STAT 154 Lab 2: Cross-validation and Singular value decomposition

Yuansi Chen and Raaz Dwivedi

Feb 4, 2019

1 Cross-validation on simple estimators

In this problem, we will revisit the concept of cross-validation introduced in lectures. In particular, we will use the two simplest *covariate independent* estimators namely, mean and median.

1.1 Mean, median and data split

We first see the effect of mean and median on some toy data.

1. Define the mean estimator and median estimator for n data points $Y \in \mathbb{R}^n$. Note that for general data $(X, Y) \in \mathbb{R}^{(n \times d)} \times \mathbb{R}^n$, you can also see it as an estimator that completely ignores X.

2. Generate 100 i.i.d. standard normal random variable in R. Compute its mean and median.

```
n <- 100
x <- rnorm(n)
meanx <- mean(x)
medianx <- median(x)
meanx
```

```
## [1] -0.1333167
```

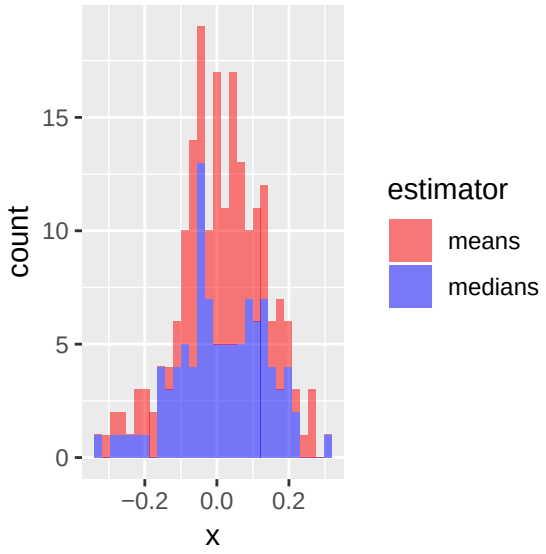
```
medianx
```

```
## [1] -0.1858159
```

3. Repeat this procedure 100 times and produce a histogram of means and medians on the same plot.

```
library(ggplot2)
repeats <- 100
means <- mat.or.vec(repeats ,1)
medians <- mat.or.vec(repeats ,1)
for(i in 1:repeats){
  x <- rnorm(n)
  means[i] <- mean(x)
  medians[i] <- median(x)
}
```

```
df <- data.frame(x=c(means, medians), label=c(rep("mean", repeats), rep("medians", repeats)))

ggplot(df, aes(x=x, fill=label)) + geom_histogram(alpha=0.5) +
  scale_fill_manual(name='estimator', values=c("red","blue"), labels=c("means", "medians"))
```



4. Generate 100 i.i.d. standard normal random variable in R. Split the data into two parts: 90% training set + 10% validation set (where the data is split randomly).

```
set.seed(123)

n <- 100
x <- rnorm(n)
smp_size <- floor(n*.9)

train_ind <- sample(seq(1:n), smp_size)
train <- x[train_ind]
val <- x[-train_ind]
```

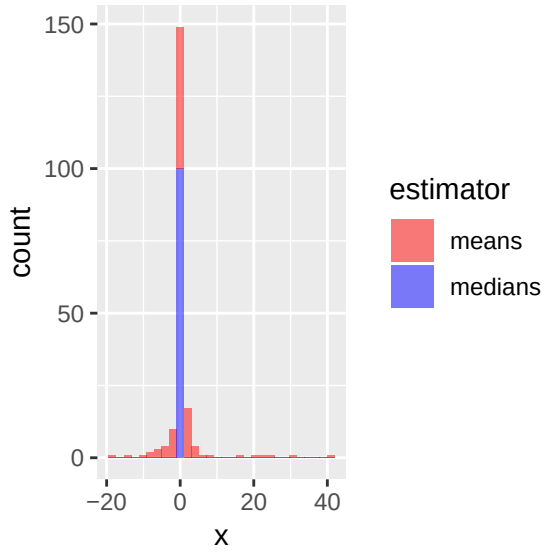
5. Estimate mean and median on the training set and evaluate their (a) mean squared error and (b) mean absolute error, on the validation set.

```
ests <- c(mean(train), median(train))
mse <- c(mean((val-ests[1])**2), mean((val-ests[2])**2))
l1err <- c(mean(abs(val-ests[1])**2), mean(abs(val-ests[2])))
ests

## [1] 0.08012553 0.06175631

mse

## [1] 0.4307724 0.4348867
```

```
l1err

## [1] 0.4307724 0.5839207
```

