STAT542 HW5

Chun Yin Ricky Chue

Question 1

For $X = A - bb^T$ is invertible, we prove the matrix $Y = A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1-b^TA^{-1}b}$ is the inverse of X if and only if $XY = YX = I$. Note that $1 - b^TA^{-1}b$ is a scalar which is non-zero.

If part:

Verify that $XY = I$.

$$\begin{aligned}XY &= \left(A - bb^T\right)\left(A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1-b^TA^{-1}b}\right) = AA^{-1} - bb^TA^{-1} + \frac{AA^{-1}bb^TA^{-1} - bb^TA^{-1}bb^TA^{-1}}{1-b^TA^{-1}b}\\ &= I - bb^TA + \frac{bb^TA^{-1} - bb^TA^{-1}bb^TA^{-1}}{1-b^TA^{-1}b} = I - bb^TA^{-1} + \frac{b\left(1-b^TA^{-1}b\right)b^TA^{-1}}{1-b^TA^{-1}b}\\ &= I - bb^TA^{-1} + bb^TA^{-1} = I\end{aligned}$$

We also verify that $YX = I$.

$$\begin{aligned}YX &= \left(A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1-b^TA^{-1}b}\right)\left(A - bb^T\right) = A^{-1}A - A^{-1}bb^T + \frac{A^{-1}bb^TA^{-1}A - A^{-1}bb^TA^{-1}bb^T}{1-b^TA^{-1}b}\\ &= I - A^{-1}bb^T + \frac{A^{-1}bb^T - A^{-1}bb^TA^{-1}bb^T}{1-b^TA^{-1}b} = I - A^{-1}bb^T + \frac{A^{-1}b\left(1-b^TA^{-1}b\right)b^T}{1-b^TA^{-1}b}\\ &= I - A^{-1}bb^T + A^{-1}bb^T = I\end{aligned}$$

Hence, $XY = YX = I$.

Only if part:

Consider two cases, first if $b = 0$, then $X = A - bb^T = A$ and $Y = A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1-b^TA^{-1}b} = A^{-1}$, and the fact that $XY = AA^{-1} = I$ and $YX = A^{-1}A = I$ imply that $Y = A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1-b^TA^{-1}b}$ is the inverse of $X = A - bb^T$.

If $b \neq 0$, consider,

$$\left(A - bb^T\right)A^{-1}b = b - bb^TA^{-1}b = b\left(1 - b^TA^{-1}b\right) = \left(1 - b^TA^{-1}b\right)b$$

The last step is legitimate since $1 - b^TA^{-1}b$ is a scalar.

As $A - bb^T$ is invertible, $\left(A - bb^T\right)A^{-1}$ is also invertible since it is a product of two invertible matrices. Given $b \neq 0$, $\left(A - bb^T\right)A^{-1}b \neq 0$. Hence, $\left(1 - b^TA^{-1}b\right)b \neq 0$, which implies $1 - b^TA^{-1}b \neq 0$.

As $A - bb^T$ is invertible if and only if $1 - b^TA^{-1}b \neq 0$, the proposition is hence proven.

Question 2.

In this part, we write our own code to fit the sliced inverse regression (SIR), and validate it by comparing to the `dr` package. We compare the directions of the principle components, as well as the eigenvalues by the two approaches. Here, we present the function for our code SIR code.

```
sir_fx <- function (x, y, slice = NULL) {
  n <- length(y)    # Number of data points
  p <- dim(x)[2]    # Number of dimensions
  x_center <- scale(x, center=T, scale=F) # Centering x's.
  svd <- eigen(cov(x))
  Gamma <- svd$vectors
  Sigma_diag <- diag(1/svd$values) # Diagonal matrix D
  hat_inv_Sigma_sqrt <- Gamma %*% sqrt(Sigma_diag) %*% t(Gamma) # Sigma^(-1/2)
  z <- x_center %*% hat_inv_Sigma_sqrt # Transform X to z, which is standardized.

  # Slicing
  H = slice
  mydata = matrix(0, nrow = n, ncol = (p+1))
  mydata[,1:p] <- z
  mydata[,p+1] <- y
  mydata = mydata[order(mydata[,p+1]), ] # Sort the dataset with y values.
  mydata = mydata[,1:p] # After sorting, get the z values only.

  # Spliting the dataset in different slices, and calculate the sample means of the slices.
  id = split(1:n, cut(seq_along(1:n), H, labels = FALSE))
  id_mat = as.matrix(id) # Number of samples in each slice.
  slice_num <- lengths(id_mat)
  samplemeans = sapply(1:H, function(h, mydata, id, p) colMeans(mydata[id[[h]], ]), mydata, id)

  # Mean of the dataset.
  means = colMeans(mydata)

  # Sample mean of slice minus mean of dataset.
  diff <- sweep(t(samplemeans[,]),2,means)

  # Multiply number of samples to one of the diff matrix.
  diff_num <- sapply(1:p, function(h, diff, slice_num) diff[,h] * slice_num, diff, slice_num)
  M <- t(diff_num) %*% diff / n

  # Eigenvectors and eigenvalues of M.
  M_eig <- eigen(M)
  # Returning the eigenvalues of M and the eigenvectors in the original space.
  return(list(eigen(M)$values, hat_inv_Sigma_sqrt %*% eigen(M)$vectors))
}
```

Part a.

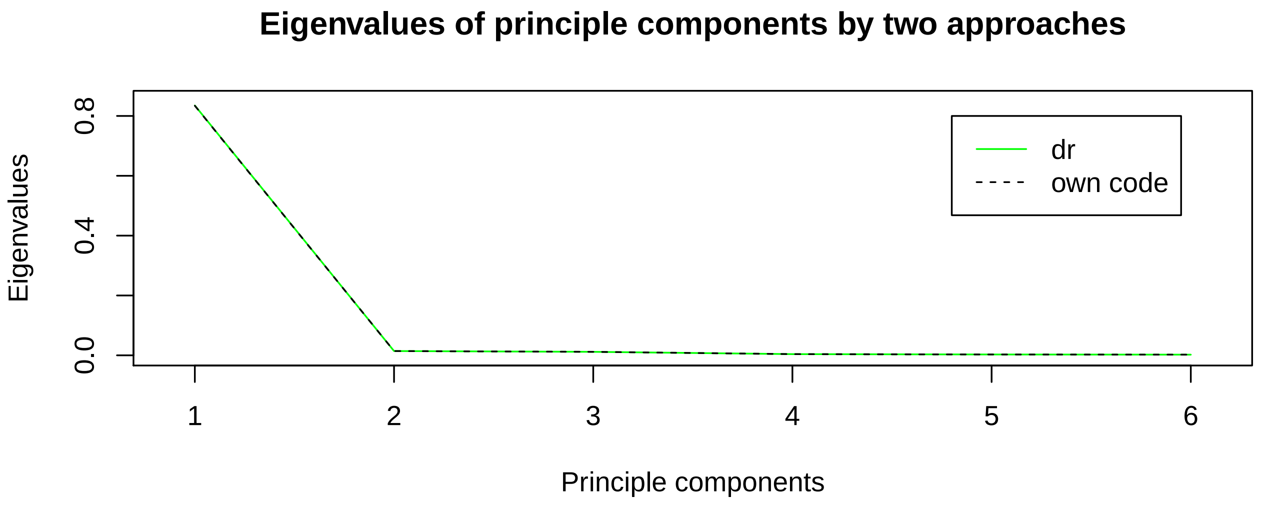
We generate 1000 observations from the model $y = 0.4(X_1 + X_2) + 0.125(X_1 + X_2)^5 + 0.5\varepsilon$, which can be detected by SIR. Here, $X = [X_1, X_2, ..., X_6]$, y is the response and ε is the standard error term.

```r
library(dr)
# generate some data with one direction
set.seed(4)
n = 1000; p = 6; H = 10
x = matrix(rnorm(n*p), n, p)
b = matrix(c(1, 1, rep(0, p-2)))
y = 0.4*(x %*% b) + 0.125*(x %*% b)^5 + 0.5*rnorm(n)

# Fitting from the dr package.
fit.sir = dr(y~., data = data.frame(x, y), method = "sir", nslices=H)
sir_eig <- sir_fx(x, y, H) # Fitting from our own code.
```

