Week 2: Regression w/multiple input variables

MULTIPLE LINEAR REGRESSION

probably the most widely used ML alg.

For house price prediction, consider multiple input variables as a vector $\vec{x}$: $\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$ x_1 = sq. ft., x_2 = bedrooms, ..., x_n = house age — with, n features

Model: $\boxed{f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b}$

where $\vec{w} = [w_1, \ldots, w_n]$ and $b \in \mathbb{R}$ are parameters.

> **Note**: in NumPy with arrays x, w you can just do np.dot(w, x) which may use hardware vectorization. Works w/ many ops!

↳ So in grad. descent you can write

```
f = np.dot(w, x) + b
w = w - 0.1 * d
```

} both vectorized!

np might do all this vectorized in a single step

Gradient descent for multiple lin. reg.

Params: $\vec{w} = [w_1, \ldots, w_n]$

Model: $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Cost fn: $J(\vec{w}, b)$

Grad Descent: Repeatedly:

for j = 1 to n: $w_j = w_j - \alpha \frac{1}{m} \sum_i^m (f(\vec{x}_i) - y_i) x_{ij}$

and: $b = b - \alpha \frac{1}{m} \sum_i^m (f(x_i) - y_i)$

until convergence.

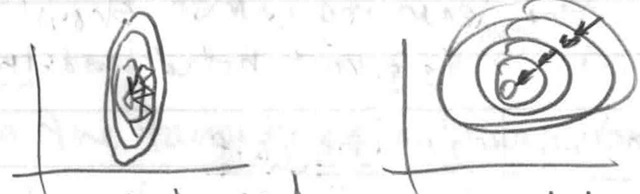
Alternative to linear regression: normal equation.
- this works only for lin. reg.! uses lin. algebra.
- Quite slow if # features is large
- May be used under the hood by lin. alg. libraries.

GRADIENT DESCENT IN PRACTICE

Feature Scaling

- When the range of a feature is relatively small, the model will learn large weights.
  And vice versa.
- With features that have very different ranges, grad. descent may take a long time to converge: the path to global minimum is much more straightforward:

Almost no harm, and speeds up GD



unbalanced

rescaled

Approaches: (want $x_i \in [-1, 1]$ $\forall i$) (or so)
- Divide by maximum: $x_i = x_i / max_i$ $\forall i$
- Mean normalization: compute mean then div. by spread

must store the scaling params for running in prod!!

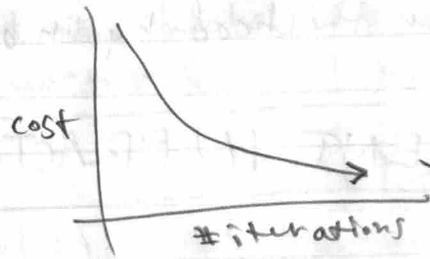
$$x_i = \frac{x_i - \mu_i}{max_i - min_i} \quad \forall i$$

- Z-score normalization: compute μ and σ (std dev.), then:

$$x_i = \frac{x_i - \mu_i}{\sigma_i} \quad \forall i$$

Checking for convergence

- plot # iterations vs. cost fn. – learning curve



you want to see this decreasing quickly and then leveling out

- Difficult to know apriori how many iterations needed – plot learning curve and tune!

- Automatic convergence test uses ε small to stop GD – but hard to pick it, and the learning curve shows other issues. (e.g. cost not decreasing monotonically)

Choosing learning rate

- Maybe learning curve not monotonically decreasing – choose larger/smaller α
- Check for bugs in the code

Tip for finding bugs: set α very small and ensure it's still monotonically decreasing! (Note: not for prod!)

Tip for picking α: Choose 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, etc.

→ Find too large, too small, then pick largest of reasonable.

FEATURE ENGINEERING

Consider features (width, height). Maybe the synthetic feature area = width × height is a better predictor!

↳ Requires insights into application

This enables Polynomial Regression!

e.g. $f_{w,b}(x) = wx + b \rightarrow w_1 x + \underbrace{w_2 x^2} + b$ etc.

note that feature scaling increasingly impt!

What features? See Course 2.

Can try w/ x^2, x^3, etc. and let GD learn the correct weights.

→ The linear in linear regression refers to expressing the target as a linear combination of features, not that the features themselves must be somehow linear.