(A \mid B)$$

And then it's simple to write out Bayes' Theorem as it's often written (we'll write it perhaps a bit more evocatively):

$$P(B) \times P(A \mid B) = P(A) \times P(B \mid A)$$

$$\begin{aligned} P(A \mid B) &= \frac{P(A) \times P(B \mid A)}{P(B)} \\ &= P(A) \times \frac{P(B \mid A)}{P(B)} \end{aligned}$$

Using the Bayesian interpretation: At first we thought the probability that $A$ occurs is $P(A)$. After we saw that $B$ happened, we re-evaluate the probability that $A$ occurs with a scaling factor $\frac{P(B|A)}{P(B)}$.

Also worth knowing the fancy terminology:

1. the ***a priori* probability** or just the **prior** is what we thought would be the probability that a random variable takes on a certain value before we observed anything. So the *a priori* probability (or just prior) for the event $A$ would be $P(A)$. If we consider $A$ to be a random variable instead of an event, we're guessing the distribution of $A$ and so $P(A)$ would be an ***a priori* distribution** (or again, just the prior).
2. the ***a posteriori* probability** or just the **posterior** is what we think the probability that a random variable takes on a certain value is after observing something. In this case, the *a posteriori* probability (or just posterior) of the event $A$ after observing $B$ is $P(A \mid B)$. If we consider $A$ to be a random variable instead of an event, we're guessing the distribution of $A$ after observing a random variable/event B and so $P(A \mid B)$ would be an ***a posteriori* distribution**, (or again, just the posterior).

Now, in cases of inference, we normally have some data on the likelihood of one of the conditionals -- let's say $P(B \mid A)$ to avoid flipping our equation around again -- via empirical counts. So we've already guessed some prior $P(A)$, and we're trying to improve it by calculating the posterior $P(A \mid B)$. But what if we don't know $P(B)$? Is our guessing and data collection all for naught!? Thankfully, not so! The answer lies right under our noses -- or in this case, in our previous calculations.

Consider $P(A, B)$ again. What would we get if we added $P(A, B)$ over all possible values of A? (Remember we said that $A \sim \alpha$, so A could have been some other event within the set $\alpha$.) That's basically just saying that we don't care what value $A$ is, so we end up with $P(B)$![1] And conveniently, we'd already have a way to estimate these values:

$$\begin{aligned} P(B) &= \sum_{A \in \alpha} P(A, B) \\ &= \sum_{A \in \alpha} P(A) \times P(B \mid A) \end{aligned}$$

Our summand is the same as the values we've estimated either by guessing ($P(A)$) or from our data ($P(B \mid A)$)! With that, we can rewrite our earlier equation as

$$P(A \mid B) = \frac{P(A) \times P(B \mid A)}{\sum_{A \in \alpha} P(A) \times P(B \mid A)}$$

With that, we can crunch the numbers and perform Bayesian inference like a champ or have a machine do it for us like a prudent delegator. Though for our everyday activities, we often don't have the luxury of having someone/something checking our heuristics. So it's always worth trying to keep in mind that oftentimes, many different factors that you may not know or take into consideration can culminate in observations that surprise you -- there's a good reason you aren't told to constantly get yourself tested for a medical condition if you don't believe to be at risk!

- DK (4/24/18)

[1] '!' used to denote excitement, not factorialization.

# 3. Constrained Optimization Problems

Motivated to do this when I was reading this paper and realized I forgot how we get to/use the KKT conditions (which is implied in Eq. 2 in the paper).

## 3.1 Equality constraints only: The Method of Lagrange Multipliers/The Langrangian

NB: Khan Academy's videos on the subject really make clear one potential intuition for one constraint.

### 3.1.1 Intuition with just one constraint

Say you want to optimize (maximize or minimize) a smooth function $f(\vec{x})$ subject to the constraint $g(\vec{x}) = c$ (let's say $\vec{x} \in R^n$ -- and we may just write $x$ instead of $\vec{x}$). (It's worth remembering that the constraints themselves can also be expressed as functions -- they just happen to be set to specific constants.) So our goal is to find the best point(s), $\vec{x}^*$. For the purposes of explanation, we'll say our goal is maximization, but it all applies for minimization too.

A way to build up the intuition is to consider the contour lines of $f(\vec{x}) = m$ for particular values of m. Our goal then is to maximize $m$ while having $\vec{x}$ satisfy $g(\vec{x}) = c$. If it didn't fulfill this second requirement, then we'd just be ignoring the constraint and solving an unconstrained optimization problem -- in which case, we'd just set the gradient equal to zero and solve (say, what a useful thought -- let's put that in our back pocket for later...).