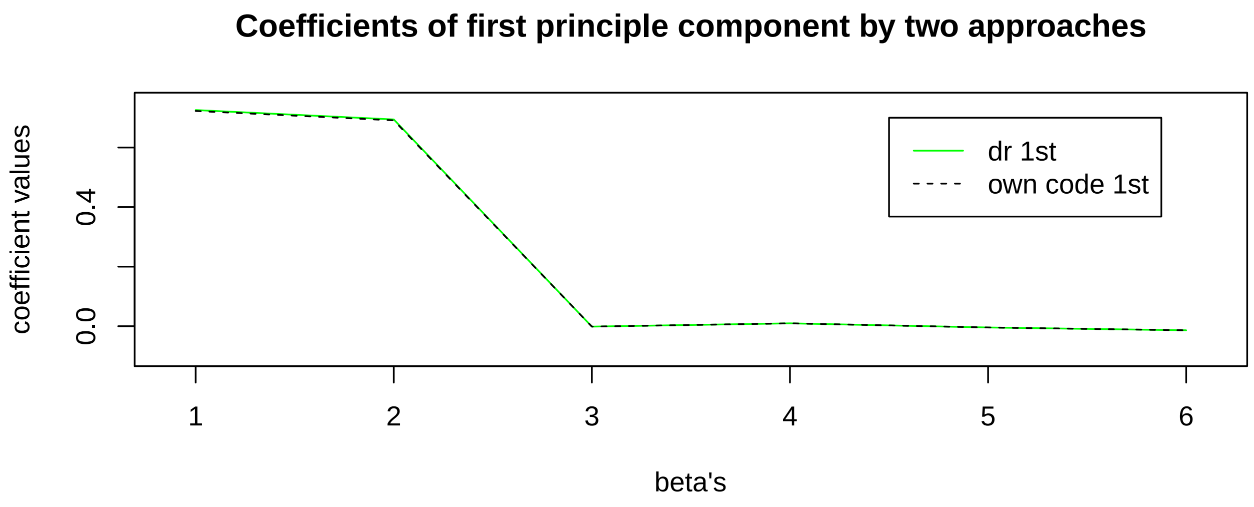
We compare first the eigenvalues of different principle components computed from the two appraoches. The solid green and dashed black lines represent the plots by the `dr` package and our own code respectively. They match on predicting the six principle components. Particularly, the first principle component is relatively important compared to the remaining five components.

```r
plot(c(0.9,6.1),c(0,0.85),type="n",xlab = "Principle components", ylab = "Eigenvalues")
lines(seq(1,p), fit.sir$evalues, col = "green")
lines(seq(1,p), sir_eig[[1]], col = "black", lty = 2)
legend(4.8,0.8, c("dr","own code"),col=c("green","black"),lty=c(1,2))
title("Eigenvalues of principle components by two approaches")
```



Next, we compare the directions of the first principle component predicted by the two approaches. The two approaches might predict directions with opposite signs, so we manually add a minus sign to the result by one approach to match the directions.

```r
# Eigenvectors predicted by own code.
sir_evec1 <- -sir_eig[[2]][,1]
plot(c(0.9,6.1),c(-0.1,0.75),type="n",xlab = "beta's", ylab = "coefficient values")
lines(seq(1,p), sir_evec1, col = "green")
lines(seq(1,p), fit.sir$evectors[,1], col = "black", lty = 2)
legend(4.5,0.7, c("dr 1st","own code 1st"),col=c("green","black"),lty=c(1,2))
title("Coefficients of first principle component by two approaches")
```



The figure shows the β's (directions) of the first principle component predicted by the two approaches. The solid green and dashed black lines represent the prediction of the first principle components from the `dr` package and own code respectively. Again, the directions predicted by the two approaches match.

The truth is, the model is predicted by the first two components of X, i.e. X_1 and X_2. As can be seen in the figure, the β's given by the first two X components are significant, and they are almost the same in magnitude, which means the two directions give equal contribution in explaining the data. The other components are close to zero, which means they barely explain any tendency in the data.

Part b.

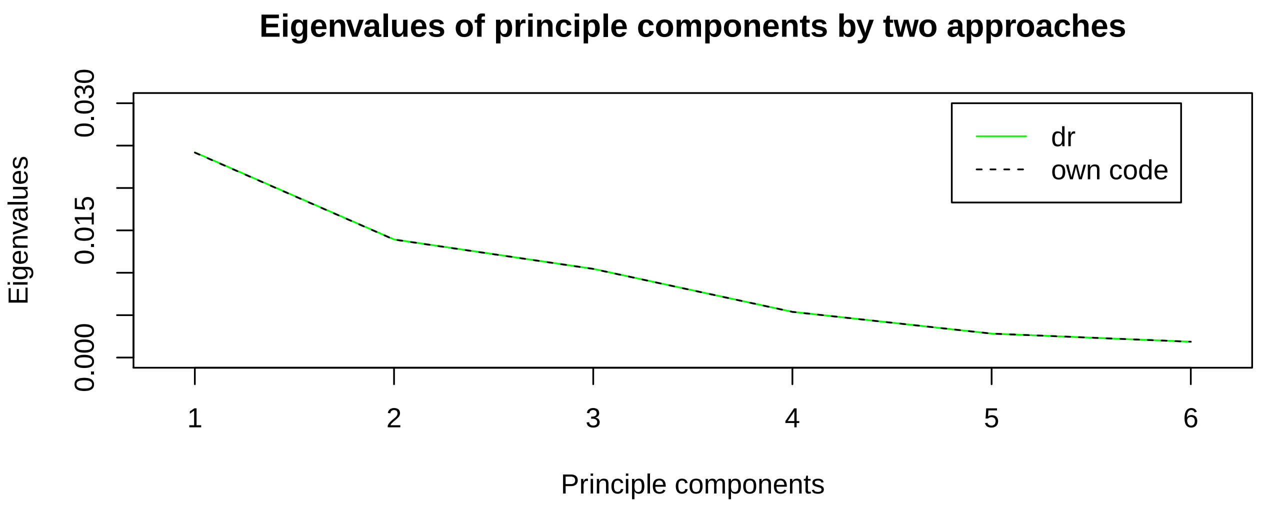
We generate 1000 observations from the model $y = 0.4(X_1 + X_2)^2 + 0.5\varepsilon$, which cannot be detected by SIR because of the quadratic behavior of the function, i.e. given y, there could be multiple values of X's. The notations are the same as in part a.

```r
set.seed(1)
x2 = matrix(rnorm(n*p), n, p)
y2 = 0.4*(x2 %*% b)^2 + 0.5*rnorm(n)

# Fitting from the dr package.
fit.sir = dr(y2~., data = data.frame(x2, y2), method = "sir", nslices=H)
sir_eig <- sir_fx(x2, y2, H) # Fitting from our own code.
```

