PYTHON AND DATA SCIENCE

Prof. Chris Jermaine
cmj4@cs.rice.edu

Python

- Old language, first appeared in 1991
  - — But updated often over the years
- Important characteristics
  - — Interpreted
  - — Dynamically-typed
  - — High level
  - — Multi-paradigm (imperative, functional, OO)
  - — Generally compact, readable, easy-to-use
- Boom on popularity last five years
  - — Now the first PL learned in many CS departments

Python: Why So Popular for Data Science?

- Dynamic typing/interpreted
  - — Type a command, get a result
  - — No need for compile/execute/debug cycle
- Quite high-level: easy for non-CS people to pick up
  - — Statisticians, mathematicians, physicists...
- More of a general-purpose PL than R
  - — More reasonable target for larger applications
  - — More reasonable as API for platforms such as Spark
- Can be used as lightweight wrapper on efficient numerical codes
  - — Unlike Java, for example

Python Basics

- Since Python is interpreted, can just fire up Python shell
  - Then start typing
- A first Python program

```
def Factorial (n):
     if n == 1 or n == 0:
          return 1
     else:
          return n * Factorial (n - 1)

Factorial (12)
```

Python Basics Continued

- Spacing and indentaton
  - Indentation important... **no** `begin/end` **nor** `{}`... indentation signals code block
  - Blank lines important; can't have blank line inside of indented code block
- Variables
  - No declaration
  - All type checking dynamic
  - Just use

Python Basics Continued

- Dictionaries
  - — Standard container type is dictionary/map
  - — Example: `wordsInDoc = {}` creates empty dictionary
  - — Add data by saying `wordsInDoc[23] = 16`
  - — Now can write something like `if wordsInDoc[23] == 16:` ...
  - — What if `wordsInDoc[23]` is not there? Will crash
  - — Protect with if `wordsInDoc.get (23, 0)` ... returns 0 if key 23 not defined
- Functions/Procedures
  - — Defined using `def myFunc (arg1, arg2):`
  - — Make sure to indent!
  - — Procedure: no `return` statement
  - — Function: `return` statement

Python Basics Continued

- Loops
  - Of form `for var in range (0, 50):`
    loops for var in {0, 1, ..., 49}
  - Or `for var in dataStruct:`
    loops through each entry in `dataStruct`
  - `dataStruct` can be an array, or a dictionary
  - If array, you loop through the entries
  - If dictionary, you loop through the keys
  - Try

```
a = {}
a[1] = 'this'
a[2] = 'that'
a[3] = 'other'
for b in a:
     a[b]
```

NumPy

- NumPy is a Python package
- Most important one for data science!
  - Can use it to do super-fast math, statistics
  - Most basic type is NumPy `array`
  - Used to store vectors, matrices, tensors
- You will get some reasonable experience with NumPy
- Load with `import numpy as np`
- Then can say, for example, `np.random.multinomial (numTrials, probVector, numRows)`

NumPy Arrays: Your Best Friend In DS

- Writing control flow code in DS programming is **BAD**
- (Kind of like in SQL)
- Python is **interpreted**
  - — Time for each statement execution generally large
- Fewer statements executed, even if work same == performance
- Goal:
  - — Try to replace dictionaries with NumPy arrays
  - — Try to replace loops with bulk array operations
  - — Backed by efficient, low-level implementations
  - — Known as "vectorized" programming

Useful Array Creation Functions

- To create a 2 by 5 array, filled with 3.14
  - `np.full((2, 5), 3.14)`
- To create a 2 by 5 array, filled with zeros
  - `np.zero((2, 5))`
- To create an array with odd numbers thru 10
  - `np.arange(1, 11, 2)` gives [1, 3, 5, 7, 9]
- To tile an array
  - `np.tile (np.arange(1, 11, 2), (1, 2)` gives [1, 3, 5, 7, 9, 1, 3, 5, 7, 9]
  - `np.tile (np.arange(1, 11, 2), (2, 1)` gives [[1, 3, 5, 7, 9], [1, 3, 5, 7, 9]]