Let's call the constraint contour line (which we aren't allowed to change and is set to some constant $c$) $G$ and the function contour line (which we can change by varying $m$) $F(m)$. If we think about it for a bit, we'll see that *solving our problem is analogous to choosing the largest value of m so that $F(m)$ "touches" $G$ in the fewest number of places while still actually "touching" $G$.* Consider the alternatives:

1. *$F(m)$ does not touch $G$ at all:* Well, that means that when we look at the set of points that comprise the contour line $f(x, y) = m$ (i.e., the **level set** of $f$ corresponding to a value of $m$), *none* of those points lie on $g(\vec{x}) = c$. So we ignored the constraint again -- oops.
2. *$F(m)$ touches $G$ too many times:* This implies that $F(m)$ cuts across $G$ (if it didn't, then where did the "extra" touches come from?) And in that case, why not increase $m$ a bit more? We're assuming $f$ is relatively well-behaved, so if you nudge $m$ up a bit to $m^+$, the contour line $F(m^+)$ will be fairly close to $F(m)$ -- and so, still cross $G$ somewhere.

(By the way, I've been using the tortured phrase "minimal number of times but not zero like c'mon don't be cheeky" because there can be more than one location on the constraint curve that have the same maximal value for $f$. Consider maximizing $f(x, y) = x^2y^2$ subject to $x^2 + y^2 = 1$. The symmetry leads to four points that satisfy the criteria -- bump up $f$'s output higher and you're off the circle, bump it down and you're cutting across the circle (and also not maximizing $f$).)

Now, if $F(m)$ and $G$ just barely touch but do not cross, then their instantaneous "slopes" at their touching point must be in the same orientation and the contour lines are moving in the exact same "direction" -- if they weren't, then the touching point is a crossing point. So how do we capture this notion?

### 3.1.2 Doin' me some gradients

The (nonzero) gradient of a function is always perpendicular to its contour lines. (Seems like a deep statement, but with a bit of thought, you can see that it just comes from the definition/intuition of contour lines (on which the value is constant) and gradients (which points toward the direction that increases the function output the largest -- with no portion of the "step" being wasted on movement that would keep the value constant).)

So, we can convert out "contour lines $F(m)$ and $G$ are in the same direction" directly to "the gradients of $f$ and $g$ are in the same direction", i.e.

$$\nabla f = \lambda \nabla g$$

where $\nabla$ is the usual gradient operator and $\lambda \neq 0$, the scalar proportionality constant (it's the *direction* that matters, not the magnitude), is called the **Lagrange multiplier** (dude got a lot of things named after him).

It's actually worth looking at this a bit more. We originally framed our goal by trying to get $F(m)$ and $G$ to touch as little as possible. But we can also frame it in terms of $\nabla f$ and its relation to the constraint functions:

$$\nabla f = \lambda \nabla g \quad \Longleftarrow \quad \nabla f \text{ is perpendicular to the contour } G$$

It makes sense that the right-hand side would be the case -- if $\nabla f$ *did* have some part of it along $G$, then we're wasting that part! Why wouldn't we go along $G$ a bit more? We'd still be meeting our constraint, and since we stepped partially along the direction of steepest ascent, we'd have increased $f$ while we were at it! It'll be worth remembering this observation a little down the line, so keep it in your pocket for later (or some other handy container, if you are doomed to the fate of clothing without functional storage capabilities).

Anyway, that's great and all, but we only have $n$ equations (each of the $n$ elements of the vector formulation above) and now we have $(n + 1)$ unknowns (all the coordinates for $x$, plus $\lambda$). Well, there's a reason the above relation used a(n) $\Longleftarrow$. On the left-hand side, we've only captured that the gradients have to be in the same direction -- we haven't added our constraint! (The right-hand side encapsulates both, since we have $G$ as the contour corresponding to the specific constraint $g(x) = c$.) So our full set of $(n + 1)$ equations with $(n + 1)$ unknowns is

$$\nabla f = \lambda \nabla g$$
$$g(x) = c$$

Now go to town! Worth remembering that all of this provides *necessary* but **not** *sufficient* conditions for optimality. Sufficient conditions would involve, for example, proving the that Hessian matrix of $f$ is negative semidefinite when trying to maximize $f$ (analogous to the second-derivative test in the single-dimensional case), and even if you manage that you're only guaranteed local maximality. Sounds like a lot of qualifications, but we've actually narrowed the search space a great deal with these conditions, so it's not as terrible as it sounds.

