CIS532:
Estimating Probability Distributions

Live Session 2

Ernest Green

Today's Live Session

Reminder: Loops vs. Vectorization and NumPy

Build a Baby Name Classification System

Additional Resources

Lecture 7 "Estimating Probabilities from Data: Maximum Likelihood Estimation" -Cornell CS4780 SP17

https://www.youtube.com/watch?v=RIawrYLVdIw

Machine Learning Lecture 8 "Estimating Probabilities from Data: Naive Bayes" -Cornell CS4780 SP17

https://www.youtube.com/watch?v=pDHEX2usCS0

Machine Learning Lecture 9 "Naive Bayes continued" -Cornell CS4780 SP17

https://www.youtube.com/watch?v=VDK0nkjFh5U

Machine Learning Lecture 10 "Naive Bayes continued" -Cornell CS4780 SP17

https://www.youtube.com/watch?v=rqB0XWoMreU

Machine Learning Lecture 11 "Logistic Regression" -Cornell CS4780 SP17

https://www.youtube.com/watch?v=GnkDzIOxfzI

Recommended Reading

The Likelihood of Tails vs. “Not Heads”

https://jasondeden.medium.com/heads-or-tails-or-23803e913c1

Log Math Basics

https://jasondeden.medium.com/taking-the-log-of-my-own-i-d0981be4ce51

Indexing and Slicing

https://jasondeden.medium.com/array-indexing-slicing-vs-loops-753484854bc8

NumPy and Vectorization vs. Loops

Python is an interpreted language, not compiled

Loop iterations are processed sequentially

NumPy has built-in array functions that first compile the instructions and run them in a lower-level language

A small performance hit for the first iteration due to compile time, but afterwards operations can be implemented in parallel

But what if you need to perform the same evaluation or operation across a large number of array values and no built-in NumPy function exists?

```
In [103]: labels = np.random.randint(0,2,30)

In [104]: labels
Out[104]:
array([0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0])
```

Problem - we want to recode all 0's as -1's

We could write a loop to iterate over the array and check if the number == 0 and set the value to -1

```
In [103]: labels = np.random.randint(0,2,30)

In [104]: labels
Out[104]:
array([0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0])
```

Problem - we want to recode all 0's as -1's

We could write a loop to iterate over the array and check if the number == 0 and set the value to 1

Or we can start by creating an index based on a simple evaluation (>, <, ==, etc.)

```
In [105]: labels == 0
Out[105]:
array([ True,  True, False,  True, False, False,  True,  True,  True,
        True,  True, False,  True, False, False,  True, False, False,
       False,  True,  True, False, False, False, False, False,  True,
       False, False,  True])
```

```
In [105]: labels == 0
Out[105]:
array([ True,  True, False,  True, False, False,  True,  True,  True,
        True,  True, False,  True, False, False,  True, False, False,
       False,  True,  True, False, False, False, False, False,  True,
       False, False,  True])
```

This is an index of all values that meet the criteria specified

```
In [105]: labels == 0
Out[105]:
array([ True,  True, False,  True, False, False,  True,  True,  True,
        True,  True, False,  True, False, False,  True, False, False,
       False,  True,  True, False, False, False, False, False,  True,
       False, False,  True])
```

This is an index of all values that meet the criteria specified

...which can be used to perform replacement operations in parallel

```
In [110]: labels[labels == 0] = -1

In [111]: labels
Out[111]:
array([-1, -1,  1, -1,  1,  1, -1, -1, -1, -1, -1,  1, -1,  1,  1, -1,  1,
        1,  1, -1, -1,  1,  1,  1,  1,  1, -1,  1,  1, -1])
```

```
In [105]: labels == 0
Out[105]:
array([ True,  True, False,  True, False, False,  True,  True,  True,
        True,  True, False,  True, False, False,  True, False, False,
       False,  True,  True, False, False, False, False, False,  True,
       False, False,  True])
```

This is an index of all values that meet the criteria specified

...which can be used to perform replacement operations in parallel

```
In [110]: labels[labels == 0] = -1

In [111]: labels
Out[111]:
array([-1, -1,  1, -1,  1,  1, -1, -1, -1, -1, -1,  1, -1,  1,  1, -1,  1,
        1,  1, -1, -1,  1,  1,  1,  1,  1, -1,  1,  1, -1])
```