Subscripting Arrays

- To compute various tabulations, need to access subarrays
  - Ex: `array` is [[1, 2, 3, 4, 5], [2, 3, 4, 5, 6], [3, 4, 5, 6, 7]]
  - `array[1:,]` or `array[1:]` is [[2, 3, 4, 5, 6], [3, 4, 5, 6, 7]]
  - Why? Gets rows 1, 2, 3, ...
  - `array[2:3,]` or `array[2:3]` is [[3, 4, 5, 6, 7]]
  - Why? Gets row 2
  - `array[0:2,]` or `array[0:2]` is [[1, 2, 3, 4, 5], [2, 3, 4, 5, 6]]
  - `array[:,1:3]` is [[2, 3], [3, 4], [5, 6]]
  - `array[:,np.array([1,2])]`is also [[2, 3], [3, 4], [5, 6]]

Aggregations Over Arrays

- In statistical/data analytics programming...
  - Tabulations, max, min, etc. over NumPy arrays are ubiquitous
- Key operation allowing this is sum
  - Ex: `array` is [1, 2, 3, 4, 5]
  - `array.sum ()` is 15
- Can sum along dimension of higher-d array.
  - Ex: `array` is [[1, 2, 3, 4, 5], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]
  - `array.sum (0)` is [3, 6, 9, 12, 15]
  - `array.sum (1)` is [15, 15, 15]

Other Useful Tabulaton Functions

- To compute max:
  - Ex: `array` is [[10, 2, 3, 4, 5], [2, 13, 4, 5, 6], [3, 4, 5, 6, 7]]
  - `array.max()` is 13
  - Can tabulate over dimensions
  - `array.max(0)` is [10, 13, 5, 6, 7]
  - `array.max(1)` is [10, 13, 7]
- To compute the position of the max:
  - Ex: `array` is [[10, 2, 3, 4, 5], [2, 13, 4, 5, 6], [3, 4, 5, 6, 7]]
  - `array.argmax()` is 6
  - `array.argmax(0)` is [0, 1, 2, 2, 2]

Now You Know Enough For Lab 3

- So let's look at some "real-life" math/stat Python code
- We'll write some code having to do with a commonly-used statistical model for text: "Latent Dirichlet Allocation" or LDA
- LDA: stochastic model for generating a document corpus
- Most widely-used "topic model"
- A "topic" is a set of words that appear to gether with high prob
  - — Intuitively: set of words that all have to do with the same subject
- Often, we want to "learn" an LDA model from an existing corpus
  - — But can also use it to generate a corpus
  - — Which we will do today...

LDA Typically Used To Analyze Text

- Idea:
  - — If you can analyze a corpus...
  - — And figure out a set of *k* topics...
  - — As well as how prevalent each topic is in each document
  - — You then know a lot about the corpus
  - — Ex: can use this prevalence info to search the corpus
  - — Two docs have similar topic compositions? Then they are similar!

OK, So What Does This Have To Do W Text?

- Basic LDA setup
  - LDA will generate *n* random documents given a dictionary
  - Dictionary is of size num_words
  - Best shown thru an example
  - In our example: dictionary will have: (0, “bad”) (1, “I”) (2, “can’t”) (3, “stand”) (4, “comp 215”), (5, “to”) (6, “leave”) (7, “love”) (8, “beer”) (9, “humanities”) (10, “classes”)

LDA Step One

- Generate each of the *k* "topics"
    - — Each topic is represented by a vector of probabilities
    - — The *w*th entry in the vector is associated with the *w*th word in the dictionary
    - — $\text{wordsInTopic}_t[w]$ is the probability that topic *t* would produce word *w*
    - — Vector is sampled from a Dirichlet (alpha) distribution
    - — So, for each *t* in $\{0...k - 1\}$, $\text{wordsInTopic}_t \sim$ Dirichlet (alpha)

LDA Step One

- Generate each of the *k* "topics"
  - — Each topic is represented by a vector of probabilities
  - — The *w*th entry in the vector is associated with the *w*th word in the dictionary
  - — $\text{wordsInTopic}_t[w]$ is the probability that topic *t* would produce word *w*
  - — Vector is sampled from a Dirichlet (alpha) distribution
  - — So, for each *t* in {0...*k* - 1}, $\text{wordsInTopic}_t \sim$ Dirichlet (alpha)
- Ex: $k = 3$
  - — $\text{wordsInTopic}_0 =$ (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — $\text{wordsInTopic}_1 =$ (0, .2, .2, .2, 0, 0, 0, 0, 0, .2, .2)
  - — $\text{wordsInTopic}_2 =$ (0, .2, .2, 0, .2, 0, .2, .2, 0, 0, 0)