### 3.1.3 The Lagrangian: a packaged function

The above system of equations works great for people, but people have also spent so much time and energy to make computers solve math problems for us! Most of these programs are particular good and finding the zeros of a function (without any fancy constraints). So how could we repackage the (n+1) equations above into one function we can find into a zero-finder?

Well, let's rewrite the above equations first:

$$\nabla f - \lambda \nabla g = 0$$
$$g(x) - c = 0$$

Alright, now what? Well, if we were to write something like

$$L(x) = f(x) - g(x)$$

we'd be *almost* there, because if we took the gradient of $L$ and set it equal to zero, we get the "direction" constraint back. But at the moment, we're making it so that the magnitudes of the two gradients have to be the same too (which doesn't have to be true) *and* we forgot to incorporate our constraint again!

Well, why don't we reintroduce $\lambda$ as a variable in such a way that it handles the proportionality problem *and* have it so that $L_\lambda = g(x) - c$ (so that we reincorporate our constraint into the function)? Might sound tricky, but in fact we can modify $L$ to satisfy these requirements fairly simply:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda \left(g(x) - c\right)$$

Now that it's achieved its final form (thankfully didn't take ten episodes of powering up), we change $L$ to $\mathcal{L}$ and call it the **Lagrangian** of $f(x)$ (because dude needs more things named after him -- and you know, he *did* revolutionize the study of classical mechanics with this formulation).

Worth noting is that if we define the Lagrangian as

$$\mathcal{L}^+(x, \lambda^+) = f(x) + \lambda^+ \left(g(x) - c\right)$$

we still get the same answer to our optimization problem -- the only difference is that compared with the $\lambda$ we get from the $\mathcal{L}$ formulation, $\lambda^+ = -\lambda$.

A neat consequence of the formulation of $\mathcal{L}$ is that we can consider $\lambda$ as a measure of how much we could improve $f(x)$ by incrementing the value of $c$ (which we've been considering a constant) by a differential amount. While it may seem to be "clear" just by taking $\mathcal{L}_c = \lambda$, it's a bit more subtle than that, since $\mathcal{L}(x, \lambda; c)$ was formulated with $c$ as a constant. The proof for this observation involves:

- forming a new function, $\mathcal{L}^*(x^*(c), \lambda^*(c), c) = \mathcal{L}^*(c)$, a single-variable function that parameterizes the input coordinates of the answer(s) to the optimization problem (and also the Lagrange multiplier) with respect to $c$;
- doing the multivariable chain rule;
- thinking a bit to notice that a lot of things equal zero to get that $\frac{d\mathcal{L}^*}{dc} = \lambda$;
- and, having remembered that we're interested specifically about the points on $\mathcal{L}$ that optimize $f$ (which is exactly what $\mathcal{L}^*(c)$ captures), realizing that the above result implies that first statement of the paragraph before this bulleted list.

What a mouthful.

### 3.1.4 Extension to more than one constraint

Would be kind of a shame if we did all of this just to solve problems with just one constraint. But thankfully, the extension is fairly simple to describe!

Say we want to optimize $f$ subject to $k$ constraints $g_1 = c_1, g_2 = c_2, \cdots, g_k = c_k$. Now, in all but the most trivial of cases, it would be impossible to have the gradients of all of these different functions in the same direction. Intuitively (-ish, and assuming you feel comfortable-ish with concepts in linear algebra), if they can't all be in the same direction, you'd think that the "next best thing" would be that the gradient of $f$ is in the same direction as some linear combination of the gradients of $g_1, g_2, \cdots, g_k$, i.e. that

$$\nabla f = \sum_{i=1}^{k} \lambda_i \nabla g_i$$

That statement above is in fact a condition that is met in the answer to our optimization problem! (How convenient.)

Now we have $n$ equations but $n + k$ unknowns. We once again fix that by actually incorporating our constraints:

$$\nabla f = \sum_{i=1}^{k} \lambda_i \nabla g_i$$
$$g_1 = c_1$$
$$\cdots$$
$$g_k = c_k$$

We can once again package everything together as a Lagrangian by having that $\mathcal{L}_{\lambda_i} = g_i - c_i$:

$$\begin{aligned}\mathcal{L}(x, \lambda_1, \cdots, \lambda_k) &= f - (\lambda_1(g_1 - c_1) + \cdots + \lambda_k(g_k - c_k)) \\ &= f - \sum_{i=1}^{k} \lambda_i(g_i - c_i)\end{aligned}$$

And Bob's your uncle.