...the key value being it took one iteration, one line of code, vs. 30 repeated iterations for the loop approach, to get to the same result

How Much Faster Is It Really?

https://jasondeden.medium.com/eat-my-dust-loops-33e5279a01de

Summary:

- Operation on an array that takes > 20 seconds using loops and around 17 seconds using a list comprehension takes under 600 milliseconds using vectorization and around 300 milliseconds using NumPy

- Performance gain of up to 70x for using optimal approach

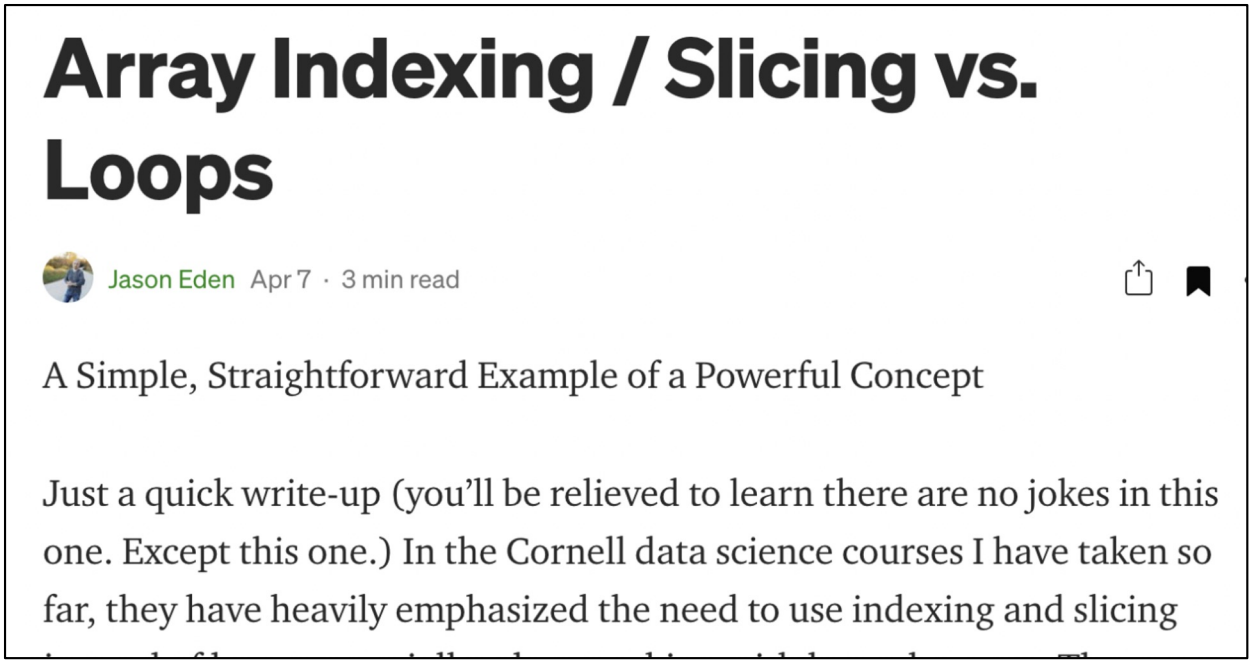
Classification Problem

In the Class Prior exercise, we were given 1's and 0's and asked to determine class priors, which could simply be calculated as np.mean(array). But what if our labels were 1 and -1? Or boy and girl?

In Python, True == 1 and False == 0. Can you use this to create code that works for any binomial distribution and computes class priors?

Use one array to index another one…

https://jasondeden.medium.com/array-indexing-slicing-vs-loops-753484854bc8



Array Indexing / Slicing vs. Loops

Jason Eden Apr 7 · 3 min read

A Simple, Straightforward Example of a Powerful Concept

Just a quick write-up (you'll be relieved to learn there are no jokes in this one. Except this one.) In the Cornell data science courses I have taken so far, they have heavily emphasized the need to use indexing and slicing

Build a Baby Name Classification System

Functions Given:

- hashfeatures
- name2features
- genTrainFeatures

Functions to write (graded):

- naivebayesPY
- naivebayesPXY
- loglikelihood
- naivebayes_pred

Challenge Exercise - Creating Structure

Take the logic from the exercise and create a feature vector of the same length for every word in your data and indicates 1 for the position of every vowel.