LDA Step Two

- Generate the topic proportions for each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{topicsInDoc}_d[t]$ is the probability that an arbitrary word in document *d* will be controlled by topic *t*
  - — Vector is sampled from a Dirichlet (beta) distribution
  - — So, for each *d* in {0...*n* - 1}, $\text{topicsInDoc}_d \sim \text{Dirichlet}(\text{beta})$

LDA Step Two

- Generate the topic proportions for each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{topicsInDoc}_d[t]$ is the probability that an arbitrary word in document d will be controlled by topic t
  - — Vector is sampled from a Dirichlet (beta) distribution
  - — So, for each d in $\{0...n - 1\}$, $\text{topicsInDoc}_d \sim$ Dirichlet (beta)
- Ex: $n = 4$
  - — $\text{topicsInDoc}_0 =$ (.98, 0.01, 0.01)
  - — $\text{topicsInDoc}_1 =$ (0.01, .98, 0.01)
  - — $\text{topicsInDoc}_2 =$ (0.02. .49, .49)
  - — $\text{topicsInDoc}_3 =$ (.98, 0.01, 0.01)

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial $(\text{topicsInDoc}_d, 1)$ distribution

    (2) Generate the word by sampling from a Multinomial $(\text{wordsInTopic}_t, 1)$ distribution
- Ex: doc 0... $\text{topicsInDoc}_0 = (.98, 0.01, 0.01)$
  - — *t* for word zero is...

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = = (.98, 0.01, 0.01)
  - — *t* for word zero is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using wordsInTopic_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:
    - (1) Figure out the topic t that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution
    - (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution
- Ex: doc 0... topicsInDoc_0 = = (.98, 0.01, 0.01) “I”
  - — t for word zero is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using wordsInTopic_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0), which is equivalent to “I”

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution
- Ex: doc 0... topicsInDoc_0 = = (.98, 0.01, 0.01) "I"
  - — Now onto the next word

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = = (.98, 0.01, 0.01) "I"
  - — *t* for word one is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = = (.98, 0.01, 0.01) "I can't"
  - — *t* for word one is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using wordsInTopic_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0), which is equivalent to "can't"

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — wordsInDoc$_d$[w] is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topicsInDoc$_d$, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic$_t$, 1) distribution

- Ex: doc 0... topicsInDoc$_0$ = = (.98, 0.01, 0.01) "I can't"
  - — Now onto the next word

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) "I can't"
  - — *t* for word two is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word w in document d
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc d:

    (1) Figure out the topic t that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) "I can't stand"
  - — t for word two is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using wordsInTopic_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0), which is equivalent to "stand"

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:
    - (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution
    - (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution
- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) “I can’t stand”
  - — Onto next word

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution
- Ex: doc 0... $\text{topicsInDoc}_0 =$ (.98, 0.01, 0.01) "I can't stand"
  - — *t* for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) "I can't stand bad"
  - — *t* for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using wordsInTopic_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), which is equivalent to "bad"

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) "I can't stand bad"
  - — Onto the last word in the document

LDA Step Three

- Generate the words in each document
  - — Each topic “controls” a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution
- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) “I can’t stand bad”
  - — *t* for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]

LDA Step Three

- Generate the words in each document
  - — Each topic "controls" a subset of the words in a document
  - — $\text{wordsInDoc}_d[w]$ is the number of occurences of word *w* in document *d*
  - — To get this vector, generate the words one-at-a-time
  - — For a given word in doc *d*:

    (1) Figure out the topic *t* that controls it by sampling from a Multinomial (topicsInDoc_d, 1) distribution

    (2) Generate the word by sampling from a Multinomial (wordsInTopic_t, 1) distribution

- Ex: doc 0... topicsInDoc_0 = (.98, 0.01, 0.01) "I can't stand bad beer"
  - — *t* for word three is zero, since we sampled (1, 0, 0) [there is a 1 in the zeroth entry]
  - — So we generate the word using wordsInTopic_0 = (.2, .2, .2, .2, 0, 0, 0, 0, .2, 0, 0)
  - — And we get (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0), which is equivalent to "beer"

In The End... For Doc 0...