6. Repeat the previous parts with data drawn Cauchy distribution with scale parameter 1?

```
library(ggplot2)
repeats <- 100
means <- mat.or.vec(repeats ,1)
medians <- mat.or.vec(repeats ,1)
for(i in 1:repeats){
  x <- rcauchy(n)
  means[i] <- mean(x)
  medians[i] <- median(x)
}

df <- data.frame(x=c(means, medians), label=c(rep("mean", repeats), rep("medians", repeats)))

ggplot(df, aes(x=x, fill=label)) + geom_histogram(alpha=0.5) +
  scale_fill_manual(name='estimator', values=c("red","blue"), labels=c("means", "medians"))
```




```
set.seed(123)

n <- 100
x <- rcauchy(n)
smp_size <- floor(n*.9)

train_ind <- sample(seq(1:n), smp_size)
train <- x[train_ind]
val <- x[-train_ind]
```

```
ests <- c(mean(train), median(train))
mse <- c(mean((val-ests[1])**2), mean((val-ests[2])**2))
l1err <- c(mean(abs(val-ests[1])**2), mean(abs(val-ests[2])))
ests
```

```
## [1] 0.3651041 0.3049536
```

```
mse
```

```
## [1] 2.154294 2.084345
```

```
l1err
```

```
## [1] 2.154294 1.209262
```

1.2 Cross Validation

We now implement cross-validation. Via this exercise, we want you to understand the process. Note that eventually we will be using packages to implement CV. In this section of the problem, try to write functions for each part so that the last three parts are like a cakewalk.

1. Generate the Gaussian data as in the previous subsection. Split the data into 10 folds.

```
generate_folds <-function(x, K){
  #Randomly shuffle the data
  xshuffled <- x[sample(length(x))]

  #Create 10 equally size folds
  folds <- cut(seq(1,length(xshuffled)), breaks=K, labels=FALSE)

  return(list("folds"=folds, "xshuffled"=xshuffled))
}

n <- 100
x <- rnorm(n)
K <- 10
```

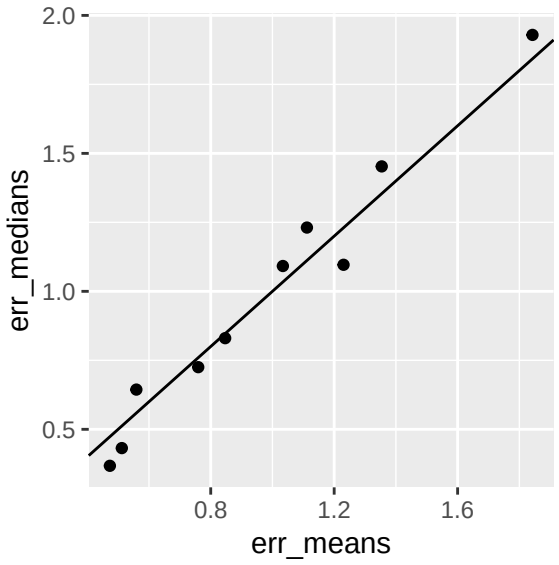
2. For each fold, estimate the mean and median on the rest 9 folds and evaluate the mean squared error on that fold.

```
#Perform 10 fold cross validation

compute_errors <- function(folds, xshuffled, K, pow=2){
  err_means <- 0*c(1:K)
  err_medians <- 0*c(1:K)
  for(i in 1:10){

      #Segement your data by fold using the which() function
      valIndexes <- which(folds==i, arr.ind=TRUE)
```

```
      valData <- xshuffled[valIndexes]
      trainData <-xshuffled[-valIndexes]

      err_means[i] <- mean(( abs(valData-mean(trainData)) )**pow)
      err_medians[i] <- mean(( abs(valData-median(trainData)) )**pow)
  }
  return(list("err_means"=err_means, "err_medians"=err_medians))
}

data <- generate_folds(x, K)
compute_errors(data$folds, data$xshuffled, K)

## $err_means
##  [1] 0.5590862 0.5119980 1.1117517 0.7595545 1.3541281 0.8468181 1.8425852
##  [8] 1.2303994 1.0337923 0.4732573
##
## $err_medians
##  [1] 0.6439156 0.4318631 1.2311626 0.7251662 1.4527709 0.8302774 1.9292685
##  [8] 1.0962430 1.0918059 0.3677728
```

