INTRODUCTION TO DATA SCIENCE

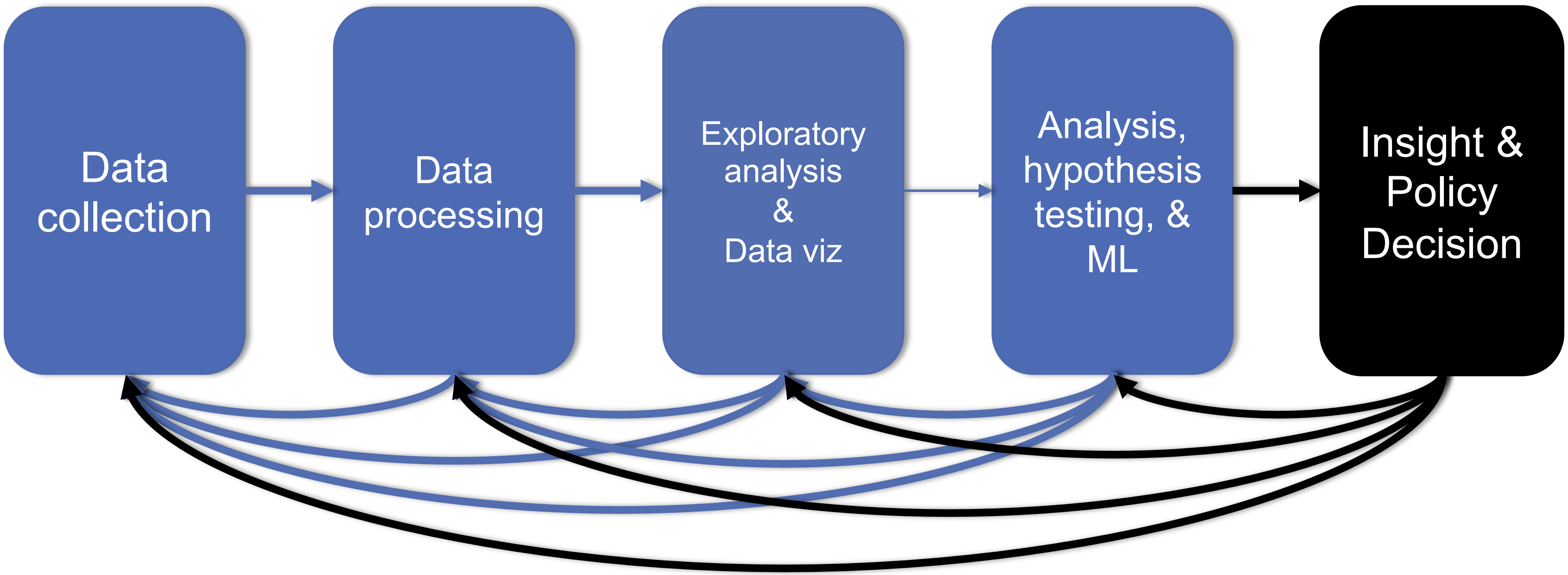
JOHN P DICKERSON

Lecture #16 – 10/17/2019
Lecture #19 – 10/29/2019

CMSC320
Tuesdays & Thursdays
5:00pm – 6:15pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

TODAY'S LECTURE

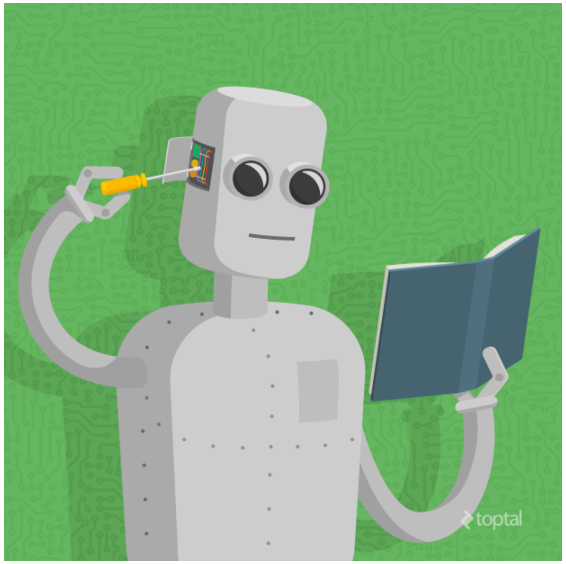


TODAY'S LECTURE

Introduction to machine learning

- How did we actually come up with that linear model from last class?
- Basic setup and terminology; linear regression & classification

Thanks to: Zico Kolter (CMU) & David Kauchak (Pomona)



First GIS result for "machine learning"

RECALL: EXPLICIT EXAMPLE OF STUFF FROM NLP CLASS

Score ψ of an instance *x* and class *y* is the sum of the weights for the features in that class:

$$\boldsymbol{\psi}_{\mathbf{x}y} = \Sigma\, \theta_n\, f_n(\mathbf{x}, y)$$

$$= \boldsymbol{\theta}^{\mathrm{T}}\, \mathbf{f}(\mathbf{x}, y)$$

Let's compute $\boldsymbol{\psi}_{\mathbf{x1}, y=hates\_cats}$ …

- $\boldsymbol{\psi}_{\mathbf{x1}, y=hates\_cats} = \boldsymbol{\theta}^{\mathrm{T}}\, \mathbf{f}(\mathbf{x}_1, y = \text{hates\_cats} = 0)$
- = 0*1 + -1*1 + 1*0 + -0.1*1 + 0*0 + 1*0 + -1*0 + 0.5*0 + 1*1
- = -1 - 0.1 + 1 = **-0.1**

$\boldsymbol{\theta}^{\mathrm{T}}$ =

0	-1	1	-0.1	0	1	-1	0.5	1

●

1	I	*hates_cats*
1	like	
0	hate	
1	cats	
0	I	*likes_cats*
0	like	
0	hate	
0	cats	
1	–	(1)

$\mathrm{f}(\mathbf{x}_1, y = 0)$

RECALL: EXPLICIT EXAMPLE OF STUFF FROM NLP CLASS

Saving the boring stuff:

- $\boldsymbol{\psi}_{\mathbf{x1},\,y=hates\_cats} = -0.1$; $\boldsymbol{\psi}_{\mathbf{x1},\,y=likes\_cats} = +2.5$
- $\boldsymbol{\psi}_{\mathbf{x2},\,y=hates\_cats} = +1.9$; $\boldsymbol{\psi}_{\mathbf{x2},\,y=likes\_cats} = +0.5$

Document 1: I like cats

Document 2: I hate cats

We want to predict the class of each document:

$$\hat{y} = \arg\max_{y} \theta^{\mathsf{T}} \mathbf{f}(\mathbf{x}, y)$$

Document 1: argmax{ $\psi_{\mathbf{x1},\,y=hates\_cats}$, $\psi_{\mathbf{x1},\,y=likes\_cats}$ **}   ????????**

Document 2: argmax{ $\psi_{\mathbf{x2},\,y=hates\_cats}$, $\psi_{\mathbf{x2},\,y=likes\_cats}$ **}   ????????**

MACHINE LEARNING

We used a linear model to classify input documents

The model parameters *θ* were given to us a priori

- (John created them by hand.)
- Typically, we cannot specify a model by hand.

Supervised machine learning provides a way to automatically infer the predictive model from labeled data.

Training Data		ML Algorithm		Predictions
$(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$ $(x^{(3)}, y^{(3)})$ …	➡	Hypothesis function $y^{(i)} = h(x^{(i)})$	➡	New example x $y = h(x)$

TERMINOLOGY

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \ldots, m$

	I	like	hate	cats
$x^{(1)T}$ =	1	1	0	1
$x^{(2)T}$ =	1	0	1	1

Outputs: $y^{(i)} \in \mathcal{Y}, i = 1, \ldots, m$

$y^{(i)} \in \{0, 1\}$ = { hates_cats, likes_cats }

Model parameters: $\theta \in \mathbb{R}^n$

θ^T =									
	0	-1	1	-0.1	0	1	-1	0.5	1

TERMINOLOGY

Hypothesis function: $h_\theta\colon \mathbb{R}^n \to \mathcal{Y}$

E.g., linear classifiers predict outputs using:

$$h_\theta(x) = \theta^T x = \sum_{j=1}^{n} \theta_j \cdot x_j$$

Loss function: $\ell\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$

- Measures difference between a prediction and the true output
- E.g., squared loss: $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- E.g., hinge loss: $\ell(y) = \max(0, 1 - t \cdot y)$

Output t = {-1,+1} based on -1 or +1 class label

Classifier score y

THE CANONICAL MACHINE LEARNING PROBLEM

At the end of the day, we want to learn a hypothesis function that predicts the actual outputs well.

And over all your training data*

Choose the parameterization that minimizes loss!

$$\text{minimize}_\theta \quad \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

Over all possible parameterizations

Given an hypothesis function and loss function

*Not actually what we want – want it over the world of inputs – will discuss later …

HOW DO I MACHINE LEARN?

1. **What is the hypothesis function?**
   - Domain knowledge and EDA can help here.
2. **What is the loss function?**
   - We've discussed two already: squared and absolute.
3. **How do we solve the optimization problem?**
   - (We'll cover gradient descent and stochastic gradient descent in class, but if you are interested, take CMSC422!)

First GIS result for "optimization"

ASIDE: LOSS FUNCTIONS

QUICK ASIDE ABOUT LOSS FUNCTIONS

Say we're back to classifying documents into:

- hates_cats, translated to label y = -1
- likes_cats, translated to label y = +1

We want some parameter vector $\boldsymbol{\theta}$ such that:

- $\boldsymbol{\psi}_{\mathbf{x}y} > 0$ if the feature vector x is of class likes_cat; (y = +1)
- $\boldsymbol{\psi}_{\mathbf{x}y} < 0$ if x's label is y = -1

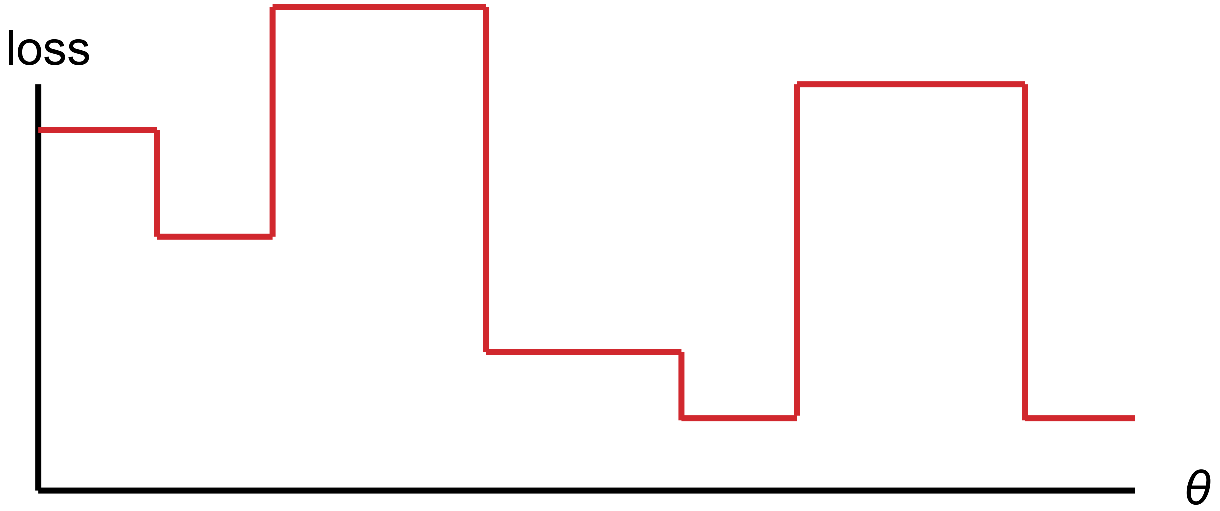
We want a hyperplane that separates positive examples from negative examples.