Then create another feature vector that indicates 1 for the position of every consonant.

Question: How long do these feature vectors need to be?

Question 2: If you combined them, could you determine the length of any given word given only the string of 1’s and 0’s you generated?

hashfeatures

```
v = np.zeros(d)
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    P = hash(prefix) % d
    v[P] = 1
```

hashfeatures

```
v = np.zeros(d)
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    P = hash(prefix) % d
    v[P] = 1
```

Assume we don't understand everything this function is doing. How can we gain that understanding?

hashfeatures

```
v = np.zeros(d)
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    P = hash(prefix) % d
    v[P] = 1
```

We define the `hashfeatures` functic
section later in this assignment, you wil

```
In [3]: hash("jason")
Out[3]: 4020064578765946322

In [4]: np.zeros(5)
Out[4]: array([0., 0., 0., 0., 0.])

In [24]: v = np.zeros(5)

In [6]: "jason"[:2] + ">"
Out[6]: 'ja>'
```

hashfeatures

```
v = np.zeros(d)
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    P = hash(prefix) % d
    v[P] = 1
```

```
In [6]: "jason"[:2] + ">"

Out[6]: 'ja>'

In [7]: prefix = "jason"[:2] + ">"

In [17]: hash(prefix)

Out[17]: 3351328919580344889

In [20]: hash(prefix) % 5

Out[20]: 4
```

hashfeatures

```
v = np.zeros(d)
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    P = hash(prefix) % d
    v[P] = 1
```

```
In [25]: P = hash(prefix) % 5

In [21]: 10 % 5
Out[21]: 0

In [22]: 10 / 5
Out[22]: 2.0

In [23]: 10 % 6
Out[23]: 4

In [26]: v
Out[26]: array([0., 0., 0., 0., 0.])

In [27]: v[P] = 1

In [28]: v
Out[28]: array([0., 0., 0., 0., 1.])
```

hashfeatures

```
v = np.zeros(d)
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    P = hash(prefix) % d
    v[P] = 1
```

```
v = np.zeros(d)
print("v before feature generation = {}".format(v))
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    print("prefix at iteration {} = {}".format(m,prefix))
    P = hash(prefix) % d
    print("P at iteration {} = {}".format(m,P))
    v[P] = 1
    print("v after iteration {} = {}".format(m,v))
```

hashfeatures

```
In [34]: v = hashfeatures("Addisyn", d=128, FIX=3, debug=True)
         v = hashfeatures("Addisyn", d=4, FIX=3, debug=True)
         v = hashfeatures("Addisyn", d=128, FIX=7, debug=True)
         v = hashfeatures("Max", d=128, FIX=4, debug=True)

v before feature generation = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
prefix at iteration 1 = A>
P at iteration 1 = 75
v after iteration 1 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
Split 1/3:      (A>, <n),       1s at indices [75, 102]
prefix at iteration 2 = Ad>
P at iteration 2 = 40
v after iteration 2 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

hashfeatures

```
In [34]: v = hashfeatures("Addisyn", d=128, FIX=3, debug=True)
         v = hashfeatures("Addisyn", d=4, FIX=3, debug=True)
         v = hashfeatures("Addisyn", d=128, FIX=7, debug=True)
         v = hashfeatures("Max", d=128, FIX=4, debug=True)
```

```
v before feature generation = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
prefix at iteration 1 = A>
P at iteration 1 = 75
v after iteration 1 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
Split 1/3:      (A>, <n),       1s at indices [75, 102]
prefix at iteration 2 = Ad>
P at iteration 2 = 40
v after iteration 2 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

hashfeatures

```
In [34]: v = hashfeatures("Addisyn", d=128, FIX=3, debug=True)
         v = hashfeatures("Addisyn", d=4, FIX=3, debug=True)
         v = hashfeatures("Addisyn", d=128, FIX=7, debug=True)
         v = hashfeatures("Max", d=128, FIX=4, debug=True)
```

```
v before feature generation = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
prefix at iteration 1 = A>
P at iteration 1 = 75
v after iteration 1 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
Split 1/3:      (A>, <n),       1s at indices [75, 102]
prefix at iteration 2 = Ad>
P at iteration 2 = 40
v after iteration 2 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

hashfeatures

debug = True provides useful data

