Curse of Dimensionality

RK

July 29, 2014

Abstract

This document contains some aspects relating to "Curse of dimensionality" that often plays a devilish role while one tries to use nearest neighbor methods.

Contents

1 Introduction

We frequently encounter data sets that have high number of predictor variates or covariates. There are many ways to model such data, ranging from high bias + low variance multiple linear regression to the low bias + high variance nearest neighbor methods. In situations where the covariates have a true non linear relationship among them, nearest neighborhood methods work very well. Nearest neighbor methods are critically dependent on the fact that there are enough data points around each sample point so that some sort of averaging behavior can be computed. However when the dimensions increase, then things start to deteriorate. "What exactly happens in high dimensional cases?", is a question that often comes up. This note contains some pointers that can help one understand this issue.

2 Why are high dimensional spaces different ?

2.1 p dimensional hypercube

There are many notions that seem fine in low dimensional cases but are not ok in high dimensional cases. Let's explore the concept of distance. Let's start with two dimensional cube, i.e a square. We want to know the average distance of any point from the origin. The way to go about finding this is to generate random points in the square, compute the distance of each of the point from the origin, and then look at the distribution. The following code plots the distribution in the case of 2 dimensional hypercube and 100 dimensional hypercube.

```r
set.seed(1234)
n       <- 100000
p       <- 2
data    <- matrix(runif(n*p,0,1), nrow = n, ncol = p)
y       <- apply(data,1, function(z){sqrt(sum(z^2))})
densityplot(~y, plot.points=FALSE,lwd=2, xlim=c(0,2))
```

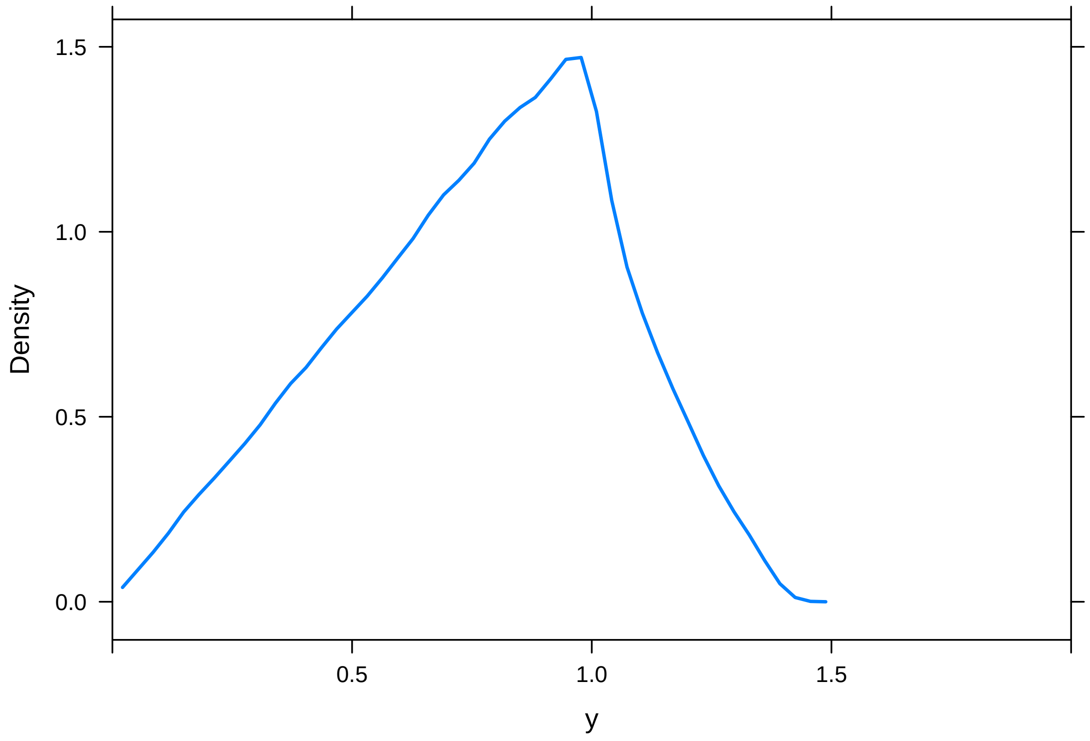

Figure 2.1: Distribution of distances from origin in p=2 dim

```
set.seed(1234)
n       <- 100000
p       <- 100
data    <- matrix(runif(n*p,0,1), nrow = n, ncol = p)
y       <- apply(data,1, function(z){sqrt(sum(z^2))})
densityplot(~y, plot.points=FALSE,lwd=2, xlim=c(0,8))
```