Why not use 0/1 loss; that is, the number of wrong answers?

$$\arg\min_{\theta} \sum_{i=1}^{n} \mathbf{1}\left[y^{(i)} \cdot \langle \theta, x^{(i)} \rangle \leq 0\right]$$

MINIMIZING 0/1 LOSS IN A SINGLE DIMENSION

$$\sum_{i=1}^{n} \mathbf{1}\left[y^{(i)} \cdot \langle \theta, x^{(i)} \rangle \leq 0\right]$$



Each time we change θ such that the example is right (wrong) the loss will increase (decrease)

MINIMIZING 0/1 LOSS OVER ALL Θ

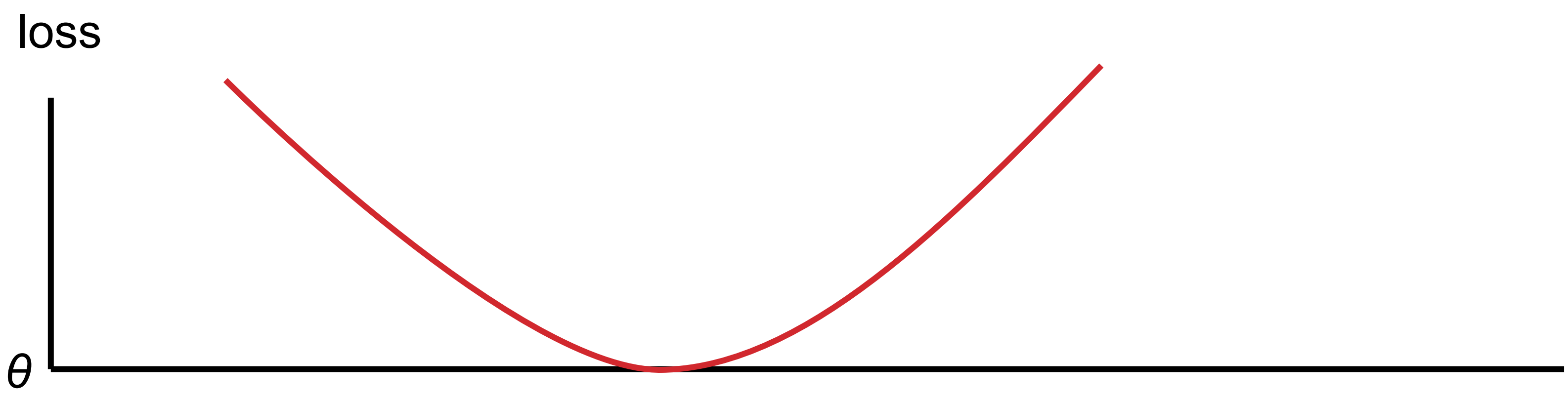
$$\arg\min_{\theta} \sum_{i=1}^{n} \mathbf{1}\left[y^{(i)} \cdot \langle \theta, x^{(i)} \rangle \leq 0\right]$$

This is NP-hard.

- Small changes in any *θ* can have large changes in the loss (the change isn't continuous)
- There can be many local minima
- At any give point, we don't have much information to direct us towards any minima

Maybe we should consider other loss functions.

DESIRABLE PROPERTIES



What are some desirable properties of a loss function????????

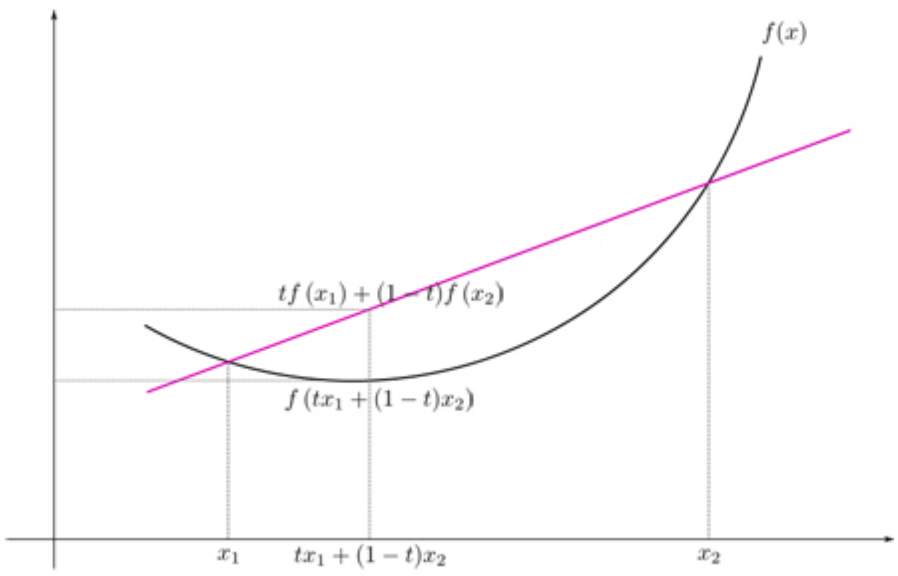
- **Continuous so we get a local indication of the direction of minimization**
- **Only one (i.e., global) minimum**

CONVEX FUNCTIONS

“A function is convex if the line segment between any two points on its graph lies above it.”

Formally, given function *f* and two points x, y:

$$f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}) \quad \forall \lambda \in [0,1]$$



SURROGATE LOSS FUNCTIONS

For many applications, we really would like to minimize the 0/1 loss

A surrogate loss function is a loss function that provides an upper bound on the actual loss function (in this case, 0/1)

We'd like to identify convex surrogate loss functions to make them easier to minimize

Key to a loss function is how it scores the difference between the actual label y and the predicted label y'

SURROGATE LOSS FUNCTIONS

0/1 loss: $\ell(\hat{y}, y) = \mathbf{1}\left[y\hat{y} \leq 0\right]$

Any ideas for surrogate loss functions ??????????
Want: a function that is continuous and convex and upper bounds the 0/1 loss.

- **Hinge:** $\ell(\hat{y}, y) = \max(0, 1 - y\hat{y})$
- **Exponential:** $\ell(\hat{y}, y) = e^{-y\hat{y}}$
- **Squared:** $\ell(\hat{y}, y) = (y - \hat{y})^2$

What do each of these penalize?????????

