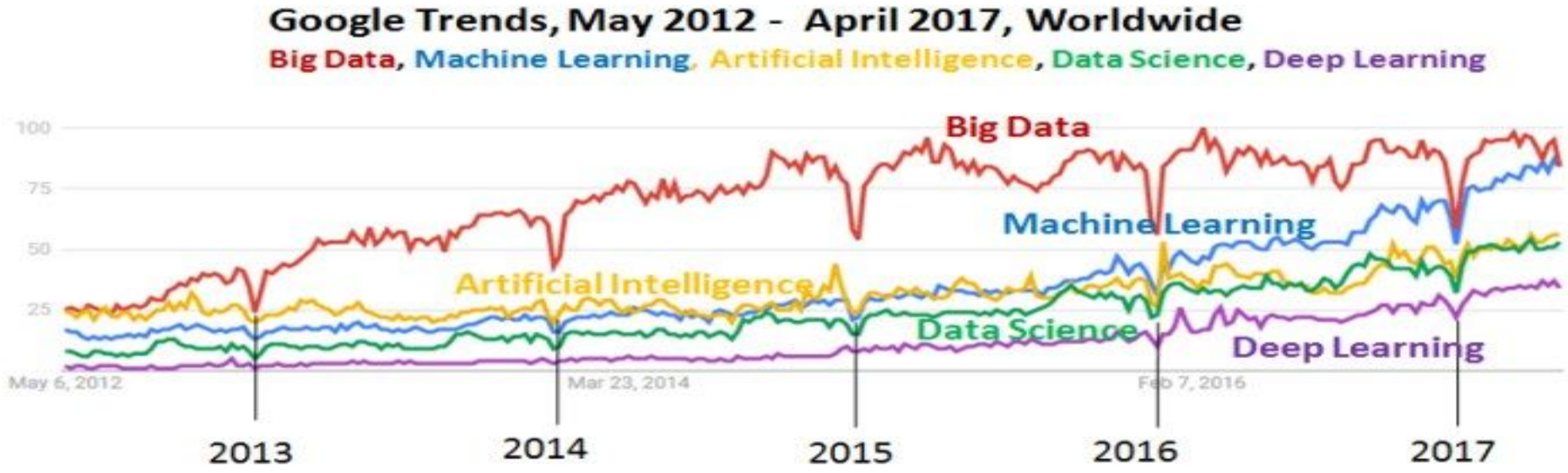
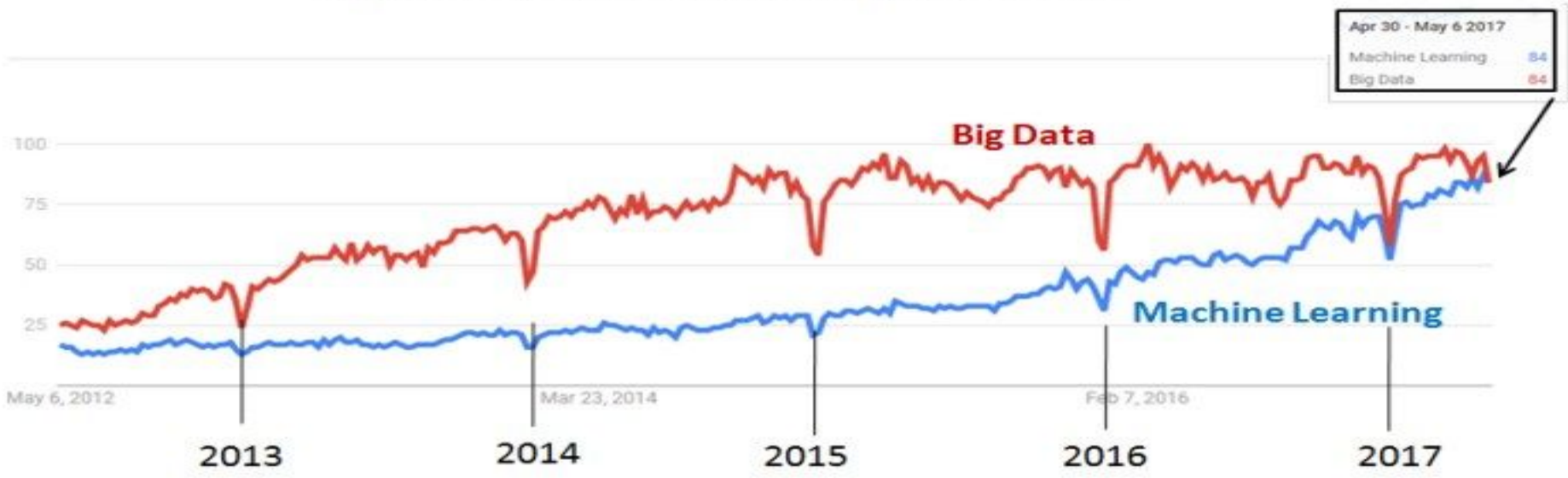
Intro to Data Analysis and Machine Learning

Xiaohui Xie, UCI

AI is everyone


Google Trends, May 2012 - April 2017, Worldwide
Big Data, Machine Learning, Artificial Intelligence, Data Science, Deep Learning
100
75
50
25
Big Data
Machine Learning
Artificial Intelligence
Data Science
Deep Learning
May 6, 2012
Mar 23, 2014
Feb 7, 2016
2013
2014
2015
2016
2017

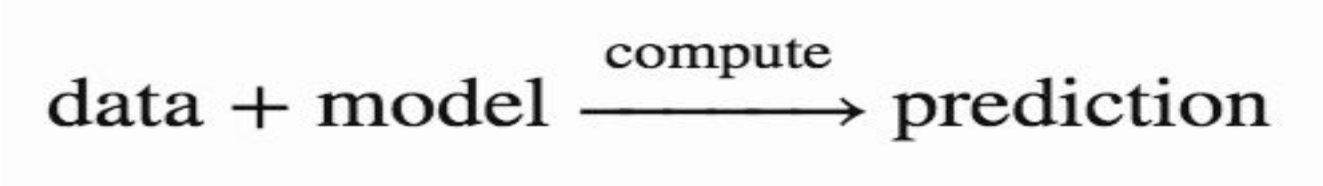

Google Trends, May 2012 - April 2017
Big Data vs Machine Learning search terms
Apr 30 - May 6 2017
Machine Learning 84
Big Data 84
Big Data
Machine Learning
100
75
50
25
May 6, 2012
Mar 23, 2014
Feb 7, 2016
2013
2014
2015
2016
2017

GARTNER'S FIVE-STEP HYPE CYCLE


Connected Home
Deep Learning
Virtual Assistants
IoT Platform
Machine Learning
Smart Robots
Autonomous Vehicles
Edge Computing
Nanotube Electronics
Augmented Data Discovery
Cognitive Computing
Blockchain
Smart Workspace
Commercial UAVs (Drones)
Conversational User Interfaces
Brain-Computer Interface
Cognitive Expert Advisors
Volumetric Displays
Quantum Computing
Digital Twin
Serverless PaaS
5G
Human Augmentation
Neuromorphic Hardware
Enterprise Taxonomy and Ontology Management
Deep Reinforcement Learning
Software-Defined Security
Virtual Reality
4D Printing
Artificial General Intelligence
Augmented Reality
Smart Dust
Plateau will be reached in:
less than 2 years
2 to 5 years
5 to 10 years
more than 10 years
As of July 2017
Expectations
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
Time

What is machine learning?

At its most basic level machine learning is a combination of



Three Key Components of ML

- **Prediction function f(x)**
  - a function which is used to make the predictions. It includes our beliefs about the regularities of the universe, our assumptions about how the world works.

- **Objective function** C(y, f(x))
  - a function which defines the cost of misprediction.

- **Learning algorithm**
  - Generate an optimal prediction function by optimizing the objective function based on training data

Examples of prediction functions

- Linear functions, Generalized linear functions (GL)
- Polynomials, or other basis functions
- Radial basis function (RBF), Kernels
- Decision trees, or ensemble of decision trees (Random Forest)
- Neural networks

Examples of objective functions

- Mean Square Errors (MSE)
- Cross-entropy loss
- Hinge Loss (margin maximization)
- Likelihood function, Pseudo-likelihood, Partial-likelihood, Maximum a posterior (MAP)
- Kullback-Leibler divergence (KL)

Examples of learning algorithms

- Gradient descent – batch gradient descent, stochastic gradient descent (SGD)
- Newton’s method, quasi-Newton, or other second order methods
- Global optimization methods (simulated annealing, greedy algorithms, genetic algorithms, etc)
- Expectation-Maximization, Expectation Propagation

Inference algorithms in probabilistic models

- Message Passing Algorithms (Max-Product, Sum-Product)
- Various Monte Carlo based methods:
  - MCMC (such as Gibbs sampling)
  - Importance sampling
- Variational Methods

ML Challenges

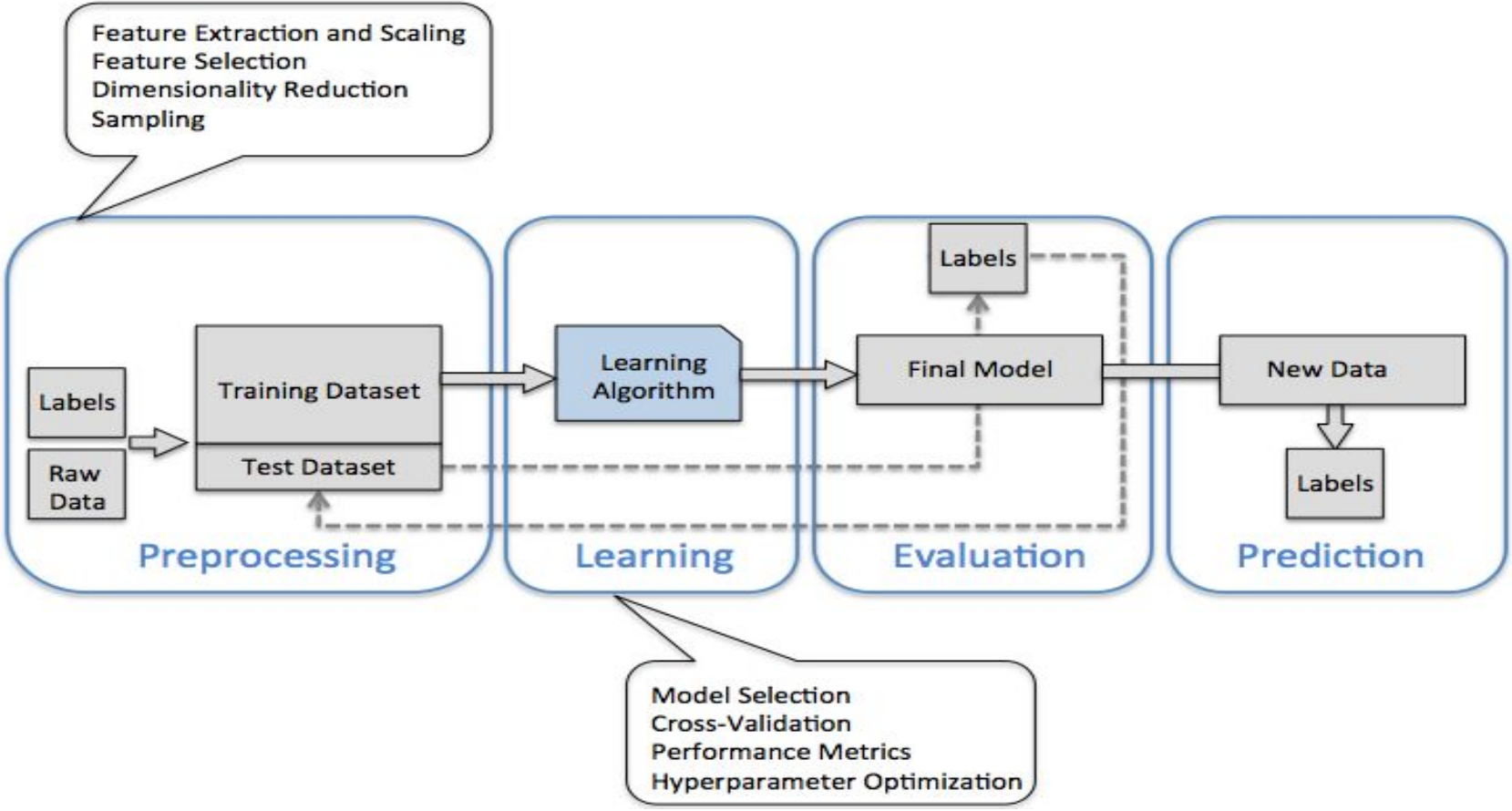
Uncertainty - uncertainty associated each of three key components

- Uncertainty in the **prediction function** arises from scarcity of training data
  - Mismatch between the set of prediction functions we choose and all possible prediction functions.

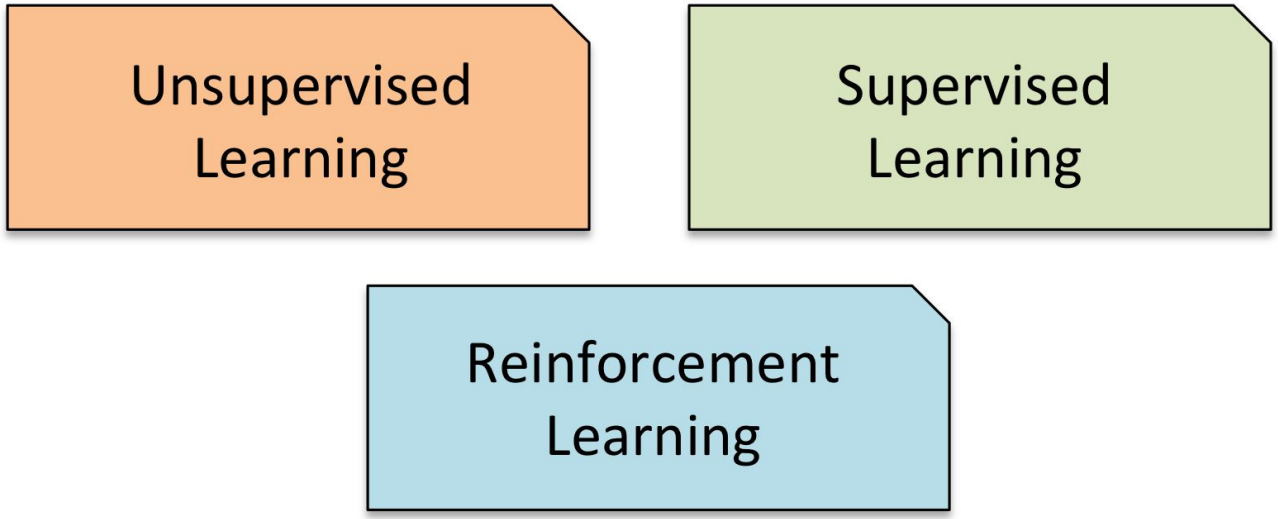
- There are also challenges around specification of the **objective function**

- **Learning and inference algorithms** in general have no guaranteed optimality.

