featurization

By the end of this section, you'll be able to

- tailor a hypothesis class by designing features that reflect domain knowledge
- recognize the geometric patterns that common nonlinear featurizations help express

DESIGNING FEATURIZATIONS — Remember: our motto in Units 1 and 2 is to *learn linear maps flanked by hand-coded nonlinearities*. That is, we consider hypotheses of this format:

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

In this section and the next we'll design those non-learned functions — the featurizers and readouts, respectively — to construct a hypothesis class $\mathcal{H}$ suitable given our domain knowledge and our goals. In this section, we'll discuss how the design of features determines the patterns that the machine is able to express; feature design can thus make or break an ML project.

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 darknesses 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

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

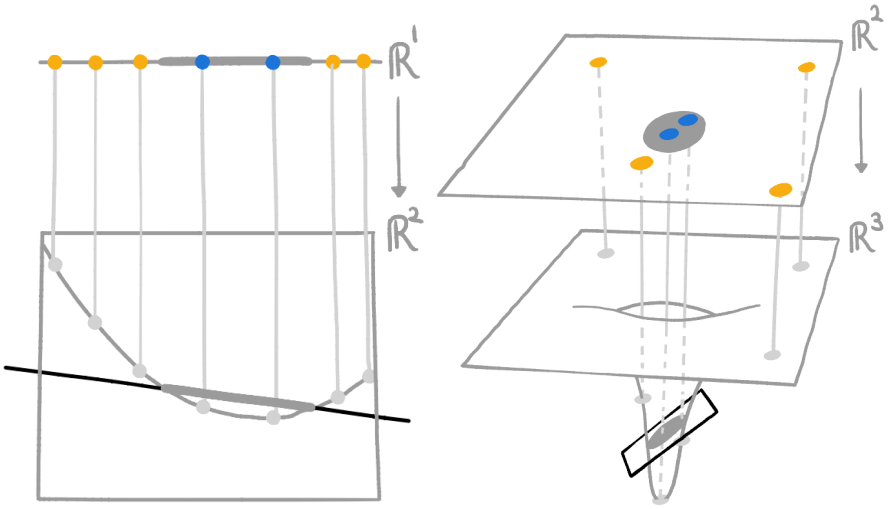
The most important case helps us featurize *categorical* attributes (e.g. kind-of-chess-piece, biological sex, or letter-of-the-alphabet): if an attribute takes K possible values, then each value induces a subset of $\mathcal{X}$ and thus a feature. These features assemble into a map $\mathcal{X} \to \mathbb{R}^K$. 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.

This bias trick is non-linear in that it shifts the origin. Fancier non-linearities enrich our vocabulary of hypotheses in fancier ways. Take a look these decision boundaries using a degree-2 monomial feature (left) and (right) a non-polynomial 'black hole' feature for two cartoon datasets:



Say $\mathcal{X} = \mathbb{R}^2$ has as its two raw features the latitude of low-orbit satellite and the latitude of a ground-based receiver, measured in degrees-from-the-equator. So the satellite is in earth's north or southern hemisphere and likewise for the receiver. We believe that whether or not the two objects are in the *same* hemisphere — call this situation 'visible' — is highly relevant to our ultimate prediction task.
Food For Thought: For practice, we'll first try to classify xs by visibility. Define a degree-2 monomial feature $\varphi : \mathcal{X} \to \mathbb{R}^1$ whose sign (± 1) tells us whether x is visible.° *Can direct linear use of* x*, even with the bias trick, predict the same?*
In reality we're interested in a more complex binary classification task on $\mathcal{X}$. For this we choose as features all monomials of degrees $0, 1, 2$ to get $\varphi : \mathcal{X} \to \mathbb{R}^6$.°
Food For Thought: Qualitatively describe which input-output rules $h : \mathcal{X} \to \{+1, -1\}$ we can express. For instance, which of these rules in the margin° can we express? (Formally, we can 'express' h if $h(x) = \text{sign}(w \cdot \varphi(x))$ for some w.)

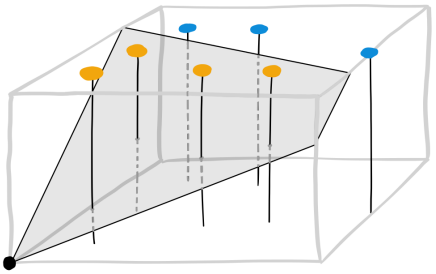
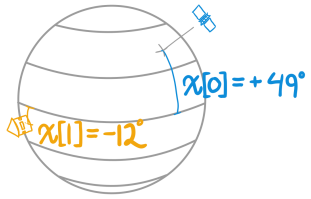


Figure 13: **The bias trick helps us model 'offset' decision boundaries.** Here, the origin is the lower right corner closer to the camera. Our raw inputs $x = (x[0], x[1])$ are 2-D; we can imagine them sitting on the bottom face of the plot (bottom ends of the vertical stems). But, within that face, 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.

