Machine Learning Problemset #1

Nandan Rao

1

We prove:

$$|m - M| \leq \sqrt{2}\sigma$$

By using Chebyshev's inequality along with the fact that standard deviation, σ, is the square root of the variance:

$$P(|X - \mathbb{E}[X]| \geq a) \leq \frac{var[X]}{a^2}$$
$$P(|X - \mathbb{E}[X]| \geq \sqrt{2}\sigma) \leq \frac{\sigma^2}{(\sqrt{2}\sigma)^2}$$
$$P(|X - \mathbb{E}[X]| \geq \sqrt{2}\sigma) \leq \frac{1}{2}$$

For a random continuous variable, the median is defined as the point at which the probability of a a realization being greater is exactly one half (the middle of the distribution!). If the probability of any random variable being more than $\sqrt{2}\sigma$ from the mean is less than one half, and the median is the point at which all points greater are have probability $1/2$ or less, than the median cannot be more than $\sqrt{2}\sigma$ from the mean!

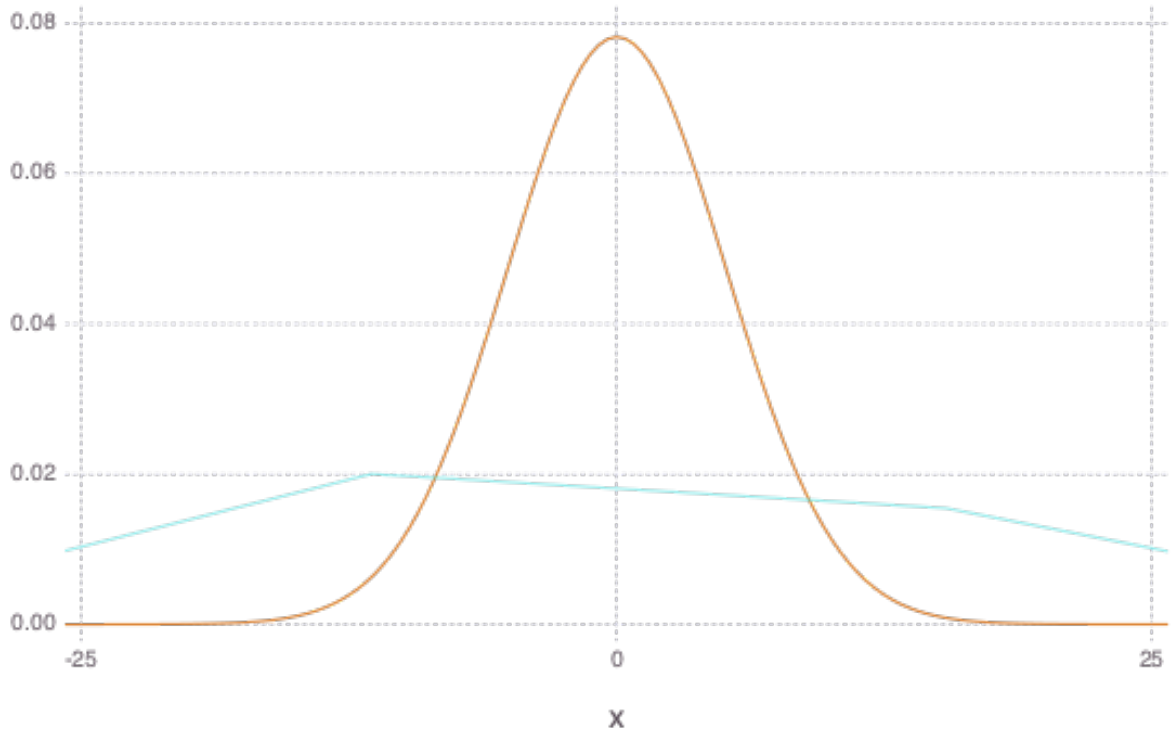
2

```
using Distributions, Gadfly, DataFrames, Iterators

function nfolds(x::AbstractArray, n::Int)
    l = length(x)
    [x[round(Int64,(i-1)*l/n)+1 : min(l, round(Int64, i*l/n))] for i in 1:n]
end

mom(X, k) = median([mean(i) for i in nfolds(X, k)])

function generate_means(D::Sampleable, N::Int, M::Int, K::Int)
    [(mean(r), mom(r,K)) for r in [ rand(D, N) for i in 1:M ]]
end

function compare_means(means::Array{Tuple{Float64,Float64}})
    geom = Geom.density(bandwidth=5)
    plot(
        layer(x = [i for (i,_) in means], geom,
              Theme(default_color=colorant"rgba(50,220,220,0.5)")),
        layer(x = [j for (_,j) in means], geom,
```

```
            Theme(default_color=colorant"rgba(220,130,50,1)")),
        Coord.cartesian(xmin=-25, xmax=25)
    )
end
```