Roadmap For Building Machine Learning



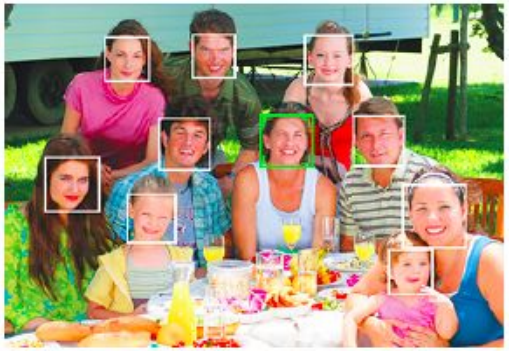
Three Types of Machine Learning



Types of prediction problems

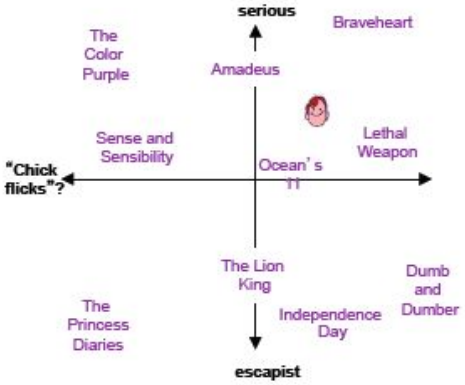
- Supervised learning
  - "Labeled" training data
  - Every example has a desired target value (a "best answer")
  - Reward prediction being close to target

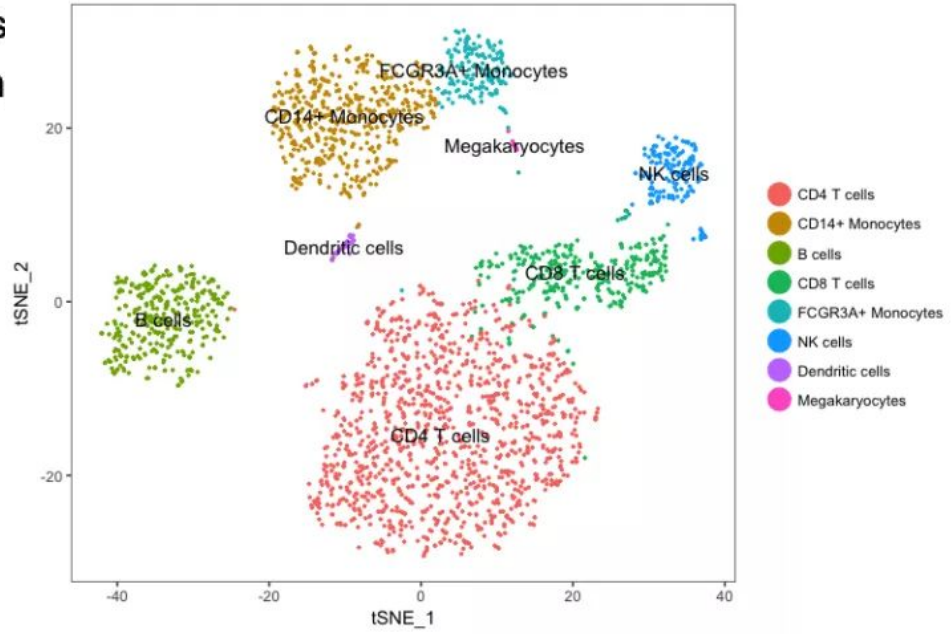
  - Classification: a discrete-valued prediction
  - Regression: a continuous-valued prediction



Types of prediction problems

- Supervised learning
- Unsupervised learning
  - No known target values
  - No targets = nothing to predict?
  - Reward “patterns
  - Often, data minin





Types of prediction problems

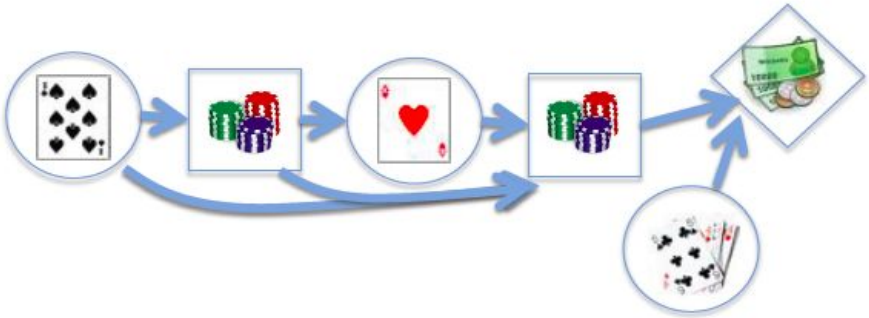
- Supervised learning
- Unsupervised learning
- Semi-supervised learning
  - Similar to supervised
  - some data have unknown target values

- Ex: medical data
  - Lots of patient data, few known outcomes
- Ex: image tagging
  - Lots of images on Flikr, but only some of them tagged

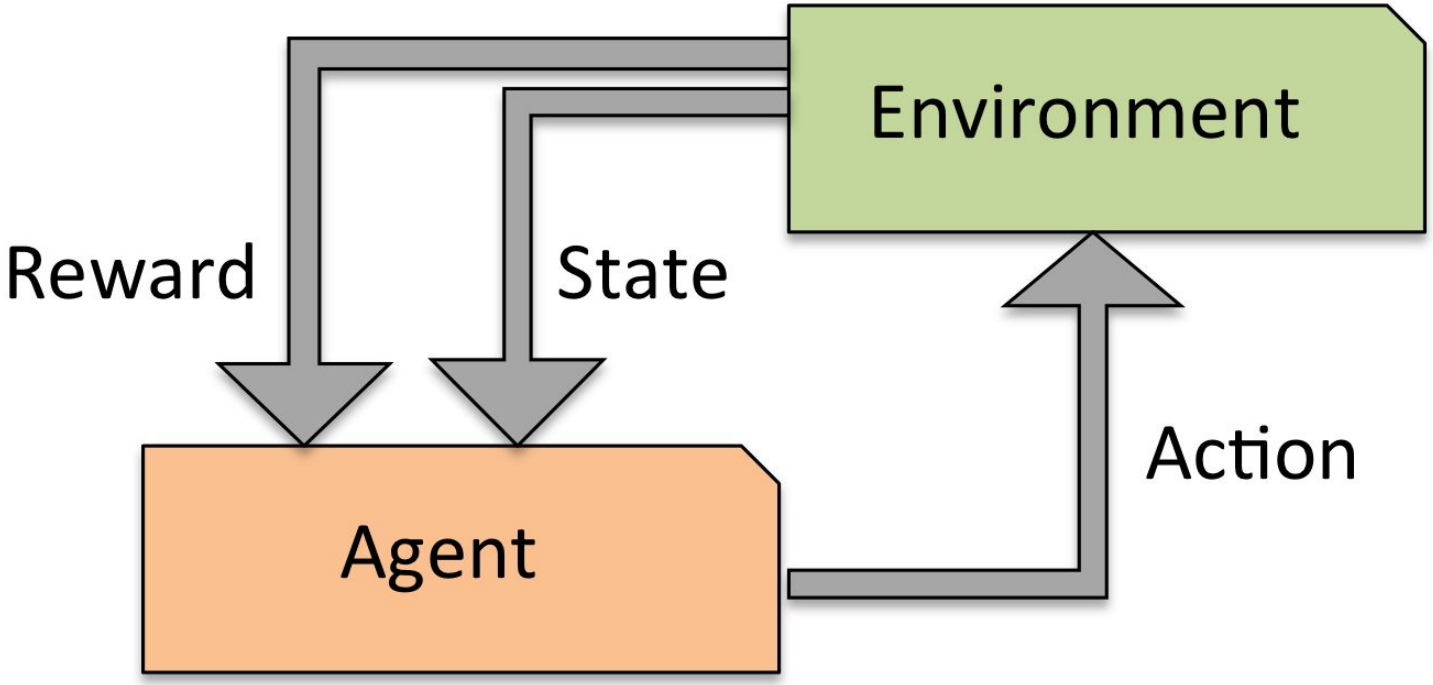
Types of prediction problems

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

- "Indirect" feedback on quality
  - No answers, just "better" or "worse"
  - Feedback may be delayed



Reinforcement Learning: Interacting with Environments



Data Exploration

- Machine learning is a data science
  - Look at the data; get a “feel” for what might work
- What types of data do we have?
  - Binary values? (spam; gender; …)
  - Categories? (home state; labels; …)
  - Integer values? (1..5 stars; age brackets; …)
  - (nearly) real values? (pixel intensity; prices; …)
- Are there missing data?
- “Shape” of the data? Outliers?

Scientific Software

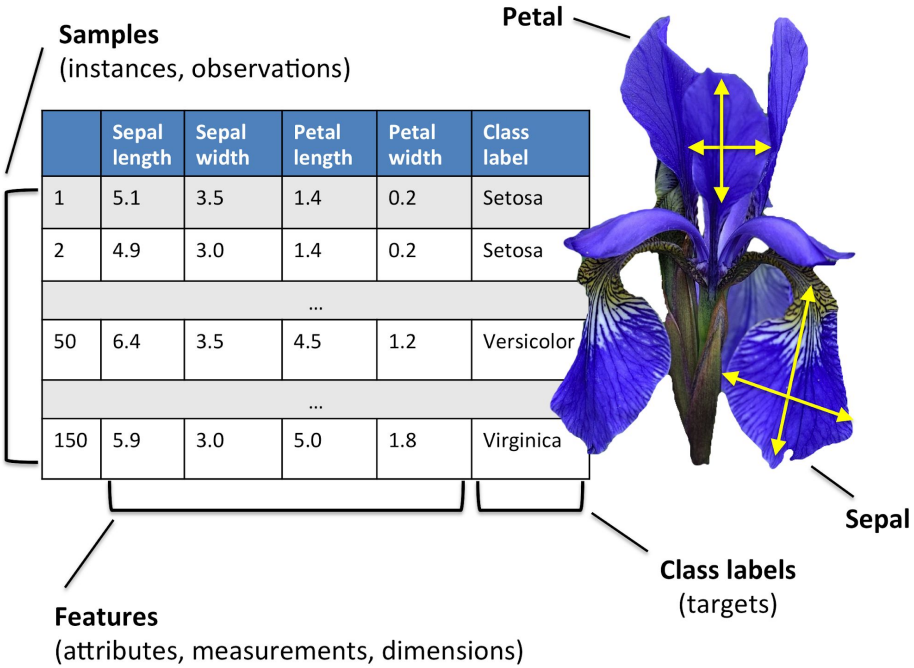
- Python
  - Numpy, MatPlotLib, SciPy…
- Matlab
  - Octave (free)
- R
  - Used mainly in statistics
- C++
  - For performance, not prototyping
- And other, more specialized languages for modeling…

Representing Data

Example: Fisher's "Iris" data
http://en.wikipedia.org/wiki/Iris_flower_data_set

- Three different types of iris
  - "Class", y
- Four "features", x1,…,x4
  - Length & width of sepals & petals
- 150 examples (data points)

Intro to Basic Terminology and Notations



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Representing the data

- Have m observations (data points)
  $\left\{ x^{(1)} \ldots, x^{(m)} \right\}$
- Each observation is a vector consisting of n features
  $x^{(j)} = [x_1^{(j)} x_2^{(j)} \ldots x_n^{(j)}]$
- Often, represent this as a “data matrix”

$$\underline{X} = \begin{bmatrix} x_1^{(1)} & \ldots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \ldots & x_n^{(m)} \end{bmatrix}$$

```
import numpy as np    # import numpy
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X = iris[:,0:4]       # load data and split into features, targets
Y = iris[:,4]
print X.shape         # 150 data points; 4 features each
   (150, 4)
```

Basic Statistics

Look at basic information about features

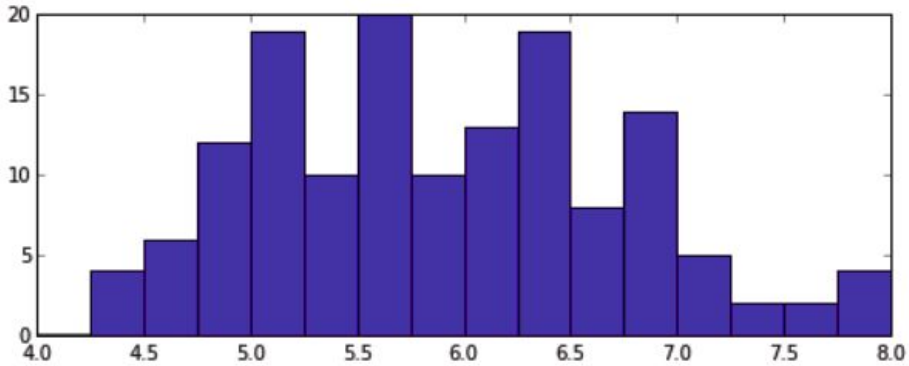
- Average value? (mean, median, etc.)
- “Spread”? (standard deviation, etc.)
- Maximum / Minimum values?

```
print np.mean(X, axis=0)          # compute mean of each feature
 [ 5.8433   3.0573   3.7580   1.1993 ]
print np.std(X, axis=0)           #compute standard deviation of each feature
 [ 0.8281   0.4359   1.7653   0.7622 ]
print np.max(X, axis=0)           # largest value per feature
 [ 7.9411   4.3632   6.8606   2.5236 ]
print np.min(X, axis=0)           # smallest value per feature
 [ 4.2985   1.9708   1.0331   0.0536 ]
```

Histogram

Count the data falling in each of K bins

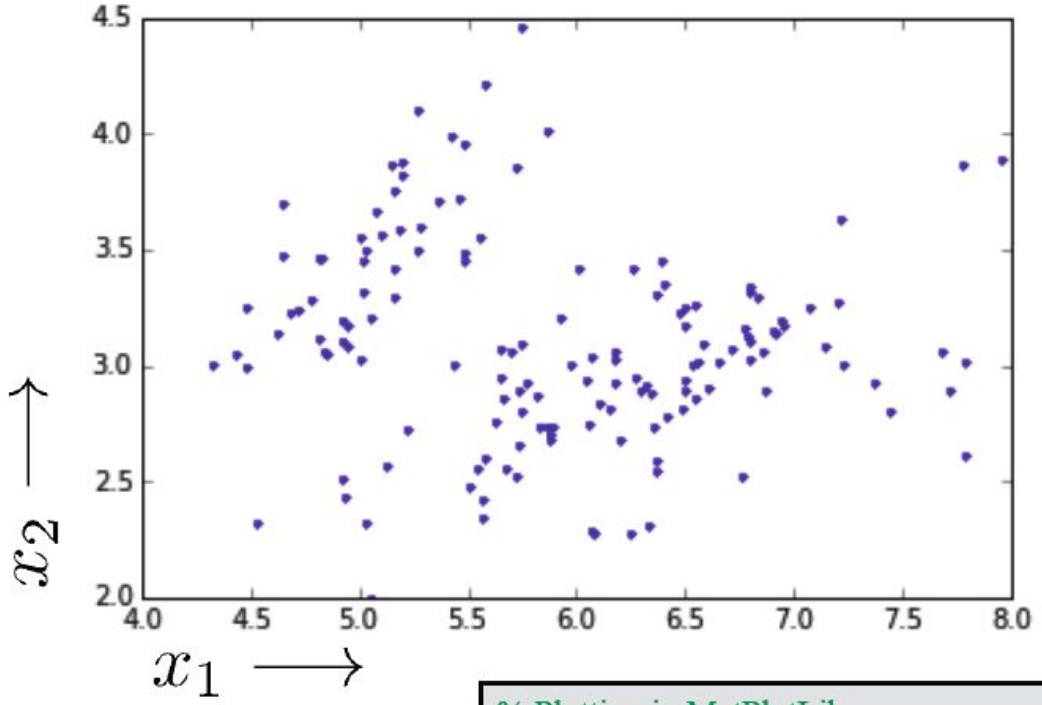
- "Summarize" data as a length-K vector of counts (& plot)
- Value of K determines "summarization"; depends on # of data
  - K too big: every data point falls in its own bin; just "memorizes"
  - K too small: all data in one or two bins; oversimplifies




```
% Histograms in MatPlotLib
import matplotlib.pyplot as plt
X1 = X[:,0]                    # extract first feature
Bins = np.linspace(4,8,17)     # use explicit bin locations
plt.hist( X1, bins=Bins )      # generate the plot
```

