```r
library(dplyr)
library(car)
library(caret)
library(glmnet)

setwd("/Users/Darshan/Documents/Online_News_Popularity")

# Train - Test data split
news <- read.csv("OnlineNewsPopularity.csv", header = TRUE)

set.seed(10)

# shuffle the data set
news <- news[sample(1:nrow(news)),]

# 20 % of data will be separated for testing
n_test_samples <- round(nrow(news) * 0.20)

news_test <- news[(1:n_test_samples),]
news_train <- news[((1+n_test_samples):nrow(news)),]

write.csv("Train.csv", row.names = FALSE, x = news_train)
write.csv("Test.csv", row.names = FALSE, x = news_test)
```

```r
# This function cleans the train dataset
data_cleaning <- function(news){

  # non-predictor
  news$timedelta <- NULL

  # Removing instances which don't have any text content in it.
  news <- filter(news, n_tokens_content != 0)

  news$n_non_stop_words <- NULL # Constanct predictor
  news$kw_min_min <- NULL # More than 50% instances contain -1 value
  news <- filter(news, n_unique_tokens <= 1) # Outlier value greater than 1

  news <- filter(news, kw_min_avg >= 0, kw_avg_min>=0)
  # Outlier value less than 1

  # Log transformation becuase there are high number of outliers
  news$LDA_00 <- log(news$LDA_00 + 1)
  news$LDA_01 <- log(news$LDA_01 + 1)
  news$LDA_02 <- log(news$LDA_02 + 1)
  news$LDA_03 <- log(news$LDA_03 + 1)
  news$LDA_04 <- log(news$LDA_04 + 1)

  news$self_reference_avg_sharess <- log(news$self_reference_avg_sharess + 1)

  news$kw_max_min <- log(news$kw_max_min + 1)
  news$kw_avg_min <- log(news$kw_avg_min + 1)

  news$kw_min_avg <- log(news$kw_min_avg + 1)
```

```
  news$kw_min_max <- log(news$kw_min_max + 1)
  news$kw_max_avg <- log(news$kw_max_avg + 1)
  news$kw_avg_avg <- log(news$kw_avg_avg + 1)

  news$kw_avg_max <- log(news$kw_avg_max + 1)
  news$kw_max_max <- log(news$kw_max_max + 1)

  return(news)

}

# This function handle the multi-collinearity in the train dataset.
correlation_cleaning <- function(news){

  news$rate_negative_words <- NULL
  # n_non_stop_unique_tokens and n_unique_tokens have correlation of 0.887
  #  n_non_stop_unique_tokens is removed from the analysis as both the predictors
  # are semantically similar
  news$n_non_stop_unique_tokens <- NULL

  # self_reference_min_shares and self_reference_max_shares has high corelation with
  # self_reference_avg_sharess
  news$self_reference_min_shares <- NULL
  news$self_reference_max_shares <- NULL

  news$i_n_unique_tokens_content <- news$n_unique_tokens + news$n_tokens_content
  # 0.751 colinearity between n_unique_tokens and n_tokens_content
  news$n_unique_tokens <- NULL
  news$n_tokens_content <- NULL

  news$i_title_sub_sent_polarity <- (news$title_subjectivity +
                                     news$abs_title_sentiment_polarity) / 2.0
  # 0.71 colinearity between title_subjectivity and abs_title_sentiment_polarity
  news$title_subjectivity <- NULL
  news$abs_title_sentiment_polarity <- NULL

  # 0.719 colinearity between min_negative_polarity and avg_negative_polarity
  news$i_min_avg_negative_pol <- (news$min_negative_polarity +
                                  news$avg_negative_polarity) / 2.0
  news$min_negative_polarity <- NULL
  news$avg_negative_polarity <- NULL