```
v = hashfeatures("Addisyn", d=128, FIX=3, debug=True)
v = hashfeatures("Addisyn", d=4, FIX=3, debug=True)
v = hashfeatures("Addisyn", d=128, FIX=7, debug=True)
v = hashfeatures("Max", d=128, FIX=4, debug=True)
```

```
Split 1/3:      (A>, <n),       1s at indices [101, 31]
Split 2/3:      (Ad>, <yn),     1s at indices [34, 121]
Split 3/3:      (Add>, <syn),   1s at indices [28, 70]
Feature vector for Addisyn:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]

Split 1/3:      (A>, <n),       1s at indices [1, 3]
Split 2/3:      (Ad>, <yn),     1s at indices [2, 1]
Split 3/3:      (Add>, <syn),   1s at indices [0, 2]
Feature vector for Addisyn:
[1 1 1 1]

Split 1/7:      (A>, <n),       1s at indices [101, 31]
Split 2/7:      (Ad>, <yn),     1s at indices [34, 121]
Split 3/7:      (Add>, <syn),   1s at indices [28, 70]
Split 4/7:      (Addi>, <isyn), 1s at indices [2, 89]
Split 5/7:      (Addis>, <disyn),       1s at indices [38, 44]
Split 6/7:      (Addisy>, <ddisyn),     1s at indices [47, 28]
Split 7/7:      (Addisyn>, <Addisyn),   1s at indices [52, 32]
Feature vector for Addisyn:
[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]

Split 1/4:      (M>, <x),       1s at indices [26, 86]
Split 2/4:      (Ma>, <ax),     1s at indices [27, 89]
```

hashfeatures Psuedocode

Initialize an array (v) of d zeros

Generate a prefix feature (m sequence of characters)

Hash to make this numeric

Take modulus d of hash to generate a remainder / index value

Set index at that location = 1

Generate a suffix feature (m sequence of characters)

(Repeat steps for prefix)

Iterate/repeat, m += 1 (or m = m+1) until reach FIX+1

If debug = True, print some info

Don’t Forget to Clean Up

```
v = np.zeros(d)
#print("v before feature generation = {}".format(v))
for m in range(1, FIX+1):
    prefix = baby[:m] + ">"
    #print("prefix at iteration {} = {}".format(m,prefix))
    P = hash(prefix) % d
    #print("P at iteration {} = {}".format(m,P))
    v[P] = 1
    #print("v after iteration {} = {}".format(m,v))
```

name2features

Load a file of names

Use hashfeatures to generate a vector for each one

How might you find out how many boy and girl names you have in your data?

Why might this matter later?

genTrainFeatures

Load girl names and run through name2features

Load boy names and run through name2features

Concatenate the two outputs

Generate labels (girls = -1, boys = 1)

Shuffle data such that labels still match original vectors

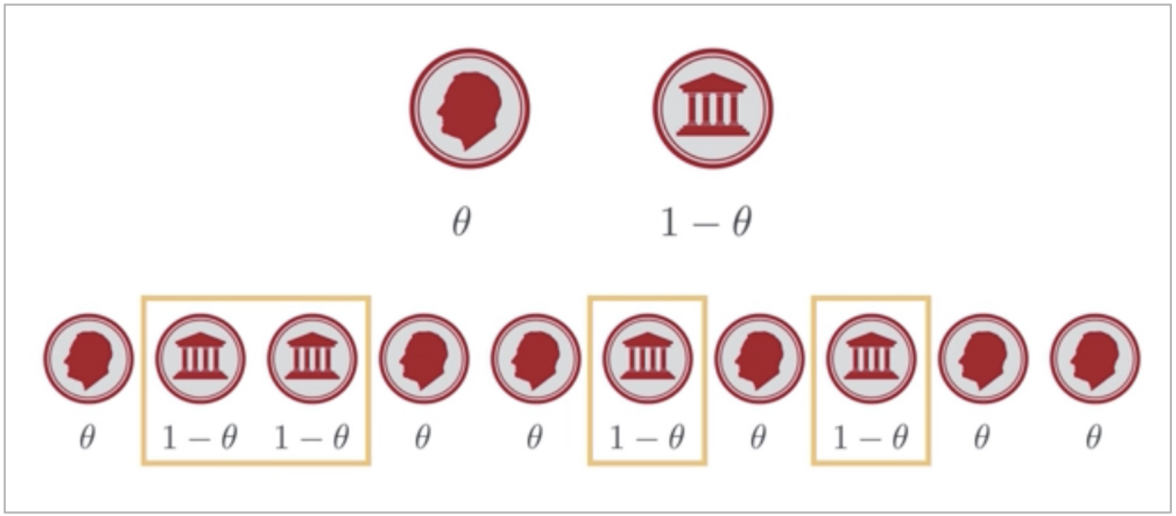
Next: Hand this data to the data scientist for evaluation! :)

naivebayesPY

Hint: Forget that you know anything about the data. If this were a series of coin flips, what do we assume the overall probability to be based only on label distribution?

P(y)

- P(girl)
- P(boy)



Use Comments as Guides for Function Format