Scatterplots

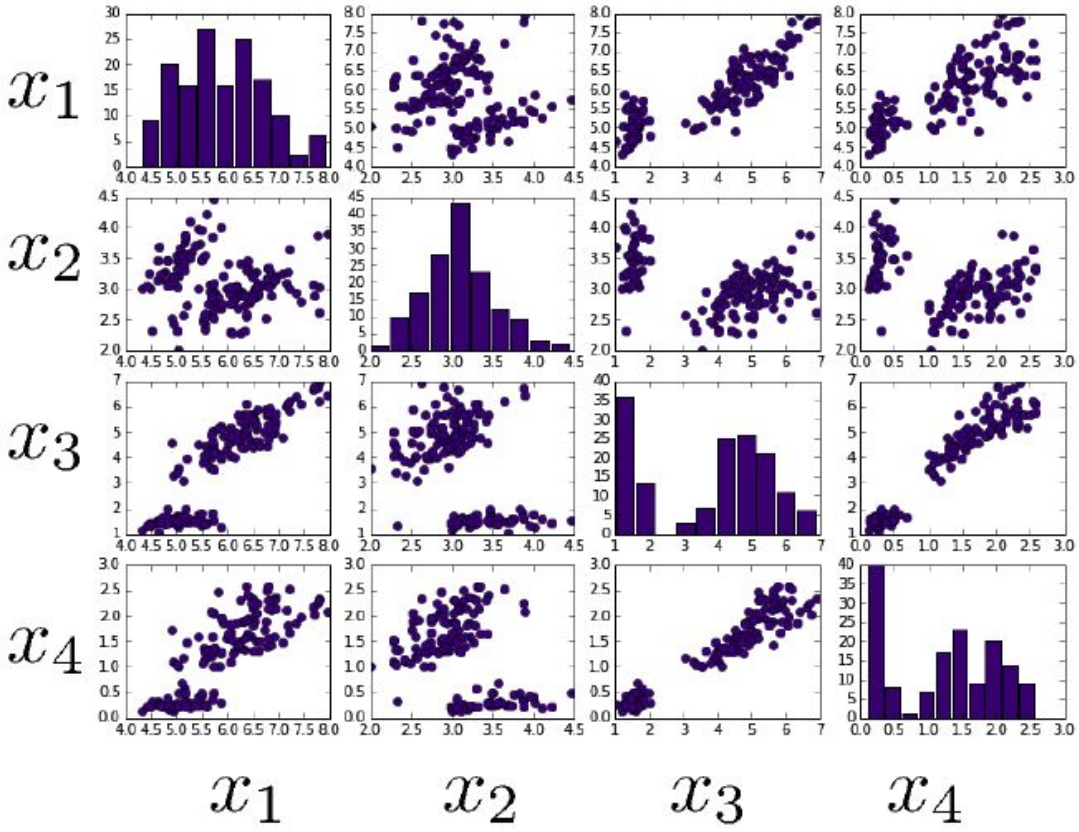
- Illustrate the relationship between two features




```
% Plotting in MatPlotLib
plt.plot(X[:,0], X[:,1], 'b.');   % plot data points as blue dots
```

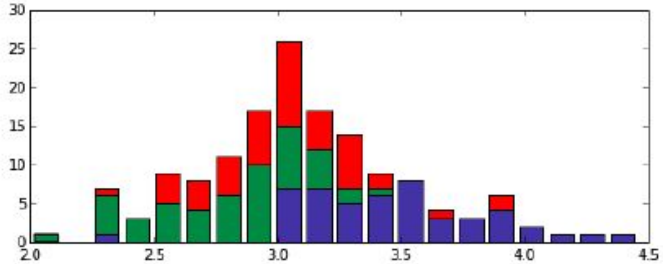
Scatterplots

For more than two features we can use a pair plot:

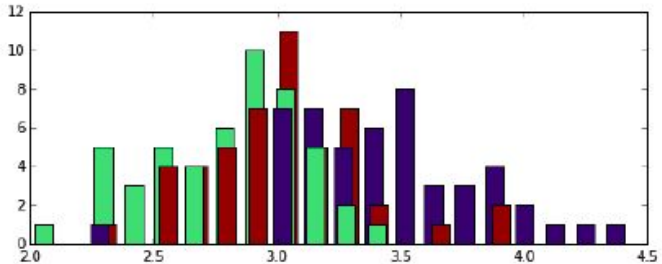


Supervised Learning and Targets

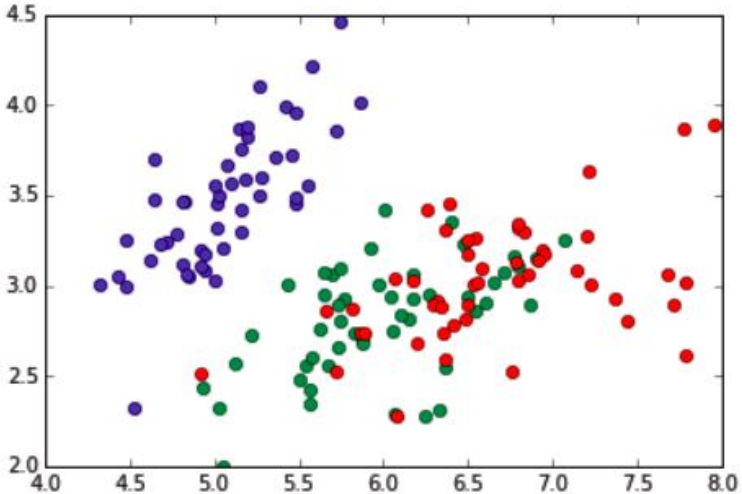
- Supervised learning: predict target values
- For discrete targets, often visualize with color



```
plt.hist( [X[Y==c,1] for c in np.unique(Y)] ,
          bins=20, histtype='barstacked')
```



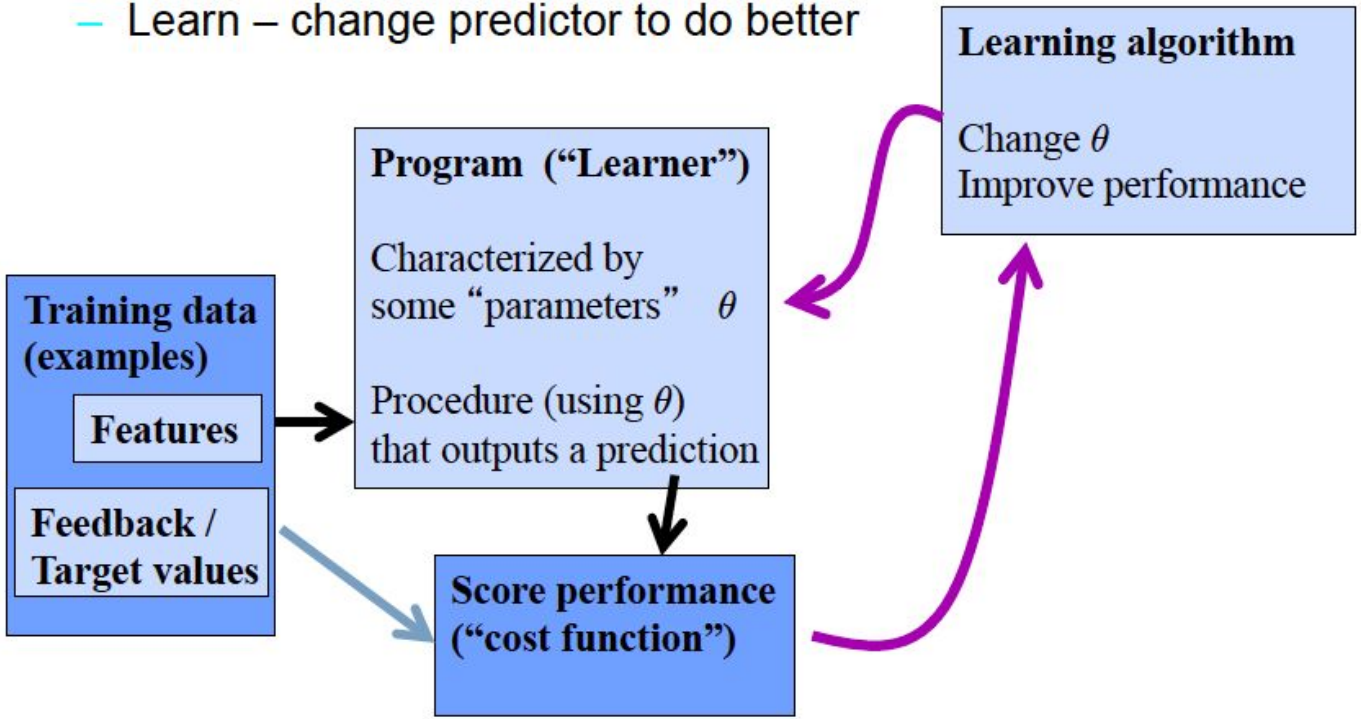
```
ml.histy(X[:,1], Y, bins=20)
```



```
colors = ['b','g','r']
for c in np.unique(Y):
    plt.plot( X[Y==c,0], X[Y==c,1], 'o',
              color=colors[int(c)] )
```

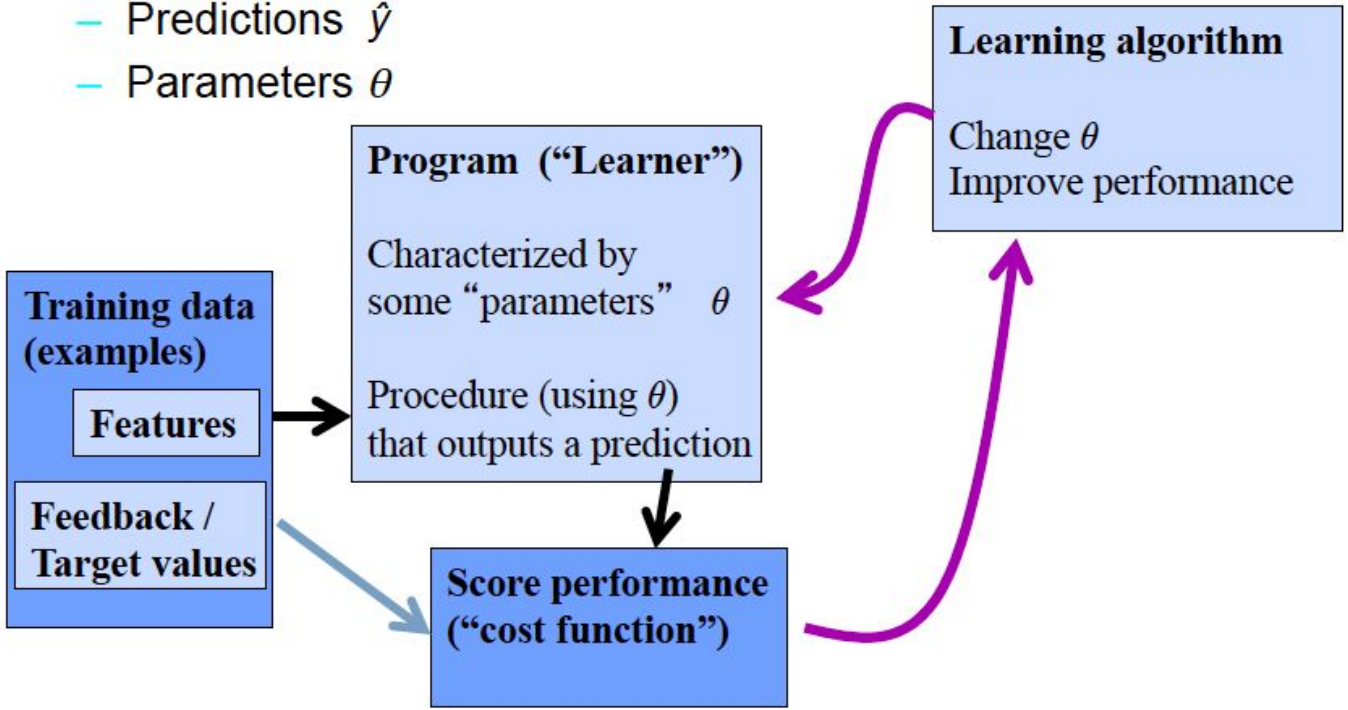
How does machine learning work?

- "Meta-programming"
  - Predict – apply rules to examples
  - Score – get feedback on performance
  - Learn – change predictor to do better

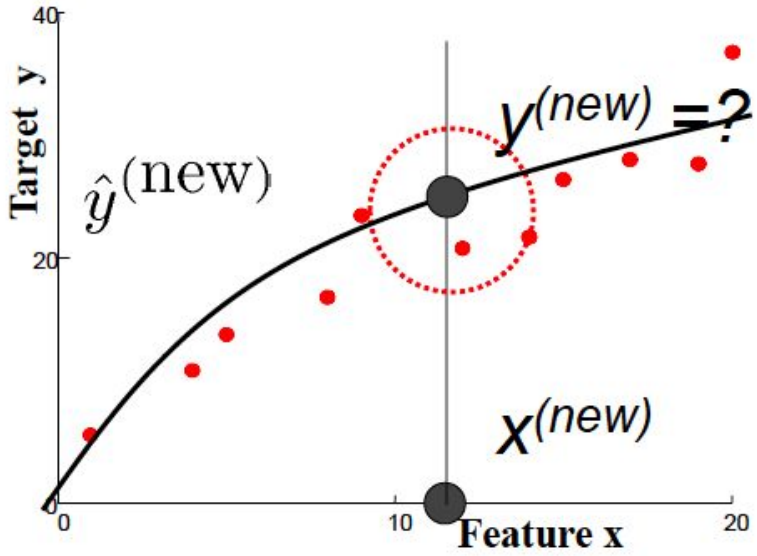


Supervised Learning

- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y}$
  - Parameters θ

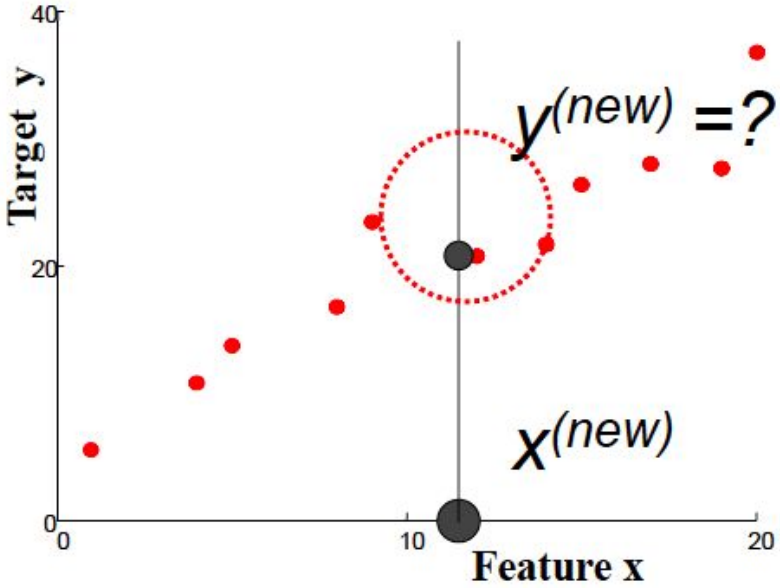


Regression; Scatter plots



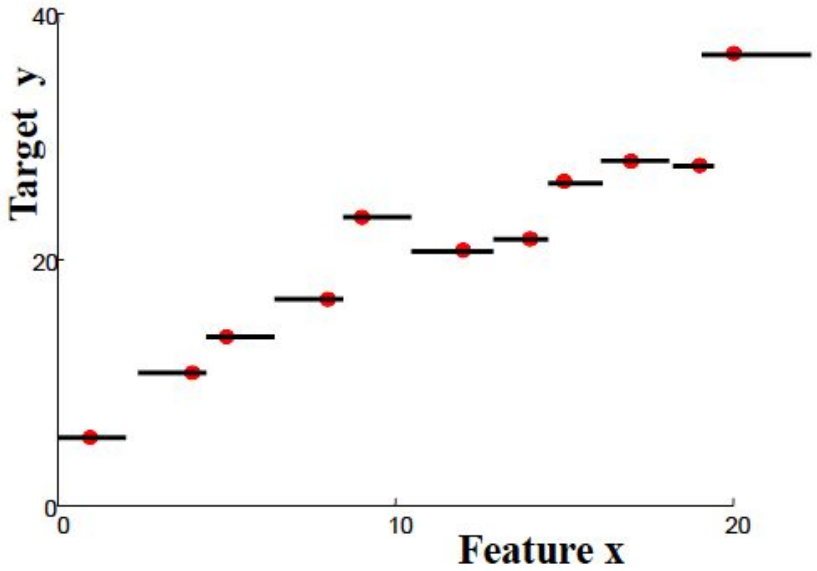
- Suggests a relationship between x and y
- *Prediction*: new x, what is y?

