Tutorial 1

Overview

Notation and Convention

Notation and Convention

- Vectors $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ are denoted by lower-case letters.

Notation and Convention

- Vectors $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ are denoted by lower-case letters.
- Matrices $\mathrm{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ are denoted by upper-case letters.

Notation and Convention

- Vectors $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ are denoted by lower-case letters.
- Matrices $\mathrm{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ are denoted by upper-case letters.
- X above is a 2 by 3 matrix ("nrow by ncol").

Notation and Convention

- Vectors $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ are denoted by lower-case letters.
- Matrices $\mathrm{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ are denoted by upper-case letters.
- X above is a 2 by 3 matrix ("nrow by ncol").
- Vectors are by default column vectors (ie. $d \times 1$ matrices)

Notation and Convention

- Vectors $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ are denoted by lower-case letters.
- Matrices $\mathrm{X} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ are denoted by upper-case letters.
- X above is a 2 by 3 matrix ("nrow by ncol").
- Vectors are by default column vectors (ie. $d \times 1$ matrices)
- One can sometimes save space by using transpose operator and write a column vector on a single line.

$$b = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 8 \end{bmatrix}^T$$

Data Types

matrices and vectors/arrays

- `A = [ 1 2 3 ; 4 5 6 ; 7 8 9] % a 3x3 matrix`
- `b = [1 2 3] % a row vector`
- `c = [1; 2; 3] % a column vector`
- `d = [2] % this is a scalar, equivalent to ‘‘d = 2’’`

Data Types

matrices and vectors/arrays

- `A = [ 1 2 3 ; 4 5 6 ; 7 8 9] % a 3x3 matrix`
- `b = [1 2 3] % a row vector`
- `c = [1; 2; 3] % a column vector`
- `d = [2] % this is a scalar, equivalent to ‘‘d = 2’’`

multiplication operator is overloaded

- `A*d` matrix-scalar multiplication
- `A*c` matrix-vector multiplication
- `A*A` matrix-matrix multiplication

Data Types

matrices and vectors/arrays

- `A = [ 1 2 3 ; 4 5 6 ; 7 8 9] % a 3x3 matrix`
- `b = [1 2 3] % a row vector`
- `c = [1; 2; 3] % a column vector`
- `d = [2] % this is a scalar, equivalent to ''d = 2''`

multiplication operator is overloaded

- `A*d` matrix-scalar multiplication
- `A*c` matrix-vector multiplication
- `A*A` matrix-matrix multiplication

solving linear systems

- `A\b` solves the linear system $Ax = b$

Data Types

accessing elements

- `c(1)` note the bracket type (parenthesis) and one-indexing
- `A(1,2)` gives a scalar
- `A(1,:)` gives a row vector (slicing)
- `A(2:3,:)` gives a size-2 row vector
- `A(2:end,:)` same as previous
- `b=[1,3] ; A(b,:)` in case you want a non-contiguous slice

Data Types

accessing elements

- `c(1)` note the bracket type (parenthesis) and one-indexing
- `A(1,2)` gives a scalar
- `A(1,:)` gives a row vector (slicing)
- `A(2:3,:)` gives a size-2 row vector
- `A(2:end,:)` same as previous
- `b=[1,3] ; A(b,:)` in case you want a non-contiguous slice

"booleans"

- In MATLAB, true and false are almost synonymous with 1 and 0. (ie. `true == 1` and `false == 0` both return `1`)
- `A([true, false, true],:) == A([1,3],:)` (ie. a mask)
- `true` and `false` can be used as 1 and 0 (`true * 10 == 10`)

Data Types

accessing elements

- `c(1)` note the bracket type (parenthesis) and one-indexing
- `A(1,2)` gives a scalar
- `A(1,:)` gives a row vector (slicing)
- `A(2:3,:)` gives a size-2 row vector
- `A(2:end,:)` same as previous
- `b=[1,3] ; A(b,:)` in case you want a non-contiguous slice

"booleans"

- In MATLAB, true and false are almost synonymous with 1 and 0. (ie. `true == 1` and `false == 0` both return `1`)
- `A([true, false, true],:) == A([1,3],:)` (ie. a mask)
- `true` and `false` can be used as 1 and 0 (`true * 10 == 10`)
- caveat: `A([1,0,1],:)` fails. Nice try MATLAB.