We begin by looking at density estimates of the estimators, this gives a sense of the distribution and overall behavior of the estimator itself, as we know we are trying to estimate the mean which should be 0. We start with a Cauchy to compare, where the median of means has beautiful sub-gaussian tails even with this difficult distribution:

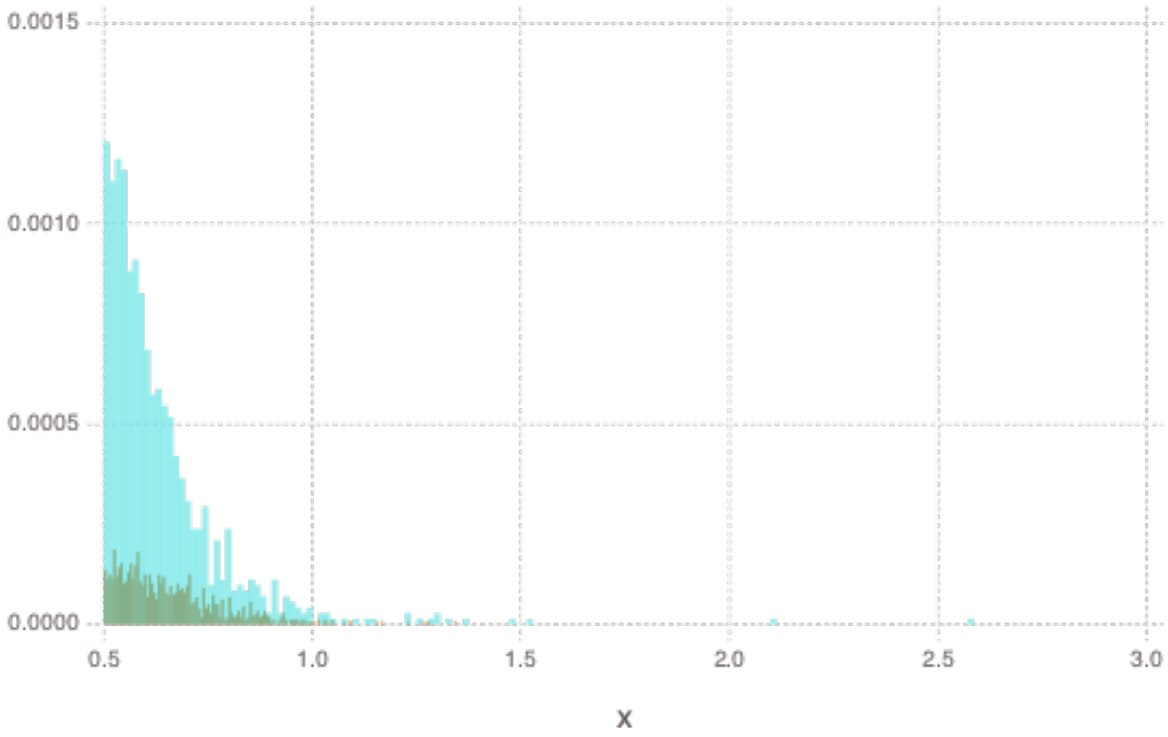
```
compare_means(generate_means(Cauchy(), 100, 10000, 5))
```



We then take a look at some more well-behaved dsitributions: we consider the Student's T. Here we will zoom into the tail of the distribution in order to see things more effectively, we see that even with 3 degrees of freedom, the Median of Means is performing much better in the tails:

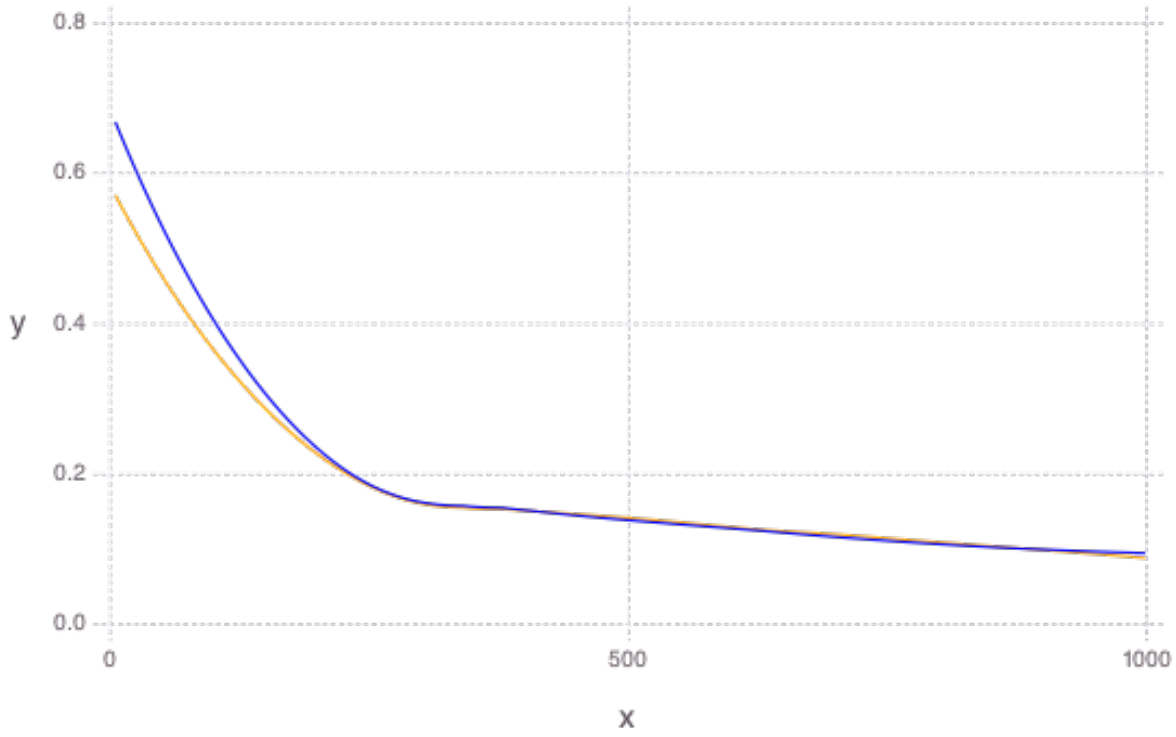
```
function compare_tails(means::Array{Tuple{Float64,Float64}}, tail = 1)
    geom = Geom.histogram(density=true)
    f = filter(t -> all(x->(x>tail), t), means)
    plot(
        layer(x = [i for (i,_) in f], geom,
              Theme(default_color=colorant"rgba(50,220,220,0.5)")),
        layer(x = [j for (_,j) in f], geom,
              Theme(default_color=colorant"rgba(220,130,50,1)")),
        Coord.cartesian(xmin=tail, xmax=3)
    )
end
```

```
compare_tails(generate_means(TDist(3), 50, 100000, 5), .5)
```



And we can also look at the extreme values to see how the worst deviation is for an even more well behaved distribution, Student's T with 5 degrees of freedom:

```
worst_case(means::Array{Tuple{Float64,Float64}}) =
    maximum([map(abs, a) for a in means])

function plot_worst_cases(D::Sampleable, M::Int, K::Int, sizes::AbstractArray{Int})
    cases = [worst_case(generate_means(D, N, M, K)) for N in sizes]
    geom = Geom.smooth()
    plot(
        layer(y = [i for (i,_) in cases], x = sizes, geom,
              Theme(default_color=colorant"blue")),
        layer(y = [j for (_,j) in cases], x = sizes, geom,
              Theme(default_color=colorant"orange"))
    )
end

plot_worst_cases(TDist(5), 30, 5, 5:5:1000)
```



3

Using the exponent rules $e^x \geq 1 + x$ and the provided $e^- x \leq 1 - x + \frac{x^2}{2}$, we work some magic after multiplying both sides of the inequality in our original probability by -1, so that we can apply Markov's rule, and assuming continuity so that $p(x > y) = p(x \geq y)$:

$$
\begin{aligned}
p\left(\frac{-1}{n}\sum X_i > t-m\right) &\leq \frac{\mathbb{E}[\frac{-1}{n}\sum X_i]}{t-m} \\
p\left(e^{\frac{-\lambda}{n}\sum X_i} > e^{\lambda(t-m)}\right) &\leq \frac{\mathbb{E}[e^{\frac{-\lambda}{n}\sum X_i}]}{e^{\lambda(t-m)}} \\
&\leq \frac{\mathbb{E}[e^{\frac{-\lambda}{n}X_i}]^n}{e^{\lambda(t-m)}} \\
&\leq \frac{\mathbb{E}[1+\frac{\lambda^2 X_1^2}{2n^2}-\frac{\lambda X_1}{n^2}]^n}{e^{\lambda(t-m)}} \\
&\leq \frac{\left(1+\frac{\lambda^2 a^2}{2n^2}-\frac{\lambda m}{n^2}\right)^n}{e^{\lambda(t-m)}} \\
&\leq \frac{\left(\exp\left\{\frac{\lambda^2 a^2}{2n^2}-\frac{\lambda m}{n}\right\}\right)^n}{e^{\lambda(t-m)}} \\
&\leq \exp\left\{\frac{\lambda^2 a^2}{2n}-\lambda m-\lambda(t-m)\right\} \\
&\leq \exp\left\{\frac{\lambda^2 a^2}{2n}-\lambda t\right\}
\end{aligned}
$$

Here we minimize with respect to λ:

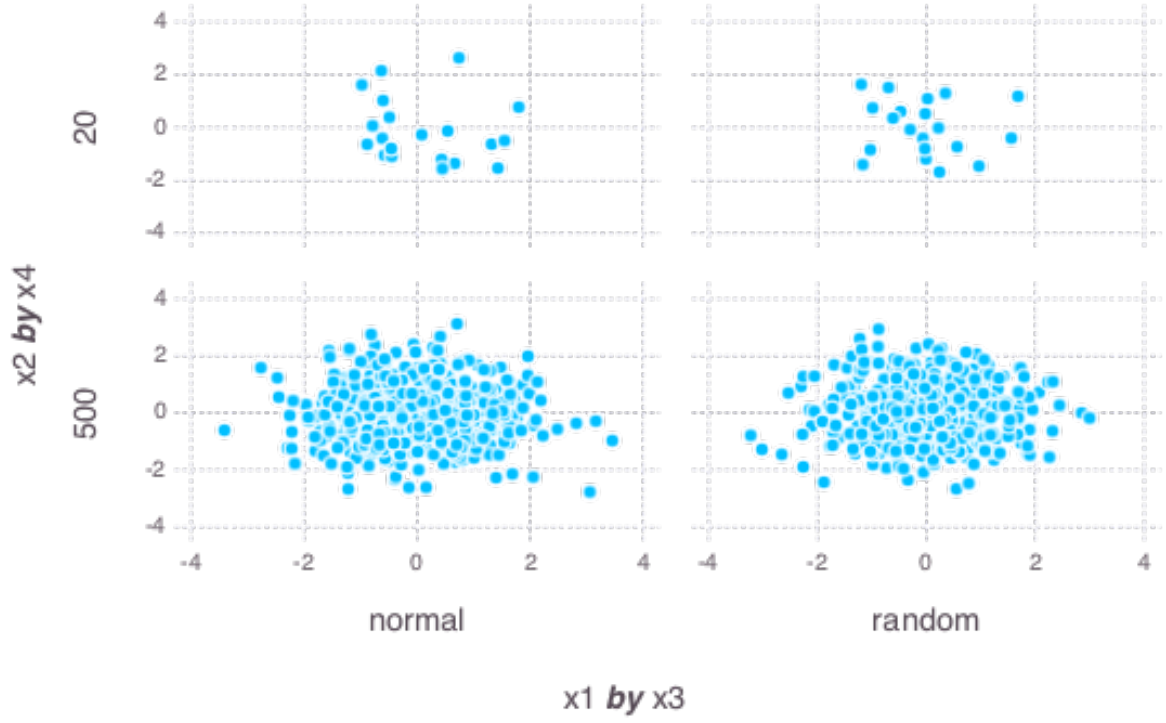
$$
\begin{aligned}
\frac{\lambda a^2}{n} - t &= 0 \\
\lambda &= \frac{tn}{a^2}
\end{aligned}
$$

Plugging this back into our equation we solve:

$$
\begin{aligned}
p\left(\frac{1}{n}\sum X_i < m-t\right) &\leq \exp\left\{\frac{t^2n^2a^2}{2na^4}-\frac{t^2n}{a^2}\right\} \\
&\leq \exp\left\{\frac{nt^2}{2a^2}-\frac{nt^2}{a^2}\right\} \\
&\leq \exp\left\{-\frac{nt^2}{2a^2}\right\}
\end{aligned}
$$

