CURSE OF DIMENSIONALITY

- After overfitting, the biggest problem in machine learning is the curse of dimensionality. Many algorithms that work fine in low dimensions become intractable when the input is high dimensional. This is what makes machine learning both necessary and hard.
- This report illustrates all the problems of CoD and also discusses some techniques to solve these problems. To run the code in this report interactively and see the results for yourself, run the R Notebook CoD.Rmd.

DATA SPARSITY

- One major problem of CoD is that even with a moderate dimension of 100 and a huge training set of a trillion examples, the latter covers only a fraction of about 10^−18 of the input space. The following code snippet illustrates how sparsity of data increases as the number of dimensions increase. If we consider outliers as those points in an unit hyper-sphere which are present in the 1% outer shell of the space, the plot of fraction of outliers vs. no. of dimensions is as in FIGURE 1 below.

```
dimension_list <- 1:50
N <- 1000
data_list <- lapply(dimension_list, function(dimensions) replicate(dimensions, runif(N)))
num_outliers <- function(data){
        mean(apply(data, 1, function(co_ordinates) any(co_ordinates < 0.01 | co_ordinates > 0.99)))
}
result <- sapply(data_list, num_outliers)
plot(result, type = "b",
xlab = "Number of Dimensions",
```

```
ylab = "Proportion of 1% outliers",
main = "Sparsity of Data")
grid()
```

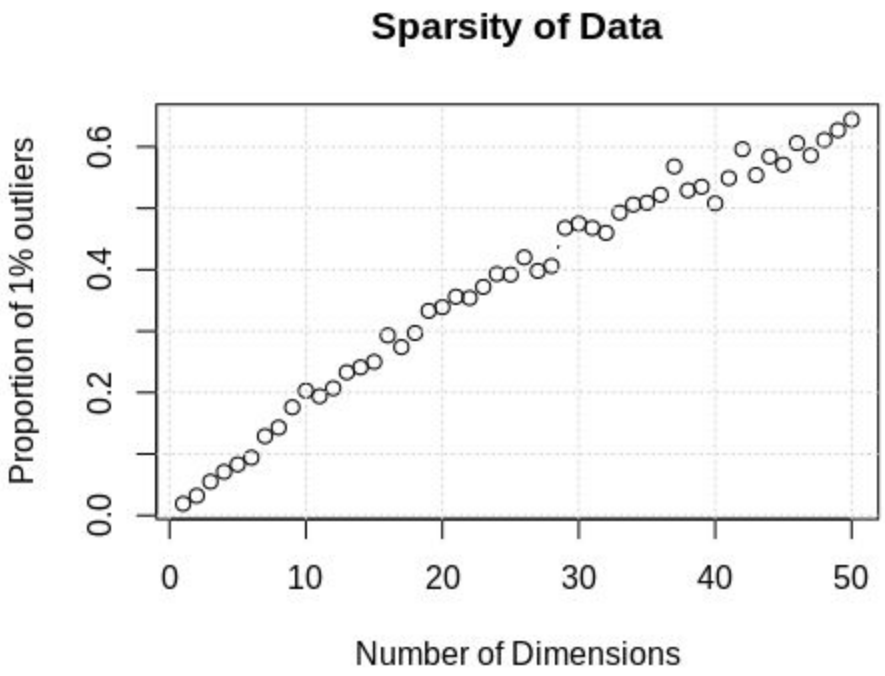

FIGURE 1

- Thus, our intuitions, which come from a three-dimensional world, often do not apply in high-dimensional ones. In high dimensions, most of the mass of a multivariate Gaussian distribution is not near the mean, but in an increasingly distant “shell” around it; and most of the volume of a high-dimensional orange is in the skin, not the pulp. If a constant number of examples is distributed uniformly in a high-dimensional hyper-cube, beyond some dimensionality most examples are closer to a face of the hyper-cube than to their nearest neighbor.

