Wine Quality

Liam Spoletini

6/8/2022

Part 1: Wine

Overview

Dataset Name: **Wine Quality**

Response Variable: **Quality** (Scale of 1-10, actual range of 3-8)

Predictor Variables: 1. fixed acidity ØØ 2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide 7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol

Number of Instances: **1599** Number of Missing Values: **0**

Setup and Exploratory Data Analysis

Setup

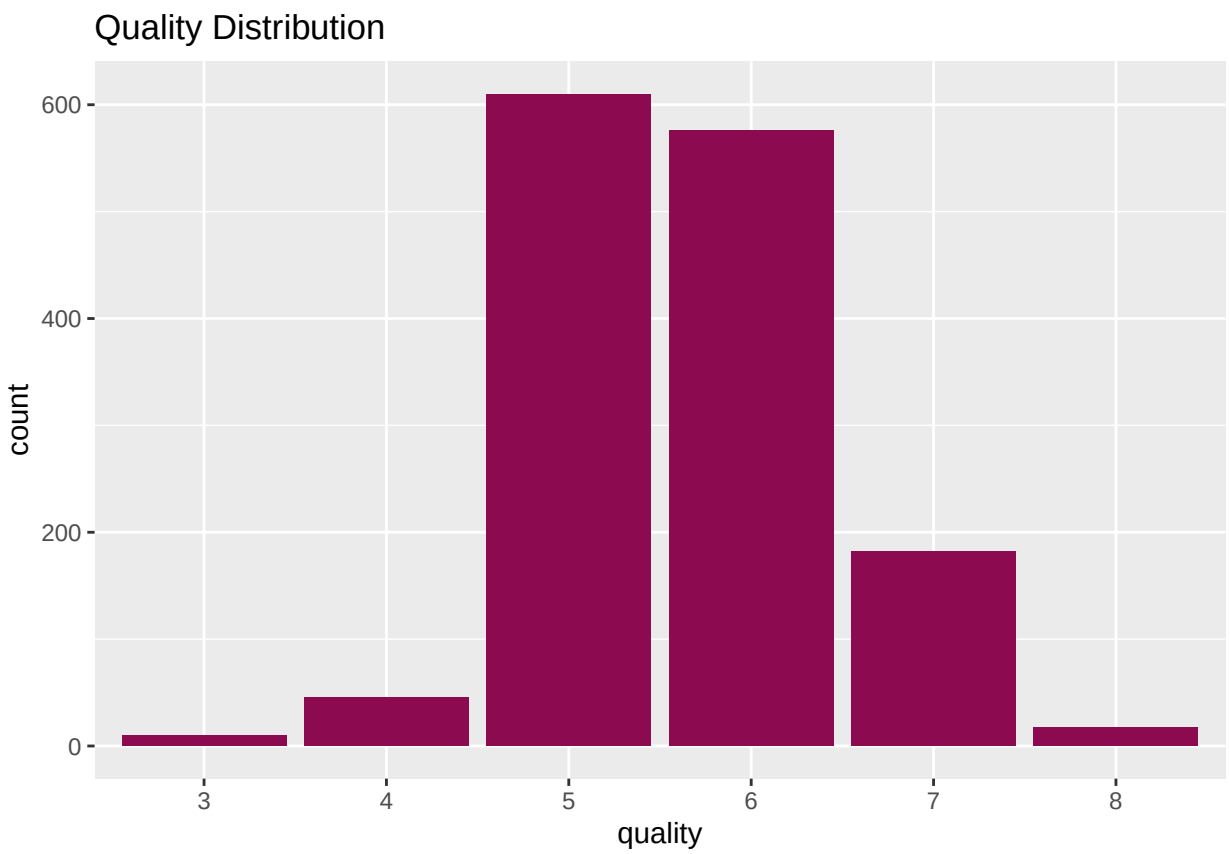
```
source(here::here("R", "Setup.R")) # See Setup.R for details
```

