Math 611 HW6

Jeff Gould

10/8/2020

1. Time Series

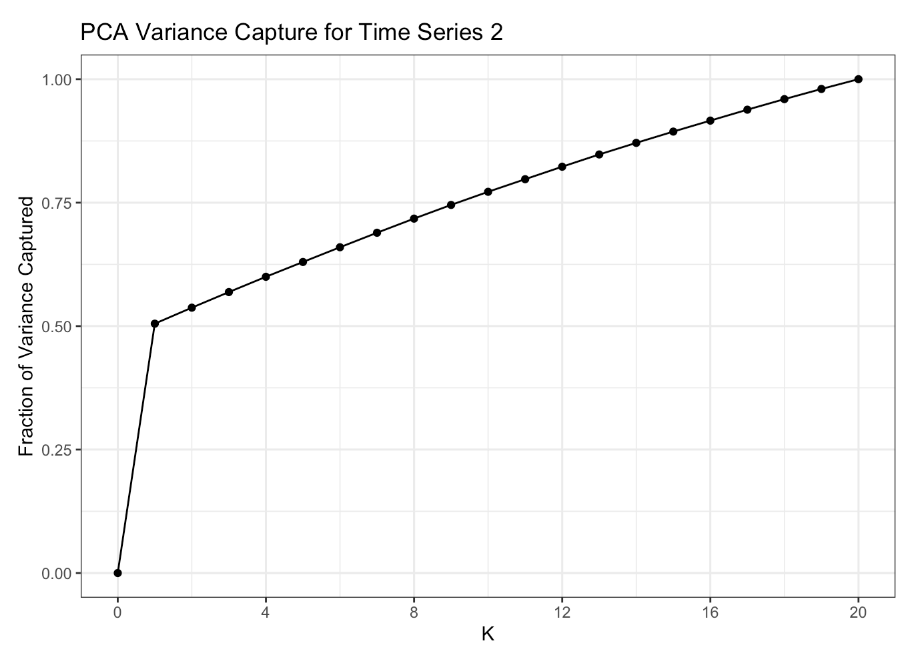
a)

As we have more variables, we will need to use more dimensions of PCA in order to capture the increase in variance. For `TimeSeries2`, since there are just two base series that we need to differentiate the variation between, we should be able to se K to either 1 or 2 and capture most of the variance. As we increase the number of base series to differentiate, we will likely need to increase K by a similar amount. So if we find $K = 1$ to be sufficient for `TimeSeries2`, then $K = 2$ should be sufficient for `TimeSeries3`, and $K = 3$ for `TimeSeries4`.

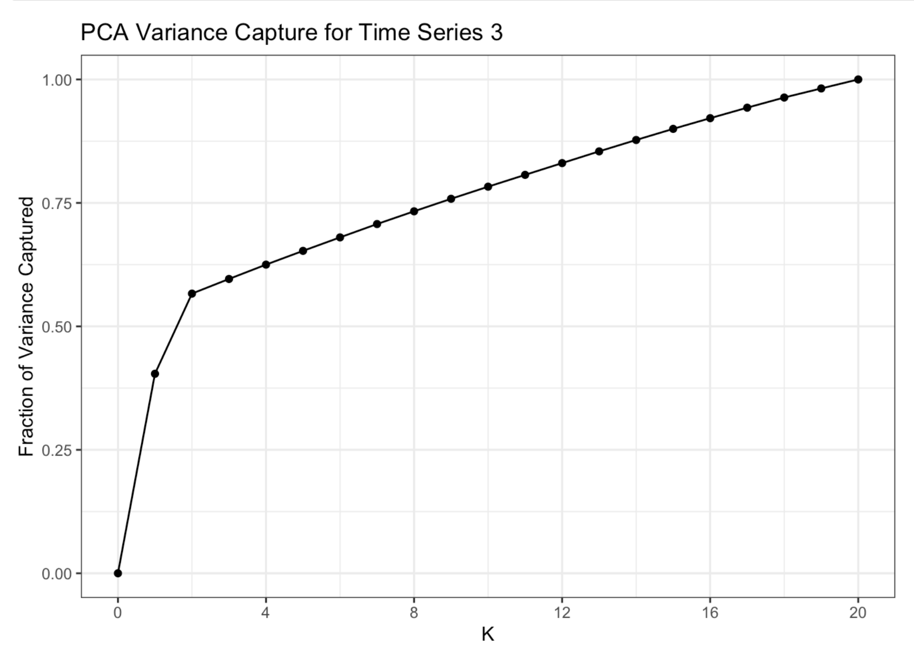
b)