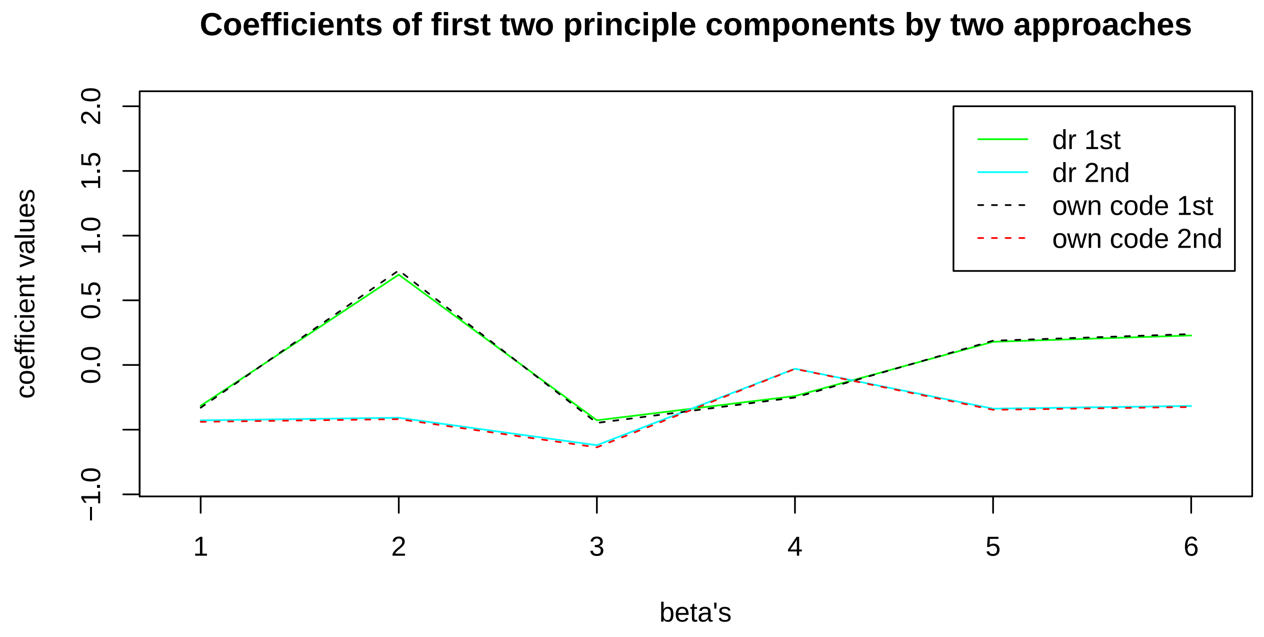
We compare first the eigenvalues of different principle components computed from the two appraoches. The solid green and dashed black lines represent the plots by the `dr` package and our own code respectively. They match on predicting the six principle components. Particularly, the first 2 principle components are relatively important compared to the remaining four components.

```r
plot(c(0.9,6.1),c(0,0.03),type="n",xlab = "Principle components", ylab = "Eigenvalues")
lines(seq(1,p), fit.sir$evalues, col = "green")
lines(seq(1,p), sir_eig[[1]], col = "black", lty = 2)
legend(4.8,0.03, c("dr","own code"),col=c("green","black"),lty=c(1,2))
title("Eigenvalues of principle components by two approaches")
```



Next, we compare the directions of the first two principle components predicted by the two approaches.

```
# Eigenvectors predicted by own code.
sir_evec1 <- -sir_eig[[2]][,1]
sir_evec2 <- sir_eig[[2]][,2]
plot(c(0.9,6.1),c(-0.9,2),type="n",xlab = "beta's", ylab = "coefficient values")
lines(seq(1,p), sir_evec1, col = "green")
lines(seq(1,p), sir_evec2, col = "cyan")
lines(seq(1,p), fit.sir$evectors[,1], col = "black", lty = 2)
lines(seq(1,p), fit.sir$evectors[,2], col = "red", lty = 2)
legend(4.8,2, c("dr 1st","dr 2nd","own code 1st","own code 2nd"),
       col=c("green","cyan","black","red"),lty=c(1,1,2,2))
title("Coefficients of first two principle components by two approaches")
```



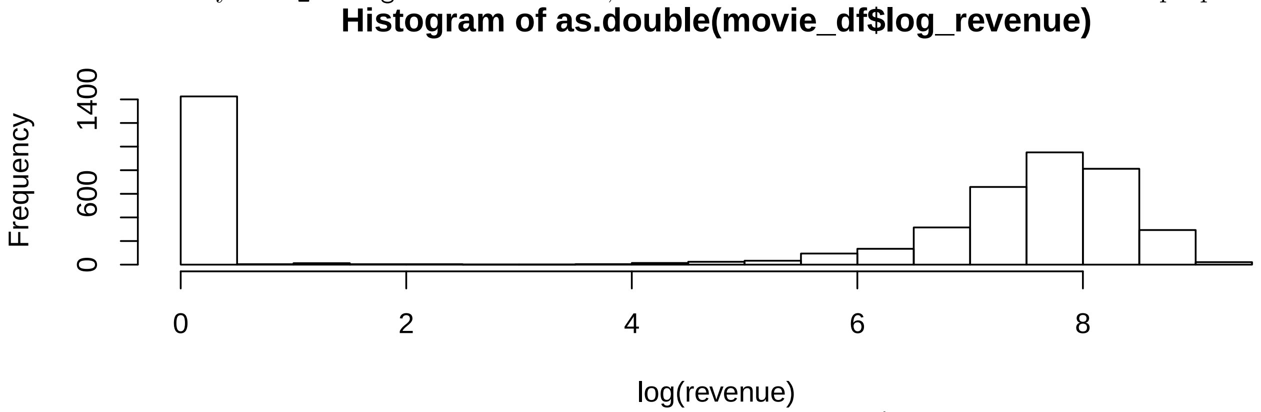
In this case, the β's apart from the first two X's are comparable to those from β_1 and β_2 (coefficients of X_1 and X_2). That means, the SIR is not able to identify the most important directions X_1 and X_2. The results from the `dr` and own code match though.