3. Scatter plot the two errors and then compute the averaged mean squared error across 10 folds.

```
scatter_plot <- function(err_means, err_medians){
  ggplot() + geom_point(aes(x=err_means, y=err_medians)) + geom_abline(slope = 1, intercept = 0)
}
errors <- compute_errors(data$folds, data$xshuffled, K)
scatter_plot(errors$err_means, errors$err_medians)
```

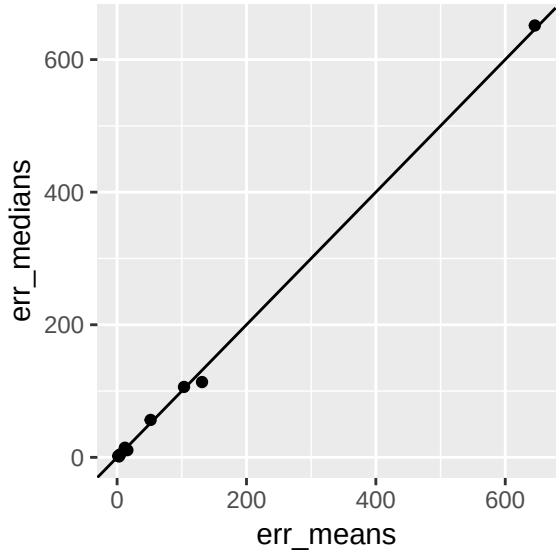



```
mean(errors$err_means)

## [1] 0.9723371
```

```
mean(errors$err_medians)

## [1] 0.9800246
```

4. Repeat the previous three parts for Cauchy data.

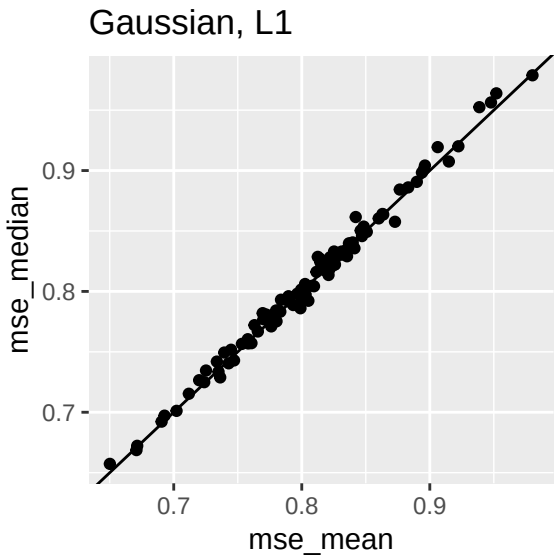
```
x <- rcauchy(n)
data <- generate_folds(x, K)
errors <- compute_errors(data$folds, data$xshuffled, K)
scatter_plot(errors$err_means, errors$err_medians)
```