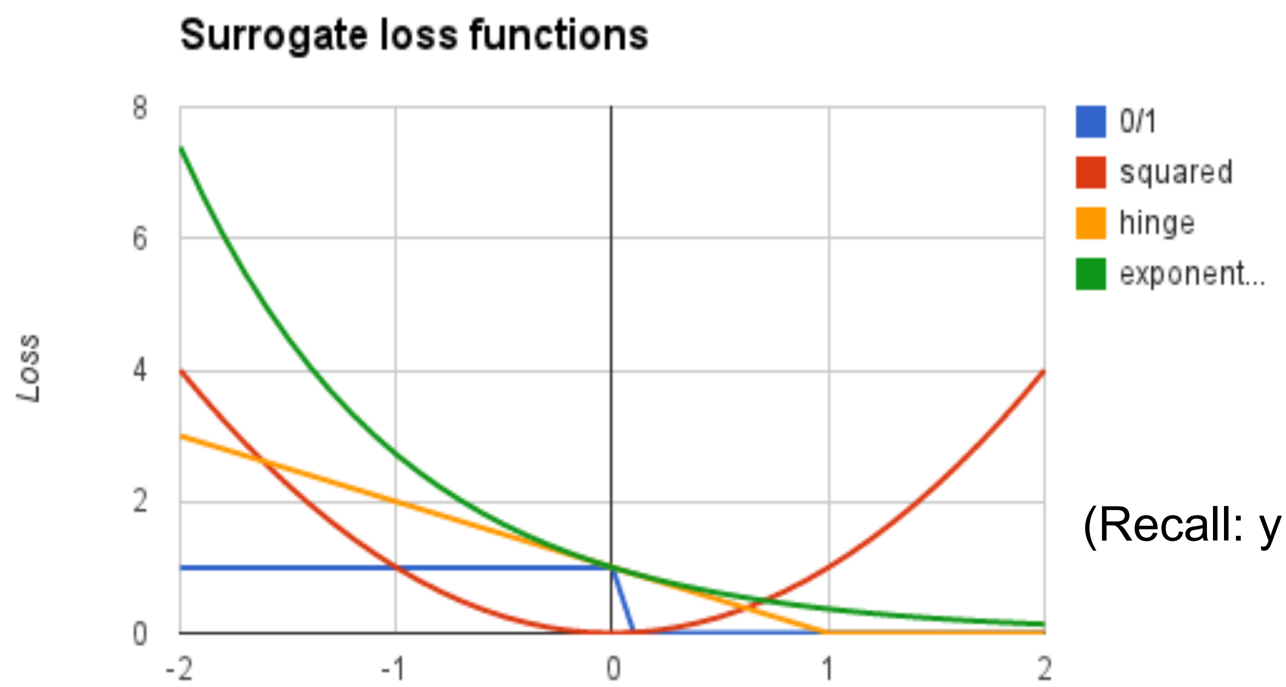
SURROGATE LOSS FUNCTIONS

0/1 loss: $$\ell(\hat{y}, y) = \mathbf{1}\left[y\hat{y} \leq 0\right]$$

Hinge: $$\ell(\hat{y}, y) = \max(0, 1 - y\hat{y})$$

Exponential: $$\ell(\hat{y}, y) = e^{-y\hat{y}}$$

Squared loss: $$\ell(\hat{y}, y) = (y - \hat{y})^2$$



(Recall: y in {-1, +1})

