Foundations of Data Science - Homework 2 Solution

Mohammad Afzal Shadab (ms82697)

mashadab@utexas.edu

Notes on the homework:

- This assignment is due by 11:59pm on Monday, February 22, 2021
- You are encouraged to discuss the homework problems with each other, but solutions and code must be written by yourself. Copying homework solutions from another student, webpage, or textbook is not allowed.
- For the coding problems, please include your code as well as the requested plotted output. A screenshot of the code, uploaded as an image, will suffice.
- You are required to typeset your solutions and submit them through Gradescope. You may find the .tex file for this homework under Files in Canvas, and type your answers to the questions directly there. Also under Files in Canvas, you will find instructions on how to submit homework through Gradescope.

Exercises adapted from textbook (Ch. 1, *High-dimensional space*)

1. **Show that for any $c \geq 1$, there exist distributions for which Chebyshev's inequality is tight, that is, Prob$(|X - \mathbb{E}[X]| \geq c) = \frac{\mathbf{Var}[X]}{c^2}$.**
   Proof by construction: Consider a random variable X with probability mass function $p_X(x)$ as follows:

$$p_X(x) = \begin{cases} p, & x = \mu - k \\ p, & x = \mu + k \\ 1 - 2p, & x = \mu \end{cases} \tag{0.1}$$

   Here, $\mathbb{E}[X] = p \cdot (\mu - k) + p \cdot (\mu + k) + (1 - 2p) \cdot \mu = \mu$
   and $Var(X) = p \cdot (\mu - k)^2 + p \cdot (\mu + k)^2 + (1 - 2p) \cdot \mu^2 = 2pk^2$.
   For making the variance equal to σ^2, setting $p = \frac{\sigma}{2k^2}$. Now, for this random variable X, we have

$$P(|X - \mu| \geq k) = 2p = \frac{\sigma^2}{k^2} \quad \square \tag{0.2}$$

2. **Consider drawing a random point x on the surface of the unit sphere in $\mathbb{R}^d$. What is the variance of x_1, the first coordinate of x? See if you can give an argument without doing integrals.**

   Answer: As the random point $\mathbf{x}$ lies on the surface of the unit sphere, we have $|\mathbf{x}| = 1$, i.e. $\sum_{i=1}^{d} x_i^2 = 1$. Now, taking the expectation of both sides, we get

$$\mathbb{E}[x_1^2] + \mathbb{E}[x_2^2] + \ldots + \mathbb{E}[x_d^2] = 1 \tag{0.3}$$

   Invoking the spherical symmetry argument, we get $\mathbb{E}[x_1] = \mathbb{E}[x_2] = \ldots = \mathbb{E}[x_d] = 0$ and $\mathbb{E}[x_1^2] = \mathbb{E}[x_2^2] = \ldots = \mathbb{E}[x_d^2]$

   Plugging it in Equation (0.3), we get

$$d\mathbb{E}[x_1^2] = 1 \tag{0.4}$$

$$\Rightarrow E[x_1^2] = \frac{1}{d} \tag{0.5}$$

   Since $\mathbb{E}(x_i) = 0 \forall i \in \{1, 2, \ldots, d\}$;
   $Var(x_1) = \mathbb{E}(x_1^2) - \mathbb{E}(x_1)^2 = \frac{1}{d} - 0 = \frac{1}{d} \quad \square$

3. **How large must ϵ be for 99% of the volume of a 1000-dimensional unit-radius ball to lie in the shell of ϵ-thickness at the surface of the ball?**
   Using an important property of any object A in $\mathbb{R}^d$ being shrunk by a small amount ϵ to produce a new object $(1 - \epsilon)A = \{(1 - \epsilon)x | x \in A\}$,

$$\text{volume}((1 - \epsilon)A) = (1 - \epsilon)^d \text{volume}(A) \tag{0.6}$$

$$\frac{\text{volume}((1 - \epsilon)A)}{\text{volume}(A)} = (1 - \epsilon)^d \tag{0.7}$$

For the given problem, the volume of the shrunk sphere is 100-99=1% of the original volume. So,

$$\frac{1}{100} = (1-\epsilon)^{1000} \tag{0.8}$$
$$\therefore \epsilon = 0.00459458 \quad \square \tag{0.9}$$