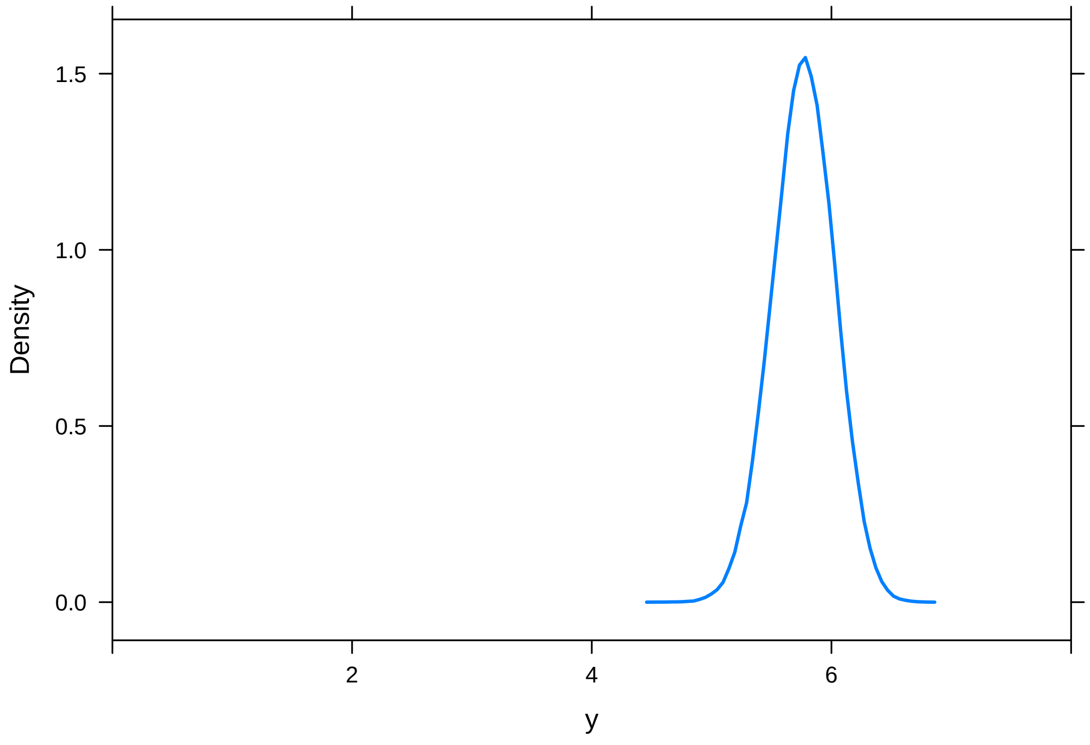

Figure 2.2: Distribution of distances from origin in p=100 dim

As one can see, in the high dimensional space, most of the points are at a distance of at least 4 units away from the origin.So, if you want to predict the value at the origin by nearest neighbor methods, there is a problem. There are no local points.

One can easily compute the average distance between between origin and every point by basic probability. Any point in the p dimensional hypercube is a point sampled from $[0,1]^p$. Denote the sample point by $z_i \in \mathbb{R}^p$. Hence the distance from the origin to this point is

$$d(0,x) = \sqrt{\sum_{p=1}^{100} z_i^2}$$

Now as p increases the $\sum_{p=1}^{100} z_i^2$ can be approximated as $E(z_i^2) \times p = p \times (1/4 + 1/12)$. Hence the distance as p becomes large is $\sqrt{p/3}$. In the case of $p = 100$, it is 5.7735. So I really need not have simulated to get this idea but simulation gives a visual reinforcement that strengthens long term memory. I mean, if one sees at the above density plot, it is likely that he/she will remember that distances are highly concentrated around a particular value.