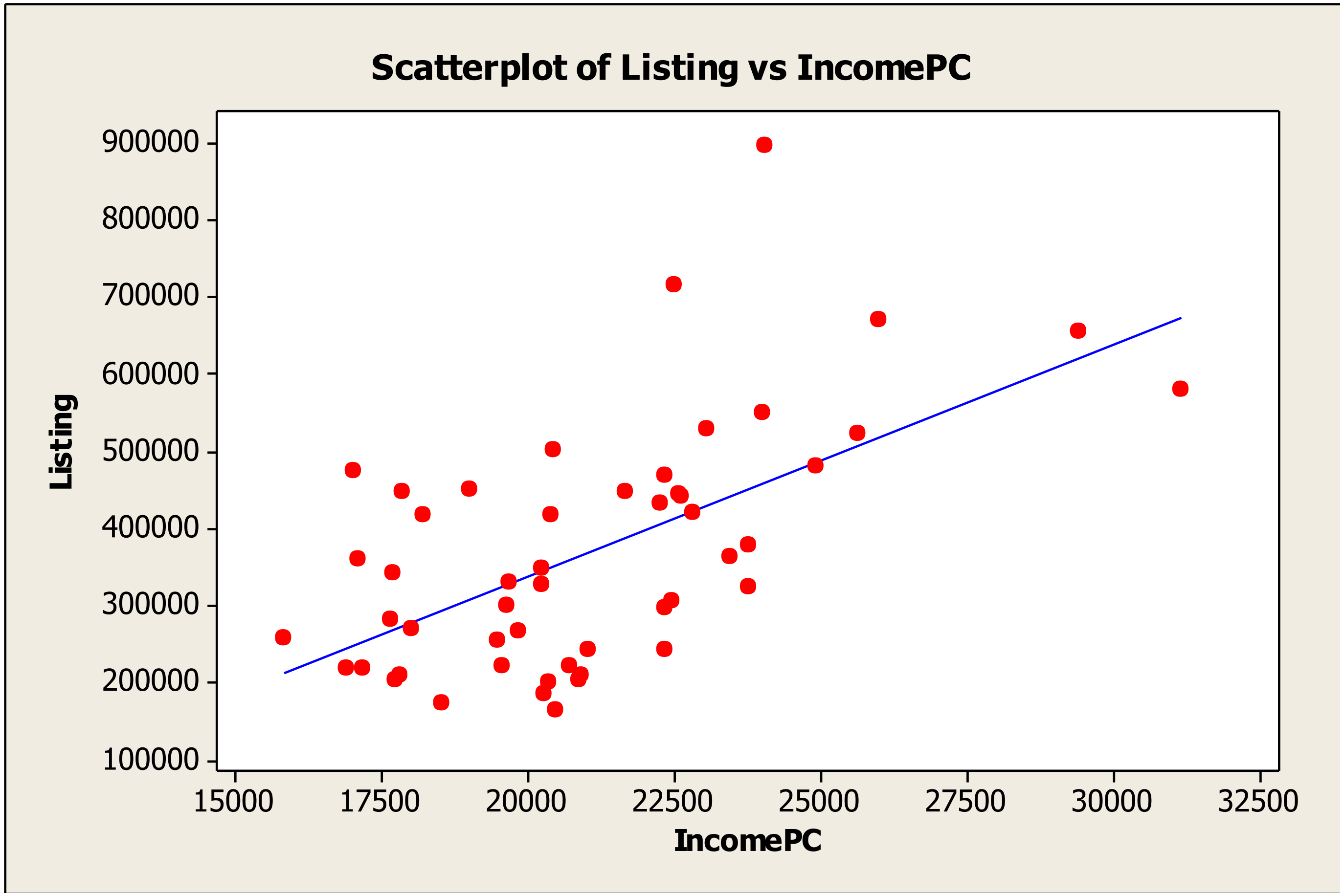
SOME ML ALGORITHMS

Name	Hypothesis Function	Loss Function	Optimization Approach
Least squares	Linear	Squared	Analytical or GD
Linear regression	Linear	Squared	Analytical or GD
Support Vector Machine (SVM)	Linear, Kernel	Hinge	Analytical or GD
Perceptron	Linear	Perceptron criterion (~Hinge)	Perceptron algorithm, others
Neural Networks	Composed nonlinear	Squared, Hinge	SGD
Decision Trees	Hierarchical halfplanes	Many	Greedy
Naïve Bayes	Linear	Joint probability	#SAT

Follow the white rabbit: https://en.wikipedia.org/wiki/List_of_machine_learning_concepts

EXAMPLE

RECALL: LINEAR REGRESSION



LINEAR REGRESSION AS MACHINE LEARNING

Let's consider linear regression that minimizes the sum of squared error, i.e., least squares …

1. **Hypothesis function: ????????**
   - Linear hypothesis function $h_\theta(x) = \theta^T x$

2. **Loss function: ????????**
   - Squared error loss $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

3. **Optimization problem: ????????**

$$\text{minimize}_\theta \quad \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$$

LINEAR REGRESSION AS MACHINE LEARNING

Rewrite inputs:

Each row is a feature vector paired with a label for a single input

m labeled inputs

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix} \in \mathbb{R}^{m \times n}, \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

n features

Rewrite optimization problem:

$$\text{minimize}_\theta \ \frac{1}{2}\|X\theta - y\|_2^2$$

*Recall: $\|x\|_2^2 = z^T z = \sum_i z_i^2$

GRADIENTS

In Lecture 11, we showed that the mean is the point that minimizes the residual sum of squares:

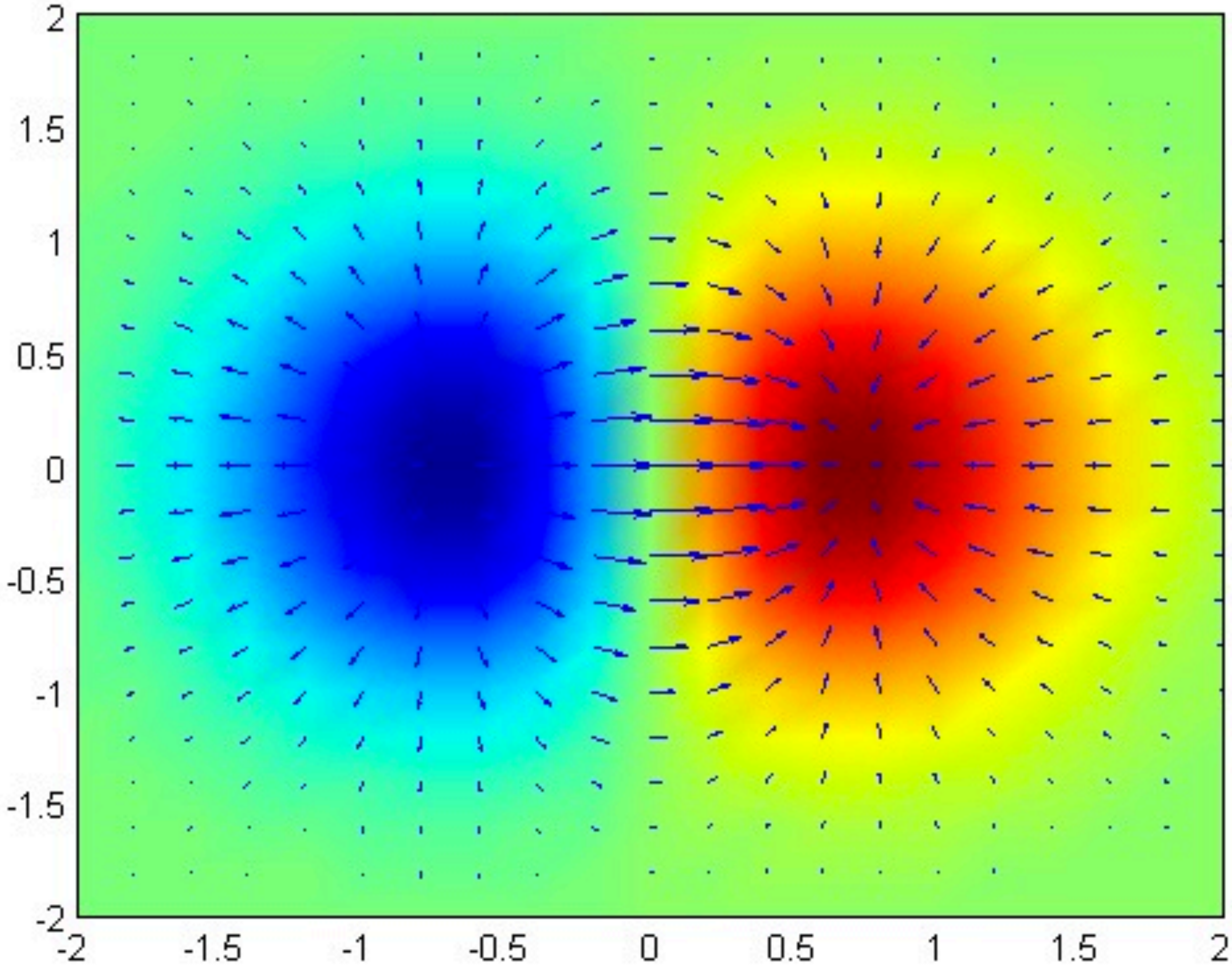
- Solved minimization by finding point where derivative is zero
- (Convex functions like RSS → single global minimum.)

