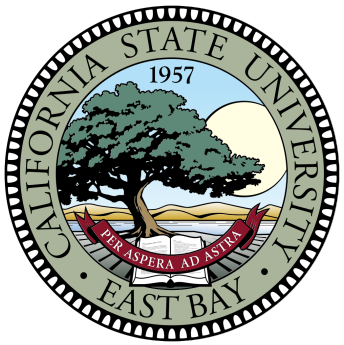


CS 6820 – Machine Learning

Lecture 5

Instructor: Eric S. Gayles, PhD.

Jan 16, 2018

ML Tools - Updated

- Matlab
- GNU Octave
- SciPy, Numpy, SciKit
  - Recommend installing Jupyter from www.anaconda.com (evolved from iPython)
  - Good Read–Eval–Print Loop (REPL) environment
  - https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks
- Basic Linear Algebra Subprograms (BLAS)
- LAPACK — Linear Algebra PACKage
- Shogun-toolbox
- MLlib – Apache
- Deeplearn.js
- ConvnetJS
- MLPack
- TensorFlow

NumPy and SciPy

- NumPy and SciPy are open-source add-on modules to Python that provide common mathematical and numerical routines in pre-compiled, fast functions. These are growing into highly mature packages that provide functionality that meets, or perhaps exceeds, that associated with common commercial software like MatLab.
  - NumPy extends Python to support efficient operations on large arrays and multidimensional matrices.

- The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data.

- The SciPy (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques.
  - Modules for scientific computing
  - https://docs.scipy.org/doc/scipy/reference/tutorial/

- matplotlib provides visualization tools

Python Libraries Used with ML

- **NumPy** - *mainly* useful for its *N*-dimensional array objects
- **pandas** - Python data analysis library, including structures such as dataframes
- **matplotlib** - 2D plotting library producing publication quality figures
- **scikit-learn** - the machine learning algorithms used for data analysis and data mining tasks

Scikit-Learn

- Conceived as an extension to the SciPy library, scikit-learn is built on the popular Python libraries NumPy and matplotlib.

Pandas

- Pandas is an open source library that provides data structures and analysis tools for Python.

- Pandas is a powerful library, and several books describe how to use pandas for data analysis.

- We will use a few of panda's convenient tools for importing data and calculating summary statistics.

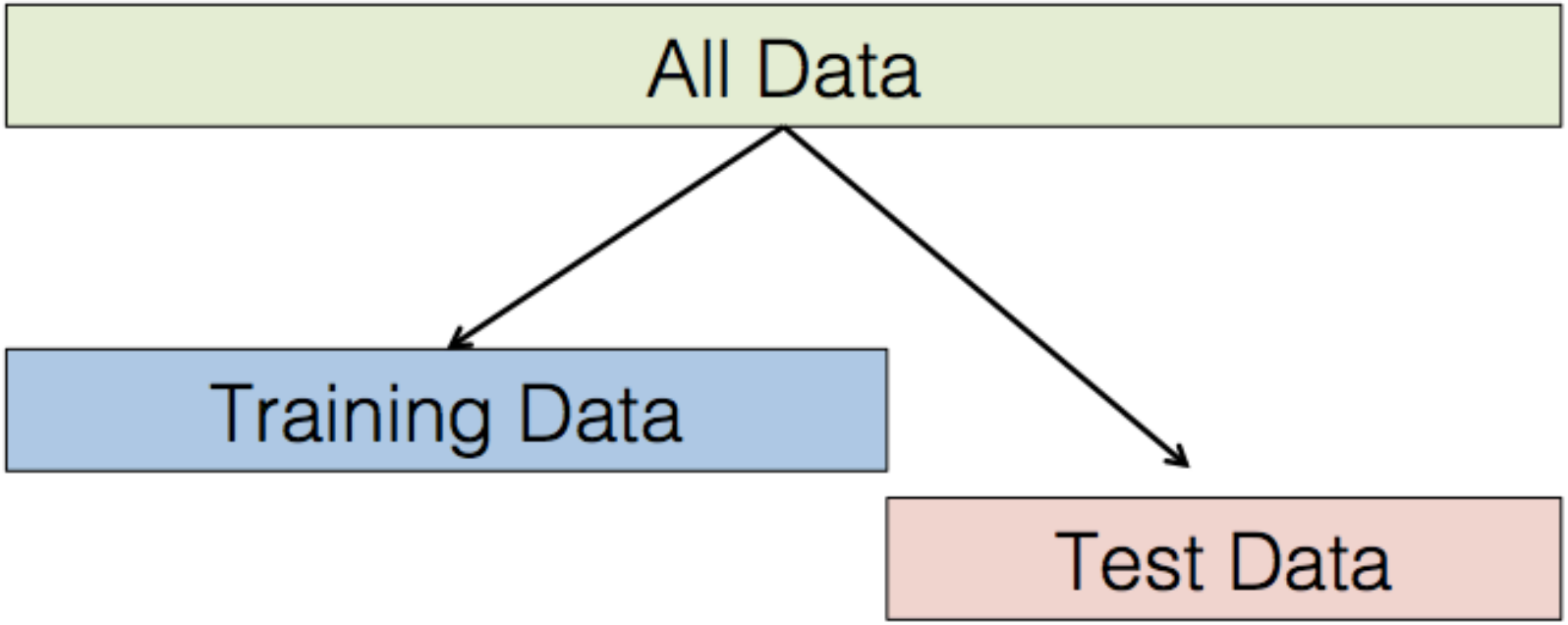
Some Definitions

- Test Set - A collection of examples that is used to assess the performance of a program

- Dimensionality reduction - the process of discovering the explanatory variables that account for the greatest changes in the response variable (also used for data visualization)

- Note: The test set is a similar collection of observations that is used to evaluate the performance of the model using some performance metric.
  - It is important that no observations from the training set are included in the test set.
  - If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

Test Set



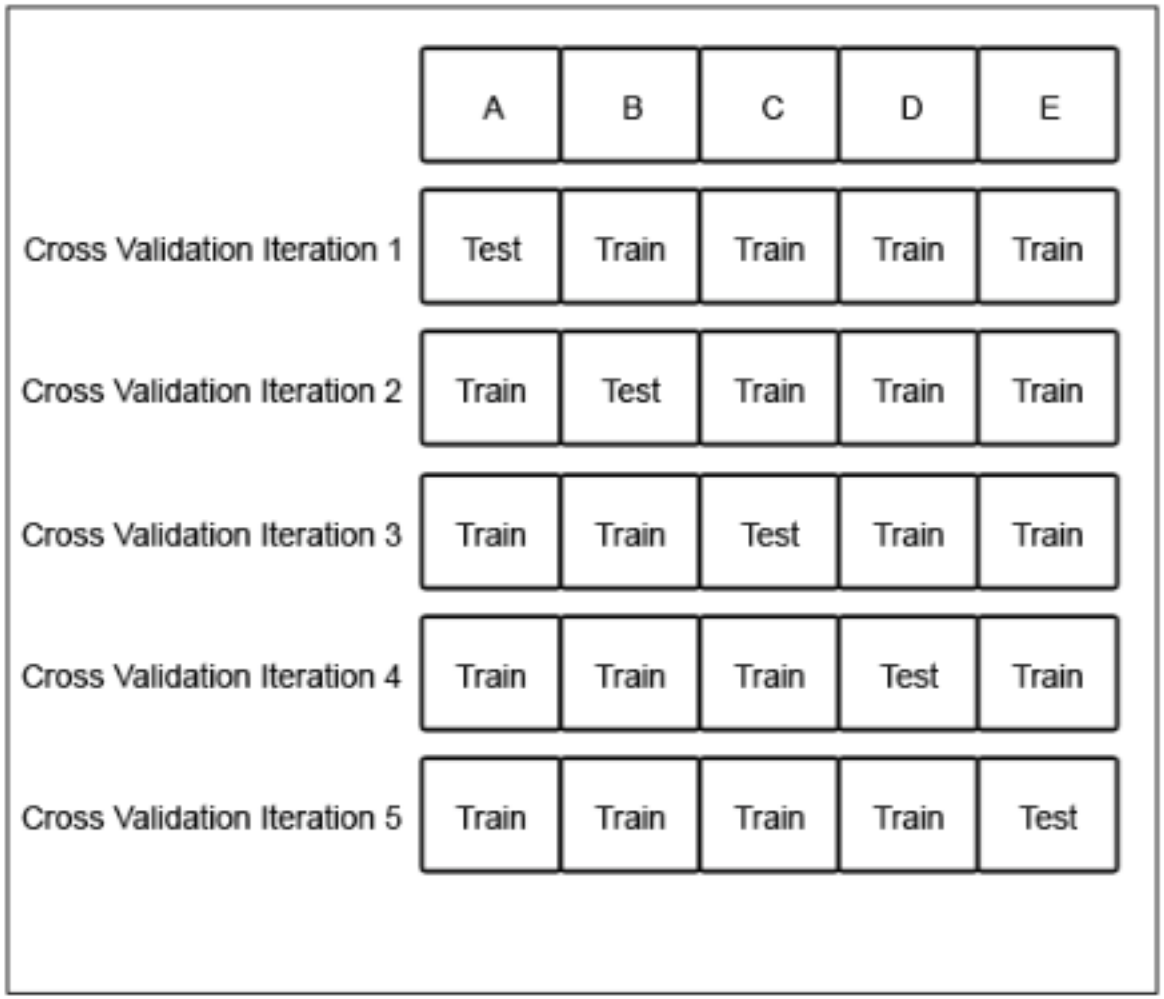
Typically:

- 75% : 25%
- 2/3 : 1/3

Some Definitions

- **Cross-Validation** can be used to train and validate an algorithm on the same data.
- In cross-validation, the training data is partitioned.
- The algorithm is trained using all but one of the partitions, and tested on the remaining partition.
- The partitions are then rotated several times so that the algorithm is trained and evaluated on all of the data.
- The following diagram depicts cross-validation with five partitions or **folds**:

Cross Validation



Cross Validation

- The original dataset is partitioned into five subsets of equal size, labeled **A** through **E**. Initially, the model is trained on partitions **B** through **E**, and tested on partition **A**.
- In the next iteration, the model is trained on partitions **A**, **C**, **D**, and **E**, and tested on partition **B**.
- The partitions are rotated until models have been trained and tested on all of the partitions.
- Cross-validation provides a more accurate estimate of the model's performance than testing a single partition of the data.

Variance and Bias

- Many metrics can be used to measure whether or not a program is learning to perform its task more effectively.
- For supervised learning problems, many performance metrics measure the number of prediction errors.
- There are two fundamental causes of prediction error: a model's **bias** and its **variance**.

Variance and Bias

- Assume that you have many training sets that are all unique, but equally representative of the population.
- A model with a high bias will produce similar errors for an input regardless of the training set it was trained with
- The model biases its own assumptions about the real relationship over the relationship demonstrated in the training data.
- A model with high variance, conversely, will produce different errors for an input depending on the training set that it was trained with.

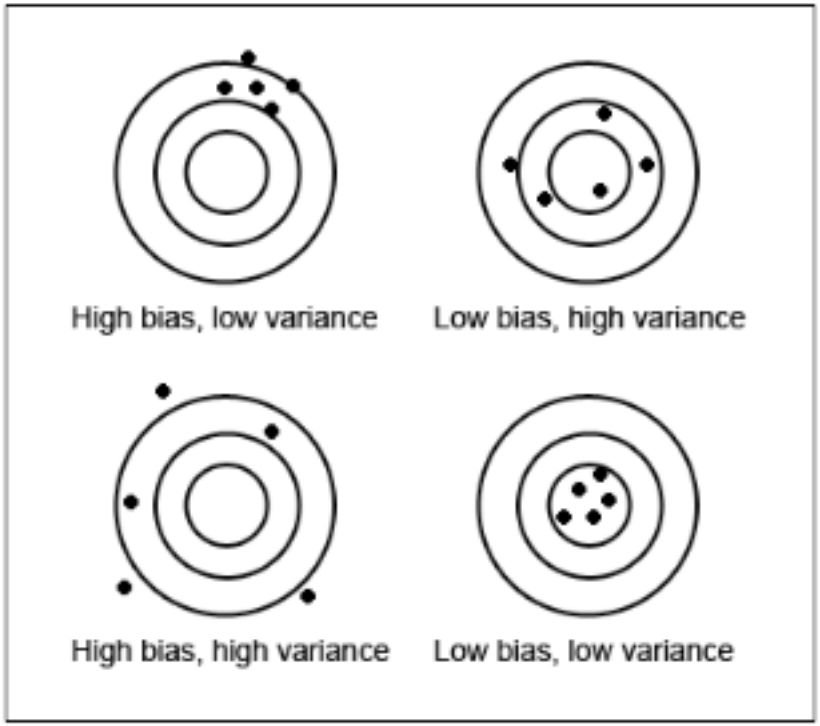
Variance and Bias

- A model with high bias is less flexible
- A model with high variance may be so flexible that it models the noise in the training set.
- A model with high variance over-fits the training data, while a model with high bias under-fits the training data.

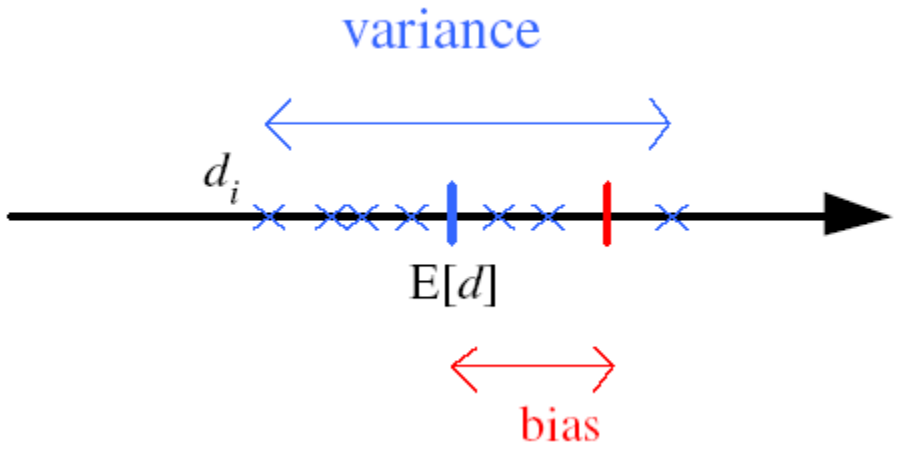
Variance and Bias

- Ideally, a model will have both low bias and variance, but efforts to decrease one will frequently increase the other.
- This is known as the **bias-variance trade-off**. We will discuss the biases and variances of many of the models introduced in this class.

Variance and Bias



Variance and Bias

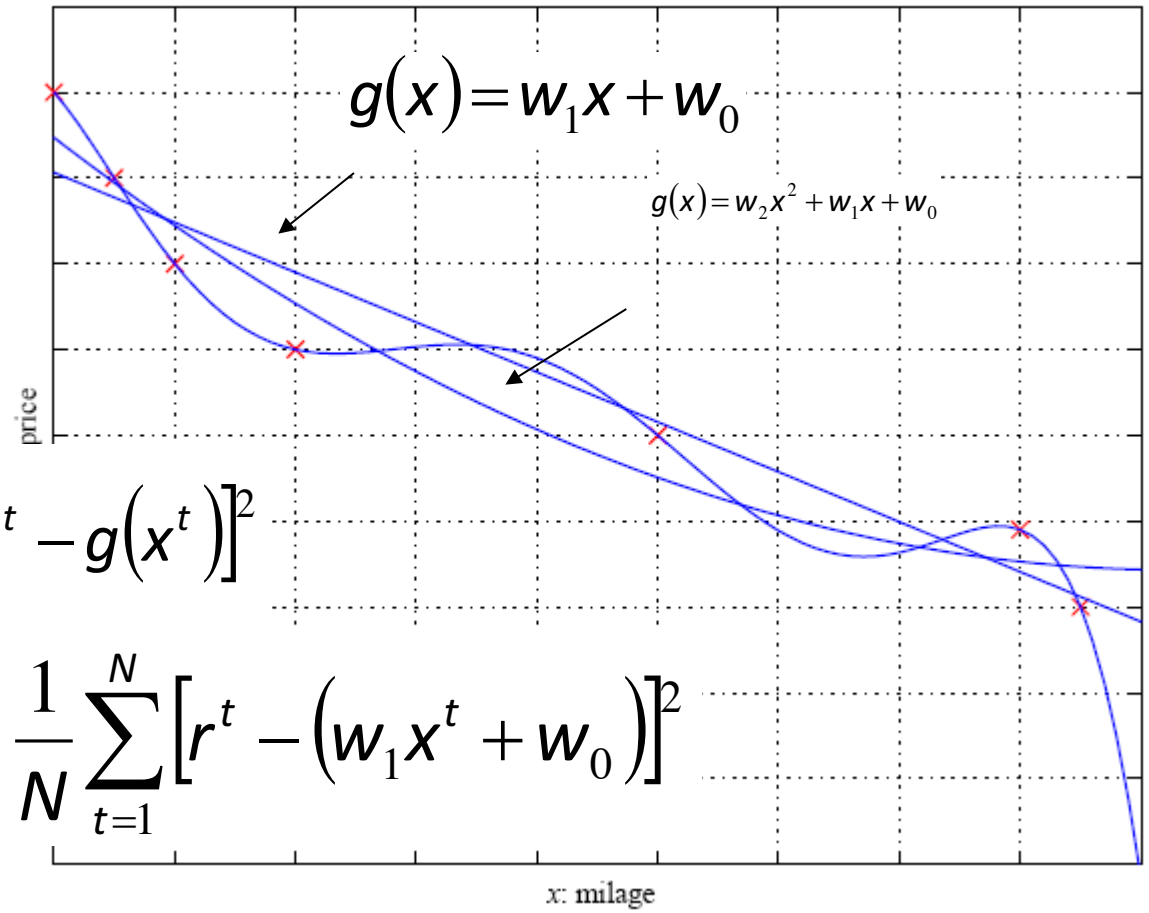


Regression

$$\mathcal{X} = \left\{x^t, r^t\right\}_{t=1}^{N}$$

$$r^t \in \Re$$

$$r^t = f\left(x^t\right) + \varepsilon$$



$$E(g \mid \mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}\left[r^t - g\left(x^t\right)\right]^2$$

$$E(w_1, w_0 \mid \mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}\left[r^t - \left(w_1 x^t + w_0\right)\right]^2$$