Nearest neighbor regression



- Find training datum $x^{(i)}$ closest to $x^{(new)}$
  Predict $y^{(i)}$

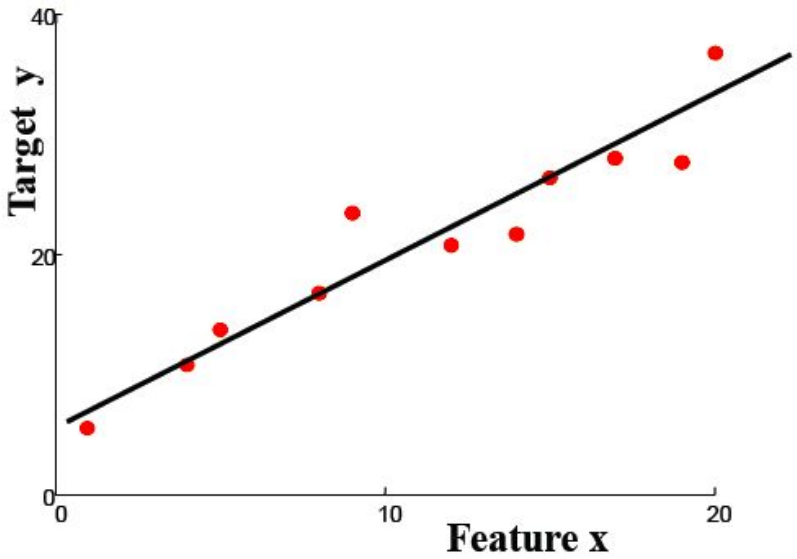
Nearest neighbor regression



"Predictor":
Given new features:
Find nearest example
Return its value

- Defines a function f(x) implicitly
- "Form" is piecewise constant

Linear regression



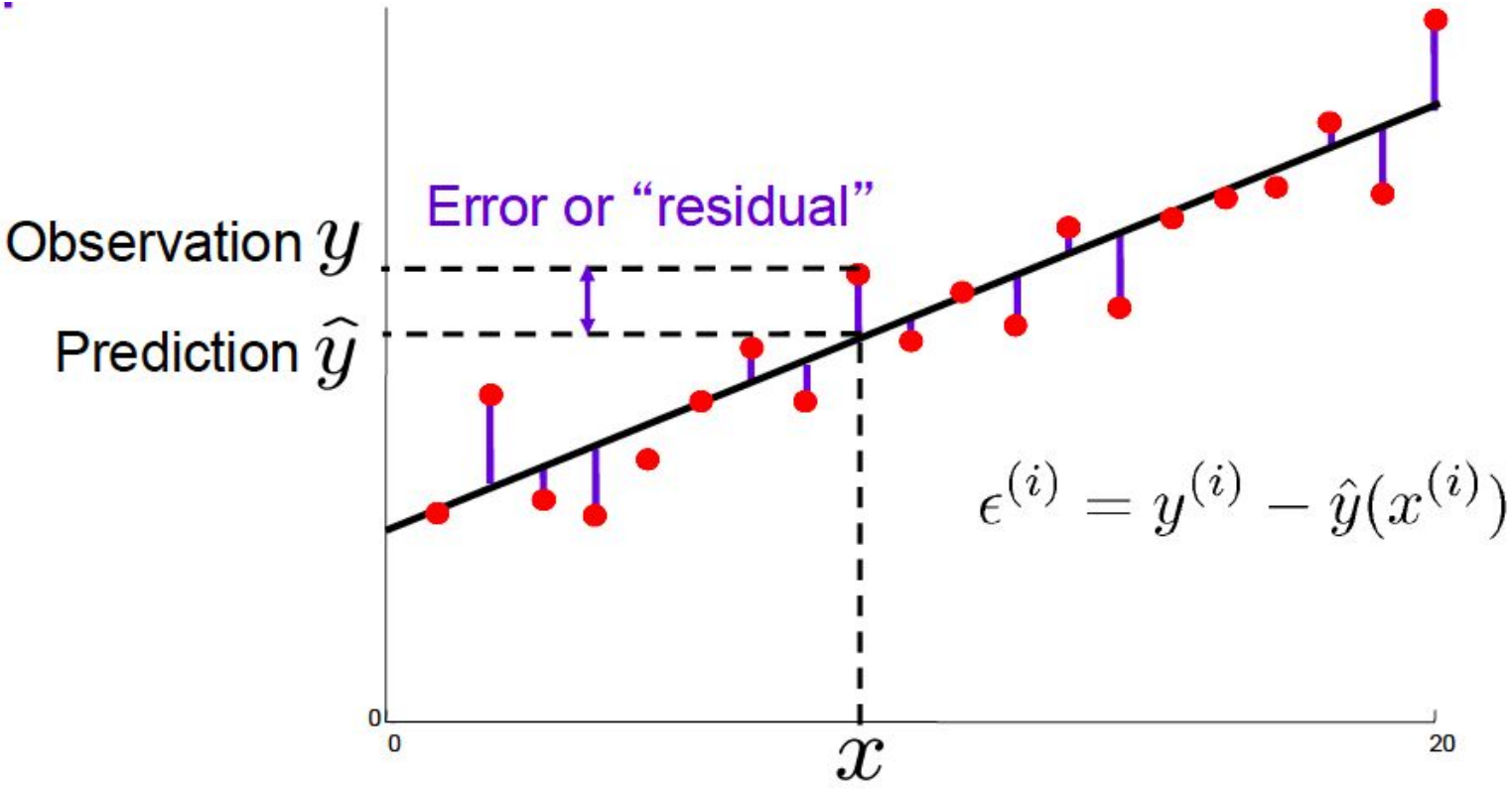
"Predictor":
Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

return r

- Define form of function f(x) explicitly
- Find a good f(x) within that family

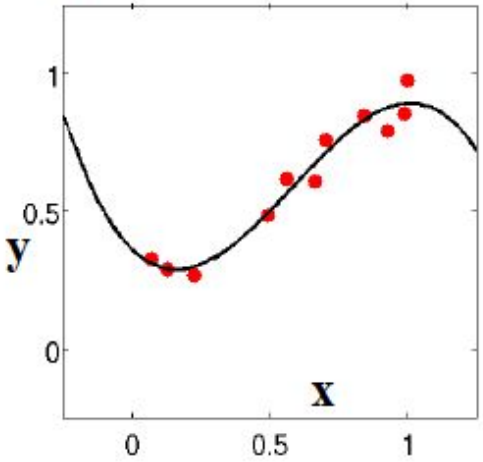
Measuring Errors



$$\text{MSE} = \frac{1}{m} \sum_i \left(y^{(i)} - \hat{y}(x^{(i)}) \right)^2$$

Regression vs. Classification

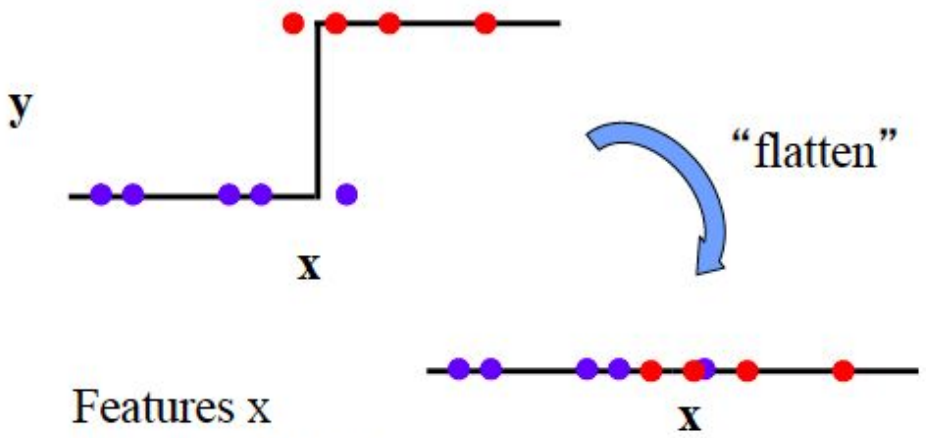
Regression



Features x
Real-valued target y

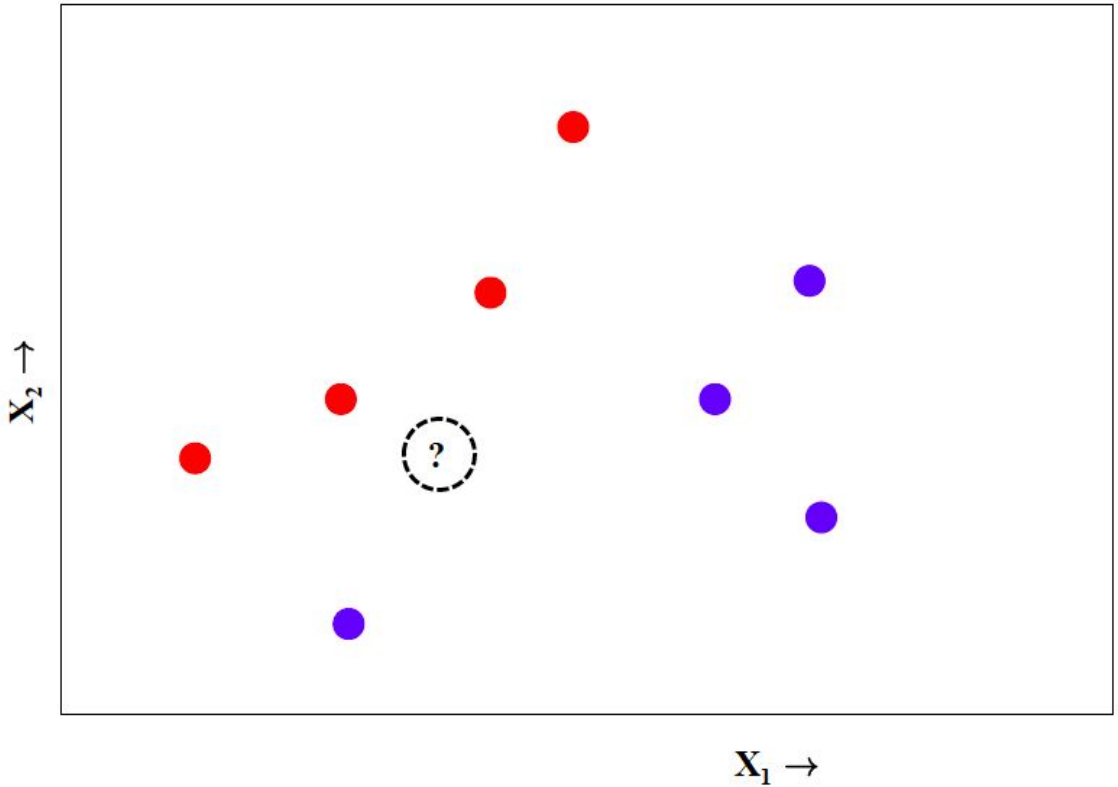
Predict continuous function $\hat{y}(x)$

Classification

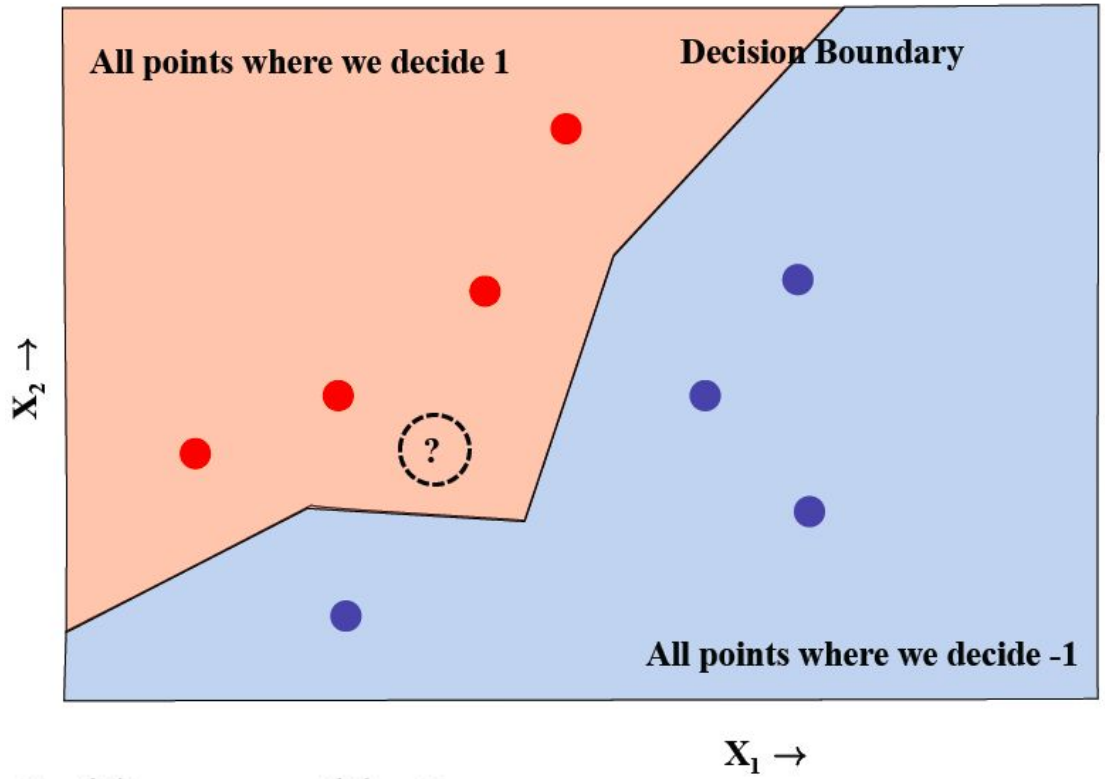


Features x
Discrete class c
(usually 0/1 or +1/-1)
Predict discrete function $\hat{y}(x)$