2.2 p dimensional hypersphere

Let's explore the same aspect in a p dimensional hypersphere. Firstly, how does one generate data on a a circle. One naive method goes like this : Generate random numbers in a square centered at origin and then project those points on the circle inscribed in the square. There is a problem with this approach. A visual will more than clearly illustrate the issue

```
set.seed(1234)
n       <- 10000
p       <- 2
data    <- matrix(runif(n*p,-1,1), nrow = n, ncol = p)
x       <- data
y       <- apply(x,1, function(z){sqrt(sum(z^2))})
x[,1]   <- ifelse(y >1 , x[,1]/y, x[,1])
x[,2]   <- ifelse(y >1 , x[,2]/y, x[,2])
xyplot(x[,1]~x[,2], xlab = "x1",ylab  = "x2", pch = 19, cex=0.5)
```

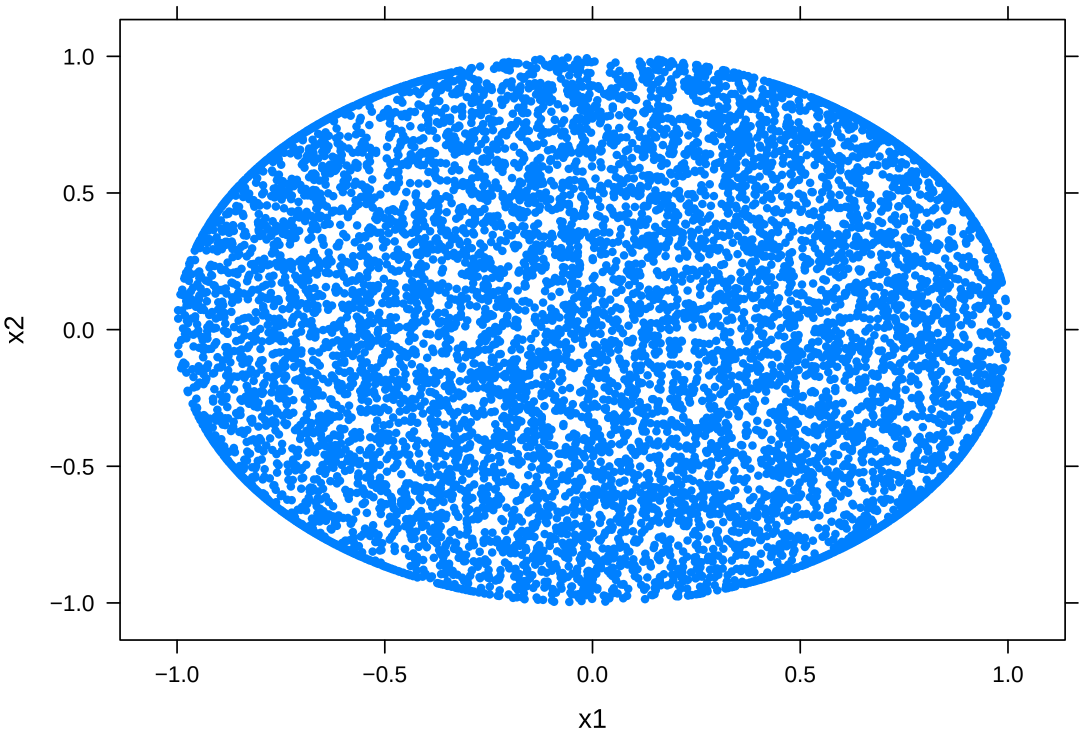

Figure 2.3: Generating samples by projection 2d

The visual shows that some parts of the circle are over sampled, i.e the border of the circle. An alternate method is via "acccept reject"

```
cond     <- y <= 1
z        <- data[cond,]
xyplot(z[,1]~z[,2], xlab = "x1",ylab  = "x2")
```