4

```
scale_and_center(p) = (p - mean(p))/var(p)
project(n::Int, m = eye(n), d = 2) = rand(Normal(), d, n) * m

function compare_plots(n::Int)
    s = scale_and_center(project(n))
    r = rand(Normal(), 2, n)
    DataFrame([s' fill("normal", n) fill(n, n);
               r' fill("random", n) fill(n, n)])
end

function compare_all_plots(nums = [20,500])
    c = vcat([compare_plots(n) for n in nums])
    plot(c, x = "x1", y = "x2", xgroup="x3", ygroup="x4",
         Geom.subplot_grid(Geom.point))
end

compare_all_plots([20,500])
```

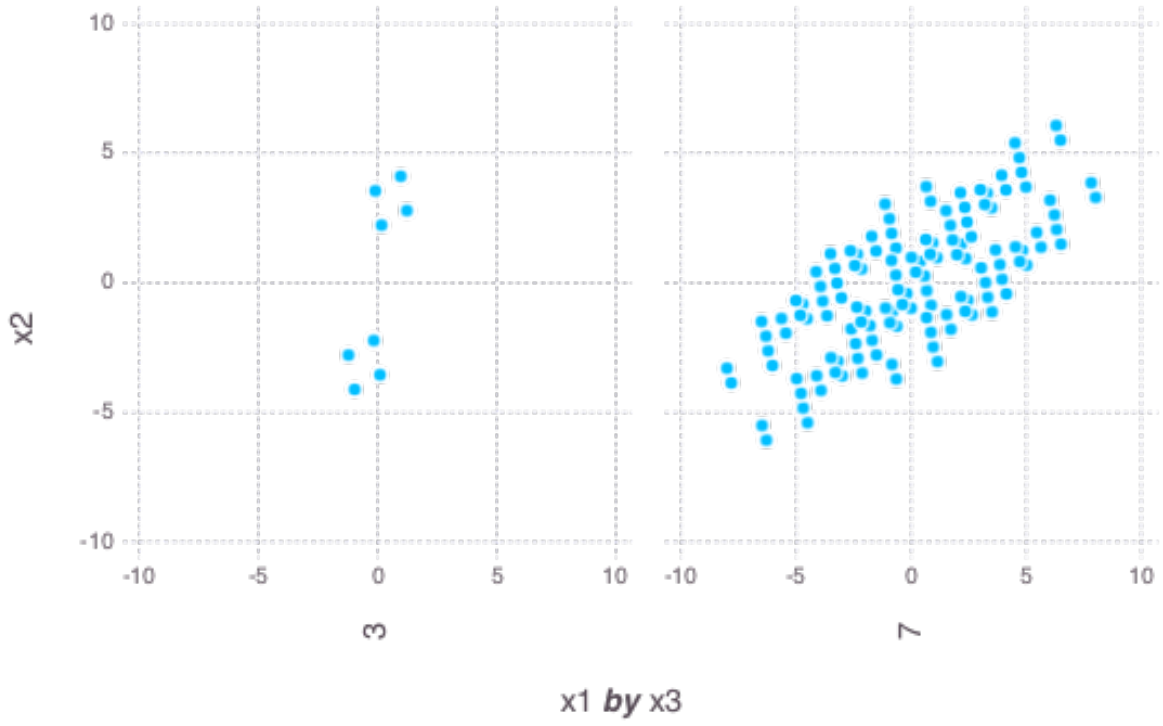



```
function boxes(n::Int)
    c = collect(product(repeat([[-1,1]], outer=n)...))
    reinterpret(Int, c, (n, 2^n))
end

project_boxes(n::Int) = project(n, boxes(n))'
```

We see that the projected basis vectors resemble precisely a multivariate normal distribu-

tion. This is not especially surprising, seeing as a set of standard basis vectors can also be represented as the identity matrix, and a random normal matrix multiplied by the identity matrix should look fairly normal!

```
function make_df_from_calls(fn::Function, arr)
    m = reduce((a,b) -> [a; fn(b) fill(b, 2^b)], Array{Any}(0,3), arr)
    DataFrame(m)
end

function plot_projections(arr::Vector{Int})
    df = make_df_from_calls(project_boxes, arr)
    plot(df, x = "x1", y="x2", xgroup="x3", Geom.subplot_grid(Geom.point))
end

plot_projections([3, 7])
```



I think we can all agree that these projections are beautiful. We can see what can really be described as the shadow of the box, projected onto a 2-dimensional surface. The angle of the light is chosen by the random normal projection matrix, but the shape of the box and the structure of its points remains, at least as long as the dimension of the box does not get TOO large (at which point, there is still some structure, but it gets harder and harder to distinguish as more and more points get compressed onto our tiny 2-dimensional canvas).