

# CS251 Spring 2017 Lecture Notes

Stephanie R Taylor

## 1 Lecture 1: Wednesday Feb 1, 2017

### 1.1 Data Visualization and analysis

- Data: the plural of datum. According to the Oxford English dictionary, a datum is
  1. (a) Chiefly in pl. An item of (chiefly numerical) information, esp. one obtained by scientific work, a number of which are typically collected together for reference, analysis, or calculation.  
(b) In pl. Computing. The quantities, characters, or symbols on which operations are performed by a computer and which may be stored or transmitted in the form of electrical signals and held on recording media. Also (in non-technical contexts): items of information represented in digital form.
  2. (a) Something given or granted; something known or assumed as fact, and made the basis of reasoning; an assumption or premise from which inferences are drawn.  
(b) Philos. Anything immediately apprehended by or presented to the mind or senses. Freq. in datum of consciousness, datum of sense.
  3. Chiefly Surveying. A line, point, etc., forming a basis for measurement; a baseline, benchmark, or reference point

Data is a set of facts or items of information. It is important to remember that it can take on many forms, e.g. a number, many numbers, a

book, a collection of newspapers. Also, data are most easily analyzed if they are stored in a computer, but not all interesting data are digitized.

- Meta-data: Data about data. Meta-data tells you the meaning of the numbers or words or images. Suppose you are given a list of numbers [38,39,44]. It could be the ages of professors at Colby. Or it could be the heights of their children. Without that information, you can't learn anything from the data. Yes, you can compute a statistic about it, such as the mean or standard deviation. But you wouldn't know how to interpret it. You would learn nothing.
- Data visualization: the process of connecting data with our brains. While data visualization primarily refers to using our eyes to connect the data with our brains, data visualization is also possible via all of our other senses. The goal of data visualization is to enable the computational machinery in our brains to identify patterns, trends, or other interesting and salient characteristics of the data. While not all of the patterns we find in data are meaningful, our brains have incredible pattern identification and recognition capabilities, and data visualization enables us to make effective use of it.
- Data analysis: the process of computationally extracting information from data. Note that both visualization and analysis have the same ultimate goal: synthesizing knowledge from data. Data visualization enhances the ability of our biological computers to extract knowledge. Data analysis uses the computer to automate the process. Both have a role to play, as it can be difficult to define data analysis strategies without having some idea of the structures that exist in the data. Conversely, visualization can be difficult without some simple computational analyses to make the patterns more obvious.

## 1.2 An example: GapMinder World

We use the GapMinder World software at <http://www.gapminder.org>.

For the data he shows in the talk, the basic visualization has these elements:

- one bubble per country.
- x - income
- y - life expectancy
- color - continent
- size - population
- animate over time (display year as background)

We use GapMinder in the browser.

- Students choose a data set. Look at it.
- We describe the features they liked.
- What was intuitive?
- Was anything difficult?

## 1.3 Data Terminolgy

- data point, data vector, or feature vector: one or more numbers representing a single measurement event.
- variable or feature: a symbol that connects a set of numbers to a meaningful description. A variable/feature usually refers to a single number within a data point or data vector.
- multi-variable data: a data set whose data points consist of more than one measurement.

- dimension: the number of variables/features/measurements in a data point.
- min: the minimum value of a feature within a data set.
- max: the maximum value of a feature within a data set.
- range: the upper and lower bounds of potential values for a feature, sometimes refers to max - min.
- independent variable: a direct measurement or value that does not depend on another value in the data point. In the example  $y = mx + b$ , the variables  $m$ ,  $x$  and  $b$  are independent variables.
- dependent variable: a variable calculated from or that is a result of other variables or measurements in the data point. In the example  $y = mx + b$ ,  $y$  is the dependent variable because it is completely defined by the variables on the right side of the equation.
- missing data: sometimes a data point will not contain all of the measurements that other data points in the set possess.
- meta-information or meta-data: a description of the variables in a data set, often including their source, method of measurement, valid range, or valid values for the variable.
- precision: a description of the number of significant figures in a measurement, which is based on the repeatability, or reproducibility of a measurement. Note that a repeatable measurement is not necessarily correct.
- accuracy: a description of how close a measurement is to the true value.
- scaling: multiplying data by a scale factor to change its units (e.g. from lbs to pixels).
- normalizing: Transforming data by a linear or nonlinear function so that all values are between 0 and 1.

To see a list of TED talks about visualizations, check out this link:

<http://www.ted.com/topics/visualizations>

To follow tech news, which often talks about Big Data, check out these links:

- slashdot.org Slashdot is a technology-related current affairs website. Summaries of stories and links to news articles are submitted by Slashdot's own readers, and each story becomes the topic of a threaded discussion among users. (description stolen from Wikipedia)
- geekwire.com GeekWire is an independent technology news site and online community based in Seattle, Wash. covering the people, companies and innovations emerging from the Pacific Northwest and impacting the world.

## 1.4 Going over the syllabus

The syllabus is on the course website. No need to repeat it here.

## 1.5 Homework assignment

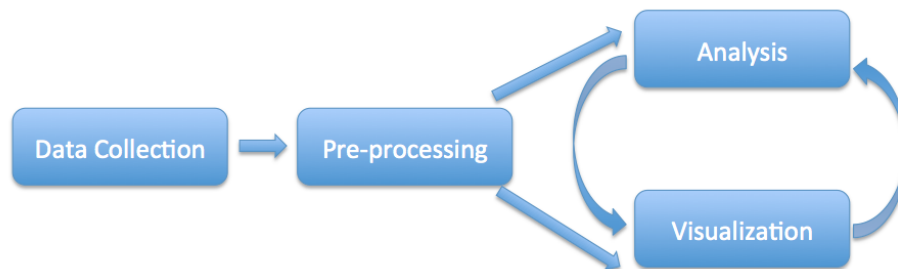
The usual pattern of this class will be to have homework on Wed that help you to prepare for a quiz on Friday. This week, we will have HW but not a quiz. To get a head start on future projects and to solidify your understanding of the terminology, find a data set (you can search the web, if you like), and send it to me, along with a statement indicating how many data points it has, the dimension of the data, the type of each data feature, and the meaning of each feature (e.g. the third feature is the weight in pounds of the most recent meal someone ate).

Send the data to Stephanie by 10pm on Thursday and she will respond to any questions/comments you might have.

## 2 Lecture 2: Friday Feb 3, 2017

### 2.1 The Process and our Projects

Here is an overview of the steps we must take to learn something from data:



- Data Collection
  - Task: We collect/generate data.
  - Project Code: It depends upon how you found the data. Maybe you needed to write scripts to do things with it, or maybe you were able to compile it in a spreadsheet.
- Reading the data file
  - Task: To read in a data file of a specialized format and to store it in an object that allows for easy access.
  - Project Code: You will do that with Data class in data.py.
- Pre-processing
  - Task: The pre-processing is done so that visualization and analysis make sense. I.e. we may pre-scale to get the units to be comparable.
  - Project Code: Any data-specific scripts you may have written. These scripts should generate files that can be read in by the Data class in data.py.
- Analysis

- Task: To run some formal analysis such as principal component analysis or clustering (on either the data or the results of another analysis)
- Project Code: a set of functions in `analysis.py`
- Visualization
  - Task: To visualize the data/analysis results interactively
  - Project Code: `display.py` and `view.py`. `Display.py` has the code to control the GUI (i.e. windows and buttons). `View.py` has the code to handle the projection of 3D data onto a 2D surface.

## 2.2 TkInter

In lab on Monday, we will begin the display program, focussing on aspects of the GUI. We will be using TkInter, which is the Python interface to the Tk GUI toolkit. Tk is a commonly used toolkit for GUIs and is automatically included in both the Windows and Max OS X installations.

Bruce has a nice set of notes on building a GUI with Python here:

<http://www.cs.colby.edu/courses/S14/cs251/labs/lab01/CS251-S12-LabNotes01.pdf>

Stephanie has a few comments to add:

- Help on the internet: The most helpful websites I have found are `effbot` and `stackoverflow` (`tkinter` is the key word to include in any search). Also, don't be afraid to try something you don't fully understand – experiment a bit, and it may make itself clear!
- Style: You will be writing a lot of code to create these widgets. Please follow the advice I gave in CS151/2. Use local variables if you can. Don't make a new field unless another method needs access to a particular value. For example, the `root` and `canvas` widgets are going to be accessed in a lot of methods, so they belong as fields. Buttons and menus, in general, don't need to be fields because your code likely will not be changing the appearance of the button or the contents of the

menu. If you get to the point at which you need to change either of those things, then make it a field.

- Handling mouse buttons when you have a track pad: In general, the way to mimic the right-mouse button (which in Tk is referred to as mouse button 2) is to press the control key and the track pad (which is referred to as mouse button 1). To mimic the middle-mouse button (which in Tk is referred to as mouse button 3), the standard key is option. So, for example, here is my code to handle all of my mouse-bindings:

```
self.canvas.bind( '<Button-1>', self.handleMouseButton1 )
self.canvas.bind( '<Control-Button-1>', self.handleMouseButton2 )
self.canvas.bind( '<Button-2>', self.handleMouseButton2 )
self.canvas.bind( '<Option-Button-1>', self.handleMouseButton3 )
self.canvas.bind( '<Button-3>', self.handleMouseButton3 )
self.canvas.bind( '<B1-Motion>', self.handleMouseButton1Motion )
self.canvas.bind( '<B2-Motion>', self.handleMouseButton2Motion )
self.canvas.bind( '<Control-B1-Motion>', self.handleMouseButton2Motion )
self.canvas.bind( '<B3-Motion>', self.handleMouseButton3Motion )
self.canvas.bind( '<Option-B1-Motion>', self.handleMouseButton3Motion )
```

## 3 Lecture 3: Mon Feb 6, 2017

### 3.1 Coordinate systems

Coordinate systems are at the heart of visualization techniques. Most of the work involved in generating visualizations is simply transforming data from one coordinate system to another. The starting coordinate system is the data space itself, the native coordinate system of the data. The final coordinate system is the visualization device, for example, a window on a computer screen. It is useful to create a number of intermediate coordinate systems between the initial and final coordinates to make the process of visualization tractable. The following is a fairly standard sequence of coordinate systems used in visualization.

**Data Coordinates:** the native coordinates of the data space. The max, min, and average values of each variable are defined in this space. For any data set, there is a bounding box in data coordinates within which all of the data resides.

**View Volume Coordinates:** the volume of the data space the user wants to view. The volume may be defined by the max and min variable values, or it may include only a subset of the data. If the view volume is constrained to be axis-aligned, then it is defined by an origin and an extent in each data variable direction. If the view volume can have arbitrary orientations, then it is defined by an origin and a set of orthonormal axes that define its orientation. The size of the volume is determined by an extent measured along the orthonormal axes.

**Normalized Viewing Coordinates:** A scaling of the data so that the visible data points—those within the view volume—fit within the range  $[0, 1]$  in all dimensions.

**Screen Coordinates:** A scaling, possibly a translation, and a projection to convert the normalized coordinates into screen coordinates where they are drawn.

## 3.2 Displaying in 2D

To display the data, we need to transform each point from its native coordinates to screen coordinates. This is a 3 step process:

1. Convert to view volume coordinates. Translate the data so the new origin becomes zero.
2. Convert to normalized view volume coordinates. Scale the data so that each coordinate is between zero and 1.
3. Convert to screen coordinates. The screen coordinates have their origin in the upper left corner. This involves a scaling and then a translation.

## 3.3 2D Example

Suppose we have this data set:

| Country        | GPD/capita | Life Expectancy |
|----------------|------------|-----------------|
| Czech Republic | 22860      | 78              |
| USA            | 41728      | 79              |
| China          | 8848       | 73              |
| Russia         | 14738      | 69              |

We will display two dimensions of the information in the x-y-plane. Suppose the area of the screen is 500 pixels high and 300 pixels wide. Let's make the x-axis represent GDB and the y-axis represent life expectancy. Since we want to see all the data, the range in data coordinates is [8848,41728] for x and [69,79] for y. The extent in x is  $41728-8848=32880$  and the extent in y is  $79-69=10$ .

To display the data, we convert each data point from data coordinates to screen coordinates.

1. Convert to the view volume coordinates. Our first data point is (22860,78) in data coordinates. To convert to view coordinate, we need to move (*translate*) the data so that the minimum values of each range map to the origin. In other words, we need to subtract the min values.  $(22860-8848, 78-69)=(14012, 9)$  is the point in view coordinates.

2. Convert to normalized view volume coordinates. We need to make sure all values are between 0 and 1. The min got mapped to 0 and the max gets mapped to 1. To do that, we need to *scale* the coordinates by the extent. I.e. we need to scale the x-coordinate by 32880 and the y-coordinate by 10. Our first data point is then  $(14012/32880, 9/10)=(0.42616, 0.9)$  in normalized view coordinates.
3. Convert to screen coordinates. First we need to *scale* the data - we stretch it and flip its y-direction. So the point becomes  $(0.42616*300, 0.9*(-500))=(127.8467, -450)$ . Then, we need to *translate* it so that the origin is in the upper left corner:  $(213.08+0, -450+500)=(127.8467, 50)$ .

Here are all of the coordinates in all of the coordinate systems for the example (with GDP/capita abbreviated to GDP and Life Expectancy abbreviated to LE). vc indicates view coordinates, nvc indicates normalized view coordinates, and sc1 indicates screen coordinates without the translation and sc indicates screen coordinates in their final form.

| Country        | GDP   | LE | GDP(vc) | LE(vc) | GDP(nvc) | LE(nvc) | GPD(sc1)  | LE(sc1) | GPD(sc)   | LE(sc) |
|----------------|-------|----|---------|--------|----------|---------|-----------|---------|-----------|--------|
| Czech Republic | 22860 | 78 | 14012   | 9      | 0.4262   | 0.9     | 127.84679 | -450    | 127.84679 | 50     |
| USA            | 41728 | 79 | 32880   | 10     | 1        | 1       | 300       | -500    | 300       | 0      |
| China          | 8848  | 73 | 0       | 4      | 0        | 0.4     | 0         | -200    | 0         | 300    |
| Russia         | 14738 | 69 | 5890    | 0      | 0.1791   | 0       | 53.7409   | 500     | 53.7409   | 0      |

### 3.4 Formalizing the coordinate transformation process

In the 2D case, it is relatively straight-forward to transform the coordinates - it requires only scaling and translation. But once we get to 3D, there will be rotation and projection as well. To perform these coordinate transformations, we can use some tools from linear algebra. In particular each coordinate can be represented as a vector and each transformation accomplished by a matrix multiplication.

#### 3.4.1 Homogeneous Coordinates

First, let's formalize the representation of the coordinates of a data point. We will use homogeneous coordinates.

Homogeneous coordinates are the basis for building a generic manipulation system for rigid objects. The homogeneous coordinates for a 2-D Cartesian

point  $(x_c, y_c)$  are  $(x, y, h)$ . The relationship between the homogeneous and standard coordinates is given in (3.4.1).

$$x_c = \frac{x}{h}$$

$$y_c = \frac{y}{h}$$

### Example

Let's represent the USA's data in homogeneous coordinates.

$$\begin{pmatrix} 41728 \\ 79 \\ 1 \end{pmatrix}$$

### 3.4.2 Matrix-Vector Multiplication

Now we turn to matrix multiplication. Each transformation will happen when we multiple the coordinate vector by a particular matrix.

**Dot Product** Matrix multiplication is built upon the dot product, also called the scalar product or inner product, of two vectors. Given two N-element vectors  $\vec{\alpha} = [\alpha_0 \ \alpha_1 \ \dots \ \alpha_{N-1}]$  and  $\vec{\beta} = [\beta_0 \ \beta_1 \ \dots \ \beta_{N-1}]$  their scalar product is defined as the multiplication of corresponding elements in the two vectors, followed by the summation of the products. The dot product of two vectors is always a single scalar value.

$$d(\vec{\alpha}, \vec{\beta}) = \sum_{i=0}^{N-1} \alpha_i \beta_i$$

**Matrix-Vector multiplication is a bunch of dot products** To multiply a matrix by a vector, we perform a dot product between each row of the matrix and the vector like this:

$$A\vec{b} = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} d(a_{0,*}, \vec{b}) \\ d(a_{1,*}, \vec{b}) \\ d(a_{2,*}, \vec{b}) \end{pmatrix} = \begin{pmatrix} a_{0,0}b_0 + a_{0,1}b_1 + a_{0,2}b_2 \\ a_{1,0}b_0 + a_{1,1}b_1 + a_{1,2}b_2 \\ a_{2,0}b_0 + a_{2,1}b_1 + a_{2,2}b_2 \end{pmatrix}$$

## Translation

Let's start with a small example.

$$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 41728 \\ 79 \\ 1 \end{pmatrix}$$

The result is

$$\begin{pmatrix} 1 * 41728 + 0 * 79 + 3 * 1 \\ 0 * 41728 + 1 * 79 + -1 * 1 \\ 0 * 41728 + 0 * 79 + 1 * 1 \end{pmatrix} = \begin{pmatrix} 41731 \\ 78 \\ 1 \end{pmatrix}$$

The effect of the above matrix is to translate the data - it translates it in x by 3 and in y by -1. How would you get it to translate by different amounts? Note that we can't translate in the final dimension - the homogeneous coordinate is playing a special role - it is only because we have the extra coordinate that we can do the translation in the other two).

In class, we update this matrix to do the translation we needed for our GDP example. It is

$$\begin{pmatrix} 1 & 0 & -8848 \\ 0 & 1 & -69 \\ 0 & 0 & 1 \end{pmatrix}$$

## Scaling

What would happen if we multiplied by

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}?$$

Nothing. This is the identity matrix (because it is the multiplicative identity in matrix multiplication).

Now let's construct a new matrix. How would we scale the data in x by 2 and y by 3?

Multiply it by

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In class, we develop the matrix that performs the scaling we need to translate the GDP example from view coordinates to normalized view coordinates. It is

$$\begin{pmatrix} 1/32880 & 0 & 0 \\ 0 & 1/10 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### A general form for scaling and translation matrices

I am going to write the formal system for 3-D coordinates. To have the 2-D version, simply remove the 3rd row and 3rd column.

We can perform multiple coordinate transformations by multiplying by multiple matrices. These operations are applied from right to left. If I wanted to translate and then scale a 3-D system, I would perform this set of multiplications:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

where  $s_x$ ,  $s_y$ , and  $s_z$  are the scale factors and  $t_x$ ,  $t_y$ , and  $t_z$  are the translation amounts. Although the math can be performed from left to right (it is possible to multiply the matrices together and matrix multiplication is transitive), I prefer to do the math from right to left (first translate, then scale).

Note that matrix-multiplication is not commutative – order matters!

## 3.5 Completing our 2D GDP example

Above, we spoke about the process in 4 steps (OK, it was 3 steps, but the third step is really two):

1. Convert to the view volume coordinates. This is a translation. We developed this above. Here it is again.

$$\begin{pmatrix} 1 & 0 & -8848 \\ 0 & 1 & -69 \\ 0 & 0 & 1 \end{pmatrix}$$

2. Convert to normalized view volume coordinates. This is a scaling we developed above. Here it is again.

$$\begin{pmatrix} 1/32880 & 0 & 0 \\ 0 & 1/10 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Convert to screen coordinates.

- (a) Scale and reverse the direction for the y-coordinate. (A flipping will be just a negative scaling).

$$\begin{pmatrix} 300 & 0 & 0 \\ 0 & -500 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- (b) Translate so that the origin is in the upper left corner

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 500 \\ 0 & 0 & 1 \end{pmatrix}$$

Line them all up to convert the data for the US to screen coordinates like this:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 500 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 300 & 0 & 0 \\ 0 & -500 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/32880 & 0 & 0 \\ 0 & 1/10 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -8848 \\ 0 & 1 & -69 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 41728 \\ 79 \\ 1 \end{pmatrix}$$

## 4 Lecture 4: Wed Feb 8, 2017

### 4.1 Matrix-Matrix Multiplication

You can represent a set of coordinates as a matrix - instead of 1 column you can have C columns for C data points. Then you can multiple the transformation matrix by the coordinate matrix and the result will be a matrix of transformed coordinates, with each column containing the coordinates for a point.

In matrix-matrix multiplication, the dot product uses the corresponding row in the first matrix and the corresponding column in the second. For example, consider two 3x3 matrices given below and their product written as a set of dot products.

$$\begin{bmatrix} d(a_{0,*}, b_{*,0}) & d(a_{0,*}, b_{*,1}) & d(a_{0,*}, b_{*,2}) \\ d(a_{1,*}, b_{*,0}) & d(a_{1,*}, b_{*,1}) & d(a_{1,*}, b_{*,2}) \\ d(a_{2,*}, b_{*,0}) & d(a_{2,*}, b_{*,1}) & d(a_{2,*}, b_{*,2}) \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix} \quad (1)$$

The same rule applies for matrices that are not square. The only requirement for matrix multiplication to be valid is that the number of columns in the first matrix must match the number of rows in the second matrix. If the input matrices are RxN and NxN, then the output matrix will be RxN. Matrix multiplication is not commutative, in general.

For our purposes, the most important case of matrix multiplication with non-square matrices is the case of multiplying a matrix and a vector.

$$\begin{bmatrix} d(a_{0,*}, b_{*,0}) \\ d(a_{1,*}, b_{*,0}) \\ d(a_{2,*}, b_{*,0}) \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \end{bmatrix} \quad (2)$$

If this is new to you, you might want to check out the Khan Academy tutorial at

<https://www.khanacademy.org/math/precalculus/precalc-matrices/multiplying-matrices/v/matrix-multiplication-intro>

## 5 Lecture 5: Fri Feb 8, 2017

### Numpy

Numpy is the library that supports linear algebra operations. The two most important data structures are the n-dimensional array and the matrix. The matrix always has 2 dimensions. It is the matrix you should be using for most of your code in this course, so let's look at some matrix operations in the Python interpreter.