### 3.1.5 Explaining the convenience, and a more generalizable intuition.

Earlier we just kind of accepted the convenience of our guess, but it's worth figuring out why it works. Remember that observation you kept in your pocket (or other handy container)?

$$\nabla f = \lambda \nabla g \quad \Longleftarrow \quad \nabla f \text{ is perpendicular to the contour } G$$

If $\nabla f$ had any part of it along $G$, then we could step along $G$ and further increase $f$. This sounds extensible to more than one constraint! And in fact, the "convenient" result captures this for the contour line created by the intersection of all the constraints:

$$\nabla f = \sum_{i=1}^{k} \lambda_i \nabla g_i \iff \nabla f \text{is perpendicular to the intersection of all constraint contours } \bigcap_i G_i$$

Here, $\bigcap_i G_i$ serves as the one contour line on which we can move along that satisfies all the constraints. (Presumably such continuous arcs exist -- otherwise, there are only discontinuities (i.e. discrete points), and so each point in the set would have to be checked individually.)

Now, since we're dealing with the same construct (a contour along which we can move that satisfies the constraints), the same reasoning applies pretty much verbatim – ***if*** $\nabla f$ ***weren't perfectly perpendicular to*** $\bigcap_i G_i$***, we'd be wasting the component of the gradient going along*** $\bigcap_i G_i$***,*** $\nabla f^{\parallel}$***. So we'd just step along the contour and improve our result.***[1] So that explains why the right-hand side makes sense. But how does that imply the left-hand side? Specifically, how do we get the linear combination part, $\sum_{i=1}^{k} \lambda_i \nabla g_i$?

Well, since we're all on contour lines at the same time, $\bigcap_i G_i$ is necessarily perpendicular to all the gradients $\nabla g_i$. Then any linear combination $\sum_{i=1}^{k} \lambda_i \nabla g_i$ is perpendicular to $\bigcap_i G_i$. In fact, the gradients form a *basis* for the space perpendicular to $\bigcap_i G_i$, because of the symmetric property of the perpendicularity relation. More curtly, call the span of the gradients $S$. By construction, $\bigcap_i G_i \perp S$, and by symmetry, $S \perp \bigcap_i G_i$. We want $\nabla f$ to be perpendicular to $\bigcap_i G_i$ (as we've said before). Well then, that means $\nabla f \in S$, which implies that it $\nabla f$ can be written as a linear combination of $S$'s basis vectors, i.e.,

[1] Excessive bolding, italicizing, and a gratuitous footnote used to emphasize the fact that this is the mathematically rigorous "intuition" to have/remember.

$\nabla f = \sum_{i=1}^{k} \lambda_i \nabla g_i$. Bam! Stick that beautiful $Q.E.D.$ square in the corner, we are done! ~~Would do it myself were I writing this in *LaTeX* and not Markdown.~~ Well, since we've migrated to LaTeX, I think we owe ourselves a box!

■

Totally worth it.

- DK, 4/28/18

## 3.2 Constraints with Inequalities: the Karush-Kuhn-Tucker (KKT) Conditions

### 3.2.1 Not nearly as scary as they make it out to be.

For all the pomp and circumstance around this, with the caravan of names in the name itself and the esoteric terms used in its description like "complementary slackness" and "dual feasibility", the Karush-Kuhn-Trucker (KKT) conditions aren't nearly as hard to follow as one would expect if the method of Lagrange multipliers for multiple constraints makes sense/is comfortable.

First, we pose the optimization problem in "standard form" (which mainly just saves us from lugging around extra constants like we did with $c$ in the Lagrange multipliers explanation):

Optimize $f(x)$ subject to $g_i(x) \leq 0$, $h_j(x) = 0$, with $i \in \{1, \cdots, k\}$ and $j \in \{1, \cdots, l\}$ (so $k$ inequality constraints and $l$ equality constraints). Below, we'll assume "optimize" = "maximize". We'll point out where changes will occur if you're minimizing instead.

#### Primal feasibility

This time, before we do anything else, we're going to stick down the original constraints so we don't forget they exist:

$$g_i(x) \leq 0 \; \forall \; i, \; h_j(x) = 0 \; \forall \; j$$

This is called the **primal feasibility** condition because it's a condition for the feasibility of the original, main, "primal" problem.