i

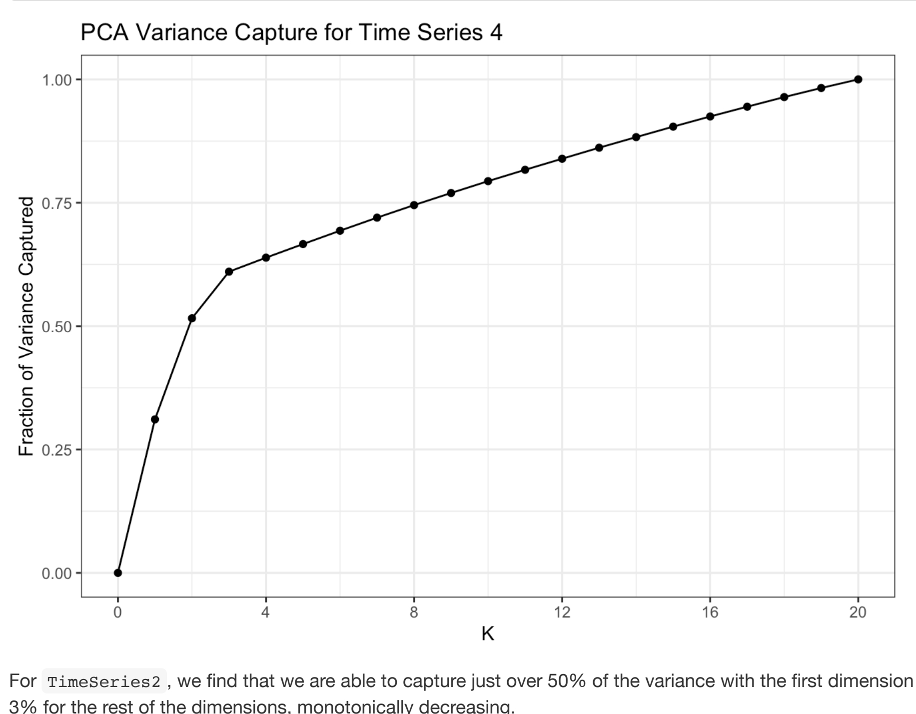
```
TimeSeries2 <- read_csv("TimeSeries_K2.csv")
TimeSeries3 <- read_csv("TimeSeries_K3.csv")
TimeSeries4 <- read_csv("TimeSeries_K4.csv")


makePCAvariancePlot <- function(timeSeries, plotTitle = NULL){
  ## Center data
  mu <- colMeans(timeSeries)
  centeredTS <- t(apply(timeSeries, 1, function(x){x-mu}))

  ## Calculate covariance matrix
  CovTS <- t(centeredTS) %*% centeredTS

  ## Compute eigendata, take eigen values
  eigVals <- eigen(CovTS)$values

  ## create dataframe of eigen values, percent variance captured, K
  percentCaptured <- data.frame(eigValue = c(0, eigVals)) %>%
    mutate(percentCaptured = cumsum(eigValue) / sum(eigValue),
           K = row_number() - 1)

  graph <- ggplot(data = percentCaptured, aes(x = K, y = percentCaptured)) +
    geom_point() +
    geom_line() +
    theme_bw() +
    scale_y_continuous(limits = c(0,1)) +
    scale_x_continuous(limits = c(0,20), breaks = seq(0,20,4)) +
    labs(x = "K", y = "Fraction of Variance Captured")

  if(!is.null(plotTitle)){
    graph <- graph + labs(title = plotTitle)
  }

  return(graph)

}

makePCAvariancePlot(TimeSeries2, "PCA Variance Capture for Time Series 2")
```




```
makePCAvariancePlot(TimeSeries3, "PCA Variance Capture for Time Series 3")
```




```
makePCAvariancePlot(TimeSeries4, "PCA Variance Capture for Time Series 4")
```