CoD AND kNN

- The similarity based reasoning that machine learning algorithms depend on like a kNN break down at high dimensions. Suppose the class is just x1 ∧ x2. If there are no other features, this is an easy problem. But if there are 98 irrelevant features x3,…, x100, the noise from them completely swamps the signal in x1 and x2, and the nearest neighbor effectively makes random predictions.
- Another problem with high-dimensional data is that, the time complexity of the learning algorithm becomes really huge. Most of the time, the growth is exponential to the number of dimensions.
- The effect of CoD on a kNN classifier can be seen in FIGURE 2 and FIGURE 3 below. We classify on a dataset with different numbers of dimensions. The dimensions are obtained from a dimensionality reduction technique called Principal Components Analysis (PCA). How PCA works is explained later. The graphs below illustrate clearly how the accuracy of kNN decreases with increasing no. of dimensions as well as how the time taken increases. (The code below will take a while to execute. It took around 2 minutes on our machine).

```
data <- read.csv("audit_risk.csv",colClasses = "numeric")
times_taken <- rep(0, ncol(data)-2)
accuracies_pca <- rep(0, ncol(data)-2)

for(iter in 3:ncol(data)-1){
        for(num in 1:50){
start.time <- Sys.time()

nor <-function(x) {
        if(max(x) != min(x)){
```

```
        x = (x -min(x))/(max(x)-min(x))
}
else{
        x = x-x
}
x
}

data_norm <- as.data.frame(lapply(data[,1:ncol(data)-1], nor))

ran <- sample(1:nrow(data), 0.9 * nrow(data))
mypca <- princomp(data_norm, cor=FALSE, score=TRUE)
data_norm = data.frame(mypca$scores)
v <- 1:iter
data_norm = data_norm[, v]

data_train_pca <- data_norm[ran,]
data_test_pca <- data_norm[-ran,]

data_target_category <- data[ran,ncol(data)]
data_test_category <- data[-ran,ncol(data)]

library(class)

pr_pca <- knn(data_train_pca,data_test_pca,cl=data_target_category,k=250)

tab_pca <- table(pr_pca,data_test_category)
```

```
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
accuracies_pca[iter-1] = accuracies_pca[iter-1] + accuracy(tab_pca)

end.time = Sys.time()
times_taken[iter-1] = times_taken[iter-1] + end.time - start.time
}

}

times_taken = times_taken/50
accuracies_pca = accuracies_pca/50

plot(3:ncol(data)-1,times_taken, type = "b",
xlab = "Number of dimensions",
ylab = "Average time taken in seconds",
main = "Effect of CoD on time taken in PCA + kNN")

plot(3:ncol(data)-1,accuracies_pca, type = "b",
xlab = "Number of dimensions",
ylab = "Average accuracy",
main = "Effect of CoD on accuracy in PCA + kNN")

grid()
```