#### The dual formulation

We refer to the original problem as the "primal" problem to contrast it with the "dual" problem. That sounds all fancy, but we've made a dual problem before when we formed the Lagrangian for our equality-constraints-only version. That is, the **dual problem** is simply a reframed but equivalent form of the primal problem. We did it before by solving a function that had all our constraints wrapped in one clean package (the Lagrangian form of the problem). And hey, that was both a neat *and* a useful idea, and those don't come around all that often, so let's use it until it goes out of style.

Worth noting is that we want to make our dual problem mirror the primal problem exactly (in *optimal value* as well as optimal location), i.e. form a strong duality, i.e. have no duality gap. The following provide *necessary* conditions, but not *sufficient* conditions for a strong duality.

So what would be the dual problem? We can do the same thing we did for the Lagrangian -- make a new function with some extra variables whose partial derivatives yield the constraints of our problem. Let's try it:
We define a function

$$L(x, \mu_1, \cdots, \mu_k, \lambda_1, \cdots, \lambda_l) = f(x) + \sum_i \mu_i g_i(x) + \sum_j \lambda_j h_j(x)$$

Maximize $L$ (with no external constraints; i.e., find the locations where $\nabla L = \vec{0}$). (We'll come back to minimization later.)

Alright, well that's certainly something. Now we can recover the constraints by noting that $L_{\mu_i} = g_i$ and $L_{\lambda_j} = h_j$. But there are some things that are still funky with the inequality constraints here, so let's work on those.

(By the way, there are some signs we'd have to flip if we were doing a minimization instead. We'll discuss that later on.)

**Dual feasibility**

For one thing, our dual problem won't mimic our primal problem of optimizing $f$ at all if we let any $\mu_i$ be less than zero. If we did, then we'd easily "win" the optimization game by choosing some $x$ such that $g_i(x) < 0$ (which is still satisfies our primal feasibility conditions), then setting the corresponding $\mu_i$ to arbitrariliy large negative numbers -- who cares about $f(x)$ anyway!? Oh wait, we do. So we should probably make sure that we can't break our problem:

$$\mu_i \geq 0$$

This is called the **dual feasibility** condition because, well, otherwise our dual problem isn't all that useful.

**Complementary slackness and stationarity**

For one thing, we can notice that there are two possibilities for the value $g_i$ takes at an optimal point:

1. $g_i(x^*) < 0$: That means the inequality constraint is not actually stopping the function from getting to a "better" location where $g_i > 0$ (our optimal-point finder didn't hit a wall – there's an open neighborhood around $x^*$ that still satisfies the constraint), so the constraint isn't being restrictive at all. So we don't have to worry about it!

2. $g_i(x^*) = 0$: The inequality constraint *is* potentially stopping the function from reaching a more optimal point, so the constraint is actually affecting the outcome. So we should be sure to be othorgonal to the contour in our final answer.

These observations inform the following two constraints, **complementary slackness** and **stationarity**:

$$\begin{aligned} \mu_i g_i = 0 \; \forall \; i \qquad & \text{complementary slackness} \\ \nabla f = \sum_i \mu_i \nabla g_i + \sum_j \lambda_j \nabla h_j \qquad & \text{stationarity} \end{aligned}$$

Let's once again consider our two cases:

1. $g_i(x^*) < 0$: This constraint is inactive and so is not part of the contour lines we use to find the set of points on which $f$ must lie. Recall that we form this set by evaluating the intersection of all the active constraints $C$; that the span of the gradients of the active constraint functions form a space $S$ orthogonal to $C$; and that since $\nabla f$ must be orthogonal to $C$ in order to be a potential extremum, $\nabla f \in S$ and can be written as a linear combination of the gradients of the active constraints. (Boy, another mouthful.) Well, $g_i$ is not an active constraint, and so $\nabla g_i$ is not part of the basis for $S$. So we want its contribution in the stationary condition to be $\vec{0}$. We can do this by setting the corresponding $\mu_i := 0$. Not-so-coincidentally, this necessary consequence leads to compliance with the complementary slackness condition as well.

2. $g_i(x^*) = 0$: The complementary slackness condition is met no matter what value $\mu_i$ takes. And that's perfect: the constraint is active, so $\nabla g_i$ is part of the basis for $S$. So we *need* $\mu_i$ to be nonzero so that we can describe any vector in $S$ (of which $\nabla f$ is an element, as we talked about when we "explained the convenience" for the multiple-equality-constraint formulation earlier).

