Teaching Calendar

- 8/22/2024. Introduction to class. See more details
- 8/26/2024. Didn't It Rain? Exploring data Also here's a fix for WSL and Python
- 8/28/2024. Present findings from weather data. Class Notes. Discuss Bayes' Theorem.
  - HW: Finish Bayes' Theorem notebook. (html version). Please read this intro to jupyter and python lists if you need some pointers.
- 9/3/2024 (Tuesday) Turn in Bayes homework. Quick library orientation. New notes on Linear Regression – new notes provided.
  - HW: Complete the linear regression notebook, (html version) for classwork/homework. As an application, do a linear regression on the London weather dataset (this part is *not* HW yet but will be). For next class: pick some new ideas from your *Money List* to discuss.
- 9/5/2024 (Thursday) Turn in Linear Regression homework. Go over Bayes and Regression notebooks in class. Discuss goodness of fit measures for categorical and quantitative data. Notes on Pearson's Correlation Coefficient.
  - Classwork: perform a linear regression on the London weather dataset.
  - Then find your own dataset somewhere and perform a linear regression on it. In both cases use the LR algorithm you coded already; do not use built-in linear regression tools, please. Make your data analysis into a nice, brief write-up (emphasis on brief) and turn it in before class ends (as a jupyter notebook).
  - Classwork: Discuss some recent *money list* ideas
  - HW: Finish classwork, any other old notebooks that aren't done yet

Homework 01

Goals for the day

- Set up python dev environment on school or personal laptop
- Raspberry Pi Project
- Money List

Set up dev environment

You need a decent amount of software to do real machine learning and your life will be much easier if we all have the same dev environment, to a certain degree. Here are steps we can take to try to get on the same page (these instructions are not perfect and if you get stuck, try to figure it out or we'll fix it in class)

1. Install a real text editor (VSCode, Atom, Sublime, `vim`, `emacs`, ...). If you don't know or care, do VSCode.

2. On a Windows machine, install `WSL` (Windows Subsystem for Linux) following these instructions. Be sure to select Ubuntu 22 for your distribution (unless you are sure you will never need to ask me for help).
3. Inside linux (WSL or Mac/Linux bash shell), install `asdf` (a version manager for executables) according to these instructions.
4. Set up python
    1. Restart your shell (in WSL or Mac/Linux bash shell)
    2. `asdf plugin add python`
    3. `asdf install python 3.11.9`
    4. `asdf global python 3.11.9` or `asdf local python 3.11.9`
    5. Make a directory for your ML projects `mkdir ~/ml` then open the directory `cd ~/ml` and set up a virtual environment (next)
5. Set up a virtual environment (a directory with specific version of python and libraries to use in your projects.)
    1. In your ml directory (`cd ~/ml`)
    2. Ensure you're running python 3.11.9: `python --version` (if not redo step 4.4 above)
    3. Create a virtual environment `python -m venv env` (This creates a directory named `env` with a local install of python)
    4. Activate the environment `source env/bin/activate`
    5. Copy this file to your folder `~/ml` as `requirements.txt`.
    6. Install the libraries `pip install -r requirements.txt`
    7. Everything should install with no errors (warnings are OK.)
    8. See if jupyter is working: `jupyter-lab` (should open a browser)

Action Item: (As much as possible...) Set up a functioning python 3.11.9 dev environment with Jupyter and a text editor. Use the provided `requirements.txt` file to create a virtual environment with the packages needed for class.

Optional (last resort)

If the WSL route is too complicated for now or just not working, Anaconda will get you up and running. The reason this is not recommended is that Anaconda installs a LOT of software and does a LOT of setup behind the scenes. And when you want to change something, or bypass Anaconda for a reason, it can be tricky to disable it. That said, it is very popular and good at what it does.

Install Anaconda for Windows

Raspberry Pi Project

I have about 25 Raspberry Pi 3s and would like you to do *something* with one. Go home, read about them, find a fun project or something to install or make and come tell me what it is. If I approve, then I'll give you a pi for a few days to make it work. In addition to the Pi we have sensors and peripherals and things so if you want to do something fancy we might be able to find the stuff for it. Just note: this is not a *big* project. Just a quick thing to get something up and

running.