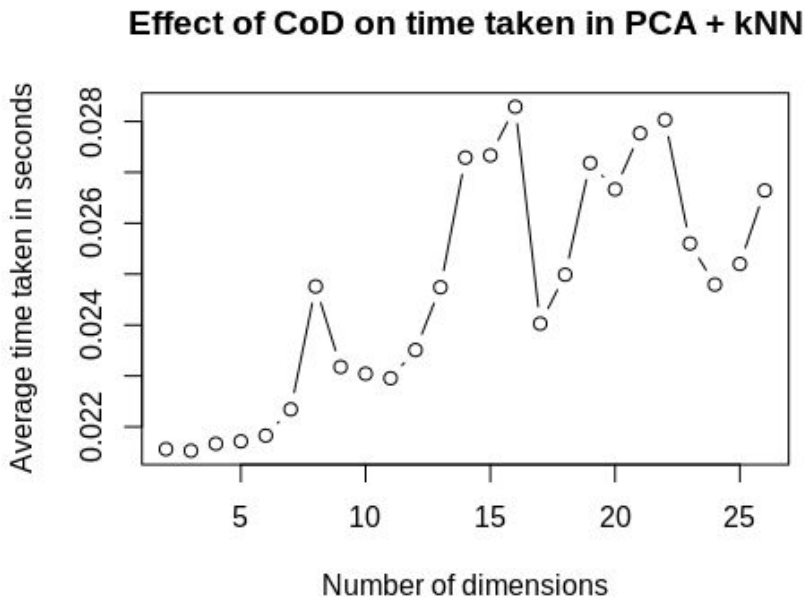

FIGURE 2

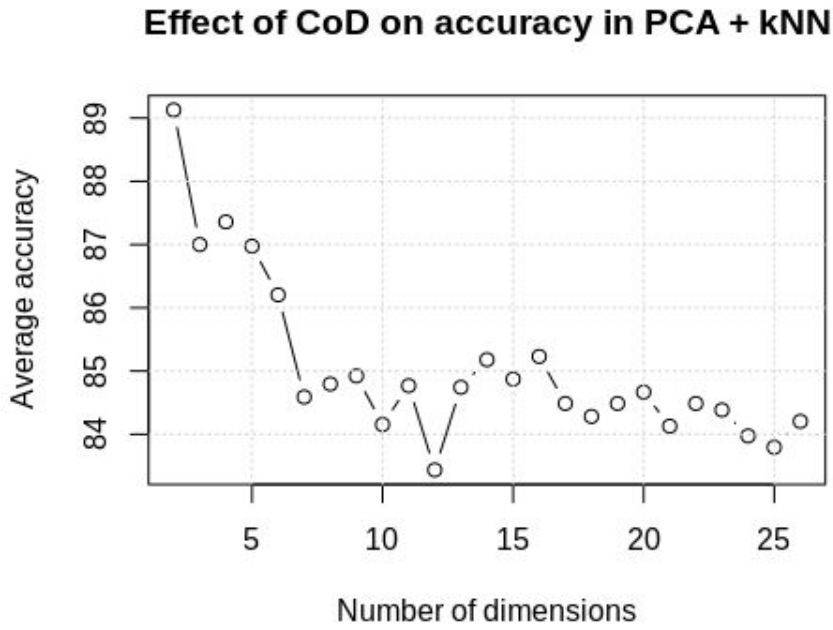

FIGURE 3

- Hence, it is always desirable to reduce the no. of dimensions when it is very high. But, care should also be taken that we don't reduce too much, else, a lot of information may be lost.

PCA VS. FLD

- Principal Component Analysis(PCA) is one of the most popular linear dimension reduction techniques. Sometimes, it is used alone and sometimes as a starting solution for other dimension reduction methods. It is a projection based method which transforms the data by projecting it onto a set of orthogonal axes. In the context of PCA, the different evaluated eigenvectors of the covariance matrix of the dataset represent a direction or axis which is a linear combination of pre-existing axes. The eigenvalue corresponding to each eigenvector represents the degree of variance of dataset entries along the new direction. Hence we reduce the number of dimensions involved by only choosing that we believe can contribute significantly to classification. The component of the current data point in the direction of the new dimension can be obtained by the dot product between it and the eigenvector.
- Though a very useful technique, it falls short when our primary aim is classification of the dataset. In such cases, another technique called the Fisher's Discriminant Analysis (FDA) is preferred. FDA also tries to find linear combinations of features like PCA, but unlike PCA whose aim is to find the direction of maximum variance, FDA's goal is to maximize separation between different classes in the data. It tries to find the hyperplane which maximises the ratio between variability of inter-class separation and intra-class seperation.

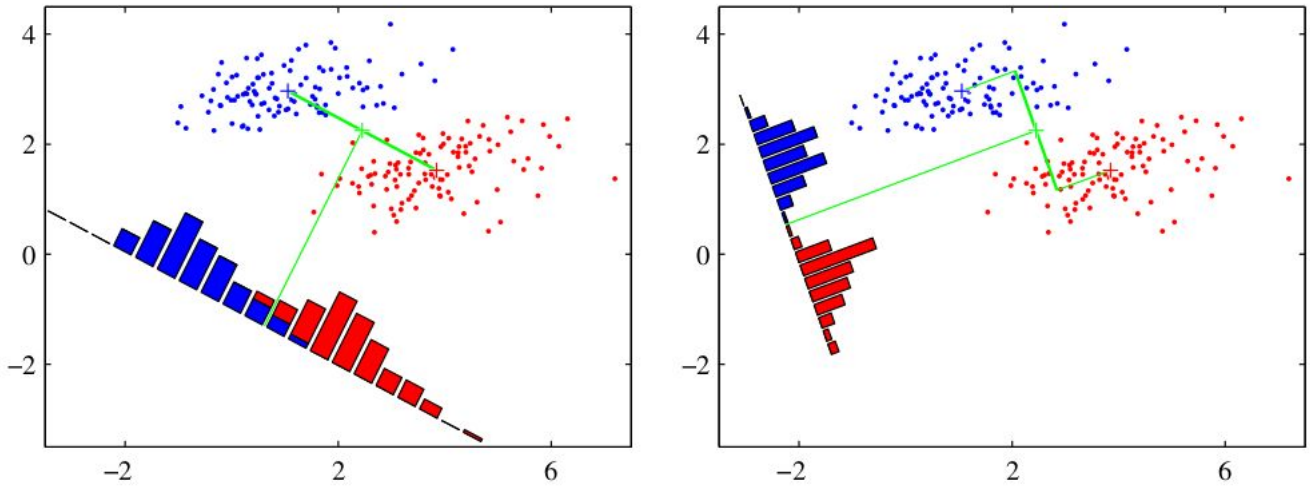

Figure 4.6 The left plot shows samples from two classes (depicted in red and blue) along with the histograms resulting from projection onto the line joining the class means. Note that there is considerable class overlap in the projected space. The right plot shows the corresponding projection based on the Fisher linear discriminant, showing the greatly improved class separation.