First, let's make a row vector, storing it as a matrix.:

```
>>> import numpy as np
>>> a = np.matrix( [[1,3]] )
>>> a
matrix([[1, 3]])
```

To make a column vector, we take the transpose.

```
>>> b = a.T
>>> b
matrix([[1],
        [3]])
```

But b is not a copy of a, merely another way to view the same data. We see this by changing a value of a and then looking the the contents of b (they have been affected, too). To access an element we use two indices (always use 2 indices for a matrix, even if it has only one row or only one column!). The row index is first and the column index is second.

```
>>> a[0,0] = 4
>>> a
matrix([[4, 3]])
>>> b
matrix([[4],
        [3]])
```

To decouple b from a, we make a copy of a:

```
>>> b = a.copy().T
>>> b
matrix([[4],
        [3]])
>>> a[0,1] = 2
```

```
>>> a
matrix([[4, 2]])
>>> b
matrix([[4],
        [3]])
```

To see the shape of `a`, we use the `shape` field. This is very helpful for debugging. Instead of printing the contents of `a`, you can print the shape. It is a tuple: the first element is the number of rows, the second is the number of columns.

```
>>> print a.shape
(1, 2)
```

Now let's look at slicing. First make a  $3 \times 3$  matrix.

```
>>> A = np.matrix([ [1,2,3], [4,5,6], [7,8,9] ])
>>> A
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
```

Now we take the third column and then the second row.

```
>>> a2 = A[:,2]
>>> print a2
[[3]
 [6]
 [9]]
>>> ab = A[1,:]
... ]
>>> ab
matrix([[4, 5, 6]])
```

And we try to change an entry in the row vector `ab` using only one index. It fails:

```
>>> ab[1] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 1 is out of bounds for axis 0 with size 1
```

So, like good little programmers, we use two indices and it works. Also we see that we are changing the data in `A`. So, the slices are just views on the same data.

```

>>> ab[0,1] = 10
>>> a2
matrix([[3],
        [6],
        [9]])
>>> a2[1,0] = 3.0
>>> a2
matrix([[3],
        [3],
        [9]])
>>> A
matrix([[ 1,  2,  3],
        [ 4, 10,  3],
        [ 7,  8,  9]])

```

The take-home message:

- Matrices are always 2D and you should use two indices to slice them or index into them
- Slicing and transposing a matrix does not also copy the matrix.

## 6 Lectures 7-9: Week of Feb 13-17, 2017

### Moving on to 3D

Today, we move on to 3D coordinates. But first, let's recap of what we did in 2D:

1. We represented a data point in 2D data space using a homogeneous coordinate vector (it has 3 entries)
2. We transformed that homogeneous coordinate vector to the screen by multiplying on the left by four transformation matrices.

In 3D space, the basic idea is the same, but the details are different:

1. It is 3D - not 2D. We represent a data point in 3D space using a homogeneous coordinate vector (it has 4 entries).

2. The transformation is more complicated. We transform the point from data space to the screen by multiplying on the left by several more transformation matrices.

As you can see, step 1 is simple. It is just a matter of putting data into a vector. All the work is in making the transformation matrices. Together, they are called the viewing pipeline. We are interested in an orthographic viewing pipeline. That means that when we project the 3D data onto the 2D screen, we use the simplest project possible – an orthographic projection. In an orthographic projection, we simply use the (transformed) X and Y coordinates and ignore Z.

In other words, our goal is to transform a vector like this:

$$\begin{pmatrix} x_{norm} \\ y_{norm} \\ z_{norm} \\ 1 \end{pmatrix}$$

to a vector like this:

$$\begin{pmatrix} x_{screen} \\ y_{screen} \\ z_{ignore} \\ 1 \end{pmatrix}$$

where we simply pluck the first two entries out and use them as the x- and y- values for the Tk graphics object.

## 6.1 Orthographic Viewing Pipeline

The 3D viewing pipeline requires more input than the screen size and the bounding box of the data. We need to know what angle to use as our view reference point (where are we standing and how are we leaning as we look at the data?)

So there are several things we need to know in order to project 3D data onto a 2D screen.

- VRP: view reference point and center of the view window; origin of the view reference coordinates
- $\vec{VPN}$ : view plane normal, direction of viewing
- $\vec{VUP}$ : view up vector, up orientation of the view volume
- $\vec{U}$ : x-axis of view reference coordinates
- extent:  $(E_x, E_y, E_z)$ , size of the bounding box in data space in view reference coordinates
- screen:  $(s_x, s_y)$ , size of the output device window in pixels

## 6.2 Setting up the view axes

We choose the position in the data we would like to view and the position from which we would like to do the viewing (this is the view reference point). From this information and an educated guess about how we should orient the view axes (i.e. what is up), we can build an orthonormal set of axes.

To do this, we will need the cross product.

### 6.2.1 Tool we need: Cross Product

The cross product, also called the outer product, or vector product, is a standard operation on two vectors. The cross product of two vectors is a third vector that is orthogonal to both original vectors. For 3D vectors, the cross product is defined as in (3).

$$\vec{v} = \begin{bmatrix} y_0 z_1 - y_1 z_0 \\ z_0 x_1 - z_1 x_0 \\ x_0 y_1 - x_1 y_0 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \times \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \quad (3)$$

We ignore the homogeneous coordinate when calculating the cross product.

### 6.2.2 Build the view reference axes

Now we are ready to build the view reference coordinates. (Notice that this doesn't mean we are yet building the transformation matrices we will use. The view reference axes will be used to determine what the entries of the transformation matrices are.)

Normalize the axes  $\vec{U}$ ,  $V\vec{U}P$ , and  $V\vec{P}N$  by dividing each vector by its length (square root of sum of squares).

$$\begin{aligned}\vec{U} &= V\vec{U}P \times V\vec{P}N \\ V\vec{U}P' &= V\vec{P}N \times \vec{U}\end{aligned}\tag{4}$$

(Note that we use normalize in two different ways: To normalize a column of data, we want each entry to have a value between 0 and 1. To normalize a vector, we want the square of the entries to sum to 1.)

**Running example setup.** Let's look at a set of points that look like an F floating in a normalized data space. We will look at the center of the data in the Z=0 plane, (0.5,0.5,0) from the center of the data in the Z=1 plane (0.5,0.5,1). We will construct the view reference axes using just the 3 usual coordinates, because we are going to be using the cross-product, which is really meant for just the 3 coordinates.

The VPN is

$$\begin{pmatrix} 0.5 \\ 0.5 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.5 \\ 0.5 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

The up vector begins as

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

We create the u vector by crossing the up and VPN vectors

$$\vec{u} = v\vec{u}p \times v\vec{p}n = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$$

then recompute the up vector by crossing the view normal and u vectors

$$\vec{up} = v\vec{pn} \times \vec{u} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

(Why do we need to recompute the up vector? Because we simply guessed its value. There is no reason we can assume it started out orthogonal to the VPN vector. We need to do this cross product to map it orthogonal to the VPN).

### 6.3 Creating a transformation matrix

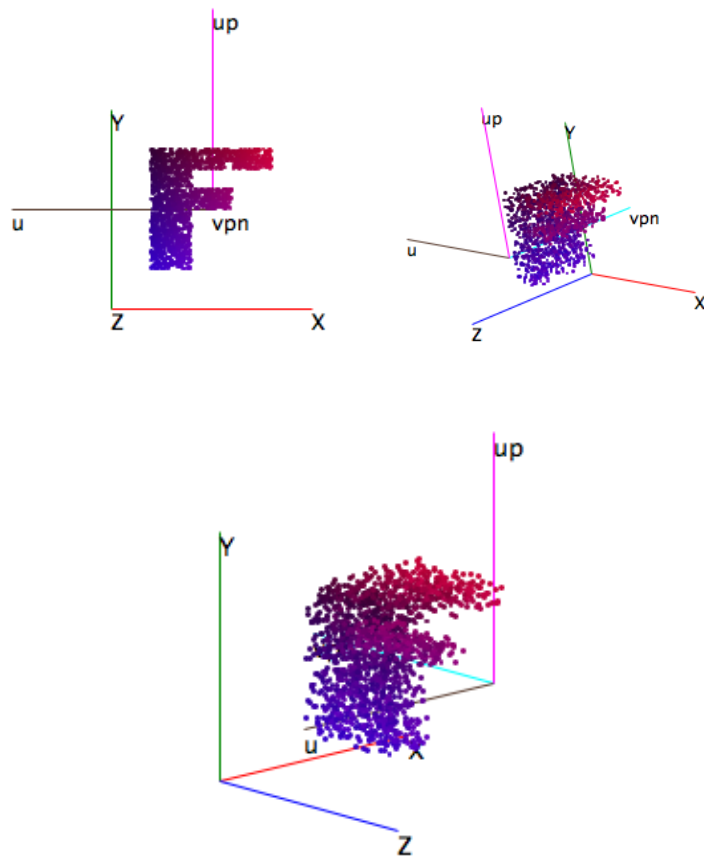
Let's follow the process with our F example.

1. Set the view transformation matrix to a 4x4 identity matrix.

$$V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Running example.** Below are two views of the data. We keep the X, Y, and Z axes fixed (and color-code them red for X, green for Y, and blue for Z). We show the view reference axes (color-coded so that the view normal (vpn) is cyan, the up vector is magenta, and the u vector is brown) and the data as they have been affected by the transformation so far. One thing to keep in mind is that the origin of the view reference axes is the view reference point. The data will always look the way we want it to when we look from the view reference point. Our goal is to get the data at the appropriate X, Y positions for them to be plotted on the screen.

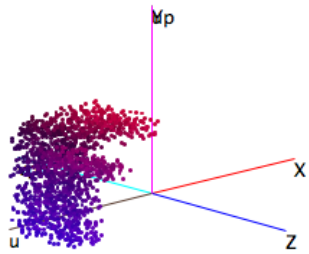
The view on the left is from the  $Z=1$  plane looking back. The view on the right and the view on the bottom have been rotated so we can see the relationship between the axes. The  $u$  vector is going in the negative  $X$  direction. The  $vpn$  vector is going in the negative  $Z$  direction. And the  $up$  vector is going in the  $Y$  direction.



2. Translate VRP to the origin.

$$V = T(-\text{VRP}_x, -\text{VRP}_y, -\text{VRP}_z)V$$

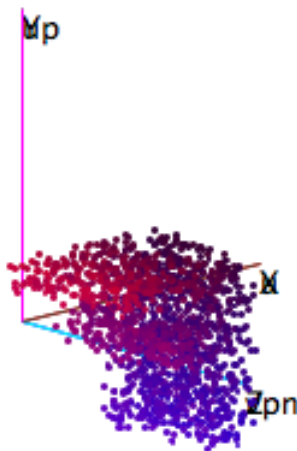
**Running example.** Below are the axes after they have been translated. The relationship between the two sets of vectors is more clear (up and Y are the same, u is opposite X, vpn is opposite Z).



3. Align the axes.

$$V = \begin{bmatrix} U_x & U_y & U_z & 0 \\ \text{VUP}'_x & \text{VUP}'_y & \text{VUP}'_z & 0 \\ \text{VPN}_x & \text{VPN}_y & \text{VPN}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} V$$

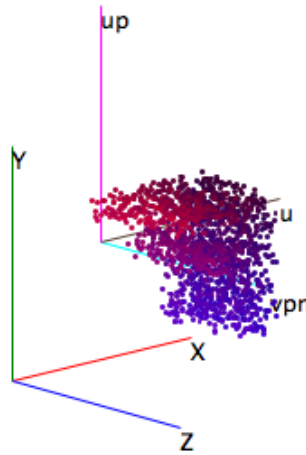
**Running example.** Below we show the axes after they have been aligned. Up is aligned with Y, u is aligned with X, and vpn is aligned with Z. Notice that the center of the data is now the origin of the axes. That is not what we want, ultimately.



4. Translate the lower left of the view window to the origin. (option 2: skip this step)

$$V = T(\frac{1}{2}E_x, \frac{1}{2}E_y, 0)V$$

**Running example.** Below are the axes after they have been translated so that the data are all positive again.



5. Scale the view volume to normalize the view volume.

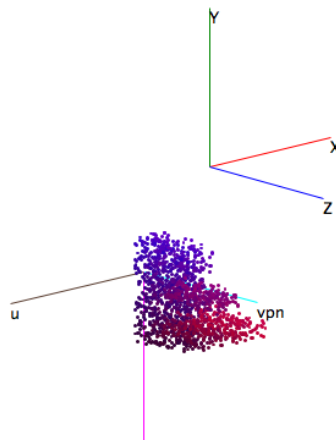
$$V = S(\frac{1}{E_x}, \frac{1}{E_y}, \frac{1}{E_z})V$$

**Running example.** Since the data in the running example were already normalized, the extents were 1, and this has no effect.

6. Scale to screen coordinates and invert the x and y axes.

$$V = S(-s_x, -s_y, 1)V$$

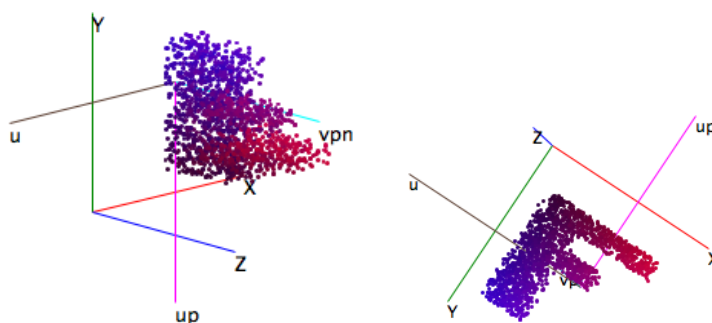
**Running example.** Below are the axes after the U and Up vectors have been inverted.



7. Translate by the screen size (option 2:  $T(\frac{1}{2}s_x, \frac{1}{2}s_y, 0)$ )

$$V = T(s_x, s_y, 0)V$$

**Running example.** Below are the axes after they have been translated. Notice that it gets them back up into the  $X > 0, Y > 0, Z > 0$  octant. But, in the image on the left, they still look flipped. Why? Because we still need to remember that we are viewing from the view reference point. The image on the right is oriented to make it easy for use to look from the view reference point. We see that the up vector is shooting up out of the “top” of F. The u vector is shooting out of the left of F. And the X and Y positions make sense for plotting on the screen. The left of F has small positive X value. The right of F has larger positive X values. The top of F has small positive Y values. The bottom of F has larger positive Y values.



## 6.4 Example to do on your own:

### 3D Data manipulation

Consider a 3D data set with a range of  $[6, 10]$  in dimension A,  $[10, 20]$  in dimension B, and  $[20, 40]$  in dimension C. The mean value of the data is  $\mu = (8, 15, 30)$ . You want to set up the view such that the user is looking at the mean data location from the point  $v\vec{r}p = (10, 20, 40)$  using a view volume that has an extent of  $[20, 20, 20]$ . To properly orient the view volume, we also need to know which direction is up. We can use dimension B as a default up direction, in which case  $v\vec{u}p = (0, 1, 0)$ .

1. Convert to view volume coordinates. If we position the viewer at the center of one end of the view volume, the process has three steps.
  - (a) Translate the view reference point to the origin  $T(-10, -20, -40)$
  - (b) Orient the axes of the data space and the view volume. The three axes of our view space are the View Plane Normal, which is the direction we're looking, the View Up Vector, and the U vector. We can calculate them using the following process.

$$v\vec{pn} = \text{lookat} - \text{viewer} = (8, 15, 30) - (10, 20, 40) = (-2, -5, -10) \quad (5)$$

$$\vec{u} = v\vec{up} \times v\vec{pn} \quad (6)$$

$$v\vec{up}' = v\vec{pn} \times \vec{u} \quad (7)$$

Use normalized versions of these three vectors to orient the axes.

- (c) Once the axes are oriented, we want to shift the volume so the lower left corner of the viewing face is at the origin. If the viewing face has size  $(du, dv)$ , then the translation is  $T(0.5du, 0.5dv) = T(10, 10)$ .
2. Convert to normalized view coordinates. Scale each dimension of the view volume to 1:
 
$$S(\frac{1}{du}, \frac{1}{dv}, \frac{1}{dw}) = (0.05, 0.05, 0.05).$$
3. Convert to screen coordinates. Scale the two dimensions of the view volume that are perpendicular to the view direction to the screen coordinates, flip the axes, and then translate. Note that both the X and Y axes are reversed from screen coordinates because of the way we set up our view reference coordinate system.
  - (a) Scale and flip the coordinate systems:  $S(-s_x, -s_y)$
  - (b) Translate so the window is all positive:  $T(s_x, s_y)$

---

In summary, 3D viewing from arbitrary locations is a bit more complex than 2D. The important parameters are the **View Reference Point**, which is the center of the view volume face through which the viewer is looking, the

**View Up Vector**, which orients the view volume, and the **View Volume Extent**, which is given in View Reference coordinates (which have units similar to the data space). From these parameters we can calculate all of the necessary transformations into normalized view coordinates. Note that, if the data space has drastically different ranges, then arbitrary viewing may not produce good visualizations without scaling the data so that distances in each dimension have roughly similar meanings. We'll be looking at how to do this a bit later.

## 7 Lectures 10-12: Week of Feb 20-24, 2017

### Connecting User Input and Viewing

Interactive data viewing requires us to connect user input to changes in the view system parameters. User actions generate either specific changes in the view parameters or incremental modifications. The forms of interactive view modification include panning (translating), scaling, and rotating.

- Panning/Translating: should move the VRP around the view plane, keeping the orientation of VUP and U fixed.
- Scaling: should increase or decrease the extent of the view volume in the view plane, keeping other parameters fixed.
- Rotation:
  - In 2D, should rotate the VUP vector about the z-axis, modifying the VUP vector and the derived U vector, but leaving their Z values at 0.
  - In 3D, the VRP may move in a circle around a fixed point (e.g. center of the current view volume). The same rotation can be applied to the VPN, VUP, and U vectors. An alternative interface method is to rotate the view direction according to mouse motions, keeping the VRP constant during the rotation.

The mouse gives the user the ability to move within 2 dimensions. Therefore, whether we are translating, rotating, or scaling, we have to have a method for translating user actions into modifications of the view reference coordinates.

- Translation of the mouse: translation of the VRP in the view plane
- Translation of the mouse: rotation of the VUP vector
- Translation of the mouse: scaling of the extent of the view window

A typical mouse has three buttons (left, right, center) which translate into three separate button clicks. Many mice also have a wheel or trackball on them, which provides an additional input device. The keyboard and menu items provide additional inputs that we can use as modifiers to the mouse action.

The key is how we connect the user interface elements to the parameters of the view transformation. Some key design questions include the following.

- Which mouse actions, buttons, or menu states connect to which view parameters?
- How does each action or button press relate to changes in one or more view parameters?
- What is the control law for the relationship?
- What are the control parameters of the relationship?
- Why is this a useful relationship?

Note that the first four questions are up to the programmer. You can make the relationships be whatever you want. The last question is what should guide the design process, and is often the motivation for user studies that look at the effects of different design decisions.

## 7.1 Interfaces as Control Laws

Any time we are connecting an input to an action, we have to write a control law. Consider, for example, connecting the motion of a mouse to panning of the data. The mouse moves in screen coordinates. Therefore, the input to the system is in pixels. Pixels, however, are not a meaningful unit in the data space. We can write the relationship between mouse motion and motion in the data space as follows, where  $\Delta U$  is the motion in the data space,  $\Delta x$  is motion in screen space, and  $k_p$  is a proportional constant relating the two motions.

$$\Delta U = k_p \Delta x \quad (8)$$

We can control the relationship between the two spaces by varying the value of  $k_p$ . If we set  $k_p = \frac{E_x}{S_x}$ , which is the ratio of the data space extent to the screen space, then the data will appear to track the mouse as it moves. However, we could make  $k_p$  smaller or larger than that ratio and get different effects. If  $k_p$  is smaller, the data will lag the mouse motion. If  $k_p$  is bigger, the data will precede the mouse motion. The former is useful for fine tuning of a visualization, the latter is useful for moving quickly between different parts of the data.

We are not limited to a simple proportional relationship, however. Consider, for example, the use of an inertia term.

$$\Delta U_t = k_p \Delta x + k_n \Delta U_{t-1} \quad (9)$$

The inertia term means that the motion of the data is dependent not only on the current user input, but also the prior user input. A more complex form of the motion results if we make  $k_n$  dependent upon whether the user is actively controlling the device. Under active control, we can make  $k_n = 0$ , changing it to something like  $k_n = 0.9$  when the user stops actively controlling the mouse. The iPod, and other multi-touch devices make use of this type of relationship to enable quick flipping and scrolling through lists.

Control theory also gives us an intuitive way of understanding the results of different design choices.

1. overdamped - system doesn't respond rapidly enough (sluggish)

2. critically damped - system responds just right
3. underdamped - system responds too strongly and in an effort to correct the overshoot, you end up oscillating
4. unstable - system just doesn't do the right thing

## 7.2 Panning within the view plane

- Each change in the mouse position corresponds to a change in the position in the data space
- Horizontal motion in the view plane should move along the U axis
- Vertical motion in the view plane should move along the VUP axis
- The view volume extent and the screen size tell us how to scale pixel motion to data space motion

Process

1. Calculate how much the mouse moved on the screen  $\Delta x, \Delta y$ .
2. Scale the motion into data space by dividing by the screen size and multiplying by the extent.

$$(\Delta u, \Delta v) = (\Delta x \frac{E_x}{S_x}, \Delta y \frac{E_y}{S_y}) \quad (10)$$

3. Multiply the horizontal screen motion by the U axis and the vertical screen motion by the V axis to get the motion of the VRP in data space.

$$\begin{aligned} \Delta \text{VRP}_x &= \Delta u U_x + \Delta v \text{VUP}_x \\ \Delta \text{VRP}_y &= \Delta u U_y + \Delta v \text{VUP}_y \\ \Delta \text{VRP}_z &= \Delta u U_z + \Delta v \text{VUP}_z \end{aligned} \quad (11)$$

4. Add the motions onto the VRP.

$$\text{VRP} = (\text{VRP}_x + \Delta \text{VRP}_x, \text{VRP}_y + \Delta \text{VRP}_y, \text{VRP}_z + \Delta \text{VRP}_z) \quad (12)$$

5. Recalculate the view transformation matrix
6. Calculate the new view locations of the data
7. Adjust the coordinates of the visual objects

### 7.3 Scaling the extent

- The extent should scale uniformly in all directions
- The user should be able to scale up and down using the same motion
- An easy solution is to translate vertical motion into scaling

Process

1. Store the initial mouse click as a reference point  $P_0$
2. Store the initial view extent  $E_0$
3. For each mouse motion
  - (a) Calculate the vertical distance between the initial click and the current mouse location.

$$\Delta v = P_{iy} - P_{0y} \quad (13)$$

- (b) Generate a multiplication factor from the difference. Set the value of  $k$  to control the speed of the scaling.

$$f = 1.0 + k\Delta v \quad (14)$$

- (c) Bound the factor on the zoom side to a small number larger than zero (e.g. 0.05)
- (d) Multiply the initial view extent by the factor and update the view extent

$$E_i = fE_0 \quad (15)$$

- (e) Recalculate the view matrix
- (f) Calculate the new view locations of the data
- (g) Adjust the coordinates of the visual objects

## 7.4 Rotating the view

### 7.4.1 Rotation that circles around the data

Rotation (Circle around data): rotate the view plane around the center of the view volume.

