# In Class Assignment 9

*Lydia Strebe*

*April 15, 2019*

## Using the outcome of PCA as the regressors in a Random Forest

We have a data set containing 2308 gene expressions from 83 tissue samples. The tissue samples are from 4 types of cancerous cells. We want to predict which type of cancer is present in each tissue sample using Random Forest. However, 2308 is way too many regressors, so we will use Principal Component Analysis to reduce the number of regressors before fitting a Random Forest model. We want to use enough principal components to capture at least 90% of the variance in the data set.

### Performing PCA and choosing the number of regressors

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.5.2
```

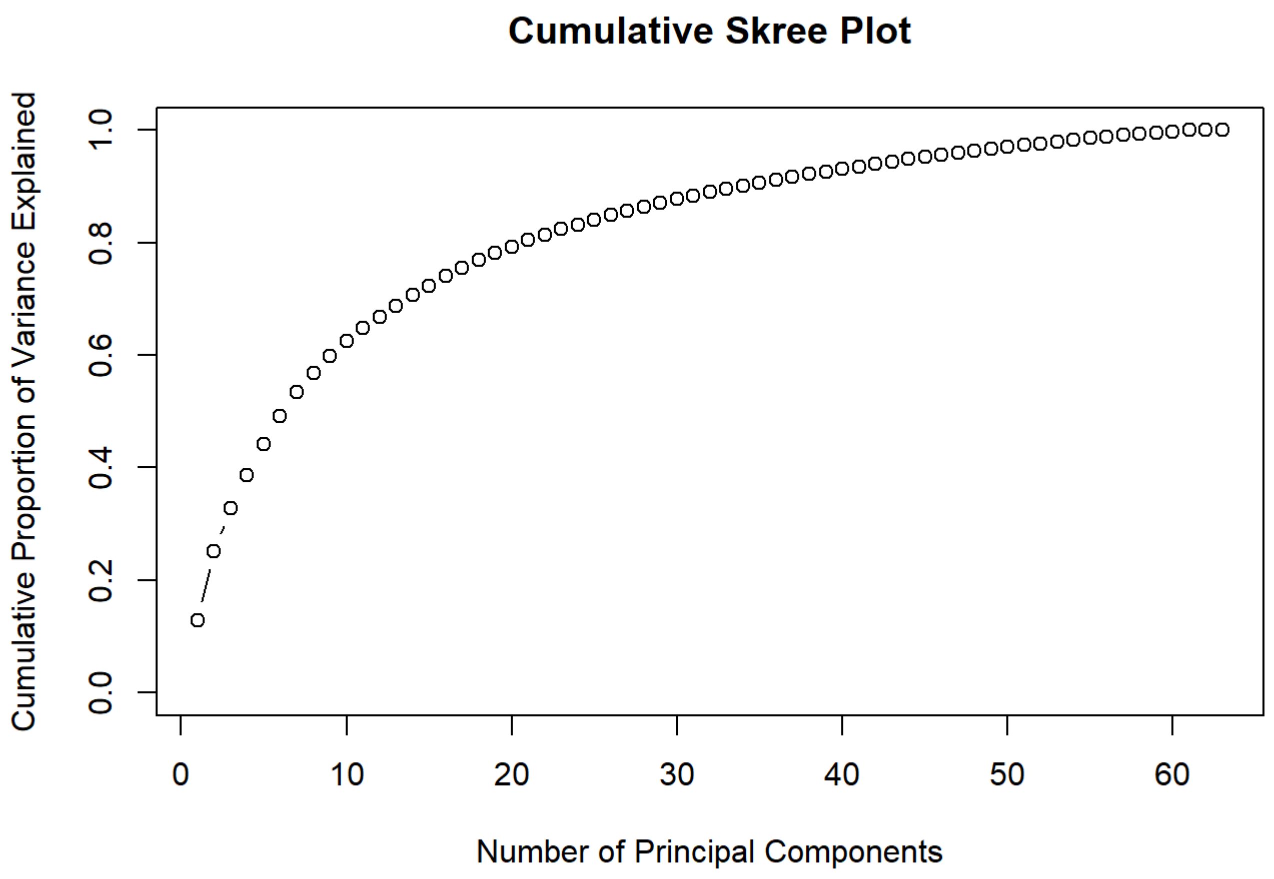
```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.2
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
#Performing PCA on Khan data set (xtrain data)
pc.Khan=prcomp(Khan$xtrain, scale=TRUE)