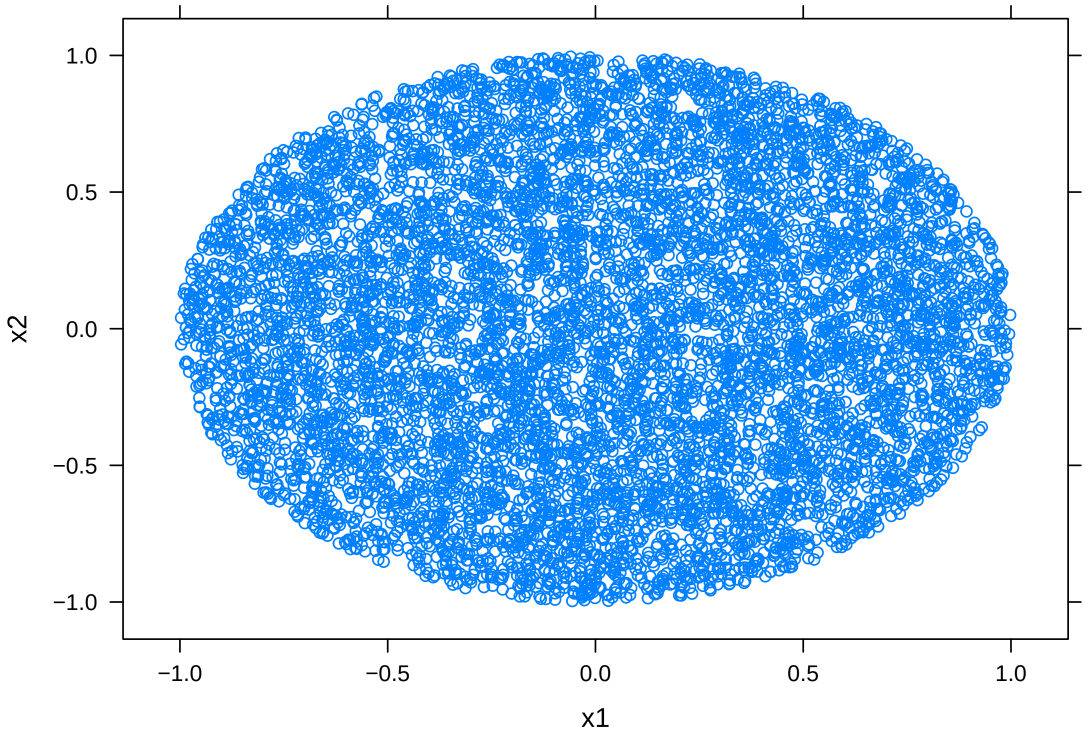

Figure 2.4: Generating samples by Rejection Sampling

This method works in lower dimensions but is not efficient in higher dimensions. Why ? The major problem with the algorithm is the running time. It grows worse than exponentially in p. The volume of a p dimensional unit ball is

$$V(p) = \frac{\pi^{p/2}}{\Gamma(1+p/2)}$$

and hence decreases more than exponentially as $p \to \infty$. Hence in plain english, this means the number of rejects will far outnumber the number of accepts.

The way to generate in higher dimensions is via multivariate normals. To give the rationale, let's look at two dimension case :

Let X_1 and X_2 be two IID standard normals. It can be easily seen that $Y_1 = \sqrt{X_1^2 + X_2^2}$ and $Y_2 = X_1/\sqrt{X_1^2 + x_2^2}$ are independent variables. The former is like the radius and the latter is like the direction. They are independent variables. You can easily do a change of variables and check that the density splits. Hence an easy way to generate random samples in a ball is choose the radius randomly and choose the direction randomly. Here are the details

- Choosing the distance for a p dimensional hypersphere with radius R

$$p(r_{sample} \leq r) = \left(\frac{r}{R}\right)^p$$

  Hence simulating $Ru^{1/p}$ where $u \sim Unif(0,1)$ gives the r_{sample}, the radius of the sample point
- Choosing the direction entails simulating a multivariate normal and dividing by its euclidean norm

$$X \sim N(\mathbf{0}, \mathbf{I}_p)$$

  Direction vector is $\frac{X}{||X|_2}$

Let's look at the code

```
set.seed(1234)
n       <- 10000
p       <- 2
data    <- rmvnorm(n, mean = rep(0,p), sigma = diag(p))
data    <- t(apply(data, 1, function(z){runif(1)^(1/p)*z/sqrt(sum(z^2))}))
xyplot(data[,1]~data[,2], xlab = "x1",ylab  = "x2", pch = 19, cex=0.5)
```