4. **Let x and y be independent d-dimensional zero-mean, unit-variance Gaussian vectors. Prove that x and y are almost orthogonal by considering their dot product.**
This proof requires direct usage of the multiple theorems, and proving lemmas given below.
Theorem 1 (**Master Tail Bound**): Let $z = z_1 + z_2 + \ldots + z_n$ where $z_1, z_2, \ldots, z_n$ are mutually independent random variables with zero mean and variance at most σ^2. Let $0 \leq a \leq \sqrt{2}n\sigma^2$. Assume that $\mathbb{E}(z_i^s) \leq \sigma^2 s!$ for $s = 3, 4, \ldots, a^2/4n\sigma^2$). Then,

$$P|z| \geq a \leq 3e^{-a^2/(12n\sigma^2)}$$

Theorem 2 (**Gaussian Annulus Theorem**): For a $d-$dimensional spherical Gaussian with unit variance in each direction, for any $\beta \leq \sqrt{d}$, all but at most $3e^{c\beta^2}$ of the probability mass lies within the annulus $\sqrt{d} - \beta \leq |\mathbf{x}| \leq \sqrt{d} + \beta$, where c is a positive constant.
Definition 1: Double exclamation mark !! symbol is used to denote for any positive integer n:

$$n!! = \begin{cases} n(n-2)(n-4)\ldots(3)(1), & \text{if } n \text{ is odd} \\ n(n-2)(n-4)\ldots(4)(2), & \text{if } n \text{ is even} \end{cases} \tag{0.10}$$

Lemma 1: The higher moments of zero-mean, unit-variance Gaussian are given by:

$$\mathbb{E}[(X-\mu)^n] = \mathbb{E}[(X)^n] = \begin{cases} 0 \text{ if } n \text{ is odd} \\ (n-1)!! \text{ if } n \text{ is even} \end{cases} \tag{0.11}$$

Proof:

$$\mathbb{E}[(X-\mu)^n] = \mathbb{E}[X^n] \tag{0.12}$$
$$= \int_{-\infty}^{+\infty} X^n \frac{1}{\sqrt{2\pi}} e^{-\frac{X^2}{2}} dX \tag{0.13}$$

For odd n, the RHS of the above equation is an odd function over symmetrical domain resulting in zero whereas for the even function let's take

$n = 2k$. Therefore,

$$\begin{aligned}
\mathbb{E}[(X-\mu)^{2k}] &= \mathbb{E}[X^{2k}] && (0.14)\\
&= \int_{-\infty}^{+\infty} X^{2k} \frac{1}{\sqrt{2\pi}} e^{-\frac{X^2}{2}} dX && (0.15)\\
&= 2\int_{0}^{+\infty} X^{2k} \frac{1}{\sqrt{2\pi}} e^{-\frac{X^2}{2}} dX && (0.16)\\
&= 2^k \int_{0}^{+\infty} \frac{1}{\sqrt{\pi}} Y^{k-1/2} e^{-Y} dY \text{ (substituting } Y = \sqrt{2X}) \\
&= \frac{2^k}{\pi}\Gamma\left(k+\frac{1}{2}\right) && (0.17)\\
&= (2k-1)!! = (n-1)!! && (0.18)
\end{aligned}$$

Lemma 2: Let $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ and $\mathbf{y} = (y_1, y_2, \ldots, y_d)^T$ be independent $d-$dimensional zero-mean, unit-variance Gaussian vectors. Then, for

$$\mathbb{E}[x_i^s y_i^s] \leq s! \tag{0.19}$$

Proof: Using the AM-GM inequality for any real numbers $a, b \in \mathbb{R}$

$$ab \leq \frac{a^2+b^2}{2}$$

we can infer that,

$$\begin{aligned}
x_i^s y_i^s &\leq \frac{x_i^{2s} + y_i^{2s}}{2} && (0.20)\\
&\text{Taking expectations} \\
\mathbb{E}[x_i^s y_i^s] &\leq \mathbb{E}\left[\frac{x_i^{2s} + y_i^{2s}}{2}\right] = \frac{\mathbb{E}[x_i^{2s}] + \mathbb{E}[y_i^{2s}]}{2} && (0.21)\\
&\text{Using Lemma 1 for RHS (even), we get} \\
\therefore \mathbb{E}[x_i^s y_i^s] &\leq (s-1)!! \leq s! && (0.22)
\end{aligned}$$