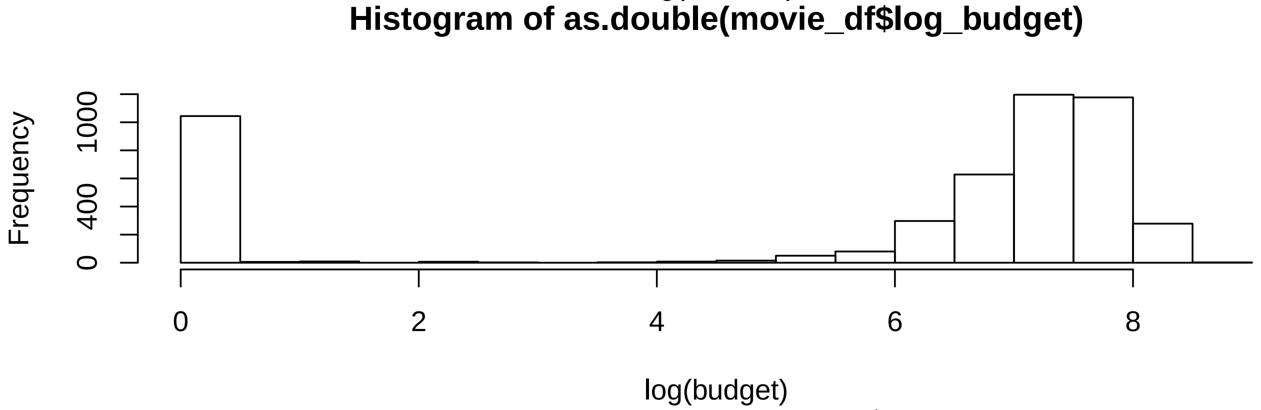
Question 3.

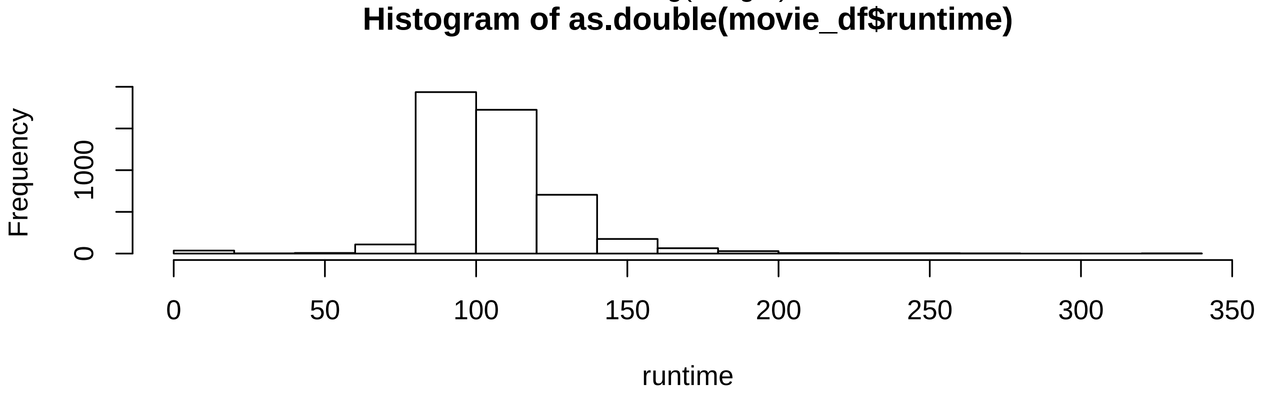
We perform a data analysis on tmdb movie. We predict two variables `revenue` by regression and if `vote_average` is greater than 7 by classification. This is a log transformation for the budget and revenues, i.e. $Y = \log_{10}(1 + X)$.

Data preprocessing

We perform data pre-processing. Firstly, we eliminate the rows with NA entries. Moreover, some missing values of `revenue`, `budgets` and `runtime` are imputed by 0, which are shown in the following histograms that skew the distribution. For simplicity, we eliminate those rows with `revenue`, `budget` or `runtime` equal 0. The skewnesses in `revenue` and `budget` are more obvious if the features are log-transformed. We further classify `vote_average` above 7 to be 1, and those below 7 to be -1 for classification purpose.