Simple Linear Regression

$\hat{y}_i$ is linear in the features

$$\hat{y}_i = f(x_i) = \hat{\beta}_0 + \sum_{j=1}^{p} x_{ij}\hat{\beta}_j$$

$$\hat{Y} = X\hat{\beta}$$

matrix representation

Simple Linear Regression – 2D

- Matrix form

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} \alpha + \beta X_1 \\ \alpha + \beta X_2 \\ \vdots \\ \alpha + \beta X_n \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \times \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

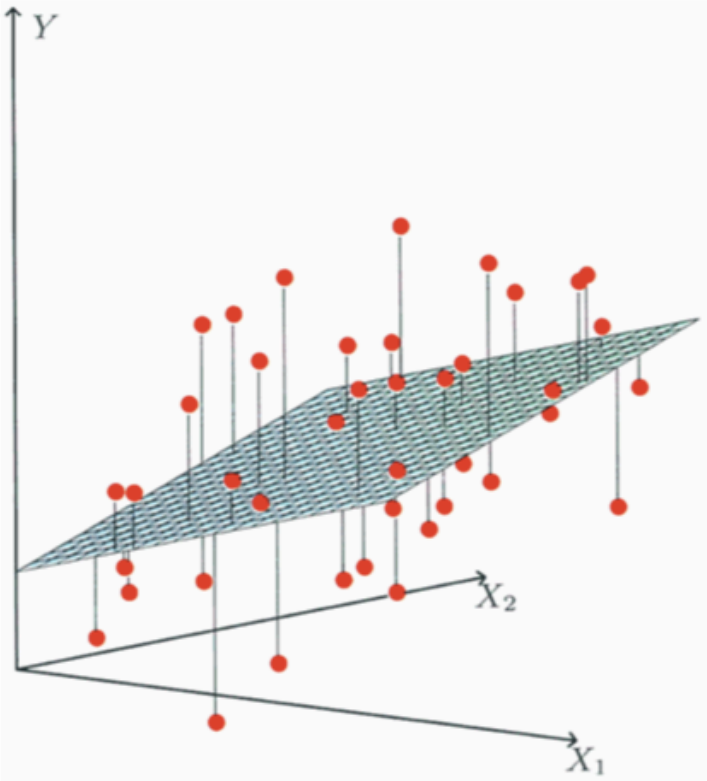
Multiple Linear Regression

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

$$Y = X\beta$$

Linear Regression

- Input "feature" vector $\boldsymbol{x} := (1 \equiv x^{(0)}, x^{(1)}, ..., x^{(d)}) \in \mathbb{R}^{d+1}$
- Real-valued noisy response $y \in \mathbb{R}$.
- Linear regression model:
  $\hat{y} = f_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 x^{(0)} + ... + w_d x^{(d)}$
- Data: $D = (\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$
- Error or loss function:
  Example: Residual sum of squares:
  $\text{Loss}(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2$
- Least squares (LSQ) regression:
  $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \text{Loss}(\boldsymbol{w})$



- Example: Person's weight y as a function of age x^1, height x^2.

* Bishop, via Hutter

Linear Regression

Minimize a loss function to find the β giving the "best fit"

$$\mathcal{L}(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

Optimization

- Optimization is a key component of many machine learning methods.

- We are optimizing is the loss function for the model.
  - For a given set of training data **X** and outcomes **y**, we want to find the model parameters **w** that minimize the total loss over all **X, y**.

Closed Form Solution

- Calculate sum of squared loss (SSL) and determine **w**:

$$\text{SSL} = \sum_{j=1}^{N} (y_j - \sum_{i=0}^{d} w_i \cdot x_i)^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}} \cdot (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\mathbf{y} = \text{vector of all training responses } y_j$$

$$\mathbf{X} = \text{matrix of all training samples } \mathbf{x}_j$$

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

$$\hat{y}_t = \mathbf{w} \cdot \mathbf{x}_t \qquad \text{for test sample } \mathbf{x}_t$$

- Outside of the scope of this class – however we can prove with Calculus that **w** minimizes SSL.

Variance

Variance is a measure of how far a set of values is spread out. If all of the numbers in the set are equal, the variance of the set is zero. A small variance indicates that the numbers are near the mean of the set, while a set containing numbers that are far from the mean and each other will have a large variance. Variance can be calculated using the following equation:

$$\operatorname{var}(x) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$$

Covariance

Covariance is a measure of how much two variables change together. If the value of the variables increase together, their covariance is positive. If one variable tends to increase while the other decreases, their covariance is negative. If there is no linear relationship between the two variables, their covariance will be equal to zero; the variables are linearly uncorrelated but not necessarily independent. Covariance can be calculated using the following formula:

$$\operatorname{cov}(x, y) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Linear Regression in the ML Context

$$SSE = \sum_{i=1}^{n} (e_i)^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Which in turn allows us to estimate σ^2:

$$\hat{\sigma}^2 = \frac{SSE}{n-2}$$

- As well as an important statistic referred to as the coefficient of determination:

$$r^2 = 1 - \frac{SSE}{SST} \qquad SST = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

* Basic Business Statistic, Berenson et. al.

ML Solutions for Regression Coefficients

- Implementing the Closed Form solution for large dimensional regression problems in not practical. Hence the need for other techniques like ML.

- The values of the regression parameters β_0, and β_1 are not known. They are estimated from data.

- β_1 indicates the change in the mean response per unit increase in X.

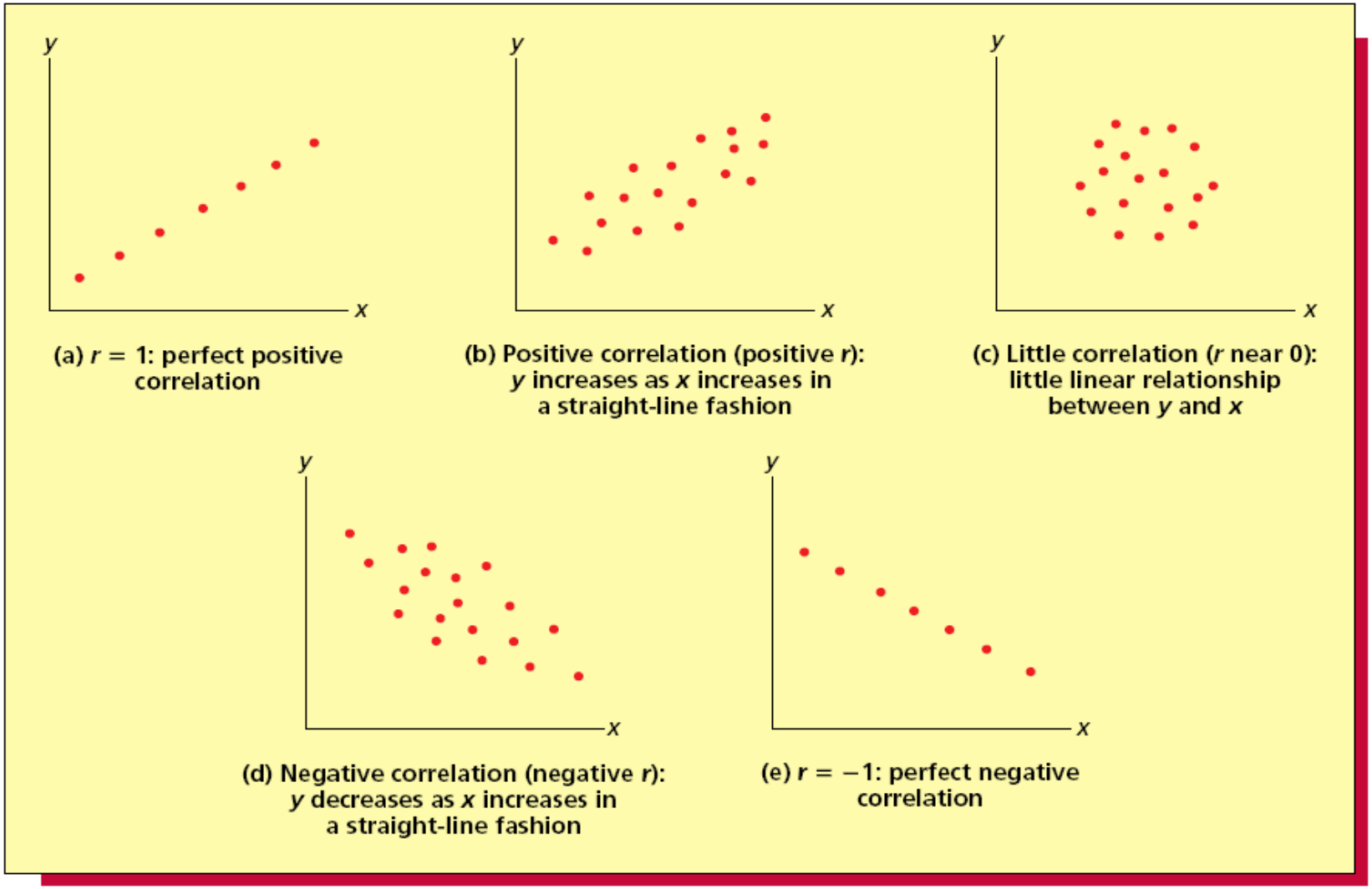
Coefficient of Determination, r^2

- The coefficient of determination is the portion of the total variation in the dependent variable that is explained by variation in the independent variable
- The coefficient of determination is also called r-square and is denoted as r^2

$$r^2 = \frac{SSR}{SST} = \frac{\text{regression sum of squares}}{\text{total sum of squares}}$$

$$0 \leq r^2 \leq 1$$

Coefficient of Determination, r^2



r^2 and our Intuition

- When r close to 0 our hypothesis is not good for making predictions.