Lemma 3: For $\mathbf{x}$ and $\mathbf{y}$ defined in Lemma 2, $\mathbb{E}[x_i y_i] = 0$, $Var(x_i y_i) = 1$.
Proof: From the independence of (x_i, y_i), $\mathbb{E}[x_i y_i] = \mathbb{E}[x_i]\mathbb{E}[y_i] = 0$.
Also, $Var(x_i y_i) = \mathbb{E}[(x_i y_i - \mathbb{E}[x_i y_i])^2] = \mathbb{E}[(x_i y_i)^2] = \mathbb{E}[x_i^2]\mathbb{E}[y_i^2]$

Lemma 3: For $\mathbf{x}$ and $\mathbf{y}$ defined in Lemma 2, $P[|\mathbf{x}\cdot\mathbf{y}| \geq a] \leq 3e^{-\frac{a^2}{12d}}$.
Proof: We will use Theorem 1 (Master Tail Bound) by substituting $\mathbf{z} = \mathbf{x}\cdot\mathbf{y}$ and $n = d$. From Lemma 2, we know that $\mathbb{E}[x_i^s y_i^s] \leq (s-1)!! \leq s!$ and from Lemma 3, we know that $Var(x_i y_i) = Var(z_i) = 1$. Since the conditions for using Theorem 1 (Master Tail Bound) are satisfied,

$$P[|\mathbf{x}\cdot\mathbf{y}| \geq a] \leq 3e^{-\frac{a^2}{12d}} \tag{0.23}$$

Lemma 5: $P[|\mathbf{x} - \sqrt{d}| \geq \beta] \leq 3e^{-c\beta^2}$, c is a positive constant.
Proof: Direct consequence of Theorem 2 (Gaussian Annulus Theorem).

Proof of Question 4:
Using the basic definition of dot product

$$\begin{aligned} \mathbf{x}\cdot\mathbf{y} &= |\mathbf{x}||\mathbf{y}|\cos\phi && (0.24)\\ \Rightarrow |\cos\phi| &= \frac{|\mathbf{x}\cdot\mathbf{y}|}{|\mathbf{x}||\mathbf{y}|} && (0.25)\end{aligned}$$

Now, we want to check the probability $P\left[\frac{a}{(\sqrt{d}+\beta)^2} \leq |\cos\phi| \leq \frac{a}{(\sqrt{d}-\beta)^2}\right]$. Among many possible ways of occurrence of this event, one way is the union of these events: $|\mathbf{x}-\sqrt{d}| \leq \beta, |\mathbf{y}-\sqrt{d}| \leq \beta, |\mathbf{x}\cdot\mathbf{y}| \leq a$. Now, taking a union bound over these events using Lemmas 4 and 5, we get

$$\begin{aligned} P\left[\frac{a}{(\sqrt{d}+\beta)^2} \leq |\cos\phi| \leq \frac{a}{(\sqrt{d}-\beta)^2}\right] &\geq P[|\mathbf{x}-\sqrt{d}| \leq \beta]P[|\mathbf{y}-\sqrt{d}| \leq \beta]P[|\mathbf{x}\cdot\mathbf{y}| \leq a] \\ &\geq (1-3e^{-c\beta^2})^2(1-3e^{-a^2/12d}) \qquad (0.26)\end{aligned}$$

Also, we know that

$$P\left(0 \leq |\cos\phi| \leq \frac{a}{(\sqrt{d}-\beta)^2}\right) \geq P\left[\frac{a}{(\sqrt{d}+\beta)^2} \leq |\cos\phi| \leq \frac{a}{(\sqrt{d}-\beta)^2}\right] \qquad (0.27)$$

Combining inequalities (0.26) and (0.27)

$$P\left(0 \leq |\cos\phi| \leq \frac{a}{(\sqrt{d}-\beta)^2}\right) \geq (1-3e^{-c\beta^2})^2(1-3e^{-a^2/12d}) \qquad (0.28)$$