Data Exploration Basics

- Use builtin functions to read data

```
data = csvread('london2012.csv');
```

- Each row is an athlete of the London 2012 summer Olympics.

Data Exploration Basics

- Use builtin functions to read data

```
data = csvread('london2012.csv');
```

- Each row is an athlete of the London 2012 summer Olympics.
- Check dataset size

```
size(data)
```

Data Exploration Basics

- Use builtin functions to read data

```
data = csvread('london2012.csv');
```

- Each row is an athlete of the London 2012 summer Olympics.
- Check dataset size

```
size(data)
```

- This dataset has the following 7 descriptive features:

$$\begin{pmatrix} \text{Age, Height, Weight, Gender(1==female)} \\ \text{\# Bronze, \# Silver, \# Gold Medals} \end{pmatrix}$$

Histogram

- Let's visualize some information.
- What was the distribution of age?

Histogram

- Let's visualize some information.
- What was the distribution of age?
- Plot a histogram with 20 bins

```
age = data(:,1); hist(age,20)
```

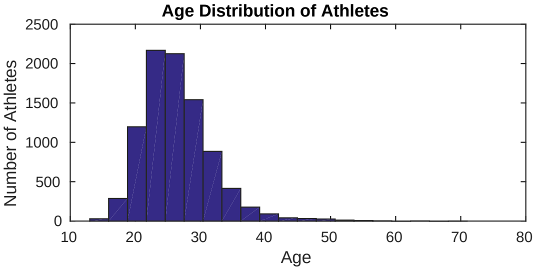
Histogram

- Let's visualize some information.
- What was the distribution of age?
- Plot a histogram with 20 bins

```
age = data(:,1); hist(age,20)
```

- Add some axis labels and a title

```
xlabel('Age'); ylabel('Number of Athletes')
title('Age Distribution of Athletes')
```



Scatterplot

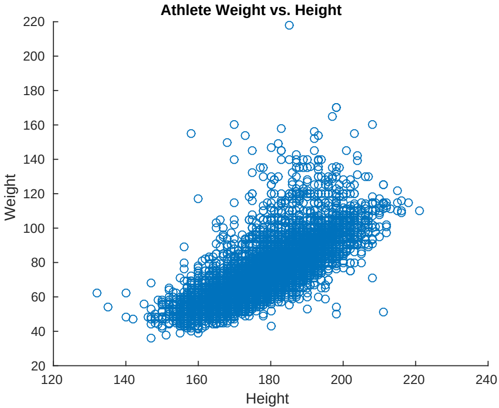
- Plot a scatterplot of height vs. weight

```
height = data(:,2); weight = data(:,3)
scatter(height, weight)
```

Scatterplot

- Plot a scatterplot of height vs. weight

```
height = data(:,2); weight = data(:,3)
scatter(height, weight)
```



Scatterplot

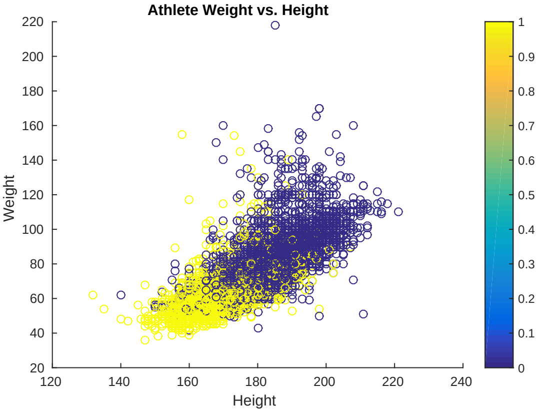
- `scatter` has options for marker size and color.

```
gender = data(:,4)
scatter(height, weight, [], gender)
```