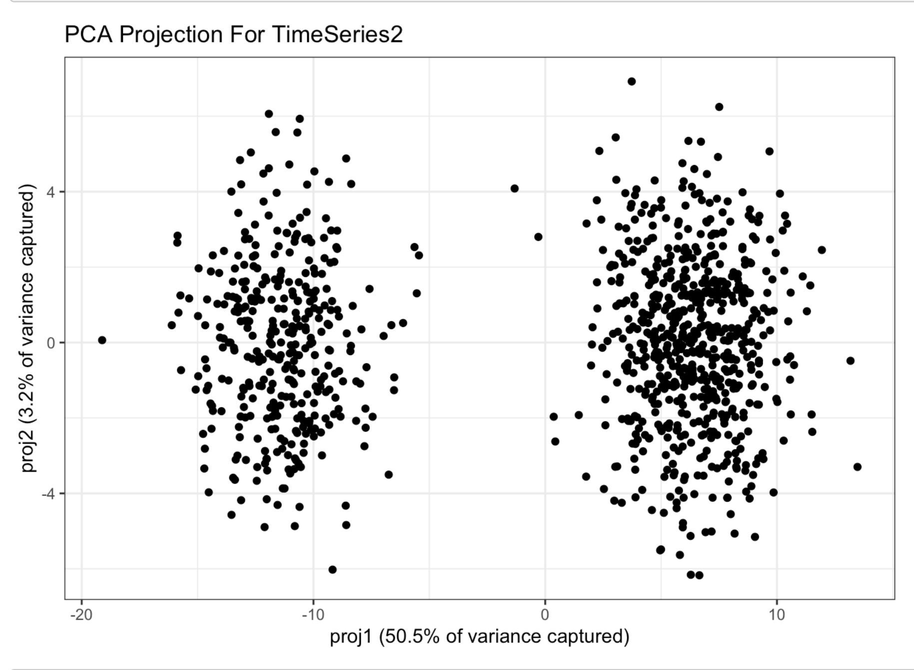
For `TimeSeries2`, we find that we are able to capture just over 50% of the variance with the first dimension of the PCA, and then gain around 2-3% for the rest of the dimensions, monotonically decreasing.

For `TimeSeries3`, we capture about 40% of the variance with the first dimension, around 57% after capturing the first two dimensions, and then around 2-3% for each additional dimension, monotonically decreasing.

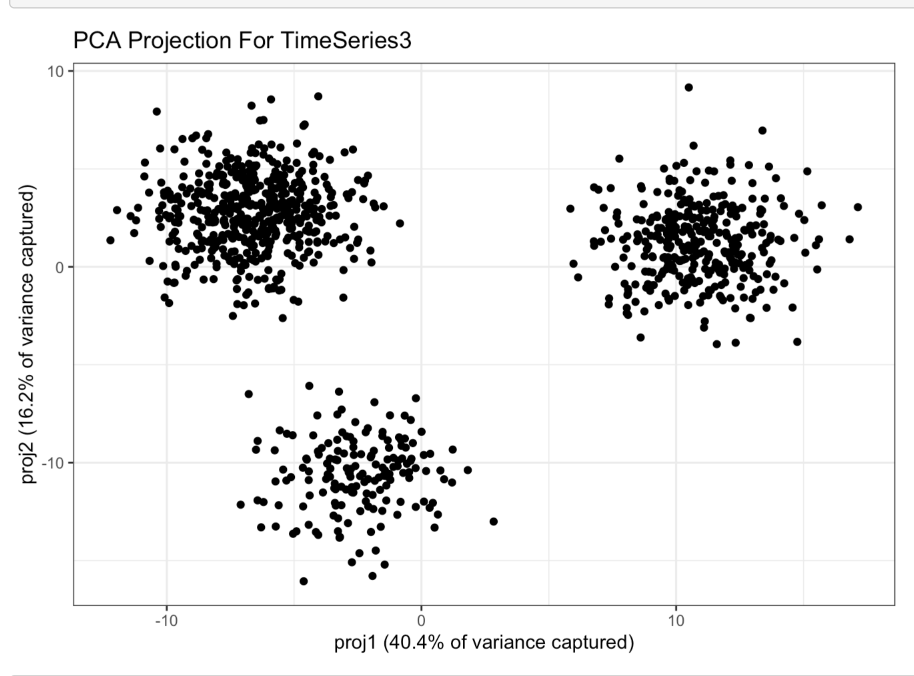
For `TimeSeries4` we capture 31% of the variance after the first dimension, 52% after the second dimension, and 61% after three dimensions. Again, we then gain about 2-3% per additional dimension, monotonically decreasing.

