Building Solutions
Naive Bayes
So...what are YOU doing this summer?

I HIGHLY recommend having a project of some sort.

- Volunteer Work - non-profit organizations
- Independent Projects
  - With friends
  - Form a team by interests/skills/goals
- Summer Research Opportunities
  
  “If you know of a student who may be interested in summer research but is not yet matched to a research group, please have them fill out this application: https://go.cs.illinois.edu/reuapplication”
You Are Doing Better Than You Think...

You have not lost your mind. This is a LOT of work.

Everyone is in the same boat and the curve is generous.

Other cohorts of students have had the same assignment with only minor variations.

The staff is not crying. (this is how I know things are normal)
Naive Bayes: The magic we’d LIKE to see:

main (argc, argv) {
    “Open stream”
    my_stream>>entirely_organized_object_containing_all_useful_functionality;
    “Close stream”
    “Use entirely_organized_object_containing_all_useful_functionality”
}
Top Down vs Bottom Up

If you start at:

```
entirely_organized_object_containing_all_useful_functionality
```

and work your way down to the details, you are practicing top down design.

It’s a good way to design and plan but it may be onerous for code construction. Nothing works until everything works.
How to start?

1. Use existing things (data types, data structures, classes)
2. Create simple things
3. Combine simple things to make more complex things
4. Take baby steps
5. Use new things when:
   a. It provides convenience to reach your goals
   b. It is inescapable
6. Refactor
   a. Reorganize your code
   b. Move things
   c. Remove unnecessary code
Where to start?

Baby Step:

Open and read a file line-by-line*

1. Can I open a file?
2. Can I read to the end?
3. How do I know I’ve done that?

*what about skipping around or reading all of the file at once?
skipping around or reading all of the file at once

Complicated

Skipping around requires maintaining more information and putting more responsibility on your code.

Reading the whole file at once requires enough memory.
Where to start?

Baby Step:

Open and read a file line-by-line*

1. Can I open a file?
2. Can I read to the end?
3. How do I know I’ve done that?

*Requirement: use streams
Capturing Data

I want to store data in an organized manner.

Remember types of classes:

1. Entity
2. Control
3. Boundary
4. Value

I have an “entity” around which to construct a class: Image

Create a class for my “entity”.
Entity Class: Image

I need:

1. Label
2. Rows
3. Columns
4. Data - Flexibility of dimensions at time of instantiation.
Entity Class: Image

I need:

1. Label
2. Rows
3. Columns
4. Data - Flexibility of dimensions at time of instantiation.

Initially:

1. I will just store the raw data: lines of text
2. I will assume that columns = rows
I need:

1. Label
2. Rows
3. Columns
4. Data - Flexibility of dimensions at time of instantiation.

Ultimately, I want a statistical model for the occurrence of each pixel in each unique classification of image. I will want doubles that are stored for convenient accessibility.
Entity Class: Image

I need:

1. Label
2. Rows
3. Columns
4. Data - **Flexibility of dimensions at time of instantiation.**

What data structure allows this kind of flexibility?

*(Let’s ignore my Array2D class as an option)*
Entity Class: Image

FOR EVERY IMAGE IN DATA FILE*:

1. Label
2. Rows
3. Columns
4. Data - **
   a. Raw: a vector of strings (str.length() should always = “rows”) 
   b. Occurrences: integers accessible by row, column 
   c. Model: double accessible by row, column 

* ultimately, only for unique labels

**this is my set of proposed baby steps
Read ONE set of data directly into an Image

I want to write code that will:

1. Just read this data assuming it’s all perfect
   a. Label
   b. 28 lines of strings representing image
   c. Strings are 28 wide

2. Add logic to check validity (e.g. - line length is what is expected)
   a. One line with “label”
   b. Followed by “rows” count of lines
   c. each line is of length “columns”
Check my work

Output the contents of my “Image”

I can visually inspect the contents of the data file and compare my output.

I can overload “<<” to make this easier
Read and store ALL of the data

1. Read all of the data by looping until the end of the file.
   a. Read an entire image at a time
   b. Now may be a good time to overload “>>”
2. Create a vector of images (vector<image>) to store ALL of the raw data.
3. Push_back EVERY Image to the vector.
4. Loop through the vector and verify it worked.

Notice that I have not started using formal testing yet.

Is this OK? If so, why? If not, why not?
Formal Testing

Behaviors - publicly accessible

At what point did I enter this territory?
Adding convenience to Image

1. Now may be a good time to overload “<<” so that I can examine the contents of each object easily.
2. Next, I’ll overload “>>” - adding something new
   a. It’s a convenient time
   b. Adds convenient functionality
Discussion: Formal Tests

Notice that I have not started using formal testing yet.

Is this OK? If so, why? If not, why not?
Counting Pixels

1. count shaded pixels
2. store counts
3. combine the counts from multiple images with the same label
Identify Unique Labels

Every time we encounter a new label, push_back to:

    vector<string> unique_labels

How do we know if it’s new?

    Search through unique_labels to see if it’s already there.

*I’m still storing EVERY image. Now I have a vector of unique labels.
Unique Labels

Now I know how to identify when an incoming image is new or if it already exists.

Why do I care here?
Unique Labels

Now I know how to identify when an incoming image is new or if it already exists.

Why do I care here?

1. I can choose to store a new image or combine data.
2. Combining data ULTIMATELY means counting the occurrences of shaded pixels.
Since I have all the data...

Iterate through the entire list of data

For each image:

    For each pixel in Image
        check if shaded
        Spit out 1 for shaded, 0 for not.
        You can “see” the image in the output.

*I haven’t stored that data yet.
Since I have all the data...

Iterate through the entire list of data

For each image:

   For each pixel in Image

       check if shaded

       Spit out 1 for shaded, 0 for not.

You can “see” the image in the output.

What if I returned a set of data accessible by (row, column) of 1s and 0s?
Should I return...

Goals:

For each unique label:

**Statistical model** consisting of integers representing the count for each pixel

It would be convenient if I could:

1. Store and retrieve these by their unique identifier
2. Update the stat_model with new image data
   a. Add a matrix of integers to an existing matrix with same dimensions?
   b. Update a row in a matrix of counts with a single “line” of raw input?
   c. Update a stat_model with a raw image?
I need:

1. String label (do we really need this?)
2. Int Rows
3. Int Columns
4. Int “Number_of_images_in_model” (excessively long and descriptive name)
5. Data - rows x columns of integers containing counts for pixels
So now we are at a point where we can...

1. Read a line
2. Determine if it starts a new image
   a. Creates a new Image pixel count object and puts it in our vector
   b. OR Updates the pixels counts of an existing image

- We can process one line of input at a time and update the stat_model
- create a stat_model of 0 and 1 for any given image (raw input data) and write a matrix addition method or overload “+”.
Output: baby steps

Open a file for writing

Write something to it

Write some properly formatted data to that file

Write all of your data in a convenient form
Input stat_model data: baby steps

Steps:

Open a file for reading (we already know how to do that now)
Correctly read in one model record in the current (convenient) format
Read to the end of the file and populate a data structure with all of the models
Naive Bayes Week2

https://courses.grainger.illinois.edu/cs126/sp2021/assignments/naive-bayes-week2/