Scatterplot

- `scatter` has options for marker size and color.

```
gender = data(:,4)
scatter(height, weight, [], gender)
```



Boxplot

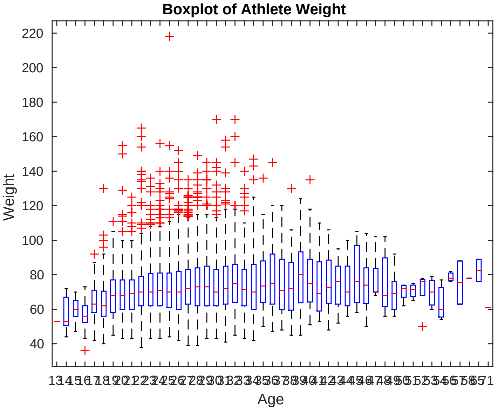
- Plots the first, second, and third quartiles.
- A boxplot of weight for each age:

```
boxplot(weight, age)
```

Boxplot

- Plots the first, second, and third quartiles.
- A boxplot of weight for each age:

```
boxplot(weight, age)
```



Boxplot

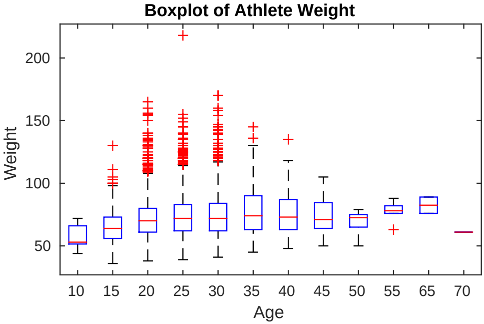
- Okay that was messy. Let's first discretize age (ie. bin/bucket) into groups of 5.
- This rounds everybody's age down to nearest fifth:

```
dage = discretize(age, 10:5:80)*5;
boxplot(weight, dage)
```

Boxplot

- Okay that was messy. Let's first discretize age (ie. bin/bucket) into groups of 5.
- This rounds everybody's age down to nearest fifth:

```
dage = discretize(age, 10:5:80)*5;
boxplot(weight, dage)
```



When In Doubt

- If you know what function to use but don't know how to use it, check the `help` command. e.g.

```
>> help plot
```

- Otherwise, use online resources (Google, Piazza, etc.)

Probability Notation

Probability Notation

- Events - "Let A be the event that the coin lands on heads"

Probability Notation

- Events - "Let A be the event that the coin lands on heads"
- Random variables - "Let X be the result of the coin flip, where 1 and 0 would represent *heads* and *tails* respectively"

Probability Notation

- Events - “Let A be the event that the coin lands on heads”
- Random variables - “Let X be the result of the coin flip, where 1 and 0 would represent *heads* and *tails* respectively”
- Assigning probabilities to events - e.g. $P(A), P(X = 1)$

Probability Notation

- Events - “Let A be the event that the coin lands on heads”
- Random variables - “Let X be the result of the coin flip, where 1 and 0 would represent *heads* and *tails* respectively”
- Assigning probabilities to events - e.g. $P(A), P(X = 1)$
- $P(X = \cdot)$ (or $P_X(\cdot)$) is a function that, when given a value x, returns the probability of the event $(X = x)$

$$P(X = x) = P_X(x) = 0.5^x(1 - 0.5)^{1-x}$$

Probability Notation

- Events - "Let A be the event that the coin lands on heads"
- Random variables - "Let X be the result of the coin flip, where 1 and 0 would represent *heads* and *tails* respectively"
- Assigning probabilities to events - e.g. $P(A), P(X = 1)$
- $P(X = \cdot)$ (or $P_X(\cdot)$) is a function that, when given a value x, returns the probability of the event $(X = x)$

$$P(X = x) = P_X(x) = 0.5^x(1 - 0.5)^{1-x}$$

or

$$P_X(x) = \begin{cases} 0.5 & x = 0 \\ 0.5 & x = 1 \end{cases}$$

Exercise 1: Conditional Probability Review

- Flip 2 coins. If I tell you that the first coin landed heads, what is the probability that the second coin landed heads?