- When r close to 1 our hypothesis is makes more reliable predictions.

Scikit-Learn

- The sklearn.linear_model.LinearRegression class is an **estimator**.
- Estimators predict a value based on the observed data.
- In scikit-learn, all estimators implement the fit() and predict() methods.
- The former method is used to learn the parameters of a model, and the latter method is used to predict the value of a response variable
  for an explanatory variable using the learned parameters.
- The fit method of LinearRegression learns the parameters of the following model for simple linear regression:
  - $y=\alpha+\beta x$

Scikits-Learn

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
```

Pizza Example

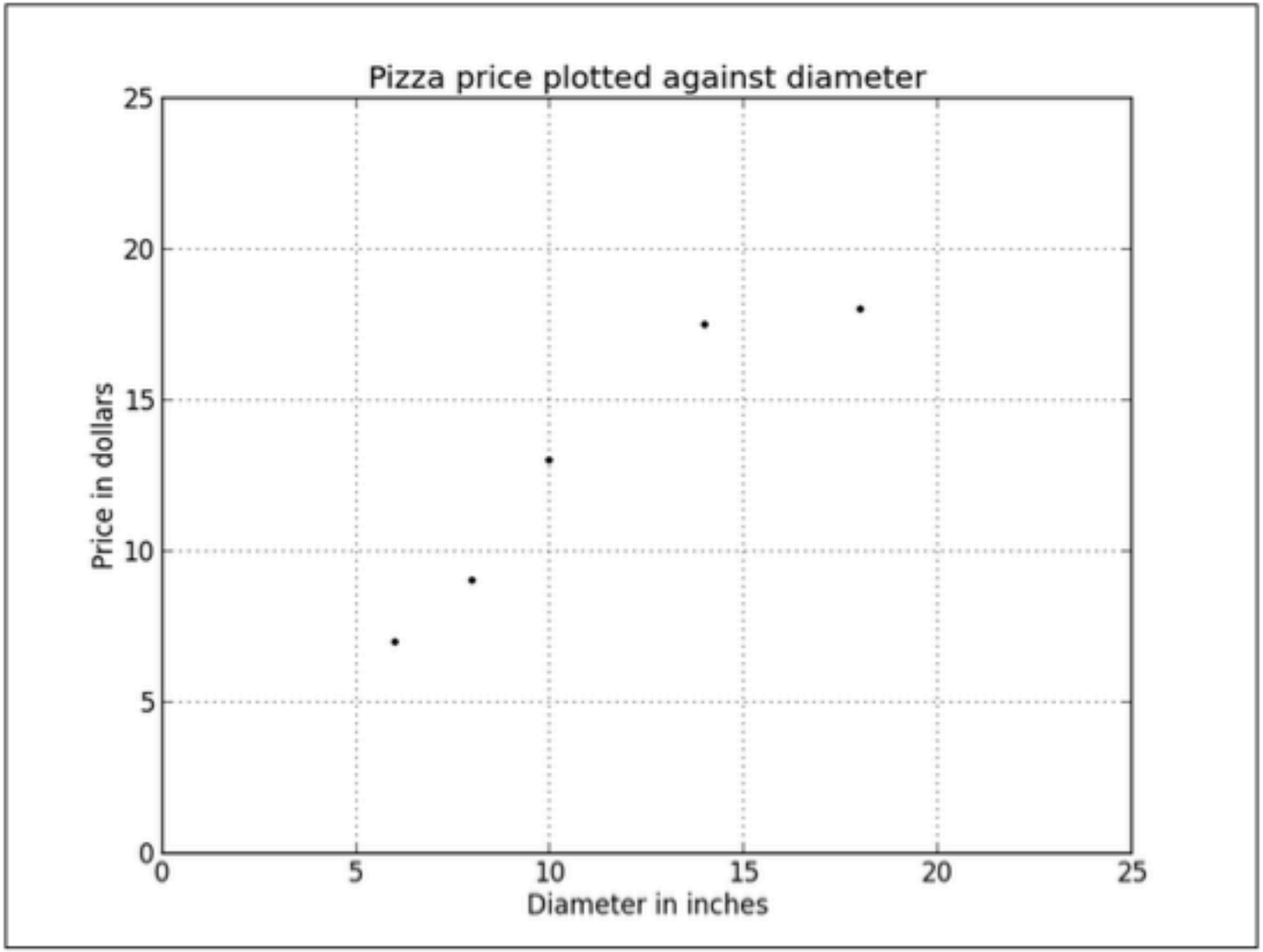
Training instance	Diameter (in inches)	Price (in dollars)
1	6	7
2	8	9
3	10	13
4	14	17.5
5	18	18

We can visualize our training data by plotting it on a graph using `matplotlib`:

```
>>> import matplotlib.pyplot as plt
>>> X = [[6], [8], [10], [14],   [18]]
>>> y = [[7], [9], [13], [17.5], [18]]
>>> plt.figure()
>>> plt.title('Pizza price plotted against diameter')
>>> plt.xlabel('Diameter in inches')
>>> plt.ylabel('Price in dollars')
>>> plt.plot(X, y, 'k.')
>>> plt.axis([0, 25, 0, 25])
>>> plt.grid(True)
>>> plt.show()
```

* Mastering Machine Learning with scikit-learn - Hackeling

Pizza Example



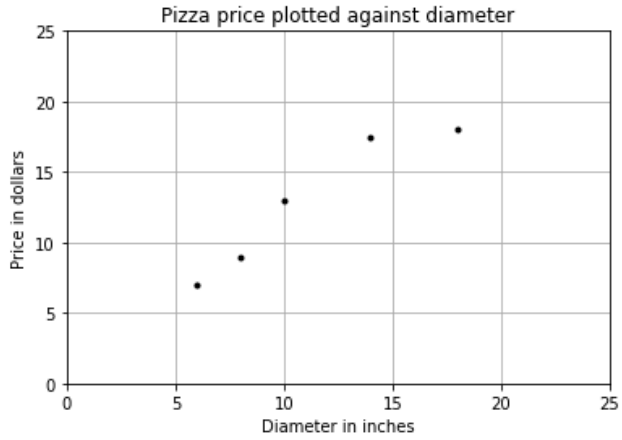
Pizza Example

jupyter Simple Linear Regression Last Checkpoint: 3 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

In [2]:
```
import matplotlib.pyplot as plt
X = [[6], [8], [10], [14],   [18]]
y = [[7], [9], [13], [17.5], [18]]
plt.figure()
plt.title('Pizza price plotted against diameter')
plt.xlabel('Diameter in inches')
plt.ylabel('Price in dollars')
plt.plot(X, y, 'k.')
plt.axis([0, 25, 0, 25])
plt.grid(True)
plt.show()
```



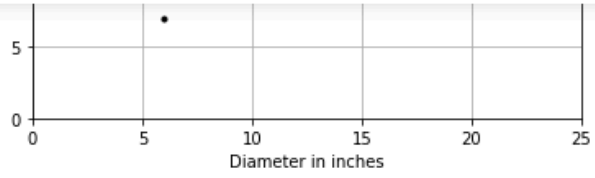
In [ ]:

Pizza Example

jupyter Simple Linear Regression Last Checkpoint: 13 minutes ago (unsaved changes) Logout

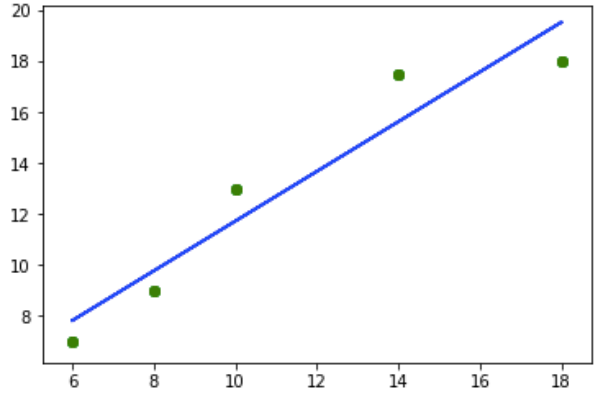
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code



In [12]:
```
from sklearn.linear_model import LinearRegression
# Training data
X = [[6], [8], [10], [14],   [18]]
y = [[7], [9], [13], [17.5], [18]]
# Create and fit the model
model = LinearRegression()
model.fit(X, y)
plt.plot(X, model.predict(X),color='b')
plt.show()
```