FIGURE 4

- Thus, FDA naturally is a better dimensionality reduction technique for classification purposes. (See Figure 4). The working of FDA is illustrated below: (The code below will take a while to execute. It took around 2 minutes on our machine).

```
library("lfda")
data <- read.csv("audit_risk.csv",colClasses = "numeric")
times_taken <- rep(0, ncol(data)-2)
accuracies_fld <- rep(0, ncol(data)-2)

for(iter in 3:ncol(data)-1){
        start.time <- Sys.time()
```

```
        nor <-function(x) {
        if(max(x) != min(x)){
        x = (x -min(x))/(max(x)-min(x))
        }
        else{
        x = x-x
        }
        x
}
kernel <- kmatrixGauss(data[, 1:ncol(data)-1])
targets <- data[, ncol(data)]
model <- klfda(kernel, targets, iter, metric = "plain")
transformed <- data.frame(model$Z)

ran <- sample(1:nrow(data), 0.9 * nrow(data))
data_train_fld <- transformed[ran,]
data_test_fld <- transformed[-ran,]

data_target_category <- data[ran,ncol(data)]
data_test_category <- data[-ran,ncol(data)]

library(class)
pr_fld <- knn(data_train_fld,data_test_fld,cl=data_target_category,k=250)
tab_fld <- table(pr_fld,data_test_category)
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
accuracies_fld[iter-1] = accuracies_fld[iter-1] + accuracy(tab_fld)
```

```
end.time = Sys.time()
times_taken[iter-1] = times_taken[iter-1] + end.time - start.time
}

plot(3:ncol(data)-1,times_taken, type = "b",
xlab = "Number of dimensions",
ylab = "Average time taken in seconds",
main = "Effect of CoD on time taken in FLD + kNN")

plot(3:ncol(data)-1,accuracies_fld, type = "b",
xlab = "Number of dimensions",
ylab = "Average accuracy",
main = "Effect of CoD on accuracy in FLD + kNN")
grid()
```