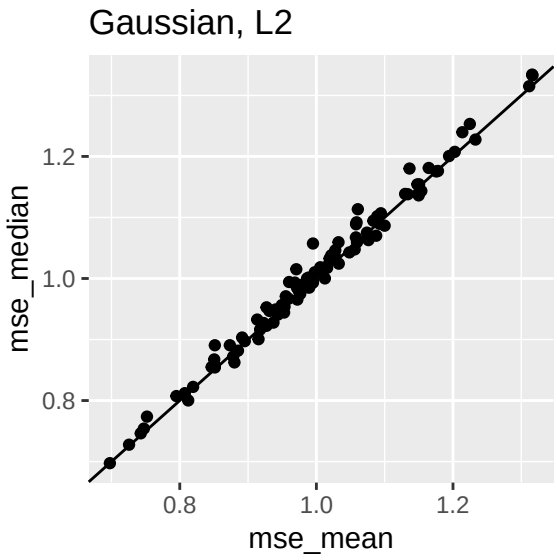


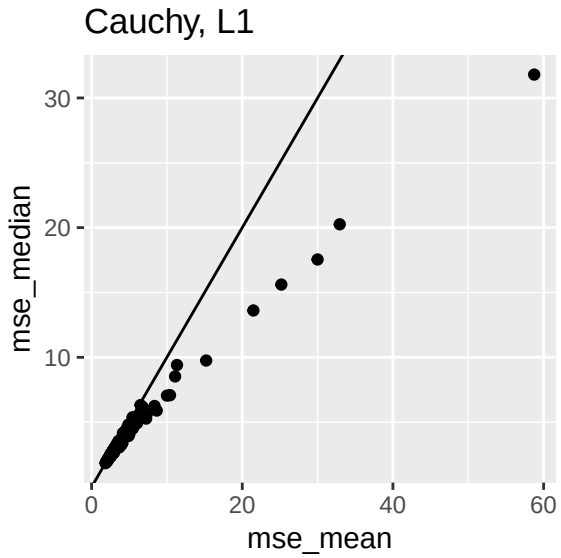
5. Repeat the data generation and estimation 100 times. Scatter plot them. Repeat the same with absolute error. What do you think CV might be doing better comparing to the single validation set approach?

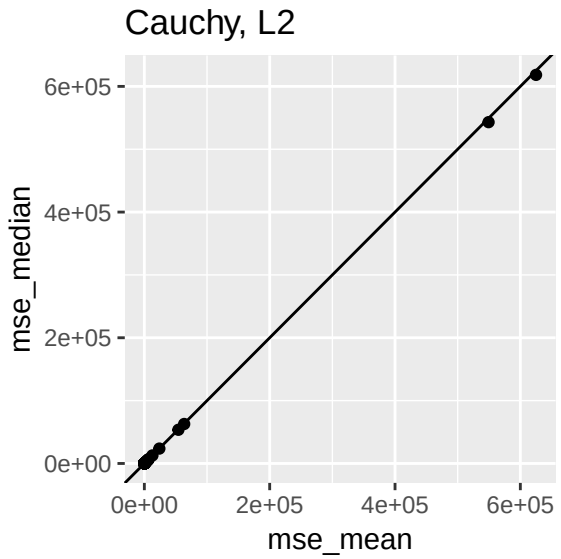
```
legends = c('Gaussian, L1', 'Gaussian, L2', 'Cauchy, L1', 'Cauchy, L2')
j = 1
for (fun in list(rnorm, rcauchy)){
for (pow in 1:2) {

  repeats <- 100
  mse_mean <- 0*c(1:repeats)  # mat.or.vec(repeats, 1)
  mse_median <- 0*c(1:repeats)

  for (i in 1:repeats){
    data <- generate_folds(fun(n), K)
    errors <- compute_errors(data$folds, data$xshuffled, K, pow)
    mse_mean[i] <- mean(errors$err_means)
    mse_median[i] <- mean(errors$err_medians)
  }
```

```
  p<- ggplot() + geom_point(aes(x=mse_mean, y=mse_median)) + geom_abline(slope = 1, intercept = 0)+
  print(p)
  j <- j+1
  }
}
```