#Deciding on number of PCs

#calculating variance
pc.var=pc.Khan$sdev^2

#percent variance explained by each PC
pve=pc.var/sum(pc.var)

#cumulative skree plot
plot(cumsum(pve), xlab="Number of Principal Components", ylab="Cumulative Proportion of Variance
Explained", ylim=c(0,1),type='b',main="Cumulative Skree Plot")
```



```
#Choose number of PCs to explain at least 90% of variance
which(cumsum(pve)>=0.9)
```

```
##  [1] 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
## [24] 57 58 59 60 61 62 63
```

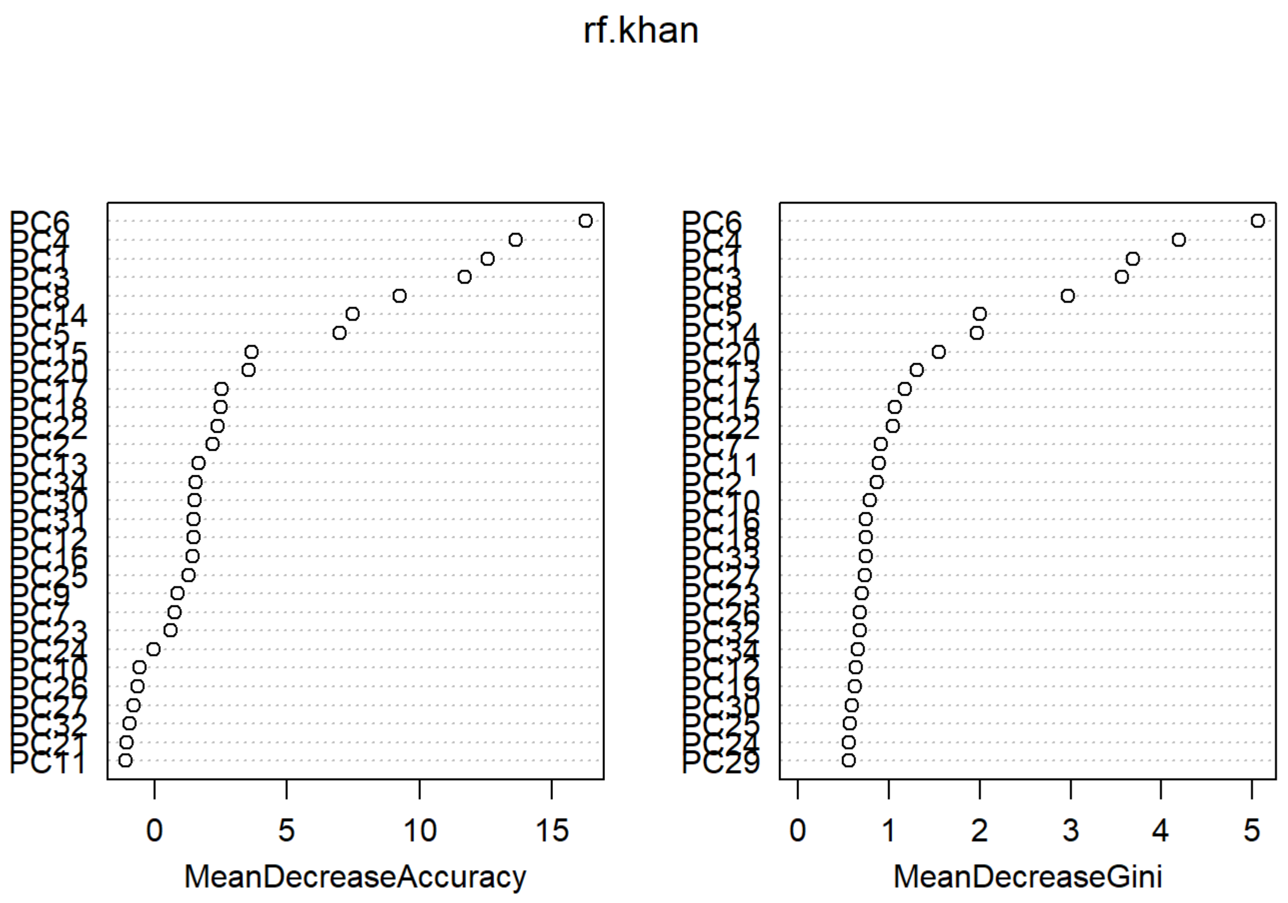
To explain at least 90% of the variance we have to use 34 principal components.

## Building Random Forest model

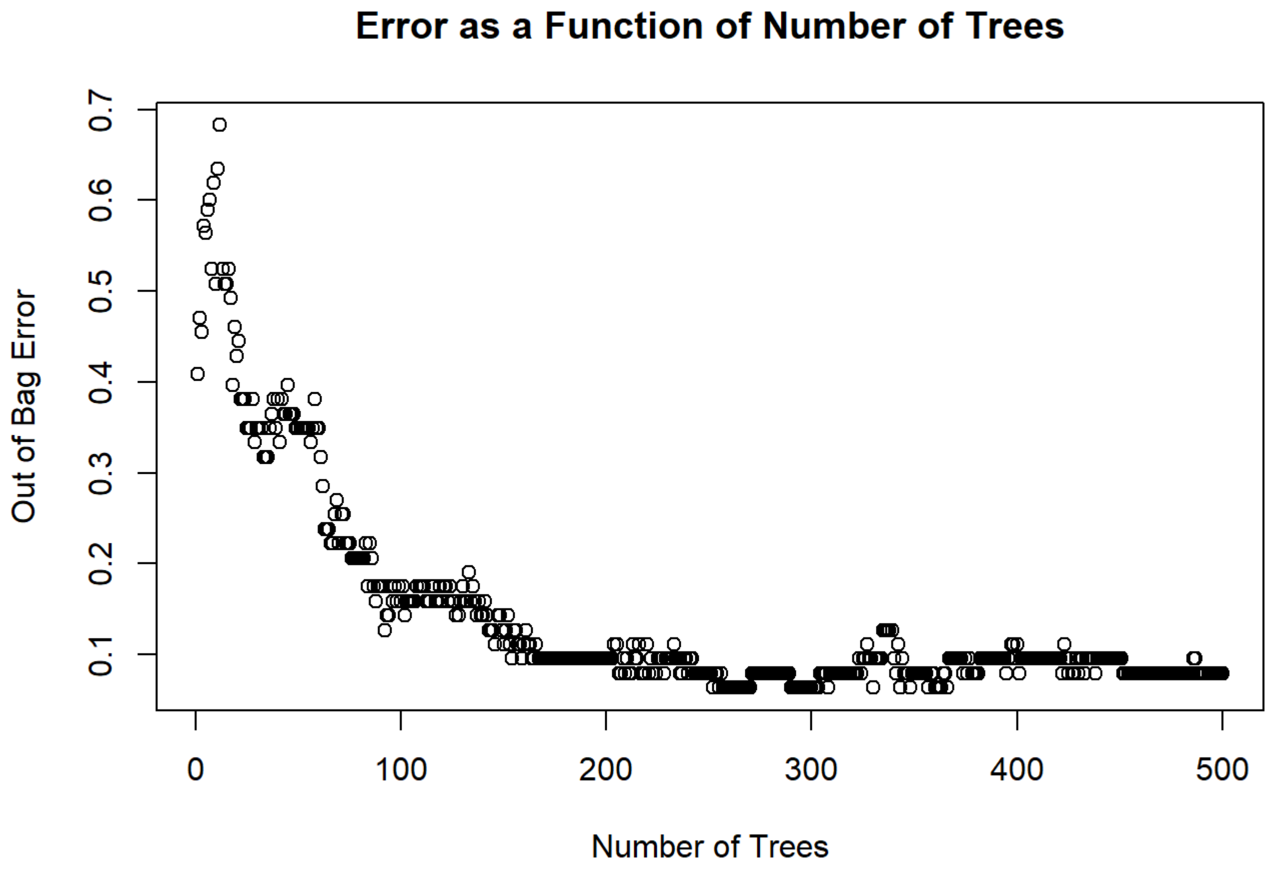
```
#make y-variable categorical (train data)
Cancer=as.factor(Khan$ytrain)

