# Asmt 3: Distances and LSH

Turn in through Canvas by 2:45pm, then come to class:
Wednesday, February 5
100 points

## Overview

In this assignment you will explore LSH and Euclidean distances.

You will use a data set for this assignment:

- http://www.cs.utah.edu/ jeffp/teaching/cs5140/A3/R.csv`http://www.cs.utah.edu/~jeffp/te`

*It is recommended that you use LaTeX for this assignment (or other option that can properly digitally render math). If you do not, you may lose points if your assignment is difficult to read or hard to follow. Find a sample form in this directory: http://www.cs.utah.edu/ jeffp/teaching/latex/*

## 1 Choosing $r, b$ (35 points)

Consider computing an LSH using $t = 160$ hash functions. We want to find all object pairs which have Jaccard similarity above $\tau = .85$.

**A: (15 points)** Use the trick mentioned in class and the notes to estimate the best values of hash functions $b$ within each of $r$ bands to provide the S-curve

$$f(s) = 1 - (1 - s^b)^r$$

with good separation at $\tau$. Report these values. Solution:
We know that $b = -\frac{log(t)}{log(\tau)}, r = \frac{t}{b}$ On substituting values to above equations, we get

$$b = -\frac{log(160)}{log(0.85)} = 31.288$$

b has to be integer. So, rounding it to 32.

$$r = \frac{160}{32} = 5$$

b = 32, r = 5

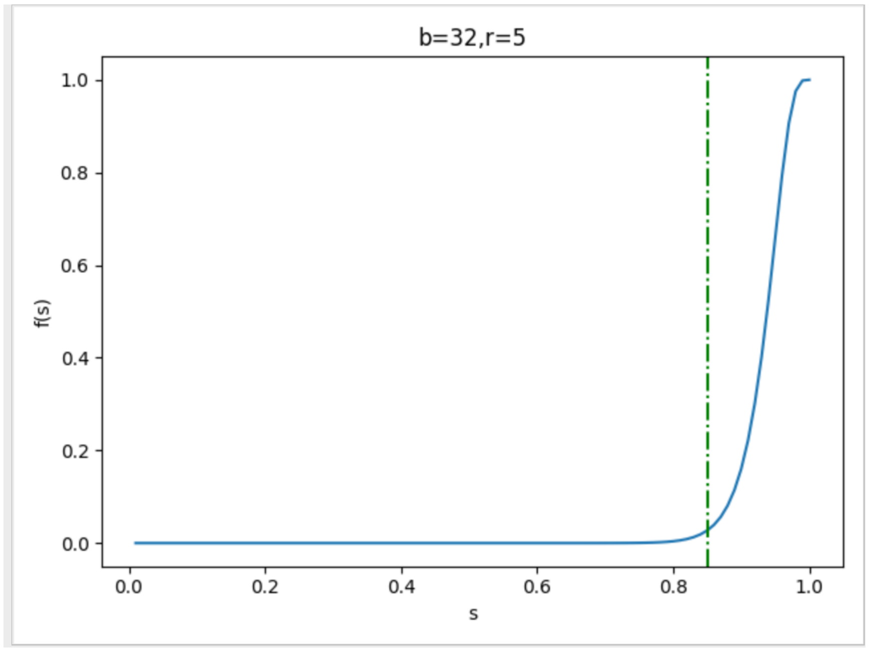

Figure 1: Question 1: Part A - plot of s Vs f(s)

Below is the code to generate values for s vs f(s)

```
import matplotlib.pyplot as plt
import numpy as np
b = 32
r = 5
t = 160
s = []
f_s = []

s = list(np.linspace(.01, 1.0, num=100))
for item in s:
    f_s.append(1-(1-item**b)**r)
```

```
plt.plot(s, f_s)
plt.title("b=32,r=5")
plt.xlabel("s")
plt.ylabel("f(s)")
plt.show()
```

**B: (15 points)** Consider the 4 objects $A, B, C, D$, with the following pair-wise similarities:

| | A | B | C | D |
|---|---|---|---|---|
| A | 1 | 0.77 | 0.25 | 0.33 |
| B | 0.77 | 1 | 0.20 | 0.55 |
| C | 0.25 | 0.20 | 1 | 0.91 |
| D | 0.33 | 0.55 | 0.91 | 1 |

Please refer Figure 1.
Using your choice of $r$ and $b$ and $f(\cdot)$, what is the probability of each pair of the four objects for being estimated to having similarity greater that $\tau = 0.85$? Report 6 numbers. *(Show your work.)*
r = 5, b = 32

| | | | |
|---|---|---|---|
| JS(A,B) | 0.77 | $1-(1-s^b)^r$ | $1-(1-(0.77)^{32})^5 = .0011653883351100403$ |
| JS(A,C) | 0.25 | $1-(1-s^b)^r$ | $1-(1-(0.25)^{32})^5 = 0.0$ |
| JS(A,D) | 0.33 | $1-(1-s^b)^r$ | $1-(1-(0.33)^{32})^5 = 2.220446049250313e-15$ |
| JS(B,C) | 0.20 | $1-(1-s^b)^r$ | $1-(1-(0.20)^{32})^5 = 0.0$ |
| JS(B,D) | 0.55 | $1-(1-s^b)^r$ | $1-(1-(0.55)^{32})^5 = 2.4579670854230073e-08$ |
| JS(C,D) | 0.91 | $1-(1-s^b)^r$ | $1-(1-(0.91)^{32})^5 = 0.2217361224130674$ |