To understand the complete picture, let's pick the typical values of the constants as given in the textbook, also implying that $\mathbf{x} = \mathbf{y} = |\mathbf{x}\cdot\mathbf{y}| = \mathrm{O}(\sqrt{d})$. Let, $\beta = \sqrt{d}/2, c = 96$, and $a = 10\sqrt{d}$. Then, we get

$$P\left(0 \leq |\cos\phi| \leq \frac{40}{\sqrt{d}}\right) \geq (1-3e^{-d/384})^2(1-3e^{-100/12})$$

Now, let d be large

$$\begin{aligned} P[|\cos\phi| \approx 0] &\geq 1\times(1-0.000721) && (0.29)\\ &\approx 0.999279 && (0.30)\end{aligned}$$

For large d, it is easy to observe the validity of the bounds. Also, we found that the probability of $\mathbf{x}$ and $\mathbf{y}$ being almost orthogonal ($|\cos\phi| \approx 0$) is very high (≈ 1). □

5. (a) **Write a computer program that generates points uniformly distributed in the $d-$dimensional unit ball.**
The best way to achieve it is to think in polar coordinates:
   - Picking a random point on the surface of the unit ball with uniform distribution: *makes all direction equally likely*
   - Picking a radius where the likelihood of a radius corresponds to the surface area of a ball with that radius in d dimensions: *makes all points on the surface of ball within the unit ball equally likely.*

Output:

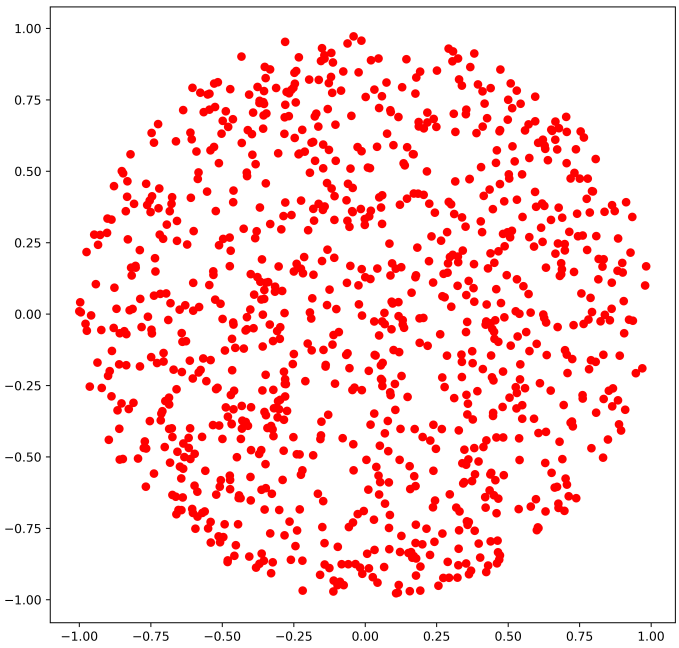

Figure 1: Unit ball A in 2 dimensions

The program is enclosed below:
Code:

```
#High dimensional unit ball
from numpy import random, linalg, zeros_like
import matplotlib.pyplot as plt

# Generate "num_points" random points in "dimension" that
                                    have uniform
# probability over the unit ball scaled by "radius" (
                                    length of points
# are in range [0, "radius"]).
def random_ball(num_points, dimension, radius=1):
    # First generate random directions by normalizing the
                                        length of a
    # vector of random-normal values (these distribute
                                        evenly on ball).
    random_directions = random.normal(size=(dimension,
                                        num_points))
    random_directions /= linalg.norm(random_directions,
                                        axis=0)
    # Second generate a random radius with probability
                                        proportional to
    # the surface area of a ball with a given radius.
    random_radii = random.random(num_points) ** (1/
                                        dimension)
    # Return the list of random (direction & length)
                                        points.
```

```
    return radius * (random_directions * random_radii).T

d     = 2        #dimensions
N     = 1000     #number of random points
A     = random_ball(N, d, ) #generating the random ball

plt.figure(figsize=(15,7.5) , dpi=100)
plt.scatter(A[:,0],A[:,1],color='red')
plt.axis('scaled')
plt.tight_layout()
plt.savefig(f'random_ball_N{N}_dimension{d}.png',
                                    bbox_inches='tight', dpi
                                     = 600)
```