  # 0.779 colinearity between rate_positive_words and global_sentiment_polarity
  news$i_rate_pos_gsent_polarity <- (news$rate_positive_words *
                                       news$global_sentiment_polarity)
  news$rate_positive_words <- NULL
  news$global_sentiment_polarity <- NULL

  #kw_max_min and kw_avg_min have correlation of 0.901
  news$i_kw_max_avg_min <- (news$kw_max_min + news$kw_avg_min) / 2.0
  #kw_max_avg and kw_avg_avg have correlation of 0.899
  news$i_kw_max_avg_avg <- (news$kw_max_avg + news$kw_avg_avg) / 2.0
```

```
  # High collinearity after applying log transformation on kw_min_avg and kw_min_max
  # Log transformation has improved the r-squared value
  news$kw_min_max<- NULL

  # High collinearity after applying log transformation on kw_avg_max and kw_max_max
  # Log transformation has improved the r-squared value
  news$i_kw_avg_max_max <- (news$kw_avg_max + news$kw_max_max) / 2.0
  news$kw_avg_max <- NULL
  news$kw_max_max <- NULL

  news$kw_max_min <- NULL
  news$kw_avg_min <- NULL
  news$kw_max_avg <- NULL
  news$kw_avg_avg <- NULL

  # After trying different interactions between the predictors,
  # correlation did not decrease significantly, so
  # self_reference_min_shares and self_reference_max_shares
  # predictors are both removed.
  news$self_reference_min_shares <- NULL
  news$self_reference_max_shares <- NULL

  return(news)

}

# This function applies the Box-Cox transformation on responce variable
target_transformation <- function(news) {

  p <- powerTransform(news$shares)
  shares_transformed <- bcPower(news$shares, p$lambda)
  news$shares <- shares_transformed

  return(list("news"=news, "lambda"=p$lambda))
}

# This function returns the actual value of the responce variable
# from the Box-Cox transformation
target_inverse <- function(shares, lambda) {
  if (lambda == 0) {
    shares <- exp(shares)
  }
  else {
    shares <- (lambda*shares + 1)^(1/lambda)
  }

  return(shares)
}

# This funciton normalize continuous variables of the train dataset
normalization <- function(news_train){

  # All Column names
```

```r
  column_names <- names(news_train)

  # Column names which needs to be ignored due to categorical and target feature
  ignored_column_names <- c("url", "timedelta", "data_channel_is_lifestyle",
                            "data_channel_is_entertainment", "data_channel_is_bus",
                            "data_channel_is_world", "data_channel_is_socmed",
                            "data_channel_is_tech", "weekday_is_monday",
                            "weekday_is_tuesday",
                            "weekday_is_wednesday", "weekday_is_thursday",
                            "weekday_is_friday",
                            "weekday_is_saturday", "weekday_is_sunday", "is_weekend",
                            "shares")

  needed_columns <- setdiff(column_names,ignored_column_names)

  # Normalized Train Data
  #news_train_norm <- news_train %>% mutate_each_(funs(scale),vars=needed_columns)

  # Saving standard deviation of the columns which are normalized
  sd_values <- Map(sd, news_train[,needed_columns])

  # Saving mean of the columns which are normalized
  mean_values <- Map(mean, news_train[,needed_columns])

  news_train[,needed_columns] <- (news_train[,needed_columns] - mean_values) / sd_values