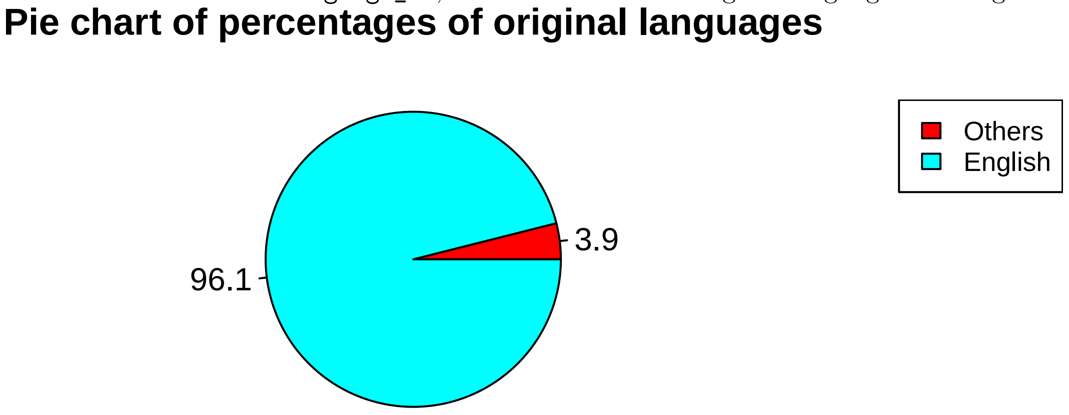




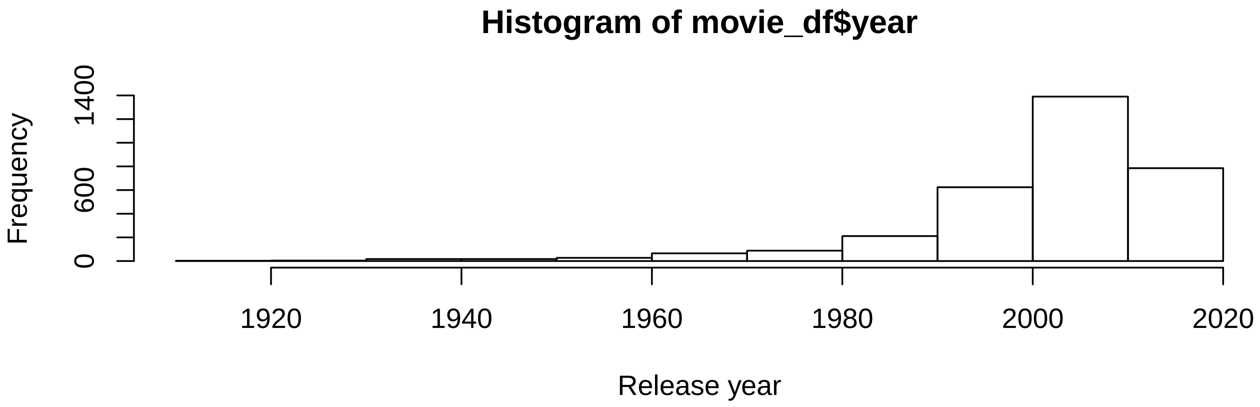


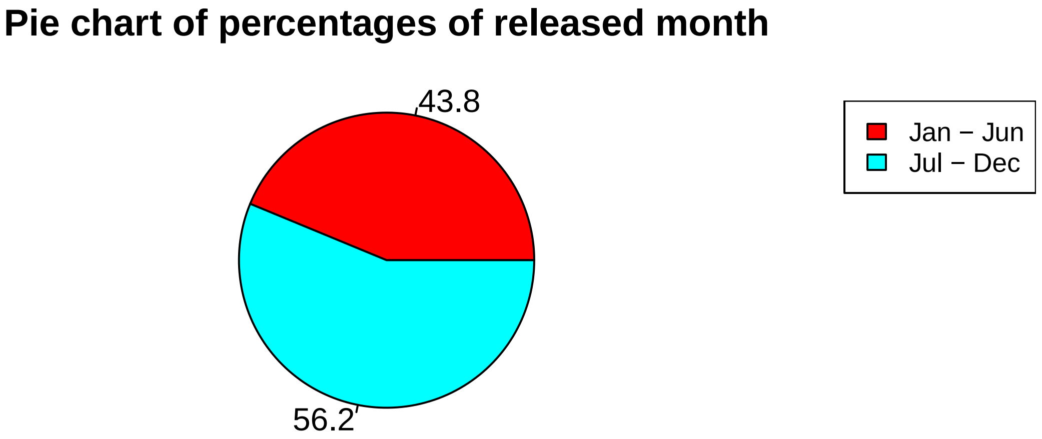
2 data points are with `NA` entries, and 1572 entries are with either `budget`, `revenue` or `runtime` being 0. We remove all of those data points and so the number of data points remaining is: $4803 - 2 - 1572 = 3229$.

Next, we note that a majority of movies (3102 out of 3229, 96.1%) are with original language in English (`en`). We introduce a feature `language_en`, which takes 1 if the original language is in English and -1 otherwise.

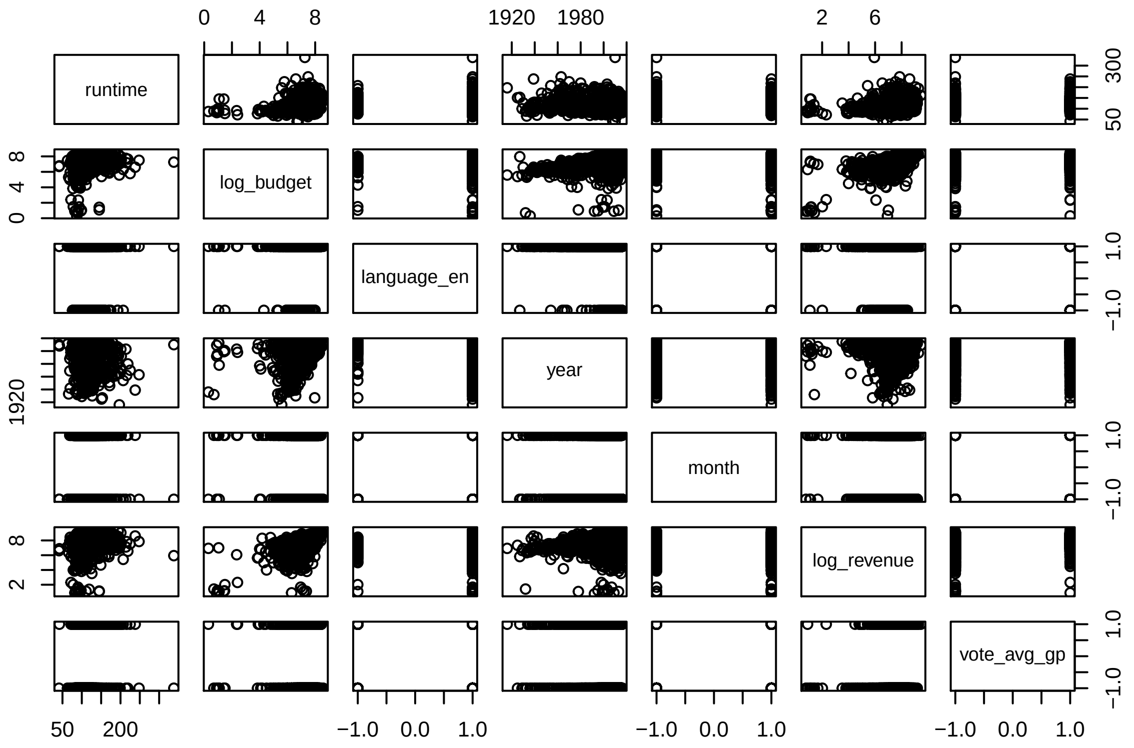


For `release_date`, we treat the release year as a continuous variable, and the distribution is shown in the following histogram. For released month, we make it as a categorical variable, with movies released between January and June classified as -1, and otherwise 1.





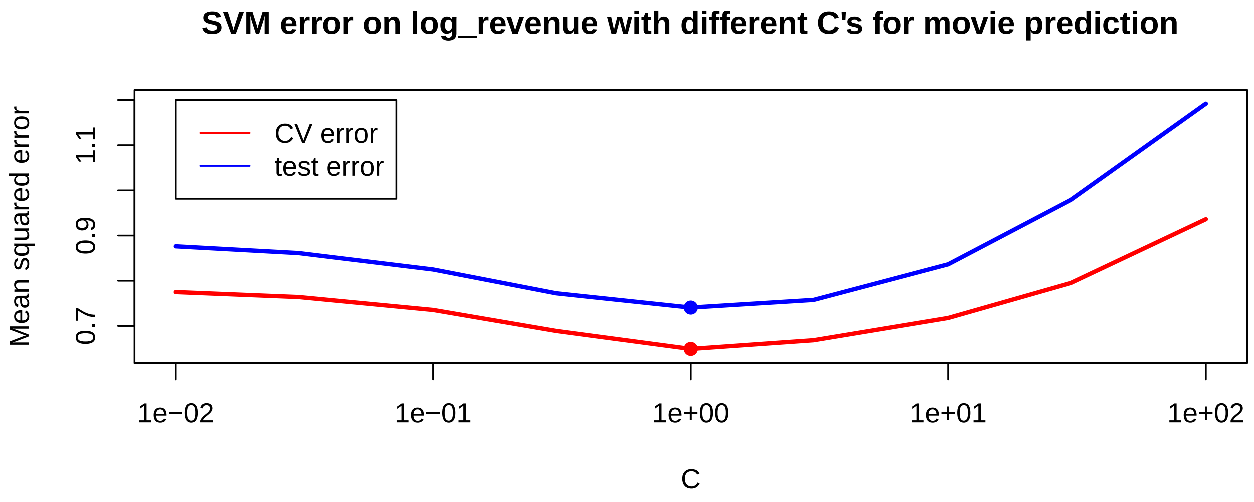
We discover the correlations between the features and the responses. We visualize it by pairwise scatter plots. For the first inspection, we see there is positive (although small) correlations between `runtime` and `log_budget` to `log_revenue`.