#Creating data frames

#All 63 PCs
xtrain63=data.frame(pc.Khan$x)

#First 34 PCs
xtrain34=xtrain63[,-c(35:63)]

#34 PCs plus Cancer data (y)
Khan.train34=data.frame(Cancer,xtrain34)

set.seed(3)

#Build RF model
rf.khan=randomForest(Cancer~.,data=Khan.train34,importance=TRUE)

#summary info
importance(rf.khan)
```

```
##               1          2           3           4 MeanDecreaseAccuracy
## PC1   8.84856410  4.1586974  9.32694528  9.61890401          12.59868680
## PC2   3.72607108  2.4085963 -0.09541759 -0.47020813           2.19902389
## PC3   1.68617308 10.0450556 10.87674544 -0.53994538          11.70618762
## PC4  13.90367868  3.8661576  7.23300595  4.20765016          13.63955969
## PC5   4.82864788  0.5170009  7.43048157  1.70421866           6.98159543
## PC6   5.00393059 14.2916575  3.13033103 12.81239368          16.27495657
## PC7  -0.94949865  0.4137270  0.93621363  1.86006980           0.78519393
## PC8   3.59362119  5.0564785  6.31872623  6.81275027           9.26283910
## PC9   0.01865497  0.4750550  1.34923580  0.51239183           0.86813970
## PC10  2.64184543 -1.5889409 -0.87902167 -0.92199076          -0.57092239
## PC11 -0.02612850 -0.8755451 -1.75342730  1.30820654          -1.06354507
## PC12  1.63437807 -1.3443995  2.06479764  0.97719780           1.48393158
## PC13  3.56468721  1.5291592  0.42945700 -0.59635013           1.66889076
## PC14  2.01083504  4.5886905  6.37789383  3.38548609           7.47981375
## PC15  2.59309134  2.2144404  1.98999146  1.87191561           3.66572706
## PC16 -1.54318209  2.7419444  0.09348384 -0.59341952           1.44332810
## PC17  0.61956122  3.4591585  0.85558537 -1.48202014           2.54584405
## PC18  0.29212805  1.3601796  0.81167679  1.88466430           2.51938997
## PC19 -1.77463861 -1.4252783 -0.18495226  1.39024267          -1.07739696
## PC20  3.38812763  2.7928682  4.09951582 -1.84252683           3.57424250
## PC21 -0.62041234 -0.8378180 -0.95826391  0.03049319          -1.02619431
## PC22  2.46852977  1.8098189 -0.85996830  1.45862198           2.38000209
## PC23  0.00000000  0.0719220  0.93770552 -0.06763278           0.60656240
## PC24 -1.17832948  0.1560911 -1.54412463  2.68831628          -0.02578776
## PC25 -0.41859177  0.8275901  0.09845874  1.43077840           1.29958239
## PC26 -0.59453644 -0.3236532 -0.85144698 -0.01507419          -0.63892836
## PC27 -1.73727046  0.3509704 -1.09633243  0.52051273          -0.76852227
## PC28 -0.59155125 -2.0632610 -1.35019858 -1.45257200          -2.91114984
## PC29 -2.30063155 -0.9441737 -1.19949435  0.17126220          -1.40877718
## PC30  1.98217798  0.9765129  1.17124964  0.01847456           1.50729037
## PC31 -0.09643573  3.0010123 -0.22789610 -0.46734566           1.49371052
## PC32 -2.17958570  1.6689930 -1.78489757 -0.75382916          -0.92548182
## PC33  0.59216390 -1.8235346  0.40067398 -1.26691015          -1.64758569
## PC34 -0.67750995 -0.1890206  0.09161674  2.68118574           1.57602923
##      MeanDecreaseGini
## PC1         3.6842403
## PC2         0.8627675
## PC3         3.5654299
## PC4         4.1973652
## PC5         2.0032120
## PC6         5.0619113
## PC7         0.9052595
## PC8         2.9687690
## PC9         0.5073788
## PC10        0.7919004
## PC11        0.8922250
## PC12        0.6315116
## PC13        1.3065201
## PC14        1.9686454
## PC15        1.0664902
## PC16        0.7482003
## PC17        1.1719945
## PC18        0.7415028
## PC19        0.6192962
## PC20        1.5441472
## PC21        0.5515219
## PC22        1.0452874
## PC23        0.7051665
## PC24        0.5560440
## PC25        0.5685941
## PC26        0.6809126
## PC27        0.7290478
## PC28        0.5232311
## PC29        0.5541378
## PC30        0.5871750
## PC31        0.3822456
## PC32        0.6790612
## PC33        0.7394082
## PC34        0.6515267
```

