linear approximation

By the end of this section, you'll be able to
- define a class of linear, probabilistic hypotheses appropriate to a given classification task, by: designing features; packaging the coefficients to be learned as a matrix; and selecting a probability model (logistic, perceptron, SVM, etc).
- compute the loss suffered by a probabilistic hypothesis on given data

TWO THIRDS BETWEEN DOG AND COW — Remember: our Unit 1 motto is to *learn linearities flanked by hand-coded nonlinearities*:

$$\mathcal{X} \xrightarrow[\text{not learned}]{\text{featurize}} \mathbb{R}^2 \xrightarrow[\textbf{learned!}]{\text{linearly combine}} \mathbb{R}^1 \xrightarrow[\text{not learned}]{\text{read out}} \mathcal{Y}$$

We design the nonlinearities to capture domain knowledge about our data and goals. Here we'll design nonlinearities to help model *uncertainty* over $\mathcal{Y}$. We can do this by choosing a different read-out function. For example, representing distributions by objects `{3:prob_of_three, 1:prob_of_one}`, we could choose:

```
prediction = {3 : 0.8 if threeness[0]>0. else 0.2,
              1 : 0.2 if threeness[0]>0. else 0.8 }
```

If before we'd have predicted "the label is 3", we now predict "the label is 3 with 80% chance and 1 with 20% chance". This hard-coded 80% *could* suffice.° But let's do better: intuitively, a 3 is more likely when `threeness` is huge than when `threeness` is nearly zero. So let's replace that 80% by some smooth function of `threeness`. A popular, theoretically warranted choice is $\sigma(z) = 1/(1+\exp(-z))$:°

← As always, it depends on what specific thing we're trying to do!

← σ, the **logistic** or **sigmoid** function, has linear log-odds: $\sigma(z)/(1-\sigma(z)) = \exp(z)/1$. It tends exponentially to the step function. It's symmetrical: $\sigma(-z) = 1-\sigma(z)$. Its derivative concentrates near zero: $\sigma'(z) = \sigma(z)\sigma(-z)$. Food For Thought: Plot $\sigma(z)$ by hand.

```
sigma = lambda z : 1./(1.+np.exp(-z))
prediction = {3 :    sigma(threeness[0]),
              1 : 1.-sigma(threeness[0]) }
```

Given training inputs x_i, a hypothesis will have "hunches" about the training outputs y_i. Three hypotheses $h_{\text{three!}}$, h_{three}, and h_{one} might, respectively, confidently assert $y_{42} = 3$; merely lean toward $y_{42} = 3$; and think $y_{42} = 1$. If in reality $y_{42} = 1$ then we'd say h_{one} did a good job, h_{three} a bad job, and $h_{\text{three!}}$ a very bad job on the 42nd example. So the training set "surprises" different hypotheses to different degrees. We may seek a hypothesis $h_\star$ that is minimally surprised, i.e., usually confidently right and when wrong not confidently so. In short, by outputting probabilities instead of mere labels, we've earned this awesome upshot: *the machine can automatically calibrate its confidence levels!* It's easy to imagine how important this calibration is in language, self-driving, etc.

INTERPRETING WEIGHTS — We note two aspects of the 'intuitive logic' of weights.

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. See the figure, left three panels.

Moreover, transforming coordinates, even linearly, can alter predictions. For example, if we shear two features together — say, by using cooktime-plus-preptime and cooktime as features rather than preptime and cooktime as features — this can impact the decision boundary. Of course, the decision boundary will look different because we're in new coordinates; but we mean something more profound: if we train in old coordinates and then predict a datapoint represented