- text is “I can’t stand bad beer” (equiv. to “1 2 3 0 8”)
- topicsInDoc_0 = (.98, 0.01, 0.01)
- wordsInDoc_0 = (1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0)
  - — Why? Word 0 appears once, word 1 appears once, word 4 zero times, etc.
- produced_0= (1, 1, 1, 1, 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)
  - — Why? Topic 0 (associated with first line) produced 5 words

    Those words were (1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0)
  - — Topic 1, topic 2 produced no words
  - — “produced” always a matrix with num_words cols, *k* rows

Repeat For Each Doc in the Corpus!

For Example, Let's Look At Doc 2...

- topicsInDoc_2 = (.02, 0.49, 0.49)
- Imagine that when we generate doc 2, we get:
  - Word 0: produced by topic 2, is 1 or "I"
  - Word 1: produced by topic 2, is 7 or "love"
  - Word 2: produced by topic 2, is 8 or "beer"
  - Word 3: produced by topic 1, is 1 or "I"
  - Word 4: produced by topic 1, is 2 or "can't"
  - Word 5: produced by topic 2, is 7 or "love"
  - Word 6: produced by topic 1, is 9 or "humanities"
  - Word 7: produced by topic 1, is 10 or "classes"
- wordsInDoc_2 = (0, 2, 1, 0, 0, 0, 0, 2, 1, 1, 1)
- produced_2= (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
  (0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1)
  (0, 1, 0, 0, 0, 0, 0, 2, 1, 0, 0)

OK, Back To Python!

- Let's look at some code that (mostly) implements LDA
  - Check out `cmj4.web.rice.edu/LDADictionaryBased.html`

Uses Lot's o' NumPy Functionality

- `np.random.multinomial (numTrials, probVector, numRows)`
  - Take numRows samples from a Multinomial (probVector, numTrials) dist
- `np.random.multinomial (numTrials, probVector, numRows)`
  - Take numRows samples from a Multinomial (probVector, numTrials) dist
  - Put in a matrix with numRows rows
- `np.flatnonzero (array)`
  - Return array of indices of non-zero elements of array
- `np.random.dirichlet (paramVector, numRows)`
  - Take numRows samples from a Dirichlet (paramVector) dist
- `np.full (numEntries, val)`
  - Create a NumPy array with the spec'ed number of entries, all set to val

NumPy

- Can you complete the activity?
  - cmj4.web.rice.edu/LDADictionaryBased330.html

Problem: Bad Code!

- No one should write statistical/math Python code this way
- Vectorized is Better!

Better Code

- Check out `cmj4.web.rice.edu/LDAArrays330.html`
- No dictionaries here! Just arrays.
  - Can you complete the code?

Advantages of Vectorization

- Co-occurence analysis
  - fundamental task in many statistical/data mining computations
- In text processing...
  - Given a document corpus
  - Want to count number of times (word1, word2) occur in same doc in corpus
- Your task in Lab 4: build three implementations
  - Utilizing varying degrees of vectorization
  - We will time each, see which is faster