You will need a monitor, HDMI cable, keyboard and wired mouse to complete the initial Pi setup. After that it can run "headless" on your home network. If you don't have these things at home, we can probably get you to do the initial setup here at school.

Action Item: come to class with some ideas for projects. Be ready to discuss with friends and/or the teacher.

The Money List

Your money list is a list of ideas that will make you money! You'll add to it all year. It starts off simple: write down things that annoy you and think of ways to fix them. The annoying things can be literally anything (that's the point of brainstorming) and the "ways to fix them" can be outlandish. But, the goal is to occasionally stumble on an idea that is generally useful to lots of people and whose solution is something you can work on. If your idea is good enough and your solution works, then sell it. Make money!

Action Item: Start the list somewhere semi-permanent. In a notebook, on your computer, Google Drive, or your phone. (the best might be a cloud file shared to multiple devices). Try to find 3 ideas by next class. Don't be too picky

Exploring data

The file weather-daylight.csv contains observational weather data for Leesburg, VA. We want to analyze the hypothesis "the fall of 2023 had more cloudy and rainy weekends than normal." As a class, let's look at the data and talk about our ideas for processing it. Open this file in Microsoft Excel, or something like it.

What questions do you have? What pre-processing is relevant? What types of calculations would support or refute the claim? Do you know how to make those calculations in Python or another language or tool?

Look at It! LOOK at IT!

(Bonus points if you know the Seinfeld reference). The first thing to do is just look at the data set. What do you see? Here are some questions you should ask? - How many rows, how many columns? - Is the data rectangular (are all rows the same length?) - What types of data? (Numerical, categorical?) - What domain of data (numerical: min and max, precision, mean, variance; categorical: number of categories) - Any missing data? - Is the file clean (read/write errors? paragraphs of text before or after? anything else weird?) - Find meaning: What do the columns are rows mean? Are there headers? Are they defined? - What is the source? Is this data reliable? - **Weather columns**

Analyse it

- What question are you asking?
- What does the data say about the question?
- Repeat the last 2 steps as needed!

Jupyter Notebook

- Follow this link to a jupyter notebook
- I believe you can save and open notebooks from this interface, as long as you are using the same Chrome profile and history (it uses local storage to save state).

Homework

- Open the jupyter notebook above from class on mybinder.org (or you can run it locally as `jupyter-lab` in your `ml` folder on WSL. Make sure to copy the notebook to your `ml` folder first – download it).
- Devise a different way to analyse the question about the weather in Leesburg and try to analyse it using pandas. What conclusion can you draw?
- Make some interesting plots from this dataset. You will need to read up on plotting in dataframes using pandas and possibly some things about pyplot.
- Consider the London Weather dataset. Investigate the question "Has the weather in London gotten worse in the last 50 years?" Analyse the data and make a claim that you can support. Source for dataset: https://www.kaggle.com/datasets/emmanuelfwerr/london-weather-data, which retrieved the data from https://www.ecad.eu/dailydata/index.php.
- Be prepared to discuss your findings and present to the class if asked to! (Your research doesn't need to be profound but I do need to see you're learning how to use `pandas`).
- To be safe, you should download any notebooks you create on `mybinder.org` because I don't know how reliable its storage system is.
- WSL Problems: Fix posted!
- **In general** I'm always happy to answer homework questions. *Remind* is the best way to reach me in the evening so don't shy away from asking. If I can't help, I'll say so. Otherwise I'll try my best!