- Left-right motion: translate the center of view volume extent to the origin, align the VRC axes, rotate around VUP, unalign, untranslate

$$X_h(\theta_h) = T(\text{VRP} + \frac{E_z}{2}\text{VPN})R_{xyz}^t R_y(\theta_h)R_{xyz}T(-\text{VRP} - \frac{E_z}{2}\text{VPN}) \quad (16)$$

- up-down motion: translate the center of view volume extent to the origin, align the VRC axes, rotate around U, unalign, untranslate

$$X_v(\theta_v) = T(\text{VRP} + \frac{E_z}{2}\text{VPN})R_{xyz}^t R_x(\theta_v)R_{xyz}T(-\text{VRP} - \frac{E_z}{2}\text{VPN}) \quad (17)$$

We need to use these matrices to transform the VRP point (homogeneous coordinate of 1) and the U, VUP, and VPN vectors (homogeneous coordinate of 0). Then rebuild the view matrix to update the data. The effect is that of the data rotating in space while the user stays fixed.

We can combine the two matrices thus:

$$X(\theta_h, \theta_v) = X_v(\theta_v)X_h(\theta_h) \quad (18)$$

$$T(\text{VRP} + \frac{E_z}{2}\text{VPN})R_{xyz}^t R_x(\theta_v)R_y(\theta_h)R_{xyz}T(-\text{VRP} - \frac{E_z}{2}\text{VPN}) \quad (19)$$

Why? That is an exercise left to the reader. It is a simple matter of the matrices being inverses of each other. They cancel out..

Note:  $T$  indicates a translation matrix,  $R_{xyz}$  indicates an alignment matrix,  $R_y(\theta)$  indicates a rotates around the Y axis, and  $R_x$  indicates a rotation around the X axis. We have seen the first two matrices in the 2- and 3-D viewing pipeline.

To rotate about the Y-axis:

$$R_y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To rotate about the X-axis:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Process:

1. Store the initial mouse click as a reference point  $P_0$
2. Store the initial view reference axes (VRP, UP, U, and VPN) as  $\text{VRP}_0$ ,  $\text{UP}_0$ ,  $\text{U}_0$ , and  $\text{VPN}_0$ .
3. For each mouse motion

- (a) Calculate the horizontal distance between the initial click and the current mouse location.

$$\Delta x = P_{ix} - P_{0x} \quad (20)$$

- (b) Convert it to the rotation angle (for rotating about UP). Here, I am assuming that dragging the mouse across the entire window leads to a complete ( $2\pi$ ) rotation:

$$\theta_h = \frac{-\Delta x}{0.5S_x} \pi \quad (21)$$

- (c) Calculate the vertical distance between the initial click and the current mouse location.

$$\Delta y = P_{iy} - P_{0y} \quad (22)$$

- (d) Convert it to the rotation angle (for rotating about U)

$$\theta_v = \frac{\Delta y}{0.5S_y} \pi \quad (23)$$

- (e) Compute  $X(\theta_h, \theta_v)$
- (f) Recompute the VPN, UP, U, and VRP by putting  $\text{VRP}_0$ ,  $\text{UP}_0$ ,  $\text{U}_0$ , and  $\text{VPN}_0$  into homogeneous coordinates and multiplying them by  $X$  ( $X$  is on the left)
- (g) Recalculate the view matrix
- (h) Calculate the new view locations of the data
- (i) Adjust the coordinates of the visual objects

### 7.4.2 Fly-through rotation

Rotation (fly-through): rotate around VUP, anchored at the VRP, for right-left motion of the mouse and rotate around the U-vector, anchored at the VRP, for up-down motion. The rotations apply only to the orientation of the view coordinate system, which is defined as a set of vectors and is translation invariant.

- Left-right motion: translate VRP to the origin, align the axes, rotate by  $\theta_h$  about the Y axis, invert the alignment, then translate back. To undo a rotation, we can multiply by the inverse matrix. The inverse of a rotation matrix happens to be its transpose, which makes inverting rotations an easy thing to do.

$$X_h(\theta_h) = T(\text{VRP})R_{xyz}(U, \text{VUP}, \text{VPN})^t R_y(\theta_h) R_{xyz}(U, \text{VUP}, \text{VPN})T(-\text{VRP}) \quad (24)$$

- Up-down motion: align axes, rotate by  $\theta_v$  about the X axis, invert the alignment

$$X_v(\theta_v) = T(\text{VRP})R_{xyz}(U, \text{VUP}, \text{VPN})^t R_x(\theta_v) R_{xyz}(U, \text{VUP}, \text{VPN})T(-\text{VRP}) \quad (25)$$

Use  $X_u(\theta_u)$  and  $X_v(\theta_v)$  to transform the VRP point (homogeneous coordinate of 1) and the U, VUP, and VPN vectors (homogeneous coordinate of 0). Then rebuild the view matrix to update the data. The effect is that of turning your head around while the world stays fixed. Note that if both  $\theta_u$  and  $\theta_v$  are non-zero, you can do a single transformation process with the two rotation matrices in the center of the expression.

## 7.5 Controlling with keys instead of the mouse

Translation: using typical keyboard controls (e.g. wasd for forward/left-/back/right) works well. Number pad controls also work well.

- Forward/backward moves the VRP along the VPN by some step size  $\gamma$ .

$$\text{VRP}_{t+1} = \text{VRP}_t + \gamma \text{VPN} \quad (26)$$

- Left/right moves the VRP along the U axis.

$$\text{VRP}_{t+1} = \text{VRP}_t + \gamma U \quad (27)$$

- Up/down moves the VRP along the VUP axis.

$$\text{VRP}_{t+1} = \text{VRP}_t + \gamma \text{VUP} \quad (28)$$

## 8 Lectures 13-15: Week of Feb 27 - Mar 3, 2017

### Probability distribution functions, Basic Statistics, Noise

Probability distribution functions assign probabilities to each of the possible outcomes of a random experiment. These distributions can be continuous (e.g. grading on a 4.0 scale), or discrete (e.g. grading with letter grades) (see Fig. 2).

A probability distribution function specifies the probability of a random variable  $X$  taking on a particular value. The most commonly used PDF is the Gaussian distribution, or normal distribution. A PDF for a single continuous random variable can be any function with an integral of 1.

$$\int_{-\infty}^{\infty} \text{PDF}(X) = 1 \quad (29)$$

and for a discrete random variable is

$$\sum \text{PDF}(X) = 1 \quad (30)$$

#### 8.0.1 Gaussian (Normal) distribution

The Gaussian distribution is the most commonly used probability distribution function for analyzing data.

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Probability Distribution Function (PDF): assigns a probability to each of the possible outcomes of a random experiment

### Continuous Distributions

characterized by a probability density function (pdf): probability of falling into an interval is the integral of the pdf.



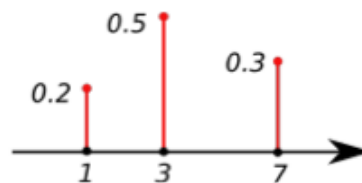
pdf for the Gaussian (normal) distribution.

To compute the probability of  $X$  being in the interval  $[a, b]$ , integrate the pdf  $f(x)$ :

$$P[a \leq X \leq b] = \int_a^b f(x) dx.$$

### Discrete Distributions

characterized by a probability mass function (pmf): probability that a discrete random variable is exactly equal to some value



pmf for an example discrete distribution. The probability of  $X$  being 7 is 0.3.

(images from Wikipedia)

Figure 1: Terminology for continuous and discrete probability distributions.

- $\mu$ : mean of the distribution
- $\sigma$ : standard deviation of the distribution

The Gaussian distribution is heavily used in part because of the central limit theorem, which states that the sum of many independent identically distributed random variables will tend to be distributed according to the normal distribution. In other words, the output of any process that is the result many smaller processes, each contributing a similar type noise to the system, will tend to look like a normal distribution. The upshot is that physical quantities that are expected to be the sum of many independent processes (such as measurement errors) often have a distribution very close to normal. (Wikipedia).

### 8.0.2 Uniform distribution

A uniform distribution describes random processes in which values within a range ( $a$  to  $b$ ) are equally probable. (For the continuous uniform distribution, this means all intervals of the same length on the distribution's support are equally probable.)

The pdf for the continuous uniform distribution is

$$f(x; a, b) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & x < a \text{ or } x > b \end{cases}$$

where  $a$  is the lower bound and  $b$  is the upper bound.

### 8.0.3 Exponential distribution

The exponential distribution (a continuous distribution) describes random processes in which there occurs lots of outcomes with small values and very few outcomes with large values. It describes the time between events in a Poisson process.

The pdf of an exponential distribution is

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where  $\lambda$  is the rate parameter.

The discrete analog is the geometric distribution.

## 8.1 Histograms

Viewing the raw data in its native space as a scatter plot is one method of visualization of a data set, and can be very useful as a method of exploration. However, there are many other ways to think about representing characteristics of data.

- Histograms are representations of distributions
- For discrete data, how many elements of each discrete element exist in the data set?
- For continuous data, what is the distribution of data values?

Histograms can be N-dimensional, although beyond 2 or 3 they get unwieldy to represent. 2-D histograms that use color or a 3D representation are common.

Stephanie thinks of histogram-plotting as a useful step in the data pre-processing step because it gives her an idea about how the data behave in each dimension. Looking at the individual dimensions gives an idea of how the data are distributed in data space, but doesn't necessarily give any information about how the different features may be related.

For continuous data, we often break the space into bins in order to represent the distribution, counting how many elements fall into each bin. The number of bins to use for a particular data set depends on a number of factors. If we have bins covering the entire range of possible values, we may lose discriminability in the central portion of the distribution. If the bins are too big or too small, we lose the ability to see coherent patterns in the data. Using bookend bins is a common choice, but can result in strange looking

plots where the frequency bumps up on the outermost bins since they cover more of the data space. The following questions can help us to tailor a histogram to the needs of the user.

- What is the frequency of real change in the distribution? → Bin size should show real changes.
- What is the precision of the measurement? → Bin size should not be finer than precision.
- What is the purpose of the histogram? → Bin size and range should support the goals.

Here is a fun web app that allows you to see what happens when you adjust the bin size. Can the bin size be adjusted to make the viewer come to different conclusions?

<http://www.shodor.org/interactivate/activities/Histogram/>

### **8.1.1 Histograms and Probability Distribution Functions**

Histograms are a method of estimating probability distribution functions from data.

Therefore, we can convert a histogram to a PDF by dividing each bin value by the sum of the bins. PDFs are useful for sampling problems, modeling problems, perception tasks, and learning problems. The characteristics of the PDF are also useful for data analysis and characterization.

## **8.2 Bonus notes (not discussed in class)**

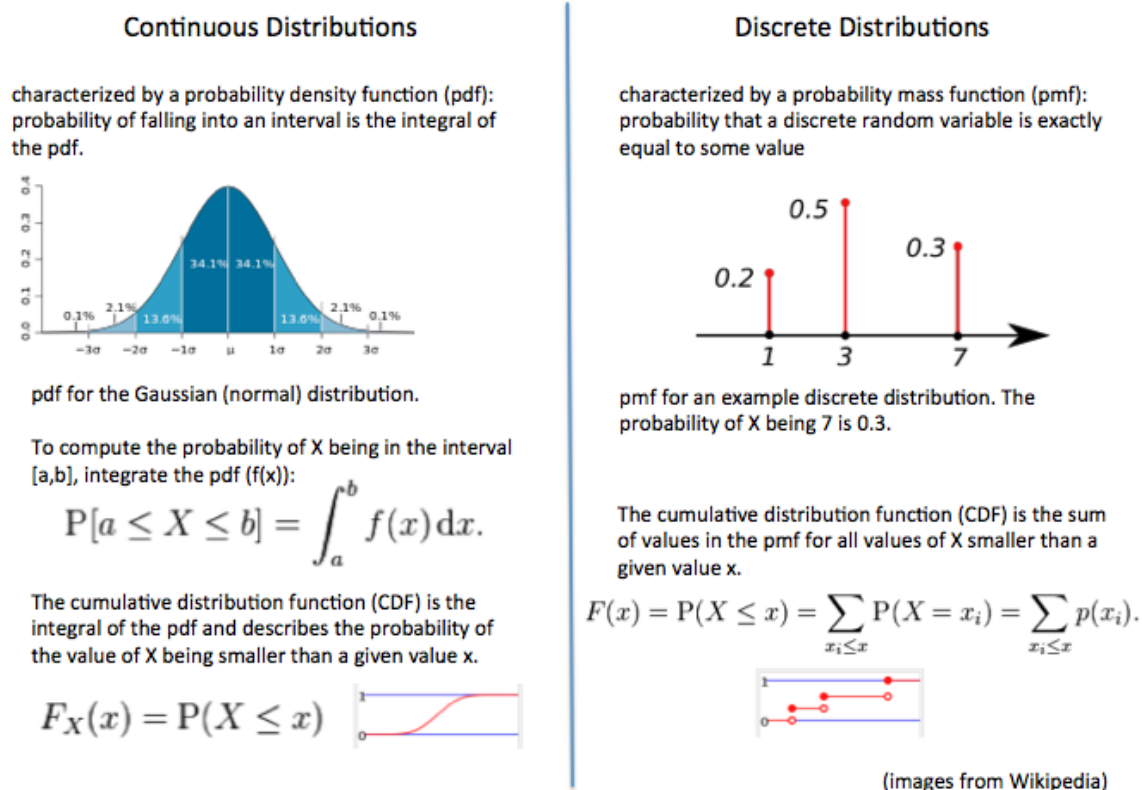
### **8.2.1 Cumulative distribution functions**

The integral of the probability distribution function, also called the cumulative distribution function [CDF], is an alternative method of visualizing a distribution, and one that is applicable to continuous data without binning. The CDF indicates the probability that a value in the data set is below a certain value.

A CDF can be used as a lookup table to emulate drawing data from a particular distribution. First, pick a number uniformly and randomly in the range [0, 1]. Second, find the position on the CDF graph that corresponds to that value on the y-axis. The x-value of the point on the CDF graph corresponds to a value in the data range. Areas of the CDF with steep slopes correspond to highly probable values of the data.

Fig. 2 shows the updated function terminology table.

**Probability Distribution Function (PDF):** assigns a probability to each of the possible outcomes of a random experiment



(images from Wikipedia)

Figure 2: Terminology for continuous and discrete probability distributions.

### 8.2.2 Taking noise into account when making histograms

If we decide to bin data, we may want to make use of the fact that data has noise characteristics and distribute each data point accordingly. Otherwise, regular binning may lead to **aliasing**.

- Rounding to the nearest bin center is not necessarily correct
- If a measurement has noise that is on the order of the bin size we should distribute the value of the data point between neighboring bins

For large numbers of measurements, discrete histograms become more accurate. For small numbers, we usually want to distribute weight according to the measurement noise. An alternative is to run a smoothing filter over the histogram after it's built.

One method of distributing a count across multiple bins is to use a Gaussian model for the noise in the measurement. The integral under the Gaussian that lies within the central bin is the weight added to that bin. The same calculation provides the weight for the bin's neighbors. Usually, the process is limited to a specific number of bins (2 or 3), or to those within some threshold, such as  $2\sigma$ . Unfortunately, calculating the integral under a Gaussian can be an expensive computation (it does not have a closed form solution).

One approximation that is often used, is to distribute the weight of a data point between the two nearest bins. This approach is appropriate when the size of the bins is approximately  $2\sigma$ , where  $\sigma$  is the standard deviation of the noise in the measurement.

Distributing weights across bins is more important for high resolution histograms where the bin size is close to the measurement noise. However, even low-resolution histograms will exhibit aliasing if the count weights are discrete.

## 8.3 Computing some basic statistics

Basic concepts are pretty straightforward. When computing a statistic from data, the term is “sample mean”, “sample variance”, etc. I have left off the “sample” for the sake of simplicity.

- Mean: sum of the data values divided by the number of data values

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x_i \quad (31)$$

- Median: data value with equal numbers of values above and below it

$$\text{median} = \text{sort}(\{x_0, \dots, x_{N-1}\})[N/2] \quad (32)$$

- Mode: data value with the highest frequency (usually applied to categorical or binned data)

$$\text{mode} = \max(\text{histogram}(\{x_0, \dots, x_{N-1}\})) \quad (33)$$

- Standard Deviation: a measure of the dispersion of the data as a distance from the mean

$$s_N = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^2} \quad (34)$$

- Variance: the square of the standard deviation

$$v_N = \frac{1}{N} \sum_{i=0}^{N-1} (x_i - \bar{x})^2 \quad (35)$$

When they fail to be meaningful is a different issue.

- Bi-modal or multi-modal distributions cause problems
- Missing data values can cause problems

## 8.4 Distributions inform visualization (Or, range selection and normalization)

How do we automatically select ranges for display? We have to pick ranges for many aspects of a visualization, including spatial extent, colors, sizes, or other methods of representing data values. Part of the purpose of interactive visualization is to avoid having to automatically select ranges, but even if the user has control over the spatial view, we may still have to select ranges for the color and size of the plotted points.

Visualization has several competing goals. A single visualization or series of visualizations has to find an appropriate balance.

- Correct characterization of the data
- Discriminability of relevant characteristics
- Stability and comparability across data sets

Below is a list of potential strategies for range selection. For each strategy, we also describe how one would normalize the data using that strategy.

1. Pick a range based on the min and max of the data.

If we choose a simple property of the data such as max and min to set the range, then plots of similar kinds of data can look extremely different.

- Outliers in the data control the range of colors, locations, sizes, etc.
- Data with similar values in different data sets may end up with different visualization properties
- Discriminability in the center of the data range may be impossible

We would like to use invariant properties of the data space to select the range for visualization. Or even if they can't be invariant, properties that are less susceptible to the above problems.

**Normalization.** To normalize the data, we linearly transform it so that the min becomes 0 and the max becomes 1.

**Example.** We have weights and heights for a population of males. If there are outliers (e.g. infants or giants), then we will not be able to see the trends for the average person as well (see Figure 3).

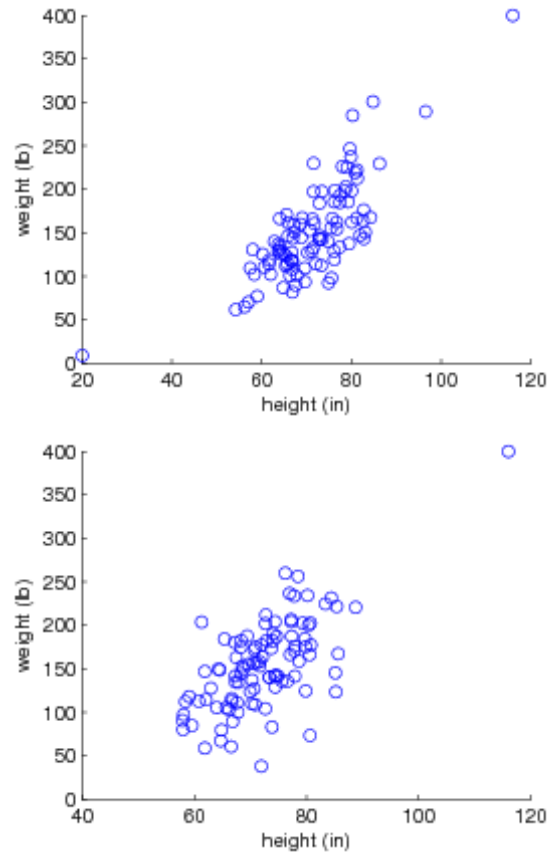


Figure 3: The presence of outliers makes it harder to focus on the data we are interested in. The data set of the right includes measurements for an infant and a giant. The data set on the right includes a giant only. At first glance, it looks like the population has gotten smaller from one data set to the next. But really, we had a change in outliers. Also, it is a little hard to see the trends for the more average population.

2. Pick a range based on the mean and standard deviation of the data.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (x_i - \mu)^2}$$

Data with a Gaussian distribution will be distributed so that most of the data (about 95%) is within 2 standard deviations of the mean. For such data sets, a range such as  $\pm 2$  standard deviations will generally produce a consistent, useful visualization with reasonable discriminability.

Using standard deviation to set the range does not work as well on data that does not follow a Gaussian distribution or has a significant number of outliers. Outliers cause the standard deviation to expand, compressing the middle of the plot and reducing discriminability.

**Normalization.** To normalize the data, we linearly transform. We may also need to make it possible to omit data points outside the 0 .. 1 range from the display. Doing the linear transformation is a two-step process:

First, compute the z-statistic according to

$$z_i = \frac{x_i - \mu}{\sigma} \tag{36}$$

The transformation in (36) creates what is called a z-value in statistics. It represents how far away from the mean a particular value lies in units of the standard deviation of the distribution. The z-value, therefore, is invariant to the particular mean and standard deviation of the variable. It represents a statistically meaningful differential and indicates where in the distribution the particular value lies.

Second, pick a cutoff, such as  $\pm 2\sigma$  as the extent of the view volume. Note that this means that not all of the data is initially visible. Since the “units” are in  $\sigma$ , the additional transformation is straight-forward. Here it is for the range  $[-\sigma, \sigma]$

$$\hat{x}_i = \frac{z_i + 1}{2} \tag{37}$$

**Example.** We continue looking at the heights/weights data. Looking within 2 standard deviations of data in both the x and y directions allows us to focus on the part of the data we are interested in. (see Figure 4).

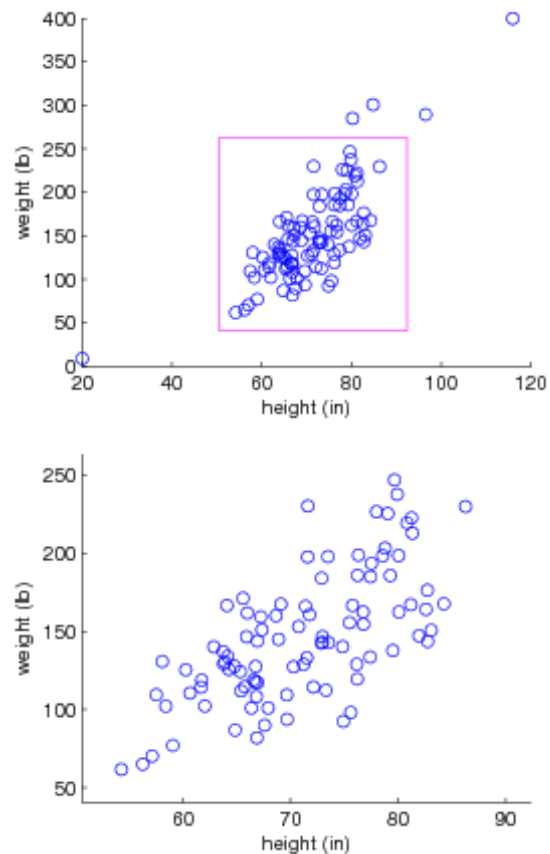


Figure 4: Two standard deviations lets us look at most of our data. On the left is the original plot, with the area covered by 2 standard deviations outlined in magenta. On the right is the view restricted to that box.