The gradient is the multivariate generalization of a derivative.

For a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$, the gradient is a vector of all *n* partial derivatives:

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^n$$

GRADIENTS



Gradient of $f(x,y) = xe^{-(x^2 + y^2)}$

GRADIENTS

Minimizing a multivariate function involves finding a point where the gradient is zero:

$$\nabla_\theta f(\theta) = 0 \text{ (the vector of zeros)}$$

Points where the gradient is zero are local minima

- If the function is convex, also a global minimum

Let's solve the least squares problem!

We'll use the multivariate generalizations of some concepts from MATH141/142 …

- Chain rule: $\nabla_\theta f(X\theta) = X^T \nabla_{X\theta} f(X\theta)$
- Gradient of squared ℓ^2 norm: $\nabla_\theta \|\theta - z\|_2^2 = 2(\theta - z)$

LEAST SQUARES

Recall the least squares optimization problem:

$$\text{minimize}_\theta \ \frac{1}{2}\|X\theta - y\|_2^2$$

What is the gradient of the optimization objective ????????

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 =$$

Chain rule:
$\nabla_\theta f(X\theta) = X^T \nabla_{X\theta} f(X\theta)$

$$X^T \nabla_{X\theta} \frac{1}{2}\|X\theta - y\|_2^2 =$$

Gradient of norm:
$\nabla_\theta \|\theta - z\|_2^2 = 2(\theta - z)$

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 = X^T(X\theta - y)$$

LEAST SQUARES

Recall: points where the gradient equals zero are minima.

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 = X^T(X\theta - y)$$

So where do we go from here?????????

Solve for model parameters θ

$$X^T(X\theta - y) = 0$$

$$X^T X\theta - X^T y = 0 \Rightarrow X^T X\theta = X^T y$$

$$(X^T X)^{-1} X^T X\theta = (X^T X)^{-1} X^T y$$

$$\boxed{\theta = (X^T X)^{-1} X^T y}$$

ML IN PYTHON

Python has tons of hooks into a variety of machine learning libraries. (Part of why this course is taught in Python!)

Scikit-learn is the most well-known library:

- Classification (SVN, K-NN, Random Forests, …)
- Regression (SVR, Ridge, Lasso, …)
- Clustering (k-Means, spectral, mean-shift, …)
- Dimensionality reduction (PCA, matrix factorization, …)
- Model selection (grid search, cross validation, …)
- Preprocessing (cleaning, EDA, …)

Built on the NumPy stack; plays well with Matplotlib.

LEAST SQUARES IN PYTHON

$$\theta = (X^T X)^{-1} X^T y$$

You don't need Scikit-learn for OLS …

```
# Analytic solution to OLS using Numpy
params = np.linalg.solve(X.T.dot(X), X.T.dot(y))
```

But let's say you did want to use it.

```
from sklearn import linear_model

X = [[0,0], [1,1], [2,2]]
Y = [0, 1, 2]

# Solve OLS using Scikit-Learn
reg = linear_model.LinearRegression()
reg.fit(X, Y)
reg.coef_
```

```
array([ 0.5, 0.5])
```

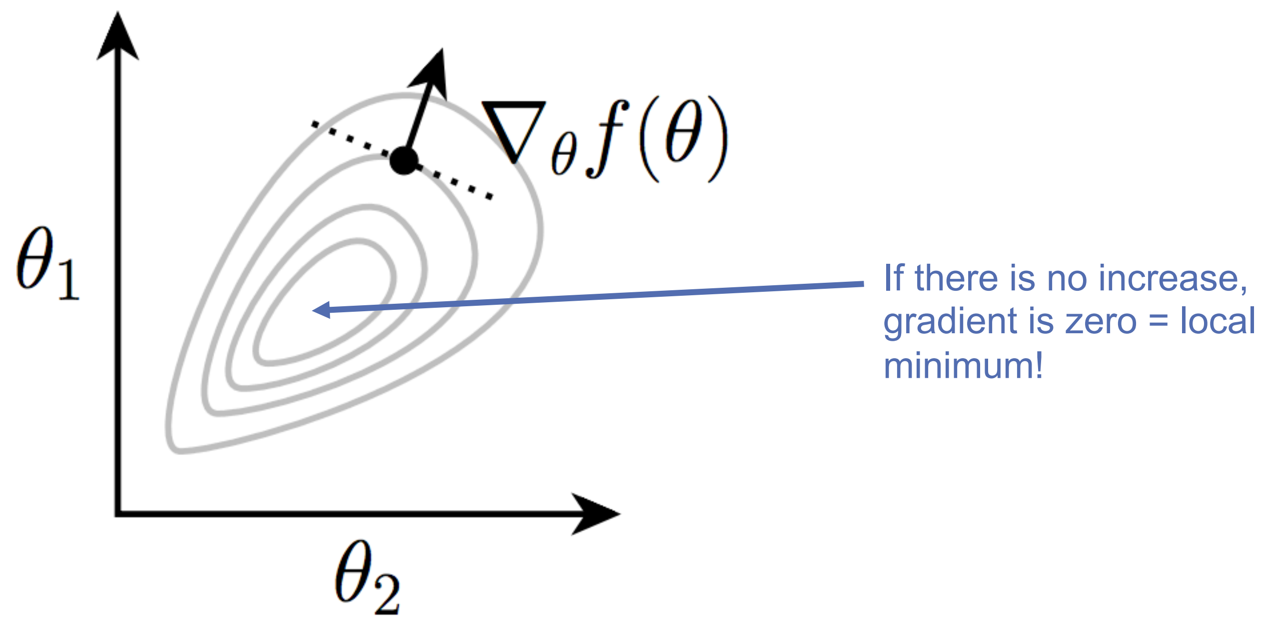
NEXT, OR NEXT CLASS:

(STOCHASTIC) GRADIENT DESCENT

TODAY: GRADIENT DESCENT

We used the gradient as a condition for optimality

It also gives the local direction of steepest increase for a function:



Intuitive idea: take small steps against the gradient.