Fixing WSL

If you tried my first instructions and ran into some kind of python packaging error (looks like mysql is a culprit), here's a patch

First, you should switch to your `ml` folder in WSL and delete the old environment.

```
$ cd ml # or whatever
$ deactivate # may not do anything if you didn't activate the env
```

```
$ rm -rf env # this deletes the old environment
$ asdf local python 3.11.9 # make sure you're on the right python
$ python -m venv env # make the virtual environment
$ source env/bin/activate # enable the new environment
$ pip install jupyterlab numpy scikit-learn matplotlib pandas
$ jupyter-lab # make sure it works
```

When I tried this on one student machine it worked. Jupyterlab runs in a browser and you may have to click on a link that appears in your console to get it to run. (the link will contain `localhost` or `127.0.0.1`)

The problem seems to be an incompatibility with the packages I defined in `requirements.txt`. The Python Package Manager (`pip`) tries to resolve dependencies in a consistent manner but does not always succed (i.e. it usually fails on big projects.) I was hopeful the environment I carefully curated would work on windows AET machines but it doesn't. This short `pip install` command installs what we need right now and will be fine until I can properly debug with my own machine.

August 28

Today in class - 5 minutes for computer bugfixing - Present findings for weather data - Upload HW to server - wsl: `cd` into the `ml` directory where your jupyter notebook is located (download it if it's online) - wifi: AET-3142 - use wsl `ftp` and `ftp username@ubuntu` - your username is `llllllf`, (first 7 letters of last name, then first initial, according to phoenix) - passwd is your student ID - to save the file: `put filename` (tab completion should probably work here) - to check it's there: `ls` - to disconnect `ctrl-D` or (I think) `exit` - Discuss Bayes' Theorem, including Jupyter notebook and python concepts * lists, arrays and comprehensions in python * for loops in python * add, delete cells in Jupyter * Markdown cells and syntax

Homework is complete the Bayes' Theorem Jupyter notebook.

Bayes' Theorem

Bayes' Theorem gives us a way to *invert* conditional probabilities. The formula comes from the definition of conditional probability

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

this implies the following

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

Solving for $P(A|B)$ we get

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

Though this is the final form, in practice you will need to compute $P(B)$ using the following

$$P(B) = P(B|A)P(A) + P(B|\overline{A})P(\overline{A})$$

which says the probability of B is the sum of the probability of B given A and B given not A. (A is either true or false so these are the only two options)

Exercise 1: Plot a Venn Diagram

Using matplotlib, draw a simple Venn diagram representing two sets A, B with a non-null intersection.

```
## Code here. Add cells as needed
```

Exercise 2: Compute Bayes' Probabilities

We want to replicate the computation carried out in class. If a doctor performs a test that has a given accuracy, for a disease with a given incidence rate, determine the probability that a randomly selected person with a positive test result has the disease. You are given *accuracy* and *incidence* as input, both in the range $(0, 1]$

```
def get_bayes_probability(acc, inc):
    ## code here
    return bayes
```

Check some results below. The first one comes from class

```
get_bayes_probability(0.97,0.001)
```

```
get_bayes_probability(0.97,0.01)
```

```
get_bayes_probability(0.97,0.1)
```

```
get_bayes_probability(0.99,0.001)
```

```
get_bayes_probability(0.50,0.001)
```

Exercise 3: Plot

You will create two plots in the section. For a fixed incidence rate, plot the bayes probability as the accuracy of the test ranges from 0 to 100%.

Then, for a fixed accuracy, plot the bayes probability as the incidence rate increases.

Note, to avoid 1/0 errors you'll probably want to *not* go all the way to 0 or 1.

State a conclusion about the results. What's the correlation? What do you observe? What do you think about accuracy measures for tests now?

Hint: create two arrays `X,Y` (python lists) of the same length containing the X values in one array and the Y values in another. List comprehensions are the best way to do this in python, though a for loop is fine too (append to an initially empty list)

then use `plt.plot(X,Y)`

```
from matplotlib import pyplot as plt

# code here. add as needed
```

Exercise 4: Beautify plots

Now go back and beautify your plots. Add a title and a legend. Some axis labels. Maybe read about matplotlib styles and change up the colors. Try a different type of plot. Just experiment some. Results below.

Quick Jupyter Intro and Python Loops

Jupyter notebook is arranged into cells. Cells can contain Input (computations/code), Output and Markdown (text). A cell can be *selected* or *active*. To *select* a cell, click to the left of it and it should highlight with a colored border. To activate a cell, click inside of it so the cursor is visible. To run a code cell, or render a text cell, type `shift-return`.