ii

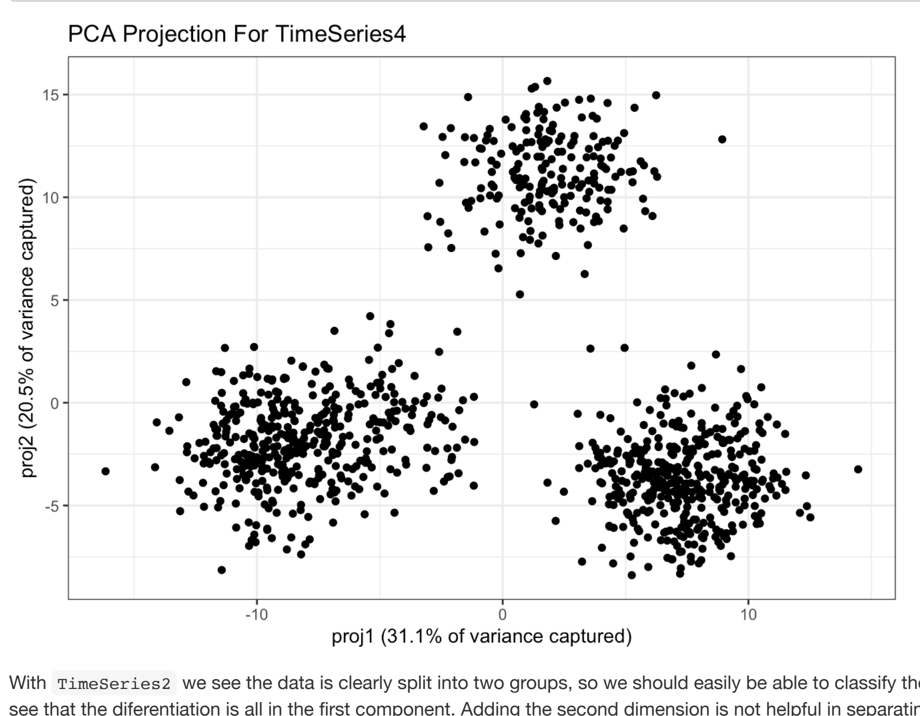
```
make2dPCAplot <- function(timeSeries, plotTitle = NULL){

  ## Center data
  ## Center data
  mu <- colMeans(timeSeries)
  centeredTS <- t(apply(timeSeries, 1, function(x){x-mu}))

  ## Calculate covariance matrix
  CovTS <- t(centeredTS) %*% centeredTS

  ## Compute eigendata, take eigenvectors
  ev <- eigen(CovTS)
  eigVec = ev$vectors
  eigVals = ev$values

  label1 <- glue::glue("proj1 ({round(100 * eigVals[1] / sum(eigVals),1)}% of variance captured)")
  label2 <- glue::glue("proj2 ({round(100 * eigVals[2] / sum(eigVals),1)}% of variance captured)")

  proj1 <- centeredTS %*% eigVec[,1]
  proj2 <- centeredTS %*% eigVec[,2]

  projectionPlot <- data.frame(proj1 = proj1,
                               proj2 = proj2)

  graph <- ggplot(data = projectionPlot, aes(x = proj1, y = proj2)) +
    geom_point() +
    theme_bw() +
    labs(x = label1, y = label2)

  if(!is.null(plotTitle)){graph <- graph + labs(title = plotTitle)}

  return(graph)



}

make2dPCAplot(TimeSeries2, plotTitle = "PCA Projection For TimeSeries2")
```




```
make2dPCAplot(TimeSeries3, plotTitle = "PCA Projection For TimeSeries3")
```




```
make2dPCAplot(TimeSeries4, plotTitle = "PCA Projection For TimeSeries4")
```