```
"""
naivebayesPY(Y) returns [pos,neg]

Computation of P(Y)
Input:
    X : n input vectors of d dimensions (nxd)
    Y : n labels (-1 or +1) (nx1)

Output:
    pos: probability p(y=1)
    neg: probability p(y=-1)
"""
```

Use Comments as Guides for Function Format

```
"""
naivebayesPY(Y) returns [pos,neg]

Computation of P(Y)
Input:
    X : n input vectors of d dimensions (nxd)
    Y : n labels (-1 or +1) (nx1)

Output:
    pos: probability p(y=1)
    neg: probability p(y=-1)
"""
```

naivebayesPXY

P(x|y)

- P(“index 0 = 1” |y)
- P(“index 1 = 1” |y)
- P(“index 2 = 1” |y)
- …
- P(“index 54 = 1” |y)
- …

naivebayesPXY

P(x|y)

- P(“index 0 = 1” |y)
- P(“index 1 = 1” |y)
- P(“index 2 = 1” |y)
- …
- P(“index 54 = 1” |y)
- …

```
X[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
X[1]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 0., 0.])
```

Calculate Probability for Each Feature

	F1	F2	F3	F4	F5	F6	F7	...	F128
Boy1	0	0	0	0	0	0	1	...	0
Boy2	0	0	1	0	0	1	0	...	0
Boy3	1	0	1	0	0	1	0	...	1
Boy4	0	0	0	0	1	0	0	...	1
Boy5	0	0	1	1	0	0	1	...	0
...	...	...	...	...	...	...	...	...	...
Boy54	0	1	0	1	0	0	0	...	0
	?	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**

naivebayesPXY

P(x|y)

- P(“index 0 = 1” |y)
- P(“index 1 = 1” |y)
- P(“index 2 = 1” |y)
- …
- P(“index 54 = 1” |y)
- …

How large or small are these probabilities going to be if there are hundreds of names being evaluated?

naivebayesPXY

P(x|y)

- P(“index 0 = 1” |y)
- P(“index 1 = 1” |y)
- P(“index 2 = 1” |y)
- …
- P(“index 54 = 1” |y)
- …

What is this doing?