We split movies with odd `id` and even `id` to be training and testing datasets respectively.

Regression

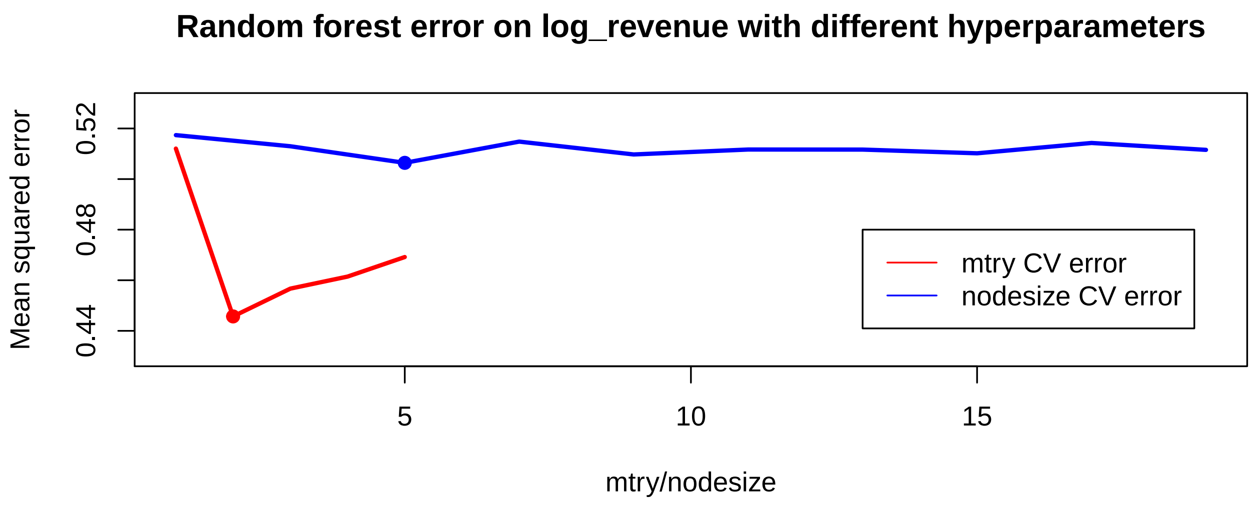
We perform regression of predicting `log_revenue` by using multiple algorithms, here we use SVM, random forest. In all algorithms, we perform K-fold cross validation (CV) on the training set to look for tune the hyperparameters, and report the test error using the tuned hyperparameter and the trained model on the test set. For regression, we use mean square error for error measurement metric. We firstly present the result predicted by SVM.



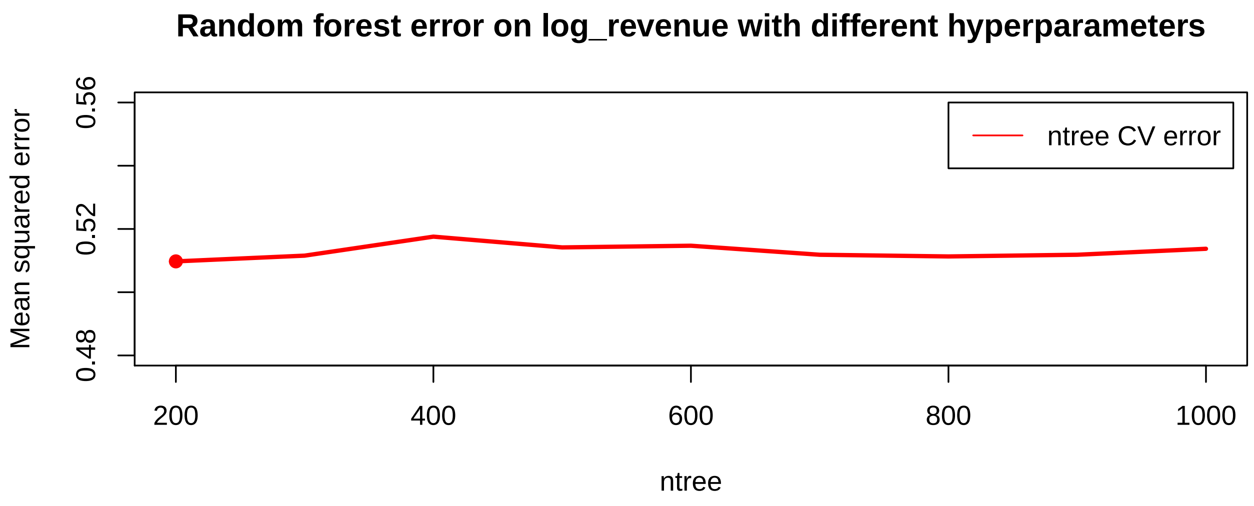
For the figure above, we show the CV and test errors. In both cases, $C = 1$ is the best tuned cost in CV. It turns out it gives the best prediction results on the test set, with mean squared error of `log_revenue` of:

```
## [1] 0.740614
```

We then try random forest algorithm. There are three hyperparameters, namely `ntree`, `mtry` and `nodesize` respectively. We can do a 3D grid search for training, but that is too time-consuming. We instead tune one of the three parameters, keeping the other two fixed at default value. Then we choose the best tuned parameters in each case for testing set.



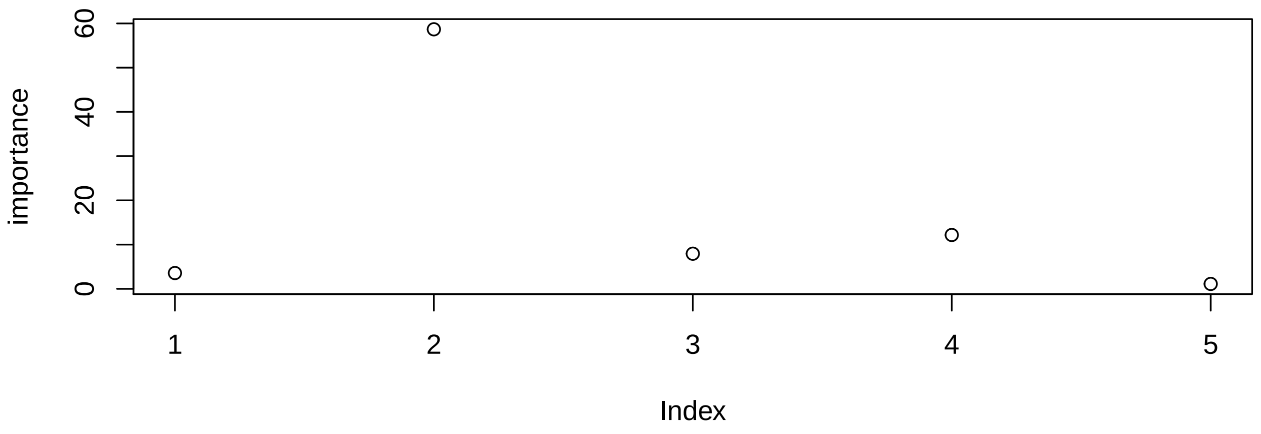
The above figure shows the CV error when varying `mtry` and `nodesize`, represented by red and blue lines respectively. The optimal `mtry` $= 2$, which is in line with the recommended value $p/3 = 5/3 = 2$, where p is the number of features. The CV error does not depend much on the nodesize, and the optimal value is the package recommended value, which is 5.