```
A 12" pizza should cost:  [[ 13.68103448]]
```



In [ ]:

Pizza Example

$$SS_{res} = \sum_{i=1}^{n} \left(y_i - f(x_i) \right)^2$$

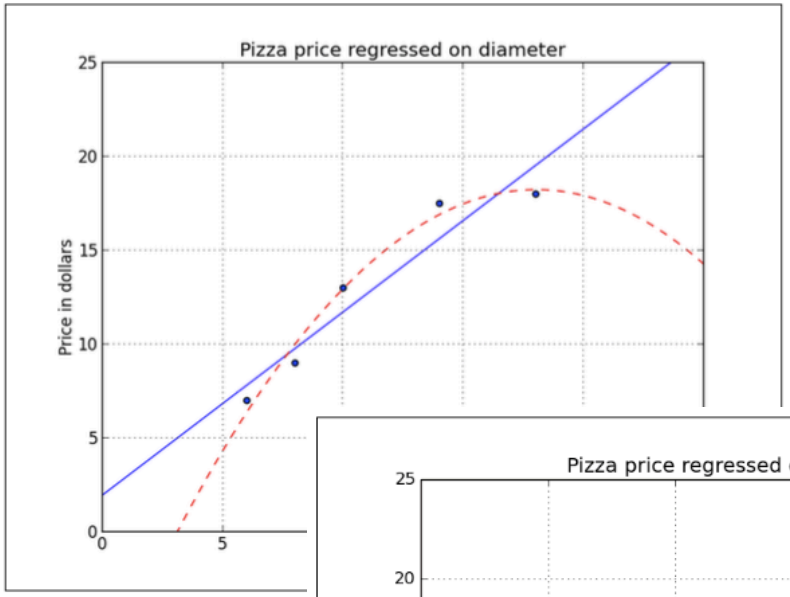
- We can produce the best pizza-price predictor by minimizing the sum of the residuals.
- That is, our model fits if the values it predicts for the response variable are close to the observed values for all of the training examples. This measure of the model's fitness is called the **residual sum of squares** cost function. Formally, this function assesses the fitness of a model by summing the squared residuals for all of our training examples. The residual sum of squares is calculated with the formula in the following equation, where y_i is the observed value and $f(x_i)$ is the predicted value:

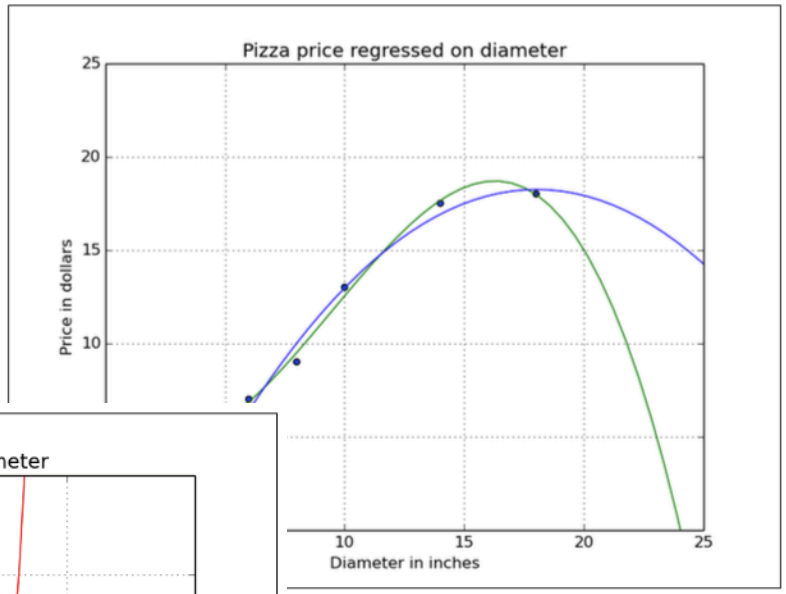
Polynomial Linear Regression

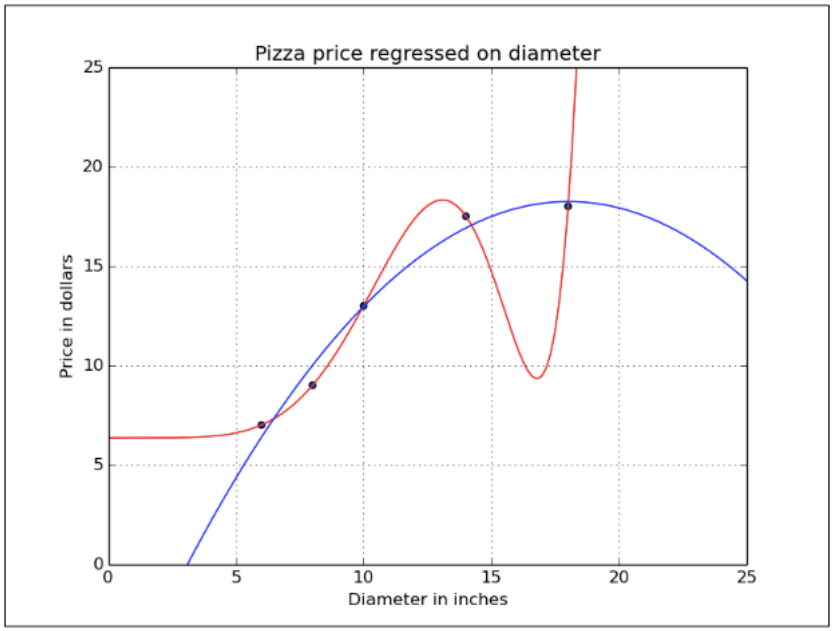
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Toward More Complex Models

Polynomial Linear Regression







Scikit-Learn

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

Regression Types

Simple Linear Regression

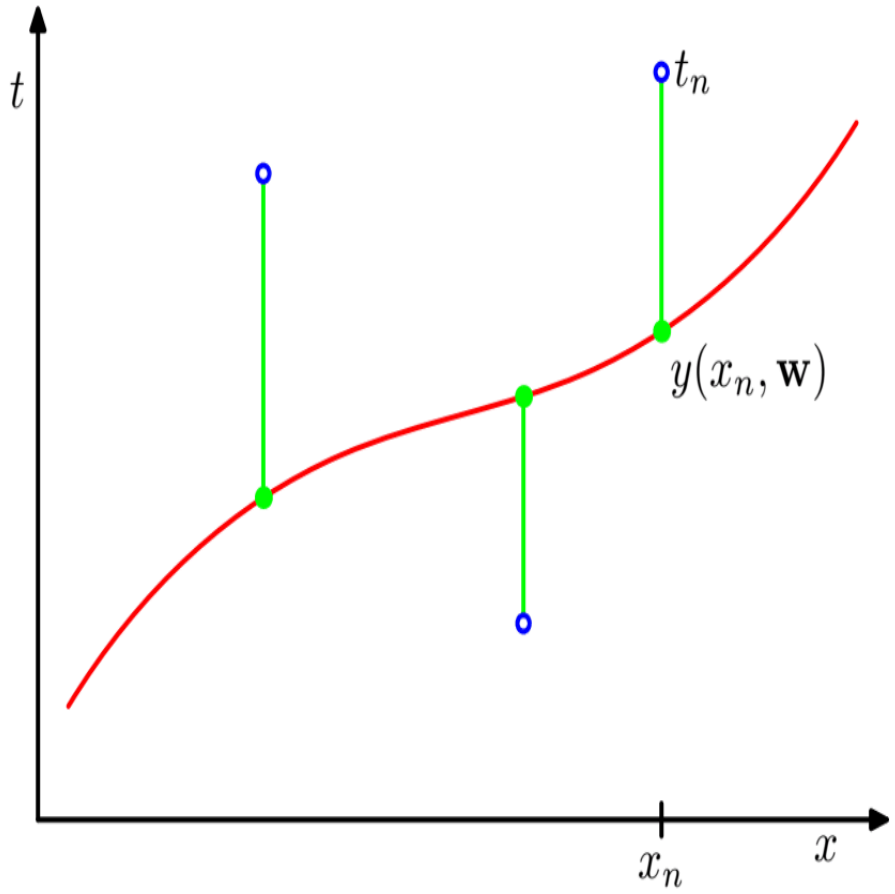
$$y = b_0 + b_1 x_1$$

Multiple Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

Polynomial Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$


t
t_n
$y(x_n, \mathbf{w})$
x_n
x

Linear Basis Function Models

- The estimation of a regression function of one or more independent variables *f*(x) can be modeled as a linear combination of a family of basis functions

$$f(x) = \sum_{k=1}^{K} \beta_k \phi_k(x),$$

Linear Basis Function Models

- Linear combination of fixed nonlinear basis functions

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

- parameter $\mathbf{w} = (w_0, \ldots, w_{M-1})^T$
- basis functions $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{M-1})^T$
