recitation 02 (optional 6.86x notes)

You do not need to read these notes at all to get an A in this course; conversely, you may not cite these notes when solving homework or exams.

B. linear models: the basics

linear approximations

He had bought a large map representing the sea,
Without the least vestige of land:
And the crew were much pleased when they found it to be
A map they could all understand.
— charles dodgson

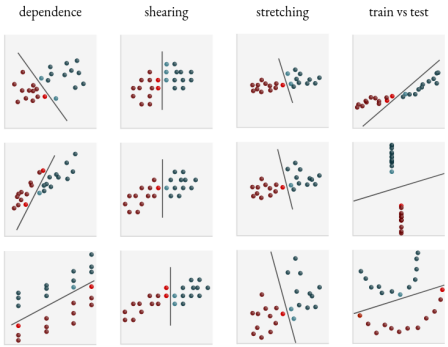
FEATURIZATION — As in the prologue, we represent our input x as a fixed-length list of numbers so that we can treat x with math. represented each photograph by 2 numbers: height and darkness. We could instead have represented each photograph by 784 numbers, one for the brightness at each of the $28 \cdot 28 = 784$ many pixels. Or by 10 numbers, each measuring the overlap of x's ink with that of "representative" photos of the digits 0 through 9.

When we choose how to represent x by a list of numbers, we're choosing a **featurization**. We call each number a "feature". For example, height and darkness are two features.

TODO: mention one-hot, etc TODO: mention LOWRANK (sketching; also, for multiregression) There are lots of interesting featurizations, each making different patterns easier to learn. So we judge a featurization with respect to the kinds of patterns we use it to learn. TODO: graphic of separability; and how projection can reduce it Learning usually happens more accurately, robustly, and interpretably when our featurization is abstract (no irrelevant details) but complete (all relevant details), compressed (hard to predict one feature from the others) but accessible (easy to compute interesting properties from features).

data-based featurizations via kernels will soon learn featurizations hand featurization in kaggle and medicine

TODO: projectivization (say this foreshadows kernel discussion?)



GEOMETRY OF FEATURE-SPACE — Now say we've decided on a **featurization** of our input data x.

$$f_{a,b}(x) = \begin{array}{l} 0 \text{ if } a \cdot \text{width}(x) + b \cdot \text{darkness}(x) < 0 \text{ else } 1 \end{array}$$

Just because two features both correlate with a positive label ($y = +1$) doesn't mean both features will have positive weights. In other words, it could be that the *blah*-feature correlates with $y = +1$ in the training set and yet, according to the best hypothesis for that training set, the bigger a fresh input's blah feature is, the *less* likely its label is to be $+1$, all else being equal. That last phrase "all else being equal" is crucial, since it refers to our choice of coordinates. Illustrate 'averaging' of good features vs 'correction' of one feature by another (how much a feature correlates with error) In fact, t This is the difference between *independence* and *conditional independence*.

Shearing two features together — e.g. measuring cooktime-plus-preptime together with cooktime rather than preptime together with cooktime — can impact the decision boundary. Intuitively, the more stretched out a feature axis is, the more the learned hypothesis will rely on that feature.

Stretching a single feature — for instance, measuring it in centimeters instead

of meters — can impact the decision boundary as well. Intuitively, the more stretched out a feature axis is, the more the learned hypothesis will rely on that feature.

Note that

TODO: example featurization (e.g. MNIST again?)

RICHER OUTPUTS — We've learned how to construct a set $\mathcal{H}$ of candidate patterns

$$f_{\vec{w}}(\vec{x}) = \text{step}(\vec{w} \cdot \vec{x})$$

that map (a featurization of) a prompt $\vec{x}$ to a binary answer $y = 0$ or $y = 1$.

What if we're interested in predicting a richer kind of y? For example, maybe there are k many possible values for y instead of just 2. Or maybe there are infinitely many possible values — say, if y is a real number or a length-l list of real numbers. Or maybe we want the added nuance of predicting probabilities, so that f might output "20% chance of label $y = 0$ and 80% chance of label $y = 1$" instead of just "$y = 1$".

I'll write formulas and then explain.

$$f_{\vec{w}_i : 0 \le i < k}(\vec{x}) = \text{argmax}_i(\vec{w}_i \cdot \vec{x})$$

$$f_{\vec{w}}(\vec{x}) = \vec{w} \cdot \vec{x}$$

TODO: add multi-output regression?

$$f_{\vec{w}_i : 0 \le i < k}(\vec{x}) = \text{normalize}(\exp(\vec{w}_i \cdot \vec{x}) : 0 \le i < k)$$

TODO: interpret

TODO: discuss measures of goodness!