## 2 Generating Random Directions (30 points)

**A: (10 points)** Describe how to generate a single random unit vector in $d = 10$ dimensions using only the operation $u \leftarrow \mathsf{unif}(0,1)$ which generates a uniform random variable between 0 and 1. *(This can be called multiple times.)* Solution:
I generated two random numbers $u_1, and\ u_2$ from random.uniform function in python. Using these two numbers I generated gaussian random variable two independent 1-dimensional Gaussian random variables as

$$y_1 = \sqrt{-2lnu_1} * cos(2 * \pi * u_2)$$

$$y_2 = \sqrt{-2lnu_1} * sin(2 * \pi * u_2)$$

I generated 10 such numbers and store them in a list, say *y_vector*. Then I have taken square root of sum of individual squares to get the magnitude of a 10 - dimensional vector. Then I divide the magnitude to each element in the *y_vector*. I store this into a new list say *unit_vector* which is the random unit vector in d = 10 dimension. Code to generate single random unit vector in d = 10 dimensions:

```
import math
import random
import numpy as np
def gausian_number_generator():
    u1 = random.uniform(0,1)
    u2 = random.uniform(0,1)
    y1 = (math.sqrt(-2*(np.log(u1))))*math.cos(2*math.pi*u2)
    y2 = (math.sqrt(-2*(np.log(u1))))*math.sin(2*math.pi*u2)
    return y1,y2

y_vector = []
for i in range(0,5):
    y1, y2 = gausian_number_generator()
    y_vector.append(y1)
    y_vector.append(y2)


squared_y_vector = []
for item in y_vector:
    squared_y_vector.append(item*item)


magnitude_y_vector = math.sqrt(sum(squared_y_vector))
unit_vector = []
for item in y_vector:
    unit_vector.append(item/magnitude_y_vector)

print(unit_vector)
```

**B: (20 points)** Generate $t = 160$ unit vectors in $R^d$ for $d = 100$. Plot of cdf of their pairwise dot products (yes, you need to calculate $\binom{t}{2}$ dot products).

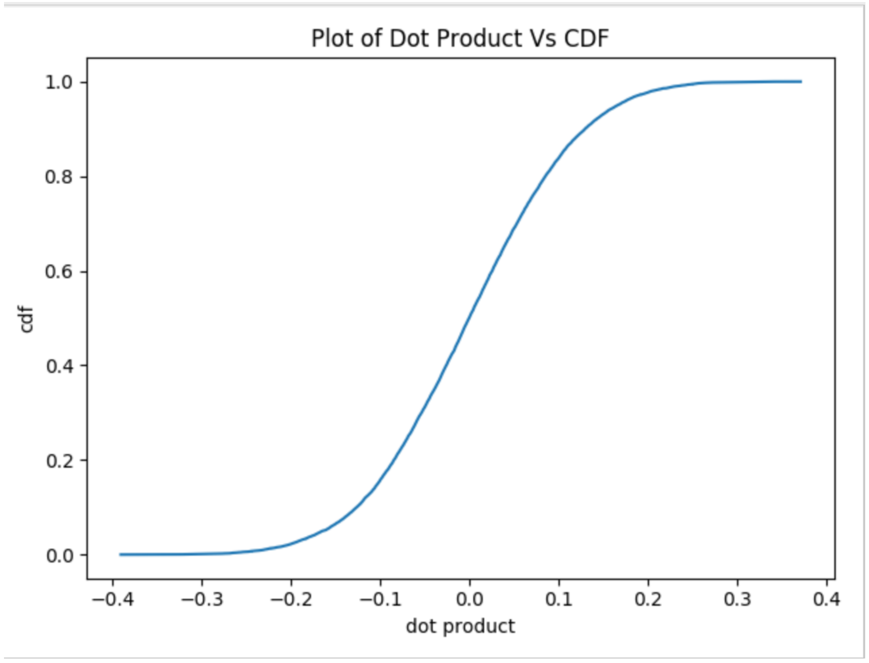

Figure 2: Question 2: Part B

## 3 Angular Hashed Approximation (35 points)

Consider the $n = 500$ data points in $R^d$ for $d = 100$ in data set $R$, given at the top. We will use the angular similarity, between two vectors $a, b \in R^d$:

$$s_{\text{ang}}(a, b) = 1 - \frac{1}{\pi} \arccos(\langle \bar{a}, \bar{b} \rangle)$$

If $a, b$ are not unit vectors (e.g., in $S^{d-1}$), then we convert them to $\bar{a} = a/\|a\|_2$ and $\bar{b} = b/\|b\|_2$. The definition of $s_{\text{ang}}(a, b)$ assumes that the input are unit vectors, and it takes a value between 0 and 1, with as usual 1 meaning most similar.