- convention $\phi_0(\mathbf{x}) = 1$
- w_0 is the bias parameter

* Bishop, via Hutter

Math Essentials

- Addition of two matrices
  - matrices must be same size
  - add corresponding elements:

$$c_{ij} = a_{ij} + b_{ij}$$

  - result is a matrix of same size

$$\mathbf{C} = \mathbf{A} + \mathbf{B} =$$

$$\begin{pmatrix} a_{11} + b_{11} & \cdots & a_{1n} + b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & \cdots & a_{mn} + b_{mn} \end{pmatrix}$$

- Scalar multiplication of a matrix
  - multiply each element by scalar:

$$b_{ij} = d \cdot a_{ij}$$

  - result is a matrix of same size

$$\mathbf{B} = d \cdot \mathbf{A} =$$

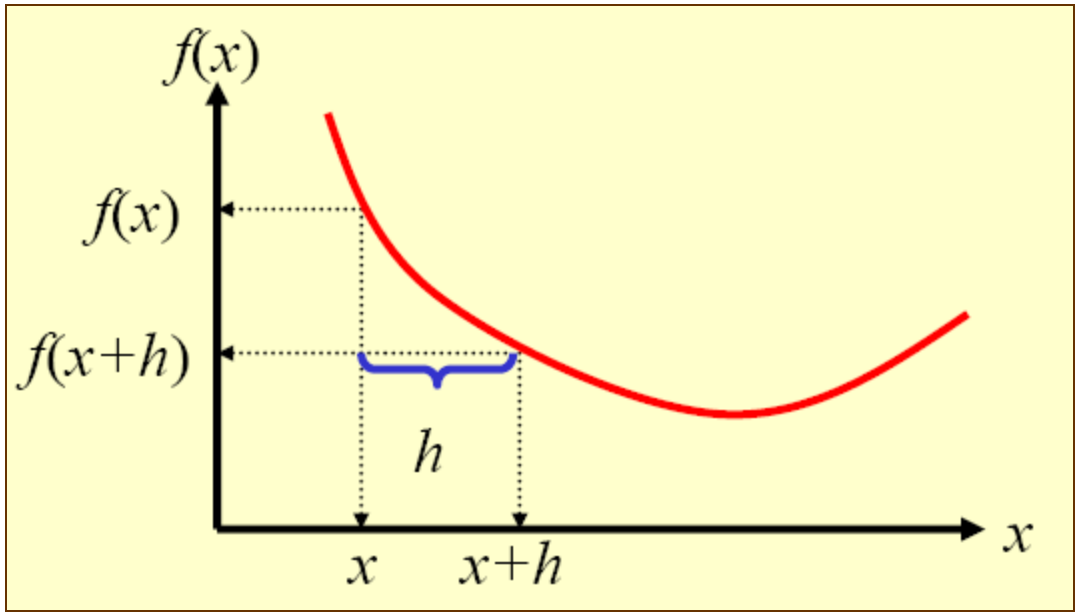
$$\begin{pmatrix} d \cdot a_{11} & \cdots & d \cdot a_{1n} \\ \vdots & \ddots & \vdots \\ d \cdot a_{m1} & \cdots & d \cdot a_{mn} \end{pmatrix}$$

Math Essentials

The derivative of $f: R \rightarrow R$ is a function $f': R \rightarrow R$ st.

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

if the limit exists.



Math Essentials

- The gradient is an important concept from calculus in the context of machine learning.
- Gradients generalize derivatives to scalar functions of several variables.

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \text{i.e.} \quad [\nabla f]_i = \frac{\partial f}{\partial x_i}$$

Math Essentials

- Gradient is a vector
  - Each element is the slope of function along direction of one of variables
  - Each element is the partial derivative of function with respect to one of variables

$$\nabla f(\mathbf{x}) = \nabla f(x_1, x_2, \ldots, x_d) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1} \quad \frac{\partial f(\mathbf{x})}{\partial x_2} \quad \cdots \quad \frac{\partial f(\mathbf{x})}{\partial x_d} \right]$$

  - Example:

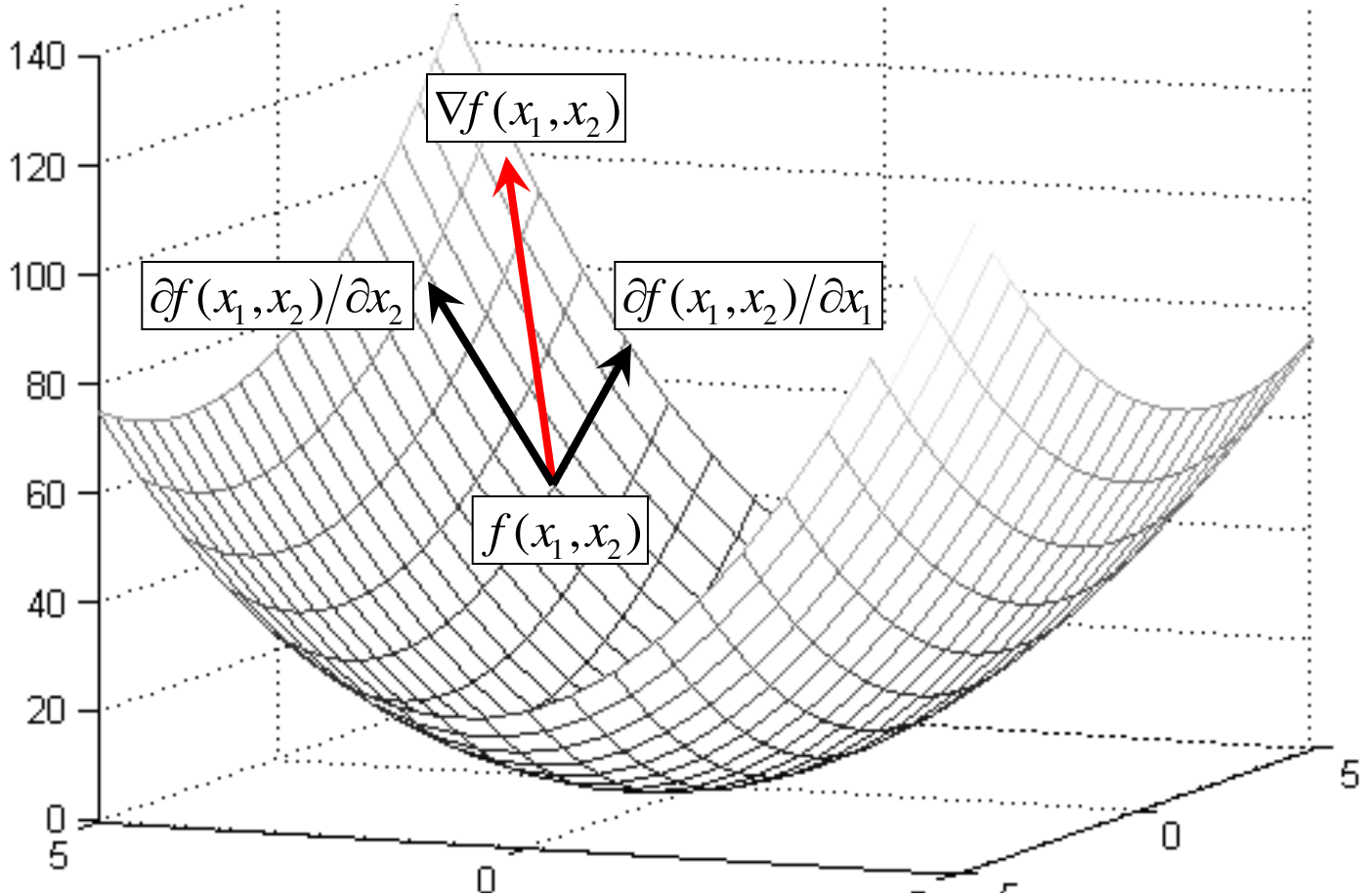
$$f(\mathbf{x}) = f(x_1, x_2) = {x_1}^2 + x_1 x_2 + 3{x_2}^2$$