So the two conditions are tied to one another in this interesting way that makes it a more-or-less direct extension of the multiple-equality-constraint formulation!

**Finishing our duel with duals.**

...And with that, we've formed our dual problem with only equalities! We'll copy them here to show we have enough equations for our unknowns:

$$\begin{aligned} h_j(x) &= 0 && \text{primal feasibility} \\ \mu_i g_i &= 0 \;\forall\; i && \text{complementary slackness} \\ \nabla f &= \sum_i \mu_i \nabla g_i + \sum_j \lambda_j \nabla h_j && \text{stationarity} \end{aligned}$$

That's $k + l + n$ equations for $k + l + n$ unknowns! Now stick the system of equations into a (probably numerical) zero-finder and you're done!

**On minimizing $f$**

Let's not forget that we had done all this assuming we were *maximizing* $L$ (and thus $f$). If we were *minimizing* $f$, we would be trying to minimize $L$ and we could change the above equations in one of the following ways:

1. Replace $L$ with $L(x, \mu_1, \cdots, \mu_k, \lambda_1, \cdots, \lambda_l) = f(x) - \sum_i \mu_i g_i(x) - \sum_j \lambda_j h_j(x)$ (note the minus signs). Replace the stationarity condition with $-\nabla f = \sum_i \mu_i \nabla g_i + \sum_j \lambda_j \nabla h_j$ (again, note the minus sign).

2. Keep $L$ the same. Replace the dual feasibility condition with $\mu_i \leq 0$

The sign changes just ensure that our dual problem is still well-formed for the minimization problem (if you don't flip the sign somewhere, then we can just make arbitrarily "good" values by playing with $\mu_i$ again; and if you flip the sign in the formulation of $L$, you have to make sure you update the gradient expression accordingly (Option 1)).

### 3.2.2 If it's not that bad, why did you talk so much?

...OK, so maybe this was a lot more to explain than I had given credit at first. But it's really not that bad! The main intuition is that either:

1. the inequality constraint is lax and doesn't actually do any "constraining"; or
2. the inequality constraint is actually stopping us, in which case it just becomes another equality constraint!

All the blah-blah-blah is to make sure we dotted our $i$'s and crossed our $t$'s. (And boy, did we...)

- DK, 4/30/18

# 4. Machine learning methods.

## 4.1 Boosting.

There's a lot of buzz about the winning models of many Kaggle competitions use gradient-boosted trees. Considering its apparent effectiveness, it'd be worth understanding what "gradient-boosted trees" are and how they work. But before we jump into *that*, we should probably figure out what "boosting" means in this context.

### 4.1.1 An analogy to Taylor expansions.

**Boosting** is a meta-algorithm involving the ensembling of many "weak" learners to form arbitrarily "strong" learners. A "weak" learner is a classifier/regressor (which we'll just call a *predictor*) that can only be weakly correlated with the true underlying classication/regression (*prediction*) model we are trying to learn. A "strong" learner should be able to approximate the true prediction model arbitrarily closely.

To make sense of the somewhat vague denotation above, I like to think of writing a Taylor expansion of some (presumably non-polynomial) function, say $f(x) = e^x$. Let's say we're expanding about $x_0 = c$.[1] Then the n'th-order Taylor expansion is

$$T_{n,c}(x) = \sum_{i=0}^{n} f^{(i)}(c) \times \frac{(x-c)^i}{i!}$$

In a sense, each term of the sum is weakly correlated with the target function $f$ in that if we consider the $i$'th term $T_n[i]$, its $i$'th derivative is equal to that of $f$ at $x = c$ and so can be called a "weak predictor" of $f$:

$$T_{n,c}[i]^i(c) = f^{(i)}(c)$$

Another important note is that none of our weak predictors are redundant; each of them contains at least *some* new information about $f$.[2] Alone, $T_n[i]$ is a pretty underwhelming approximator – it's only guaranteed to approximate $f$ in a specific way at one

[1] Since the Taylor expansion is focused on being accurate *in the neighborhood of the point the expansion is centered around*, we can interpret this "Taylor model" as weighting inputs/examples near $c$ as far more important than examples from other parts of the input space.

[2] If you fancy appropriating a linear algebra term, you can also consider each of the weak predictors "linearly independent" of each other.

