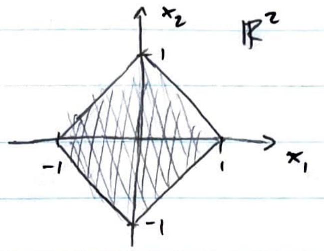
Robert Heeter
ELEC 378
1/20/23

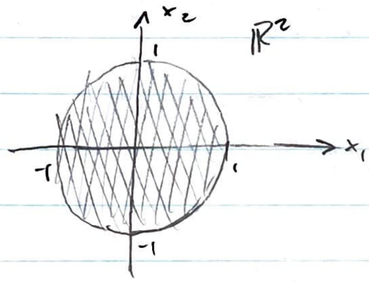
Problem Set #1

1. $\|\underline{x}\|_p = \left(\sum_{i=1}^{d} |x_i|^p\right)^{1/p}$

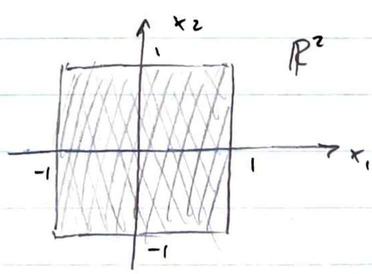
a. In $\mathbb{R}^2$: $p=1$ gives $\|\underline{x}\|_1 = |x_1| + |x_2| \rightarrow \ell_1$ ball defined as $|x_1| + |x_2| \leq 1$ as shown



$p=2$ gives $\|\underline{x}\|_2 = (x_1^2 + x_2^2)^{1/2} \rightarrow \ell_2$ ball defined as $x_1^2 + x_2^2 \leq 1^2$, or a circle with radius 1

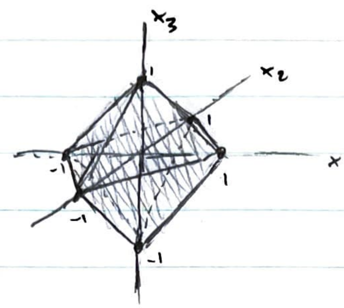


$p=\infty$ gives $\|\underline{x}\|_\infty = \max(|x_1|, |x_2|) \rightarrow \ell_\infty$ ball defined as $|x_1| = 1$ with $|x_2| \leq 1$, or $|x_2| = 1$ and $|x_1| \leq 1$, or a square as shown

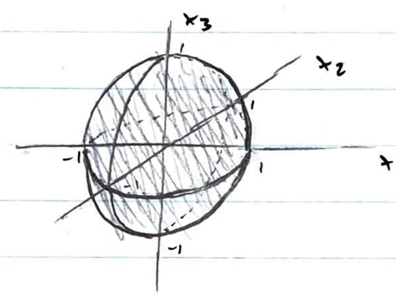


In $\mathbb{R}^3$: Next page

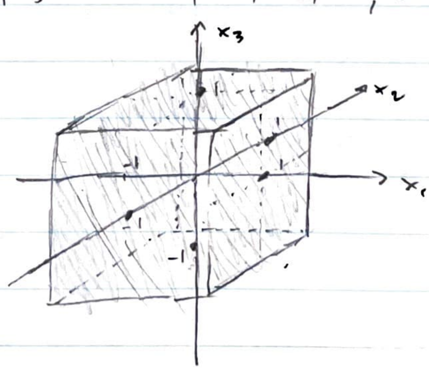
(1). (a). In $\mathbb{R}^3$: $p=1$ gives $\|\underline{x}\|_1 = |x_1| + |x_2| + |x_3| \rightarrow \ell_1$ ball defined as $|x_1| + |x_2| + |x_3| \le 1$ as shown



$p=2$ gives $\|\underline{x}\|_2 = (x_1^2 + x_2^2 + x_3^2)^{1/2} \rightarrow l_2$ ball defined as $x_1^2 + x_2^2 + x_3^2 \le 1^2$, or a sphere with radius 1



$p=3$ gives $\|\underline{x}\|_\infty = \max(|x_1|, |x_2|, |x_3|) \rightarrow l_\infty$ ball defined as $|x_1| = 1$ with $|x_2|, |x_3| \le 1$, $|x_2| = 1$ with $|x_1|, |x_3| \le 1$, or $|x_3| = 1$ with $|x_1|, |x_2| \le 1$, or a cube as shown

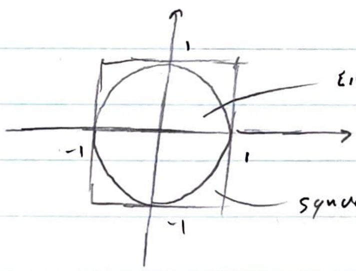


b. Next page

(1). b. See code for simulation and graph.

The simulation estimates the volume between the l_2 and l_∞ balls for increasing dimensionality by producing a large set of random vectors that lie within the l_∞ ball (i.e. each component in the vector takes a random value between -1 and +1) for each dimension increment, and then calculating the number of vectors in the l_∞ ball set that lie outside of the l_2 ball (i.e. have 2-norm >1). To better estimate the changes in volume, this number of vectors is divided by the total number of vectors, resulting in a ratio of the number of vectors in the l_∞ ball but outside the l_2 ball to the total number of vectors.

As the dimensionality grows large (i.e >10 dimensions), the vast majority (almost 100%.) of the volume in the l_∞ ball lies outside of the l_2 ball. This result seems a bit unintuitive compared to the $\mathbb{R}^2$ case, where l_2 is a unit circle and l_∞ is a square with side length 2 (-1 to +1) centered at the origin. In this case, the ratio of space in l_∞ outside of l_2 is $\approx 21.5\%$ of the total l_∞ space:



circle area $= \pi r^2 = \pi(1)^2 = 3.14 \text{ un}^2$

square area $= 2^2 = 4 \text{ un}^2$

$\rightarrow 1 - \frac{3.14}{4} \approx 21.5\%$

As the dimensionality grows, this percentage becomes very small, meaning the concentration of volume of a hypercube in high dimensions primarily resides outside of the l_2 ball. There are not the same number of vectors living in each region of the hypercube, and the ratio of the number of vectors in the l_2 ball to the number of vectors in the l_∞ hypercube changes with the dimension of the hypercube.

2. Next page

2. p-dimensional hypercube $[0,1]^p$ → each side is length 1

a. $p=1$ (unit interval)

line is 1 un long

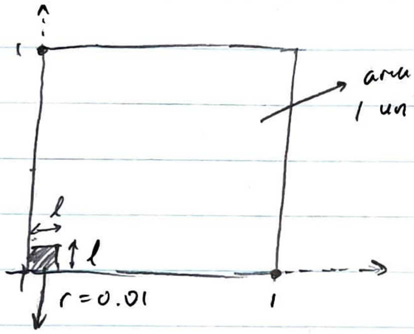
segment is 0.01 units long
(i.e. 1% of 1)

$\boxed{l = r = 0.01}$

The expected length of the interval that captures r of the training data in $p=1$ is $r \times 1$ units long.

$\boxed{l = r}$

b. $p=2$ (unit square)



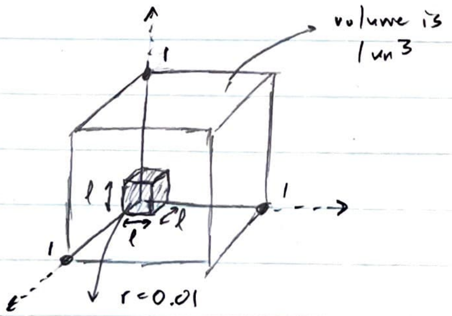
square is 0.1 × 0.1 units big
(i.e. square area = $0.1^2 = 0.01$)

$\boxed{l = r^{1/2} = 0.01^{1/2} = 0.1}$

The expected dimensions (side lengths) that capture r of the training data in $p=2$ is $r^{1/2}$ units long

$\boxed{l = r^{1/2}}$

c. $p=3$ (unit cube)



cube is $0.01^{1/3} \times 0.01^{1/3} \times 0.01^{1/3}$ units big,

or $\approx 0.215 \times 0.215 \times 0.215$ units big

(i.e. cube volume $= (0.01^{1/3})^3 = 0.01$

$\boxed{l = r^{1/3} = 0.01^{1/3} \approx 0.215}$

The expected dimensions (side lengths) that capture r of the training data in $p=3$ is $r^{1/3}$ units long

$\boxed{l = r^{1/3}}$

d. Next page

(2). d. Arbitrary p (hypercube)

For a local hypercube of "volume" r (i.e. that captures r percent of the training data), side length ℓ, and dimension p:

$\ell^p = r \rightarrow \ell = r^{1/p}$

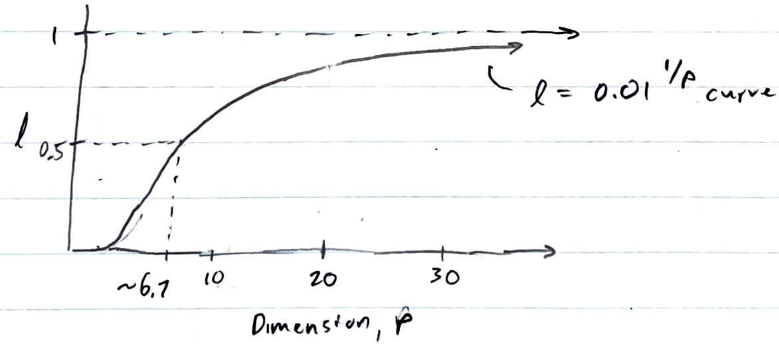
$\boxed{\text{side length } \ell = r^{1/p}}$ $\longrightarrow$ Thus, for $\begin{cases} p=1, & \ell = r^{1/1} = r \checkmark & \text{(part A)} \\ p=2, & \ell = r^{1/2} \checkmark & \text{(part B)} \\ p=3, & \ell = r^{1/3} \checkmark & \text{(part C)} \end{cases}$

$\downarrow$

Thus, when $r = 0.01$,

$\boxed{\ell = (0.01)^{1/p}}$

e. As p grows very large, the hypercube that captures a ratio r of the training data points (even for small r) is not very "local." At high dimensionality, the necessary side length of the local hypercube become very large (the side length approaches an asymptote of 1, shown below for $r = 0.01$



Consequently, the "local" hypercube at high dimensionality becomes a large space (compared to the concentrated space at low dimensions). Given a local hypercube of some fixed side length, a 1 dimension increase will not "scale" the local hypercube and unit hypercube space equally (which would be necessary to maintain the same ratio (r), or percent, of the total training data). To account for this, the size of the local hypercube (the side length) must also increase as the dimensionality increases. In other words, for a fixed local hypercube side length, an increase by 1 dimension will result in a smaller fraction of the total training data being captured. For the case where $r = 1\% = 0.01$, when p is very large, the region that captures $r = 1\%$ of the training data approaches (but never equals) the side length of the overall hypercube of all training data.

3. $\begin{cases} c_1(t) \to 1 \\ c_0(t) \to 0 \end{cases}$ Transmitter sends $x(t) + n(t) \to y(t)$ received signal

$\uparrow$ noise signal

a. The Cauchy - Schwarz inequality can be used to decode a digital signal from a corrupt BFSK signal. Given a received noisy signal $y(t)$ and known carrier signals $c_0(t)$ and $c_1(t)$, the absolute value of the inner product (dot product) between the noisy signal and each of the carrier signals can be compared for each bit length to determine which carrier signal (and corresponding bit value) the length of noisy signal represents.

A higher absolute value inner product value indicates a greater degree of similarity following the Cauchy - Schwarz inequality. A lower absolute value inner product indicates less similar:

$$\left.\begin{aligned} 0 &\le |y \cdot c_1| \le \|y\|_2 \|c_1\|_2 \\ 0 &\le |y \cdot c_0| \le \|y\|_2 \|c_0\|_2 \end{aligned}\right\}$$

compare $|y \cdot c_1|$ and $|y \cdot c_0|$ to determine if $y \to c_1 \to 1$ bit or $c_0 \to 0$ bit

$\uparrow$ lower bound when signals y and c_1 or c_0 are most different

$\uparrow$ upper bound when signals y and c_1 or c_0 are most similar.

The two inner products of a segment of y with c_0 or y with c_1 can be compared against each other to determine if such segment of y is more similar to c_0 (meaning a 0 bit) or c_1 (meaning a 1 bit).

b. See code for code and output. The system is effective in decoding the corrupt signal correctly (the output appears properly decoded). With a strong enough level of noise, it is possible for the output to be damaged as some bits are improperly decoded.

c. Next page

(3). c. The corrupt signal ("y.wav") has static and noise sounds, but some of the underlying bit tones are audible through the noise.

The decoded signal ("x.wav") does not have noise static and the two tones for the 1 and 0 bits are clearly audible compared to the noisy signal. I am satisfied with the decoder's performance.

ELEC378-HW1

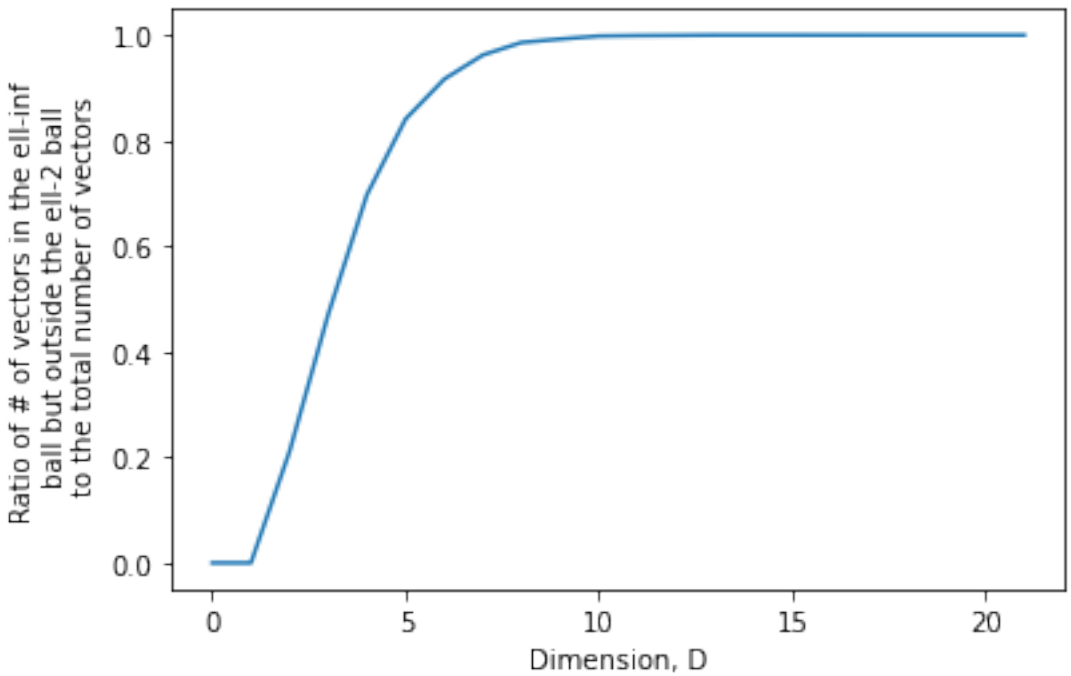
January 20, 2023

[1]:
```
# ROBERT HEETER
# ELEC 378 Machine Learning
# 20 January 2023

# PROBLEM SET 1
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
# PROBLEM 1

# max dimension to simulate
D = 22

# store the estimated volume in each dimension
volume = np.squeeze(np.zeros((1,D)))

# in d dimensions
for d in np.arange(1,D):
    # generate N random vectors uniformly distributed throughout the␣
 ↪ell-infinity ball
    # and stack them as rows in the matrix X
    N = 1000*d
    X = np.random.uniform(-1,1,(N,d))

    # estimate the volume between the ell-infinity and ell-2 ball using the␣
 ↪random samples
    # use the ratio of vectors in the ell-infinity ball that lie outside of the␣
 ↪ell-2 ball
    # (i.e. the number of vectors with 2-norm greater than 1) to the total␣
 ↪number of vectors
    # as a metric for the distribution between the ell-infinity and ell-2 balls
    volume[d] = sum((np.linalg.norm(X,ord=2,axis=1) > 1)*1)/N

# plot the dimension vs. volume between the ell-infinity and ell-2 ball
plt.plot(volume)
plt.xlabel('Dimension, D')
```

```
plt.ylabel('Ratio of # of vectors in the ell-inf\n ball but outside the ell-2␣
↪ball\n to the total number of vectors')
plt.show()
```



[4]:
```
# PROBLEM 3

from scipy.io import loadmat
from scipy.io.wavfile import write
import numpy as np
```

[5]:
```
data = loadmat('cauchy_schwarz_decoding.mat')

y = data['y']
c0 = data['c0']
c1 = data['c1']

# construct the matrix C which contains as columns the carriers c0 and c1
C = np.transpose(np.array([c0[0],c1[0]]))

# construct the matrix Y which contains as rows the received (noisy) carrier
# tones corresponding to each transmitted bit
# the width of Y should be equal to the length of one carrier tone
Y = y.reshape(-1,np.shape(C)[0])
```

```
# use matrix multiplication of C and Y to compute the sequence of inner products
# between each received (noisy) carrier tone and each known carrier tone (c0␣
↪and c1)
S = np.matmul(Y,C)

# use argmax to decode according to cauchy schwarz
# bits should have shape (N,) where N is the number of decoded bits
bits = np.argmax(np.abs(S), axis=1)
```

[6]:
```
# conversion from binary to uint8
strResult = ''.join(str(n) for n in bits)
byteResult = list(int(strResult[i : i+8][::-1], 2) for i in range(0,␣
↪len(strResult), 8))
arrayResult = np.asarray([byteResult]).astype('uint8')

# writing decoded bits as a .jpg
f = open('decoded.jpg','wb')
f.write(arrayResult)
f.close()
```

[7]:
```
# construct the matrix X which contains in its i^th column the carrier tone
# corresponding to the i^th bit transmitted
X = C[:,bits]

# construct the signal x(t) by playing the carrier tones in sequence
x = X.flatten('F')
y = y.flatten()

# listen to the noisy received signal y(t) vs your denoised reconstruction x(t)
fs = 44100
write("y.wav",fs,y)
write("x.wav",fs,x)
```

[ ]:
```

```