Classification

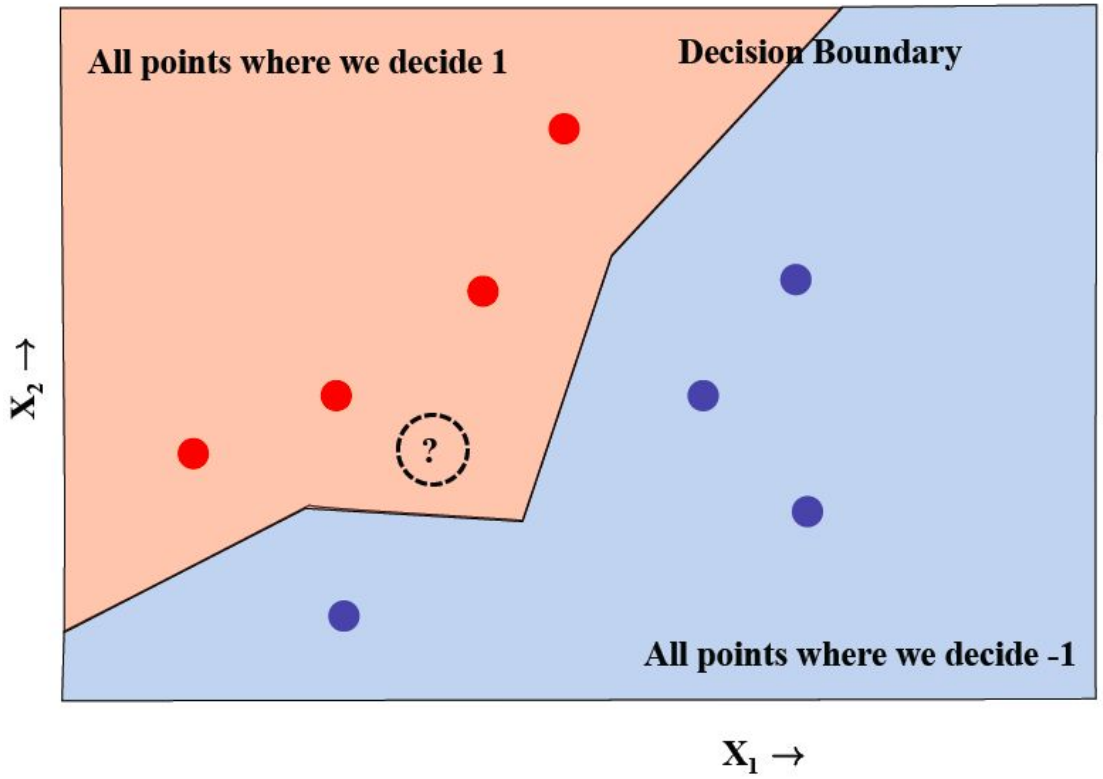


Measuring Error

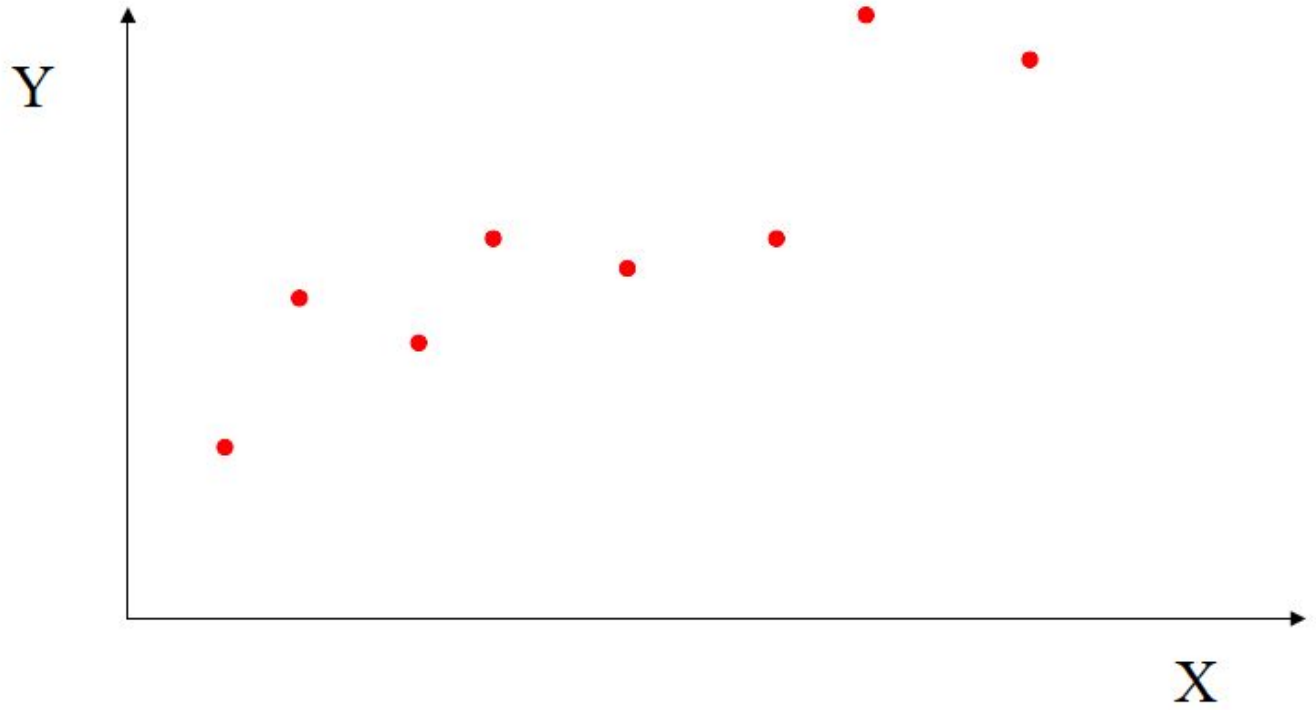


$$\text{ERR} = \frac{1}{m} \sum_{i} \left[y^{(i)} \neq \hat{y}(x^{(i)}) \right]$$