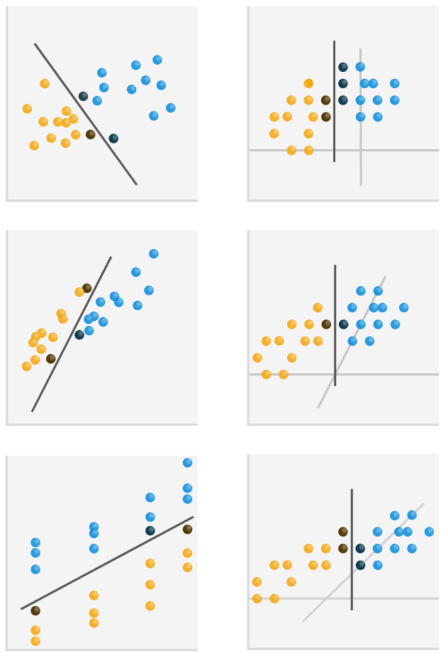

Figure 10: **Relations between feature statistics and optimal weights.** Each of these six figures shows a different binary classification task along with a maximum-margin hypothesis. We shade the datapoints that achieve the margin. — **Left:** *positive weights don't imply positive correlation!* — **Right:** *presenting the same information in different coordinates alters predictions!*

in old coordinates, we might get a different prediction than if we train in new coordinates and then predict a datapoint represented in new coordinates! See the figure, right three panels: here, the intersection of the two gray lines implicitly marks a testing datapoint that experiences such a change of prediction as we adopt different coordinates. *Intuitively, the more stretched out a feature axis is, the more the learned hypothesis will rely on that feature.*

Food For Thought: Understand this paragraph from the point of view of the L2 regularizer.

DESIGNING FEATURIZATIONS — We represent our input x as a fixed-length list of numbers so that we can "do math" to x. For instance, we could represent a 28×28 photo by 2 numbers: its overall brightness and its dark part's width. Or we could represent it by 784 numbers, one for the brightness at each of the $28 \cdot 28 = 784$ many pixels. Or by 10 numbers that respectively measure the overlap of x's ink with that of "representative" photos of the digits 0 through 9.

A way to represent x as a fixed-length list of numbers is a **featurization**. Each map from raw inputs to numbers is a **feature**. Different featurizations make different patterns easier to learn. We judge a featurization not in a vacuum but with respect to the kinds of patterns we use it to learn. information easy for the machine to use (e.g. through apt nonlinearities) and throw away task-irrelevant information (e.g. by turning 784 pixel brightnesses to 2 meaningful numbers).

Here are two themes in the engineering art of featurization.°

← For now, we imagine hand-coding our features rather than adapting them to training data. We'll later discuss adapted features; simple examples include thresholding into **quantiles** based on sorted training data (*Is x more than the median training point?*), and choosing coordinate transforms that measure similarity to **landmarks** (*How far is x from each of these 5 "representative" training points?*). Deep learning is a fancy example.

Predicates. If domain knowledge suggests some subset $S \subseteq \mathcal{X}$ is salient, then we can define the feature

$$x \mapsto 1 \text{ if } x \text{ lies in } S \text{ else } 0$$

The most important case helps us featurize *categorical* attributes (e.g. kind-of-chess-piece, biological sex, or letter-of-the-alphabet): if an attribute takes K possible values, then each value induces a subset of $\mathcal{X}$ and thus a feature. These features assemble into a map $\mathcal{X} \to \mathbb{R}^K$. This **one-hot encoding** is simple, powerful, and common. Likewise, if some attribute is *ordered* (e.g. $\mathcal{X}$ contains geological strata) then interesting predicates may include **thresholds**.

Coordinate transforms. Applying our favorite highschool math functions gives new features $\tanh(x[0]) - x[1]$, $|x[1]x[0]| \exp(-x[2]^2)$, $\cdots$ from old features $x[0], x[1], \cdots$. We choose these functions based on domain knowledge; e.g. if $x[0], x[1]$ represent two spatial positions, then the distance $|x[0] - x[1]|$ may be a useful feature. One systematic way to include nonlinearities is to include all the monomials (such as $x[0]x[1]^2$) with not too many factors — then linear combinations are polynomials The most important nonlinear coordinate transform uses all monomial features with 0 or 1 many factors — said plainly, this maps

$$x \mapsto (1, x)$$

This is the **bias trick**. Intuitively, it allows the machine to learn the threshold above which three-ishness implies a three.