When a cell is selected you can do cell operations by typing single letters such as

- `A` add a cell above this one
- `B` add a cell below this one
- `X` cut the cell to the clipboard
- `C` copy the cell to the clipboard
- `V` paste the cell on the clipboard
- `D` delete the cell
- `M` convert a cell to markdown style
- `Y` convert a markdown cell back to Input style

You can also drag a selected cell around the notebook with the mouse. You should also get familiar with the menu bar and toolbars. There are several useful operations hidden there. For example, *rerun all cells* is handy as is *Clear Outputs of all cells*

Python Loops

I'll show some basic examples of how to do things in python. You may want to run these in Jupyter to verify.

- Print the numbers 0-99 on separate lines `python for i in range(100): print(i)`
- Print the numbers 10-99 on separate lines `python for i in range(10,100): print(i)`
- Print the *even* numbers 10-19 on separate lines `python for i in range(10,20,2): print(i)`
- Print all characters in "hello computer", all on one line `python for c in "the word": print(c, end="")`
- Compute the largest Fibonacci number less than 1000 (note the use of parallel assignment) `python a,b = 1,0 while a<1000: a,b = a+b,a`

Python Lists

Python's regular list data type is not a true array, such as you would find in C or Java. An array is, properly, fixed length and single type. Python uses *lists* which are dynamic (auto-resizing) and can contain *any* type. For example

```
l1 = [1,2,3]
l2 = ["bird",2,-10.3,"cow"]
l3 = [l1,l2,3]
print(l1)
print(l2[2])
print(l3[1][1])
```

```
[1, 2, 3]
-10.3
2
```

You see lists can even contain lists. Here are some familiar tropes in Python

```
l = [0]*10 # a list of 10 zeros
for i in range(len(l)):
    l[i] = i  # replace the zeros with the index
for i in range(10,20):
    l.append(i) # add more elements to l
for i in range(20,30):
    l = l + [i] # another way to add
print(l)
```

This prints the integers from 0 to 29 inclusive.

Lists are very flexible and there are many operations that are easy to do such as sort, find, replace, merge, delete. If you want to perform a list operation, look it up in online and see if it is already a built-in operation

Comprehensions

Comprehensions are beautiful and lovely ways to build lists quickly. I'll just give a couple examples here

This code

```
l = []
for i in range(10):
    l.append(i**2)
```

can be replaced with this comprehension

```
l = [i**2 for i in range(10)]
```

It is quite lovely, isn't it. It reads like a math set definition

$$L = \{i^2 \mid 0 \leq i < 10\}$$

And make great tools for plotting, say

```
from math import sin
from matplotlib import pyplot as plt
X = [i/100 for i in range(628)]
Y = [sin(i/100) for i in range(628)]
plt.plot(X,Y);
```

will plot one period of a sine curve.

In practice, numerical algorithms will use `numpy arrays` instead of python lists because they are much much faster and work like traditional C arrays. But lists still come in extremely handy when you need to collect data and speed is not a priority.

Linear Regression

Given n points $(x_1, y_1) \ldots (x_n, y_n)$ and an assumed relation $y = f(x) + \epsilon, \epsilon \sim N(\mu, \sigma)$ we want to find a model $\tilde{y}_i = ax_i + b$ such that the residual squared error

$$\mathrm{RSS}(a, b) = \sum_{i=1}^{n} (\tilde{y}_i - y_i)^2$$

is minimized.

RSS is a function of the line parameters a and b. To minimize it we set both partial derivatives to zero. (This could technically find a maximum – but it's reasonably clear this function has no maximum value because the error can always be increased.)

