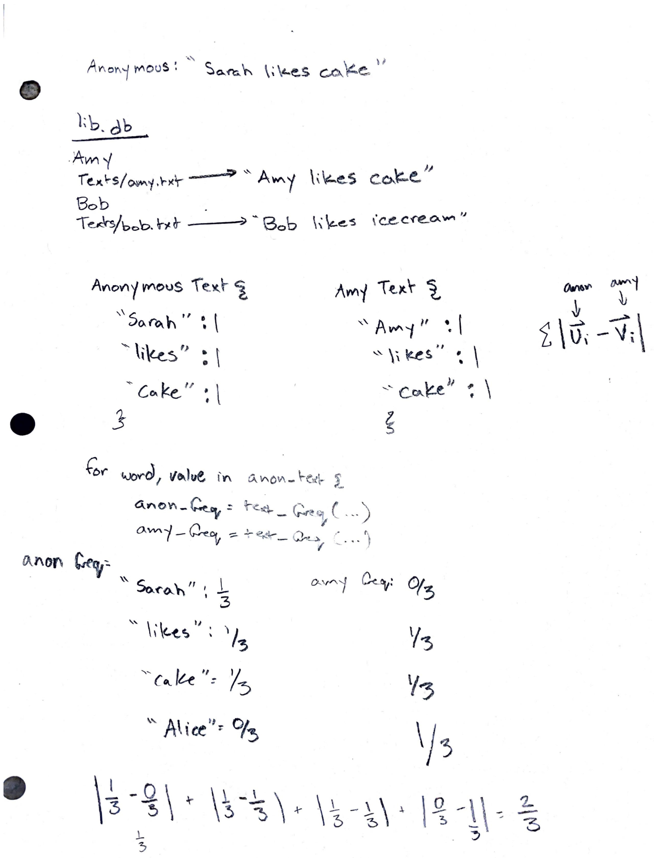
Assignment 7: Author Identification

Aniket Pratap

March 2022

1 Introduction

The purpose of this assignment is to observe the writing styles of certain author, given a data base, and when given an unknown author, determine who it is based on a k-nearest neighbors algorithm. The goal is to parse in text using regex, hash-tables and a bloom filter, and use various distance formulas in order to determine the nearest neighbor. A simple example will look something like this:



Anonymous: " Sarah likes cake "

lib.db

Amy
Texts/amy.txt → " Amy likes cake "
Bob
Texts/bob.txt → " Bob likes icecream "

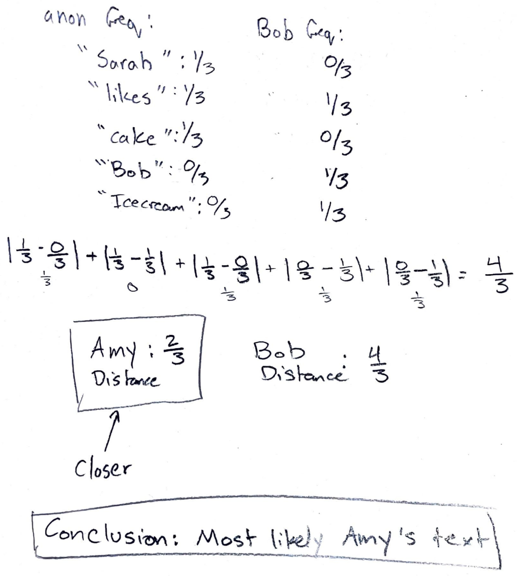
```
Anonymous Text {          Amy Text {
  "Sarah" : 1               "Amy" : 1
  "likes" : 1               "likes" : 1
  "cake" : 1                "cake" : 1
}                         }
```

$$\sum \left| \overset{\text{anon}}{\vec{U}_i} - \overset{\text{amy}}{\vec{V}_i} \right|$$

```
for word, value in anon-text {
    anon-freq = text_freq(...)
    amy-freq = text_freq(...)
```