Figure 14: **Fancier nonlinearities help us model curvy patterns using linear weights. Left**: Using a quadratic feature (and the bias trick, too) we can learn to classify points in $\mathcal{X} = \mathbb{R}^1$ by whether they are in some *interval* **gray region**. We don't know the interval beforehand, but fortunately, different linear decision boundaries (**black lines**) in feature-space give rise to *all possible* intervals! We may optimize as usual to find the best interval.
Right: No line through $\mathcal{X} = \mathbb{R}^2$ separates our raw data. But a **(black) hyperplane** *does* separate our featurized data. We are thus able to learn a hypothesis that predicts the blue label for xs inside the **gray region**. Intuitively, the 'black hole' feature measures whether an input x is nearby a certain point in $\mathcal{X}$.
Food For Thought: Our 'black hole' decision boundary actually has two parts, one finite and one infinite. The infinite part is slightly 'off-screen'. Sketch the full situation out!



← Thus, degree-2 monomials help model 2-way interactions between features.

← Food For Thought: φ maps to $\mathbb{R}^6$, i.e., we have exactly 6 monomials. Verify this!

← RULE A "+1 exactly when the satellite is between the Tropics of Cancer and of Capricorn" RULE B "+1 exactly when satellite and receiver are at the same latitude, up to 3° of error" RULE C "+1 exactly when both objects are above the Arctic Circle"

INTERPRETING WEIGHTS — The features we design ultimately get used according to a weight vector. So we stand to gain from deeper understanding of what weights 'mean'. Here we'll discuss three aspects of the 'intuitive logic' of weights.

First, **weights are not correlations**. A feature may correlate with a positive label (say, $y = +1$) yet fail to have a positive weight. That is: the *blah*-feature could correlate 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.

In Figure 15's center-left panel, the weight for brightness is negative even though both features positively correlate with blue! This is because brightness correlates *even better* with the *error* of soley-width-based prediction. So the optimal hypothesis, intuitively, uses the brightness as a 'correction' to width. This contrasts with the top-left panel, where the both correlations are still positive and both weights are positive. Intuitively, the optimal hypothesis here reduces noise by averaging a solely-brightness-based prediction with a solely-width-based one.

Second, **representing the same information two different ways can alter predictions**. A featurization doesn't just supply raw data: it also suggests to the machine which patterns are possible and, among those, which are plausible.

The bias trick and other nonlinear coordinate transforms illustrate this. But even *linear*, origin-preserving coordinate-transforms can alter predictions. For example, if we shear two features together — say, by using {preptime-plus-cooktime and cooktime} as features rather than {preptime and cooktime} — 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 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 right three panels: the intersection of the two gray lines implicitly marks a testing point for which we predict different labels as we change coordinates. *Intuitively, the more stretched out a feature axis is, the more the learned hypothesis will rely on that feature.*°

Third, **features interact**. It's useful to think about 'the goodness' of individual features, but it's but it's also good to realize that the reality is messier: a feature's predictive usefulness depends on which other features are present!

That is, a whole can be more (or less!) than the sum of its parts: the usefulness of a set of features ain't 'additive' the way the weight of a set of books is. Here's a simple example: let's flip two fair coins. We want to predict whether they ended up the same or not. Knowing the value of just the first coin is totally useless to prediction! Same with the value of just the second coin. BUT the two coins together help us predict 100% correctly.° Likewise, in the bottom-left panel the width is *independent* of the class label but not *conditionally-independent*-given-brightness. It's this that explains why we can't just compute the correlation of each feature with the predictand and then call it day.

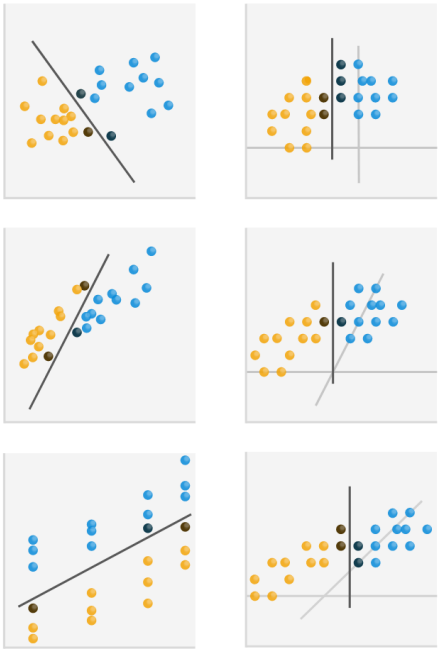


Figure 15: **Relations between feature statistics and optimal weights.** Each panel shows a different 2D binary classification task and a maximum-margin hypothesis. We shade margin-achieving points. To save ink we refer to the vertical and horizontal features as **brightness** and **width**; but you should be more imaginative. **Left:** *positive weights are consistent with positive, negative, or zero correlation!* **Right:** *presenting the same information in different coordinates (here, all 2D) alters predictions!*
Food For Thought: Think of classification tasks (and feature-pairs) that could plausibly give rise to the data depicted in each panel.

← Food For Thought: Explain the preceding intuition in terms of the L2 regularizer.

← Food For Thought: Find an analogue of the coins story where the whole is less than the sum of its parts, predictivity-wise.