This is actually more on-the-nose than you might think. Just as we can think of vector spaces of $R^n$ as being spanned by $n$ linearly independent basis vectors, we can think of a *function space* being spanned by an infinite number of linearly independent basis vectors, some of which are $(x-c)^0, (x-c)^1, (x-c)^2, \cdots$. So a Taylor expansion approximates a function $f$ using a linear combination of only the basis vectors corresponding to polynomials, with each basis vectors scaled by a certain amount ($f^{(i)}(c) \times i!$).

specific point. That isn't necessarily useful for our goals — if we tried approximating $e^x$ with the Taylor expansion's second term $T_{n,c}[i = 1](x) = (x - c)$, we'd be off by hundreds, thousands, millions almost everywhere!

The power of our "weak predictors" comes when we *combine* them in some way — in the case of the Taylor expansion, a uniformly weighted sum. As we use more terms (i.e., more "weak predictors"), the ensemble becomes arbitrarily accurate to the target function $f$ near $c$ and so is a "strong predictor" of $f$.

### 4.1.2 Meta-algorithm vs. algorithm

Importantly, we said that boosting is a *meta*-algorithm, a meta-strategy that we apply onto a strategy that *actually* performs the approximating. In our "Taylor boosting" method, the strategy we use to approximate $f$ is to create an approximator $T_{n,c}[i]$ that has the same $i$'th derivative of $f$ at $c$, and the meta-strategy was to combine all those approximators together via addition. We can perform boosting on other strategies. For example, consider a "Dirac boosting" method, where our weak approximators are functions $D_i$ where

$$D_i(x) = \delta(x - i) \times f(x) = \mathbb{1}[x = i] \times f(i)$$

where $\delta$ is the Dirac delta function and $\mathbb{1}$ is the indicator function. So basically, our approximator memorizes the function at a single point perfectly and guesses that it equals zero everywhere else. Alone, this weak predictor is pretty terrible, but we can perform "Dirac boosting" by combining many such Dirac approximators $D_i$ in the following way: given an input $x$, we find the Dirac approximator whose center is closest to $x$, and output that as the result.[3] The greater the number/density of Dirac approximators along the input space, the more accurate our boosted approximator becomes![4]

Even though we changed the algorithm by which we approximated our function (using our Dirac approximators instead of Taylor approximators), we still employed the *meta*-algorithm of boosting to combine our weak predictors into a strong predictor. This should hopefully make the distinction between the two clear.

### 4.1.3 Where the analogy is left wanting.

Now of course, the analogy isn't perfect. A Taylor expansion is performed on a known target function $f$, but in a data science context, we don't have access to the actual function

[3] (Basically a *k-nearest neighbors* regressor where $k = 1$.)

[4] Fun fact: Although there are cases where a Taylor expansion can perfectly recreate the target function (e.g., $f(x) = e^x$), our "Dirac boosting" method cannot *ever* perfectly recreate *almost any* function whose input space is over $\mathbb{R}$, even if we use an infinite number of Dirac approximators in our ensemble! More specifically, it can never perfectly recreate any function $f$ if

$$\lambda(\{x \mid f(x) \neq 0\}) > 0$$

where $\lambda$ is the *Lebesgue measure*. This is because our Dirac ensemble can only be made up of a *countable* number of approximators which each only memorize a single output, and it can be shown (via, e.g., Cantor's diagonlization argument) that the set of real numbers is a larger sort of infinity (whose size is *uncountably infinite*) than the set of natural numbers (whose size is *countably infinite*).

that is responsible for our data. We *do* have a dataset, a set of samples from the actual function, which can serve as an *approximation* of the true function. So the best we can do is to use the data to create such an approximation (our model).

Since the dataset is almost certain not to contain all of the intricacies of the underlying function, we normally don't want to make our model fit the data *too* well (good ol' *overfitting*). Instead, we tune our model so that it minimizes a *loss function (or objective function)*, which we hope we've set up cleverly enough so that the model is a really good approximation of the underlying function when it reaches a minimum of the loss function (for example, by introducing terms in the loss function that punish the model for overfitting the dataset). We didn't specify an explicit loss function when improving our "Taylor" and "Dirac" ensemble examples since we could explicitly observe the actual function we were trying to fit.

### 4.1.4 Gradient boosting.

Alright then, so what is gradient boosting?