TODO: talk about structured (trees/sequences/etc) output!

iterative optimization

Hey Jude, don't make it bad
Take a sad song and make it better
Remember to let her under your skin
Then you'll begin to make it
Better, better, better, better, better, better, ...
— paul mccartney, john lennon

(STOCHASTIC) GRADIENT DESCENT — We have a collection $\mathcal{H}$ of candidate patterns together with a function $1 - \text{trn}_{\mathcal{S}}$ that tells us how good a candidate is. In §A we found a nearly best candidate by brute-force search over all of $\mathcal{H}$; this doesn't scale to the general case, where $\mathcal{H}$ is intractably large. So: *what's a faster algorithm to find a nearly best candidate?*

← We view $1 - \text{trn}_{\mathcal{S}}$ as an estimate of our actual notion of "good": $1 - \text{tst}$.

A common idea is to start arbitrarily with some $h_0 \in \mathcal{H}$ and repeatedly improve to get $h_1, h_2, \cdots$. Two questions are: *how do we select* h_{t+1} *in terms of* h_t? And *how do we know when to stop*? We'll discuss termination conditions later — for now, let's agree to stop at h_{10000}.

As for selecting a next candidate, we'd like to use more detailed information on h_t's inadequacies to inform our proposal h_{t+1}. Intuitively, if h_t misclassifies a particular $(x_n, y_n) \in \mathcal{S}$, then we'd like h_{t+1} to be like h_t but nudged a bit in the direction of accurately classifying (x_n, y_n).

FILL IN FOR PROBABILITIES (LOGISTIC) MODEL

This is the idea of **gradient descent**. MOTIVATE AND GIVE PSEUDOCODE FOR STOCHASTIC GD

Given a list of training examples, a probability model, and a hypothesis $\vec{w}$, we can compute $\vec{w}$'s asserted probability that the training xs correspond to the training ys. It's reasonable to choose $\vec{w}$ so that this probability is maximal. This method is called **maximum likelihood estimation (MLE)**.

LOGISTIC MODELS —

The probability of a bunch of independent observations is the same as the product of probabilities of the observations. Taking logarithms turns products into more manageable sums. And — this is a historical convention — we further flip signs $\pm$ to turn maximization to minimization. After this translation, MLE with the logistic model means finding $\vec{w}$ that minimizes

$$\sum_i \log(1 + \exp(-y_i \vec{w} \cdot \vec{x}_i)) = \sum_i \text{softplus}(-y_i \vec{w} \cdot \vec{x}_i)$$

Here, softplus is our name for the function that sends z to $\log(1 + \exp(z))$.

mention convexity and convergence? show trajectory in weight space over time – see how certainty degree of freedom is no longer redundant? ("markov")

HUMBLE MODELS — Let's generalize logistic classification to allow for *unknown unknowns*. We'll do this by allowing a classifier to distribute probability mass not only among the labels $\mathcal{Y}$ but also to a special class $\star$ that means "no comment" or "alien input". A logistic classifier always sets $\mathbb{P}_{y|x}[\star|x] = 0$. But we could use other probability models that put nonzero mass on "no comment"; different models give different learning programs. Here are four models we might try:

	LOGISTIC	PERCEPTRON	SVM	GAUSS-GEN
$\mathbb{P}_{y\|x}[-1\|x]$	$u^-/(u^- + u^+)$	$u^- \cdot (u^- \wedge u^+)/2$	$u^- \cdot (u^-/e \wedge u^+)/2$	$u^- \cdot \text{bumps}$
$\mathbb{P}_{y\|x}[+1\|x]$	$u^+/(u^- + u^+)$	$u^+ \cdot (u^- \wedge u^+)/2$	$u^+ \cdot (u^- \wedge u^+/e)/2$	$u^+ \cdot \text{bumps}$
$\mathbb{P}_{y\|x}[\star\|x]$	$1 -$ above $= 0$	$1 -$ above	$1 -$ above	$1 -$ above
loss name	softplus$(\cdot)$	srelu$(\cdot)$	hinge$(\cdot)$	quad$(\cdot)$
formula	$\log(1 + \exp(\cdot))$	$\max(1, \cdot) + 1$	$\max(1, \cdot + 1)$	??
update	$1/(1 + \exp(+yd))$	step$(-yd)$	step$(1 - yd)$	??
outliers	vulnerable	robust	robust	vulnerable
inliers	sensitive	blind	sensitive	blind
humility	low	low	high	high