The CV error does not depend on `ntree`. In the above CV, we choose (`mtry`,`nodesize`,`ntree`) = (2,5,200), and report the prediction error by Random Forest.

As can be seen, Random Forest outperforms SVM in this dataset, both on CV error and test errors. As random forest gives a better prediction than SVM, we calculate the variable importance from the Random Forest algorithm. The variable importance is based on the mean decrease in accuracy after removing the feature averaged over all trees.

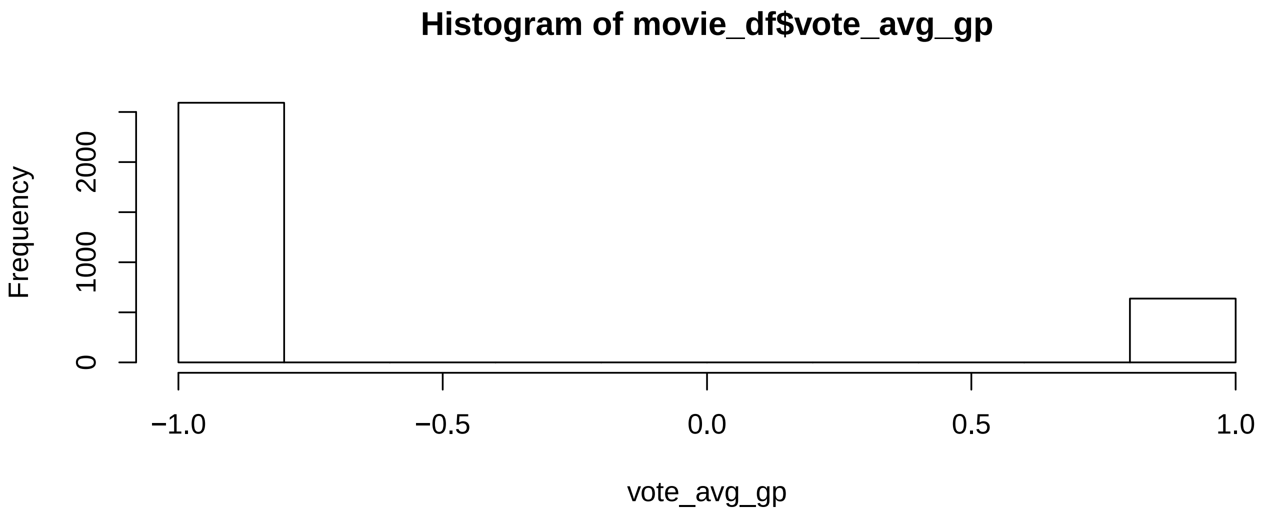
```
##               %IncMSE
## runtime      3.567489
## log_budget  58.672850
## language_en  7.938786
## year        12.173541
## month        1.107028
```

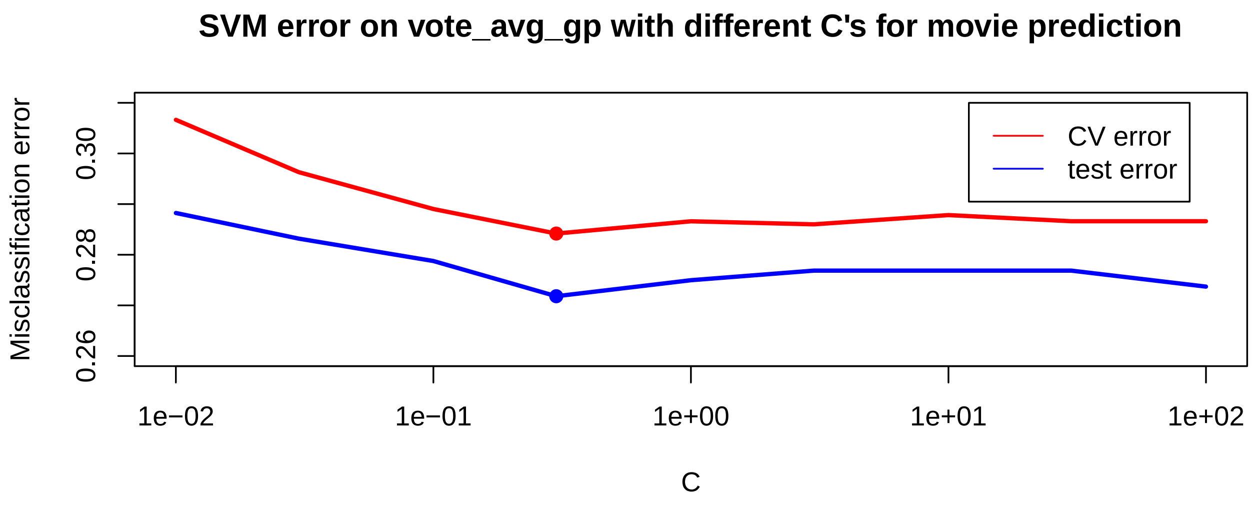


As can see in the above plot, budget accounts for 62% of the importance in the prediction, which is the most important features predicting the `revenue`. `Runtime` and `month` account for less than 5% of importance, those are not important indicators whether the movie has a high revenue or not.

Classification

We again use SVM and Random Forest to classify if the movie is getting a high rate (over 7) or not. The following histogram shows the frequencies of the movie with rating lower than 7 (group -1) versus that with rating higher than 7 (group +1). There is an imbalance of frequencies that we have to deal with. Here, we put in `class.weights` equal to the inverse of occurences in each class in the algorithm, i.e. The movie with higher rating would receive higher weights due to smaller number of occurrence.