$$\nabla f(\mathbf{x}) = \nabla f(x_1, x_2) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1} \quad \frac{\partial f(\mathbf{x})}{\partial x_2} \right] = \left[2x_1 + x_2 \quad x_1 + 6x_2 \right]$$

* Howbert

Math Essentials

- Gradient vector points in direction of steepest ascent of function



Math Essentials

The **Hessian** matrix of $f : \mathbb{R}^d \to \mathbb{R}$ is a matrix of second-order partial derivatives:

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix} \quad \text{i.e.} \quad [\nabla^2 f]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

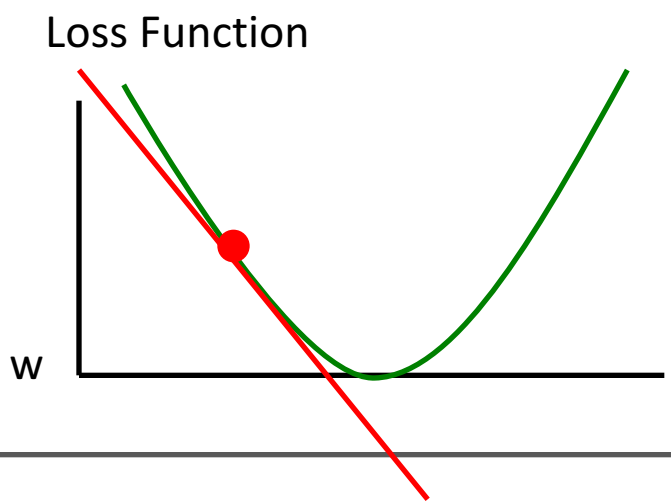
Basic Intro: Gradient Descent

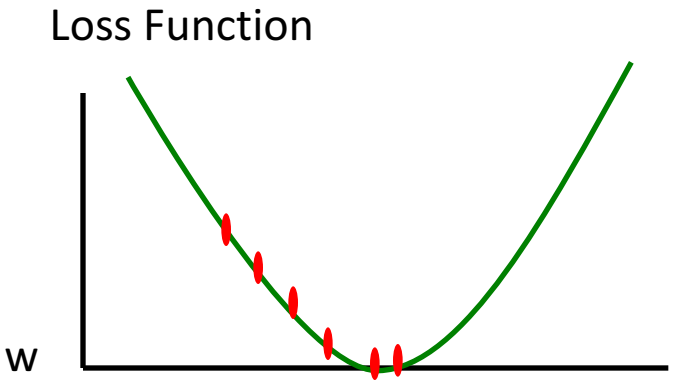
Partial derivatives provide the slope to direct the solution toward minimization of the loss function.

- Pick initial solution
- Repeat until optimized {
  - Evaluate slope and minima conditionality
  - Pick dimension(s)
  - Move a small amount (learning step) in the direction decreasing loss

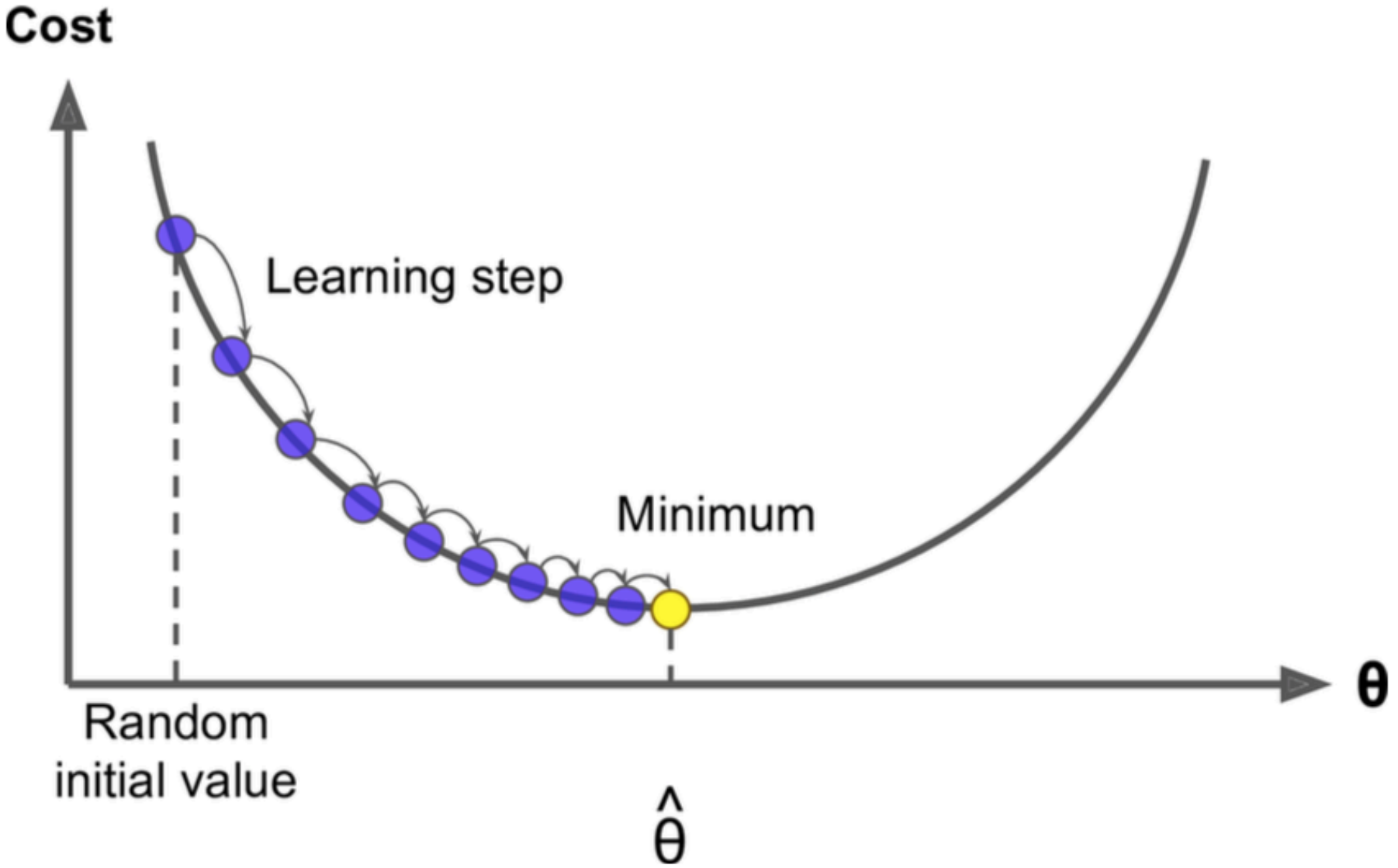
  }

$$\theta_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$$
$$\theta_2 := \theta_2 - \alpha \frac{\partial J}{\partial \theta_2}$$
$$\vdots$$
$$\theta_k := \theta_k - \alpha \frac{\partial J}{\partial \theta_k}$$

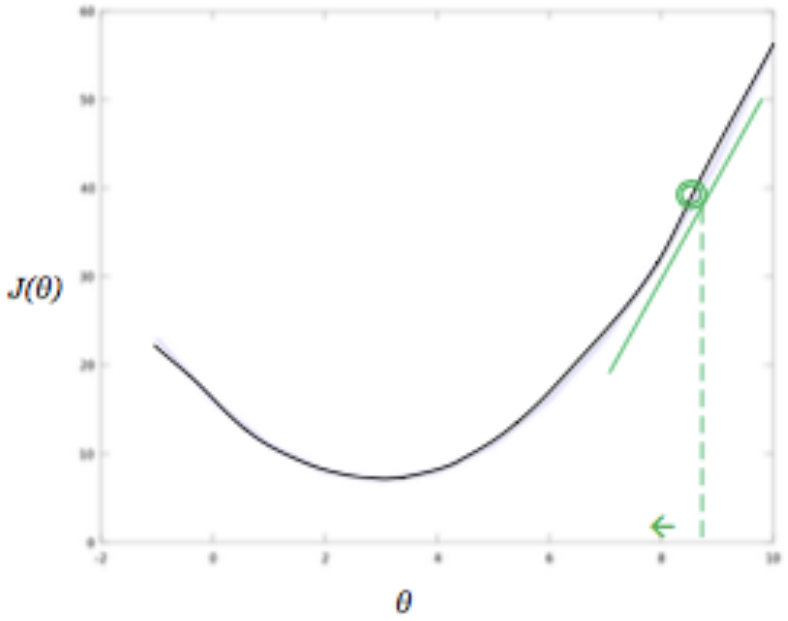


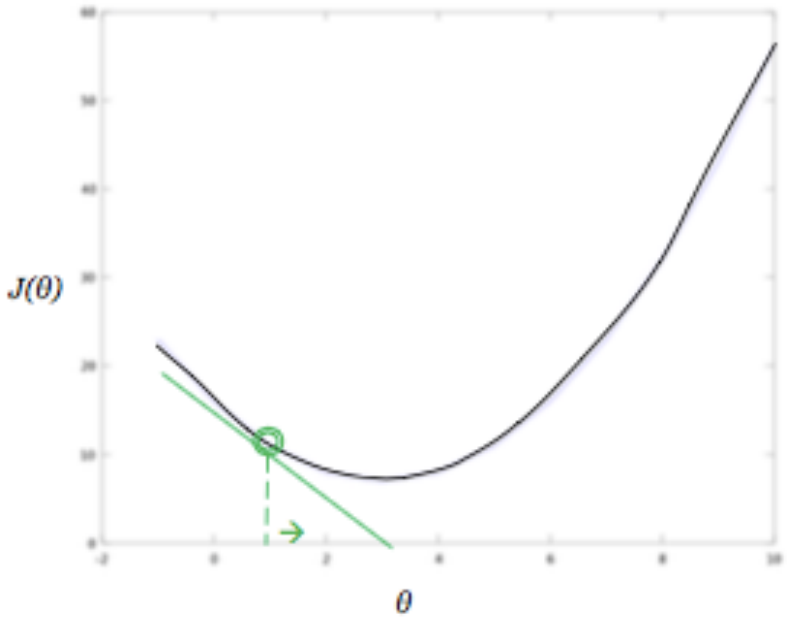


Basic Intro: Gradient Descent



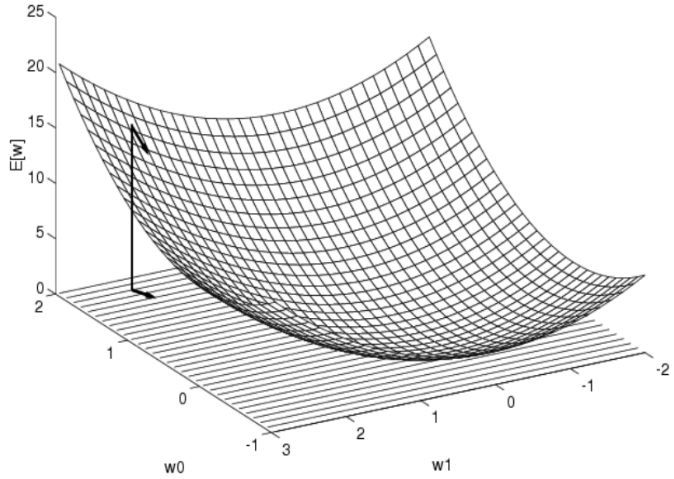
Basic Intro: Gradient Descent





* A Beginners Tutorial for Machine Learning Beginners, Hao

Basic Intro: Gradient Decent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Basic Intro: Gradient Descent

- Simple concept: follow the gradient *downhill*
- Process:
  1. Pick a starting position: $\mathbf{x}^0 = (x_1, x_2, \ldots, x_d)$
  2. Determine the descent direction: $- \nabla f(\mathbf{x}^t)$
  3. Choose a learning rate: η