```
# add one positive and negative example to avoid division by zero ("plus-one smoothing")
n, d = X.shape
X = np.concatenate([X, np.ones((2,d)), np.zeros((2,d))])
Y = np.concatenate([Y, [-1,1,-1,1]])
```

naivebayesPXY

If it's not obvious, recreate a simple example using the grader function

Still not obvious? Generate a slightly more complex example and look at the outputs.

Example shows two observations with d = 2.

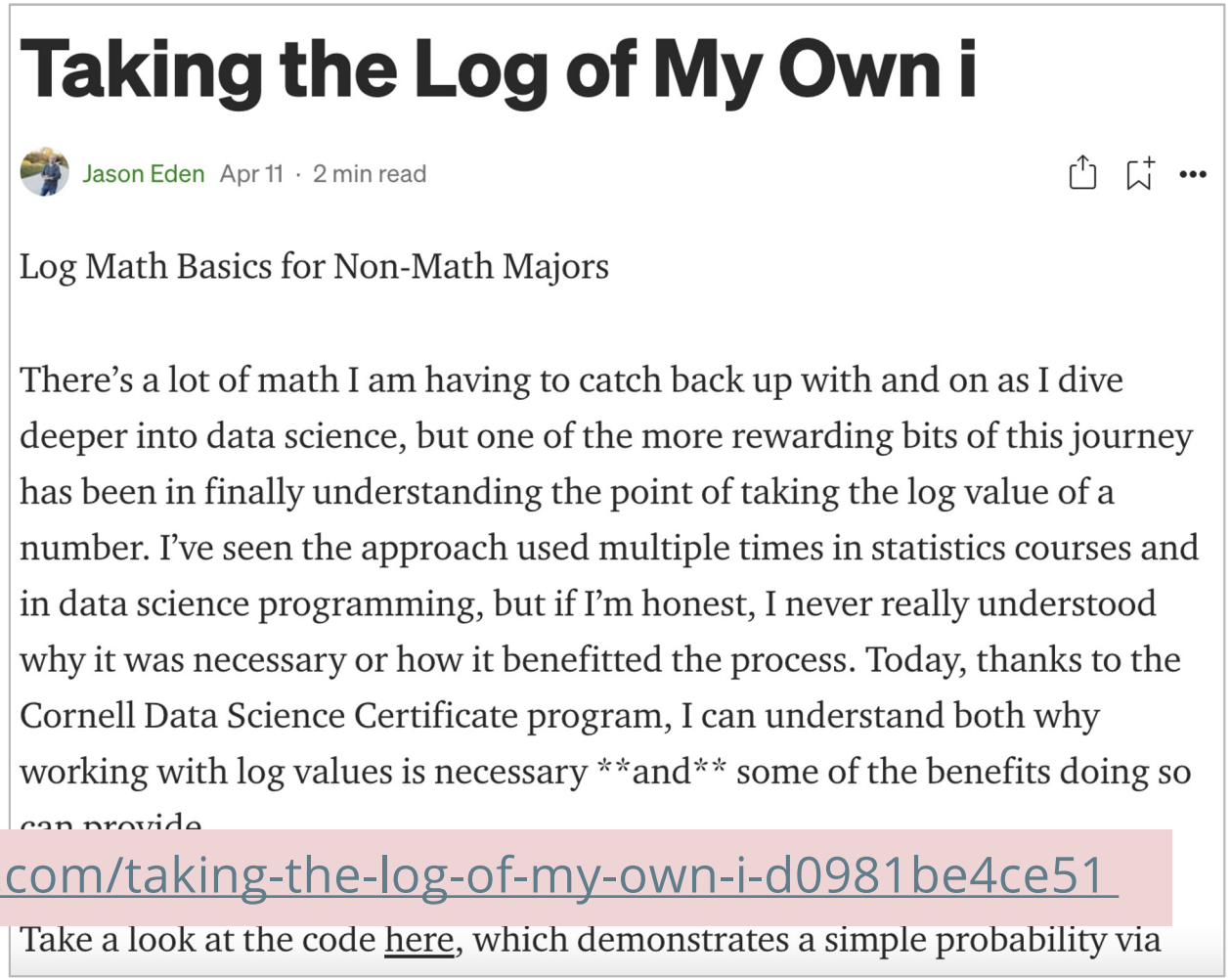
```
In [11]:  x = np.array([[0,1],[1,0]])
          y = np.array([-1,1])

In [12]:  pos0, neg0 = naivebayesPXY_grader(x,y)

In [13]:  pos0
Out[13]: array([0.66666667, 0.33333333])

