Introduction to Machine Learning

Homework Solution - 1

Instructor: Prof Chinmay Hegde *TA:* Devansh Bisla, Maryam Majzoubi

Problem 1
Let $\{x_1, x_2, \ldots, x_n\}$ be a set of points in d-dimensional space. Suppose we wish to produce a single point estimate $\mu \in \mathbb{R}^d$ that minimizes the squared-error:

$$\|x_1 - \mu\|_2^2 + \|x_2 - \mu\|_2^2 + \ldots + \|x_n - \mu\|_2^2$$

Find a closed form expression for μ and prove that your answer is correct.

(Solution) We want to minimize the following error:

$$Error = \sum_{i=1}^{n} (x_i - \mu)^2$$

so we take the derivative w.r.t μ and set it to zero:

$$Error = -2 * \sum_{i=1}^{n} (x_i - \mu) = 0$$

This will result in following closed form solution, which is the mean of x_i's:

$$\mu^* = \frac{\sum_{i=1}^{n} x_i}{n}$$

Problem 2

Not all norms behave the same; for instance, the ℓ_1-norm of a vector can be dramatically different from the ℓ_2-norm, especially in high dimensions. Prove the following norm inequalities for d-dimensional vectors, starting from the definitions provided in class and lecture notes. (Use any algebraic technique/result you like, as long as you cite it.)

(a) $\|x\|_2 \le \|x\|_1 \le \sqrt{d}\|x\|_2$

(b) $\|x\|_\infty \le \|x\|_2 \le \sqrt{d}\|x\|_\infty$

(c) $\|x\|_\infty \le \|x\|_1 \le d\|x\|_\infty$

(Solution)

$$||x||_1 = \sum_{i=1}^{d} |x_i|, \qquad ||x||_2 = \sqrt{\sum_{i=1}^{d} |x_i|^2}, \qquad ||x||_\infty = max\{|x_i| : i = 1, .., d\}$$

(a) For left hand side, we know that,

$$\sum_{i=1}^{d} |x_i|^2 \le \Big(\sum_{i=1}^{d} |x_i|\Big)^2 \implies ||x||_2^2 \le ||x||_1^2 \implies ||x||_2 \le ||x||_1$$

For right hand side, according to Cauchy Swartz inequality, for any two d-dimensional vectors u and v,

$$\left|\sum_{i=1}^{d} u_i v_i\right|^2 \le \sum_{j=1}^{d} |u_j|^2 \sum_{k=1}^{d} |v_k|^2$$

Now, let $u_i = |x_i|$ and $v_i = 1\ \forall i$ therefore,

$$\sum_{i=1}^{d} |x_i| \cdot 1 \le \sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} 1^2} \implies ||x||_1 \le \sqrt{d}||x||_2$$

Hence, $\|x\|_2 \le \|x\|_1 \le \sqrt{d}\|x\|_2$

(b) The left hand side is evident from the definition itself, as the max element of a vector will be less than sum of squares of all the elements i.e

$$\Big(max\{|x_i| : i = 1, .., d\}\Big)^2 \le \sum_{i=1}^{d} |x_i|^2 \implies ||x||_\infty \le ||x||_2$$

For right hand side note that, sum of the squares of the elements of a vector will always at best be equal to summing the largest element d times i.e

$$\sum_{i=1}^{d} |x_i|^2 \le \sum_{i=1}^{d} \Big(max\{|x_i| : i = 1, .., d\}\Big)^2 \implies ||x||_2^2 \le d||x||_\infty^2 \implies ||x||_2 \le \sqrt{d}||x||_\infty$$

(c) We can obtain the inequality from (a) and (b)

Problem 3
When we think of a Gaussian distribution (a bell-curve) in 1, 2, or 3 dimensions, the picture that comes to mind is a "blob" with a lot of mass near the origin and exponential decay away from the origin. However, the picture is very different in higher dimensions (and illustrates the counter-intuitive nature of high-dimensional data analysis). In short, we will show that *Gaussian distributions are like soap bubbles*: most of the mass is concentrated near a shell of a given radius, and is empty everywhere else.

(a) Fix $d = 3$ and generate 10,000 random samples from the standard multi-variate Gaussian distribution defined in $\mathbb{R}^d$.

(b) Compute and plot the histogram of Euclidean norms of your samples. Also calculate the average and standard deviation of the norms.

(c) Increase d on a coarsely spaced log scale all the way up to $d = 1000$ (say $d = 50, 100, 200, 500, 1000$), and repeat parts (a) and (b). Plot the variation of the average and the standard deviation of Euclidean norm of the samples with increasing d.

(d) What can you conclude from your plot from part (c)?

(e) **Bonus, not for grade.** Mathematically justify your conclusion using a formal proof. You are free to use any familiar laws of probability, algebra, or geometry.

(Solution)
(e)** The expected value of squared Euclidean norm of a d dimensional Gaussian is as follows:

$$E[||X||_2^2] = E[X_1^2 + \cdots + X_d^2] = E[X_1]^2 + \cdots + E[X_d]^2 = Var(X_1) + \cdots + Var(X_d) = d$$

Therefore, the expected value of Euclidean norm will converge to $\sqrt{d}$.

Thanks to Dimitrios Chaikalis for the following solution for (a - d):

HW1_Q3

February 4, 2020

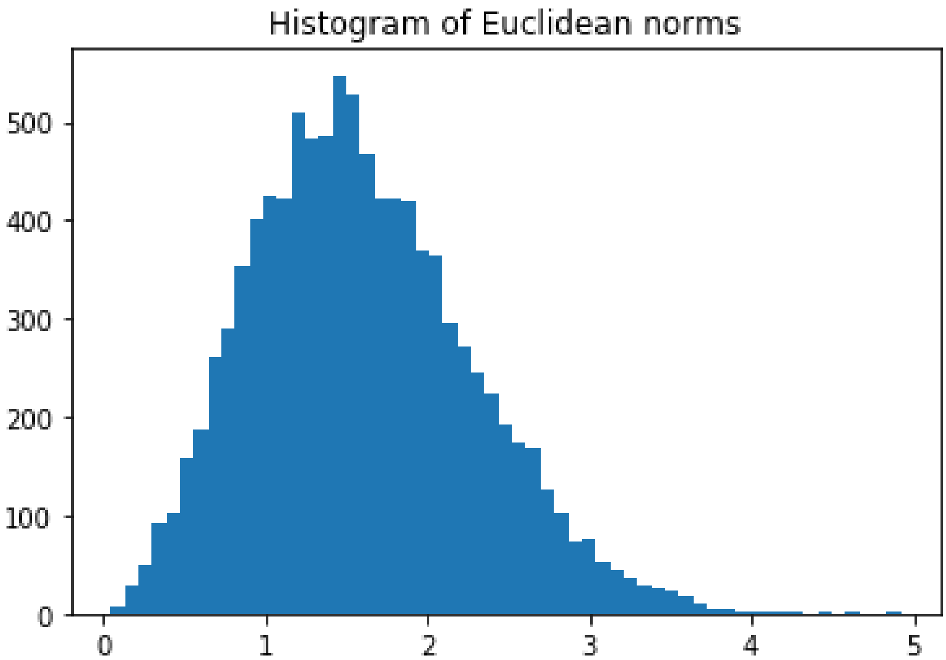
[0]:
```
import numpy as np
import matplotlib.pyplot as plt

d = 3
means = np.zeros(d)
cov = np.eye(d)
x = np.random.multivariate_normal(means,cov,10000)
```

Samples have been generated.

[0]:
```
y = np.zeros(10000)
for i in range(10000):
  y[i] = np.linalg.norm(x[:][i],2)
_ = plt.hist(y,bins='auto')
plt.title('Histogram of Euclidean norms')
```

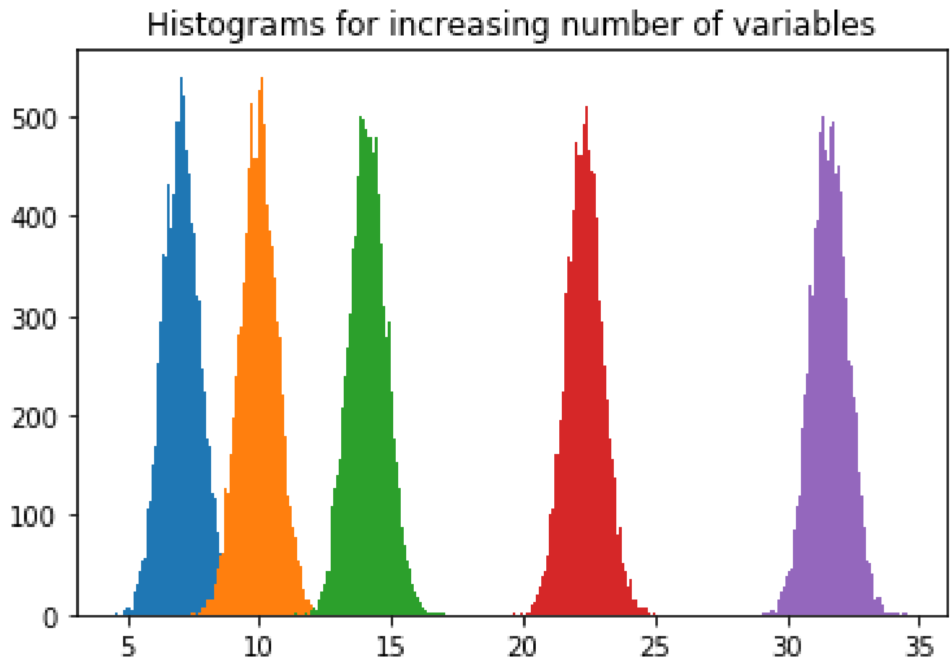
[0]: `Text(0.5, 1.0, 'Histogram of Euclidean norms')`



[0]:
```
m = np.mean(y)
std = np.std(y)
print(m,std)
```

```
1.5983079784218295 0.6739420978301629
```

[0]:
```
D = [50,100,200,500,1000]
M = np.zeros(5)
Std = np.zeros(5)
for i in range(5):
  means = np.zeros(D[i])
  cov = np.eye(D[i])
  x = np.random.multivariate_normal(means,cov,10000)
  y = np.zeros(10000)
  for j in range(10000):
    y[j] = np.linalg.norm(x[:][j],2)
  _ = plt.hist(y,bins='auto')
  M[i] = np.mean(y)
  Std[i] = np.std(y)
  plt.title('Histograms for increasing number of variables')
```



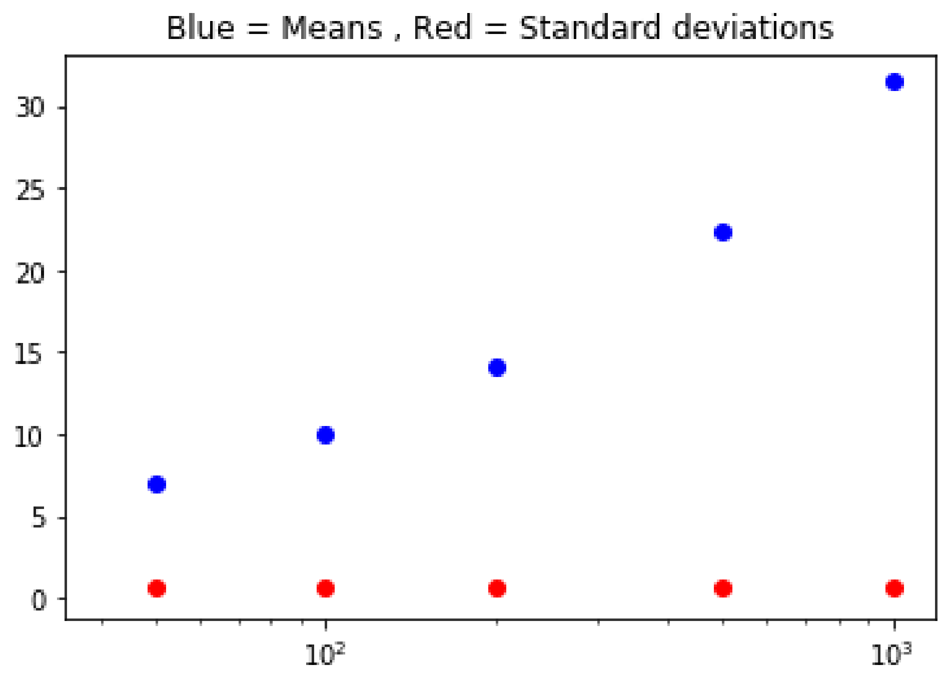
[0]:
```
print(M)
```

```
[ 7.05201213  9.97963425 14.12533543 22.34299976 31.6195839 ]
```

[0]:
```python
print(Std)
```

```
[0.70726022 0.70237244 0.71003798 0.70606778 0.71128948]
```

[0]:
```python
_ = plt.scatter(D,M,color='blue')
_ = plt.scatter(D,Std,color='red')
plt.xscale('log')
plt.title('Blue = Means , Red = Standard deviations')
plt.show()
```



We can see that, apparently, as we increase the number of variables in the distributions, the norms of the resulting vectors increase in magnitude as well, apparently converging to the sqrt(d). Hence the mean value of the norms increases. Their deviation however remains unaffected.

Problem 4
The goal of this problem is to implement a very simple text retrieval system. Given (as input) a database of documents as well as a query document (all provided in an attached .zip file), write a program, in a language of your choice, to find the document in the database that is the best match to the query. Specifically:

(a) Write a small parser to read each document and convert it into a vector of words.

(b) Compute tf-idf values for each word in every document as well as the query.

(c) Compute the cosine similarity between tf-idf vectors of each document and the query.

(d) Report the document with the maximum similarity value.

(Solution) Thanks to Dimitrios Chaikalis for the following solution for (a - d):

HW1_Q4

February 8, 2020

[0]:
```
import numpy as np
import math

d1 = open('d1.txt','r')
d2 = open('d2.txt','r')
d3 = open('d3.txt','r')
d4 = open('d4.txt','r')
d5 = open('d5.txt','r')
dq = open('d_query.txt','r')

def get_words(fileid):
  words = []
  temp = []
  while(True):
    r = fileid.read(1)
    if(r.isalpha()):
      temp.append(r)
    else:
      word = ''.join(temp)
      if(word!=""):
        in_lowercase = word.casefold()
        words.append(in_lowercase)
        temp = []
    if(r==''):
      break
  return words

D1 = get_words(d1)
D2 = get_words(d2)
D3 = get_words(d3)
D4 = get_words(d4)
D5 = get_words(d5)
DQ = get_words(dq)
```

In the above cell, we read the text files, ignore all non-words (e.g. numbers and punctuation) then turn all words into lowercase to avoid ambiguities, and make a vector for each text file.

[0]:
```
Dictionary = []
D = [D1 , D2 , D3 , D4 , D5 , DQ]
for i in range(6):
  for j in range(len(D[i])):
    temp = D[i][j]
    index = Dictionary.count(temp)
    if(index==0):
      Dictionary.append(temp)
```

Dictionary, meaning a vector with all the words appearing in the documents, is computed. Crucial for finding the 'idf' value and also dictates our vector length.

[0]:
```
idf_map = {}
for i in range(len(Dictionary)):
  temp = Dictionary[i]
  count = 0
  for j in range(6):
    if(temp in D[j]):
      count = count + 1
  idf_map[temp] = math.log(6/count)
```

'idf_map' is a python dictionary structure, assigning every word its 'idf' value.

[0]:
```
def get_tf_map(word_vector):
  tf_map = {}
  for i in range(len(word_vector)):
    temp = word_vector[i]
    if(temp in tf_map):
      tf_map[temp] = (tf_map[temp] + 1)
    else:
      tf_map[temp] = 1
  return tf_map

tf_1 = get_tf_map(D1)
tf_2 = get_tf_map(D2)
tf_3 = get_tf_map(D3)
tf_4 = get_tf_map(D4)
tf_5 = get_tf_map(D5)
tf_q = get_tf_map(DQ)
```

'tf_i' is a map assigning every word of document 'i' its 'tf' value.

[0]:
```
def get_complete_vector(idf,tf):
  V = []
  for word in idf:
    if(word in tf):
      val = idf[word]*tf[word]
    else:
```

```
      val = 0
    V.append(val)
  return V

V1 = get_complete_vector(idf_map,tf_1)
V2 = get_complete_vector(idf_map,tf_2)
V3 = get_complete_vector(idf_map,tf_3)
V4 = get_complete_vector(idf_map,tf_4)
V5 = get_complete_vector(idf_map,tf_5)
VQ = get_complete_vector(idf_map,tf_q)
```

'Vi' is the final vector for document 'i', containing the 'tf*idf' value of each word in the appropriate cell.

[0]:
```
def get_cosine(vec,q_vec):
  dot = np.dot(vec,q_vec)
  nv = np.linalg.norm(vec,2)
  nq = np.linalg.norm(q_vec,2)
  cosine = dot/(nv*nq)
  return cosine

c1 = get_cosine(V1,VQ)
c2 = get_cosine(V2,VQ)
c3 = get_cosine(V3,VQ)
c4 = get_cosine(V4,VQ)
c5 = get_cosine(V5,VQ)
```

Finally, we calculate the cosine similarity, and find which document gets the greatest value.

[0]:
```
C = [c1,c2,c3,c4,c5]
num = np.argmax(C) + 1
print('The best match, is document d%d.txt'%num)
```

```
The best match, is document d4.txt
```