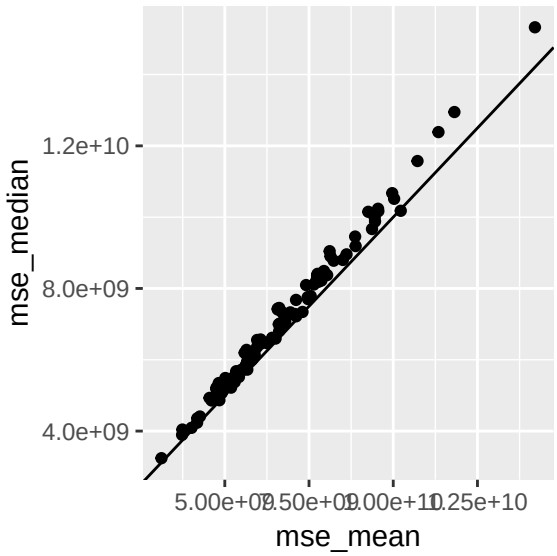




6. Download the "ames.csv" file from `https://www.openintro.org/stat/data/?data=ames`. Load in the file and use its SalePrice column as the data. Instead of 100 i.i.d. gaussian random variables, we now draw 100 independent samples from this column and use mean and median to estimate the data. Repeat the previous part with this data. Report your observations from the plot and discuss.

```
ames <- read.csv("ames.csv")
mse_mean <- 0*c(1:repeats)  # mat.or.vec(repeats, 1)
mse_median <- 0*c(1:repeats)
xdata <- ames$SalePrice
for (i in 1:repeats){
  data <- generate_folds(sample(xdata, n), K)
  errors <- compute_errors(data$folds, data$xshuffled, K)
  mse_mean[i] <- mean(errors$err_means)
  mse_median[i] <- mean(errors$err_medians)
```

```
}
ggplot() + geom_point(aes(x=mse_mean, y=mse_median)) + geom_abline(slope = 1, intercept = 0)
```



2 Singular value decomposition (SVD)

In the second part of the lab, you will learn about singular value decomposition (SVD) and its relevance in data analysis. [1]

2.1 Math of SVD

1. **Singular value**. Given a matrix $M \in \mathbb{R}^{(m \times n)}$ (assume $m \geq n$). The non-zero singular values of M correspond to the square roots of the non-zero eigenvalues of either $M^\top M$ or $MM^\top$.

2. **Singular value decomposition**. For real matrix in finite dimension, it is always to write M in the following decomped form

$$M = UDV^\top, \tag{1}$$

   where

   - U is an $m \times n$ matrix of left singular vectors.
   - D is a $n \times n$ diagonal matrix of singular values.
   - V is a $n \times n$ matrix of right singular vectors.

3. Given a singular value decompsition (1), show that D^2 correspond to the eigenvalues of the matrix $M^\top M$.

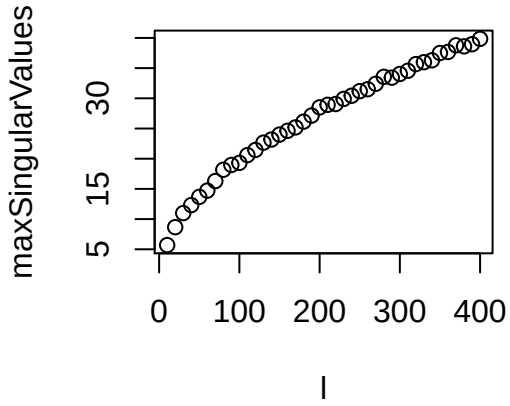
4. When do the singular values of a matrix equal to its eigenvalues?

5. Take a look at the **svd** function in R. What does this function return? Tell how R handles the case $m < n$.

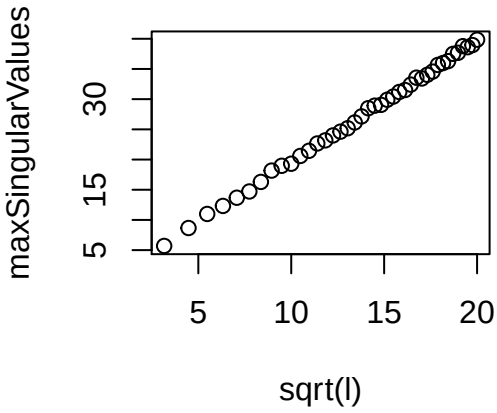
[1] The questions are based on the Lab 2 from Gaston Sanchez's Stat 154 in Spring 2018 (see link distributed under Creative Commons Attribution-ShareAlike 4.0 International License).