With `TimeSeries2` we see the data is clearly split into two groups, so we should easily be able to classify the data to one of the series. We also see that the diferentiation is all in the first component. Adding the second dimension is not helpful in separating the series. This also makes sense, as the first dimension captures 50% of the variance, and the second component only captures 3.2%

For `TimeSeries3`, the data is also grouped into three distinct clusters. We could can also group the series into three clusters pretty easily.

In `TimeSeries4`, we only get three clusters instead of 4, and the space between the clusters is not as distinct as in `TimeSeries2` and `TimeSeries3`. It looks as though the top cluster might be one series, the bottom right cluster may be one series, and the bottom left cluster may be a blend of the two other series, but it is impossible to say for certain. We would need to expand to three dimensions in order to better separate the clusters.

iii

$c_1^{(i)} = (x^{(i)} - \mu) \cdot q^{(1)}, c_2^{(i)} = (x^{(i)} - \mu) \cdot q^{(2)}$

```
#### TS2
mu <- colMeans(TimeSeries2)

centeredX <- t(apply(TimeSeries2, 1, function(x){x-mu}))
CovTS <- t(centeredX) %*% centeredX
ev <- eigen(CovTS)
q <- ev$vectors

c1 <- apply(centeredX, 1, function(x){sum(x * q[,1])})
c2 <- apply(centeredX, 1, function(x){sum(x * q[,2])})
cor(c1, c2)
```

```
## [1] 3.186462e-16
```

```
#### TS3
mu <- colMeans(TimeSeries3)

centeredX <- t(apply(TimeSeries3, 1, function(x){x-mu}))
CovTS <- t(centeredX) %*% centeredX
ev <- eigen(CovTS)
q <- ev$vectors

c1 <- apply(centeredX, 1, function(x){sum(x * q[,1])})
c2 <- apply(centeredX, 1, function(x){sum(x * q[,2])})
cor(c1, c2)
```

```
## [1] -6.072804e-16
```

```
sd(c1)^2
```

```
## [1] 66.63511
```

```
ev$values[1]
```

```
## [1] 66568.48
```

```
sd(c2)^2
```

```
## [1] 26.77718
```

```
ev$values[2]
```

```
## [1] 26750.41
```

```
#### TS4
mu <- colMeans(TimeSeries4)

centeredX <- t(apply(TimeSeries4, 1, function(x){x-mu}))
CovTS <- t(centeredX) %*% centeredX
ev <- eigen(CovTS)
q <- ev$vectors

c1 <- apply(centeredX, 1, function(x){sum(x * q[,1])})
c2 <- apply(centeredX, 1, function(x){sum(x * q[,2])})
cor(c1, c2)
```

```
## [1] 9.690308e-17
```