Table 1: **Four popular models for binary classification. Top rows:** Modeled chance given x that $y = +1$ or -1 or $\star$. We use $d = \vec{w} \cdot \vec{x}$, $u^{\pm} = e^{\pm d/2}$, $a \wedge b = \min(a, b)$ to save ink. And bumps $= (\phi(d-1) + \phi(d+1))/2$ with $\phi(\cdot)$ the standard normal density function. **Middle rows:** neg-log-likelihood losses. An SGD step looks like $\vec{w}_{t+1} = \vec{w}_t + (\eta \cdot \text{update} \cdot y\vec{x})$. **Bottom rows:** All models respond to misclassifications. But are they robust to well-classified outliers? Sensitive to well-classified inliers? Exercise: Fill in the ??s in the rightmost column of the table above.

MLE with the perceptron model, svm model, or gauss-gen model minimizes the same thing, but with $\text{srelu}(z) = \max(0, z) + 1$, $\text{hinge}(z) = \max(0, z + 1)$, or $\text{quad}(z) = \cdots$ instead of $\text{softplus}(z)$.°

← Other models we might try will induce other substitutes for softplus. E.g. $z \mapsto \text{hinge}(z)^2$ or $z \mapsto \text{avg}(z, \sqrt{z^2 + 4})$.

Two essential properties of softplus are that: (a) it is convex and (b) it upper bounds the step function. Note that srelu, hinge, and quad also enjoy these properties. Property (a) ensures that the optimization problem is relatively easy — under mild conditions, gradient descent is guaranteed to find a global minimum. By property (b), the total loss on a training set upper bounds the rate of erroneous classification on that training set. So loss is a *surrogate* for (in)accuracy: if the minimized loss is nearly zero, then the training accuracy is nearly 100%.

The perceptron satisfies (b) in a trivial way that yields a trivial bound of 100% on the error rate.

training behavior!! response to outliers support vectors

IDEAS IN STOCHASTIC GRADIENT DESCENT — learning rate as metric; robustness to 2 noise structures nesterov momentum decaying step size; termination conditions batch normalization

priors and generalization

A child's education should begin at least 100 years before [they are] born.
— oliver wendell holmes jr

ON OVERFITTING — $\mathbb{E}$ and $\max$ do not commute point estimates vs bayesian decision theory interpolation does not imply overfitting

LOG PRIORS AND BAYES — fill in computation and bases visual illustration of how choice of L2 dot product matters ℓ^p regularization; sparsity eye regularization example!

HIERARCHY, MIXTURES, TRANSFER — k-fold cross validation dimension-based generalization bound bayesian information criterion

ESTIMATING GENERALIZATION — k-fold cross validation dimension-based generalization bound bayesian information criterion

G. math refresher

linear algebra

Linear algebra is the part of geometry that focuses on when a point is the origin, when a 'line' is a straight, and when two straight lines are parallel. Linear algebra thus helps us deal with the preceding pictures mathematically and automatically. The concept of 'straight lines' gives a simple, flexible model for extrapolation from known points to unknown points. That is intuitively why linear algebra will be crucial at every stage of 6.86x.

COLUMN VECTORS AND ROW VECTORS — The elements of linear algebra are **column vectors** and **row vectors**.° We have a set V of "column vectors". We're allowed to find V's zero vector and to add or scale vectors in V to get other vectors in V. V is the primary object we hold in our mind; perhaps, if we're doing image classification, then each column vector represents a photograph. We use the word "space" or "vector space" when talking about V to emphasize that we'd like to exploit visual intuitions when analyzing V. In short: we imagine each column vector as a point in space, or, if we like, as an arrow from the zero vector to that point.

← Though we represent the two similarly in a computer's memory, they have different geometric meanings. We save much anguish by remembering the difference.

Now, associated to V is the set of "row vectors". Under the hood, a row vector is a linear function from V to the real numbers $\mathbb{R}$. We imagine each row vector not as an arrow but as a "linear" heat map or altitude map that assigns to each point in V a numeric "intensity". We can visualize a row vector the same way makers of geographic maps do: using contours for where in V the row vector attains values $\cdots, -2, -1, 0, +1, +2, \cdots$. These will be a bunch of uniformly spaced parallel "planes". The spacing and orientation of the planes depends on and determines the row vector. In short, we imagine each row vector as a collection of parallel planes in space.