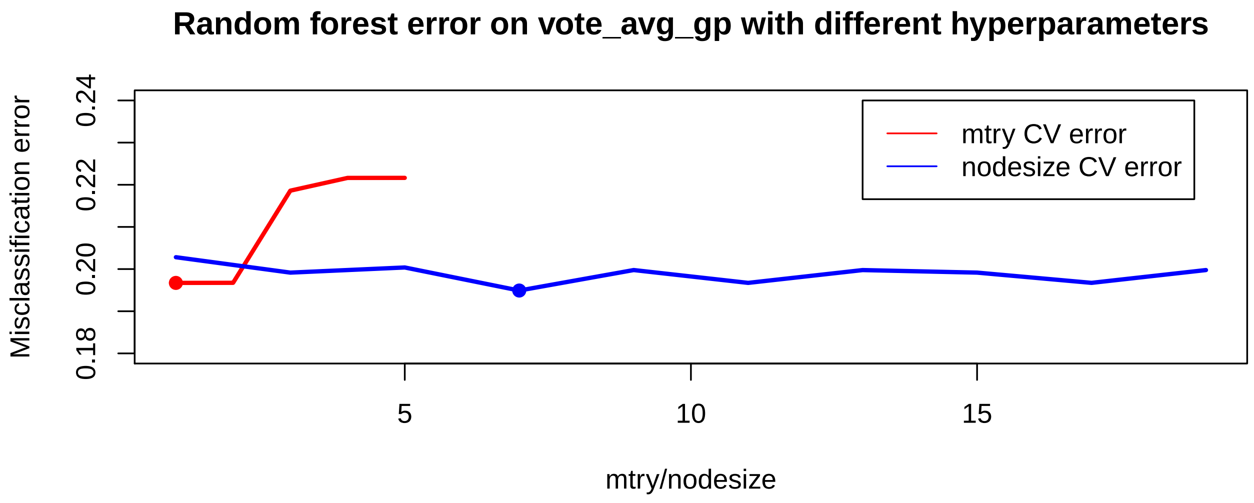




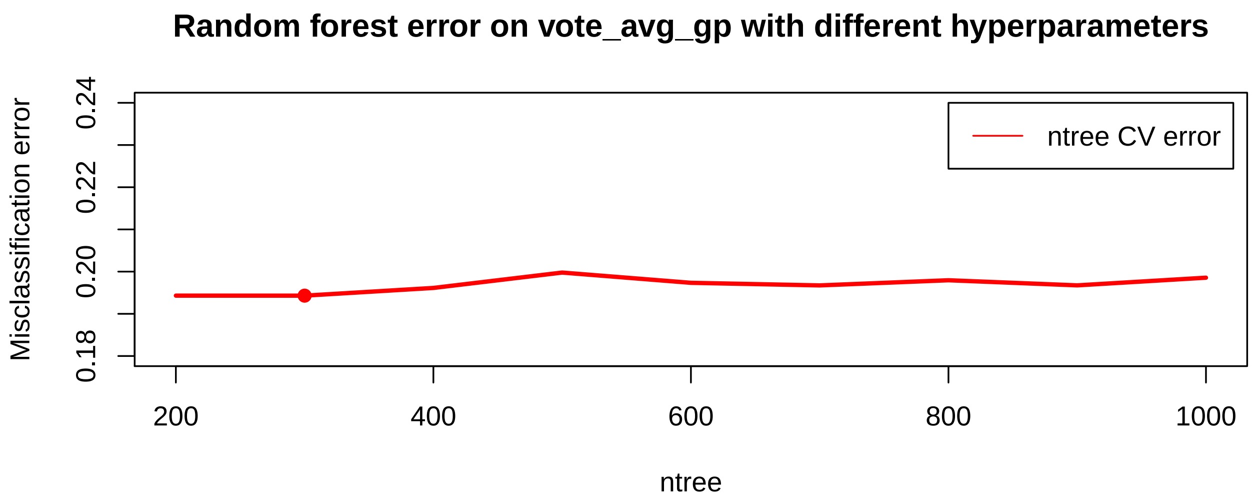
For the figure above, we show the CV and test misclassification errors. In both cases, $C = 0.3$ is the best tuned cost in CV. It turns out it gives the best prediction results on the test set, with misclassification error of `vote_avg_gp` of:

```
## [1] 0.2718078
```

We repeat by random forest algorithm. We tune parameters `ntree`, `nodesize` and `mtry` for testing set.



The above figure shows the CV error when varying `mtry` and `nodesize`, represented by red and blue lines respectively. The optimal `mtry` $= 1$, which is in line with the recommended value $p/3 = 5/3 = 2$, where p is the number of features. The CV error does not depend much on the nodesize, and the optimal value is 7.

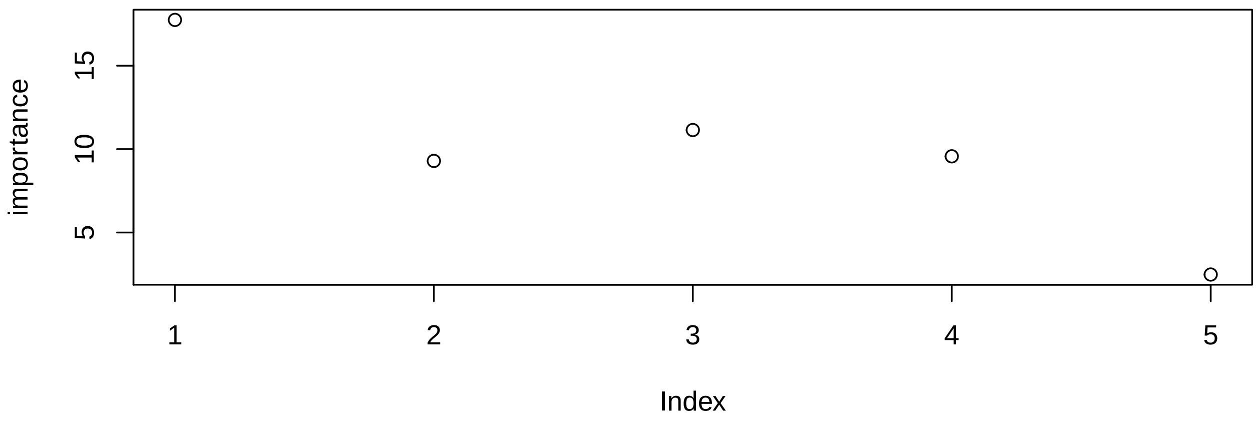


The CV error does not depend on `ntree`. In the above CV, we choose (`mtry`,`nodesize`,`ntree`) = (1,7,300), and report the misclassification error by Random Forest.

```
## [1] 0.1883692
```

As can be seen, Random Forest again outperforms SVM in this dataset, both on CV error and test errors. As random forest gives a better prediction than SVM, we calculate the variable importance from the Random Forest algorithm. The variable importance is based on the mean decrease in accuracy after removing the feature averaged over all trees.

```
##              MeanDecreaseAccuracy
## runtime                 17.746167
## log_budget               9.292577
## language_en             11.146008
## year                     9.568712
## month                    2.481267
```



As can see in the above plot, `runtime` accounts for 17% of the importance in the prediction, which is the most important features predicting the `vote_avg_gp`. `month` accounts for less than 2.5% of importance, that is not an important indicator whether the movie has a high vote or not.

Prediction on Star Wars

From `imdb.com`, the length of the movie Star Wars is 152 mins, released in December 15, 2017, so the `year` and `month` features would be 2017 and 1. `language_en` is 1 since the film was produced in English. The budget is 2.068 billion dollars, which is $\log_{10}(2.068 \times 10^9) = 9.31555$.

We make a prediction on the `revenue` and `vote` of Star Wars.

```
##        1        1
## 8.682557 1.000000
```

The predicted `revenue` is $10ˆ{8.682557} = $ $481 million, and it is predicted to have a high rating!