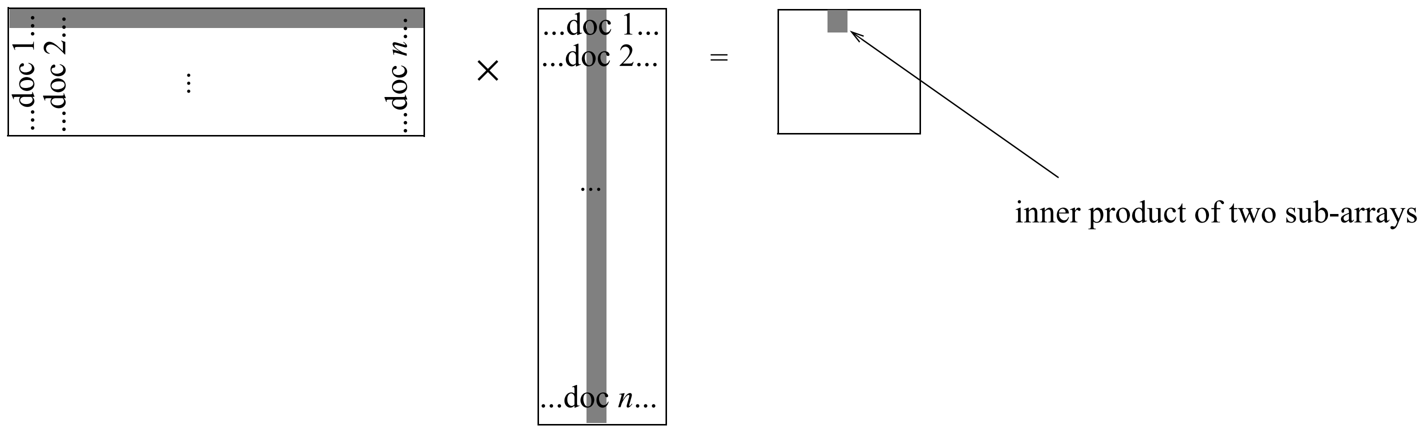
Imp 1: Pure Dictionary-Based

- Pure nested loops implementation
  - — Has advantage that wordsInCorpus is sparse
  - — Only numDocs $\times$ (numDistinctWordsPerDoc)^2 execs of inner loop
  - — But Python is an interpreted language!!

Imp 2: Vector-Based with Loop over Docs

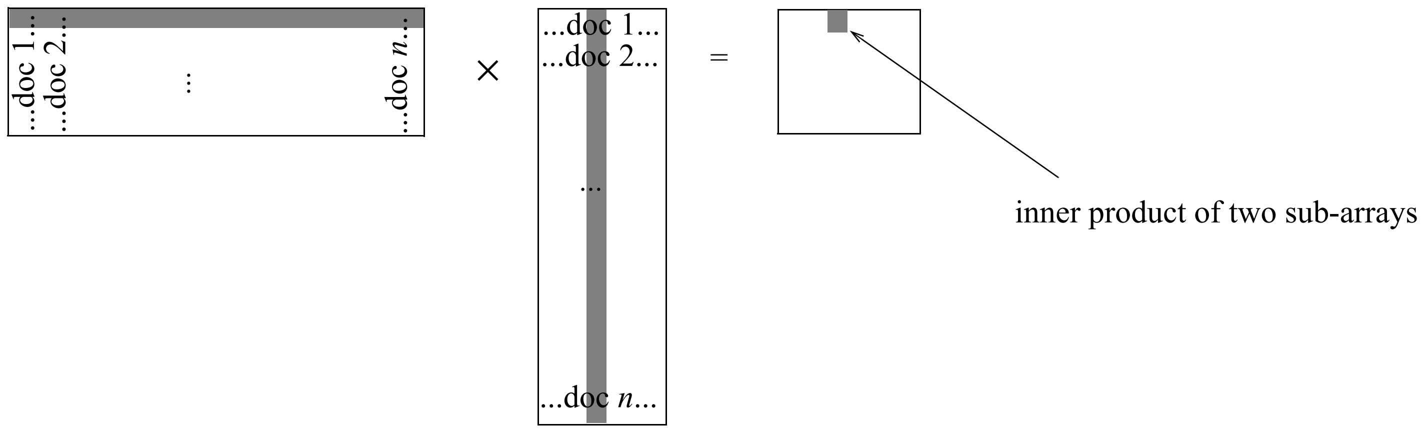
- Given a 1-d array `array` = [0, 0, 3, 1, 0, 1...]...
  - The *outer product* of `array` with itself creates a 2-d matrix
  - Where *i*th row is `array[i] × array`
  - So if an `array` gives number of occurs of each word in a doc...
  - And we *clip* `array` so [0, 0, 3, 1, 0, 1...] becomes [0, 0, 1, 1, 0, 1...]
  - Then take outer product of `array` with itself...
  - Entry at pos `[i, j]` is number of co-occurs of dictionary words `i, j` in doc
- Note:
  - `np.outer (arrayOne, arrayTwo)` is outer product of arrays
  - `np.clip (array, low, high)` clips all entries to max of `high`, min of `low`

Imp 3: Pure Vector-Based



- Note that after matrix multiply
  - — Entry at pos `[i, j]` is inner product of row `i` from LHS, col `j` from RHS
  - — So if row `i` is number of occurs of word `i` in every doc
  - — And if col `j` is number of occurs of word `j` in every doc
  - — Entry at pos `[i, j]` is number of co-occurs of words `i, j`
  - — Suggests a super-efficient algorithm

Imp 3: Pure Vector-Based



- Some notes:
  - — `np.transpose (array)` computes transpose of matrix in `array`
  - — `np.dot (array1, array2)` computes dot product of 1-d arrays, matrix multiply of 2-d

These Three Implementations: Lab 4

- Questions?