Informally: a column vector is a noun or thing whereas a row vector is a adjective or property. The degree to which a property holds on a thing (or a description is true of a thing) is gotten by evaluating the row vector on the column vector — remember, a row vector is a function, so we can do this evaluation. Geometrically, if a row vector is a bunch of planes and a column vector is an arrow, the two evaluate to a number: the number of planes that the arrow pierces. Intuitively, an example of a column vector might be "this particular photo of my pet cow"; an example of a row vector might be "redness of the left half (of the input photo)". If we evaluate this row vector on this column vector, then we get a number indicating how intensely true it is that the left half of that particular photo is red.

INNER PRODUCTS — Now, here is the key point: the engine behind generalization in machine learning (at least, the machine learning we'll do in Units 1 and 2; and less visibly but still truly in more than half of each of Units 3,4,5) is the ability to translate between things and properties. If "my pet cow" is a thing, then "similar to my pet cow" is a property. The whole project of machine learning is to define and implement this word "similar to". When we define and implement well, our

We can draw an analogy with syntax vs semantics. This pair of concepts pops up in linguistics, philosophy, circuit engineering, quantum physics, and more, but all we need to know is that: semantics is about things while syntax is about descriptions of things. The two concepts relate in that, given a set of things, we can ask for the set of all descriptions that hold true for all those things simultaneously. And if we have a set of descriptions, we can ask for the set of all things that satisfy all those descriptions simultaneously. These two concepts stand in formal opposition in the sense that: if we have a set of things and make it bigger, then the set of descriptions that apply becomes smaller. And vice versa. Then a column vector is an object of semantics. And a row vector is an object of syntax.

programs can generalize successfully from training examples to new, previously unseen situations, since they will be able to see which of the training examples the new situations are similar to. Since "similar to" transforms things to properties, the linear algebra math behind "similar to" is a function from column vectors to row vectors. This brings us to...

... inner products, aka kernels. An inner product is just a fancy word for a (linear) function from column vectors to row vectors. We actually demand that this linear function has two nice properties: FIRST, it should have an inverse. That is, it ought to be a two way bridge between column vectors and row vectors, allowing us to translate things to descriptions and vice versa. SECOND, it should be symmetric. This means that if we have two things, say "my pet cow" and "my pet raccoon", then the degree to which "my pet raccoon" has the property "is similar to my pet cow" ought to match the degree to which "my pet cow" has the property "is similar to my pet raccoon". Any invertible, symmetric, linear function from column vectors to row vectors is called an inner product. Kernel is a synonym here.

Beware: the same word, "kernel", has different meanings depending on context.

There are generally infinitely many inner products. Which one we choose changes the generalization properties of our machine learning program. Practically, if we are doing machine learning in a concrete situation, then we want to choose an inner product that reflects our human intuition and experience and domain knowledge about the right notion of "similarity" in that situation.

Any inner product P from column vectors to row vectors induces notions of length and angle. We just define a column vector v's length by $\sqrt{P(v)(v)}$. Call that quantity $\|v\|$. And we define the angle $\alpha(v, w)$ between two non-zero column vectors v,w by $P(v)(w) = \|v\| \cdot \|w\| \cdot \cos \alpha(v, w)$. We make these definitions so as to match the Pythagorean theorem from plane geometry. So once we choose which inner product we'll use (out of the infinitely many choices), we get the concepts of euclidean geometry for free. Immediately following from that definition of angle, we get that if two column vectors have vanishing inner product (i.e., if $P(v)(w) = 0$), then those vectors are at right angles (i.e. $\alpha(v)(w) = \pi/2$).

You can google up proofs of the Pythagorean theorem (many are quick and beautiful) if you wish to dig deeper.

Now, sometimes (but most of the time not), we are blessed in that we know more about our situation than just the space V of things. Specifically, V might come with a canonical basis. This just means that V comes marked with "the right" axes with respect to which we ought to analyze vectors in V. In this fortunate case, there is also a canonical inner product. It's called dot product. Again, I want to emphasize that a plain vector space doesn't come with a dot product. We need a basis for that.

FILL IN LINEAR DECISION BOUNDARY! (remark on featurization and argmax nonlinearities)

We may **evaluate** a row vector on a column vector. FILL IN A **dot product** is a way of translating between row and column vectors. FILL IN: DISCUSS GENERALIZATION; (DISCUSS ANGLE, TOO)