3. Pick a range based on the mean and mean absolute deviation (or average absolute deviation).

$$\text{MAD} = \frac{1}{N} \sum_{i=0}^{N-1} |x_i - \mu|$$

The mean absolute deviation is similar to the standard deviation, but it is less sensitive to outliers. For a Gaussian distribution, the MAD will be approximately 0.8 times the standard deviation.

**Normalization.** To normalize the data, we linearly transform. We may also need to make it possible to omit data points outside the 0 .. 1 range from the display.

**Example 2.** We continue looking at the heights/weights data. Looking within 2 MADs of data in both the x and y directions allows us to focus on the part of the data we are interested in. (see Figure 5). Note that although these data are normally distributed, they need not be.

4. Pick the range based on the potential data values.

For some data sets, the range of possible values is known. Selecting the range based on the set of possible values then enables consistent visualizations across different data sets with the same variables. If the actual data values cluster in one section of the range, this method of range selection will compress them, reducing discriminability of individual data points. However, if the intent is to compare the behavior of data sets and not individual points, this method enables consistent, comparable visualizations.

**Normalization.** To normalize the data, we linearly transform. We may also need to make it possible to omit data points outside the 0 .. 1 range from the display.

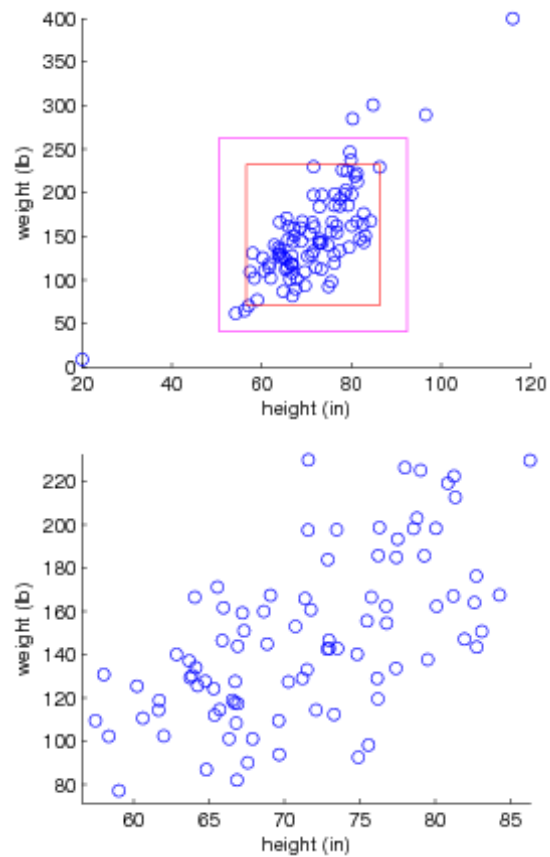


Figure 5: Two standard deviations lets us look at most of our data. On the left is the original plot, with the area covered by 2 standard deviations outlined in magenta and the area covered by two MADs outlined in red. On the right is the view restricted to the red box.

5. Pick the range based on potential values, but use bookend categories that capture outliers.

For example, pick a color range such as magenta to green. Let pure magenta represent any data value larger than an upper bound, and pure green any data value smaller than a lower bound. Pick the range between the upper and lower bound so that it covers most of the data.

We can achieve the same effect by using a sigmoid function to map data values to visualization values, such as colors. A sigmoid is a squashing function that takes the number line and squashes it to the range  $[0, 1]$ , maintaining the monotonicity of the original number line. There are a number of functions used as sigmoids, but the most common is the logistic function. All sigmoid functions are s-shaped, mapping large negative numbers to values close to zero and large positive numbers to values close to one.

$$S(x) = \frac{1}{1 + e^{-B(x-x_0)}} \quad (38)$$

The parameter  $x_0$  corresponds to the central point of the sigmoid curve. The parameter  $B$  controls the slope of the central part of the curve. Large  $B$  values produce steeper slopes, forcing the curve to be closer to a step function. Small values of  $B$  flatten the central slope, slowing the transition of the range from values close to 0 to values close to 1. The sigmoid is a generally useful function in data analysis, and we'll explore it more fully later in the course.

**Normalization.** To normalize the data, we use whichever function we want to use. For the example, this means we use the sigmoid function to accomplish the normalization for us.

**Example.** We continue looking at the heights/weights data. Now we want to include the amount each person exercises using color. In Figure 6 we show the height/weight data (no outliers) and a histogram of the exercise data (totally fabricated and in arbitrary units). There are outliers. So when we map the minimum exercise value to yellow and the maximum to blue, we find most of the data is a hideous middling color and we don't have much ability to make sense out of it (see Figure 7). If we simply bookend it between 8 and 32, we get much more discrimination (see Figure 8) and can conclude that the people who

exercise more also weigh more for their height (this must be people working out in a weight-lifting gym!). We could run it through the logistic function, but it only makes sense if the curve isn't too steep. When the curve is steep, more data points end up pure yellow or blue and we essentially split the population into two.

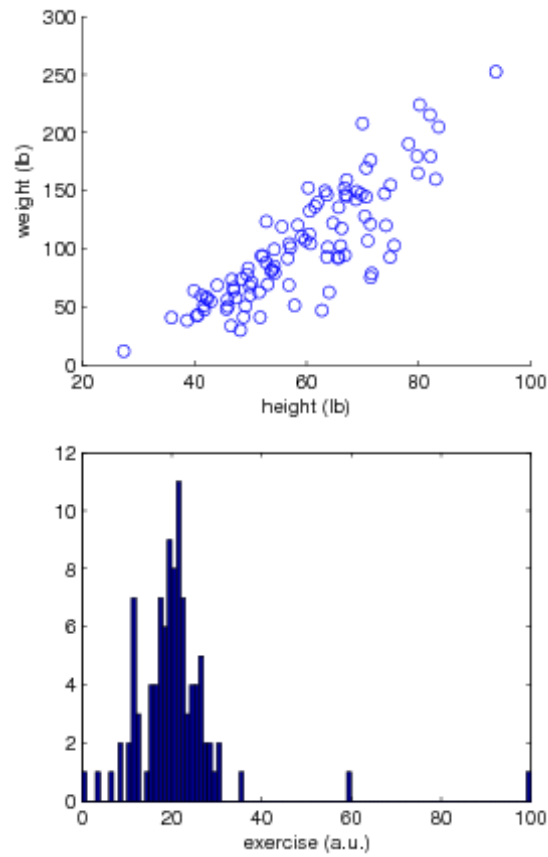


Figure 6: The data and how what we will use to color it. On the left are the heights and weights. We will color with exercise levels. On the right is a histogram of the exercise levels. There are outliers.

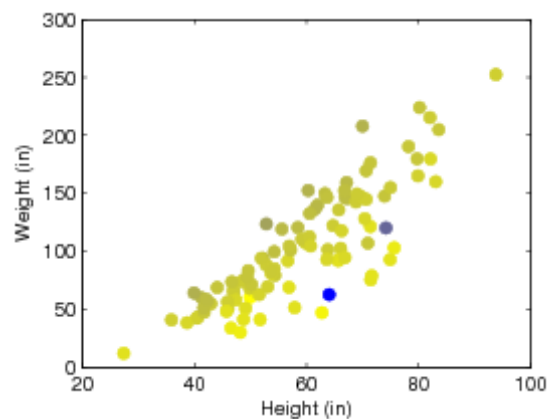


Figure 7: Color-coding linearly doesn't make sense. The outliers are yellow and blue. Everyone else is icky and too similarly colored.

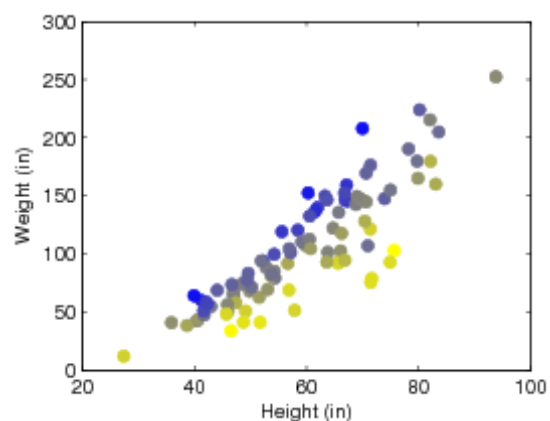


Figure 8: Bookending the data helps a lot. By making every value at or below 8 yellow and every value at or above 32 blue, we can now discriminate between the other data points. The bluer points are clearly lying “above” the yellow points.

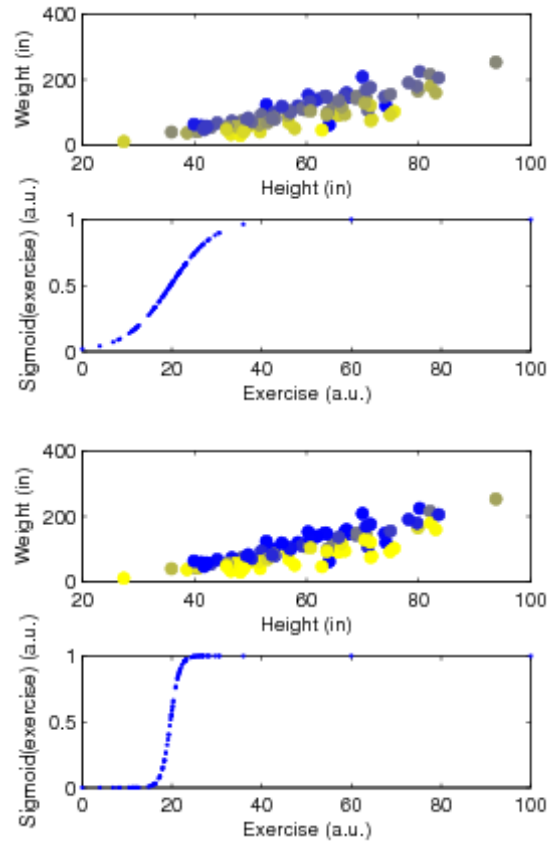


Figure 9: If we use a sigmoid, we don't want it to be that steep. On the left we use a sigmoid with  $B=0.2$  and on the right we use a sigmoid with  $B=1$ . When  $B=1$ , the data become split into to groups and there aren't many in the middle.

6. Pick the range based on desired visualization outcomes.

For example, if we're plotting temperature for a weather report, any temperature above 95F should appear as hot. Likewise, any temperature below 0F should appear as cold. These are subjective impressions, and in many situations it is not necessary to discriminate further.

**Normalization.** To normalize the data, we linearly transform. We may also need to make it possible to omit data points outside the 0 .. 1 range from the display.

7. Logarithmic scales to capture measurements with high dynamic range.

Data with high dynamic range, such as sound amplitudes, challenge linear visualizations. As the amplitude of a lawnmower may be several orders of magnitude higher than the sound of a person talking, trying to plot both on the same linear chart compresses most typical sounds into a very small range. If we were to add the amplitude of a jet engine, the lawnmower would also end up compressed.

A logarithmic scale enables us to discriminate between different quiet sounds and different loud sounds, but it makes distances less meaningful. Equal distance in log space means equal ratio transitions. Therefore, the change from 1 to 2 is spatially the same as the change from 10 to 20 or 1000 to 2000.

**Normalization.** To normalize the data, we transform by putting the data into log scale.

The above concerns apply equally well to color ranges or spatial ranges in visualization.

In good plotting programs, the user can control all aspects of the visualization. There is no universal method of picking a range that will work for all visualizations. There are methods that work reasonably well over a wide range of situations. If you know your data set, you can probably develop an algorithm that will work effectively for the user's purposes. Nevertheless, it is important to build flexibility into a visualization system and enable the user to tweak or adjust most range selection parameters.

### 8.4.1 Color

RGB: additive color scheme, used internally by computers to represent colors. While, as a programmer, we have to ultimately write pixel colors using RGB, the space is not always appropriate for color selection. In particular, distances in RGB space do not correspond to perceptual distances, as Figure 10 demonstrates.



Figure 10: Color distances. The color in the first column is equally different in RGB space from the colors in the second and third column

There are two useful magnitude axes in RGB space. People with normal color vision perceive red and green as opposite colors and yellow and blue as opposite colors. We can use this to display magnitudes on a red-green color axis or yellow-blue color axis.

$$C_{RG} = (\alpha, 1 - \alpha, 0.0)$$



$$C_{BY} = (1 - \alpha, 1 - \alpha, \alpha)$$



Interesting note: A common color spectrum for scientific data (e.g. the heat plots used to display microarray data) has been from red to black to green. But this is problematic for people with red-green color blindness (nearly 10% of the male population!). The field has been moving away from this, and in 2007, Nature Structure & Molecular Biology revised their style guide to ask that contrasts of red and green be avoided in graphs, models and schematics.

- Intensity is usually the average value of the color channels  $I = (R + G + B)/3$ .
- Intensity may also be a weighted average of the color channels  $Y = 0.299R + 0.587G + 0.114B$

## 8.5 Noise

Any discussion of data is incomplete without also discussing noise. Noise in a signal is the fraction of the signal not caused by the thing you want to measure. The signal to noise ratio [SNR] is commonly used to express the amount of noise in a system.

Technically, the SNR is the ratio of the average power of the true signal divided by the average power of the noise signal. Signal to noise ratios are often large, so engineers commonly use the decibel system to express them.

$$\text{SNR}_{\text{dB}} = 20 \log_{10} \left( \frac{A_s}{A_n} \right) \quad (39)$$

One decibel is a ratio of approximately 1.1:1. Twenty decibels is a ratio of 10:1 in amplitude. 100dB is a ratio of 100,000:1.

### Noise is described by the distribution it follows

There are many sources of noise, and there are many types of noise. From a mathematical point of view, we can subdivide noise into categories such as Gaussian, uniform, or falling in some other distribution. We can describe noise as biased or zero-mean, which describe the average value of the noise over time.

### White noise (not discussed in class)

In signal processing, noise is often described using colors. White noise is a noise signal that has equal energy at all frequencies within the bandwidth of interest. Pink noise, on the other hand, decreases in amplitude with higher frequencies. The noise at  $2f$  is half the power of the pink noise at  $f$ .

There is also a concept of white noise in space (rather than time) – a white noise image is made up of pixels whose values are chosen independently from a uniform distribution.

You can check on the Wikipedia page on white noise, if you are interested.

## 9 Monday March 6, 2017

(maybe) **Fundamentals of Data Mining and Machine Learning**

The goal for the course is to learn how to find meaningful patterns in data. We do that through judicious use of pre-processing, visualization and analysis. Before we go further, let's get some context. There are two, overlapping, subfields of computer science in which people are trying to find patterns in data. They are machine learning and data mining. Both use methods that are also used in the field of statistics.

**Machine learning.** Machine learning, a branch of artificial intelligence, is about the construction and study of systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between spam and non-spam messages. After learning, it can then be used to classify new email messages into spam and non-spam folders. (grabbed from the Wikipedia page on machine learning March 29, 2013)

**Data Mining.** The goal of data mining is finding and describing structural patterns in data (from the textbook).

A rule of thumb for the difference between machine learning and data mining is that machine learning focuses on prediction, based on known properties learned from the training data. Data mining focuses on the discovery of (previously) unknown properties on the data. (grabbed from the Wikipedia page on machine learning March 28, 2013).

We will be studying four categories of algorithms in this course.

1. Numerical prediction: the output to be predicted is not a discrete class but a numeric quantity. Regression is a popular form of numerical prediction.
2. Dimensionality reduction: Principal component analysis allows us to concisely capture structure in the data by forming a new (smaller) set of features. To do so, it takes advantage of the linear relationships among the original features.
3. Clustering: We seek to automatically group together observations with similar values.
4. Classification learning: the learning scheme is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples. This is also called *supervised* learning. We think of the majority of features as input and the classification as output. Since the training data we use in this sort of learning has known classes, it has known output and the learning is called supervised. The algorithms we will learn include Naïve Bayes method, decision trees, neural networks.

## 10 Data mining and machine learning must be performed intelligently

There are two “theorems” that together indicate there is no silver bullet. The first indicates there is no silver bullet algorithm. The second indicates there is no silver bullet distance metric.

### 10.1 No Free Lunch

The ‘No Free Lunch Theorem’ is one explanation for why machine learning methods require tuning, tweaking, and intelligent selection based on the data and the problem.

The NFL theorem states that all search and optimization algorithms have the same average performance over all problems. In other words, there is no

machine learning algorithm—which all fall into the category of search and/or optimization—that will consistently outperform all other algorithms on all problems. Since not all problems are the same, and different ML methods have different internal models, it stands to reason that some ML method is going to be optimal for a problem, but that must be balanced by the same ML method having relatively poorer performance on a different problem.

Note that the NFL theorem makes the assumption that all problems are equally likely. It may be the case that there is a bias to the type of problems that exist in the world, in which case some ML methods may dominate other methods, on average. Factors other than performance—training time, or run time, for example—are not part of the NFL theorem considerations.

It's also very important to keep in mind that factors other than the differences between general learning methods can play a much more significant role in overall performance. Decisions about how to use prior information, the distribution and number of training examples, and the particulars of the cost, error or reward functions can overwhelm any inherent positive or negative impact on performance due to the selected machine learning method.

## 10.2 Ugly Duckling

The ugly duckling theorem has to do with distance metrics (termed predicates), which are methods used to determine if two feature vectors represent the same category or different categories. The theorem considers all possible ways of measuring similarity and claims that, for any two patterns, the number of possible predicates claiming the patterns are similar is constant. As with the NFL theorem, the ugly duckling theorem implies that careful selection of the predicates for a particular set of categories is critical to the success of a pattern recognition system.

### Ugly Duckling Theorem

Given that we use a finite set of predicates that enables us to distinguish any two patterns under consideration, the number of predicates shared by any two such patterns is constant and independent of the choice of those patterns. Furthermore, if pattern similarity is based on the total number of predicates shared by

two patterns, then any two patterns are "equally similar."  
(Duda, Hart, and Stork, Pattern Classification, 2001)

Another way to look at the ugly duckling theorem is to think of the patterns as being highly dependent on the distance metric. You can get your data to follow almost any pattern if you choose the right (or wrong) distance metric. So you need to choose the distance metric that makes the most sense for your data.

**Take-home message:** There is no catch-all method that will find a reasonable pattern in all data sets. You must choose your algorithms carefully and pay attention to the details (i.e. choose an appropriate distance metric).

## 11 Monday March 6, 2017

### Linear Regression

Today we begin adding our first analysis method to the application – numerical prediction. For numerical prediction, we use the values of multiple, independent, numeric features to predict the numeric value of one dependent numeric feature.

I found the Wikipedia pages on linear least squares and regression analysis to be the most helpful. My notes are based on these pages.

- [http://en.wikipedia.org/wiki/Linear\\_least\\_squares\\_\(mathematics\)](http://en.wikipedia.org/wiki/Linear_least_squares_(mathematics))
- [http://en.wikipedia.org/wiki/Regression\\_analysis](http://en.wikipedia.org/wiki/Regression_analysis)

### 11.1 The Math

Generally, the goal of regression is to set up a model (equation) that is a function of the independent features. This model should predict the dependent feature. In linear regression, our model is a system of linear equations. Those linear equations involve the data or some function of the data. And the goal is to find the coefficients of all the lines such that the residual is minimized. What does that mean?

### 11.1.1 Simple linear regression

Let's consider a case in which we have  $N$  data points and one independent feature. We will use that independent feature  $\vec{x}$  to predict the dependent feature  $\vec{y}$ .

$$y_i = \hat{y}_i + r_i = \beta_0 + \beta_1 x_i + r_i, i = 0, \dots, N - 1$$

where  $r_i$  is the residual (the difference between the known value of  $y_i$  and the predicted value  $\hat{y}_i$ ).

The goal of regression in this case is to find the values of  $\vec{\beta} = (\beta_0, \beta_1)^T$  such that the sum of the square of the residuals is minimized:

$$\text{RSS} = \sum_{i=0}^{N-1} r_i^2 = \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2$$

### 11.1.2 Multiple linear regression

Suppose we have 3 independent features  $(\vec{x}_1, \vec{x}_2, \vec{x}_3)$ , then our model is

$$\hat{y}_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3}$$

where the notation is indicating that we want the  $i^{th}$  entry in the first, second, and third  $x$  vectors.

If there is more than one feature we would like to use, then it becomes more convenient to use matrix-vector notation.

$$\hat{\vec{y}} = X\vec{\beta}$$

where  $\hat{\vec{y}}$  is  $N \times 1$ ,  $X$  is  $N \times M$  and  $\vec{\beta}$  is  $M \times 1$ .  $X$  is constructed from a column of 1's and the data, e.g.

$$\begin{pmatrix} 1 & x_{0,1} & x_{0,2} & \dots & x_{0,M-1} \\ 1 & x_{1,1} & x_{1,2} & \dots & x_{1,M-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \dots & x_{N,M-1} \end{pmatrix}$$

where  $x_{*,i}$  is the column of data for the  $i^{th}$  independent feature.

### 11.1.3 Finding $\beta$

As long as the features are linearly independent (we cannot express one feature as a linear combination of the others), then we can use linear algebra to solve all of the equations for us. In fact, we can use Numpy's `linalg.solve` function to do it for us.

We use the **normal equations**

$$(X^T X)\hat{\beta} = X^T \vec{y}$$

This is a system with a unique value of  $\hat{\beta}$  that will satisfy the minimization problem. I.e. if we solve the normal equations for  $\hat{\beta}$ , then we have found the beta such that

$$\hat{\beta} = \arg \min_{\beta} \|\vec{y} - X\beta\|^2 = \arg \min_{\beta} \sum_{i=0}^{N-1} r_i^2 = \arg \min_{\beta} \text{RSS}$$

### 11.1.4 Subtleties

$X$  does not need to contain simply the columns of the data. If our data contains 2 features F1 and F2, our  $X$  matrix could include not only F1 and F2 but also their product or their squares. It will still be called linear regression because the model is linear once you have the  $X$  formed. (so it is linear in some transformation of the data, but not necessarily linear in the data).