6. SVD for random Gaussian matrix. For $m \in \{10, 20, 30, ..., 390, 400\}$, generate a random matrix $M \in \mathbb{R}^{m \times m}$ with each entry drawn i.i.d. from standard normal distribution, compute its maximal singular value. First, plot the maximal singular value as a function of m. Second plot the maximal singular value as a function of $\sqrt{m}$. In which plot do you see a linear relationship? (No theoretical understanding is required in this exercise.)

```
l <- seq(1: 40) *10
maxSingularValues <- rep(0., length(l))
for(i in 1:length(l)){
  m = l[i]
  M <- matrix( rnorm(m*m,mean=0,sd=1), m, m)
  res <- svd(M)
  maxSingularValues[i] <- max(res$d)
}
plot(l, maxSingularValues)
```




```
plot(sqrt(l), maxSingularValues)
```



2.2 The relevance of SVD in data analysis

One of the most attractive uses of the SVD is that it allows you to decompose a matrix M as a sum of rank-one matrices of the form $d_k \mathbf{u}_k \mathbf{v}_k^\top$. This is the result of the famous Eckart-Young-Mirsky theorem (see Wikipedia), which says that the best rank r approximation to M is given by:

$$X_r = \sum_{k=1}^{r} d_k \mathbf{u}_k \mathbf{v}_k^\top,$$

where d_k is the k-th singular value.

What does it mean to have a data matrix that has only a few non-zero singular values?

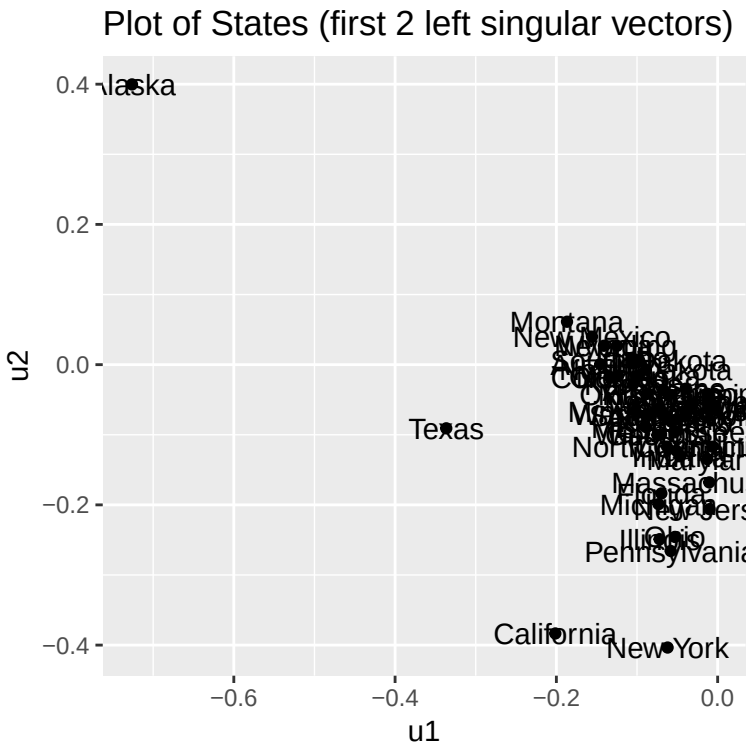
2.3 SVD on real data

we are going to use the data set state.x77 that comes in R—see ?state.x77 for more information. *state.x77* is part of a collection of data sets about "US State Facts and Figures".