(b) **Extend the computer program to generate points as follows: For a unit ball A centered at the origin and a unit ball B whose center is at $(t, 0, 0, \ldots, 0)$, draw random points from a mixture distribution: with probability 1/2, draw at random uniformly from A, with probability 1/2, draw at random uniformly from B.**
The program is enclosed together with part (c). The output for 2D case with $t = 0.5$ is given in the picture below.
Output:

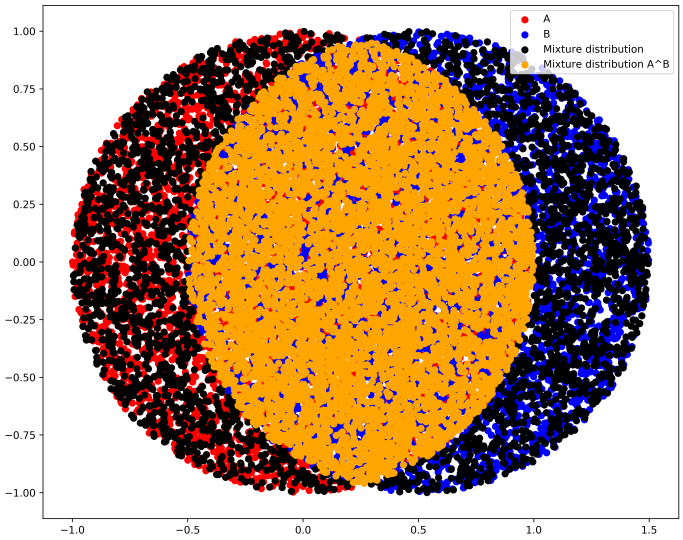

Figure 2: Unit balls A and B in 2 dimensions with $t = 0.5$

(c) **For each $d = 2, 3, 10, 100$, draw 10,000 points from the above mixture distribution and report the number of points which fall in $A \cap B$.**

Output:
d: 2 , number of intersection drawn points: 6901
d: 3 , number of intersection drawn points: 6320