**A: (15 points)** Compute all pairs of dot products *(Yes, compute $\binom{n}{2}$ values)*, and plot a cdf of their angular similarities. Report the number with angular similarity more than $\tau = 0.85$.

Number of combinations with angular similarity above 0.85 are 39283. Please refer Figure 3 Part A.

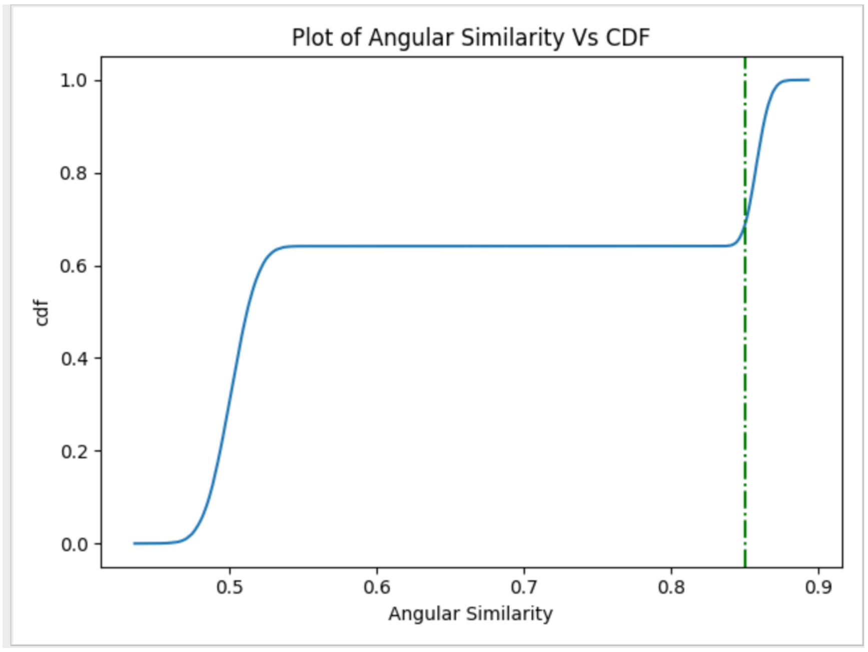

Figure 3: Question 3: Part A

**B: (20 points)** Now compute the dot products and angular similarities among $\binom{t}{2}$ pairs of the $t$ random unit vectors from Q2.B. Again plot the cdf, and report the number with angular similarity above $\tau = 0.85$.
Number of combinations with angular similarity above 0.85 is 0. Please Refer Figure 4: Question 3: Part B

## 4 Bonus (3 points)

Implement the banding scheme with your choice of $r, b$, using your $t = 160$ random vectors, to estimate the pairs with similarity above $\tau = 0.85$ in the data set $R$. Report the fraction found above $\tau = 0.85$. Compare the runtime of this approach versus a brute force search.

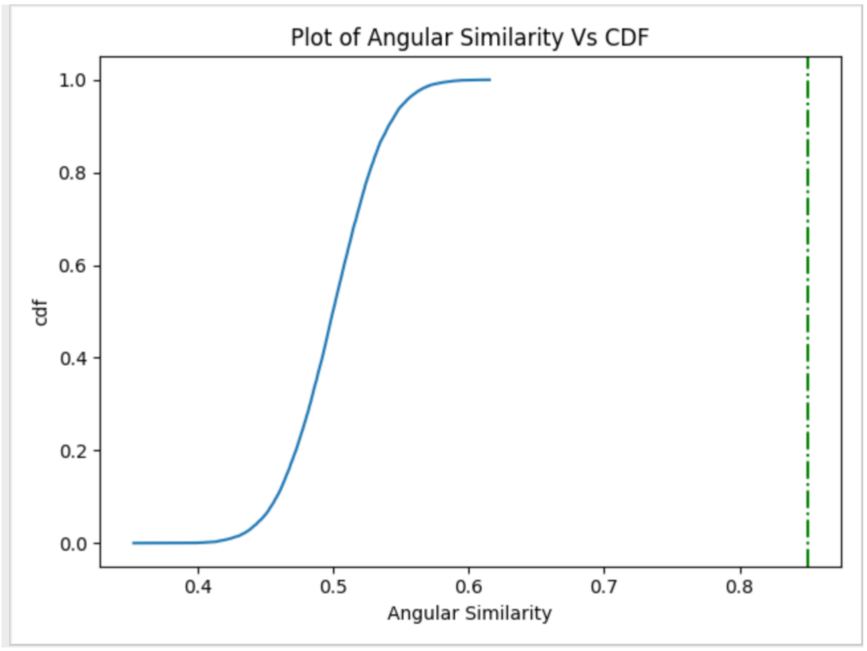

Figure 4: Question 3: Part B