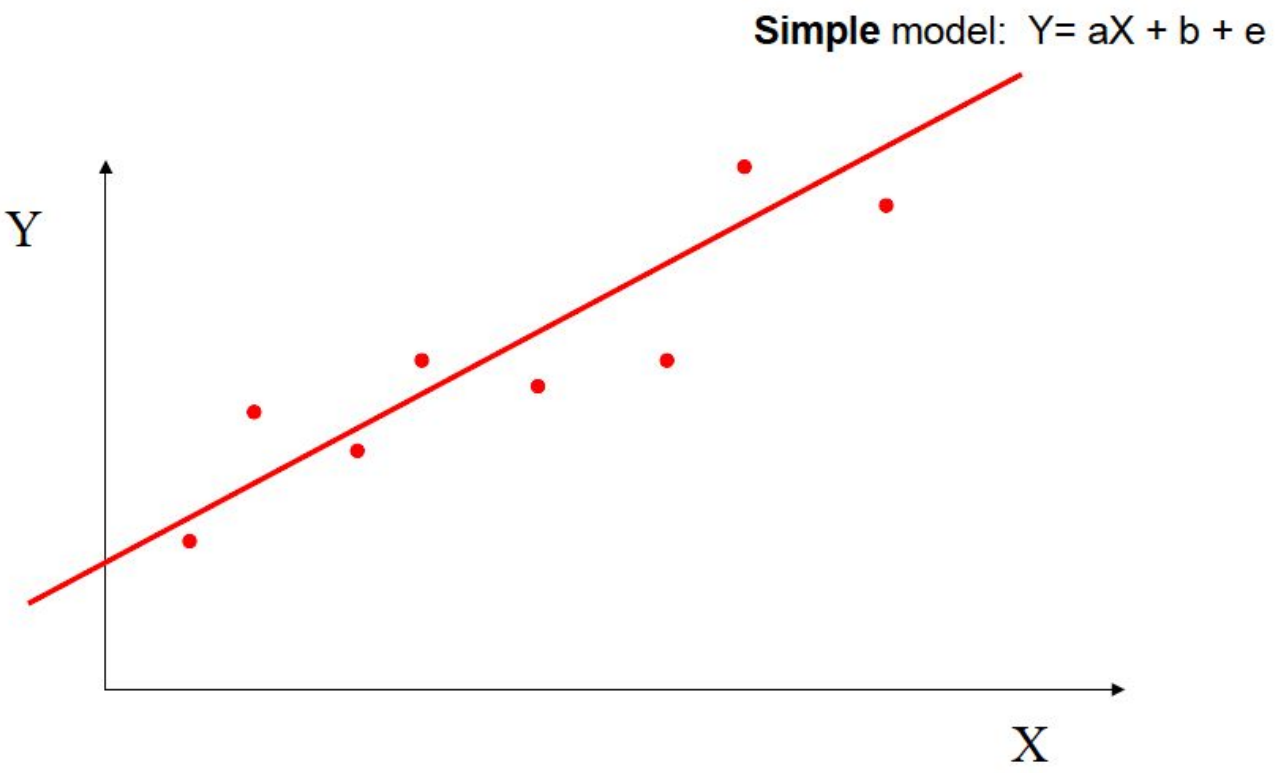
Classification



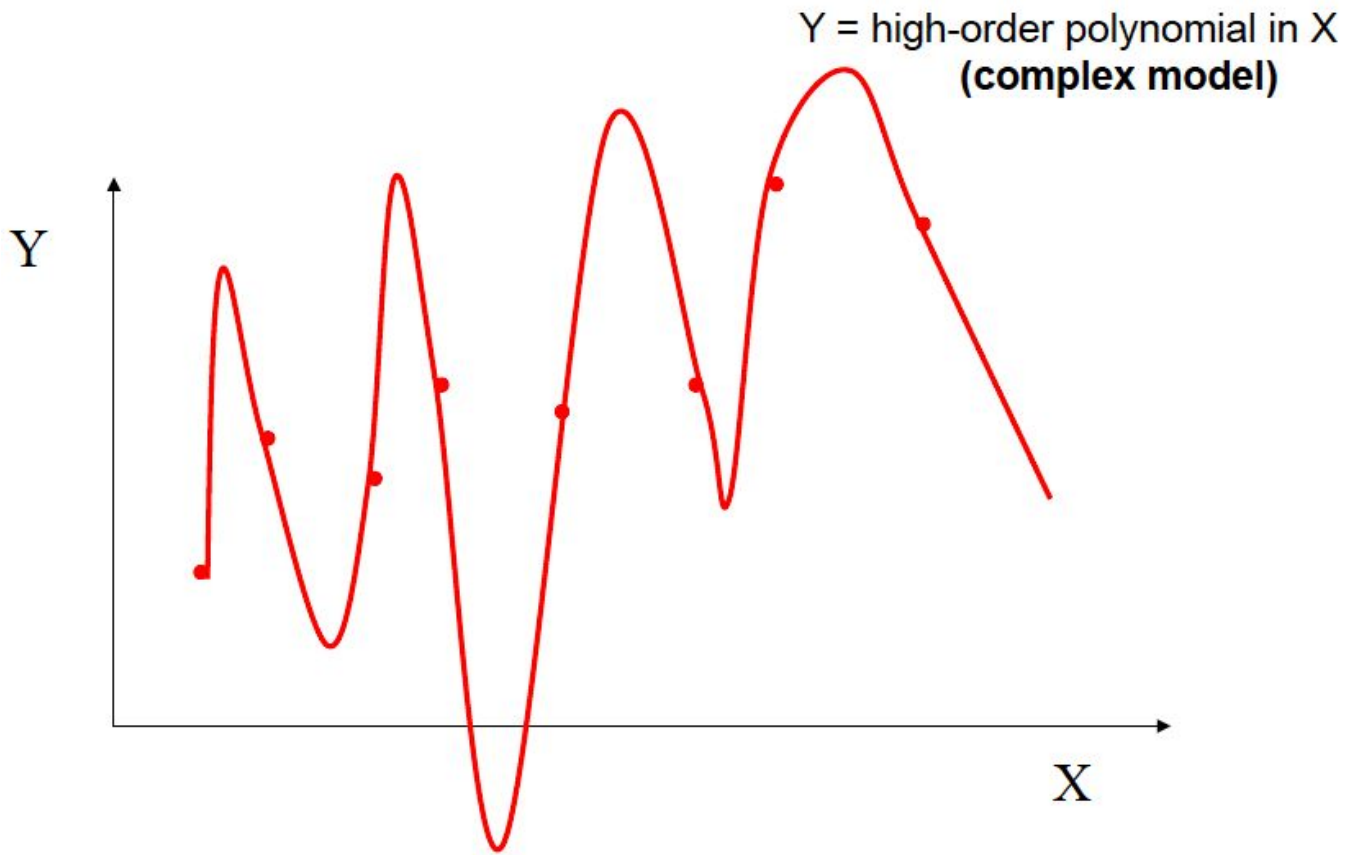
Overfitting and Complexity



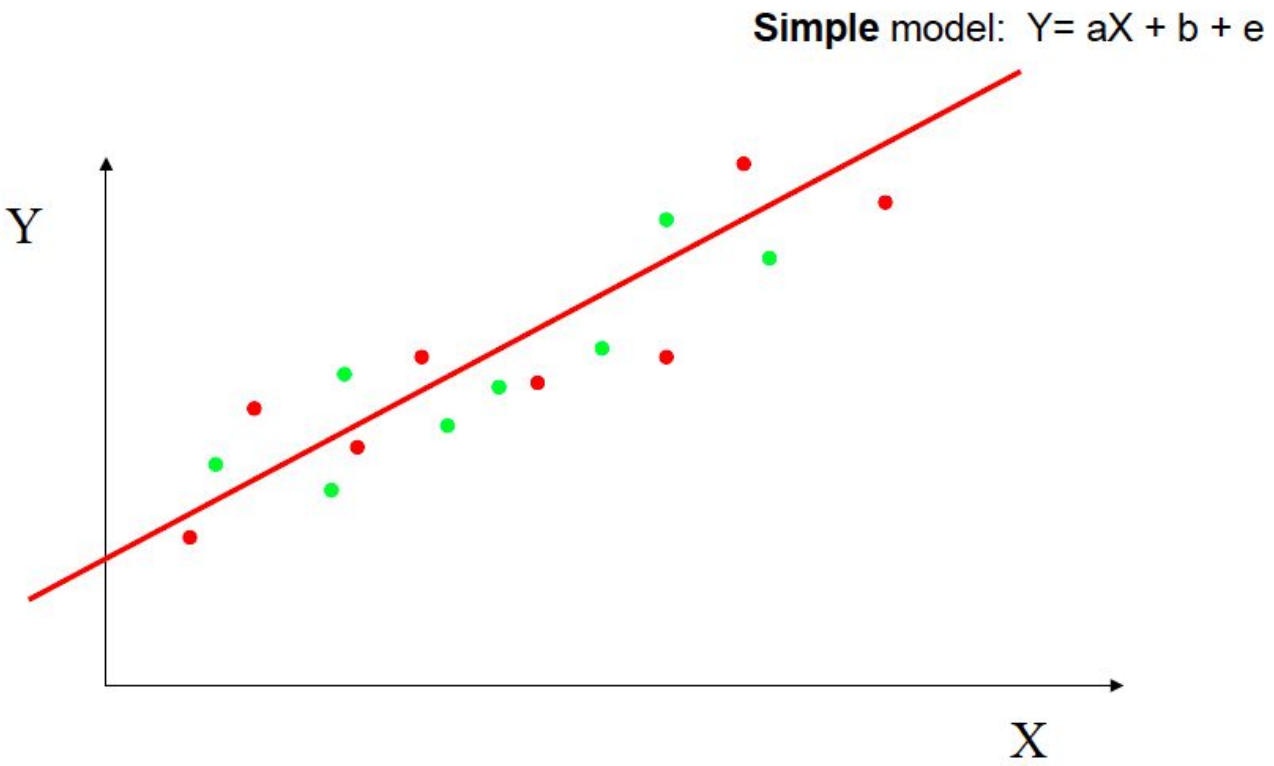
Overfitting and Complexity



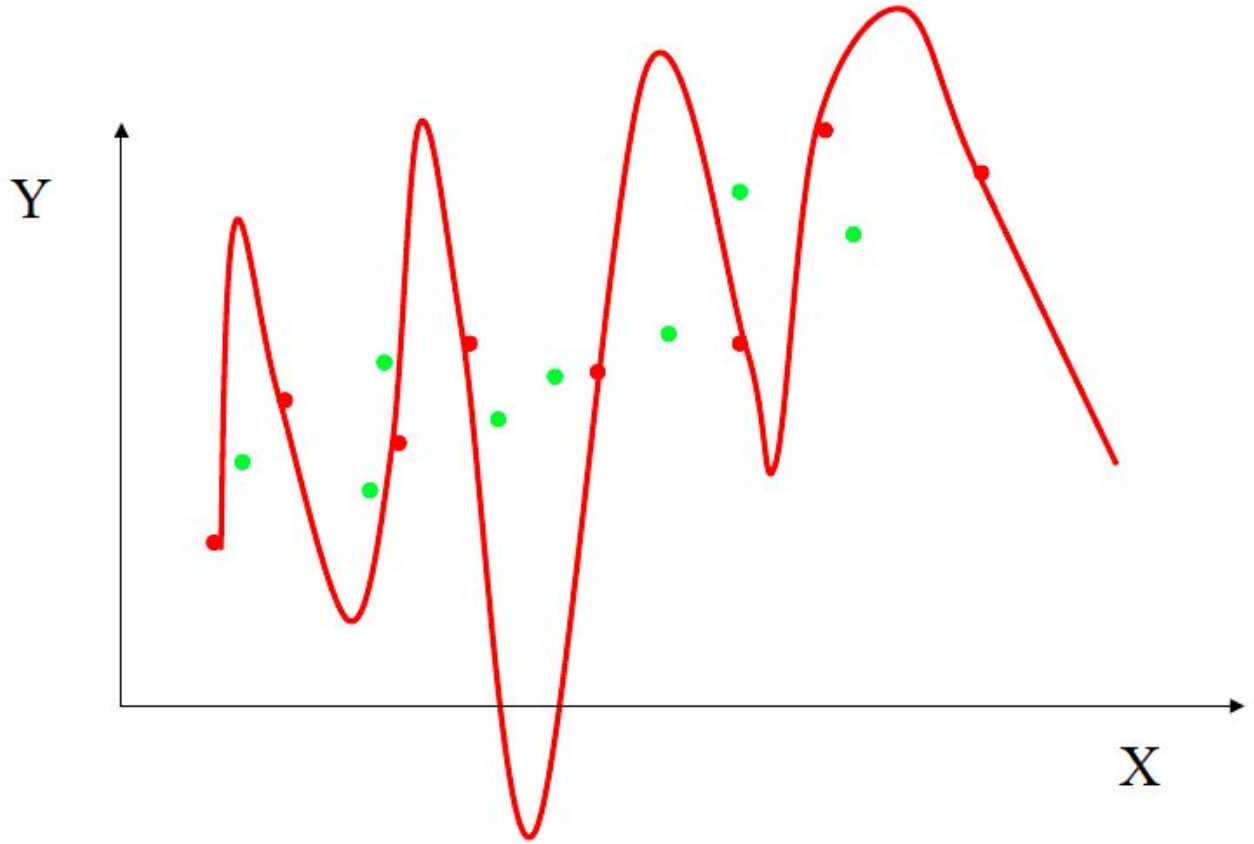
Overfitting and Complexity



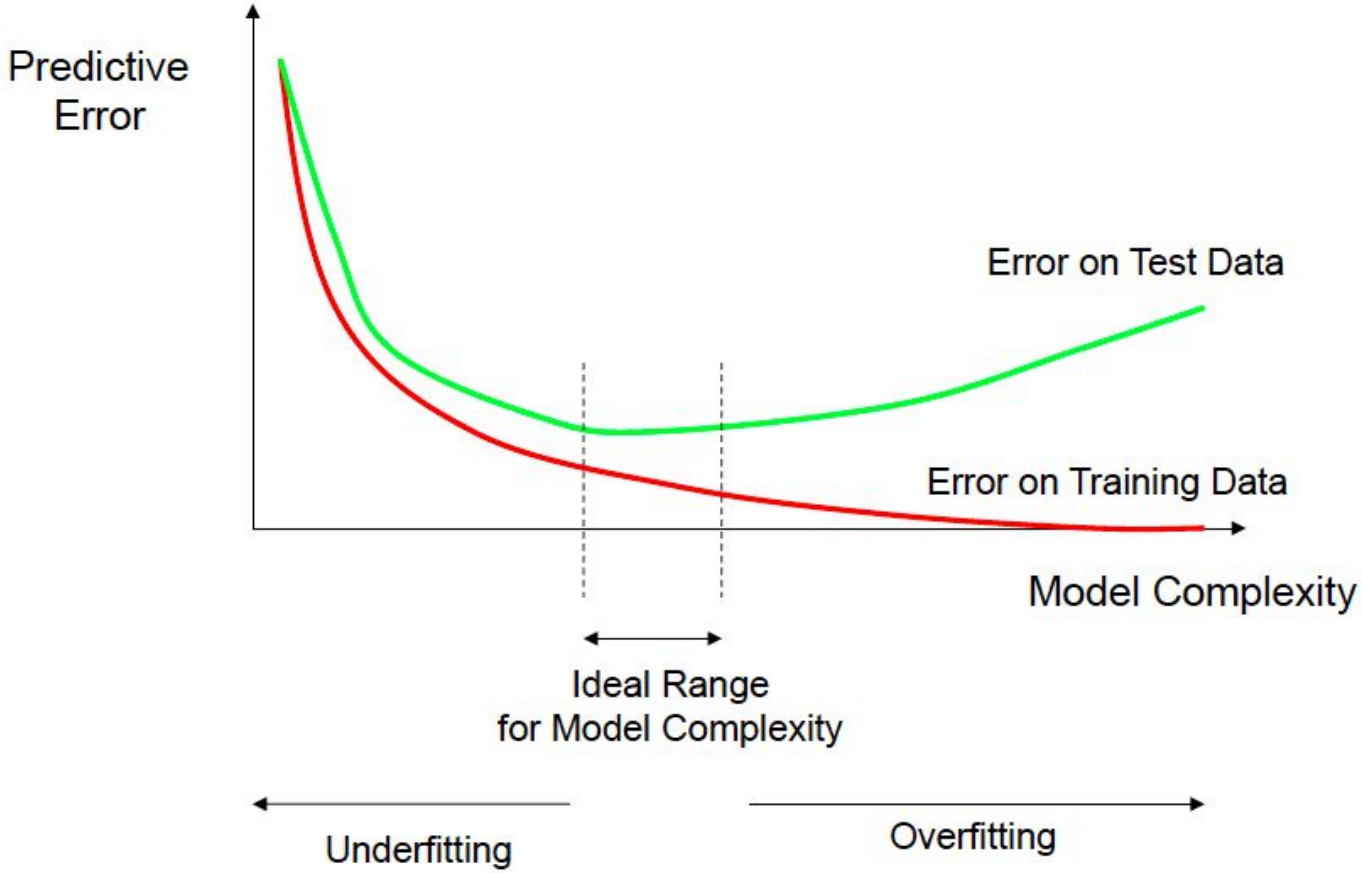
Overfitting and Complexity



Overfitting and Complexity



How overfitting affects prediction?

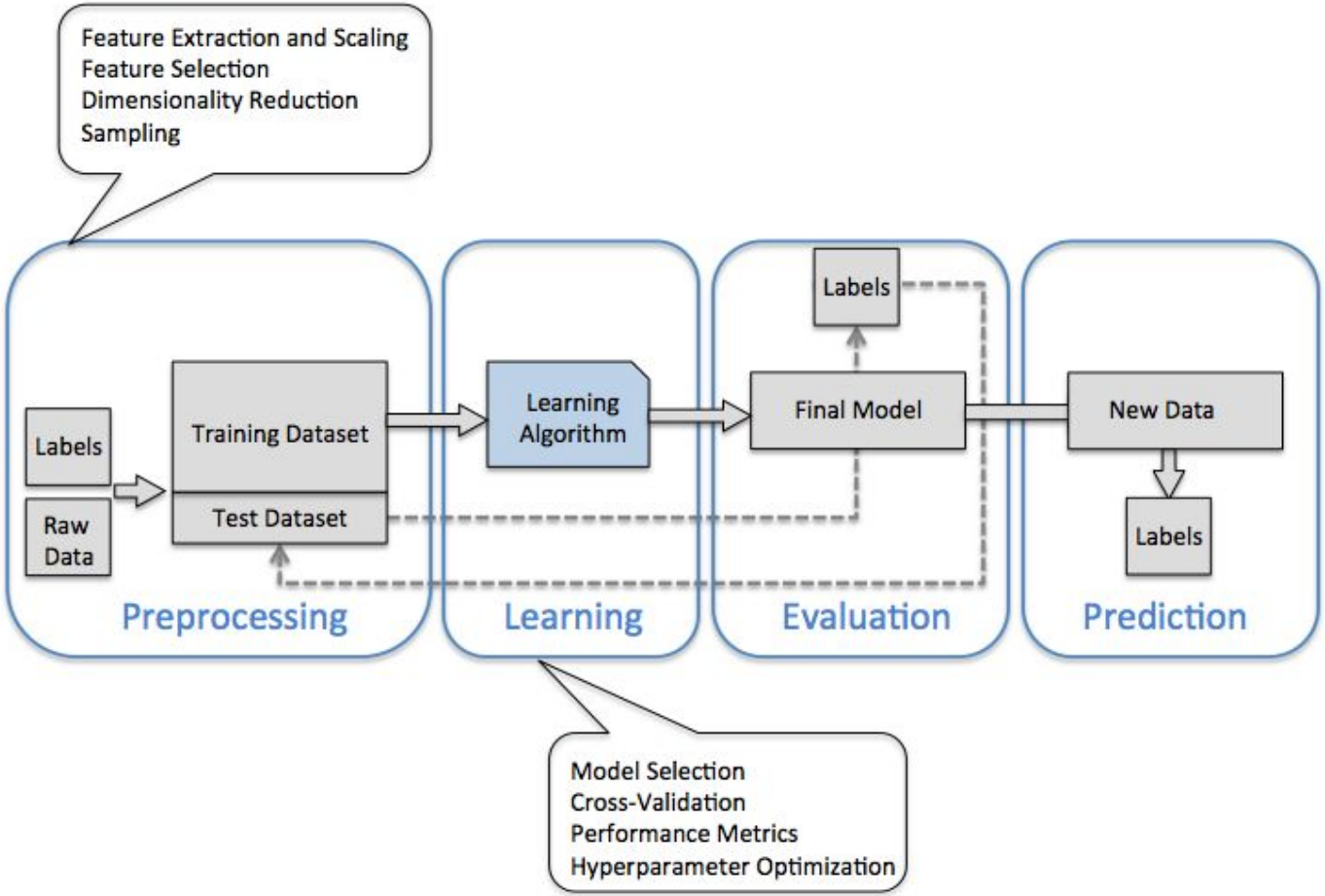


Data splitting used in competitions

- Training data
  - Used to build your model(s)
- Validation data
  - Used to assess, select among, or combine models
  - Personal validation; leaderboard; …
- Test data
  - Used to estimate “real world” performance

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UT
1	-	BrickMover *	1.21251	40	Sat, 31 Aug 2013 23:
2	new	vsu *	1.21552	13	Sat, 31 Aug 2013 20:
3	↑2	Merlion	1.22724	29	Sat, 31 Aug 2013 23:
4	↓2	Sergey	1.22856	15	Sat, 31 Aug 2013 23:
5	new	liuyongqi	1.22980	13	Sat, 31 Aug 2013 13:

Roadmap For Building Machine Learning Systems



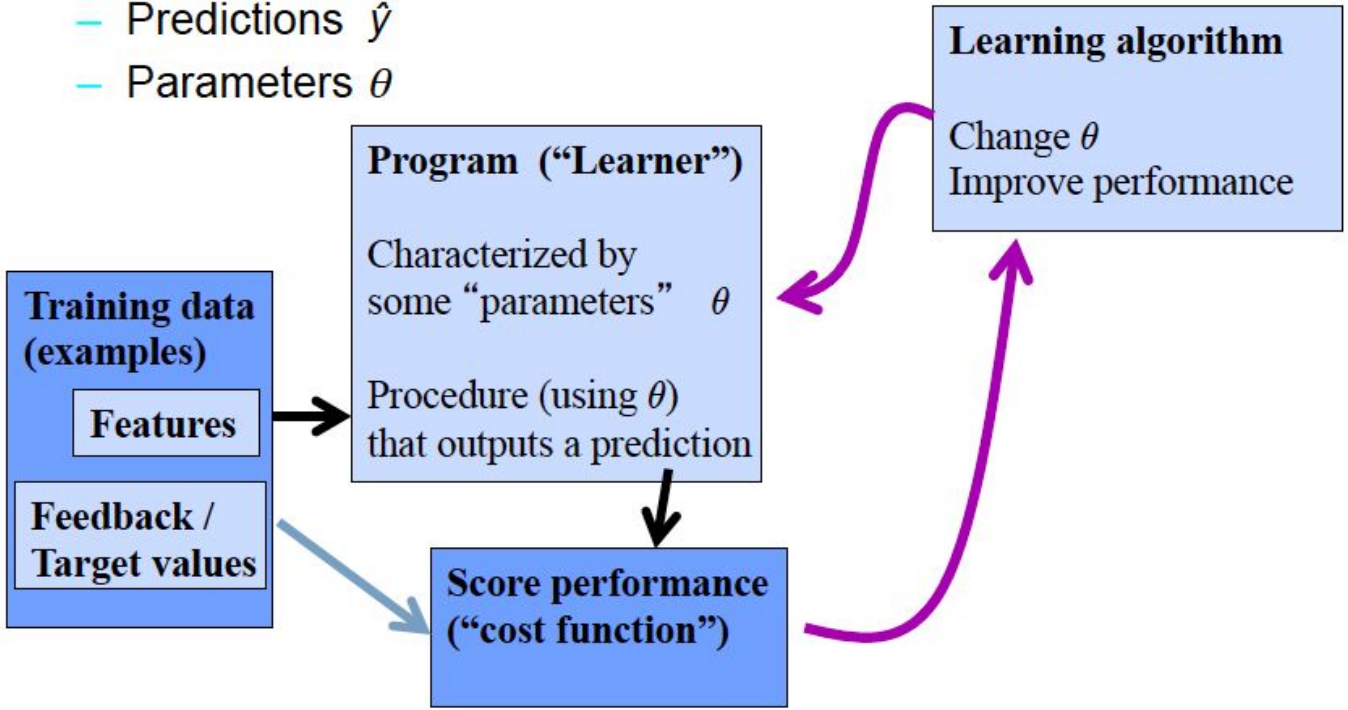
Summary so far

- What is machine learning?
  - Types of machine learning
  - How machine learning works
- Supervised learning
  - Training data: features x, targets y
- Regression
  - (x,y) scatterplots; predictor outputs f(x)
- Classification
  - (x,x) scatterplots
  - Decision boundaries, colors & symbols
- Complexity
  - Training vs test error
  - Under- & over-fitting

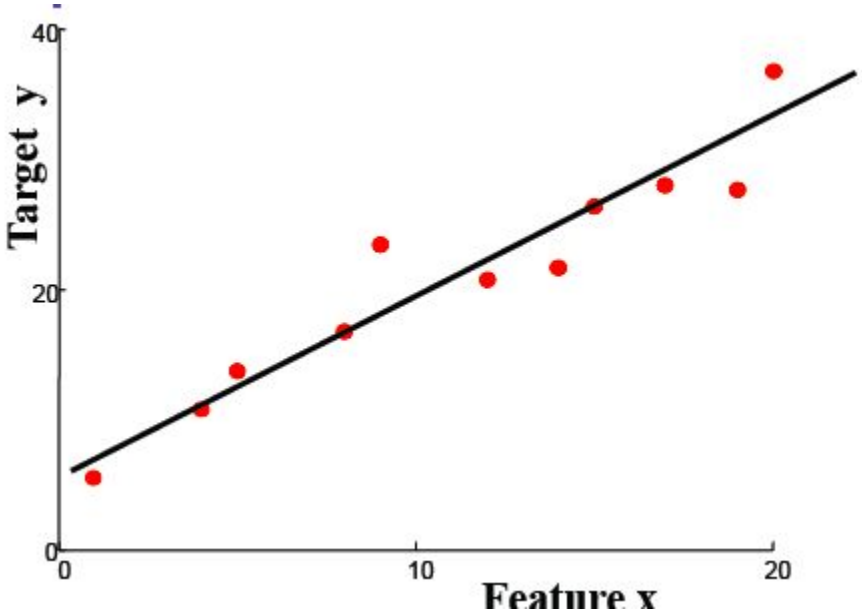
Linear Regression

Supervised Learning

- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y}$
  - Parameters θ



Linear Regression



"Predictor":
Evaluate line:

$$r = \theta_0 + \theta_1 x_1$$

return r

- Define form of function f(x) explicitly
- Find a good f(x) within that family

Notation

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

Define "feature" $x_0 = 1$ (constant)