Take partial derivatives

$$\begin{aligned}\frac{\partial RSS}{\partial a} &= 2\sum\left(\tilde{y}_c - y_i\right)\frac{\partial}{\partial a}\left(\tilde{y}_i - y_i\right)\\ &= 2\sum\left(\tilde{y}_i - y_i\right)\left(x_i\right)\\ \frac{\partial RSS}{\partial b} &= 2\sum\left(\tilde{y}_i - y_i\right)\frac{\partial}{\partial b}\left(\tilde{y}_i - y_i\right)\\ &= 2\sum\left(\tilde{y}_i - y_i\right)\left(1\right)\end{aligned}$$

Since

$$\frac{\partial}{\partial a}\tilde{y}_i = \frac{\partial}{\partial a}\left(ax_i + b\right) = x_i$$

$$\frac{\partial}{\partial b}\tilde{y}_c = \frac{\partial}{\partial b}\left(ax_c + b\right) = 1$$

And solve

$$\begin{cases}\dfrac{\partial RSS}{\partial a} = 0\\[2ex] \dfrac{\partial RSS}{\partial b} = 0\end{cases} \Rightarrow \begin{cases}\sum(\tilde{y}_i - y_i)\,x_i = 0\\[2ex] \sum\left(\tilde{y}_i - y_i\right) = 0\end{cases}$$

Since $\tilde{y}_i = ax_i + b$

$$\sum\left(ax_i + b - y_i\right)x_i = 0 \Rightarrow a\sum x_i^2 + b\sum x_i = \sum y_i x_j$$

and

$$\sum\left(ax_i + b - y_i\right) = 0 \Rightarrow a\sum x_i + b\sum 1 = \sum y_i$$

by Cramer's rule

$$\begin{aligned}a &= \begin{vmatrix}\sum x_i y_i & \sum x_i\\ \sum y_i & n\end{vmatrix} \Big/ \begin{vmatrix}\sum x_i^2 & \sum x_i\\ \sum x_i & n\end{vmatrix}\\ b &= \begin{vmatrix}\sum x_i^2 & \sum x_i y_i\\ \sum x_i & \sum y_i\end{vmatrix} \Big/ \begin{vmatrix}\sum x_i^2 & \sum x_i\\ \sum x_i & n\end{vmatrix}\end{aligned}$$

since $\sum_{i=1}^{n} 1 = n$

Taking determinants,

$$a = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$$

$$b = \frac{\sum y_i \sum x_i^2 - \sum x_i \sum x_i y_i}{n\sum x_i^2 - (\sum x_i)^2}$$

Interpretation as a ratio of variances

Students of statistics may appreciate the following manipulations

Definition of covariance

$$E(xy) - E(x)E(y) = \mathrm{Cov}(x, y)$$

Definition of variance

$$\mathrm{Var}(x) = E\left[(x-\mu)^2\right]$$

Lemma

$$\begin{aligned}\mathrm{Var}(x) &= E\left[(x-\mu)^2\right] \\ &= E\left(x^2\right) - 2\mu E[x] + E[\mu]^2 \\ &= E\left[x^2\right] - 2E[x]^2 + \mu^2 \\ &= E\left[x^2\right] - E[x]^2\end{aligned}$$

Manipulating the denominator of the equation for a on the previous page,

$$\begin{aligned}n\sum x_i^2 - \left(\sum x_i\right)^2 &= n^2\left(\frac{1}{n}\sum x_i^2 - \left(\frac{\sum x_i}{n}\right)^2\right) \\ &= n^2\left(E\left[x^2\right] - E[x]^2\right) \\ &= n^2\,\mathrm{Var}(x)\end{aligned}$$

And the numerator

$$\begin{aligned}n\sum x_i y_i - \sum x_i \sum y_i & \\ &= n^2\left(\frac{1}{n}\sum x_i y_i - \frac{1}{n}\sum x_i \cdot \frac{1}{n}\sum y_i\right) \\ &= n^2\left(E\left[xy\right] - E[x]E[y]\right) \\ &= n^2\left(E[xy] - \mu_y\mu_y\right) \\ &= n^2\,\mathrm{Cov}(x, y)\end{aligned}$$

so

$$a = \frac{E[xy] - \mu_x \mu_y}{E[x] - \mu_x^2} = \frac{\text{Cov}(x, y)}{\text{Vav}(x)}$$

Linear Least Squares

In class we derived the formula for linear least squares of one variable. In this notebook you will learn a bit of the numerical library numpy, use numpy to compute linear regression, and then compute it yourself using formulas from class