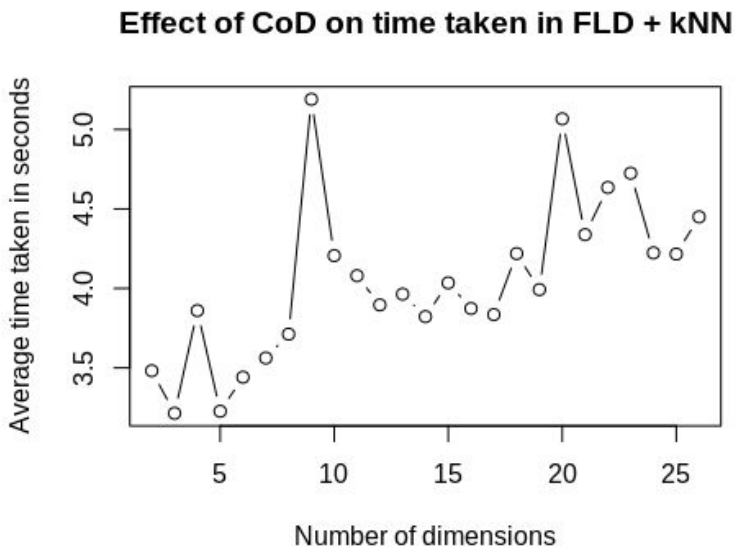

FIGURE 5

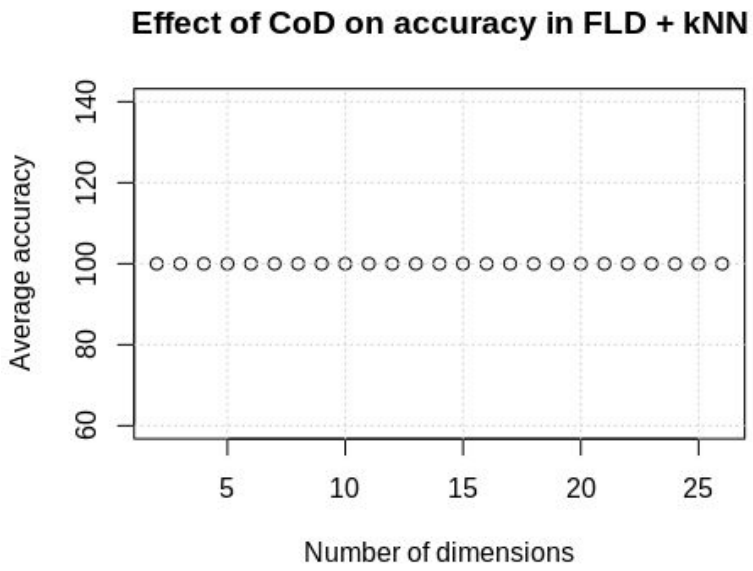

FIGURE 6

- We can see that FLD achieves 100 per-cent accuracy on the same dataset (FIGURE 6) while accuracy of PCA varied around 85-90 percent (FIGURE 3). This clearly shows how FLD is a much better technique for classification. We can also see how time taken increases with no. of dimensions above.

CoD AND OVERFITTING

- An ML Model trained with an increased number of parameters also tends to grow very dependent on training dataset and in turn overfits. This effect is illustrated in the code below, which tries to fit a multivariate regression model on a dataset. The graph below (FIGURE 7) also shows how even multivariate regression takes more time as the number of dimensions increases. (The code below will take a while to execute. It took around 3-4 minutes on our machine).

```
library("Metrics")
data <- read.csv("default_plus_chromatic_features_1059_tracks.csv",colClasses = "numeric")
times_taken <- rep(0, ncol(data)-2)
errors_train <- rep(0, ncol(data)-2)
errors_test <- rep(0, ncol(data)-2)

for(iter in 3:ncol(data)-1){
        for(num in 1:50){
        start.time <- Sys.time()

        nor <-function(x) {
        if(max(x) != min(x)){
        x = (x -min(x))/(max(x)-min(x))
        }
        else{
        x = x-x
        }
        x
        }

        data_norm <- as.data.frame(lapply(data[,1:ncol(data)-1], nor))

        ran <- sample(1:nrow(data), 0.9 * nrow(data))
        mypca <- princomp(data_norm, cor=FALSE, score=TRUE)
        data_norm = data.frame(mypca$scores)
        v <- 1:iter
        data_norm = data_norm[, v]
```

```
data_train <- data_norm[ran,]
data_test <- data_norm[-ran,]

attributes = colnames(data_train)
data_train$target__ <- data[ran,ncol(data)]
formula_str <- paste("target__", "~", paste(attributes, collapse = " + "))
model <- lm(formula_str, data = data_train)

RSS <- c(crossprod(model$residuals))
MSE <- RSS / length(model$residuals)
error_train <- sqrt(MSE)

pr <- predict(model, data_test)

error_test <- rmse(data[-ran,ncol(data)], pr)
errors_train[iter-1] = errors_train[iter-1] + error_train
errors_test[iter-1] = errors_test[iter-1] + error_test

end.time = Sys.time()
times_taken[iter-1] = times_taken[iter-1] + end.time - start.time
}
}
times_taken = times_taken/50
errors_train = errors_train/50
errors_test = errors_test/50
```

```
plot(3:ncol(data)-1,times_taken, type = "b",
xlab = "Number of dimensions",
ylab = "Average time taken in seconds",
main = "Effect of CoD on time complexity in PCA + Multi-variate Linear Regression")

plot(3:ncol(data)-1,errors_train, type = "b",
xlab = "Number of dimensions",
ylab = "Average RMSE Error",
main = "Effect of CoD on training set RMSE error in PCA + Multi-variate Linear Regression")

plot(3:ncol(data)-1,errors_test, type = "b",
xlab = "Number of dimensions",
ylab = "Average RMSE Error",
main = "Effect of CoD on test set RMSE error in PCA + Multi-variate Linear Regression")
grid()
```