anon freq=
"Sarah": $\frac{1}{3}$ amy freq: 0/3
"likes": 1/3 1/3
"cake": 1/3 1/3
"Alice": 0/3 1/3

$$\underbrace{\left|\frac{1}{3}-\frac{0}{3}\right|}_{\frac{1}{3}} + \left|\frac{1}{3}-\frac{1}{3}\right| + \left|\frac{1}{3}-\frac{1}{3}\right| + \left|\frac{0}{3}-\frac{1}{3}\right| = \frac{2}{3}$$



anon Freq:	Bob Freq:
"Sarah" : 1/3	0/3
"likes" : 1/3	1/3
"cake" : 1/3	0/3
"Bob" : 0/3	1/3
"Icecream" : 0/3	1/3

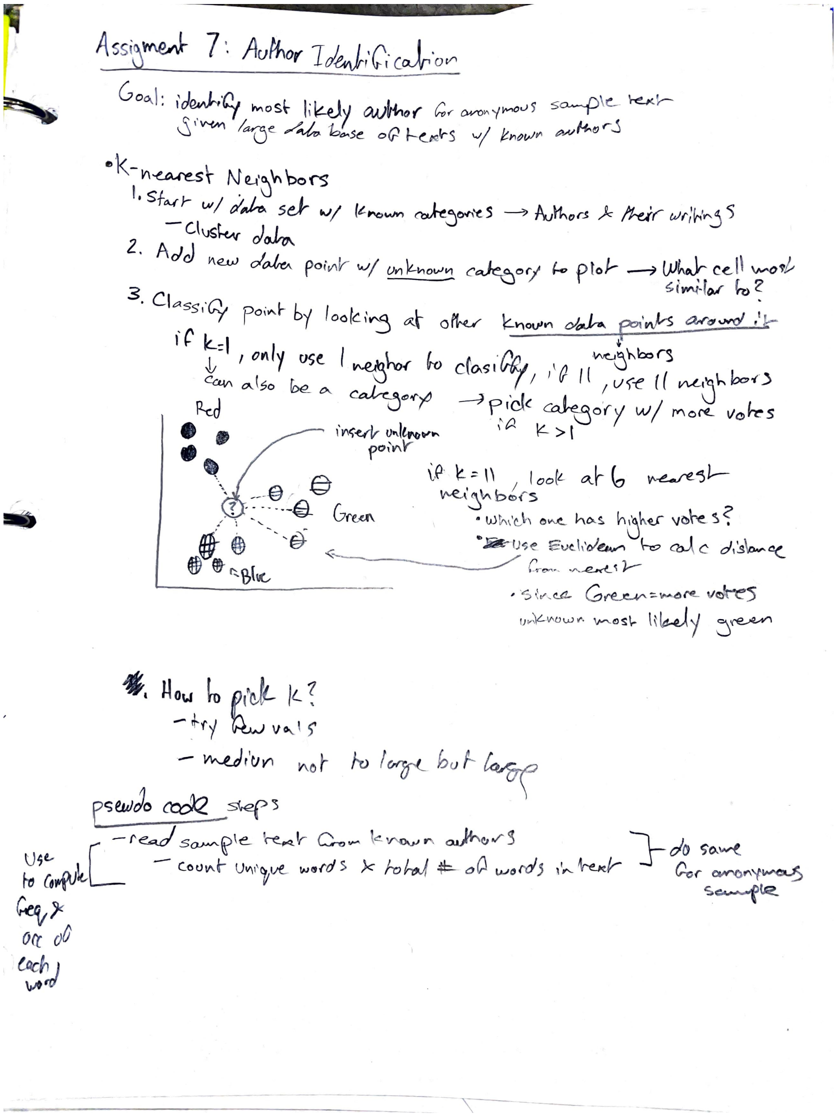
$$\underbrace{\left|\frac{1}{3}-\frac{0}{3}\right|}_{\frac{1}{3}}+\underbrace{\left|\frac{1}{3}-\frac{1}{3}\right|}_{0}+\underbrace{\left|\frac{1}{3}-\frac{0}{3}\right|}_{\frac{1}{3}}+\underbrace{\left|\frac{0}{3}-\frac{1}{3}\right|}_{\frac{1}{3}}+\underbrace{\left|\frac{0}{3}-\frac{1}{3}\right|}_{\frac{1}{3}}=\frac{4}{3}$$