  return(list("sd_values"=sd_values, "mean_values"=mean_values, "news_train"=news_train))

}

# This funciton normalize continuous variables of the test datset
apply_normalization <- function(news, means, sds) {
  # All Column names
  column_names <- names(news_train)

  # Column names which needs to be ignored due to categorical and target feature

  ignored_column_names <- c("url", "timedelta", "data_channel_is_lifestyle",
                            "data_channel_is_entertainment", "data_channel_is_bus",
                            "data_channel_is_world", "data_channel_is_socmed",
                            "data_channel_is_tech", "weekday_is_monday",
                            "weekday_is_tuesday",
                            "weekday_is_wednesday", "weekday_is_thursday",
                            "weekday_is_friday",
                            "weekday_is_saturday", "weekday_is_sunday", "is_weekend",
                            "shares")

  needed_columns <- setdiff(column_names,ignored_column_names)

  news[,needed_columns] <- (news[,needed_columns] - means) / sds

  return(news)
}
```

```r
# This function creates the factor/single categorical variable by combining
# multiple/one hot encoded variables
cat_encoding <- function(news){

  dow_cols = c("weekday_is_monday", "weekday_is_tuesday", "weekday_is_wednesday",
               "weekday_is_thursday", "weekday_is_friday", "weekday_is_saturday",
               "weekday_is_sunday")

  news$cat_dow <- 0

  for (dow in dow_cols) {
    dow_idx = which(news[,dow] == 1)
    #print(dow_idx)
    news[dow_idx,"cat_dow"] <- which(dow_cols==dow)
  }

  news$cat_dow <- as.factor(news$cat_dow)

  data_channel_cols = c("data_channel_is_lifestyle", "data_channel_is_entertainment",
                        "data_channel_is_bus", "data_channel_is_socmed",
                        "data_channel_is_tech",
                        "data_channel_is_world")

  news$data_channel <- 0

  for (channel in data_channel_cols) {
    channel_idx <- which(news[,channel] == 1)
    news[channel_idx,"data_channel"] <- which(data_channel_cols==channel)
  }

  news$data_channel <- as.factor(news$data_channel)

  news$is_weekend <- as.factor(news$is_weekend)

  return(news)

}

OUTLIERS_HIGH_CUTOFF = 0.1
OUTLIERS_LOW_CUTOFF = 0.05
outliers_removal <- function(news) {
  # sort by shares
  sorted_news <- news[order(news$shares),]

  num_rows <- nrow(news)
  # remove lower tail
  cut_low_point <- as.integer(OUTLIERS_LOW_CUTOFF*num_rows)
  cut_high_point <- as.integer((1-OUTLIERS_HIGH_CUTOFF)*num_rows)
  sorted_news <- sorted_news[cut_low_point:cut_high_point, ]
  news <- sorted_news[sample(nrow(sorted_news)),]
  return(sorted_news)
}
```

```r
# This function removes the outlier from the dataset based upon the
# cook's distance
cook_outliers_removal <- function(news){

  cutoff <- 4/nrow(news)
  model <- lm(shares ~ ., data=news)
  infl <- lm.influence(model, do.coef = FALSE)

  cooks.distance <- cooks.distance(model, infl = infl,
                                   res = weighted.residuals(model),
                                   sd = sqrt(deviance(model)/df.residual(model)),
                                   hat = infl$hat)

  index <- cooks.distance <= cutoff
  news <- news[index,]

  return(news)

}