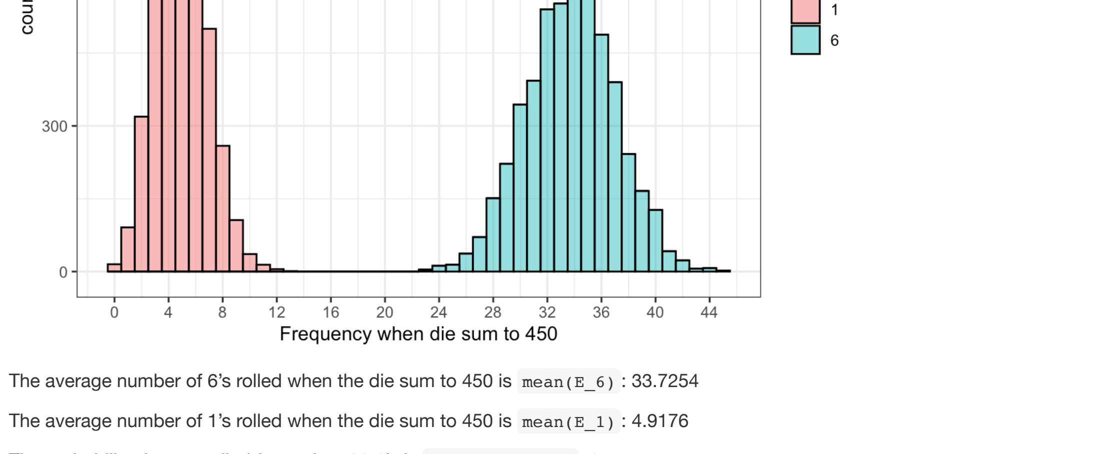
For each `TimeSeries`, the is no correlation between the $c_1^{(i)}, c_2^{(2)}$'s. The variance of $c_k^{(i)}$ for each time series corresponds to q^k, just scaled by 2 magnitudes. This is due to the normalization of the eigen data, so we can say that variance of $c_k^{(i)} = q^k$, before normalizing.

2. Roll 100 die

```
rollMH <- function(x){

  rolls <- sample(1:6, 100, replace = TRUE)

  i <- 1
  iterations <- 0

  while(sum(rolls) != x){

    new_sample <- sample(1:6, 10, replace = TRUE)

    rolls_new <- rolls
    rolls_new[(10*i - 9):(10*i)] <- new_sample

    if(sum(rolls_new) >= sum(rolls) & sum(rolls_new) <=x){rolls <- rolls_new}
    if(sum(rolls_new) <= sum(rolls) & sum(rolls_new) >=x){rolls <- rolls_new}

    i <- i + 1
    if(i == 11){
      i = 1
      iterations <- iterations + 1}

  }

  i <- 1
  iterations <- 0
  while(iterations < 1000){

    new_sample <- sample(1:6, 5, replace = TRUE)
    replace_rolls <- sample(1:100, replace = FALSE)

    rolls_new <- rolls
    rolls_new[replace_rolls] <- new_sample

    if(sum(rolls_new) == x){
      rolls <- rolls_new
    }

    i <- i + 1
    if(i == 21){
      i = 1
      iterations <- iterations + 1}

  }
  return(rolls)

}



tictoc::tic()
cl <- parallel::makeCluster(detectCores() - 1)
results <- parSapply(cl, X = rep(450, 5000), rollMH)
stopCluster(cl)
tictoc::toc()
```

```
## 258.169 sec elapsed
```

```
E_6 <- apply(results, 2, function(x)mean(x == 6)*100)
E_1 <- apply(results, 2, function(x)mean(x == 1)*100)

EVs <- data.frame(NumberInRoll = c(E_6, E_1),
                  dieNumber = c(rep("6", length(E_6)), rep("1", length(E_1))))

ggplot(EVs, aes(x = NumberInRoll, fill = dieNumber)) +
  geom_histogram(alpha = 0.5, color = "black", binwidth = 1) +
  theme_bw() +
  labs(x = "Frequency when die sum to 450") +
  scale_x_continuous(breaks = seq(0,48, 4))
```



The average number of 6's rolled when the die sum to 450 is `mean(E_6)`: 33.7254

The average number of 1's rolled when the die sum to 450 is `mean(E_1)`: 4.9176

The probability that we rolled fewer than 30 1's is `mean(E_1 < 30)`: 1

Note that there is one result where we could end up with exactly 30 1's, and that would be if we also rolled 70 6's. However, this never happened in our simulation. The probability of this event is $\frac{1}{6^{100}}$ (not conditioned on the die summing to 450)