$\boxed{\text{Amy Distance} : \frac{2}{3}}$ Bob Distance : $\frac{4}{3}$

Closer

$\boxed{\text{Conclusion: Most likely Amy's text}}$

As you can see, Amy's is closer to the anonymous text than Bob——meaning we can conclude that the anonymous text is most likely from Amy. An example of k-nearest neighbors and distance calculations can be seen by:



Assigment 7: Author Identification

Goal: identify most likely author for anonymous sample text given large data base of texts w/ known authors

- K-nearest Neighbors
  1. Start w/ data set w/ known categories → Authors & their writings
     - Cluster data
  2. Add new data point w/ unknown category to plot → What cell most similar to?
  3. Classify point by looking at other known data points around it

     if k=1, only use 1 neighbor to clasiffy, if 11 neighbors, use 11 neighbors

     ↓ can also be a category → pick category w/ more votes if k>1

Red

insert unknown point

?

Green

Blue

if k=11, look at 6 nearest neighbors

- which one has higher votes?
- use Euclidean to calc distance from nearest
- since Green=more votes unknown most likely green

. How to pick k?

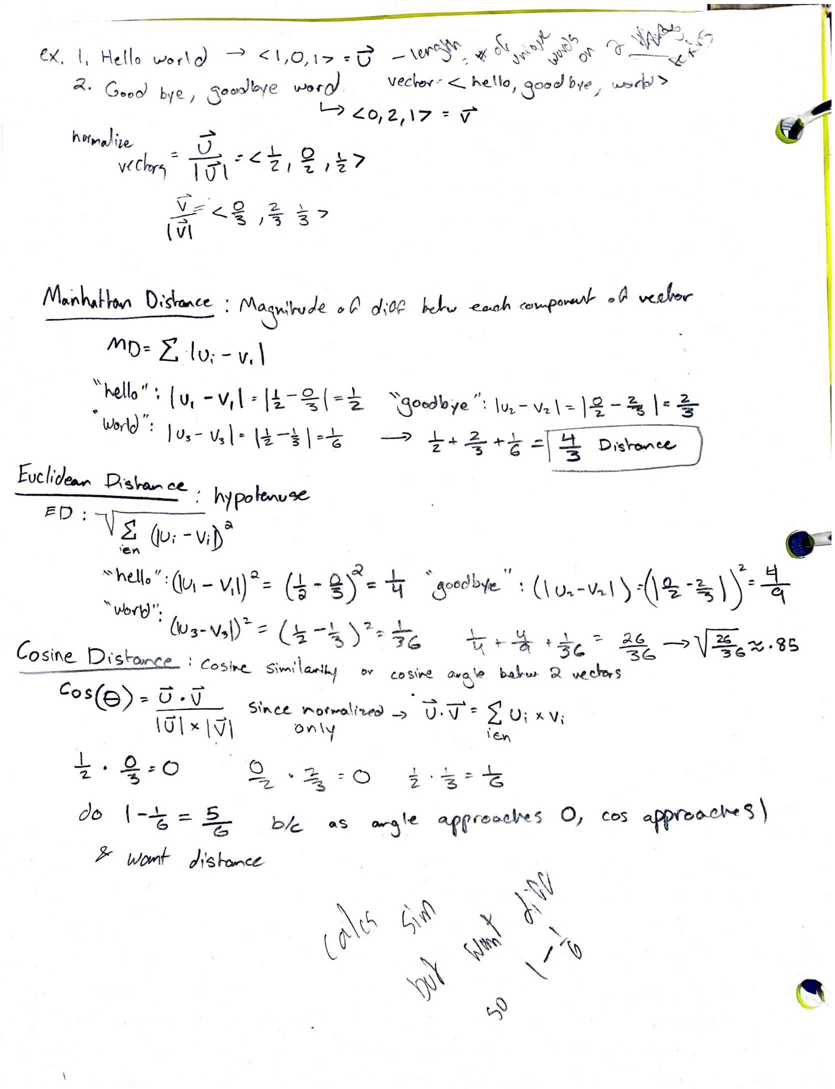
- try few vals
- medium not to large but large