Begin by running the cell below. Then go back and carefully read through all the code. There is a lot of new stuff here. Note how to create numpy arrays/matrices and how to compute a linear least squares regression.

```
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Prepare your data
# x: Independent variable (input)
# y: Dependent variable (output)
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

# Step 2: Perform linear regression using the least squares method

# Add a column of ones to the input data for the intercept (bias term)
X = np.vstack([x, np.ones(len(x))]).T

# Calculate the slope (m) and intercept (b)
a, b = np.linalg.lstsq(X, y, rcond=None)[0]

print(f"Slope (a): {a:.4f}")
print(f"Intercept (b): {b:.4f}")

# Step 3: Predict y values using the regression line
y_pred = a * x + b

# Optional: Plot the data and the regression line
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label='Regression line')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

An aside about numpy matrices

What happened to x? Here's the original x, which is an array

```
x
```

We add a row of 1s after it and take the transpose to get the input matrix X

```
X
```

Breaking this down into pieces, first let's make a python list that contains x and an array of ones

```
[x,np.ones(len(x))]
```

Now let's use numpy to make a vertical stack. The first element in the list becomes the first row

```
np.vstack([x, np.ones(len(x))])
```

And now take the transpose

```
np.vstack([x, np.ones(len(x))]).T
```

Practice with matrices

Make a numpy matrix that is a row of 5 zeros followed by a row of 5 ones, then 5 zeros, then 5 ones again. Use built in functions and `vstack` (don't just type a bunch of 0 and 1 – can you guess the name of a function that makes an array of zeros?)

Now make a similar matrix that is a row of all 1s followed by all 2s in the second row, then 3s then 4s. Again use built in function `np.ones`. Name this matrix `M`. Hint: $[2,2,2,2,2] = 2 \cdot [1,1,1,1,1]$

compute M times M transpose and M transpose times M (MM^T and M^TM). In `numpy` AB can be computed with `A @ B` for matrices `A` and `B`

A matrix M is *symmetric* if $M = M^T$. This also implies $M_{ij} = M_{ji}$ for all indices (i, j). Write a python function `is_symmetric(M)` which returns `true` if and only if M is symmetric

Test your function. Make a 5 by 5 random integer matrix (see `np.random.randint`) called M. It is a fact that MM^T is always symmetric. Check that your function return `true` for MM^T and `false` for M. Repeat this trial 100 times and verify all 100 are correct.

Linear Least Squares Regression

You can create a vector of normally distributed samples with mean μ and standard deviation σ by using the numpy function `np.random.normal(mu, sigma, n)`. Try creating a vector with 10 random samples, with a mean of 100 and a standard deviation of 5.

Now create some data for linear regression. Make a vector x of ints over the range $[0, 9]$ and let y be a linear function of x, $y = 3x + 2 + \epsilon(x)$ where $\epsilon(x)$ is a random Gaussian noise function $\epsilon(x) \sim N(0, 1)$. Make a scatter plot of y vs. x and label it

Compute the correct linear regression coefficients using numpy as above. Check they are resonable.

Now compute the regression coefficients using the formulas from class. Begin by defining some very helpful variables: `Sx, Sy` will be $\sum_i x_i$ and $\sum_i y_i$ respectively. Next `Sxx` and `Syy` are the sum of squares: $\sum_i {x_i}^2$ and $\sum_i {y_i}^2$. Finally the inner product `Sxy` $= \sum_i x_i y_i$. The quickest way to do this involves using comprehensions and the `sum` function, but you can use loops for now if you need to.

```
# Print your results
Sx, Sy, Sxx, Syy, Sxy
```

Finally determine a, b as in class. Display the absolute errors between your calculations and the ones numpy returned. (They should be close to machine precision, which is 10^{-15} give or take.

Least squares function

Did you know python can return two values? Here's an example.