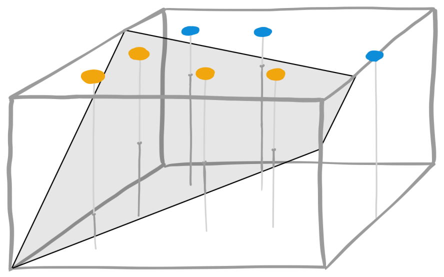

Figure 11: **The bias trick helps us model 'offset' decision boundaries.** Here, the origin is the lower right corner closer to the camera. Our raw inputs $x = (x[0], x[1])$ are 2-dimensional; we can imagine them sitting on the bottom face of the plot (bottom ends of the vertical stems). But, within that face, no line through the origin separates the data well. By contrast, when we use a featurization $(1, x[0], x[1])$, our data lies on the top face of the plot; now a plane through the origin (shown) successfully separates the data.

HUMBLE MODELS — Let's modify logistic classification to allow for *unknown unknowns*. We'll do this by allowing a classifier to allot probability mass not only among labels in $\mathcal{Y}$ but also to a special class $\star$ that means "no comment" or "alien input". A logistic classifier always sets $p_{y|x}[\star|x] = 0$, but other probability models may put nonzero mass on "no comment". For example, consider:

	LOGISTIC	PERCEPTRON	SVM
$p_{y\|x}[+1\|x]$	$\oplus/(\ominus+\oplus)$	$\oplus\cdot(\ominus\wedge\oplus)/2$	$\oplus\cdot(\ominus\wedge\oplus/e)/2$
$p_{y\|x}[-1\|x]$	$\ominus/(\ominus+\oplus)$	$\ominus\cdot(\ominus\wedge\oplus)/2$	$\ominus\cdot(\ominus/e\wedge\oplus)/2$
$p_{y\|x}[\star\|x]$	$1-$ above $=0$	$1-$ above	$1-$ above
outliers	responsive	robust	robust
inliers	sensitive	blind	sensitive
acc bnd	good	bad	good
loss name	softplus$(\cdot)$	srelu$(\cdot)$	hinge$(\cdot)$
formula	$\log_2(1+e^{(\cdot)})$	$\max(1,\cdot)+1$	$\max(1,\cdot+1)$
update	$1/(1+e^{+y\mathfrak{d}})$	$\text{step}(-y\mathfrak{d})$	$\text{step}(1-y\mathfrak{d})$

Table 1: **Three popular models for binary classification. Top rows:** Modeled chance given x that $y=+1,-1,\star$. We use $\mathfrak{d}=\vec{w}\cdot\vec{x}$, $\oplus=e^{+\mathfrak{d}/2}, \ominus=e^{-\mathfrak{d}/2}$, $a\wedge b=\min(a,b)$ to save ink. **Middle rows:** All models respond to misclassifications. But are they robust to well-classified outliers? Sensitive to well-classified inliers? **Bottom rows:** For optimization, which we'll discuss later, we list (negative log-probability) losses. An SGD step looks like

$$\vec{w}_{t+1}=\vec{w}_t+\eta\cdot\text{update}\cdot y\vec{x}$$

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

Two essential properties of softplus are that: (a) it is convex° and (b) it upper bounds the step function. Note that srelu and hinge also enjoy these properties. Property (a) ensures that the optimization problem is relatively easy — under mild conditions, gradient descent will 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%.°

← A function is **convex** when its graph is bowl-shaped rather than wriggly. It's easy to minimize convex functions by 'rolling downhill', since we'll never get stuck in a local wriggle. Don't worry about remembering or understanding this word.

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

So we have a family of related models: **logistic**, **perceptron**, and **SVM**. In Project 1 we'll find hypotheses optimal with respect to the perceptron and SVM models (the latter under a historical name of **pegasos**), but soon we'll focus mainly on logistic models, since they fit best with deep learning.

RICHER OUTPUTS: MULTIPLE CLASSES — We've explored hypotheses $f_W(x)=\text{readout}(W\cdot\text{featurize}(x))$ where W represents the linear-combination step we tune to data. We began with **hard binary classification**, wherein we map inputs to definite labels (say, $y=\text{cow}$ or $y=\text{dog}$):

$$\text{readout}(\mathfrak{d})=\text{“cow if } 0<\mathfrak{d} \text{ else dog”}$$

We then made this probabilistic using σ. In such **soft binary classification** we return (for each given input) a *distribution* over labels:

$$\text{readout}(\mathfrak{d})=\text{“chance } \sigma(\mathfrak{d}) \text{ of cow; chance } 1-\sigma(\mathfrak{d}) \text{ of dog”}$$