pseudo code steps

- read sample text from known authors
- count unique words & total # of words in text

Use to compute freq, & occ of each word

do same for anonymous sample



ex. 1. Hello world → $\langle 1,0,1\rangle = \vec{U}$ — length = # of unique words on 2 ~~things~~ texts

2. Good bye, goodbye word vector $\langle$ hello, goodbye, world $\rangle$

↳ $\langle 0,2,1\rangle = \vec{V}$

normalize vectors = $\frac{\vec{U}}{|\vec{U}|} = \langle \frac{1}{2}, \frac{0}{2}, \frac{1}{2} \rangle$

$\frac{\vec{V}}{|\vec{V}|} = \langle \frac{0}{3}, \frac{2}{3}\ \frac{1}{3} \rangle$

Manhattan Distance : Magnitude of diff betw each component of vector

$MD = \sum |u_i - v_i|$

"hello" : $|u_1 - v_1| = |\frac{1}{2} - \frac{0}{3}| = \frac{1}{2}$ "goodbye" : $|u_2 - v_2| = |\frac{0}{2} - \frac{2}{3}| = \frac{2}{3}$

"world" : $|u_3 - v_3| = |\frac{1}{2} - \frac{1}{3}| = \frac{1}{6}$ → $\frac{1}{2} + \frac{2}{3} + \frac{1}{6} = \boxed{\frac{4}{3} \text{ Distance}}$

Euclidean Distance : hypotenuse

ED : $\sqrt{\sum_{i \in n} (|u_i - v_i|)^2}$

"hello" : $(|u_1 - v_1|)^2 = (\frac{1}{2} - \frac{0}{3})^2 = \frac{1}{4}$ "goodbye" : $(|u_2 - v_2|) = (|\frac{0}{2} - \frac{2}{3}|)^2 = \frac{4}{9}$

"world" : $(|u_3 - v_3|)^2 = (\frac{1}{2} - \frac{1}{3})^2 = \frac{1}{36}$ $\frac{1}{4} + \frac{4}{9} + \frac{1}{36} = \frac{26}{36} \rightarrow \sqrt{\frac{26}{36}} \approx .85$

Cosine Distance : Cosine similarity or cosine angle betw 2 vectors

$\cos(\theta) = \frac{\vec{U} \cdot \vec{V}}{|\vec{U}| \times |\vec{V}|}$ since normalized only → $\vec{U} \cdot \vec{V} = \sum_{i \in n} u_i \times v_i$

$\frac{1}{2} \cdot \frac{0}{3} = 0$ $\frac{0}{2} \cdot \frac{2}{3} = 0$ $\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$

do $1 - \frac{1}{6} = \frac{5}{6}$ b/c as angle approaches 0, cos approaches 1

& want distance

calcs sim

but want diff

so $1 - \frac{1}{6}$

2 Pseudo-code

2.1 ht.c

This function creates the hash table and hash table iterator

htCreate()
creates the hashtable
defines variables and sets the salts for speck.c
also creates a node array for the words in text

htDelete()
deletes the hashtable along with the node array
deletes every node in node array as well

htSize()
returns size of hashtable

htLookUp()
first uses hash function provided by speck.c to get index
index is then modded by size of hashtable to get proper index
if the spot in the hashtable not null, set node to index
if node at index doesn't match looked up node, increment index until looked up found
worst case: search through whole hashtable

htInsert()
hashes word then inserts into hashtable
if slot not NULL at that point, increment modded index until null found
only insert if hashtable can fit another element

htPrint()
prints hashtable; nulls welcome for debugging

htIter()
increments count by 1 each time htIter is called
only iterates through size of elements inserted rather than whole hashtable
if slot NULL, skip over else return node