Image from Zico Kolter

GRADIENT DESCENT

Algorithm for any* hypothesis function $h_\theta : \mathbb{R}^n \rightarrow \mathcal{Y}$, loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, step size α :

Initialize the parameter vector:

- $\theta \leftarrow 0$

Repeat until satisfied (e.g., exact or approximate convergence):

- **Compute gradient:** $g \leftarrow \sum_{i=1}^{m} \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$
- **Update parameters:** $\theta \leftarrow \theta - \alpha \cdot g$

*must be reasonably well behaved

GRADIENT DESCENT

Step-size (\alpha) is an important parameter

- Too large → might oscillate around the minima
- Too small → can take a long time to converge

If there are no local minima, then the algorithm eventually converges to the optimal solution

Very widely used in Machine Learning

EXAMPLE

Function: f(x,y) = x^2 + $2y^2$

Gradient: ??????????

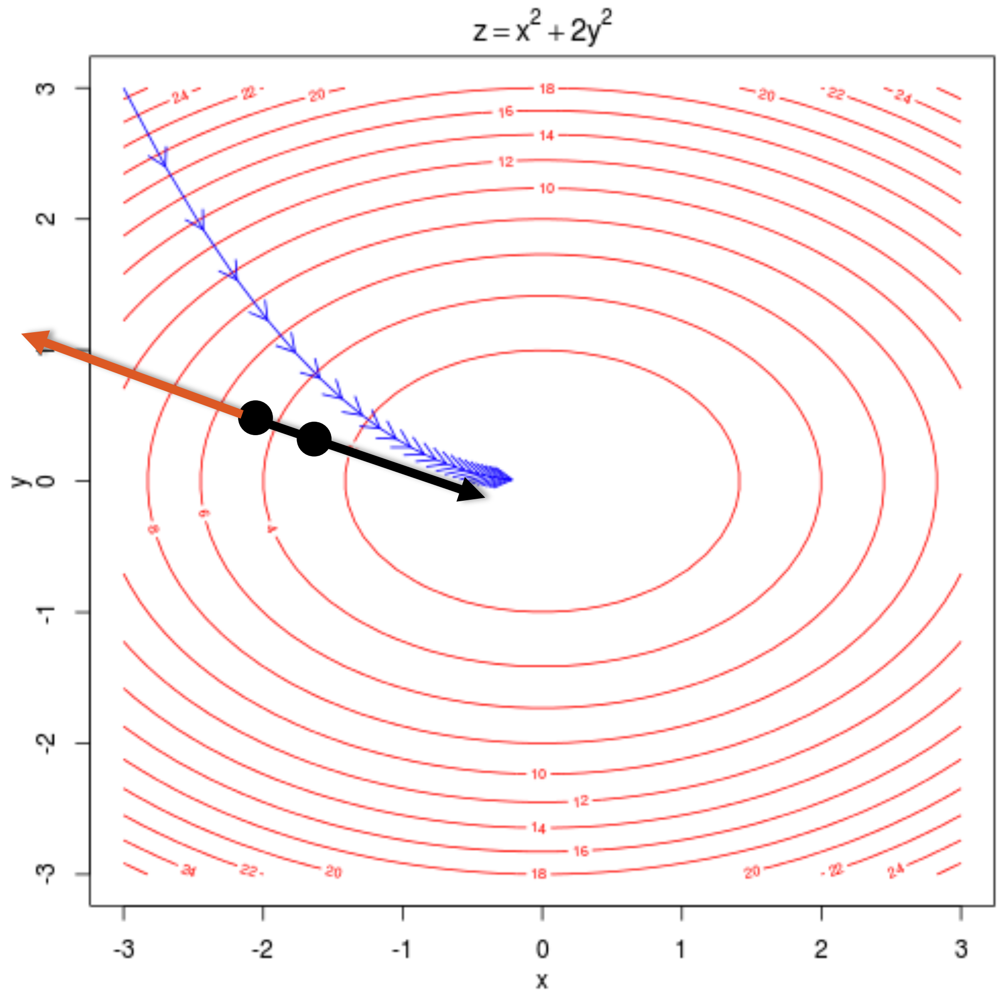
$$\nabla f(x,y) = \left[\begin{array}{c} 2x \\ 4y \end{array} \right]$$

Let's take a gradient step from (-2, +1/2):

$$\nabla f(-2,1) = \left[\begin{array}{c} 4 \\ 2 \end{array} \right]$$

Step in the direction (-4, -2), scaled by step size

Repeat until no movement



GRADIENT DESCENT FOR OLS

Algorithm for linear hypothesis function and squared error loss function (combined to $1/2\|X\theta - y\|_2^2$, like before):

Initialize the parameter vector:

- $\theta \leftarrow 0$

Repeat until satisfied:

- **Compute gradient:** $g \leftarrow X^T(X\theta - y)$
- **Update parameters:** $\theta \leftarrow \theta - \alpha \cdot g$