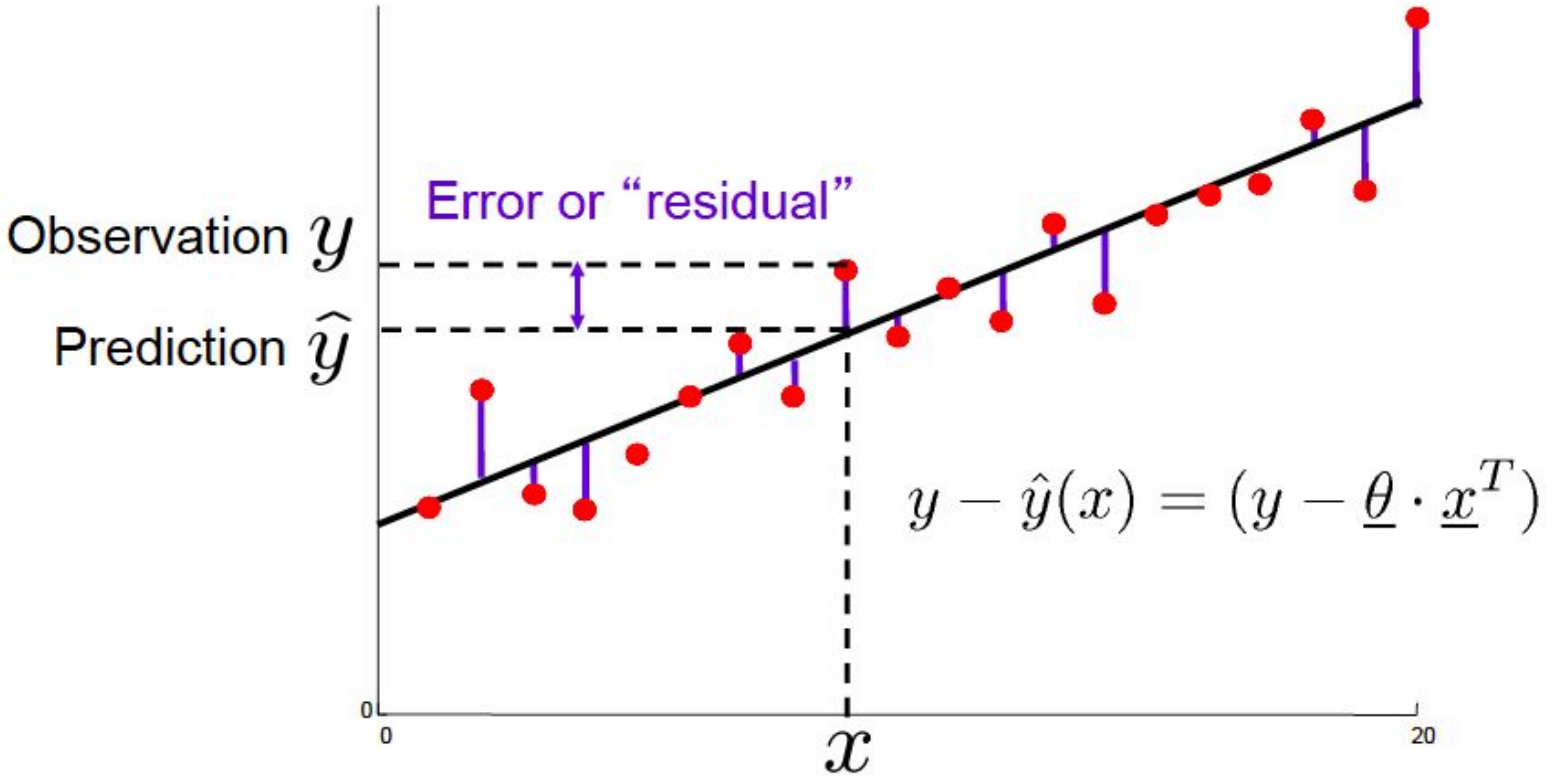
Then

$$\hat{y}(x) = \theta\, x^T$$

$$\underline{\theta} = [\theta_0, \dots, \theta_n]$$

$$\underline{x} = [1, x_1, \dots, x_n]$$

Measuring Error



Mean squared error

- How can we quantify the error?

$$\text{MSE, } J(\underline{\theta}) = \frac{1}{m}\sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2$$

$$= \frac{1}{m}\sum_j (y^{(j)} - \underline{\theta}\cdot\underline{x}^{(j)T})^2$$

- Could choose something else, of course…
  - Computationally convenient (more later)
  - Measures the variance of the residuals
  - Corresponds to likelihood under Gaussian model of "noise"

$$\mathcal{N}(y\ ;\ \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\}$$

MSE Cost Function

$$\text{MSE, } J(\underline{\theta}) = \frac{1}{m}\sum_j (y^{(j)} - \hat{y}(x^{(j)}))^2$$

$$= \frac{1}{m}\sum_j (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)T})^2$$

- Rewrite using matrix form

$$\underline{\theta} = [\theta_0, \ldots, \theta_n]$$

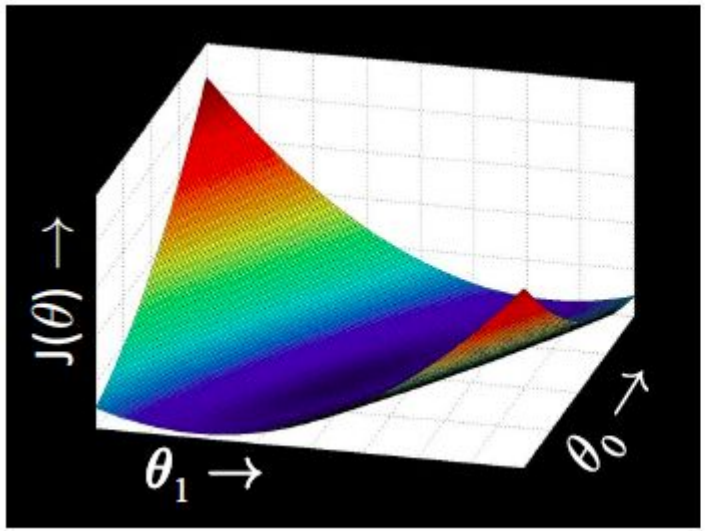
$$\underline{y} = \left[y^{(1)} \ldots, y^{(m)}\right]^T$$

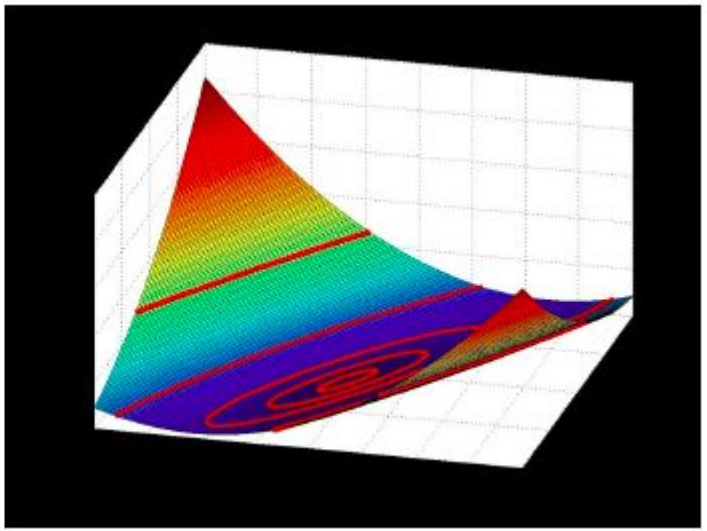
$$\underline{X} = \begin{bmatrix} x_0^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

$$J(\underline{\theta}) = \frac{1}{m}(\underline{y}^T - \underline{\theta}\,\underline{X}^T) \cdot (\underline{y}^T - \underline{\theta}\,\underline{X}^T)^T$$

```
# Python / NumPy:
e = Y – X.dot( theta.T );
J = e.T.dot( e ) / m  # = np.mean( e ** 2 )
```

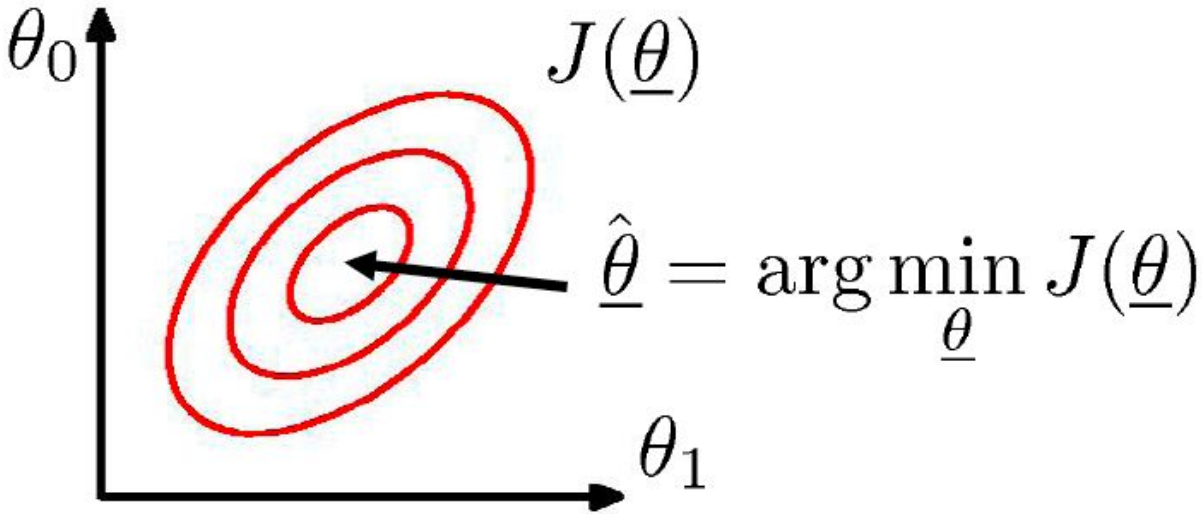
Visualizing Cost Function



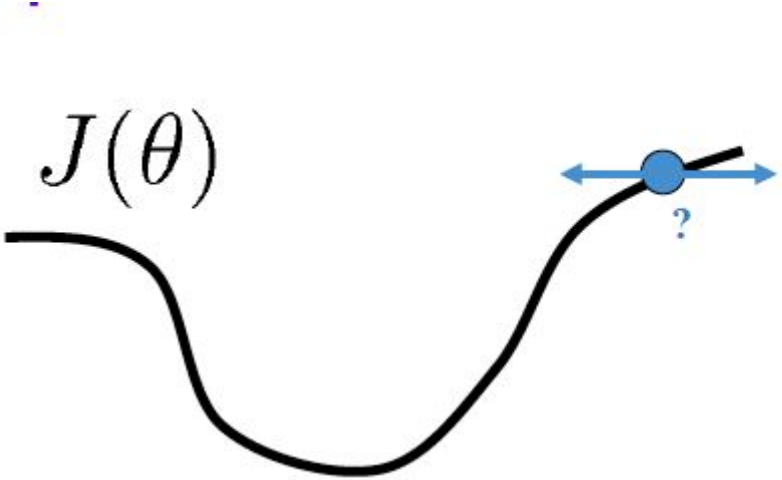


Learning: find the best parameters

- Want to find parameters which minimize our error…
- Think of a cost "surface": error residual for that θ…

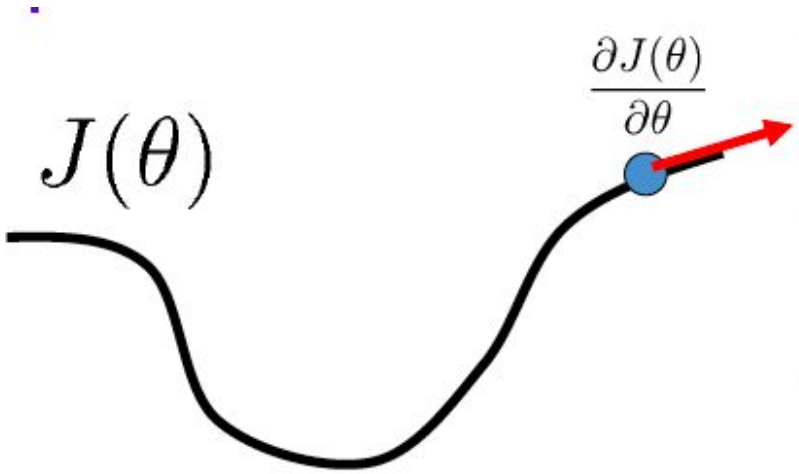


Gradient descent



- How to change θ to improve J(θ)?
- Choose a direction in which J(θ) is decreasing

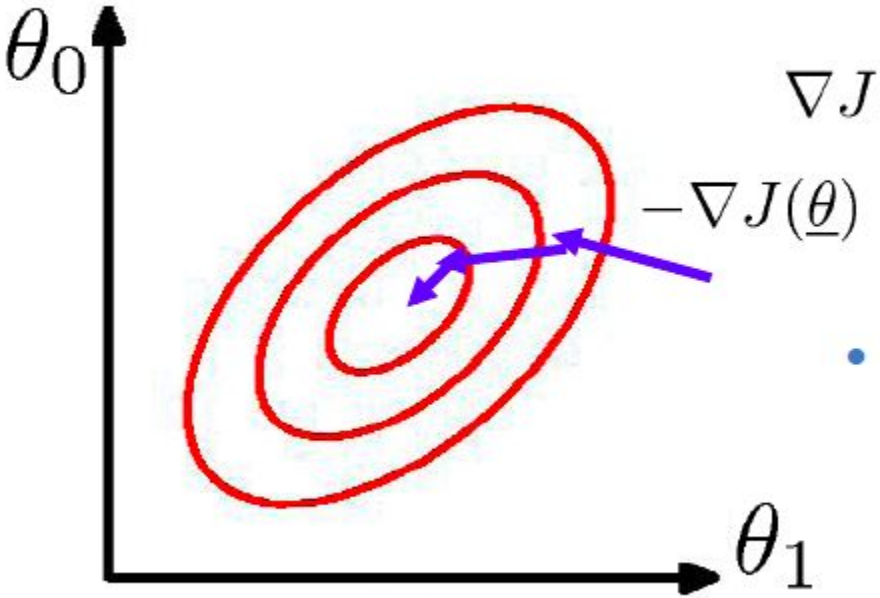
Gradient descent



- How to change θ to improve J(θ)?
- Choose a direction in which J(θ) is decreasing
- Derivative $\frac{\partial J(\theta)}{\partial \theta}$

- Positive => increasing
- Negative => decreasing

Gradient descent in more dimensions



- Gradient vector

$$\nabla J(\underline{\theta}) = \left[\frac{\partial J(\underline{\theta})}{\partial \theta_0} \quad \frac{\partial J(\underline{\theta})}{\partial \theta_1} \quad \dots \right]$$

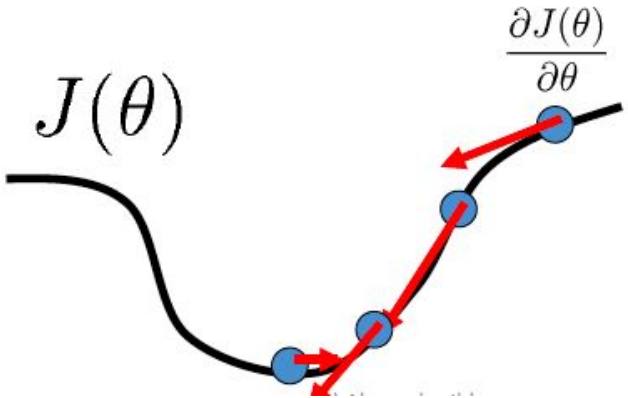
- Indicates direction of steepest ascent