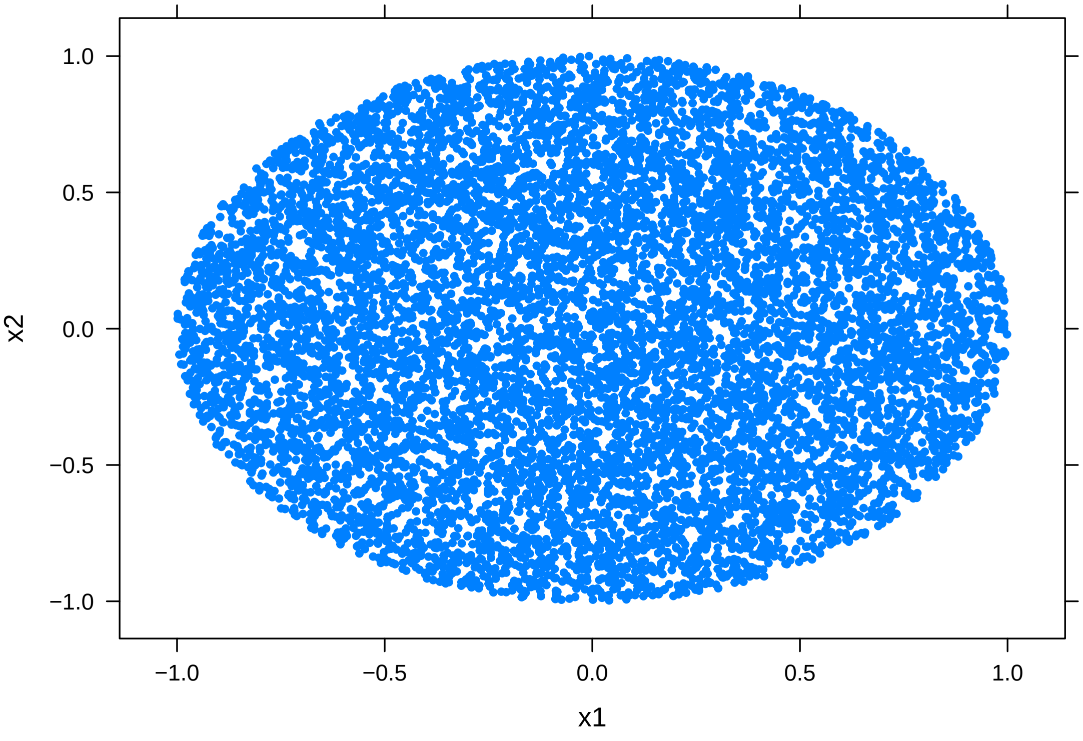

Figure 2.5: Distribution of distances from origin in 2 dim hypersphere

Let's compute the distribution of the distance from the origin.

```
set.seed(1234)
n       <- 10000
p       <- 2
data    <- rmvnorm(n, mean = rep(0,p), sigma = diag(p))
data    <- t(apply(data, 1, function(z){runif(1)^(1/p)*z/sqrt(sum(z^2))}))
y       <- apply(data,1, function(z){sqrt(sum(z^2))})
histogram(~y,nint=100, ylab="",xlab="")
```

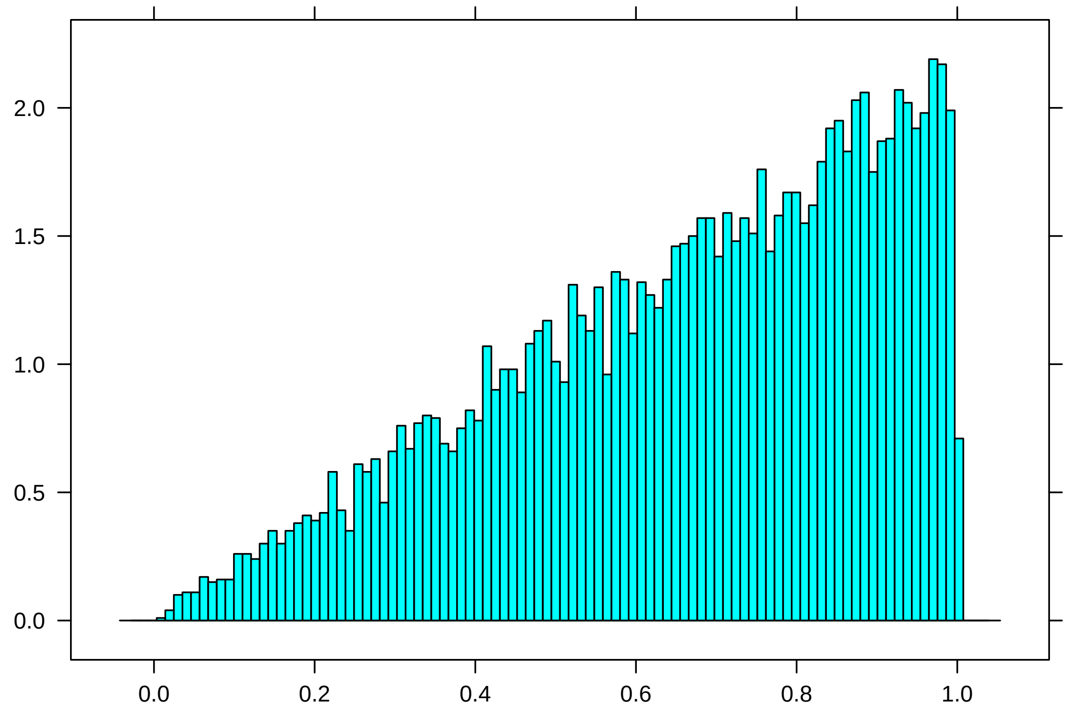

Figure 2.6: Distribution of distances from origin in a circle

```
set.seed(1234)
n       <- 10000
p       <- 100
data    <- rmvnorm(n, mean = rep(0,p), sigma = diag(p))
data    <- t(apply(data, 1, function(z){runif(1)^(1/p)*z/sqrt(sum(z^2))}))
y       <- apply(data,1, function(z){sqrt(sum(z^2))})
histogram(~y,nint=100, ylab="",xlab="")
```