d: 10 , number of intersection drawn points: 4135
d: 100 , number of intersection drawn points: 117
Code:

```
#High dimensional unit ball
from numpy import random, linalg, zeros_like, array,
                                    append, shape
import matplotlib.pyplot as plt

# Generate "num_points" random points in "dimension" that
                                    have uniform
# probability over the unit ball scaled by "radius" (
                                    length of points
# are in range [0, "radius"]).
def random_ball(num_points, dimension, radius=1):
    # First generate random directions by normalizing the
                                        length of a
    # vector of random-normal values (these distribute
                                        evenly on ball).
    random_directions = random.normal(size=(dimension,
                                        num_points))
    random_directions /= linalg.norm(random_directions,
                                        axis=0)
    # Second generate a random radius with probability
                                        proportional to
    # the surface area of a ball with a given radius.
    random_radii = random.random(num_points) ** (1/
                                        dimension)
    # Return the list of random (direction & length)
                                        points.
    return radius * (random_directions * random_radii).T

# Draw random points from a mixture distribution: with
# probability 1/2, draw at random uniformly from A, with
                                    probability
# 1/2, draw at random uniformly from B.
def draw_random_points(matA, matB, num_draw_points, probA,
                                     probB):
    mixture_dist = zeros_like(matA)
    for i in range(0,num_draw_points):
        p = random.uniform(0, 1,num_draw_points)
                                                    #
                                        generate random
                                        uniform
                                        probability
        #getting mixture distribution values
        mixture_dist[p<=probA,:]        = matA[p<=probA,:]
        mixture_dist[p>(1.0-probB),:]   = matB[p>(1.0-probB
                                        ),:]


    #find how many drawn points reside in A^B
    mixture_dist_intersect = []
    for i in range(0, num_draw_points):
        dist_from_A_center      = linalg.norm(mixture_dist
                                        [i,:])
        dist_from_B_center      = (mixture_dist[i,:]).copy
                                        ()
        dist_from_B_center[0]   = dist_from_B_center[0]-t
        dist_from_B_center      = linalg.norm(
                                        dist_from_B_center
```

```
                                        )

        if dist_from_A_center <= 1.0 and
                                        dist_from_B_center
                                         <= 1.0:
                mixture_dist_intersect.append(mixture_dist
                                                [i,:])
    mixture_dist_intersect = array(mixture_dist_intersect)
    return mixture_dist,mixture_dist_intersect

d     = 2       #dimensions
N     = 10000   #number of random points
A     = random_ball(N, d, ) #generating the random ball
t     = 0.5                 #offset in the direction of
                                first coordinate from
                                center
B     = random_ball(N, d, ) #generating the random ball at
                                 the origin
B[:,0]= B[:,0] + t          #offsetting it in the first
                                coordinate direction by
                                t
N_draw= 10000
probA = 0.5
probB = 0.5

mixture_dist, mixture_dist_intersect = draw_random_points(
                                A, B, N_draw, probA,
                                probB)
plt.figure(figsize=(15,7.5) , dpi=100)
plt.scatter(A[:,0],A[:,1],color='red',label = 'A')
plt.scatter(B[:,0],B[:,1],color='blue',label = 'B')
plt.scatter(mixture_dist[:,0],mixture_dist[:,1],color='
                                black',label = 'Mixture
                                distribution')
plt.scatter(mixture_dist_intersect[:,0],
                                mixture_dist_intersect[:
                                ,1],color='orange',label
                                 = 'Mixture distribution
                                 A^B')
legend = plt.legend(loc='best', shadow=False, fontsize='
                                medium')
plt.axis('scaled')
plt.tight_layout()
plt.savefig(f'random_ball_intersect_N{N}_dimension{d}.png'
                                ,bbox_inches='tight',
                                dpi = 600)

# Part (c)
d     = [2,3,10,100]        #dimensions
num_intersect_points = zeros_like(d)
for i in range(0,len(d)):
    N     = 10000   #number of random points
    A     = random_ball(N, d[i], ) #generating the random
                                    ball
    t     = 0.5                 #offset in the direction
                                    of first coordinate
                                    from center
    B     = random_ball(N, d[i], ) #generating the random
                                    ball at the origin
    B[:,0]= B[:,0] + t          #offsetting it in the
                                    first coordinate
                                    direction by t
```

```
N_draw= 10000
probA = 0.5
probB = 0.5

mixture_dist, mixture_dist_intersect =
                                    draw_random_points(A
                                    , B, N_draw, probA,
                                    probB)

num_intersect_points[i], dummy = shape(
                                    mixture_dist_intersect
                                    )
print('d:',d[i],', number of intersection drawn points
                                    : ',
                                    num_intersect_points
                                    [i])
```

(d) **Let x be drawn from the mixture distribution from part (b). Prove that for any $\epsilon > 0$, there exists c such that if $t \geq c/\sqrt{d-1}$, Prob[x $\in A \cap B$] $\leq \epsilon$. In other words, this amount of separation means that nearly all of the mixture distribution is identifiable.**

We can modify and restate the Theorem 2.7 in the textbook as follows:
Theorem 2.7 (modified): For $c \geq 1$ and $d \geq 1$, at most $\frac{2}{c}\exp(c^2/2)$ fraction of the volume of the d-dimensional unit ball has $|x_1| \geq \frac{c}{\sqrt{d-1}}$.

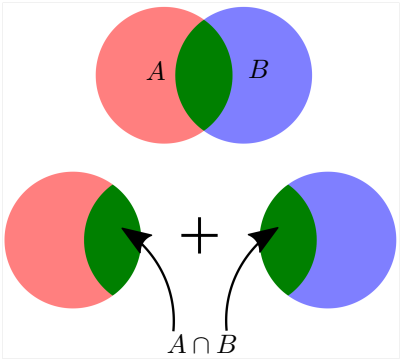

Figure 3: Universal set U consisting of $A + B$

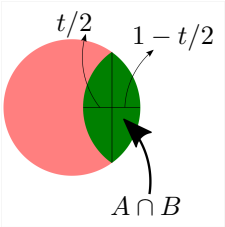

Figure 4: Venn diagram of $A \cap B$ showing t

We can see that the universal set U for this case is:

$$\text{volume}(U) = \text{volume}(A) + \text{volume}(B) = 2V(d)$$

where $V(d)$ is the volume of the ball as a function of dimension d. As $A \cap B$ is repeated twice in $A + B$ as shown in Figure 3, it is considered twice in the analysis. Also, $A \cap B$ is symmetric about hyperplane at $x_1 = 1 - t/2$, which can been from Figure 4.

$$\begin{aligned} \text{volume}(A \cap B) &= 2 \times \text{volume}\left[|x_1| \geq \frac{t}{2}\right] && (0.31) \\ P[\mathbf{x} \in A \cap B] &= 2 \times \frac{\text{volume}(A \cap B)}{\text{volume}(U)} && (0.32) \\ &= 2\left[\text{fraction of unit ball with } |x_1| \geq \frac{t}{2}\right] && (0.33) \end{aligned}$$

As we want $P[x \in A \cap B] \leq \epsilon$, equation (0.33) then implies the following:

$$\left[\text{fraction of unit ball with } |x_1| \geq \frac{t}{2}\right] \leq \frac{\epsilon}{2} \tag{0.34}$$

From Theorem 2.7 (modified), we know that

$$\left[\text{fraction of unit ball with } |x_1| \geq \frac{c^*}{\sqrt{d-1}}\right] \leq \frac{1}{c^*}\exp(c^{*2}/2), c^* \geq 1, d \geq 3 \tag{0.35}$$

$$\begin{aligned} \text{Let } c^* &= c/2 \\ \left[\text{fraction of unit ball with } |x_1| \geq \frac{c}{2\sqrt{d-1}}\right] &\leq \frac{2}{c}\exp(c^2/8), c \geq 2, d \geq 3 \end{aligned} \tag{0.36}$$

To satisfy equations (0.34) and (0.36) simultaneously, we want a value for c for a given ϵ satisfying,

$$\frac{4}{c}\exp(-c^2/8) \leq \frac{\epsilon}{2}, c \geq 2, \epsilon \in [0,1] \tag{0.37}$$

For $c \geq 2$, LHS of equation (0.37) is always decreasing with maximum $\frac{2}{\sqrt{e}} < 1$ at $c = 2$ and exponentially decreases to 0 as $c \to \infty$. Therefore, equation (0.36) is easy to satisfy for any $\epsilon \in [0,1]$. Therefore, for any $\epsilon > 0$, there exists c such that if $t \geq c/\sqrt{d-1}, P[\mathbf{x} \in A \cap B] \leq \epsilon$. □

6. **If one generates points in d-dimensions with each coordinate a unit variance Gaussian, the points will approximately lie on the surface of a sphere of radius $\sqrt{d}$. What is the distribution when the points are projected onto the line through the origin in the direction of $(1, 0, 0, 0, \ldots, 0)$? When projected onto an arbitrary line through the origin?**

   Projecting any point $\mathbf{x} = \{x_1, x_2, \ldots, x_d\} \in \mathbb{R}^d$ onto the line through the origin in the direction of first canonical basis vector $e_1 = (1, 0, \cdots, 0)^T$ gives x_1. So, the resulting distribution is a Gaussian with zero mean and unit variance.
   When $\mathbf{x}$ is projected onto an arbitrary line through origin $\mathbf{y} = (y_1, y_2, \ldots, y_d)$. The projection of point $\mathbf{x}$ on this line is given by the dot product $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{d} x_i y_i$. Since this projection is a linear combination of the Gaussian variables, it is a Gaussian variable. It's mean and variance are given below:

$$\mathbb{E}(\mathbf{x} \cdot \mathbf{y}) = \sum_{i=1}^{d} \mathbb{E}[x_i] y_i = 0 \tag{0.38}$$

$$Var(\mathbf{x}.\mathbf{y}) = \sum_{i=1}^{d} \mathbb{E}[x_i^2] y_i^2 = \sum_{i=1}^{d} 1 \cdot y_i^2 = 1 \tag{0.39}$$

$$(\because \text{independence}, \mathbb{E}[x_i^2] = 1 \ \& \ \mathbf{y} \text{ normalized}) \tag{0.40}$$

   Therefore, the projection of arbitrary line passing through origin is again a unit variance Gaussian with zero mean. □

7. **In d dimensions there are exactly d vectors that have unit length and are pairwise orthogonal. However, if you wanted a set of vectors that were almost orthogonal you might squeeze in a few more. For example, in 2 dimensions if almost orthogonal meant at least 45 degrees apart, you could fit in three almost orthogonal vectors. Suppose you wanted to find 1000 almost orthogonal vectors in 100 dimensions. Here are two ways you could do it:**

   (a) Begin with 1000 orthonormal 1000-dimensional vectors, and then project them to a random 100-dimensional space

   (b) Generate 1000 random Gaussian vectors in 100 dimensions.