```
varImpPlot(rf.khan)
```



```
#Check number of trees (plot oob error)
plot(rf.khan$err.rate[,1],xlab="Number of Trees",ylab="Out of Bag Error",main="Error as a Functi
on of Number of Trees")
```



We use the default of 500 trees for our Random Forest model. As seen in the plot above, after about 200 trees, the error seems to stabilize. We also use the default number of regressors per split which, conventionally, is the square root of the number of regressors (in this case the square root of 34).

## Projecting the test data onto the principal components and testing the Random Forest model

```
#project xtest onto the PCs
pc.xtest63=predict(pc.Khan,Khan$xtest)

#make y-variable categorical (train data)
ytest=as.factor(Khan$ytest)

#Create data frame
Khan.test34=data.frame(ytest,pc.xtest63[,-c(35:63)])

#Rename y-variable to match train data
names(Khan.test34)[names(Khan.test34)=="ytest"] <- "Cancer"

#Predicting ytest using RF model
Cancer_pred=predict(rf.khan,Khan.test34)

#Results
table(Cancer_pred,Khan.test34$Cancer)
```

```
##
## Cancer_pred 1 2 3 4
##           1 3 0 0 0
##           2 0 4 0 0
##           3 0 0 1 0
##           4 0 2 5 5
```

```
#correct classification
Cancer_correct=(3+4+1+5)/20 #65%

#13 correctly classified, 7 misclassified
```

It’s difficult to say if a different number of principal components would have performed better. I don’t think more principal components would have performed much better since the first principal components seem to be more significant and less noisy. In other words, more principal components could lead to overfitting. However, fewer principal components may have done better for exactly this reason, the regressors used would presumably be more significant and less noisy. On the other hand, fewer regressors may have made the model more biased.

## Cross validation to tune number of principal components

We could use cross validation to tune the number of principal components included in the Random Forest model. Below is the procedure for how this could be done:

1. Randomly split the training data into 10 groups (or, since we have 63 samples, 9 groups of 7).
2. Use all the groups except one to do PCA. The group that is left out will be the validation set.
3. Project the validation set onto the principal components.
4. Build Random Forest models using the the training data (without the validation set) using 1 principal component up to all 63.
5. Test these models using the validation set, making sure that the number of principal components used for testing matches what was used for training. Note the misclassification error.
6. Repeat steps 2 through 5, using each group as the validation set exactly once.
7. Average the errors for each number of principal components. For the final model, use the number of principal components with the lowest average error.
8. Use all the training data to build a final model.