## 11.2 Underlying assumptions

- The sample is representative of the population for the inference prediction.
- The error is a random variable with a mean of zero conditional on the explanatory variables.
- The independent variables are measured with no error. (Note: If this is not so, modeling may be done instead using errors-in-variables model techniques).
- The predictors are linearly independent, i.e. it is not possible to express any predictor as a linear combination of the others.
- The errors are uncorrelated, that is, the covariance matrix of the errors is diagonal and each non-zero element is the variance of the error.
- The variance of the error is constant across observations.

## 12 Lectures 19-22: Weeks of Mar 13-27, 2017

### 12.1 Covariance matrix

Variables in a data set may be related to one another, or they may be independent. The covariance matrix of a data set  $\Sigma$  gives us an idea of the first order relationships between different dimensions. The covariance matrix is defined by (40).

$$\Sigma(i, j) = \sum_{k=0}^N \frac{(x_{k,i} - \bar{x}_i)(x_{k,j} - \bar{x}_j)}{N - 1} \quad (40)$$

The diagonal entries of the covariance matrix are the variances of each dimension. The off-diagonals show the relationships between different dimensions. If the off-diagonals are close to zero, then the two dimensions are largely independent. If the off-diagonals are large, then the two dimensions are strongly related. A covariance matrix is symmetric, as the covariance of dimensions  $i$  and  $j$  is the same as the covariance of dimensions  $j$  and  $i$ .

## 12.2 Principal Component Analysis

Principal components analysis uses the covariance matrix  $\Sigma$  to identify the primary directions of variation in a set of data. The eigenvectors of the covariance matrix provide a basis space for the data that is tailored specifically to those variation directions. The eigenvalues tell us the relative importance of each of the eigenvectors.

An alternative method of computing similar information is to use SVD, or singular value decomposition to obtain the orthogonal basis vectors representing the primary modes of variation. SVD is the preferred method when the size of each data sample is large compared to the number of samples. For example, when calculating the basis vectors for a set of images of faces, each data sample is an image, potentially consisting of  $> 64,000$  pixels, while the whole data set may consist of only 200 images.

When the each data point is a small(ish) number of features and there are lots of them, we calculate the covariance matrix and compute its eigenvalues and eigenvectors.

Calculating the principal components using the covariance matrix:

- Calculate the covariance matrix for the data set. If we are use the convention of a row for each data point, then we need to tell it that the variables are in the columns, not the rows.

```
mcov = numpy.cov( m, rowvar=False )
```

- Calculate the eigenvalues and eigenvectors of the covariance matrix. The eig function in the linalg package provides this capability in numpy. The return value is two arrays. The first contains the eigenvalues. The second contains the eigenvectors as columns of the matrix (meigvec[:,i]).

```
(meigval, meigvec) = numpy.linalg.eig( mcov )
```

- The eigenvalues tell you the relative importance of the eigenvectors. Looking at the ratio of the eigenvalues is an indication of their relative importance. A commonly used analysis is to look at the cumulative sum of the eigenvalues from largest to smallest as a fraction of their total sum. If the first few eigenvalues represent 95% of the sum of

all the eigenvalues, for example, then the corresponding eigenvectors account for almost all of the variation in the data set.

- The eigenvectors tell you the directions of primary variation within the data. They are orthonormal vectors, which means the dot product of any two eigenvectors is zero and they have unit length.

Calculating the principal components using SVD:

- Calculate the mean data vector  $\vec{\mu}$ .

```
mu = m.mean( axis = 0 )
```

- Subtract  $\vec{\mu}$  from each data point to get a set of differential vectors  $\vec{d}$ .

```
mdiff = m.copy()
for i in range( mdiff.shape[0] ):
    mdiff[i] = mdiff[i] - mu
```

- Set up a matrix  $A$  where each differential vector  $\vec{d}$  is a column of  $A$ .
- Calculate the SVD of  $A$ , which generates the three matrices  $U, W, V^t$

```
(u, w, vt) = numpy.linalg.svd( mdiff.transpose() )
```

- The columns of  $U$  are the basis vectors of the columns of  $A$
- The rows of  $V^t$  are the basis vectors of the rows of  $A$
- The values in  $W$  are the singular values, and are related to the eigenvalues

$$w_i = \sqrt{e_i} \quad (41)$$

Projecting data onto the principal components

- Select how many principal components to keep. As noted above, it is useful to look at the fraction defined by the cumulative sum of the eigenvalues divided by their total sum. Sometimes you can keep only two or three eigenvectors (for visualization, for example), but other times you may want to choose enough eigenvectors to represent some percentage of the data variation (e.g. 90%).

- Take the dot product of each difference vector with the principal component directions. You can treat the eigenvector matrix as a rotation matrix. Each row of `deltadata` gets dotted with each column of `meigvec`.

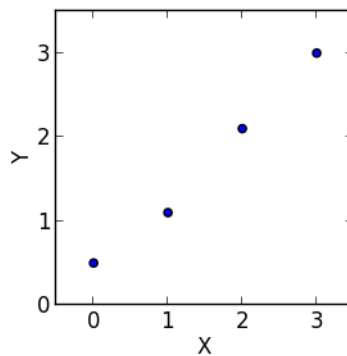
```
pdata = delldata * numpy.matrix( meigvec )
```

- The resulting set of feature vectors is a compressed representation of the data.

### 12.2.1 Toy Example

Let's perform a principle component analysis on a data set with 2 features:

| X | Y   |
|---|-----|
| 1 | 1.1 |
| 2 | 2.1 |
| 3 | 3   |
| 0 | 0.5 |



We compute the covariance  $C$ :

$$C = \begin{pmatrix} 1.6667 & 1.4167 \\ 1.4167 & 1.2158 \end{pmatrix}$$

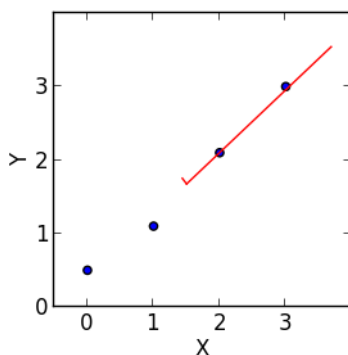
And then its eigenvectors (the columns of  $V$ ) and eigenvalues (the diagonal entries of  $D$ ):

$$V = \begin{pmatrix} -0.6492 & 0.7606 \\ 0.7606 & 0.6492 \end{pmatrix}, \quad D = \begin{pmatrix} 0.0068 & 0 \\ 0 & 2.8757 \end{pmatrix}$$

To make sense of the data, let's reorder the eigenvectors so that the one associated with the largest eigenvalue is first, and then they appear in decreasing order of significance. We call the eigenvectors the “principal components” and the eigenvector associated with the largest eigenvalue is called the first principal component.

|             |                                                           |                                                            |
|-------------|-----------------------------------------------------------|------------------------------------------------------------|
| Eigenvalue  | $\lambda_0$<br>2.8757                                     | $\lambda_1$<br>0.0068                                      |
| Eigenvector | $v_0$<br>$\begin{pmatrix} 0.7606 \\ 0.6492 \end{pmatrix}$ | $v_1$<br>$\begin{pmatrix} -0.6492 \\ 0.7606 \end{pmatrix}$ |

We learn that there is one dominant dimension of information in this case. The largest eigenvalue is 99.8% of the sum of all of the eigenvalues. If we plot the eigenvectors on the data, and scale their lengths by their eigenvalues, then we see the two orthogonal directions of information. Below, I plot the two eigenvectors on the scatter plot, scaling the length to indicate which is the dominant vector. Ideally, I would scale the length by the eigenvalue, but the second vector is too small, so I scaled it by 0.1. It still barely shows up, but you can see that it is short and perpendicular (orthogonal) to the first vector.



When we have more than 3 features, it is useful to project the data onto the first few principal components. For this example, we will do that projection in order to understand how it transforms the data.

First, we zero-center the data ( $\Delta X = X - \bar{X}$ ,  $\Delta Y = Y - \bar{Y}$ )

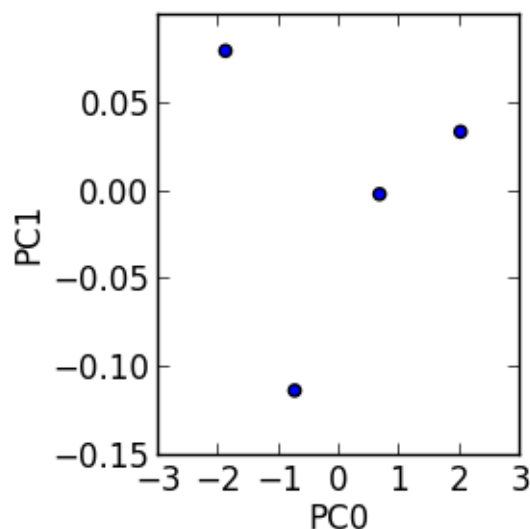
| $\Delta X$ | $\Delta Y$ |   | $\Delta X$ | $\Delta Y$ |
|------------|------------|---|------------|------------|
| 1-1.5      | 1.1-1.675  |   | -0.5       | -0.575     |
| 2-1.5      | 2.1-1.675  | = | 0.5        | 0.425      |
| 3-1.5      | 3-1.675    |   | 1.5        | 1.325      |
| 0-1.5      | 0.5-1.675  |   | -1.5       | -1.175     |

Then, we project the zero-centered data onto each of the principle components, using the notation  $p_1$  for the data projected onto the first principal component and  $p_2$  for the data projected onto the second principal component. We then scatter plot the data using the new coordinates.

$$\begin{aligned}
pc_0 &= (\Delta X \quad \Delta Y) v_0 \\
&= \begin{pmatrix} -0.5 & -0.575 \\ 0.5 & 0.425 \\ 1.5 & 1.325 \\ -1.5 & -1.175 \end{pmatrix} \begin{pmatrix} 0.7606 \\ 0.6492 \end{pmatrix} \\
&= \begin{pmatrix} -0.7536 \\ 0.6562 \\ 2.0011 \\ -1.9037 \end{pmatrix}
\end{aligned}$$

$$\begin{aligned}
pc_1 &= (\Delta X \quad \Delta Y) v_1 \\
&= \begin{pmatrix} -0.5 & -0.575 \\ 0.5 & 0.425 \\ 1.5 & 1.325 \\ -1.5 & -1.175 \end{pmatrix} \begin{pmatrix} -0.6492 \\ 0.7606 \end{pmatrix} \\
&= \begin{pmatrix} -0.11 \\ -0.0013 \\ 0.0341 \\ 0.08 \end{pmatrix}
\end{aligned}$$

When we plot the projection, we see that the data change shape - we no longer have a correlation.



### 12.3 Population data

The data set has 50 rows (one for each state) and lots of features. Two of the features are the latitude and longitude of the state's capital city. All of the remaining features are related to population. They include the population of males of all races of all ages, the population of females of all races of all ages, the population of males of all races under the age of 1, the population of males of all races between the ages of 1 and 4, and the population of Asian and Pacific Islander males of all ages. Basically, there are lots of population measures for different age groups and for either all races or for Asian and Pacific Islanders only.

We learn that the different population measures all tell the same story, and that this data set is really 1-dimensional. See Figures 11, 12, 13, and 14.

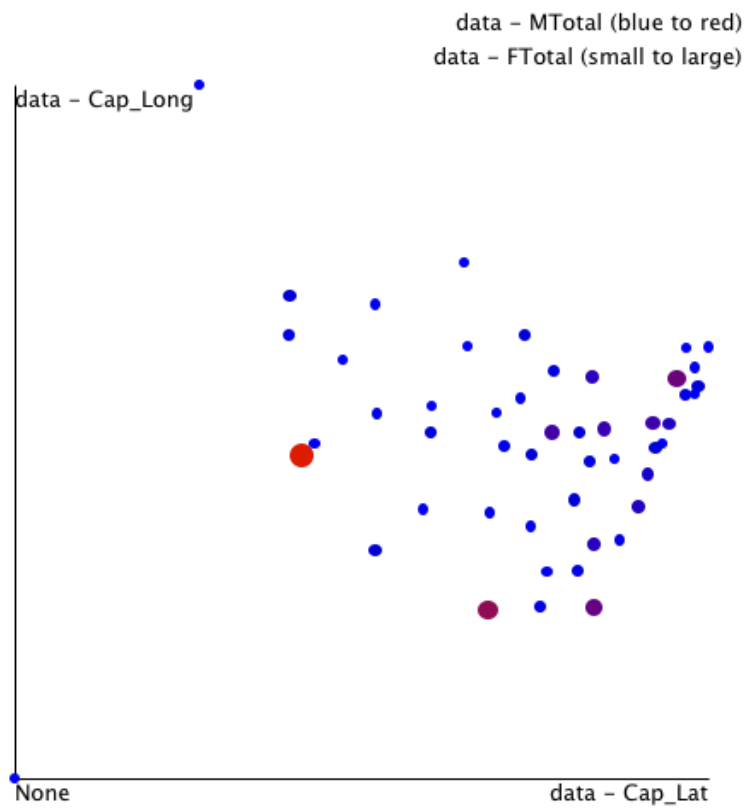


Figure 11: Plot of population data. The x- and y-coordinates are from the latitude and longitude. The color indicates the population size (blue is small and red is large), and the size indicates the number of females in the state (small is small, large is large).

|                     |                 |                 |                 |                 |
|---------------------|-----------------|-----------------|-----------------|-----------------|
| <b>Eigenvalue</b>   | <b>2.03807</b>  | <b>0.13873</b>  | <b>0.01831</b>  | <b>0.00276</b>  |
| <b>Energy</b>       | <b>0.92610</b>  | <b>0.06304</b>  | <b>0.00832</b>  | <b>0.00125</b>  |
| <b>Eigenvectors</b> |                 |                 |                 |                 |
| <b>MTotal</b>       | <b>-0.12850</b> | <b>-0.06036</b> | <b>-0.02697</b> | <b>0.02442</b>  |
| <b>Munder 1</b>     | <b>-0.12593</b> | <b>-0.04638</b> | <b>-0.16980</b> | <b>0.21655</b>  |
| <b>M1-4</b>         | <b>-0.12701</b> | <b>-0.05274</b> | <b>-0.17204</b> | <b>0.22027</b>  |
| <b>M5-9</b>         | <b>-0.12870</b> | <b>-0.06140</b> | <b>-0.15799</b> | <b>0.14111</b>  |
| <b>M10-14</b>       | <b>-0.12699</b> | <b>-0.05235</b> | <b>-0.10413</b> | <b>0.03881</b>  |
| <b>M15-44</b>       | <b>-0.12583</b> | <b>-0.04234</b> | <b>-0.06976</b> | <b>0.03029</b>  |
| <b>M15-24</b>       | <b>-0.12546</b> | <b>-0.04057</b> | <b>-0.06735</b> | <b>-0.01316</b> |
| <b>M15-19</b>       | <b>-0.12578</b> | <b>-0.04329</b> | <b>-0.05944</b> | <b>-0.05087</b> |
| <b>M15-17</b>       | <b>-0.12603</b> | <b>-0.04713</b> | <b>-0.06801</b> | <b>-0.01069</b> |
| <b>M18-19</b>       | <b>-0.12542</b> | <b>-0.03777</b> | <b>-0.04716</b> | <b>-0.10849</b> |

Figure 12: Some of the eigenvectors and eigenvectors of the population data's covariance matrix (excluding the latitude and longitude). We show only the first few values of the first view vectors. The first principal component dominates. From this, I conclude that there is really only one dimension of information – that it is population. So if your state's population of infant males is high, then your state's population of women over 85 is also high. The fact that all of the entries in the first principal component are of the same magnitude indicates that the data follow a diagonal through the F-dimensional space (where F is the number of population features). The fact that the first eigenvalue is huge indicates that most of the variation in the data is along that line, i.e. that the line is a fairly thin line of data (rather than, say, a cube). The fact that the signs on all of the entries are negative simply means that the vector is pointing back towards the origin.

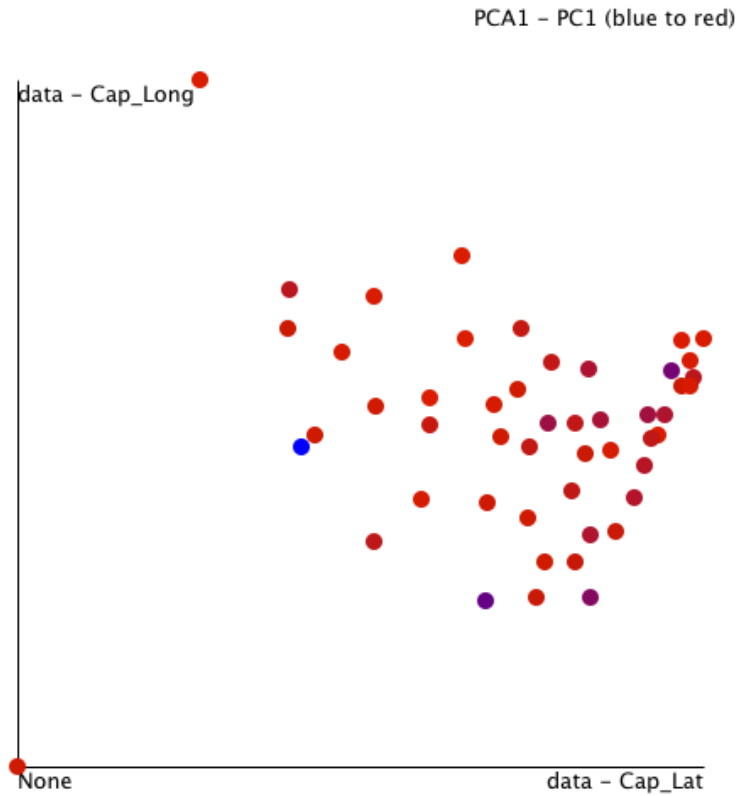


Figure 13: Plot of population data. The x- and y-coordinates are from the latitude and longitude. The color is the value along the first principal component. We can think of the first principal component as defining a new feature - one that is a linear combination of the other features. So it is an overall population feature. As shown in Figure 12, the values of the first principal component are all negative, so this new feature is going to head on the opposite direction of what we would expect. For this new feature, California will have the smallest value (blue), and Rhode Island will have the largest value (red).

|                     |                 |                 |                 |                 |
|---------------------|-----------------|-----------------|-----------------|-----------------|
| <b>Eigenvalue</b>   | <b>2.03917</b>  | <b>0.15104</b>  | <b>0.03554</b>  | <b>0.02243</b>  |
| <b>Energy</b>       | <b>0.89828</b>  | <b>0.06653</b>  | <b>0.01565</b>  | <b>0.00988</b>  |
| <b>Eigenvectors</b> |                 |                 |                 |                 |
| <b>Cap_Lat</b>      | <b>-0.00171</b> | <b>0.31436</b>  | <b>0.88819</b>  | <b>-0.26569</b> |
| <b>Cap_Long</b>     | <b>-0.02330</b> | <b>-0.06625</b> | <b>0.28121</b>  | <b>0.94103</b>  |
| <b>MTotal</b>       | <b>0.12847</b>  | <b>0.05648</b>  | <b>-0.02496</b> | <b>0.01380</b>  |
| <b>Munder 1</b>     | <b>0.12591</b>  | <b>0.04082</b>  | <b>-0.07126</b> | <b>-0.00122</b> |
| <b>M1-4</b>         | <b>0.12699</b>  | <b>0.04690</b>  | <b>-0.07357</b> | <b>-0.00355</b> |
| <b>M5-9</b>         | <b>0.12868</b>  | <b>0.05583</b>  | <b>-0.06305</b> | <b>0.00121</b>  |
| <b>M10-14</b>       | <b>0.12696</b>  | <b>0.04846</b>  | <b>-0.03497</b> | <b>0.01167</b>  |
| <b>M15-44</b>       | <b>0.12580</b>  | <b>0.03920</b>  | <b>-0.02531</b> | <b>0.01406</b>  |
| <b>M15-24</b>       | <b>0.12543</b>  | <b>0.03799</b>  | <b>-0.01690</b> | <b>0.01608</b>  |
| <b>M15-19</b>       | <b>0.12575</b>  | <b>0.04108</b>  | <b>-0.00948</b> | <b>0.01690</b>  |
| <b>M15-17</b>       | <b>0.12600</b>  | <b>0.04414</b>  | <b>-0.01999</b> | <b>0.01626</b>  |
| <b>M18-19</b>       | <b>0.12539</b>  | <b>0.03669</b>  | <b>0.00560</b>  | <b>0.01782</b>  |

Figure 14: Some of the eigenvectors and eigenvectors of the data's covariance matrix (including the latitude and longitude). We show only the first few values of the first view vectors. When we include the latitude and longitude information, the first principal component is not quite as clean as it was Figure 12. We draw the same basic conclusions – that the population measures all tell the same story. Latitude and longitude play almost no role in explaining the data (i.e. there is no linear pattern involving them).

## 12.4 SAT score data

The data set has 50 rows (one for each state) and several of features – latitude of the state’s capital city, longitude of the state’s capital city, the total male population in 2008, the total female population in 2008, the percentage of high-school students taking the SAT, the mean SAT verbal score, the mean SAT math score, the mean teacher’s salary, the mean ACT score, and the size of the state in square miles.

We learn that in states where few students take the SAT that the SAT scores are higher. We also observe that these states are located in the middle of the US, where the ACT is the more common exam. Likewise, we see higher mean ACT scores in states where the SAT is the most common test. We postulate that only the top students take both tests and that in regions in the northeast, where the SAT is common, only students who are good at taking tests or who think they are likely to do well on the ACT sign up for it. We also learn the the population and size of the state don’t seem to be related to test performance. See Figures 15, 16, 17, and 18.