# This function loads the train data set and applies the
# data cleaning operation to it.
load_processed_train_data <- function(outliers.removed=FALSE,
                                      one.hot.encoding.remove=TRUE){

  news <- read.csv("Train.csv", header = TRUE)

  news <- data_cleaning(news)
  news <- correlation_cleaning(news)

  obj <- normalization(news)
  news <- obj$news

  news <- cat_encoding(news)

  url <- news$url
  news$url <- NULL

  if(one.hot.encoding){

    categorical_var <- c("data_channel_is_lifestyle",
                         "data_channel_is_entertainment", "data_channel_is_bus",
                         "data_channel_is_world", "data_channel_is_socmed",
                         "data_channel_is_tech", "weekday_is_monday", "weekday_is_tuesday",
                         "weekday_is_wednesday", "weekday_is_thursday", "weekday_is_friday",
                         "weekday_is_saturday", "weekday_is_sunday")

    news_with_cat <- subset(news, select = categorical_var)
    news <- subset(news, select = setdiff(names(news),categorical_var))
  }

  if(outliers.removed){
    news <- cook_outliers_removal(news)
```

```
  }

  return(news)
}
```

Stepwise Regression Model

```
set.seed(464)

news <- load_processed_train_data()

K <- 10
# 10 - fold cross validation
folds <- createFolds(news$shares, k = K, list=TRUE, returnTrain=TRUE)

models <- list()
rmses <- c()
R2s <- c()

for (i in 1:K) {

  news_train <- news[folds[[i]],]
  news_val <- news[-folds[[i]],]

  null=lm(shares~1, data=news_train)
  full=lm(shares~., data=news_train)

  model <- step(null, scope=list(lower=null, upper=full), direction="both", trace=0)
  #model <- step(full, direction="backward", trace=0)

  pred <- predict(model, news_val)

  #pred <- target_inverse(pred, lamda)
  #shares_val <- target_inverse(news_val$shares, lamda)
  #mse <- sum((pred - shares_val)**2) / nrow(news_val)

  mse <- sum((pred - news_val$shares)**2) / nrow(news_val)
  rmses <- append(rmses, sqrt(mse))

  R2s <- append(R2s, summary(model)$adj.r.squared)

  models[[i]] <- model

}

# Displaying which variables are selected in the each fold
unique_coef <- c()

for(i in 1:length(models)){
  model_coef <- names(models[[i]]$coefficients)
  unique_coef <- unique(c(model_coef, unique_coef))
}
```

```
model_variables <- data.frame(matrix(NA,nrow=length(unique_coef),ncol=length(models)+1))
model_variables$X1 <- unique_coef

for(i in 1:length(models)){

  model_coef <- names(models[[i]]$coefficients)
  tf_coef <- unique_coef %in% model_coef
  var <- paste("X", toString(i+1), sep = "")
  model_variables[var] <- tf_coef

}
```

LASSO and RIDGE (Regularization)

```
set.seed(464)

# run grid search with cross validation to select best values for lambda and alpha in elastic net
select_model <- function(news, t_lambda) {
  K = 10
  # alpha = 0 -> Ridge; alpha = 1 -> Lasso
  alphas = c(0,1)
  lambdas = c(1e-05, 1e-04, 1e-03, 1e-02, 0.1, 1.,10.)

  folds <- createFolds(news$shares, k = K, list=TRUE, returnTrain=TRUE)
  # for each combination of parameters
  for (alpha in alphas) {
    for (lambda in lambdas) {
      rmses <- c()
      R2s <- c()

      # for each fold
      for (i in 1:K) {
        news_train <- news[folds[[i]],]
        news_val <- news[-folds[[i]],]

        X_train <- data.matrix(subset(news_train,select=-shares))
        y_train <- data.matrix(news_train$shares)
        X_val <- data.matrix(subset(news_val,select=-shares))
        y_val <- data.matrix(news_val$shares)

        model <- glmnet(X_train, y_train, family="gaussian", alpha=alpha, standardize=TRUE,
                        lambda=lambda, nlambda=1)

        pred_train <- predict(model, newx=X_train, s=lambda)
        shares_train <- y_train

        # calculate R^2 in the fitted data
        ssto <- sum((shares_train - mean(shares_train))**2)
        sse <- sum((pred_train - shares_train)**2)
        R2 <- 1 - (sse/ssto)

        R2s <- append(R2s, R2)
```

```r
        pred <- predict(model, newx=X_val, s=lambda)
        shares_val <- y_val

        sse <- sum((pred - shares_val)**2)
        rmse <- sqrt(sse / nrow(news_val))

        rmses <- append(rmses,rmse)
      }
      mrmse= mean(rmses)
      srmse= sd(rmses)
      mR2 = mean(R2s)
      cat(sprintf("alpha = %f, lambda = %f, avg rmse = %f, sd rmse = %f, avg R-2 = %f\n",
                  alpha, lambda, mrmse,srmse,mR2))
    }
  }

}

news <- load_processed_train_data()
select_model(news, t_lambda)
```