To follow the usual algorithm on which gradient boosting is employed, we'll use decision trees. We have our model $M(x)$, training examples $(x_a, y_a)$ and a loss function $L(y_a, y_p)$, where $y_p = M(x_a)$ is our current best prediction for $y_a$.[5] We'll be creating various weak learners $h_i(x)$, and we'll denote our ensembles of the first $i$ weak learners (including $i = 0$) as $E_i(x)$. We decide that the way we're going to ensemble our weak learners is by a summation $E_i(x) = \sum_i h_i(x)$, and we enter the rodeo.

We start with a baseline prediction, the mean of $y_a$, i.e. $M(x) \leftarrow E_0(x) = h_0(x) = \bar{y_a}$. Now, we're going to create and fit a new weak learner $h_1(x)$. But since we're going to make our stronger learner via $E_1(x) = E_0(x) + h_1(x)$, we can just fit $h_1(x)$ to a sort of *pseudo-residual determined by the loss function*, $y_{pr,1} = -\frac{dL}{dE_0}\Big|_{(x_a, y_a)}$ for all $(x_a, y_a)$ in our dataset. We then fit $h_1(x)$ to the dataset of pseudo-residuals $(x_a, y_{pr,1})$ as is appropriate for our weak learner — in the case of decision trees, we perform recursive partitioning until some user-specified condition is met (e.g. any further partitioning would yield leaves with fewer than $c$ entries, or the information gain ratio of any such partition would be smaller than some threshold) and then taking the mean of each leaf as the predictor for any inputs that fall into that leaf at prediction time). Now that we've fit $h_1(x)$ to the pseudo-residuals as best as we can, adding $h_1$ to our previous best predictor will further decrease the loss function, so we update our model $M(x) \leftarrow E_1(x) = E_0(x) + h_1(x)$.

Now we're sort of at the same place we were at earlier, just with a slightly better model. So we can create a new weak learner $h_2(x)$ by fitting it to the "second-order" pseudo-residuals $y_{pr,2} = -\frac{dL}{dM}\Big|_{(x_a, y_a)}$ (where now $M = E_1(x)$), then find the constant multiplier that minimizes the main loss function, then we update our model —and on and on we go.

Here, we can view the $(i+1)$'th weak predictor as attempting to approximate the *gradient* of the loss function when the previous best predictor was used. We fit to the

[5] It's worth remembering that the loss function can be as fancy as we'd like as long as its gradient can be computed analytically from the actual and predicted values.

*negative* of the derivative since we're trying to *minimize* our loss function, so we want to correct toward the direction opposite the derivative (and since we're ensembling via addition, the "opposite" part comes into play at the pseudo-residual level). Since the gradient of the loss function determines how we create our weak learners/ensemble, we call this method **gradient boosting**. And it apparently works wonders when used with decision trees as the weak predictors.[6]

The expression for our pseudo-residuals seem to come out of nowhere, but let's consider *actual* residuals for a moment: $y_r = y_p - y_a$. When we fit our weak learner to try and predict the negative of these residuals $\{-y_r\}$ and add this to our ensemble, we're taking a step in minimizing the mean-square-error of our predictor: $L = MSE = \frac{1}{2}\left(y_p - y_a\right)^2$. And you can see that $-\frac{d(MSE)}{dy_p} = -y_r$. So rather than stick to just this one type of residual, why not fit to the gradient of whatever loss function we'd like? Hence the term and our expression for "pseudo-residuals".

(Note: For further references and useful links, see this footnote.)[7]

- DK, 5/17/18 (shoot, it's late again...)

[6] For the right sorts of problems, when you tune it correctly!

[7] (The explanation provided by Abhishek Ghose in this Quora post is quite good and helped me properly grasp the concept of gradient boosting. Other main reference was this Kaggle blog post. This page allows for an interactive demo of gradient-boosted decision trees in action.)

# 5. Drafts and WIPs

(Avert your eyes!)

# Index

boosting (machine learning), 18

cross-entropy, 3

dual problem, 14

entropy (information theory)
    conditional, 5
    for distributions, 3
    for random variables, 4

function space, 18

information gain, 5

joint entropy, 4

KL divergence, 3

Lagrange multiplier, 9
Lagrangian, 10
Lebesgue measure, 19
level set, 8
loss function, 20

mutual information (information theory), 3

overfitting, 20

posterior (Bayesian statistics), 7
prior (Bayesian statistics), 7

relative entropy (information theory), 3

sizes of infinity
    countably infinite, 19
    uncountably infinite, 19