- If I instead tell you that at least one coin landed heads, what is the probability that both coins land heads?

Exercise 2: Why You Should Go To Tutorials

Suppose 80% of students who get above A goes to all tutorials, and 80% of students who get below A do not go to all tutorials. Suppose only 20% of students get above A.
What is the probability of a student who goes to all tutorials getting an A in the course? (Hint: much higher than 0.2)

Reminder - Bayes' Rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

(A and B can be either events or random variables.)

Definition

The notation

$$g(n) = O(f(n))$$

means "for all large n, $g(n) \leq cf(n)$ for some constant $c > 0$".

Definition

The notation

$$g(n) = O(f(n))$$

means "for all large n, $g(n) \leq cf(n)$ for some constant $c > 0$".

Examples:

- $20n + 5 = O(n)$
- $n^2 + 200n = O(n^2)$
- $1/n + 10 = O(1)$
- $\log(n) + n = O(n)$
- $n\log(n) + n = O(n\log(n))$
- $1.01^n + n^{1000} = O(1.01^n)$
- $1.01^{1.01^n} + 1.01^n = O(1.01^{1.01^n})$

Use Case in Computer Science

- "Runtime of algorithm is $O(n)$" means that: "in worst case, algorithm requires $O(n)$ operations."
- In this course, n is typically the size of a dataset.

Why is this important?

Use Case in Computer Science

- "Runtime of algorithm is $O(n)$" means that: "in worst case, algorithm requires $O(n)$ operations."
- In this course, n is typically the size of a dataset.

Why is this important?

- It's an indicator for the scalability of an algorithm.
- We can only apply $O(2^n)$ algorithms to tiny datasets.
- We can apply $O(n^2)$ algorithms to medium-sized dataset.
- We can apply $O(nd)$ algorithms if one of n or d is medium-sized.
- We can apply $O(n)$ algorithms to huge datasets.

Decision Tree Example

Suppose we have a dataset of n samples and d features.

Decision Tree Example

Suppose we have a dataset of n samples and d features.

- Classifying a single sample in depth-m decision tree: $O(m)$
  - You do a constant number of operations at each depth.
  - Note: no dependence on dimension d.

Decision Tree Example

Suppose we have a dataset of n samples and d features.

- Classifying a single sample in depth-m decision tree: $O(m)$
  - You do a constant number of operations at each depth.
  - Note: no dependence on dimension d.
- Fitting a decision stump with n objects and d features:
  - $O(n^2 d)$ if computing each score costs $O(n)$.

Decision Tree Example

Suppose we have a dataset of n samples and d features.

- Classifying a single sample in depth-m decision tree: $O(m)$
  - You do a constant number of operations at each depth.
  - Note: no dependence on dimension d.
- Fitting a decision stump with n objects and d features:
  - $O(n^2d)$ if computing each score costs $O(n)$.
- Fitting a decision tree of depth m:
  - Nave analysis: $O(2^m)$ stumps, so $O(2^mn^2d)$.
  - But each object appears once at each depth: $O(mn^2d)$.

Decision Tree Example

Suppose we have a dataset of n samples and d features.

- Classifying a single sample in depth-m decision tree: $O(m)$
  - You do a constant number of operations at each depth.
  - Note: no dependence on dimension d.
- Fitting a decision stump with n objects and d features:
  - $O(n^2d)$ if computing each score costs $O(n)$.
- Fitting a decision tree of depth m:
  - Nave analysis: $O(2^m)$ stumps, so $O(2^m n^2 d)$.
  - But each object appears once at each depth: $O(mn^2d)$.
- Finding optimal decision tree:
  - NP-Complete.
  - Hence why we approximate by using a greedy fitting algorithm.