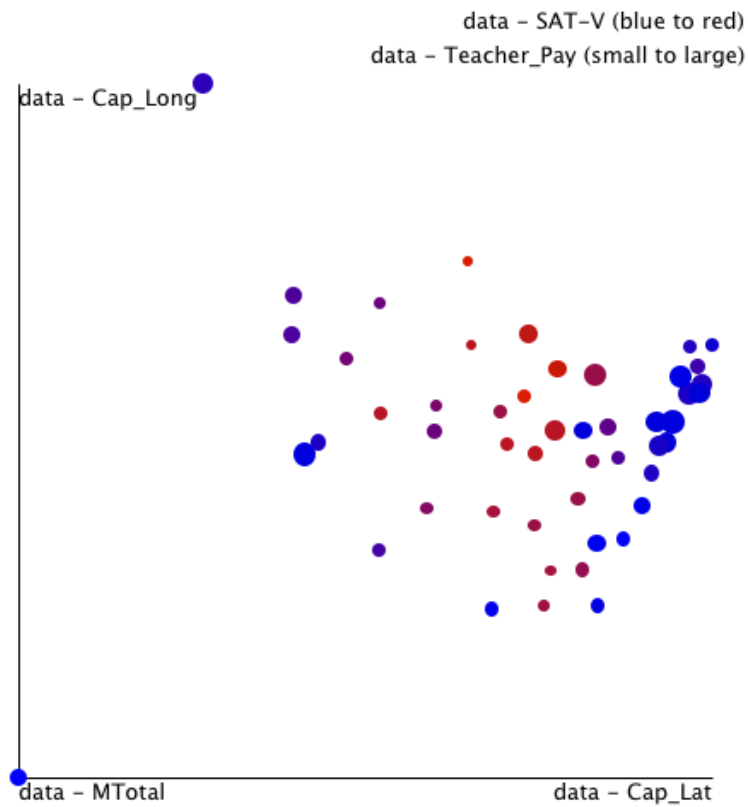


Figure 15: Plot of SAT data. The x- and y-coordinates are from the latitude and longitude. The color indicates mean SAT verbal score (blue is low and red is high), and the size indicates mean teacher pay in the state (small is low, large is high).

|              |          |          |          |          |          |          |          |
|--------------|----------|----------|----------|----------|----------|----------|----------|
| Eigenvalue   | 0.32875  | 0.07479  | 0.06776  | 0.02336  | 0.01931  | 0.00534  | 0.00198  |
| Energy       | 0.63063  | 0.14347  | 0.12999  | 0.04481  | 0.03703  | 0.01025  | 0.00379  |
| Eigenvectors |          |          |          |          |          |          |          |
| MTotat       | -0.12340 | -0.53811 | 0.27011  | -0.29964 | -0.15431 | -0.02148 | 0.03427  |
| FTotat       | -0.12730 | -0.54636 | 0.27172  | -0.30776 | -0.13968 | -0.03397 | 0.06037  |
| Teacher_Pay  | -0.32296 | -0.51273 | -0.31021 | 0.54765  | 0.46062  | 0.13896  | 0.07172  |
| %Taking      | -0.60187 | 0.14333  | -0.17893 | -0.08675 | -0.04236 | -0.75648 | -0.05979 |
| SAT-V        | 0.50818  | -0.17438 | -0.28102 | -0.00764 | 0.00479  | -0.42334 | 0.67293  |
| SAT-M        | 0.44979  | -0.29797 | -0.32235 | -0.02713 | -0.00428 | -0.27743 | -0.72606 |
| SqMiles      | 0.01250  | -0.08706 | 0.16619  | 0.65595  | -0.72403 | -0.09953 | -0.01238 |
| ACT          | -0.20322 | -0.04156 | -0.71726 | -0.27747 | -0.46738 | 0.37514  | 0.07946  |

Figure 16: Some of the eigenvectors and eigenvalues of the population data's covariance matrix (excluding the latitude and longitude). We show only the first seven vectors. The first principal component dominates, but not as strongly in the population-only data. By looking at the first vector, we conclude that the SAT math and verbal scores have similar information (same magnitude and same direction), and that the percentage of students taking the SAT is inversely related to the SAT score (the more people who take it, the lower the average score). We also conclude that teacher pay and the ACT score are weakly inversely related to the SAT score. But we don't want to make too strong a statement about those features. We can think of this first principal component as describing an overall student-performance feature. The second principal component also plays a role in this data set, and is determined by population, mostly. This data set isn't as clean as the population-only data set. There are multiple features that are only loosely related to each other.

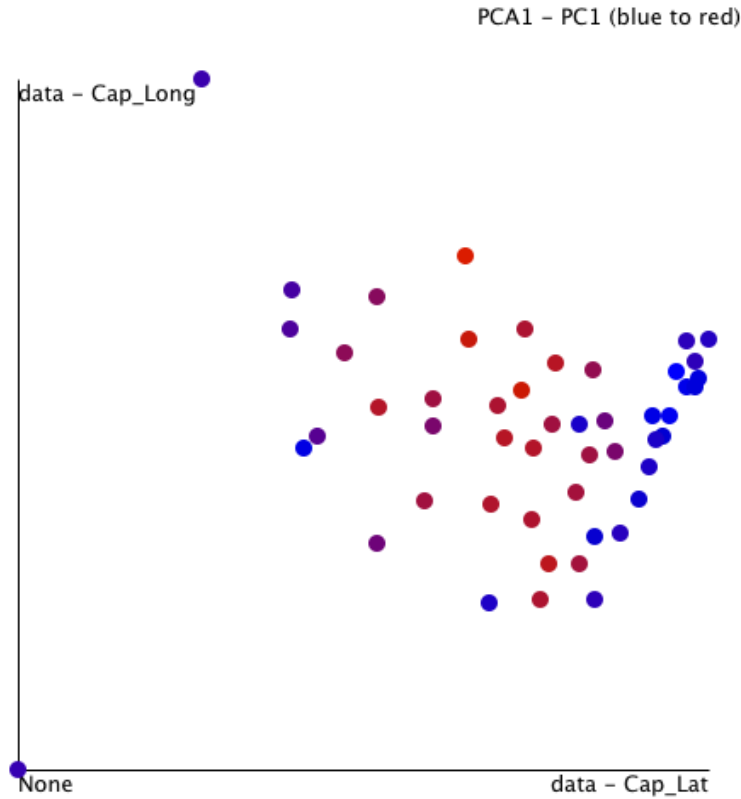


Figure 17: Plot of SAT data after having done PCA. The x- and y-coordinates are from the latitude and longitude. The color indicates value for a given state along the first principal component. You can either think of the color as an axis defined by the principal component, or you can think of it as a new feature that is a linear combination of the other features. Blue indicates small values (or large negative values) along the first principal component (lower SAT scores, higher ACT scores (sort of), higher teacher pay (sort of), higher percentages of students taking the SAT) and red indicates larger values (smaller negative values) along the principal component (higher SAT scores, lower ACT scores (sort of), lower teacher pay (sort of), lower percentages of students taking the SAT).

|              |          |          |          |          |          |          |          |          |          |
|--------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Eigenvalue   | 0.33211  | 0.08140  | 0.07488  | 0.05019  | 0.02193  | 0.01936  | 0.00500  | 0.00460  | 0.00121  |
| Energy       | 0.56224  | 0.13781  | 0.12677  | 0.08496  | 0.03713  | 0.03278  | 0.00846  | 0.00779  | 0.00205  |
| Eigenvectors |          |          |          |          |          |          |          |          |          |
| Cap_Lat      | 0.10770  | -0.29701 | -0.06154 | -0.74732 | -0.29661 | -0.41884 | 0.11372  | 0.19361  | -0.15495 |
| Cap_Long     | -0.00501 | -0.35644 | 0.01669  | 0.34783  | -0.21865 | -0.50363 | -0.00979 | -0.66764 | -0.06607 |
| MTotat       | 0.12172  | 0.20607  | 0.55096  | -0.12550 | 0.25446  | -0.20341 | -0.02090 | -0.09808 | 0.04429  |
| FTotat       | 0.12580  | 0.20544  | 0.55867  | -0.13980 | 0.24819  | -0.20524 | -0.02536 | -0.09814 | 0.04158  |
| Teacher_Pay  | 0.32193  | -0.29771 | 0.49601  | 0.10777  | -0.55909 | 0.45908  | 0.13650  | 0.05262  | 0.07312  |
| %Taking      | 0.60015  | -0.13819 | -0.15173 | 0.03009  | 0.09211  | -0.03471 | -0.75939 | 0.08233  | 0.06039  |
| SAT-V        | -0.50247 | -0.29885 | 0.14878  | -0.08144 | -0.03588 | -0.06326 | -0.30727 | 0.11751  | 0.71828  |
| SAT-M        | -0.44631 | -0.30926 | 0.27456  | 0.00364  | 0.11100  | 0.15970  | -0.38746 | 0.09123  | -0.65840 |
| SqMiles      | -0.01838 | 0.13697  | 0.11706  | 0.49145  | -0.23817 | -0.49567 | 0.01405  | 0.64555  | -0.07955 |
| ACT          | 0.20471  | -0.62340 | 0.00097  | 0.15669  | 0.58862  | -0.01671 | 0.38190  | 0.22160  | 0.05645  |

Figure 18: Some of the eigenvectors and eigenvalues of the population data's covariance matrix (including the latitude and longitude). We show only the first seven vectors. The first principal component dominates, but not as strongly as in Figure 16 which excluded the latitude and longitude data. We conclude that the latitude and longitude don't contribute to any linear patterns in the data.

We know that linear analyses are heavily affected by outliers and notice that Alaska and Hawaii are outliers for latitude and longitude and that California is an outlier for population. Will removing the outliers reveal a linear pattern in the geographic locations of the state or in their populations?

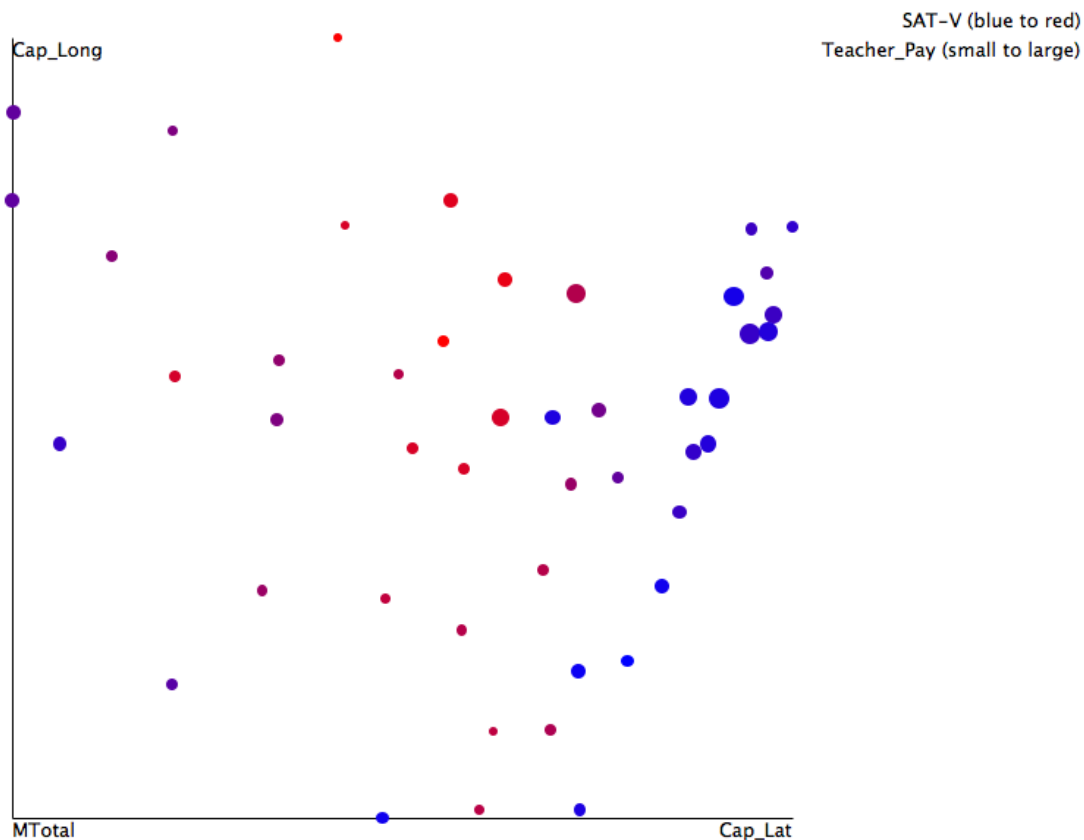


Figure 19: Plot of SAT data excluding AK, HI, and CA. The x- and y-coordinates are from the latitude and longitude. The color indicates mean SAT verbal score (blue is low and red is high), and the size indicates mean teacher pay in the state (small is low, large is high).

|                     |          |          |          |          |          |          |          |          |          |
|---------------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| <b>Eigenvalue</b>   | 0.36741  | 0.13115  | 0.08186  | 0.07016  | 0.02292  | 0.01223  | 0.00635  | 0.00448  | 0.00091  |
| <b>Energy</b>       | 0.52677  | 0.18804  | 0.11737  | 0.10059  | 0.03286  | 0.01753  | 0.00910  | 0.00642  | 0.00130  |
| <b>Eigenvectors</b> |          |          |          |          |          |          |          |          |          |
| Cap_Lat             | 0.28046  | -0.11350 | 0.24199  | -0.73393 | 0.34704  | 0.19792  | -0.36346 | 0.13964  | -0.00600 |
| Cap_Long            | -0.02039 | -0.56958 | 0.12001  | 0.38145  | 0.06917  | 0.70602  | -0.08686 | -0.02640 | -0.06116 |
| MTotat              | 0.14625  | 0.38180  | 0.43823  | 0.16860  | 0.18667  | 0.13620  | 0.15194  | -0.12337 | -0.02229 |
| FTotat              | 0.15346  | 0.38795  | 0.45449  | 0.15999  | 0.18598  | 0.15191  | 0.17595  | -0.15692 | -0.04038 |
| Teacher_Pay         | 0.30100  | -0.13176 | 0.47557  | 0.05317  | -0.73385 | -0.17552 | -0.28875 | 0.06117  | -0.08108 |
| %Taking             | 0.58118  | -0.12728 | -0.11517 | 0.13018  | 0.09347  | -0.06949 | 0.44463  | 0.63550  | -0.02521 |
| SAT-V               | -0.46654 | -0.14743 | 0.29635  | -0.13295 | 0.05629  | -0.13900 | 0.24075  | 0.25795  | -0.71279 |
| SAT-M               | -0.42358 | -0.16417 | 0.41191  | -0.03255 | 0.03657  | -0.10918 | 0.15738  | 0.33547  | 0.68714  |
| SqMiles             | -0.10299 | 0.24214  | -0.01947 | 0.42611  | 0.27246  | -0.13262 | -0.66337 | 0.45789  | -0.08145 |
| ACT                 | 0.19984  | -0.47491 | 0.17642  | 0.20430  | 0.42347  | -0.57988 | -0.04952 | -0.37896 | -0.00465 |

Figure 20: Some of the eigenvectors and eigenvectors of the population data's covariance matrix (including the latitude and longitude). We show only the first seven vectors. The first principal component dominates, but not as strongly as any other example form today. We conclude that removing outliers didn't help us.

## 13 Lectures 24-26: Week of Apr 3-7, 2017 (Clustering)

Clustering algorithms aim to separate the data into natural “clusters”. We use clustering when we are looking for groups of data points that have similar values for all the features. Our definition of “similar” will depend on the data and on the kinds of patterns we are looking for in the data.

### 13.1 K-means clustering

The goal of k-means clustering is to partition the data points (instances) into k clusters. Each cluster is defined by a “mean” and points are assigned to the cluster whose mean is closest. An effective k-means clustering will minimize the distance between each point and its cluster's “mean”.

### 13.1.1 Distance metrics

A distance metric determines the distance (a scalar) between two points ( $\vec{x}$  and  $\vec{y}$ ), so we write it as a function  $d(\vec{x}, \vec{y})$ .

The distance between two different points must be  $> 0$  and the distance between a point and itself must be 0.

Another rule we want to follow is that all distance metrics should obey the triangle inequality:

$$d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z})$$

The triangle inequality is important because it means that distances are meaningful, and there are no shortcuts or wormholes in the space. We would also like distance metrics to be commutative so that  $d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x})$ .

The classic k-means algorithm takes a geometric view of the data and uses the Euclidean distance to determine which to which cluster a point belongs. In keeping with that, it uses the centroid (arithmetic mean) to compute the “mean” of each cluster.

But it often makes sense to use a different distance metric. If you use a different distance metric, then you need to make sure you choose an appropriate method for computing the mean. So, let’s take a break and look at different distance metrics and their appropriate mean-computation methods. The following are the pairs of metrics available in Matlab’s k-means clustering function.

- Euclidean distance (also called the L2 norm). In this, the most common, metric, we view points in geometric space and use geometric metrics.

– Distance:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^F (x_k - y_k)^2}$$

Note: Matlab actually uses the square of the L2 norm, but I have listed it this way for the sake of simplicity.

- Mean: Compute the arithmetic mean of each feature independently. The mean  $\vec{m}$  is a point in  $F$ -space. For each component of  $\vec{m}$ , we compute the arithmetic mean according to

$$m_k = \frac{1}{N} \sum_{i=1}^N X_{i,k}$$

where  $N$  is the number of points in the cluster and  $X$  is the  $N \times F$  matrix of points in the cluster.

- Sum of absolute differences (also called the L1 norm).

- Distance:

$$d(\vec{x}, \vec{y}) = \sum_{k=1}^F |x_k - y_k|$$

- Mean: Each “mean” is the component-wise median of the points in that cluster.

- Correlation distance. For this metric, we are interested in how the components of two points vary (do they follow similar patterns?).

- Distance: Each point is treated as a sequence of values.

$$d(\vec{x}, \vec{y}) = 1 - \text{correlation-coefficient}(\vec{x}, \vec{y})$$

- Mean: Each “mean” is the component-wise mean of the points in that cluster, after centering and normalizing those points to zero mean and unit standard deviation.

- Hamming distance. This is suitable for binary data only.

- Distance: Percentage of bits that differ.

- Mean: Each “mean” is the component-wise median of the points in that cluster.

- Cosine distance. In this metric, we are interested in the angle between the data points (which we must view as vectors). It does not obey the triangle inequality because it is periodic. But that is OK, if we are careful how we apply it.

- Distance:

$$d(\vec{x}, \vec{y}) = 1 - \cos \theta = 1 - \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

- Mean: Compute the arithmetic mean of each feature independently, but first each point is normalized to unit Euclidean length.

$$\hat{X}_{i,*} = \frac{X_{i,*}}{\|X_{i,*}\|}$$

where  $\|\cdot\|$  is the length of the point (the square root of the sum of the squares of the entries),  $X$  is the  $N \times F$  matrix of points in the cluster, and  $\hat{X}$  is the  $N \times F$  matrix of normalized points in the cluster.

The mean  $\vec{m}$  is a point in  $F$ -space. For each component of  $\vec{m}$ , we compute the arithmetic mean according to

$$m_k = \frac{1}{N} \sum_{i=1}^N \hat{X}_{i,k}$$

where  $N$  is the number of points in the cluster.

### 13.1.2 K-means clustering algorithm

The goal of k-means clustering is to partition the data points (instances) into  $k$  clusters. Each cluster is defined by a “mean” and points are assigned to the cluster whose mean is closest.

The basic algorithm is

1. Estimate the means of each of  $k$  clusters. This can be done randomly (e.g. (a) pick the value for each feature from a uniform distribution over the range of that feature or (b) randomly pick  $k$  points from the data set) or with some human intelligence (e.g. look at the data and decide choose one point for each obvious cluster in the data).
2. While not done
  - (a) Place each point in the cluster whose mean is nearest to it

- (b) Recompute the mean of each cluster, given its members. If no point changes clusters, then the method has converged and we are done, so break out of the loop.

The above algorithm is not guaranteed to find the best possible k-means cluster. The method is sensitive to initial conditions (the original means). In practice, we run the algorithm many times from different initial conditions and use the best result we find.

## 14 Starting Week of April 10, 2017 - Classifiers

A couple of weeks ago, I mentioned that there were 4 types of machine learning algorithms that we would learn:

1. Numerical prediction: the output to be predicted is not a discrete class but a numeric quantity. Regression is a popular form of numerical prediction.
2. Dimensionality reduction (e.g. PCA)
3. Clustering: We seek to automatically group together observations with similar values.
4. Classification learning: the learning scheme is presented with a set of classified examples from which it is expected to learn a way of classifying unseen examples. This is also called supervised learning. We think of the majority of features as input and the classification as output. Since the training data we use in this sort of learning has known classes, it has known output and the learning is called supervised. The algorithms we will learn include Naïve Bayes method, decision trees, neural networks.

The goal of both clustering and classifying is to label or 'classify' the data points. Clustering does so without any knowledge of what labels the data should have. We are now going to transition to classifying the data points in

a supervised learning context. Our goal will be to examine labeled data and develop a set of rules so that we can take unlabeled data and label it so that the new data points fit in nicely with the old ones.

## 15 Week of April 10, 2017 - Naïve Bayes

One way to approach the problem of classification is through probability. Given a feature vector  $\vec{x}$ , what is the probability that it is in class  $c$ ? If we know what feature vectors are likely for members of class  $c$  and of all the other classes, then we can determine how likely it is that our feature vector describes a member of class  $c$ .

### 15.1 When there is 1 feature

First, we consider the case in which there is one feature  $x$ .

#### 15.1.1 Bayes Rule

We phrase the classification problem as a conditional probability: What is the probability the the given observation is of something in class  $c$ , given a measurement (feature)  $x$  (i.e. what is  $P(c|x)$ )?

How do we compute this probability? We use Baye's rule, which relates the conditional probability of  $c$  on  $x$  (the posterior) to the conditional probability of  $x$  on  $c$  (the likelihood), the probability of  $x$  (the evidence), and the probability of  $c$  (the prior).

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

#### 15.1.2 Doing the classification

If we have an observation  $x$ , and we want to predict its class, then we simply compute  $P(c|x)$  for each possible value of  $C$ . We pick the one with the largest probability.

Notice that the denominator will be the same for every possible class (value of  $C$ ). That means that we don't actually need to know the value of  $P(x)$  to perform the classification. So we won't try to compute it.

### 15.1.3 Computing the probabilities

But we do need to compute the likelihood  $P(x|c)$  and the prior  $P(c)$ .

A class's prior may be calculated

1. by assuming equiprobable classes (i.e., priors =  $1 / (\text{number of classes})$ ) (e.g. if there are two breeds of dogs, then we consider each breed equally likely), or
2. by calculating an estimate for the class probability from the training set (i.e., (prior for a given class) = (number of samples in the class) / (total number of samples)) (e.g. if 20% of the training set is a Jack Russell terrier, then we assume 20% of all dogs are Jack Russell terriers and that if we have a dog, there is a 20% chance of it being a Jack Russell terrier).

The likelihood is a little trickier, because we need a distribution. There are two basic approaches to finding this probability.

1. The first is to take all of the training data in the given class and to compute a histogram with it. Then, the probability of a particular value is just the fraction of training data points in its bin.
2. The second is to estimate a probability density function for a particular distribution. We use the density as a proxy of the probability. In this case, you need to know the distribution ahead of time.