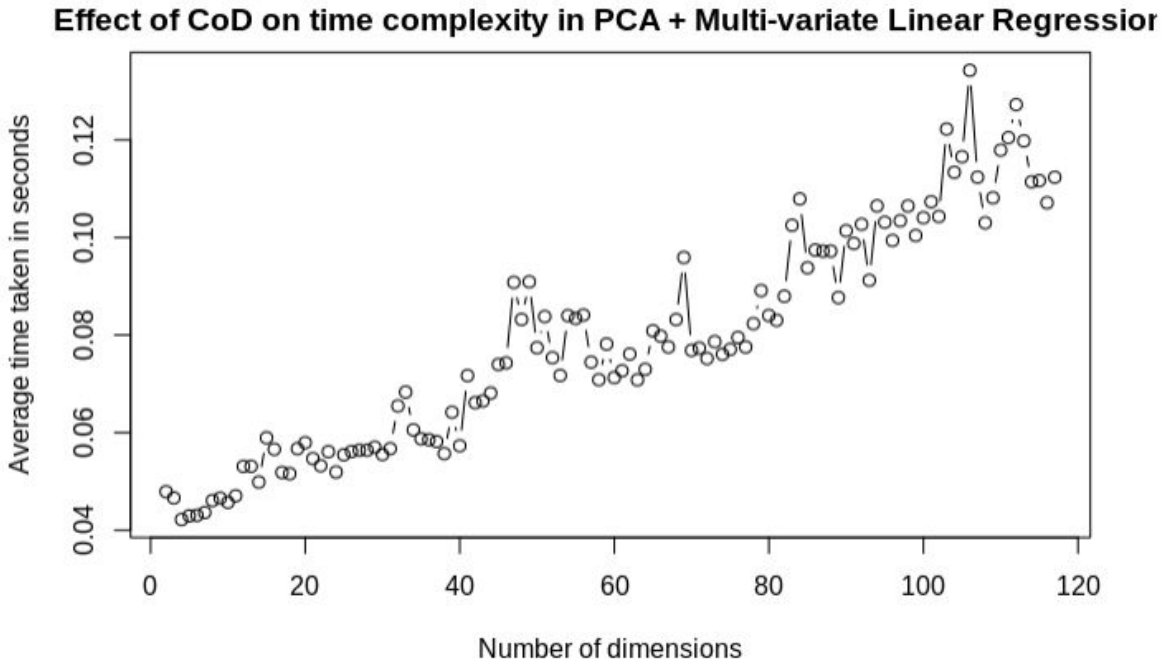

FIGURE 7

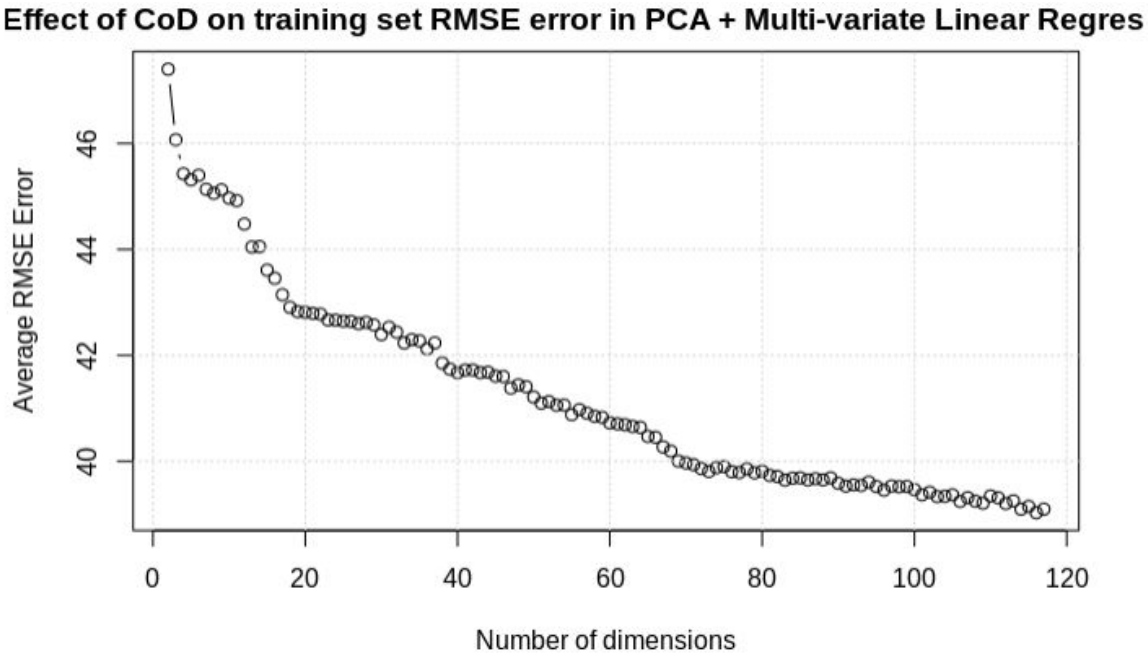

FIGURE 8

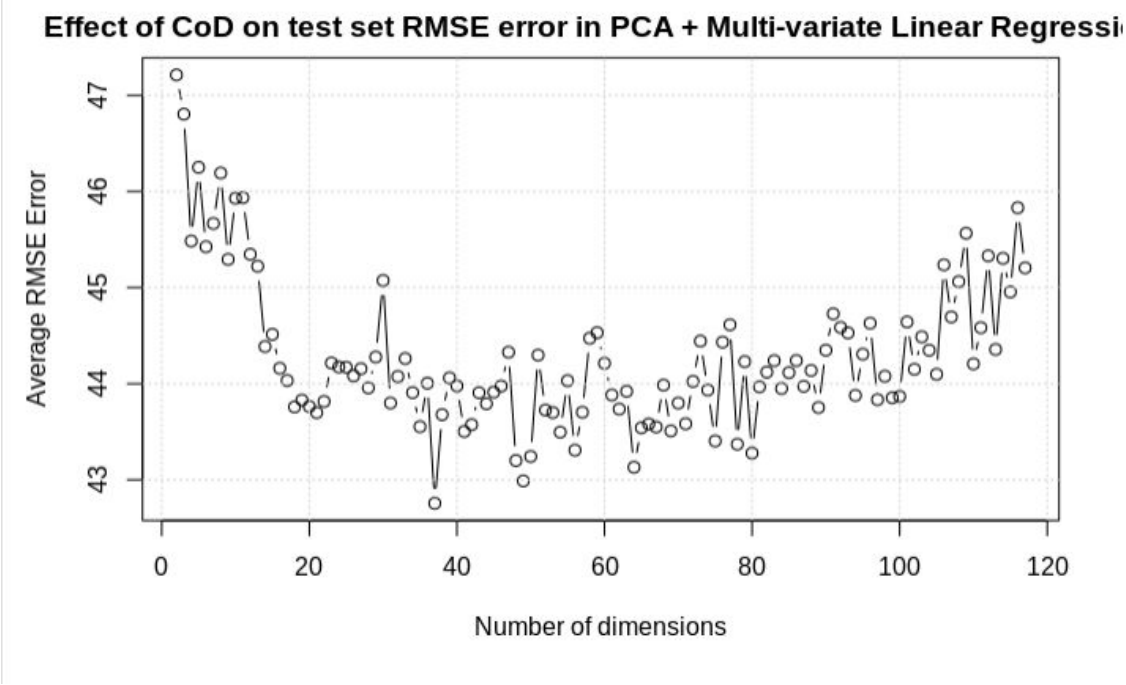

FIGURE 9

- As can be seen above, the model starts overfitting after around 80 dimensions. (The test set error goes on increasing (FIGURE 9) while the training set error goes on decreasing (FIGURE 8)). Generalizing correctly becomes exponentially harder as the dimensionality of the examples grows, because a fixed-size training set covers a dwindling fraction of the input space.

HUB PROBLEM AND DISTANCE CONCENTRATION

- Another interesting problem of CoD is the “Hub Problem”. In the finite case, some points are expected to be closer to the center than other points and are at the same time

closer, on average, to all other points. Such points closer to the center have a high probability of being hubs, i.e. of appearing in nearest neighbor lists of many other points.

- “Distance Concentration” or “Concentration of Lp Norms" is another surprising characteristic, where all points in a high dimensional space are at almost the same distance to all other points in that space (This phenomenon was already discussed in the initial parts of this report). As can be clearly seen, “Distance Concentration” and the “Hub Problem” are two closely related problems of CoD. Such problems can only be resolved by avoiding using the Euclidean distance metric and trying to come up with a new distance metric. Past research has shown that using fractional lp-Norms can mitigate this concentration effect to some extent. There also exists a class of algorithms called “Distance Metric Learning” algorithms, which try to automatically learn the distance metric from the dataset by trying to minimise distance between similar points and maximise distance between dis-similar points.