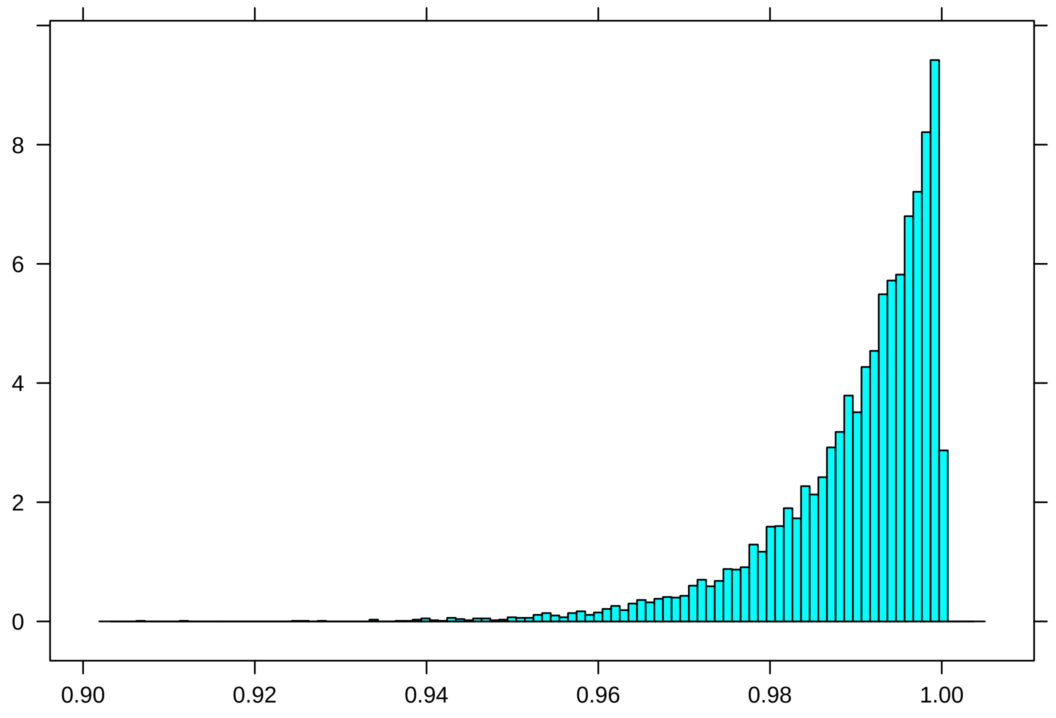

Figure 2.7: Distribution of distances from origin in 100 dim hypersphere

The histogram shows that the distance from the point to the center of the sphere is very likely to be close to 1. Hence if you use a nearest neighbor methods, the methods are no longer local.

3 Median distance of points from the origin

This section would be mainly about verifying the closed form solution for the median distance of points from the origin. The closed form solution is given in ESL.The problem goes like this :

> Consider N data points uniformly distributed in a p-dimensional unit ball centered at the origin. Suppose we consider a nearest-neighbor estimate at the origin. The median distance from the origin to the closest data point is given by the expression :

$$d(p, N) = \left(1 - 0.5^{1/N}\right)^{1/p}$$

If you know a bit of linear algebra, this expression can be easily verified. One can also simulate data and verify for a specific value of N and p. Let's say $N = 100$ and we want to plot the way in which the median distance varies as $p = 1 : 100$. The other good thing about using simulation is that we can also calculate mean distance without the need to compute the closed form solution which is very tedious. The following simulation verifies the above formula

```r
set.seed(1234)
n            <- 500
p            <- 1:50
nearest.m    <- (1-0.5^(1/n))^(1/p)
get.nearest <- function(p){
  data      <- rmvnorm(n, mean = rep(0,p), sigma = diag(p))
  data      <- t(apply(data, 1, function(z){runif(1)^(1/p)*z/sqrt(sum(z^2))}))
  res       <- apply(data,1,function(z){sqrt(sum(z^2))})
  ifelse(p==1, min(data^2),min(res))
}

sim.median   <- sapply(1:50, function(z){
                  quantile(replicate(100,get.nearest(z)),prob=0.5)
                  })

sim.avg      <- sapply(1:50, function(z){
                  mean(replicate(100,get.nearest(z)))
                  })

compares     <- data.frame(y1 = nearest.m[1:50],
                           y2=sim.median,
                           y3 = sim.avg, p = p[1:50])

xyplot(y1 + y2 + y3~p, data = compares, type=c("l","g"), ylim = c(0,1.1),
       xlab = "p", ylab="distance",
       auto.key = list(columns = 3,
          text = c("closed form","simulated median","simulated mean ")),
)
```