In CS251, we will assume equiprobable classes (option 1 way above) and a Gaussian distribution (option 2 just above)

For a particular class  $c$ , we consider all of the observations (points) in class  $c$ . We compute the sample mean  $\bar{x}_c$  and sample standard deviation  $s_c$ . We use those parameters to specify the Gaussian density function we need. The pdf we need is this one:

$$p(x|c) = \frac{1}{\sqrt{2\pi s_c^2}} e^{-\frac{(x-\bar{x}_c)^2}{2s_c^2}}$$

If we assume the Gaussian distribution, then to build the classifier in Python code, we need to store three values for each class:

1. class mean:  $\bar{x}_c$
2. class variance:  $s_c^2$
3. class scale:  $\frac{1}{\sqrt{2\pi s_c^2}}$

#### 15.1.4 1D example

Suppose there are two breeds (classes) of dogs that will be weighed – Jack Russell terriers (breed 0) and German Shepherds (breed 1). Each observation  $x$  will be the weight in pounds.

| Weight (lbs) | Breed |
|--------------|-------|
| 15.0         | 0     |
| 14.0         | 0     |
| 16.0         | 0     |
| 50.0         | 1     |
| 60.0         | 1     |
| 70.0         | 1     |

The first three rows are for Jack Russells, and the quantities we need are

- class prior: 0.5
- class mean:  $\bar{x}_c = 15.0$
- class variance:  $s_c^2 = 0.6667$
- class scale:  $\frac{1}{\sqrt{2\pi s_c^2}} = 0.4886$

The last three rows are for German Shepherds, and the quantities we need are

- class prior: 0.5
- class mean:  $\bar{x}_c = 60.0$
- class variance:  $s_c^2 = 66.6667$
- class scale:  $\frac{1}{\sqrt{2\pi s_c^2}} = 0.04886$

The posterior values for weights between 0 and 100 pounds are shown in Figure 21.

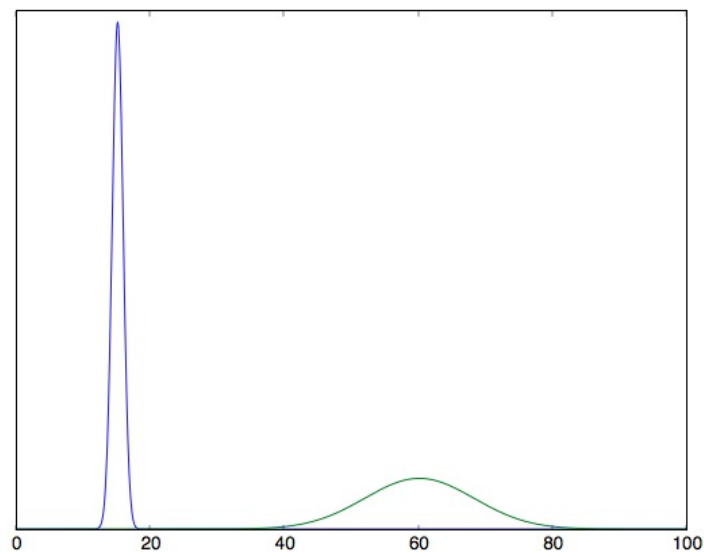


Figure 21: Posterior values for weights ranging from 0 to 100. The posterior values for Jack Russells are indicated by the blue curve (it peaks first and its peak is spiky). The posterior values for German Shepherds are indicated by the green curve (it peaks second)

## 15.2 When there is more than one feature

Now, let's return to the concept of a feature vector, instead of just a feature scalar. I.e. there is more than one feature.

### 15.2.1 Bayes Rule

We simply rephrase Bayes Rule using vector notation.

$$P(c|\vec{x}) = \frac{P(\vec{x}|c)P(c)}{P(\vec{x})}$$

which can be expanded to

$$P(c|x_1, \dots, x_F) = \frac{P(x_1, \dots, x_F|c)P(c)}{P(x_1, \dots, x_F)}$$

where  $F$  is the number of features.

Suppose there are two features, then

$$\begin{aligned} P(c|x_1, x_2) &= \frac{P(x_1, x_2|c)P(c)}{P(x_1, x_2)} \\ P(c|x_1, x_2) &= \frac{P(x_1|c)P(x_2|c, x_1)P(c)}{P(x_1)(P(x_2|x_1))} \end{aligned}$$

which requires us to know how feature 2 depends on feature 1. With more than 2 features, this equation gets more and more complicated. So what do we do? Do we throw up our hands?

No, instead, we make the wild assumption that all features are independent. Then we can multiply the probabilities of the individual features to find the probability of a combination of features. For the 2D case, this means:

$$P(c|x_1, x_2) = s \frac{P(x_1|c)P(x_2|c)P(c)}{P(x_1)P(x_2)}$$

In general, this means

$$P(c|\vec{x}) = \frac{1}{P(\vec{x})} P(c) \prod_{j=1}^F P(x_j|c)$$

where  $P(\vec{x})$  will end up being ignored when doing the classification, because it does not depend on  $c$ .

### 15.2.2 Doing the classification

We rephrase the classification in terms of feature vectors. If we have an observation  $\vec{x}$ , and we want to predict its class, then we simply compute  $P(c|\vec{x})$  for each possible value of  $c$ . We pick the one with the largest probability.

Notice that this doesn't change anything.

### 15.2.3 2D Example

Add to the 1D example the height of the dog in inches.

| Weight (lbs) | Height (in) | Breed |
|--------------|-------------|-------|
| 15.0         | 12.0        | 0     |
| 14.0         | 11.0        | 0     |
| 16.0         | 15.0        | 0     |
| 50.0         | 23.0        | 1     |
| 60.0         | 22.0        | 1     |
| 70.0         | 24.0        | 1     |

The first three rows are for Jack Russells, and the quantities we need are

- class prior: 0.5
- class mean:  $\bar{x}_c = (15.0, 12.66667)$
- class variance:  $s_c^2 = (0.6667, 2.88889)$
- class scale:  $\frac{1}{\sqrt{2\pi s_c^2}} = (0.4886, 0.2347)$

The last three rows are for German Shepherds, and the quantities we need are

- class prior: 0.5
- class mean:  $\bar{x}_c = (60.0, 23.0)$
- class variance:  $s_c^2 = (66.6667, 0.6667)$
- class scale:  $\frac{1}{\sqrt{2\pi s_c^2}} = (0.04886, 0.4886)$

## 16 Week of April 10, 2017 - K Nearest Neighbors

There is a classifier that is even simpler than Naive Bayes, but it is more computationally intensive.

It is called K Nearest Neighbors (KNN) and it is not related to k-means clustering. The K in this algorithm relates to how many close neighbors we need before we put a point into a category/class with others.

As with Naive Bayes, we need “training” data to build the classifier. Whereas with Naive Bayes, we computed histograms or parameters for a probability distribution for each class, with KNN, we simply store all the training data. It is easiest to think of it as a list of matrices with one matrix per class. The matrix that is the first element of the list contains the subset of points that are in class 0, the matrix that is the second element of the list contains the subset of points that are in class 1, etc.

To classify a new point, we compute the distances between this new point and all of the training data points (we call these the “exemplars”). Then, for each class, we find the K nearest neighbors to the new point and sum up those distances. This results in one number per class. We then determine which class the point belongs to by looking for the class associated with the smallest distance.

## 16.1 Reducing the number of exemplars

This method is expensive because we must compute the distance between a new point and every single exemplar point. If our training data set is large, this could be burdensome. In this case, we want to reduce the set of exemplars. One strategy for reducing the set of exemplars is to use k-means clustering. For each class, we perform a k-means clustering. Those k means then become the exemplar for the class. So if the class began with 50 points, we could do 10-means clustering and have 10 exemplar points.

## 17 Evaluating a Classifier

When we use a classifier, such as Naive Bayes, it's important to evaluate its effectiveness. How do we know it is working? How do we characterize the kinds of mistakes it makes?

In order to make an estimate of the quality of a classifier, we want to evaluate its performance on a test set, which should be data the classifier did not use during its training process. After running the data through the classifier, there are a number of methods of analyzing the results.

One of the most common methods of representing classifier results is as a **confusion matrix**. A confusion matrix indicates not only how many instances of each class  $C_1$  the classifier correctly labeled, but also how many instances of class  $C_i$  were labeled as each of the other classes  $C_j$ . For a 2-class problem, the confusion matrix is a 2x2 matrix. For an N class problem, the confusion matrix is NxN.

The rows represent the true class values and the columns represent the predicted class values. For example, consider the case in which we are classifying dogs as Jack Russell Terriers or German Shepherds. We could imagine a confusion matrix that looks like this:

| Label                | Predict Jack Russell | Predict German Shepherd |
|----------------------|----------------------|-------------------------|
| Jack Russell Terrier | 100                  | 0                       |
| German Shepherd      | 30                   | 70                      |

In this case the classifier has classified 200 dogs. It correctly identifies all 100 Jack Russell Terriers as such. But with German Shepherds, it doesn't do so

well (it mis-categorizes 30 of them as Jack Russells). Perhaps it is because we are accidentally considering puppies?

Next, consider the iris data set, in which we have 3 classes (one for each species of iris):

| Label                   | Iris-setosa (predicted) | Iris-versicolor | Iris-virginica (predicted) |
|-------------------------|-------------------------|-----------------|----------------------------|
| Iris-setosa (actual)    | 50                      | 0               | 0                          |
| Iris-versicolor         | 0                       | 47              | 3                          |
| Iris-virginica (actual) | 0                       | 3               | 47                         |

This classifier has done a pretty good job. However, there is some confusion. Which two iris species get confused?

## 17.1 Binary classification

When there are two classes, (i.e. it is a binary classification problem), then we use a special set of terminology –we speak of true and false positives and negatives. There are several common metrics compute to assess performance. It often useful to look at them per class value (e.g. in our dog data set to look at the “Jack Russell” class). To be general, we will call the class  $a$ .

- True Positives (TP): the number of data points in class  $a$  that are classified as class  $a$
- True Negatives (TN): the number of data points not in class  $a$  that are classified as some class other than  $a$
- False Positives (FP): the number of data points not in class  $a$  that are classified as class  $a$
- False Negatives (FN): the number of data points in class  $a$  that are classified as some class other than  $a$
- TP rate:  $TP/(TP+FN)$
- FP rate:  $FP/(FP+TN)$
- precision:  $TP/(TP+FP)$
- recall: same as TP rate

- F-Measure (used in information retrieval tasks):

$$\frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

(Note that I have defined these terms in such a way that it is possible to use them for classification with more than 2 classes. But it is more common to use them in binary classification. So I have put these terms here.)

Let's examine a confusion matrix for a binary classifier. For example, on a network inference task the confusion matrix may look like any of the three cases below.

| Label   | Predict Edge | Predict No Edge |
|---------|--------------|-----------------|
| Edge    | 60           | 40              |
| No Edge | 1            | 99              |

| Label   | Predict Edge | Predict No Edge |
|---------|--------------|-----------------|
| Edge    | 85           | 15              |
| No Edge | 13           | 87              |

| Label   | Predict Edge | Predict No Edge |
|---------|--------------|-----------------|
| Edge    | 98           | 02              |
| No Edge | 45           | 55              |

In the case (a), the system finds only 60% of the true edges, but it mostly avoids incorrectly detecting an edge when none exists. In case (b), the false negatives and false positives are balanced: the system makes false positive and false negative errors at about the same rate. In the case (c), the system finds almost all of the faces in the data set, but it also classifies many non-face examples as faces.

In my research, I vary the threshold for determining an edge is present. The lower the threshold, the more edges are detected. To see how all the method performs without having to fix the threshold, I plot a Receiver Operating Characteristic (ROC) curve. For each threshold, I compute the false positive

rate and the true positive rate. I associate the x-axis with the false positive rate and the y-axis with the true positive rate. I plot points for all the thresholds I test and find a curve that is slightly above the diagonal. This means, my method is better (but not by much) than random guessing.

Almost all machine learning methods have some type of parameter that lets the developer tune the performance of the system to encourage or discourage false positives or negatives. This means there are many possible operating points. The purpose of the system defines which operating point is most appropriate. (Note that for Naive Bayes, we don't have much to control. We will have more to control in future methods, though.)

## 17.2 Testing data

It is important to reserve some data for testing. Suppose we use all of our data to construct a tree, and that we have all pure leaves in the tree. Then we test our tree on the training data. We compute the error and find that all data points are correctly classified. Does that tell us anything about how well the tree will classify new data points? No.

There are two basic strategies:

1. Use separate data for validation and testing.
2. Use cross-validation. If you don't have a large data set, then you probably don't want to leave any data out of your training set. So, the solution is to separate your data into partitions (folds) and let each fold take a turn at being a testing set.
  - Separate your data into  $D$  folds
  - For each fold
    - Use the remaining data to train a classifier (e.g. build a tree)
    - Test the classifier using the data in the fold. This involves classifying the data point according to this tree.

Note that this means that each data point is used as a test point exactly once. So you have a predicted class for each data point. You also have the actual values of the class for each data point. So you can compute a confusion matrix and all of the other performance metrics listed above.

Regardless of which method you choose, you will want to be smart about how you partition the data. It is important to **stratify** the data – keep the proportion of data points from each class consistent with its proportion in the entire data set. So if 50% of your dogs are Jack Russells, then each partition should be 50% Jack Russell.

## 18 Week of April 17, 2017 – Decision Trees

Decisions trees are similar to the game “20 questions”, but the questions don’t have to be yes/no. The questions form a tree structure, with the first question being the starting node. Each leaf of the tree represents a label for the input feature vector. The particular set of questions in the tree depends on the form of the feature vector and the training data.

- A decision tree is a set of rules organized in a tree structure with a root node and leaves.
- Each node is a classification rule that subdivides the data set into two or more parts
- Each leaf is associated with an output rule
- To classify a novel instance, begin at the root node and use the results of each classification rule to get to a leaf node and its output rule.

Decisions trees can use many different types of classification rules.

- Enumerated types: one branch for each type, or one branch for each of a set of mutually exclusive subsets of the categories. If there is one branch for each type, the variable will not be used again lower down in the tree.
- Numeric types:
  - A simple threshold test of a variable with a constant value: two branches

- An interval test with different branches for above and below the interval: three branches
- Comparisons between different variables
- Missing data:
  - Make a special branch in the tree for the missing data category
  - Use the most popular branch for real data given the training set
  - Split the instance and send it down multiple branches, then recombine the outputs using the likelihood of each branch given the training set

Each leaf of a decision tree represents a single path through the tree from the root node. The collection of rules along the path form a single rule we could write as a (potentially complex) if statement. If we were to write down a rule for each leaf node, we would end up with a set of classification rules. We can take a novel instance and implement the same function as the decision tree using the set of leaf rules. One important characteristic of this set of rules is that the order in which they are applied to the instance is immaterial. Every instance will meet the criteria of one and only one rule in the set.

The set of if-then rules in a decision tree subdivides the problem space into many small pieces. The expectation is that, if the features are related to the desired output classes, then it should be possible to subdivide the space so the small pieces are close to uniform. There are many forms of decision trees. The key parameter of a decision tree is its size, or complexity. A more complex decision tree divides the input space into many smaller pieces, giving it the ability to specify complex and precise boundaries. However, a simpler decision tree may often capture the decision boundary more accurately and better generalize to new data.

When building decision trees, we are always trying to choose what question to ask next. When picking a question, there are two factors we want to measure. First, we would like to maximize how much we learn from the question, regardless of whether the answer is yes or no. One way of thinking about the information gain is by considering how much of the input space we discard, depending on the answer. The best we can do is discard half of the possible input space no matter what the answer is.

Second, we want the likelihood of the correct class to increase in the input space that remains. In other words, we are both subdividing the problem and making it more likely that if we make a guess about the output class that we'll be correct. The training set provides us with data with which we can evaluate different questions and choose what is the best question to ask next. The training set also tells us when a particular subset of the input space is likely to have a single output class, in which case we don't have to add more nodes to that part of the decision tree.

**Information Content** is a way of measuring how many more nodes, or questions, we are likely to need to complete a branch of a decision tree. The fewer additional nodes we need, the better the original question. A branch that is close to pure (only one output class) is likely to need fewer questions to complete (and is said to require “low information” – its information content is low). A branch that has training samples with approximately equal numbers of two output classes is likely to need more questions in order to separate the two classes (so its information value is high). The best possible question would separate the training data into one branch per output class, in which case the tree requires no additional questions and is complete.

Information theory tells us that **entropy** is the function that best measures information content. Entropy is defined as in Eq. (42).

$$E(\vec{cc}) = \sum_{k=1}^C -p(cc_k) \log_2(p(cc_k)) \quad (42)$$

where  $cc$  is a row matrix (in the context of decision trees, it specifies the “class counts” of the node).  $p(cc_k)$  is the probability that a new data point is in the  $k^{th}$  class. It is simply the fraction of data points (that are “in this node”) that are in the  $k^{th}$  class.

Consider a node that has training samples evenly spread between two classes, e.g. the class values list is  $[5, 5]$ . Its entropy is 1. We still need more branching.

Consider a node that has training samples all in the same class, e.g. the class values list is  $[0, 10]$ . Its entropy is 0. No new information is needed. If we get to this node in the tree, then we are confident we can classify the point that got us here.

To build a decision tree, we want each child node to have less information than its parent. To evaluate the effectiveness of a particular branching strategy, we

compare the entropy of the parent to the weighted average of the entropies of the children (think of this as the output information, or “out\_info”). We want the parent’s entropy to be higher than that of its children (as we progress down the tree, regardless of which branch we take, we want to lose the need for information and thereby increase our confidence in our ability to classify accurately). In decision tree parlance, this difference is called **information gain** and we want it to be as high as possible.

$$\text{information gain} = E - \text{out\_info} \quad (43)$$

For example, consider a node receiving 20 training samples of two classes that divide evenly as [10, 10]. The question at the node divides the data into 3 branches [6, 7, 7]. Within each branch, the division into two classes breaks down as ([4, 2], [5, 2], [1, 6]).

The information content in the incoming data is  $E([10, 10]) = -0.5 \log_{0.5} -0.5 \log 0.5 = 1.0$ . The information content in the outgoing branches is given by the entropy of the child nodes multiplied by their likelihood in the training set.

$$\begin{aligned} \text{out\_info} &= \frac{6}{20}E([4, 2]) + \frac{7}{20}E([5, 2]) + \frac{7}{20}E([1, 6]) \\ &= (6 * 0.918 + 7 * 0.863 + 7 * 0.591)/20 \\ &= 0.785 \end{aligned} \quad (44)$$

In this case, the information gain is  $1.0 - 0.785 = 0.215$ . Consider the same data, but a question that divides it into two branches with the subdivision into classes as ([9, 1], [1, 9]).

$$\begin{aligned} \text{out\_info} &= \frac{10}{20}E([9, 1]) + \frac{10}{20}E([1, 9]) \\ &= (10 * 0.469 + 10 * 0.469)/20 \\ &= 0.469 \end{aligned} \quad (45)$$

The alternative split would provide an information gain of 0.531, or more than twice the information gain of the first branching. Therefore, the second branching is likely a better choice.

## 18.1 Numeric attributes

There are many ways to go about making a decision node when the data is numeric:

- Fix the number of branches (e.g. 2), choose one feature to examine, and then use the information gain of the possible divisions to determine where best to split the data. (This is the one we will use.)
- Fix the number of branches and use a clustering algorithm ( $K=2$ ) to select the division point. This has the benefit of separating the data in a meaningful way, but doesn't necessarily do anything to improve the classification performance.
- Calculate the average value of the variable for each output class and pick decision points based on the means and standard deviations of the distributions.

All of these approaches are approximately equal in terms of expense. In all cases, every data value must be examined.

## 18.2 Algorithm for a 1R Tree

To make a 1R (1 rule) tree, calculate a rule for each attribute and pick the best one. In some cases, a complex decision tree is not warranted. For many tasks, there may be a single variable in the data set that acts as an effective proxy for the desired output classes. A 1R tree has a single decision node. In some cases, a 1R tree works as well or better than other classifier methods, especially for small training sets.

The information gain is a useful method of selecting which 1R tree to use. Consider an example data set with 3 features. There are three possible 1R trees. Each feature has a threshold divides the data set into two parts. The initial information content of the data set is  $[9, 11] = 0.993$ . The information gain for each of the 1R trees is given in table 1. From the table, feature three is clearly the best option and provides the highest information gain.

| Feature # | L Branch | L IC  | H Branch | H IC  | Information Gain |
|-----------|----------|-------|----------|-------|------------------|
| 1         | [5, 2]   | 0.863 | [4, 9]   | 0.890 | 0.112            |
| 2         | [4, 3]   | 0.985 | [5, 8]   | 0.961 | 0.024            |
| 3         | [8, 3]   | 0.845 | [1, 8]   | 0.503 | 0.302            |

Table 1: Information gain of 1R trees for a given data set

## 18.3 Writing the code

Stephanie wrote code to support the decision tree, making separate classes for the tree node and the tree itself. She chose to use recursive methods in the node class to execute all the necessary logic.

- `DecisionTreeNode` class
  - Stores the indices of the training points that “made it” to this node.
  - Stores the number of training points in each class that “made it” to this node.
  - Stores a reference to the full set of data and class values.
  - Stores the entropy for the node
  - Stores the class of the node (If this is a leaf node, then this is what will be used to do the classification).
  - Stores error of the node (we will talk about this later – it becomes important when we prune)
  - Has a method named `become_decision_node` which generates child nodes. It calls helper methods `find_split_point` and `compute_outinfo_of_this_split_point` to find the feature and threshold that create the best split value for the tree. This will be the value that produces the information gain. (Note that I did not make this method recursive, because it is designed to be used to build either 1R trees or trees with more depth.
  - Has a method named `classify` which takes in a data point. If this is a leaf node, it return the node’s class. Otherwise, it calls the method recursively on the appropriate child node.

The `OneRuleDecisionTree` class has a `DecisionTreeNode` as its root. To build a 1R tree, you pass in data, it creates a node, and then makes it become a decision node. The `classify` method

## 18.4 Simple Tree with more than 1 decision

There is no reason we need to stop with just one decision node and 2 leaf nodes. We could attempt to turn each leaf node into a decision node, expanding our tree as we go.

We have a helper method for `DecisionTreeNode` called `become_decision_node` that finds the best possible decision and makes the child nodes, if possible. If those children are made, then the left child will be referenced by `self.kids[0]` and the right child will be referenced by `self.kids[1]`.

To build a tree with as many branches as we need to get pure nodes (nodes with an entropy of 0), we can use this recursive method:

```
def make_simple_tree( self ):
    if self.my_entropy == 0.0:
        return
    self.become_decision_node()
    for kid in self.kids: # this will do nothing if there were no kids created.
        kid.make_simple_tree()
```

While the above procedure creates a perfectly fine decision tree, it does not balance generalization with specificity. The best decision tree algorithms use additional information to choose when to subdivide a node, and they prune branches from trees once they are completed. Otherwise, every single detail in the training data will contribute to the tree, whether or not it is relevant to the task. When the decision tree (or any classifier) captures all the noise of the data, instead of the underlying relationship between the classes and the features, it is called **over-fitting**.