```
def two_numbers():
    a = 1
    b = 10
    return a,b

A, B = two_numbers()
print(A,B)
```

Write a function `linear_least_squares(x,y)` which takes input vectors x,y and returns a,b as above. (

```
def linear_least_squares(x,y):
```

Application

Now, using $a = 5, b = -15$, run linear least squares 100 times on 100 vector pairs (x, y), where each of the 100 x are the same but the $y = ax + b + \epsilon$ each have different amounts of Gaussian noise. Plot the resulting best fit lines all on the same graph. - Use `np.arange` to make your input vector x cover the domain $[-5, 5]$ with a step-size of 0.01 - Create arrays to store all the computed a and b values (you'll use this later) - If you call `plt.plot()` in a loop, it will keep adding to the same plot - Give your plot a title!

Determine the average of the as and bs returned above. Compare these to the true a, b. Explain your result.(There is an `np.mean` function)

Make two histogram plots of the calculated a and b values `plt.hist` works nicely and adding a semicolon suppressed the nasty text output (you'll see)

Correlation Coefficient

We assume, as usual, a ground truth model $y = f(x) + \epsilon$ where f is usually unknown, a (possibly random) sample of points $(x_1, y_1), \cdots, (x_n, y_n)$ and a linear model $\tilde{y} = ax + b$. In this setting we usually need to know *how good* the linear model is – how well does if capture the ground truth $f(x)$?

One obvious measure is the sum of squared errors, which we minimized last class to derive the linear regression equations.

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

While this literally captures the error in the model on each point, it is hard to interpret, it scales with the number of points, and is in different units from the given data. We can normalize it to the mean sum of squared errors:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\tilde{y}_i - y_i)^2$$

which at least doesn't scale with the number of points but is in different units. Thus by taking a radical

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - y_i)^2}$$

we get the root mean sum of squared errors. This at least scales with the magnitude of the y values, so you can interpret it somewhat. It is also similar to a standard deviation, which is familiar to many people. (Note some texts would divide by $n-2$ instead of n to create a truly unbiased estimator for the standard deviation, but this simpler version aggrees with other data science presentations, including kaggle.)

Pearson's Correlation Coefficient

While variants of SSE have their place, one cannot escape the use of r, the Pearson's correlation coefficient. Students learn in algebra classes that a linear regression coefficient $r = 1$ is a perfect positive correlation and $r = -1$ is a perfect negative correlation and $r = 0.5$ is a weak correlation, for example. We will take a more precise approach.

One formular for r is

$$r^2 = \frac{SS_{reg}}{SS_{tot}} = \frac{\sum_i(\tilde{y}_i - \overline{y})^2}{\sum_i(y_i - \overline{y})^2}$$

Let's unpack this. SS_{reg} is the sum of squared-error due to regression and SS_{tot} is the sum of squared-error total (due to the original data). Here $\overline{y} = \frac{1}{n}\sum_i y_i$ is the mean of the observed y_i values. SS_{tot}, then, is the variance of the observed y_i values – it is the sum of the squared deviations of the observations from their mean.

SS_{reg}, on the other hand, is the variance of the predicted $\tilde{y}_i$ values, relative to the same observed mean.

The ratio of the two is the ratio of the "explained variance" to the "total variance." There is always variance in the original dataset. If our linear model very closely fits the data, then it will explain most of that original variance. That would correspond to a high r^2 value. On the other hand a low r^2 indicates that there is variance in the data that is not capture by the linear model. Something else is happening to create this data shape.

It can be helpful to think of r^2 as the percent of "explained variance". You'll notice this formula is for r^2, not r. Obviously both are < 1 but they are not identical. We may see more details of the various ways to interpret r vs r^2 but honestly for most cases this explanation is quite good enough and better than what most people understand!

Examples of correlation

Figure 1: Examples of correlation

r=0

You may have been taught that $r = 0$ implies no correlation between x and y pairs. This is often indicated in math books with an amorphous cloud of points, wandering lonely across the page, enigmatic and unknowable. Actually a number of highly correlated datasets can claim to possess $r = 0$ values as this helpful chart shows[1]

To be correct, $r = 0$ implies no **linear** correlation between x and y. If it so happens that every predicted $\tilde{y}_i$ value is identical to the mean $\overline{y}$, then $r^2 = 0$. Datasets with perfect vertical symmetry can have this property.

[1] By DenisBoigelot, original uploader was Imagecreator - Own work, original uploader was Imagecreator, CC0, https://commons.wikimedia.org/w/index.php?curid=15165296