(negative = steepest descent)

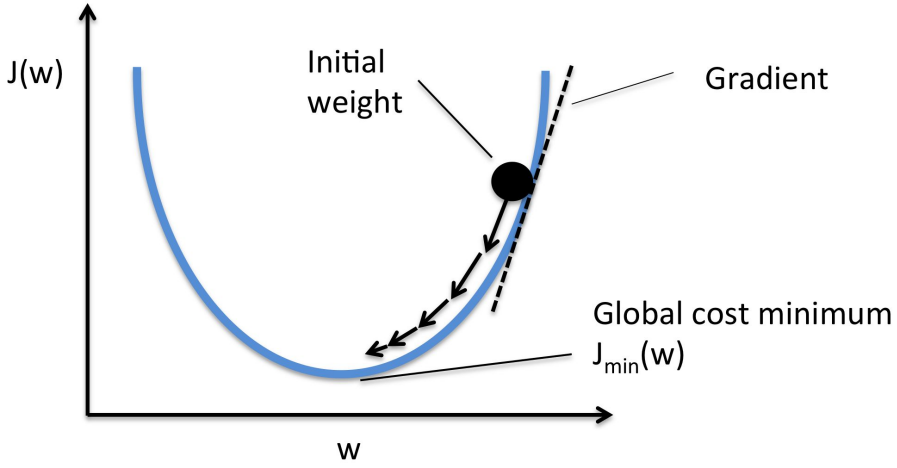
Gradient descent

- Initialization
- Step size
  - Can change as a function of iteration
- Gradient direction
- Stopping condition

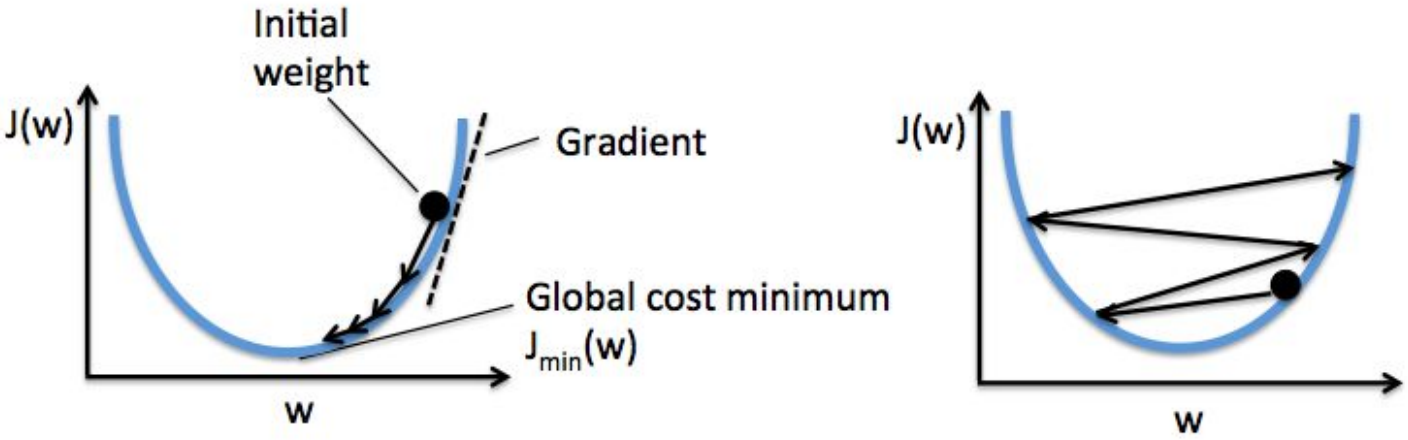
```
Initialize θ
Do {
  θ ← θ - α ∇_θ J(θ)
} while ( α||∇J|| > ε )
```



Minimizing cost functions: gradient descent



Gradient descent: Learning rate



Derivative of MSE

$$\nabla J(\underline{\theta}) = -\frac{2}{m}\sum_j (y^{(j)} - \underline{\theta}\cdot\underline{x}^{(j)T})\cdot[x_0^{(j)}x_1^{(j)}\ldots]$$

Error magnitude & direction for datum j

Sensitivity to each θ_i

- Rewrite using matrix form

$$\underline{\theta} = [\theta_0, \ldots, \theta_n]$$

$$\underline{y} = \left[y^{(1)} \ldots, y^{(m)}\right]^T$$

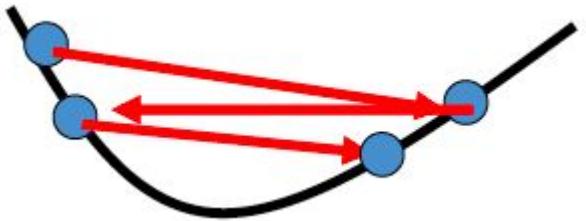
$$\underline{X} = \begin{bmatrix} x_0^{(1)} & \ldots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \ldots & x_n^{(m)} \end{bmatrix}$$

$$\nabla J(\underline{\theta}) = -\frac{2}{m}(\underline{y}^T - \underline{\theta X}^T)\cdot\underline{X}$$

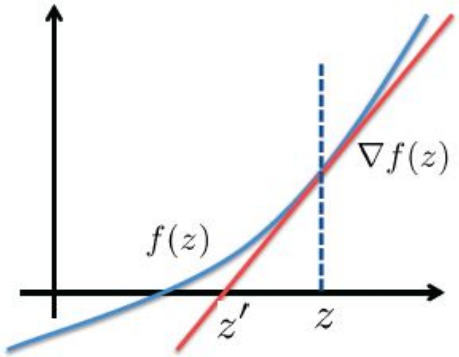
```
e = Y – X.dot( theta.T ); # error residual
DJ = - e.dot(X) * 2.0/m  # compute the gradient
theta -= alpha * DJ      # take a step
```

Comments on gradient descent

- Very general algorithm
  - we'll see it many times
- Local minima
  - Sensitive to starting point
- Step size
  - Too large? Too small? Automatic ways to choose?
  - May want step size to decrease with iteration
  - Common choices:
    - Fixed
    - Linear: C/(iteration)
    - Line search / backoff (Armijo, etc.)
    - Newton's method



Newton's method



- Want to find the roots of f(x)
  - "Root": value of x for which f(x)=0
- Initialize to *some* point x
- Compute the tangent at x & compute where it crosses x-axis

$$\nabla f(z) = \frac{0 - f(z)}{z' - z} \quad \Rightarrow \quad z' = z - \frac{f(z)}{\nabla f(z)}$$

- Optimization: find roots of $\nabla J(\theta)$

("Step size" $\lambda = 1/\nabla\nabla J$; inverse curvature)

$$\nabla\nabla J(\theta) = \frac{0 - \nabla J(\theta)}{\theta' - \theta} \quad \Rightarrow \quad \theta' = \theta - \frac{\nabla J(\theta)}{\nabla\nabla J(\theta)}$$

  - Does not always converge; sometimes unstable
  - If converges, usually very fast
  - Works well for smooth, non-pathological functions, locally quadratic

(Multivariate:
∇ J(θ) = gradient vector
∇^2 J(θ) = matrix of 2nd derivatives
a/b = a b^{-1}, matrix inverse)

Stochastic / Online Gradient Descent

- MSE

$$J(\underline{\theta}) = \frac{1}{m}\sum_j J_j(\underline{\theta}), \qquad J_j(\underline{\theta}) = (y^{(j)} - \underline{\theta}\cdot \underline{x}^{(j)^T})^2$$

- Gradient

$$\nabla J(\underline{\theta}) = \frac{1}{m}\sum_j \nabla J_j(\underline{\theta}) \qquad \nabla J_j(\underline{\theta}) = (y^{(j)} - \underline{\theta}\cdot \underline{x}^{(j)^T})\cdot [x_0^{(j)} x_1^{(j)} \ldots]$$

- Stochastic (or "online") gradient descent:
  - Use updates based on individual datum j, chosen at random
  - At optima, $\mathbb{E}\left[\nabla J_j(\underline{\theta})\right] = \nabla J(\underline{\theta}) = 0$
    (average over the data)

Online gradient descent

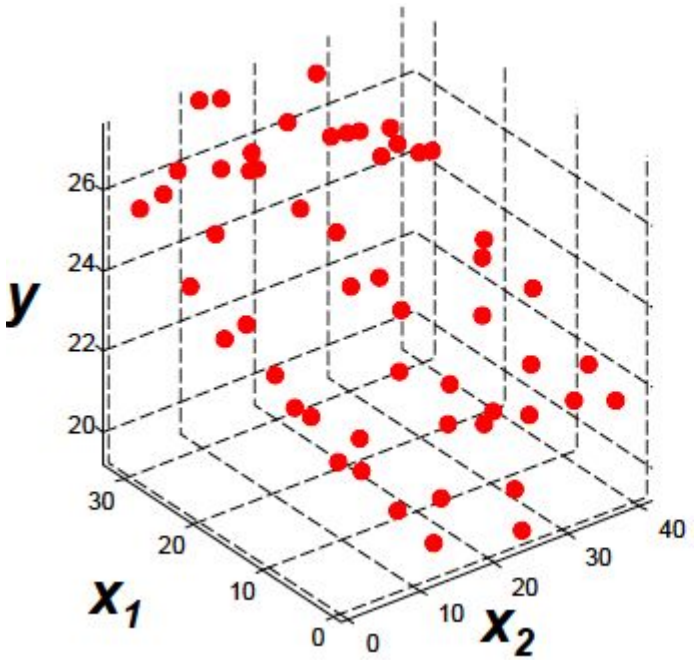
$$J_j(\underline{\theta}) = (y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)^T})^2$$

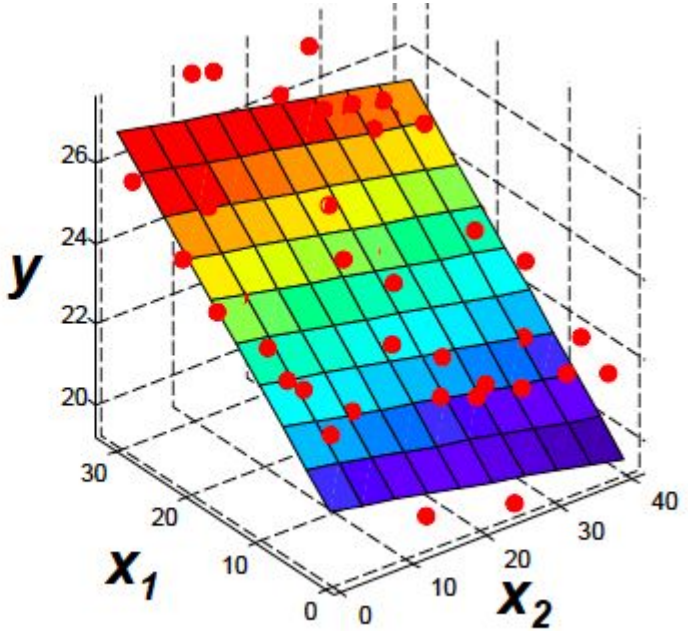
$$\nabla J_j(\underline{\theta}) = -2(y^{(j)} - \underline{\theta} \cdot \underline{x}^{(j)^T}) \cdot [x_0^{(j)} x_1^{(j)} \ldots]$$

- Benefits
  - Lots of data = many more updates per pass
  - Computationally faster
- Drawbacks
  - No longer strictly "descent"
  - Stopping conditions may be harder to evaluate
    (Can use "running estimates" of J(.), etc.)
- Related: mini-batch updates, etc.

```
Initialize θ
Do {
  for j=1:m
    θ ← θ - α ∇_θ J_j(θ)
} while (not converged)
```

More dimensions

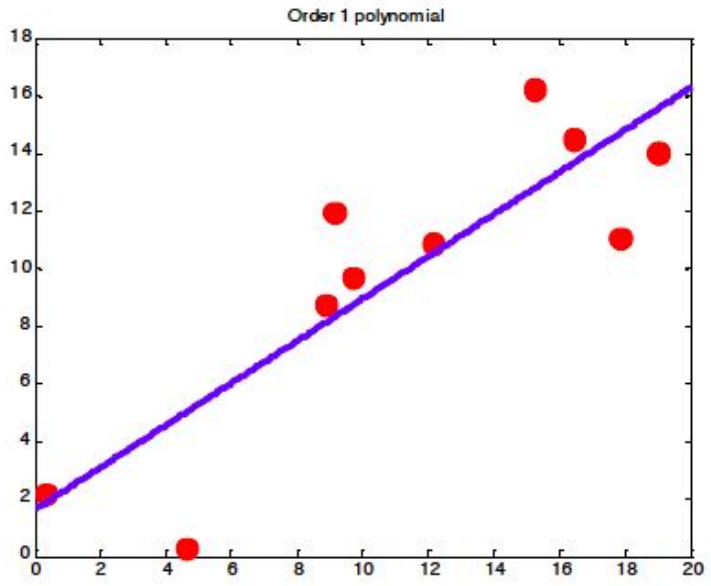


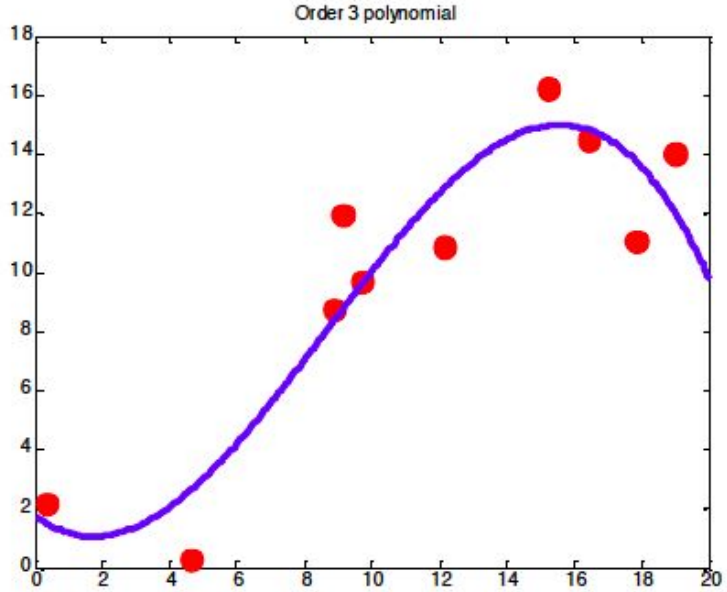


$$\hat{y}(x) = \underline{\theta} \cdot \underline{x}^T$$

Nonlinear functions

What if the underlying model is not a line?





Nonlinear functions

- Single feature x, predict target y:

$$D = \{(x^{(j)}, y^{(j)})\}$$

$$\hat{y}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Add features:

$$D = \{([x^{(j)}, (x^{(j)})^2, (x^{(j)})^3], y^{(j)})\}$$

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

Linear regression in new features

- Sometimes useful to think of "feature transform"

$$\Phi(x) = [1\,,\, x\,,\, x^2\,,\, x^3\,,\, \ldots]$$

$$\hat{y}(x) = \underline{\theta} \cdot \Phi(x)$$

Features

- In general, can use any features we think are useful

- Other information about the problem
  - Sq. footage, location, age, …
- Polynomial functions
  - Features [1, x, x^2, x^3, …]
- Other functions
  - 1/x, sqrt(x), $x_1 * x_2$, …

- "Linear regression" = linear in the parameters
  - Features we can make as complex as we want!

Overfitting and complexity

- More complex models will always fit the training data Better
- But they may “overfit” the training data, learning complex relationships that are not really present