Remembering that $\sigma(\mathfrak{d}):(1-\sigma(\mathfrak{d}))$ are in the ratio $\exp(\mathfrak{d}):1$, we rewrite:

$$\text{readout}(\mathfrak{d})=\text{“chance of cow is } \exp(\mathfrak{d})/Z_{\mathfrak{d}}\text{; of dog, } \exp(0)/Z_{\mathfrak{d}}\text{”}$$

I hope some of you felt bugged by the above formulas' asymmetry: W measures "cow-ishness minus dog-ishness" — why not the other way around? Let's describe the same set of hypotheses but in a more symmetrical way. A common theme in mathematical problem solving is to trade irredundancy for symmetry (or vice versa). So let's posit both a W_{cow} *and* a W_{dog}. One measures "cow-ishness"; the other, "dog-ishness". They assemble to give W, which is now a matrix of shape $2 \times$number-of-features. So ϑ is now a list of 2 numbers: ϑ_{cow} and ϑ_{dog}. Now $\vartheta_{\text{cow}} - \vartheta_{\text{dog}}$ plays the role that ϑ used to play.

Then we can do hard classification by:

$$\text{readout}(\vartheta) = \text{argmax}_y \vartheta_y$$

and soft classification by:

$$\text{readout}(\vartheta) = \text{"chance of } y \text{ is } \exp(\vartheta_y)/Z_\vartheta\text{"}$$

To make probabilities add to one, we divide by $Z_\vartheta = \sum_y \exp(\vartheta_y)$.

Behold! By rewriting our soft and hard hypotheses for binary classification, we've found formulas that also make sense for more than two classes! The above readout for **soft multi-class classification** is called **softmax**.

RICHER OUTPUTS: BEYOND CLASSIFICATION — By the way, if we're trying to predict a real-valued output instead of a binary label — this is called **hard one-output regression** — we can simply return ϑ itself as our readout:

$$\text{readout}(\vartheta) = \vartheta$$

← VERY OPTIONAL PASSAGE

This is far from the only choice! For example, if we know that the true ys will always be positive, then $\text{readout}(\vartheta) = \exp(\vartheta)$ may make more sense. I've encountered a learning task (about alternating current in power lines) where what domain knowledge suggested — and what ended up working best — were trigonometric functions for featurization and readout! There are also many ways to return a distribution instead of a number. One way to do such **soft one-output regression** is to use normal distributions:◦

← Ask on Piazza about the interesting world of alternatives and how they influence learning!

$$\text{readout}(\vartheta) = \text{"normal distribution with mean } \vartheta \text{ and variance 25"}$$

Or we could allow for different variances by making ϑ two-dimensional and saying $\cdots$ mean ϑ_0 and variance $\exp(\vartheta_1)$. By now we know how to do **multi-output regression**, soft or hard: just promote W to a matrix with more output dimensions.

Okay, so now we know how to use our methods to predict discrete labels or real numbers. But what if we want to output structured data like text? A useful principle is to factor the task of generating such "variable-length" data into many smaller, simpler predictions, each potentially depending on what's generated so far. For example, instead of using W to tell us how to go from (features of) an image x to a whole string y of characters, we can use W to tell us, based on an image x together with a partial string y', either what the next character is OR that the string should end. So if there are 27 possible characters (letters and space) then this is a $27+1$-way classification problem:

$$(\text{Images} \times \text{Strings}) \to \mathbb{R}^{\cdots} \to \mathbb{R}^{28} \to \text{DistributionsOn}(\{'a', \cdots, 'z', ' ', \text{STOP}\})$$

We could implement this function as some hand-crafted featurization function from Images $\times$ Strings to fixed-length vectors, followed by a learned W, followed by softmax.

Food For Thought: A "phylogenetic tree" is something that looks like

```
(dog.5mya.(cow.2mya.raccoon))
```

or

```
((chicken.63mya.snake).64mya.(cow)).120mya.(snail.1mya.slug)
```

That is, a tree is either a pair of trees together with a real number OR a species name. The numbers represent how long ago various clades diverged. Propose an architecture that, given a list of species, predicts a phylogenetic tree for that species. Don't worry about featurization.