It is important to note two attributes of the process.

- The decision about whether to build a node, and what question to ask is part of a local search process, or a **greedy search** process. That means the tree is built through a series of decisions that do not take into account global information about the structure of the tree, but

consider only the immediate value of a question on the components of the training set going through that branch.

- The decision process itself is heuristic, which means there is no provably correct method of picking a decision tree node. Experience and experimentation have yielded an algorithm that generally performs well within a greedy search process. However, the free lunch theorem suggests that certain parameters of the tree building process should be evaluated when building a tree for a particular task.

## 18.5 Pruning

Pruning decision trees is the trickiest part of building useful trees. Almost all data sets contain some noise, so creating a perfect decision may be either impossible or undesirable from a generalization standpoint. It may be better to stop subdividing a particular branch of the tree well before all of the attributes have been used or the branch is pure. Pruning uses two different methods of reducing the complexity of trees.

- Subtree replacement: eliminate a subtree and replace it with a single leaf node
- Subtree raising: replace a parent node by one of its subtrees

Subtree replacement, which is basically a traditional tree pruning operation, is the most common and effective method of decision tree pruning. The basic idea is to prune away leaves if their parent node would be “good enough”.. The approach is to look at the expected error rate of a subtree and compare it to the expected error rate of replacing the subtree with a single leaf node. If the expected error rate of the subtree is not significantly better than the error rate of the single leaf node, then prune away the leaves.

We can estimate the error rate of a node by using the formula for occurrences of a class resulting from a Bernoulli process. A Bernoulli process models coin flipping multiple times, according to some (unknown, true) probability. We want to use our knowledge of the observed error rate (the fraction of data points in the node that are not in the majority class) to estimate the true error rate. To do that, we find the upper bound of a confidence interval

around it. Given a confidence limit  $c$ , we can use a lookup table to find the  $z$ -value that corresponds to it. For example, a 25% confidence limit is a  $z$ -value of 0.69, or 0.69 standard deviations. Because we end up using training data to estimate the expected error rate (not a great idea) we have to use a pessimistic estimate of the error, which is basically an upper bound on the error (with  $c$  confidence). If  $f$  is the error rate,  $N$  is the number of samples, and  $z$  is the  $z$ -value corresponding to our chosen confidence, the upper bound on the true error is given by (46).

$$e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \quad (46)$$

If a node has 5 examples of class A and 14 examples of class B, then replacing the decision node by a leaf that guesses B results in 5 errors, or an error rate of  $f = 5/19$ . Using a confidence of 25%, or  $z = 0.69$ , the according to (46), the upper bound on the error at the node is 0.338. We can then calculate the upper bound of the error on each child node (which should be a leaf), combine the estimates using a weighted average, and compare the result to the estimated error of the node itself. If the combined error of the leaves is less than the error at the node, then we do not prune away the children. If the error of the children is greater than the estimated error of the parent, however, then we can prune away the children and replace the node with a leaf. We work from the bottom of the tree (the leaves) up, pruning as we go.

Pruning is an essential part of improving the performance of decisions on independent test data. Pruning can also generate much simpler subtrees, improving both the ability of people to understand the rules and the overall speed of the classifier.

## 18.6 Subtleties

### 18.6.1 Too many branches

One problem with using information gain directly is that it tends to prefer decision trees that subdivide the task into many small pieces. A method often used to counter this tendency is to express the information gain as the ratio of the node's information gain to the entropy of how the data gets

split by the question (intrinsic information). The entropy of the data split is simply the information content of the subdivision of the data into the branches regardless of class. A question that subdivides all of the incoming training samples into equal parts has a high entropy. A question that sends a large number of samples into one branch and few into another has a low entropy. Using the ratio of information gain to intrinsic information favors nodes that produce fewer divisions, but balances that with the ability of the question to subdivide the task in a useful manner.

The intrinsic information in the 3-way split is  $E(6, 7, 7) = 1.718$ . The intrinsic information in the 2-way split is  $E(10, 10) = 1$ . Therefore, even if the two questions produced similar information gains, the two-way split gives a smaller denominator and should be preferred because it gives a larger information gain ratio.

In some cases the gain ratio overcompensates, so a heuristic is to select the node with the largest gain ratio, so long as its information gain is at least as big as the average information gain for all nodes considered. That way a node with a small information gain, but a large gain ratio, does not dominate over a node that has a higher than average information gain, but not as equal a division of the data.

**Incorporating the subtlety:** Instead of using the information gain in our algorithms, we may want to use the gain ratio.

### 18.6.2 Missing values

Missing values can present problems when building a decision tree. As noted earlier, one solution to missing attributes is to send the instance down two branches and then combine the results of the resulting leaves using a weighted average. It is possible to do the same thing when building the decision tree by using non-integer branch weights when calculating the information gain for a node. For sub-trees after a split decision, the training sample still contributes via its likelihood of going down that branch.

## 19 Week of May 1-5: Neural Networks

Artificial neural networks (ANNs) are used to classify numeric data, taking numeric data as input, processing it through a network of nodes, and outputting the class for that input.

Before we think about how ANNs work, let's just look at their structure. An ANN is a set of interconnected nodes:

- Information flows between the nodes
- A weight is associated with each input to a node
- The output of a node is a function of its weighted inputs
- For general-purpose ANNs, the function at each node is nonlinear

A feedforward ANN is a subset of the possible ANN structures. A feedforward layer has three distinct types of nodes: input, hidden, and output.

- Information in a feedforward ANN starts from an input layer, to which an input sample is applied.
- The information then flows through one or more hidden layers
- The result of the computation is accessed in the output layer.

A feedforward ANN is like a combinational circuit in digital logic; its output is purely a function of its input. There are other kinds of ANNs where the output of at least one nodes feeds back into a prior node. These are called recurrent networks and function as sequential circuits with the outputs being a function of both the input and the prior output of the ANN.

The function executed by node  $j$  to generate its output  $O_j$  is generally of the form

$$O_j = \Phi(I_j) \tag{47}$$

where  $I_j$  is the integrated input to node  $j$  and is computed according to

$$I_j = \sum_{i=0}^{N-1} w_{ji} O_i \tag{48}$$

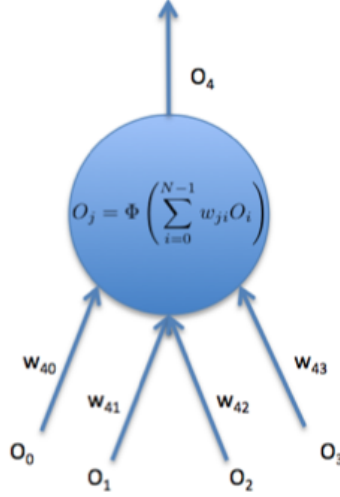


Figure 22: Artificial neural network node showing inputs, input weights, and the output.

An important practical aspect of ANNs is that each node generally has a bias term, or constant term in the summation. In other words, if there are  $N-1$  actual input values  $(O_1, O_{N-1})$ , then there is one other input  $O_0 = 1$  with a constant value. The value of the bias term weight  $w_{j0}$  is learned in exactly the same manner as every other weight in the network. Each node in the network has a connection, and therefore a weight, to the bias node.

The nonlinear function  $\Phi(x)$  is generally some kind of squashing function that produces a value within a fixed range from an arbitrary range input. The sigmoid function, for example, is commonly used.

$$\Phi(x) = \frac{1}{1 + e^{-x}} \quad (49)$$

The sigmoid function takes values on the real number line and modifies them to be in the range  $(0, 1)$ .

## 19.1 Example

Below is a simple example of a feedforward neural network trained on the XOR function. We set the values of A and B to be 0.1 for False and 0.9 for True.

- Two input nodes accept the input pattern.
- Two hidden nodes are fully connected to both input nodes.
- Two output nodes provide the result of the network.

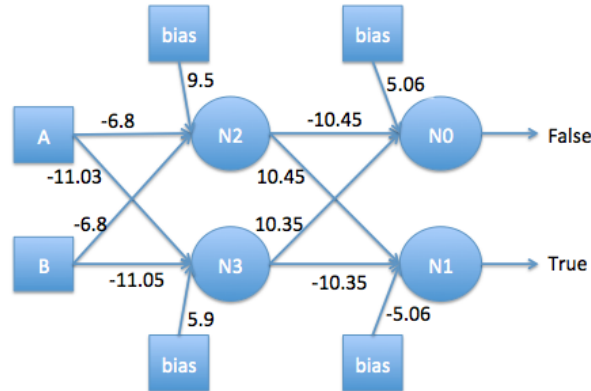


Figure 23: Artificial neural network trained to learn the XOR function. If N0 outputs a value near 1 and N1 outputs a value near 0, then the result is False. If N1 outputs a value near 0 and N1 outputs a value near 1, then the result is True.

With the given weights, here are the integrated inputs ( $I_j$ ) and outputs ( $O_j$ ) of each node, for each input.

For input A=0.1, B=0.1,

| Node   | Integrated input | Output |
|--------|------------------|--------|
| Node 2 | 8.140            | 0.999  |
| Node 3 | 3.692            | 0.976  |
| Node 0 | 4.711            | 0.991  |
| Node 1 | -4.711           | 0.009  |

For input A=0.1, B= 0.9,

| Node   | Integrated input | Output |
|--------|------------------|--------|
| Node 2 | 2.70             | 0.937  |
| Node 3 | -5.148           | 0.006  |
| Node 0 | -4.672           | 0.009  |
| Node 1 | 4.672            | 0.991  |

For input A=0.9, B=0.1,

| Node   | Integrated input | Output |
|--------|------------------|--------|
| Node 2 | 2.70             | 0.937  |
| Node 3 | -5.132           | 0.006  |
| Node 0 | -4.671           | 0.009  |
| Node 1 | 4.67117          | 0.991  |

For input A=0.9, B=0.9,

| Node   | Integrated input | Output   |
|--------|------------------|----------|
| Node 2 | -2.740           | 0.061    |
| Node 3 | -13.972          | 0.000001 |
| Node 0 | 4.426            | 0.988    |
| Node 1 | -4.426           | 0.012    |

Note that we have two outputs for two classes and two inputs for the two features. If we had 10 classes, then we would have 10 output nodes. That means networks need to be pretty big if there are lots of classes or lots of features. It is relatively easy to see why we need one input node per feature. One obvious question, though, is why not just encode the output in  $\log_2(C)$  nodes (where  $C$  is the number of classes)?. I.e. for our example, why not have one output node in which a high output value indicates True and a low output value indicates False? The answer is that we want to build some sanity checks into our output. If we assign one output node per class, then the network will give us a correct answer only if we have exactly one of the nodes with a high value and the rest with low values. If there is more than one output node with a high value, then we can detect that the network doesn't understand how to classify that input. This is important to know!

## Supervised Training

While one could create an artificial neural network by hand, it is very difficult to set the weights of a network appropriately. Instead, we want to use a training method that learns the weights from a training set consisting of labeled examples.

- Training set: a set of labeled data containing examples of all classes the network needs to see.
- Testing set: a similarly labeled data set on which the network never trains, but which is used to evaluate the network's performance.

The most common training method for feedforward neural networks is called backpropagation. It is a gradient descent algorithm that modifies the weights to improve the performance of the network on the training set. Gradient descent algorithms are greedy iterative algorithms that take one step towards the solution each iteration, making the change that maximally minimizes an error metric.

The process of training using a single pattern is as follows:

- Apply the training pattern to the network (i.e. forward propagate a data point as input)
- Calculate the output of the network and compare it to the desired output (the known class).
- Calculate the change to the weights for the output layer to make the actual and desired outputs closer.
- Move back through the network, calculating how to change the weights at each layer.

Applying all of the training patterns to the network is called an **epoch**. There are three algorithms for an epoch of training

1. On-line:

For each training pattern (input),

- Apply the training patterns as above
  - Apply the weight changes to the network, modified by a learning rate
2. Stochastic: This is the same as on-line, but randomly permutes the order in which the training patterns are applied.
  3. Batch:
    - Apply all of the training patterns as above and store (but don't apply) the weight changes
    - Apply some function of the stored the weight changes to the network, modified by a learning rate. This function could be the sum, mean, or median.

Training can often take hundreds or thousands of epochs.

You can visualize the error surface of the network as a set of valleys, hills, and plateaus. Each point on the error surface represents a set of values for the network weights. The best network, given the training and testing data, is the one at the bottom of the deepest valley, the lowest point on the error surface.

Backpropagation training generally begins with the weights of the network set to small, non-zero random values. Imagine the network starting at some randomly selected point on the error surface. By calculating the impact on the error rate of changing each weight in a specific manner, the backpropagation algorithm can determine which changes correspond to the optimal step down the error surface. By iterating that process, the network moves down the error surface towards the nearest valley.

Problems can arise with any greedy gradient-descent algorithm when the error surface is bumpy or has lots of valleys other than the deepest one. Since the network moves towards the closest valley, where the network starts can determine whether it can learn the problem optimally. The situation is similar to K-means clustering, where the initial cluster means end up determining how the data set is divided.

As with K-means clustering, one method of finding the global optimum is to train the network multiple times, starting it in different locations on the error

surface each time. Other methods modify the backpropagation algorithm by introducing terms into the optimization that allow it to skip over or bounce out of local minima (valleys).

## Algorithm for a single training pattern

1. Execute a forward pass on a training pattern to calculate the output of the network  $O$ .
2. Calculate the mean squared error between the output  $O$  and the desired output  $T$ .

$$E = \frac{1}{N_O} \sum_h (o_h - t_h)^2 \quad (50)$$

where  $N_O$  is the number of output nodes and  $h$  takes on the values of their indices.

3. Calculate the weight changes required to move  $O$  closer to  $T$ 
  - Weight changes are calculated using the **generalized delta rule**

$$\Delta w_{ji} = \eta \delta_i o_j \quad (51)$$

- $w_{ji}$  is the the weight going from node  $j$  to node  $i$
- $\Delta w_{ji}$  is the change that should be made to  $w_{ji}$
- $\eta$  is the learning constant, and governs the magnitude of the weight changes
- $\delta_i$  relates the input of node  $i$  to its influence on the output
- $o_j$  is the output of node  $j$
- The weight change ought to be proportional to the amount of change in the overall error realized by changing the weight. In other words, we need to know the derivative of the error with respect to changes in the weight.

$$\Delta w_{ji} \propto -\frac{\partial E}{\partial w_{ji}} \quad (52)$$

- To figure out this derivative, we need to have a complete expression connecting the inputs and outputs of a node. The overall input to a node is a weighted sum of the outputs of all connected nodes.

$$I_i = \sum_k w_{ki} o_k \quad (53)$$

where  $k$  takes on the values of the indices of all of the nodes that send output to node  $i$ .

- Given the expression for the overall input to a node, the output of a node is simply

$$o_j = \Phi(I_j) = \frac{1}{1 + e^{-I_j}} \quad (54)$$

- The derivative of the sigmoid function is simple to calculate.

$$\Phi'(x) = \Phi(x)(1 - \Phi(x)) \quad (55)$$

- Now we can divide the derivative from above into two parts, reflecting the change in the error with respect to the input of a node and the change in the input to a node with respect to a single weight.

$$-\frac{\partial E}{\partial w_{ji}} = -\frac{\partial E}{\partial I_i} \frac{\partial I_i}{\partial w_{ji}} \quad (56)$$

- The second term in (56) is the derivative of the input equation (53)

$$\frac{\partial I_i}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k w_{ki} o_k = o_j \quad (57)$$

and explains the  $o_j$  in the generalized delta rule. (Note about the math:  $j = w$  for one term and that is the term for which the derivative is non-zero.)

- The  $\delta$  term in the delta rule corresponds to the remaining partial derivative, which we can again expand.

$$-\frac{\partial E}{\partial I_i} = -\frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial I_i} = -\frac{\partial E}{\partial o_i} \Phi'(I_i) \quad (58)$$

- The second term of (58) is given by  $\Phi'(I_j)$ . The first term is different for output units and hidden units.
- For an output node, the differential of the error term with respect to its output is given by (59).

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i} \left( \frac{1}{N_O} \sum_h (o_h - t_h)^2 \right) = -(t_i - o_i) \quad (59)$$

In other words, the weight change for a link coming into an output node depends upon how different the output is from the target. (Note about the math:  $h = i$  for one term and that is the term for which the derivative is non-zero.)

- Combining (59) and (58), the delta value for an output node is given by

$$\delta_i = (t_i - o_i)\Phi'(I_i) \quad (60)$$

- Computing the delta value for a hidden node is more challenging, because it must take into account all of the ways in which a single weight can affect the difference between the network outputs and the target. If a hidden node connects to multiple output nodes, then changing a weight of a connection coming into a hidden node changes all of the output node values.

If there are  $H$  output nodes, we can use the chain rule to identify the relationships.

$$\frac{\partial E}{\partial o_i} = \sum_h \frac{\partial E}{\partial I_h} \frac{\partial I_h}{\partial o_i} = \sum_{h=1}^H \frac{\partial E}{\partial I_h} \frac{\partial}{\partial o_i} \sum_k w_{kh} o_k = \sum_h \frac{\partial E}{\partial I_h} w_{ih} = - \sum_h \delta_h w_{ih} \quad (61)$$

The way to interpret (61) is that the  $\delta$  for a weight going into the hidden node is the sum of the  $\delta$  values of all the connections leaving the node multiplied by their respective weights.

- The generalized delta rule for a hidden layer is, therefore, as follows.

$$\delta_i = \Phi'(I_i) \sum_h \delta_h w_{ih} \quad (62)$$

- The update for a weight connected to an output node is given in (63).

$$\Delta w_{ji} = \eta(t_i - o_i)\Phi'(I_i)o_j \quad (63)$$

- The update for a weight connected to a hidden node is given in (64).

$$\Delta w_{ji} = \eta \left( \Phi'(I_i) \sum_h \delta_h w_{ih} \right) o_j \quad (64)$$

Intuitively, the update for a weight  $w_{ij}$  connecting node  $i$  to hidden node  $j$  is proportional to  $o_i$  modified by the impact of any change in the weight on all of the  $K$  output nodes it affects.

One thing to note is that in both cases if  $o_i = 0$  then there will be no weight adjustment. Therefore, that suggests avoiding training examples where the inputs are zero. For example, the network learns much better when the training samples for the XOR problem avoid exact zero as an input, instead using numbers close to zero.

Likewise, trying to train a network to achieve a true 0.0 or a true 1.0 is difficult, and often leads to overtraining because it pushes the outputs far into the saturation zone of the sigmoid function. Instead, having training examples where the maximum desired output is 0.9, for example, rather than 1.0, tends to result in faster training and a more generalized network.

4. The weight up-date step is accomplished by

$$w_{ji} = w_{ji} + \Delta w_{ji} \quad (65)$$

## Backpropagation

Applying all of the training patterns to the network is called an **epoch**. There are three algorithms for an epoch of training

1. On-line:

For each training pattern (input),

- Apply the training pattern (i.e. compute  $\Delta w_{ji}$  for all edges)
  - Apply the weight changes to the network
  - Repeat
2. Stochastic: This is the same as on-line, but randomly permutes the order in which the training patterns are applied.
3. Batch:
- For each training pattern (input),
- Apply the training pattern (i.e. compute  $\Delta w_{ji}$  for all edges).
  - Store the weight change.

Apply the weight changes to the network. These weight changes may be the sum of all the weight changes computed in the above loop, or the mean, or the median.

## Key factors in ANN design

- Architecture complexity: big enough to learn the task, small enough to train and run

Since the input and output layers are generally determined by the task, the complexity of the ANN is determined by the number and structure of the hidden nodes. Some networks reduce complexity (the number of weights) by not fully connecting the layers. On a fully connected network, reducing or increasing the complexity of the network means adding or deleting hidden nodes.

An empirical method of network design is to start with a small number of hidden nodes and add nodes until overtraining is possible: when the training and test set errors begin to diverge after sufficient training. Alternatively, if the initial guess at the number of hidden nodes enables overtraining, then reduce the number of hidden nodes until the network's performance begins to degrade.

- Creating adequate training and testing sets: both quantity and coverage of likely inputs

ANNs learn based on their training data. They are good at learning a function in areas where there is lots of training data, and they can generalize to nearby parts of the input space. Inputs that are very different from any of their training examples will produce an output, but the output is not likely to be meaningful. To guard against outputs that are meaningless, the network either needs training examples from all parts of the the input space or it needs to produce some kind of confidence measure that indicates whether it knows how to process the input.

Note: Stephanie found the Wikipedia page on over fitting to be helpful.

- Building in expert knowledge

If you know something about the structure of the problem or relationships in the input, then you can sometimes build that knowledge into the structure of the network. A hidden node in a network works similarly to a compression method like PCA. The training process forces it to learn the set of connection weights that minimize the total error of the network. In doing so, it is compressing a set of values—the inputs—to a single value.

If you have prior knowledge that a set of inputs are strongly related, it is worth considering a network architecture where only those inputs connect to one or more hidden nodes in a pipeline separate from other inputs. The output of the specialized hidden node connects with the rest of the network at a higher layer. This architecture forces the network to learn the relationships between the limited set of inputs before learning the group's relationship to other inputs. If the prior knowledge is correct, then the network may train faster and work better than a more generic network trained on the same data.

Given an infinite amount of data, a generic architecture can learn as well or better than a specialized architecture. However, given a specific data set, researchers have shown that a modular, specialized network will generally perform better than a generic architecture.

- Confidence measures

Careful design of the network outputs enables the network to provide an indication as to whether its output is meaningful. For example, when dealing with a two class problem, it is possible to design the network

with a single output node, using a low value to indicate one class and a high value to indicate the other. A single output, however, gives no indication of the confidence the network has in the value.

An alternative is to design the network with two outputs, one for each class, and train it to make only one of the outputs high. In that situation, two high outputs or two low outputs indicates that the network is not well trained for that input pattern. While it is also possible for the network to give a meaningless 1/0 or 0/1 output, having the two outputs makes that less likely.

### **Notes from Stephanie**

The above notes were written by Bruce Maxwell in 2012 and modified by Stephanie Taylor in 2013.

A link I found helpful:

[http://galaxy.agh.edu.pl/~vlasi/AI/backp\\_t\\_en/backprop.html](http://galaxy.agh.edu.pl/~vlasi/AI/backp_t_en/backprop.html)

Also, you can read the Wikipedia entries on multilayer perceptrons and back-propagation.

Acknowledgement: Some of the material in these notes is taken from Bruce Maxwell's spring 2012 CS251 notes.