1. Load data *state.x77*. How many rows and columns are there?
2. Create a matrix *state2* by selecting (i.e. subsetting) the first five columns of *state.x77*.
3. Run SVD on *state2*.
4. Compute the best 3-rank approximation of *state2*.
5. Consider the SVD decomposition of *state.x77*. Knowing that we can approximate *state.x77* with $d_1 \mathbf{u}_1 \mathbf{v}_1^\top + d_2 \mathbf{u}_2 \mathbf{v}_2^\top$, use $\mathbf{u}_1$ and $\mathbf{u}_2$ to visualize the States with a simple scatterplot. Create your own scatterplot like the one below:

```
X <- state.x77
rsvd <- svd(X)
Xu <- data.frame(rsvd$u[, 1:2])
rownames(Xu) <- rownames(X)
colnames(Xu) <- c("u1", "u2")
```

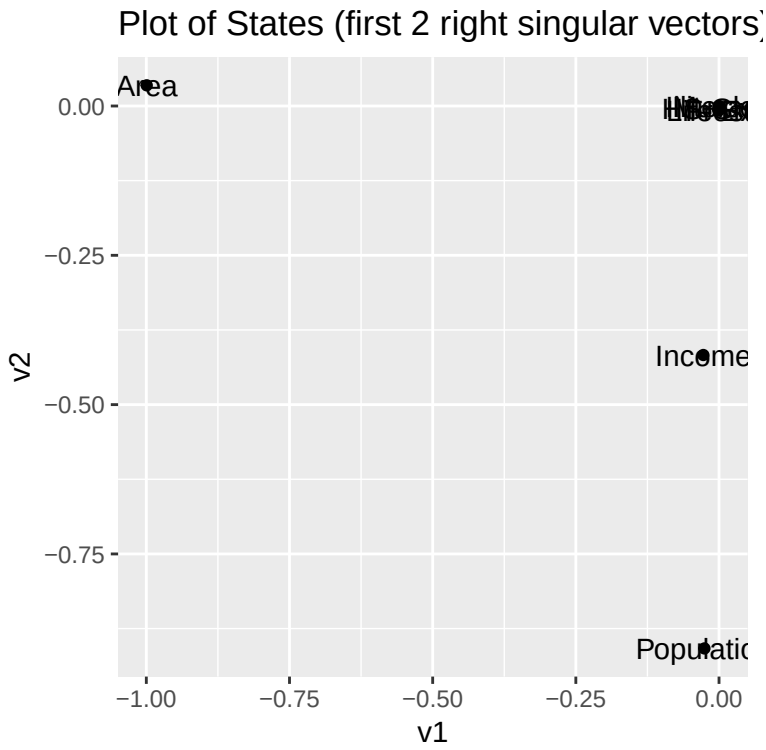
```
ggplot(Xu, aes(x=u1, y=u2)) +
  geom_point() +
  geom_text(label=rownames(Xu)) +
  labs(title="Plot of States (first 2 left singular vectors)")
```



Look at $\mathbf{u}_1$ and $\mathbf{u}_2$, tell what makes Alaska very different in the plot.

6. Consider the SVD decomposition of *state.x77*. In the same way, use the first two right singular vectors of V to graph a scatterplot (e.g. 2-dimensional representation) of the variables:

```
X <- state.x77
rsvd <- svd(X)
Xv <- data.frame(rsvd$v[, 1:2])
rownames(Xv) <- colnames(X)
colnames(Xv) <- c("v1", "v2")
ggplot(Xv, aes(x=v1, y=v2)) +
  geom_point() +
  geom_text(label=rownames(Xv)) +
  labs(title="Plot of States (first 2 right singular vectors)")
```



2.4 Computation of SVD

1. Read the section "Calculating the SVD" in Wikipedia Page on SVD. Answer the rough computational complexity for computing SVD of a matrix $M \in \mathbb{R}^{m \times n}$.

2. Test SVD on large random matrix in R and time it to get a sense the largest matrix R **svd** can deal with in less than 1 second. (see five ways to time a function in R.)

```
library(tictoc)
m <- 800
M <- matrix( rnorm(m*m,mean=0,sd=1), m, m)
tic("start computing")
res <- svd(M)
print("...stop")

## [1] "...stop"

toc()

## start computing: 1.761 sec elapsed
```