3 node.c

This is similar to what I did in huffman, but a node is defined by containing a word and count for that word
nodeCreate()
allocates memory that is size of node
word is also copied over using strdup size using pointer
set count to 0

nodeDelete()
if node doesn't equal NULL, the free and set to NULL
else no need to free

nodePrint()
debugging function
if node NULL, print NULL else print word and count

4 bv.c

similar to code.c from asgn6. Main function is to set and get bit at desired location in a byte

bvCreate()
creates a bit vector of size length(bits)
only allocates minimum bytes needed for vector
if fail„ return NULL

bvDelete()
deletes vector and bv struct
also sets to NULL

bvLength()
returns the length inputted

bvSetBit
index i can't be greater than length-1 because 0 indexing
|= with 1 because want to change only that bit
index divided by 8 and shift modded by8

bvClrBit
copied from asgn6
checks if greater than length -1 because 0 indexing
dividing by 8 gives array index and mod 8 gives amount of shifts return left shift with inverse and mod with 8 while anding

bvGetBit()
same as setBit except right shift and and with 1 since don't want to change value

bvPrint
debugging function
prints each bit by using getBit

5 bf.c

bfCreate()
create and allocate memory using bv.c
set salts to respective positions used by speck.c

bfDelete()
delete bv

bfSize()
return size

bfInsert()
hash word 3 times and get respective index
goal is to get 3 indexes
set the bit at those indexes in the bv

bfProbe()
rehash three times and check all three indexes
if all anded equals 1 , return 1; else false

bfPrint()
prints bf

6 text.c

lower()
function to convert word to lower
uses strlen to get length of string and iterate until end reached
while iterating, used tolower() to convert each char to lower

textCreate()
allocates memory and creates bf and ht with given sizes
if file NULL, return NULL pointer
if noise parameter equals NULL create noise text
parse through noise file and break once user noise limit reached
insert into bf and hashtable
if noise isn't NULL, create Text
textDelete()
deletes text
deletes ht bf
set to NULL

textDist()
create 2 iterators, 1 for each text
based on user metric input, do formula required
read next text and only get words not in text1 then distance

textFrequency()

gets normalized frequency by diving word frequency by word count in text file
textContains()
first check bloom filter
if filter false then not contained
otherwise check ht if bf true for false positive
textPrint()
debugging for text

7 pq.c

entryCreate()
create and allocate memory for an entry
this entry is defined by author and distance

entryDelete()
deletes entry and strdup if created

pqCreate()
creates pq and allocates memory
creates array of pq entries

pqDelete()
deletes every entry in pq
sets values to NULL

pqEmpty()
returns if pq empty

pqFull()
returns if pq full

pqSize()
returns current size of pq
aka what head is pointing at

minChild()
gets min child of parent

fixHeap()
fixes heap by swapping child and parent if neccessary
returns fixed array

entryPrint()
debugging function for entry struct

enqueue()
uses strdup to get author and enqueues to pq
increments head by 1
fixes heap

dequeue()
pops highest order aka least distance
pop first element in partial ordered array
swap with last element
decrease size of array by 1

pqPrint()
debugging function for pq
prints priority queue

8 identify.c

this file acts as the main for the whole program
simple getopt used from other assignments

noise text firt created using NULL
anonymous text then created
scan first line: number of author and path pairs
create pq of size
go into path then enqueue to pq with text distance calculated
dequeue with highest priority (least distance being first)