  4. Update your position: $\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \cdot \nabla f(\mathbf{x}^t)$
  5. Repeat from 2) until stopping criterion is satisfied
- Typical stopping criteria
  - $\nabla f(\mathbf{x}^{t+1}) \sim 0$
  - some validation metric is optimized

* Howbert

Basic Intro: Gradient Descent

Batch gradient: use error $E_D(\mathbf{w})$ over entire training set D

Do until satisfied:

1. Compute the gradient $\nabla E_D(\mathbf{w}) = \left[\frac{\partial E_D(\mathbf{w})}{\partial w_0} \cdots \frac{\partial E_D(\mathbf{w})}{\partial w_n}\right]$
2. Update the vector of parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_D(\mathbf{w})$

Basic Intro: Gradient Descent

Stochastic gradient: use error $E_d(\mathbf{w})$ over single examples $d \in D$

Do until satisfied:

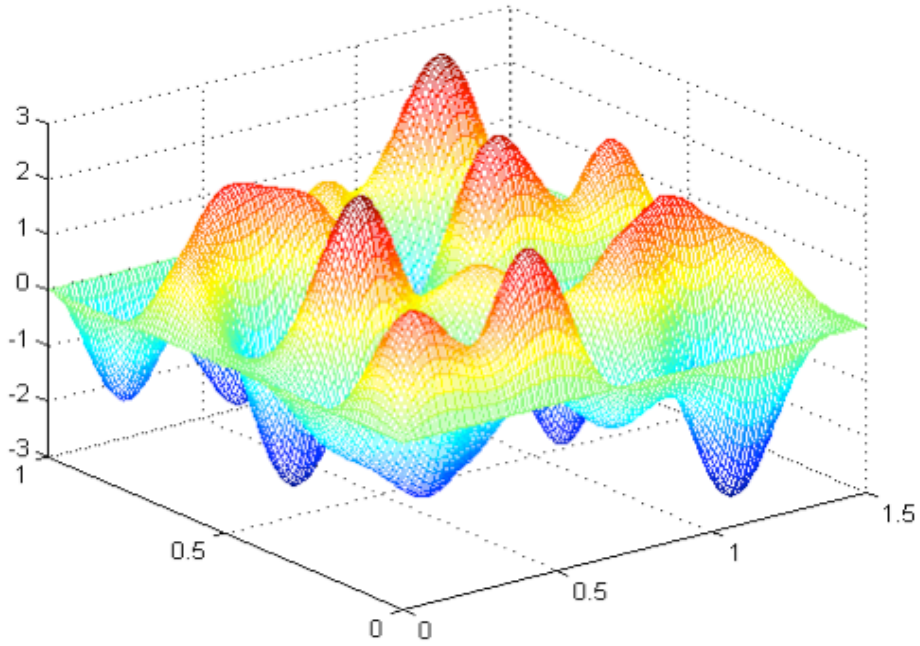
1. Choose (with replacement) a random training example $d \in D$
2. Compute the gradient just for d: $\nabla E_d(\mathbf{w}) = \left[\frac{\partial E_d(\mathbf{w})}{\partial w_0} \ldots \frac{\partial E_d(\mathbf{w})}{\partial w_n} \right]$
3. Update the vector of parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_d(\mathbf{w})$

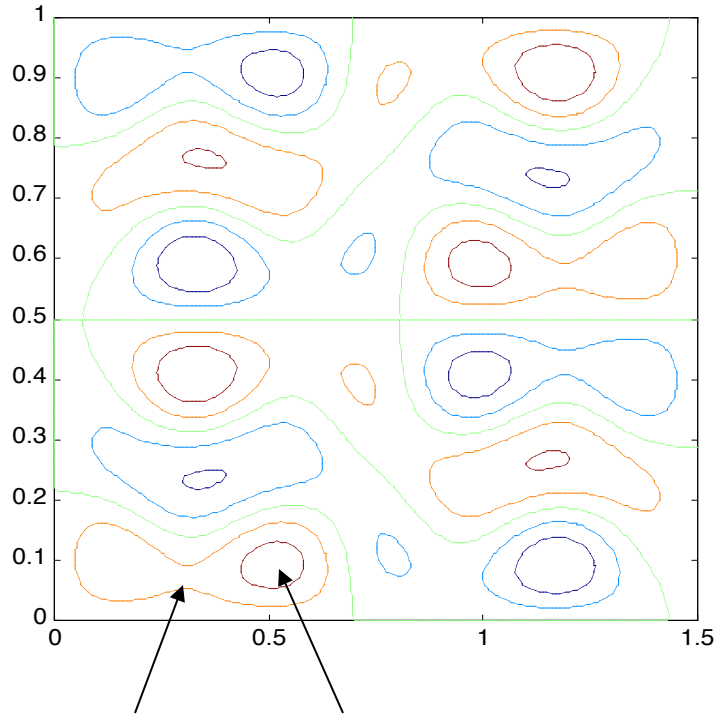
Basic Intro: Gradient Descent

- An issue with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.

- An alternative is *Stochastic Gradient Descent, which* picks a random sample in the training set at every step and computes the gradients based only on that single instance.

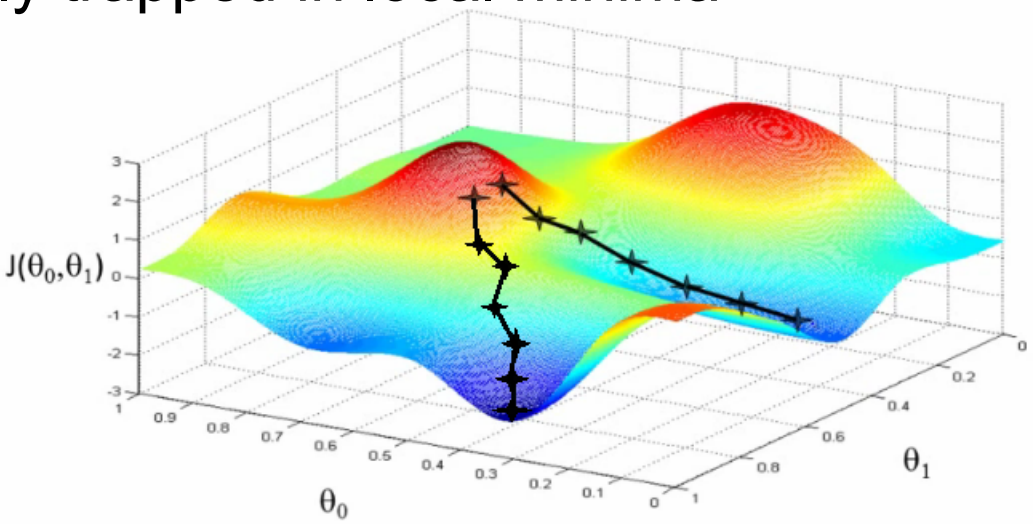
Basic Intro: Gradient Descent





Basic Intro: Gradient Descent

- Problems:
  - Choosing step size
    - too small → convergence is slow and inefficient
    - too large → may not converge
  - Can get stuck on "flat" areas of function
  - Easily trapped in local minima



Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning