Machine Learning Exercises: Set 1

Roger Garriga Calleja

January 30, 2017

Problem 1: Let X be a real-valued random variable with mean m, median M, and standard deviation. Prove that

$$|m - M| \leqslant \sqrt{2}\sigma.$$

Chebyshev's inequality states that $P(|X - \mathbb{E}X| \geqslant t) \leqslant \frac{\sigma^2}{t^2}$
First of all consider $|X - m|$ and apply Chevyshev with $t = \sqrt{2}\sigma$

$$P(|X - m| \geqslant \sqrt{2}\sigma) \leqslant \frac{\sigma^2}{2\sigma^2} = \frac{1}{2}.$$

That means that $P(X \notin [m - \sqrt{2}\sigma, m + \sqrt{2}\sigma]) \leqslant \frac{1}{2}$. Taking the complementary we get that $P(X \in [m - \sqrt{2}\sigma, m + \sqrt{2}\sigma]) \geqslant \frac{1}{2}$, which means that the median M will be in the region, so $M \in [m - \sqrt{2}\sigma, m + \sqrt{2}\sigma] \Rightarrow |m - M| \leqslant \sqrt{2}\sigma$.

Problem 2: Write a program that compares the performance of the empirical mean and the median-of-means mean estimators. Test them on randomly generated samples drawn from different distributions, including heavy-tailed ones. Try different parameters of the median-of-means estimator and different sample sizes. Compare the estimators according to different measures, such as average deviation from the true mean, as well as worst case deviation (when the random sample is re-drawn many times). You may consider the Pareto family or Student's t-distribution (with different degrees of freedom) for heavy-tailed examples.

We tested the performance of MoM and sample mean estimators using different distributions (Normal, Binomial, Poisson, Gamma, Pareto, T-Student), varying the precision parameter on MoM (δ), the sample size (n) and the parameters of the distributions. In order to test the performance of the estimators we generate a number (num) of samples and compute the average deviation and the worst case deviation.

On the tables we will use the next notation

- Parameters of the distribution in a parenthesis
- Sample size: n
- Precision of MoM: δ
- Number of divisions in MoM: K
- Number of samples: N

- Expected value: Exp
- Average sample mean: $Mean$
- Average MoM: MoM
- Worst case deviation: WC (mean) and WC (MoM)
- Average deviation: Dev (mean) and Dev (MoM)

Light tailed distributions:

Normal:

(μ, σ^2)	n	δ	K	N	Exp	M	MoM	WC M	WC MoM	Dev M	Dev MoM
(0,1)	10^3	0.05	23	100	0	0.0002	0.0009	0.0728	0.125	0.0242	0.034
(0,1)	10^3	0.05	23	10^3	0	-0.0008	-0.0006	0.125	0.155	0.0256	0.031
(0,1)	10^3	0.3	9	10^3	0	-0.0004	0.001	0.0971	0.157	0.0253	0.0308
(0,1)	10^3	0.05	23	10^4	0	0.0003	0.0002	0.126	0.17	0.0251	0.031
(0,10^3)	10^3	0.05	23	10^4	0	0.347	0.0036	119	162	25	31.4
(0,10^3)	10^3	0.001	55	10^4	0	0.0486	0.0074	124	159	25.2	31.5

Poisson:

λ	n	δ	K	N	Exp	M	MoM	WC M	WC MoM	Dev M	Dev MoM
1	10^3	0.05	23	100	1	1	0.999	0.087	0.0909	0.0228	0.0307
1	100	0.05	23	10^3	1	1	0.966	0.35	0.4	0.0787	0.0768
15	10^3	0.4	7	10^3	15	15	15	0.398	0.483	0.0987	0.12

Heavy tailed distributions:

Pareto:

α	n	δ	K	N	Exp	M	MoM	WC M	WC MoM	Dev M	Dev MoM
1.1	10^3	0.05	23	10^3	10	7.35	3.46	353	7.65	6	6.54
1.1	10^4	0.05	23	10^3	10	8.07	4.7	434	6.27	5.04	5.3
1.1	10^4	0.001	73	10^4	10	7.98	4.08	$1.5{\cdot}10^5$	6.87	5.03	5.92
1.5	10^4	0.001	73	10^3	2	2	1.73	21.4	0.503	0.166	0.278
1.5	10^3	0.2	12	10^3	2	2.06	1.7	29.4	0.81	0.373	0.318
1.5	20	0.5	5	10^3	2	1.97	1.17	179	2.84	1.08	0.94
2	20	0.5	5	10^3	1	1.03	0.744	17	2.45	0.393	0.352
2	10^3	0.1	18	10^3	1	0.996	0.923	0.742	0.298	0.0703	0.09

T-Student:

ν	n	δ	K	N	Exp	M	MoM	WC M	WC MoM	Dev M	Dev MoM
1	10^3	0.05	23	10^3	0	-4.89	-0.027	4310	1.27	8.05	0.274
1	100	0.1	18	10^3	0	-0.444	-0.00356	321	1.54	3.37	0.29
2	100	0.1	18	10^3	0	-0.0044	-0.001	8.01	0.781	0.248	0.166
2	10^3	0.01	36	10^3	0	0.0045	0.00257	0.826	0.361	0.0879	0.0701
5	100	0.1	18	10^3	0	-0.00852	-0.00511	0.446	0.509	0.104	0.122

As can be seen in the tables, in average the sample mean is closer than MoM to the expected value, but not much closer. However, if we look at the worst case deviation, there are cases in which the sample mean has a huge deviation but the MoM remains fairly close to the expected value.
For the light-tailed distributions it would be better to use the sample mean as estimator. Whereas for the heavy-tailed ones it may be dangerous to use the sample mean because there are time when the deviation from the expected value is very large. If we are sure that the underlying distribution of our data is light-tailed we can use the sample mean, otherwise using it may lead to big mistakes, so MoM is much more conservative in this sense.

Problem 3: Let $X_1, \ldots, X_n$ be i.i.d. *non-negative* random variables with mean $\mathbb{E}X_1 = m$ and second moment $\mathbb{E}X_1^2 = a^2$. Use the Chernoff bound to prove that, for all $t \in (0, m)$,

$$P\left\{\frac{1}{n}\sum_{i=1}^{n} X_i < m - t\right\} \leqslant e^{-\frac{nt^2}{2a^2}}.$$

***Hint:* use the fact that for $x > 0$, $e^{-x} \leqslant 1 - x + \frac{x^2}{2}$.**

Working a bit the equation we get

$$P\left(\frac{1}{n}\sum_{i=1}^{n} X_i < m - t\right) = P\left(t < m - \frac{1}{n}\sum_{i=1}^{n} X_i\right) = P\left(m - \frac{1}{n}\sum_{i=1}^{n} X_i > t\right) = P\left(\sum_{i=1}^{n}\mathbb{E}X - \sum_{i=1}^{n} X_i > nt\right).$$

Now we can apply Chernoff bound taking into account that X_i are independent, so

$$P\left(\sum_{i=1}^{n}\mathbb{E}X - \sum_{i=1}^{n} X_i > nt\right) \leqslant \frac{\mathbb{E}\prod_{i=1}^{n} e^{\lambda(\mathbb{E}X_i - X_i)}}{e^{\lambda t n}}.$$

Then, as we have $\mathbb{E}X_i = m \ \forall i$, we get

$$P\left(\sum_{i=1}^{n}\mathbb{E}X - \sum_{i=1}^{n} X_i > nt\right) \leqslant \frac{\mathbb{E}\prod_{i=1}^{n} e^{\lambda(\mathbb{E}X_i - X_i)}}{e^{\lambda t n}} = \frac{1}{e^{n\lambda t}}\prod_{i=0}^{n}\mathbb{E}e^{\lambda(m - X_i)} = \frac{e^{n\lambda m}}{e^{n\lambda t}}\prod_{i=0}^{n}\mathbb{E}e^{-\lambda X_i}.$$

As $e^{-x} \leqslant 1 - x + \frac{x^2}{2}$ for $x > 0$ and our X_i are non-negative, we get

$$\frac{e^{n\lambda m}}{e^{n\lambda t}}\prod_{i=0}^{n}\mathbb{E}e^{-\lambda X_i} \leqslant e^{n\lambda(m-t)}\prod_{i=1}^{n}(1 - \lambda\mathbb{E}X_i + \frac{\lambda^2}{2}\mathbb{E}X_i^2).$$

Now, since $1 + x \leqslant e^x \ \forall x \in \Re$ (on $x = 0$ both are equal and e^x derivative is greater or equal to 1 for $x \geqslant 0$ and less than 1 for $x < 0$), we get

$$e^{n\lambda(m-t)}\prod_{i=1}^{n}(1 - \lambda\mathbb{E}X_i + \frac{\lambda^2}{2}\mathbb{E}X_i^2) \leqslant e^{n\lambda(m-t)}\prod_{i=1}^{n} e^{-\lambda m + \frac{\lambda^2}{2}a^2} = e^{n\lambda(m-t)}e^{n(-\lambda m + \frac{\lambda^2}{2}a^2)} = e^{-n\lambda t + n\frac{\lambda^2}{2}a^2}.$$

We minimize the function on λ to get the bound. $0 = (-n\lambda t + n\frac{\lambda^2}{2}a^2)' = -nt + n\lambda a^2 \Leftrightarrow \lambda = \frac{t}{a^2}$. That implies

$$e^{-n\lambda t + n\frac{\lambda^2}{2}a^2} \leqslant e^{-\frac{nt^2}{2a^2}}.$$

So, $P\left(\frac{1}{n}\sum_{i=1}^{n} X_i < m - t\right) \leqslant e^{-\frac{nt^2}{2a^2}}$. Q.E.D.

Problem 4: Write a program that projects the n standard basis vectors in $\Re^n$ to a random 2-dimensional subspace. (You may do this simply by using a $2 \times n$ matrix whose entries are i.i.d. normals.) Center the point set appropriately and re-scale such that the empirical variance of the (say) first component equals 1. Plot the obtained point set. Now generate n independent standard normal vectors on the plane and compare the two plots. Do this for a wide range of values of n. What do you see?
Repeat the same exercise but now projecting the $2n$ vertices of the hypercube $\{1, 1\}^n$ instead of the standard basis vectors. (Naturally, you can only do this for small values of n, say up to $n \simeq 13$.) What do you see now?

The projected standard basis vectors of $\Re^n$ to a 2-dimensional subspace are simply random vectors following a normal distribution. Intuitively we could prognosticate that, because we multiply a random matrix whose entries are standard normal with parameters $\mu = 0$, $\sigma^2 = \frac{1}{d}$ with an identity matrix. After centering and rescaling we make force the entries to be normal with parameters $\mu = 0$, $\sigma^2 = 1$.

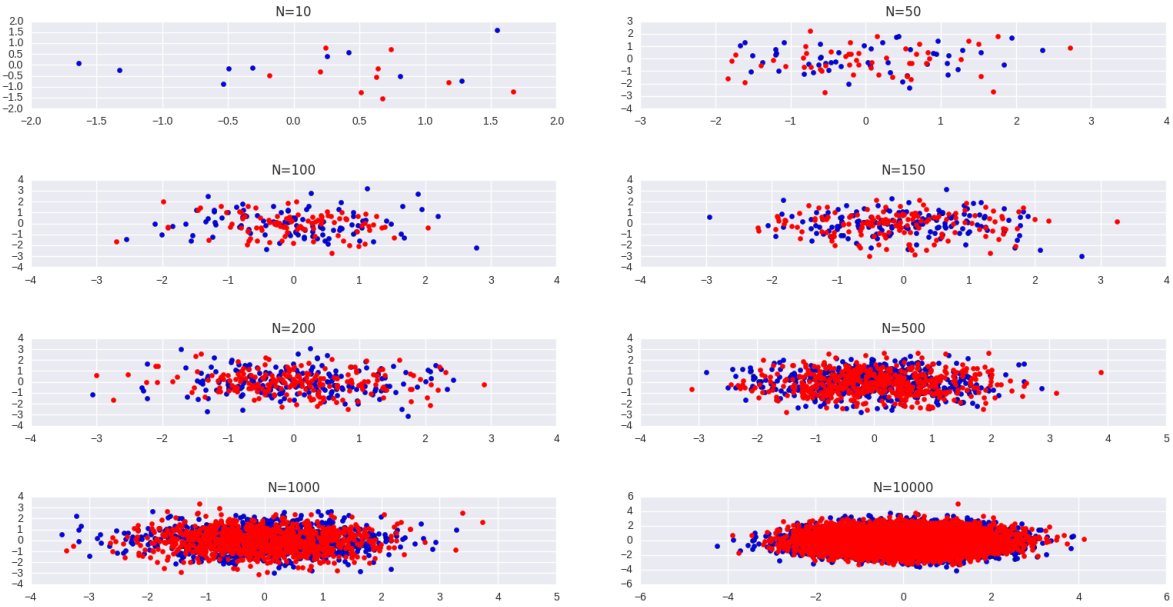

Figure 1: Projected basis (blue) and standard normal vectors (red).

The projected hypercubes to a 2-dimensional subspace preserve the structure they had before being projected. The projection is like cutting the D-space on a random plane and project the hypercube on it. The result is that the points that the points keep structure on the pairwise distance, in the sense that if the distance between a pair of points is the same as the distance between another pair of points, in the projections they will have also the same distance. The ones that are the closest remain the closest.

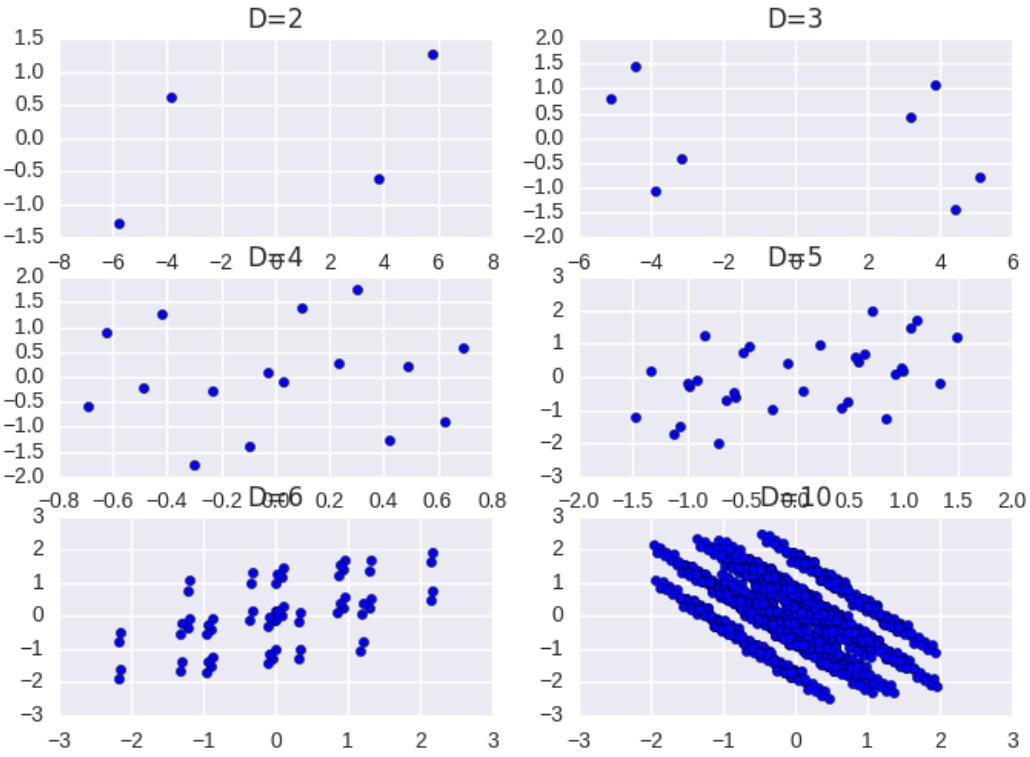

Figure 2: Projected hypercube (blue).

1 Appendix

Ex 2:

```
import sklearn
import math as mat
import numpy as np
import itertools as itt
import operator as op
import pandas as pd
import matplotlib.pyplot as plt
from numpy import random as rand
import functools as ft

#Given a sample (data) and a precision (delta). Computes de MoM estimator with
#K=[8*log(1/delta)] divisions.
def MoM(data,delta=0.1):
        K=int(8*mat.log(1/delta)) #compute num of divisions.
        rand.shuffle(data) #shuffle the data.
        sp_data=np.array_split(data,K) #split the data into K sets.
        means=list(map(np.mean,sp_data)) #computes the mean in each set.
        return np.median(means) #computes the median of the means.


#distr: Normal(mu,var), Binomial(n,p), Poisson(lbd), Gamma(k,theta),
#Pareto(a), T-Student(nu), Weibull, Lognorm
#parm1 and parm2: parameters of the distribution
#n: number of random numbers in each sample
#draws: number of times it is going to be performed
#delta: precision
def perf_MoM(distr,parm1,parm2,n,draws,delta):
        #replicate the parameters a number of times indicated by the draws.
        p1=[parm1]*draws
        p2=[parm2]*draws
        num=[n]*draws
        prec=delta
        if distr=="Normal": #Generates 'draws' iid Normal samples of 'num' element
                data=list(map(rand.normal,p1,p2,num))
                exp_value=[parm1]*draws
        elif distr=="Binomial": #Generates 'draws' iid Binomial samples of 'num' e
                data=list(map(rand.binomial,p1,p2,num))
                exp_value=[parm1*parm2]*draws
        elif distr=="Poisson": #Generates 'draws' iid Poisson samples of 'num' eler
                data=list(map(rand.poisson,p1,num))
                exp_value=[parm1]*draws
        elif distr=="Gamma": #Generates 'draws' iid Gamma samples of 'num' element
                data=list(map(rand.gamma,p1,p2,num))
                exp_value=[parm1*parm2]*draws
```

```
        elif distr=="Pareto": #Generates 'draws' iid Pareto samples of 'num' eleme
            data=list(map(rand.pareto,p1,num))
            exp_value=[parm1/(parm1-1)-1]*draws
        elif distr=="T-Student": #Generates 'draws' iid T-student samples of 'num'
            data=list(map(rand.standard_t,p1,num))
            exp_value=[0]*draws
        elif distr=="Weibull": #Generates 'draws' iid Weibull samples of 'num' eler
            data=list(map(op.mul,list(map(rand.weibull,p1,num)),p2))
            exp_value=[parm2*(mat.gamma(1+1/parm1))]*draws
        elif distr=="Lognorm": #Generates 'draws' iid Lognorm samples of 'num' eler
            data=list(map(rand.lognormal,p1,p2,num))
            exp_value=[mat.exp(parm1+parm2/2)]*draws
        MoM_est=list(map(ft.partial(MoM,delta=prec),data)) #Compute the 'draws' es
        mean_est=list(map(np.mean,data)) #Compute the 'draws' estimators using sam
        dif_mean=list(map(abs,map(op.sub,exp_value,mean_est))) #Compute the error
        dif_MoM=list(map(abs,map(op.sub,exp_value,MoM_est))) #Compute the error of
        #eval performance using worst case
        wc_mean=max(dif_mean)
        wc_MoM=max(dif_MoM)
        #eval performance using best case
        bc_mean=min(dif_mean)
        bc_MoM=min(dif_MoM)
        #eval performance using average of error
        ave_mean=sum(dif_mean)/len(dif_mean)
        ave_MoM=sum(dif_MoM)/len(dif_MoM)
        #compute the mean of the estimators over the 'draws' samples.
        ave_mean_MoM=np.mean(MoM_est)
        ave_mean_mean=np.mean(mean_est)
        #plt.hist(MoM_est,color="blue")
        #plt.hist(mean_est,color="green")
        compare=pd.DataFrame([['','Exp value','Worst case dev','Best case dev',
        'Average dev','ave_mean'],
        ['Mean',exp_value[0],wc_mean,bc_mean,ave_mean,ave_mean_mean],
        ['MoM',exp_value[0],wc_MoM,bc_MoM,ave_MoM,ave_mean_MoM]])
        return compare
```

Ex 4:

```
import numpy as np
import itertools as itt
import matplotlib.pyplot as plt
from numpy import random as rand

def proj_basis(d,D): #Projects the basis of a D-dim space to a d-dim space
        W=rand.normal(0,1/d,(d,D)) #Generate a random matrix to project D-dim vect
        basis=np.identity(D) #Generate the basis of a D-dim space
        proj_vect=np.dot(W,basis) #Project the basis
        proj_vect[0]=proj_vect[0]-np.mean(proj_vect[0]) #center first component
        proj_vect[1]=proj_vect[1]-np.mean(proj_vect[1]) #center second component
        std_dev=np.sqrt(np.var(proj_vect[0,])) #compute the std dev of the first co
```

```
        proj_vect=proj_vect/std_dev #rescale by first component
        return proj_vect

d=2
i=0
rng=[10,50,100,150,200,500,1000,10000]
for D in rng: #Plot the proj basis for a rng of dimensions D into a d-dim space
        i=i+1
        proj_vect=proj_basis(d,D)
        rnd_vect_plane=rand.normal(0,1,(2,D)) #generate random normals
        plt.subplot(4,2,i) #more than one plot
        plt.scatter(proj_vect[0],proj_vect[1])
        plt.scatter(rnd_vect_plane[0],rnd_vect_plane[1],color="red")
        plt.title("N=%d"%D) #change title

#hypercube

def proj_hypercube(d,D):
        vmax=[1]*D
        vmin=[-1]*D

        hypercube=np.transpose(np.asarray(list(itt.product(*zip(vmin,vmax))))) #ge
        W=rand.normal(0,1/d,(d,D)) #Generates the projection matrix
        proj_hyp_cube=np.dot(W,hypercube) #Projects
        proj_hyp_cube[0]=proj_hyp_cube[0]-np.mean(proj_hyp_cube[0])
        proj_hyp_cube[1]=proj_hyp_cube[1]-np.mean(proj_hyp_cube[1])
        std_dev=np.sqrt(np.var(proj_hyp_cube[1,]))
        proj_hyp_cube=proj_hyp_cube/std_dev

        return proj_hyp_cube

d=2
rng=[2,3,4,5,6,10]
i=0
for D in rng: #projects the hypercubes from different dimensions to a 2-dim subspa
        i=i+1
        proj_hyp_cube=proj_hypercube(d,D)
        plt.subplot(3,2,i) #more than one plot
        plt.scatter(proj_hyp_cube[0],proj_hyp_cube[1])
        plt.title("D=%d"%D) #change title
```