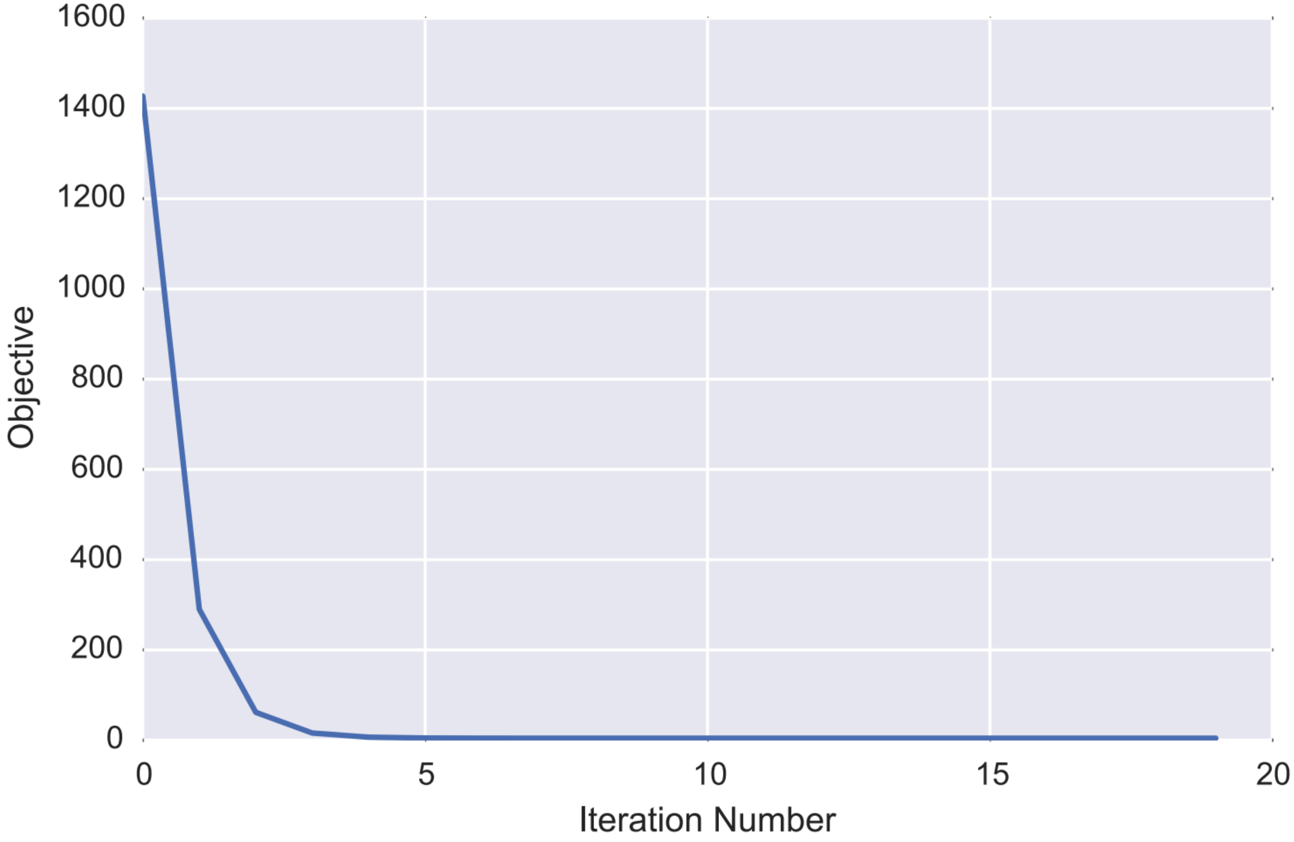
GRADIENT DESCENT IN PURE(-ISH) PYTHON

```
# Training data (X, y), T time steps, alpha step
def grad_descent(X, y, T, alpha):
    m, n = X.shape              # m = #examples, n = #features
    theta = np.zeros(n)         # initialize parameters
    f = np.zeros(T)             # track loss over time
```

```
    for i in range(T):
        # loss for current parameter vector theta
        f[i] = 0.5*np.linalg.norm(X.dot(theta) – y)**2
        # compute steepest ascent at f(theta)
        g = X.T.dot(X.dot(theta) – y)
        # step down the gradient
        theta = theta – alpha*g
    return theta, f
```

Implicitly using squared loss and linear hypothesis function above; drop in your favorite gradient for kicks!

PLOTTING LOSS OVER TIME



Why ????????

Image from Zico Kolter

ITERATIVE VS ANALYTIC SOLUTIONS

But we already had an analytic solution! What gives?

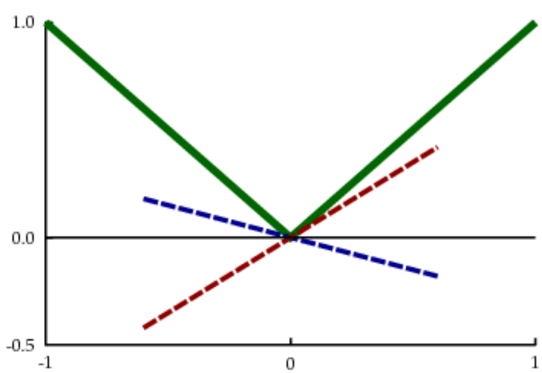
Recall: last class we discuss 0/1 loss, and using convex surrogate loss functions for tractability

One such function, the absolute error loss function, leads to:

$$\text{minimize}_\theta \sum_{i=1}^{m} \left| \theta^T x^{(i)} - y^{(i)} \right| \equiv \text{minimize}_\theta \; \| X\theta - y \|_1$$

Problems ????????

- Not differentiable! But subgradients?
- No closed form!
- So you must use iterative method



LEAST ABSOLUTE DEVIATIONS

Can solve this using gradient descent and the gradient:

$$\nabla_\theta \|X\theta - y\|_1 = X^T \operatorname{sign}(X\theta - y)$$

Simple to change in our Python code:

```
for i in range(T):
    # loss for current parameter vector theta
    f[i] = np.linalg.norm(X.dot(theta) – y, 1)
    # compute steepest ascent at f(theta)
    g = X.T.dot( np.sign(X.dot(theta) – y) )
    # step down the gradient
    theta = theta – alpha*g
return theta, f
```

BATCH VS STOCHASTIC GRADIENT DESCENT

Batch: Compute a single gradient (vector) for the entire dataset (as we did so far)

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

Incremental/Stochastic:

- Do one training sample at a time, i.e., update parameters for every sample separately
- Much faster in general, with more pathological cases

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

}

From: Andrew Ng, CS229 Lecture Notes