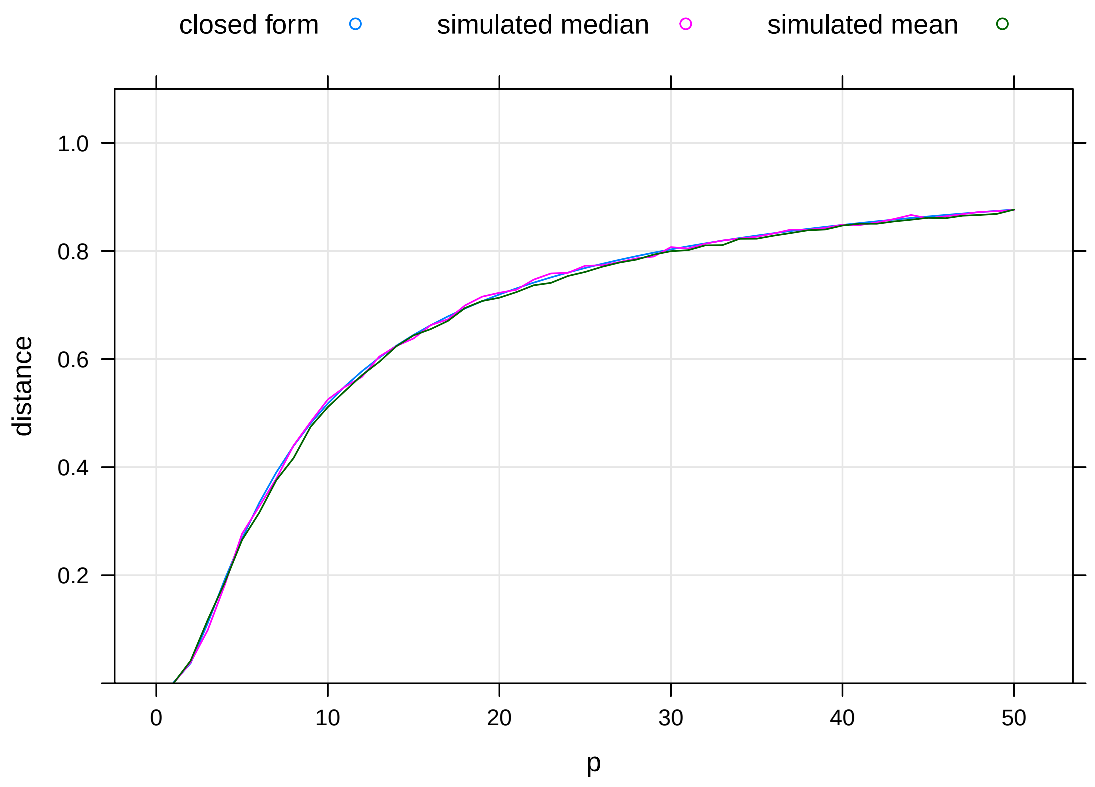

Figure 3.1: Mean and Median distance from the origin as p increases

4 Toy Example

In this section, I will work through a toy example given in ESL. Let's say the true data generating process for the predictor variable is