In [14]:  neg0
Out[14]: array([0.33333333, 0.66666667])
```

Avoid Loops! (When possible…)

How can you sort out the data (separate boy/girl names/labels) without using loops?

loglikelihood

Why are we using log?



Taking the Log of My Own i

Jason Eden Apr 11 · 2 min read

Log Math Basics for Non-Math Majors

There's a lot of math I am having to catch back up with and on as I dive deeper into data science, but one of the more rewarding bits of this journey has been in finally understanding the point of taking the log value of a number. I've seen the approach used multiple times in statistics courses and in data science programming, but if I'm honest, I never really understood why it was necessary or how it benefitted the process. Today, thanks to the Cornell Data Science Certificate program, I can understand both why working with log values is necessary **and** some of the benefits doing so can provide

https://jasondeden.medium.com/taking-the-log-of-my-own-i-d0981be4ce51

Take a look at the code here, which demonstrates a simple probability via

loglikelihood

Also simplifies chain rule calculation:

$\log(ab) = \log a + \log b$

loglikelihood

Also simplifies chain rule calculation:

log(ab) = log a + log b

To simplify your code, we recommend calculating log likelihoods for positive points ($\mathbf{x}$ with $Y = 1$) and those for negative points separately.

However, you're only going to return a single vector. How can we code so that we are calculating girls and boys separately, but returning a single vector of likelihoods, without loops?

loglikelihood

Also simplifies chain rule calculation:

log(ab) = log a + log b

$$h \log(\theta) + t \log(1 - \theta)$$

Estimate Distribution with Maximum Likelihood Estimation

loglikelihood

Also simplifies chain rule calculation:

log(ab) = log a + log b

$$h \log(\theta) + t \log(1 - \theta)$$

theta = [posprob_i | negprob_i]

loglikelihood

Also simplifies chain rule calculation:

log(ab) = log a + log b

$$h \log(\theta) + t \log(1 - \theta)$$

h = [X_i[boys] | X_i[girls]]
t = ????

loglikelihood

Also simplifies chain rule calculation:

log(ab) = log a + log b

$$h \log(\theta) + t \log(1 - \theta)$$

h = [X_i[boys] | X_i[girls]]
t = ????

How would you represent tails in terms of heads?
In other words, if h = x, t in terms of x = ?

Heads Up!*

posprob ≠ (1 - negprob)

(≠ "does not equal")

*pun intended

Heads or Tails? Or...?

Jason Eden Apr 20 · 6 min read

The Likelihood of Tails vs. "Not Heads"

When discussing probability, a commonly used example is a coin flip. It's a 50/50 proposition, presumptively, that you will observe either heads or tails on any given flip. If you assign labels like h and t to heads and tails, you would write that out as the probability of heads = 50%, or P(h) = .5. The presumptive probability of tails would also equal 50%, or P(t) = .5 as well.

That part is pretty straightforward, in theory. In reality, very little is actually that straightforward. For example, even a simple coin flip is subject to all

https://jasondeden.medium.com/heads-or-tails-or-23803e913c1

"flipping thumb" will absolutely impact the likelihood of getting one result vs. another. The starting position of the coin (heads vs. tails up) matters

naivebayes_pred

Given novel X (X_test_i) calculate likelihood this would be a boy

(Put another way, assume X_test_i label = 1. How likely was that?)

naivebayes_pred

Given novel X (X_test_i) calculate likelihood this would be a boy

(Put another way, assume X_test_i label = 1. How likely was that?)

Repeat, for girl (assume X_test_i label = -1)

Which number is bigger? That's your prediction!

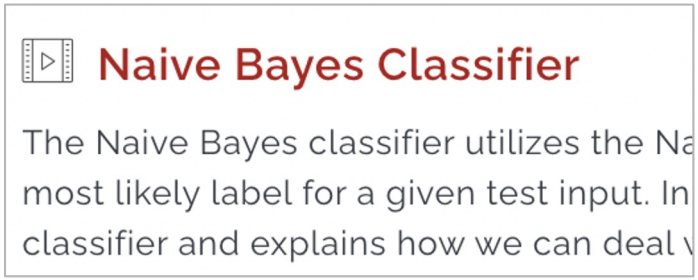
naivebayes_pred

Given novel X (X_test_i) calculate likelihood this would be a boy

(Put another way, assume X_test_i label = 1. How likely was that?)

Repeat, for girl (assume X_test_i label = -1)

Which number is bigger? That's your prediction!



This is a graded discussion: **0 points possible** due -

Improving Focus on Features

Discussion Topic:

- What are some different ways to "featurize" text strings?
- In particular, what features seem to best capture gender in names?
- Can you come up with some better features than those used in the project?
- How well do these features apply across languages?
- Are the assumptions made in Naive Bayes reasonable for this application?

Instructions:

- You are required to participate meaningfully in all course discussions.
- The instructor will review the discussion and you will receive a "complete" or "incomplete" grade based on the quality of your contributions.
- Limit your comments to 200 words.
- You are strongly encouraged to post a response to at least two of your peers' posts.

Questions?

Thank You For Attending!

End of Live Session 2