   Implement both ideas and compare them to see which does a better job.
   Both perform an equivalently accurate job, as the standard deviation ~ 0.1 in both cases. Also part (a) takes 17.6958 seconds while part (b) takes 16.8191 seconds.
   Output:
   The true value is $\pi/2 = 1.5707963267948966$
   Part (a): The value with random projection is $= 1.5709380061583484 \pm 0.10056616783159755$
   Part (b): The value with Gaussian vectors is $= 1.5706540028171097 \pm 0.10060095318670761$
   Running part (a) took 17.695838999999978 seconds
   Running part (b) took 16.819081000000097 seconds

Code:

```
#High dimensional unit ball
import numpy as np
import matplotlib.pyplot as plt
from time import perf_counter


# To get a random matrix of num_vectors in dimension
# Input : dimensions and number of vectors
# Output: num_vectors X dimension unit Gaussian random matrix
def random_Gaussian_vectors(dimension, num_vectors):
    return np.random.normal(0,1,(dimension,num_vectors))

# This function stores the projection of vector mat2 from R^{
                                    dimensions}
# to R^{reduced_num_vectors}
# Input :mat1 (dim X reduced_num_vectors) helps with the projection
#        mat2 (dim X num_vectors) is being projected onto R^{
                                    reduced_num_vectors}
# Output: projection_matrix (reduced_num_vectors X num_vectors)
def projection_matrix(mat1,mat2):
    dimension, num_vectors = np.shape(mat1)
    reduced_num_vectors    = np.shape(mat2)[1]

    projection_matrix = np.empty((num_vectors,reduced_num_vectors))

    for i in range(0,num_vectors):
        for j in range(0,reduced_num_vectors):
            projection_matrix[i,j] = np.dot(mat1[:,i], mat2[:,j]) #
                                            projection of columns
    return projection_matrix

# This function computes the dot product all possible pair of n
                                    vectors
# Input : mat (reduced_num_vectors X num_vectors)
# Output: a matrix of angles with columns (i,j,angles)
def comp_angle_matrix(mat):
    angle_matrix  = []
    num_vectors   = np.shape(mat)[1]

    for i in range(0,num_vectors):
        for j in range(0,num_vectors):
            angle = np.arccos(np.dot(mat[:,i], mat[:,j])/(np.linalg
                                            .norm(mat[:,i])*np.
                                            linalg.norm(mat[:,j])
                                            )) #calculating the
                                            angles
            if(i<j):
                angle_matrix.append([i+1,j+1,angle])
    return np.array(angle_matrix)

#Part (a)
parta_start = perf_counter()
A  =  random_Gaussian_vectors(1000,100)  #random 1000-dimensional
                                    space
B  =  random_Gaussian_vectors(1000,1000)
B,dummy = np.linalg.qr(B)                #finding 1000 orthonormal
                                    1000-dimensional vectors using QR

projMat = projection_matrix(A,B)
angleMat= comp_angle_matrix(projMat)
```

```
mean = np.mean(angleMat[:,2]); std = np.std(angleMat[:,2])
parta_stop = perf_counter()

print('The true value is \u03C0/2 = ', np.pi/2)
print(f'Part (a): The value with random projection is = {mean} \
                                    u00B1 {std}')

#Part (b)
partb_start = perf_counter()
A  =  random_Gaussian_vectors(100,1000)  #random 1000-dimensional
                                    space
angleMat= comp_angle_matrix(A)
mean = np.mean(angleMat[:,2]); std = np.std(angleMat[:,2])
partb_stop = perf_counter()
print(f'Part (b): The value with Gaussian vectors is = {mean} \
                                    u00B1 {std}')
print('\n Run times \n')
print(f'Running part (a) took {parta_stop - parta_start} seconds \n
                                    ')
print(f'Running part (b) took {partb_stop - partb_start} seconds')
```