$$Y = f(X) = e^{-8||X||_2}$$

without any measurement error. We use the 1-nearest-neighbor rule to predict y_0 at the test point $x_0 = 0$.

```r
set.seed(1234)
n              <- 1000
p              <- 1:10
get.nearest    <- function(p){
  data         <- matrix(runif(n*p,-1,1), nrow = n, ncol = p)
  y            <- exp(-8*apply(data,1, function(z){(sum(z^2))}))
  res          <- apply(data,1,function(z){sqrt(sum(z^2))})
  min.point    <- ifelse(p==1, which.min(data^2),which.min(res))
  y[min.point]
}

sim.result     <- sapply(p, function(z){
                    temp          <- replicate(100,get.nearest(z))
                    variance      <- var(temp)
```

```
                  squared.bias <-(mean(1-temp))^2
                  mse          <- variance + squared.bias
                  return(c(squared.bias,variance, mse))
                  })

bias.var          <- as.data.frame(t(sim.result))
names(bias.var)   <- c("squared.bias","variance", "mse")
mtheme            <- standard.theme("pdf", color=TRUE)
mtheme$superpose.line$lwd <- 2
xyplot( squared.bias + variance + mse~p, data = bias.var,
       type=c("l","g"), ylim = c(0,1.1),
       xlab = "p", ylab="error",lwd=2,
       par.settings=mtheme,
       auto.key = list(columns = 3, points = FALSE,
                  lines = TRUE,
                  text = c("squared bias","variance","mse")),
)
```

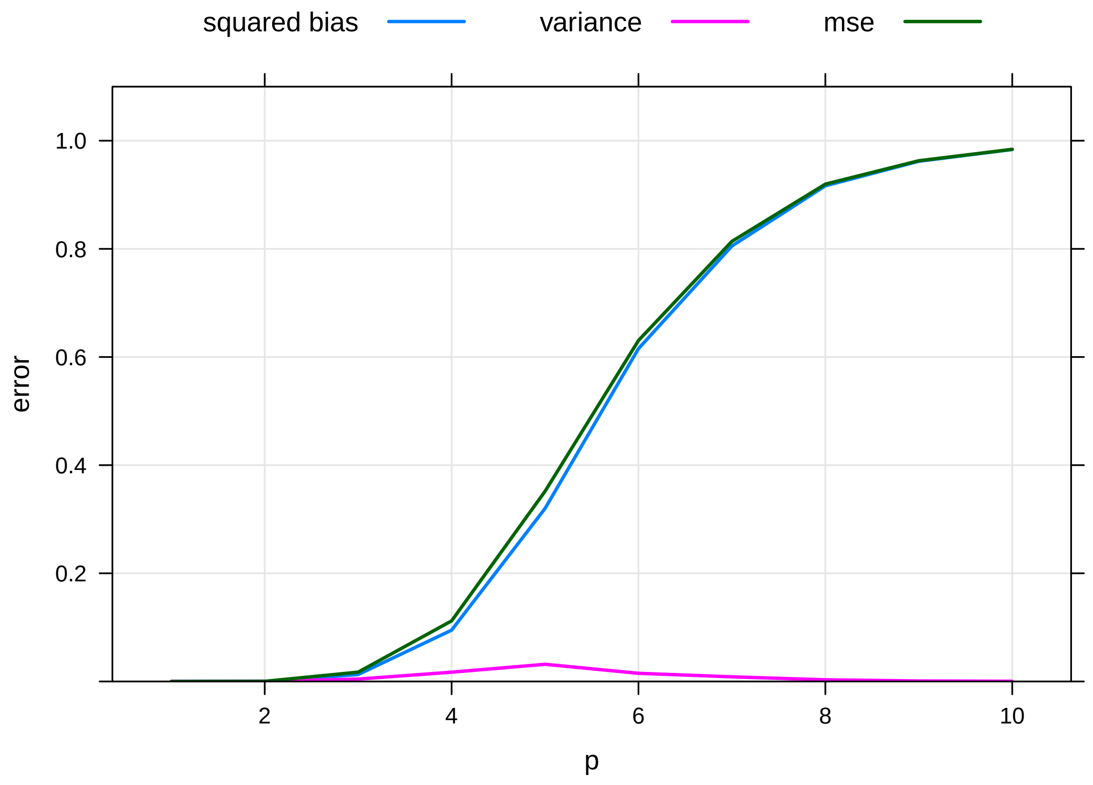

Figure 4.1: Squared Bias, Variance and MSE of 1 nearest neighbour method

The toy examples shows the MSE levels off to 1 and bias squared also levels off to 1, the maximum error possible. This toy example clearly shows that one must be careful in using nearest neighbor methods in high dimensional cases.

5 Takeaways

Our intuition does not serve well in high dimensional spaces. Hence there are few issues with using nearest neighbor methods on high dimensional data. Firstly, the methods that involve capturing a fixed neighborhood around the points gives high variance for the fit. Secondly, if you relax the fixed neighborhood criterion and try to capture a specific number of neighbors, the methods are no longer local. Hence it pays to think through these issues on whatever dataset you are working on. You might expect low variance fit but the curse of dimensionality shows up and you get a high variance fit.