Train/Test Split

```
n = nrow(reds)
set.seed(42)
Z <- sample(n,n/10)
reds_test = reds[Z,]
reds_train = reds[-Z,]
```

Graphs

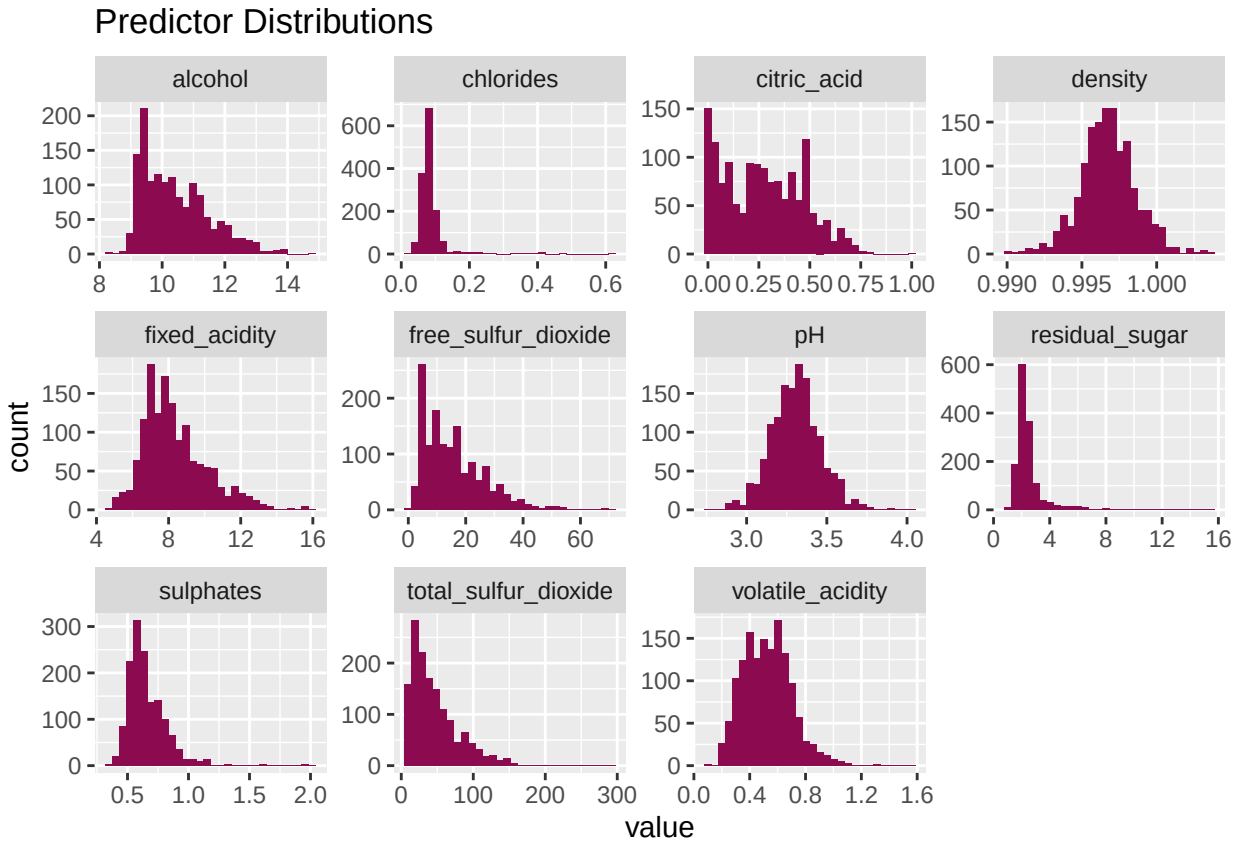
```
ggplot(reds_train) + geom_histogram(aes(x = quality), stat="count", fill = "deeppink4") + ggtitle("Quali
```



The response variable has 6 distinct values that range from 3 to 8 (the full dataset and the training data set have this in common.) This data is ordinal in nature, but there are many training instances available, so we may get good performance by using considering it to be continuous data.

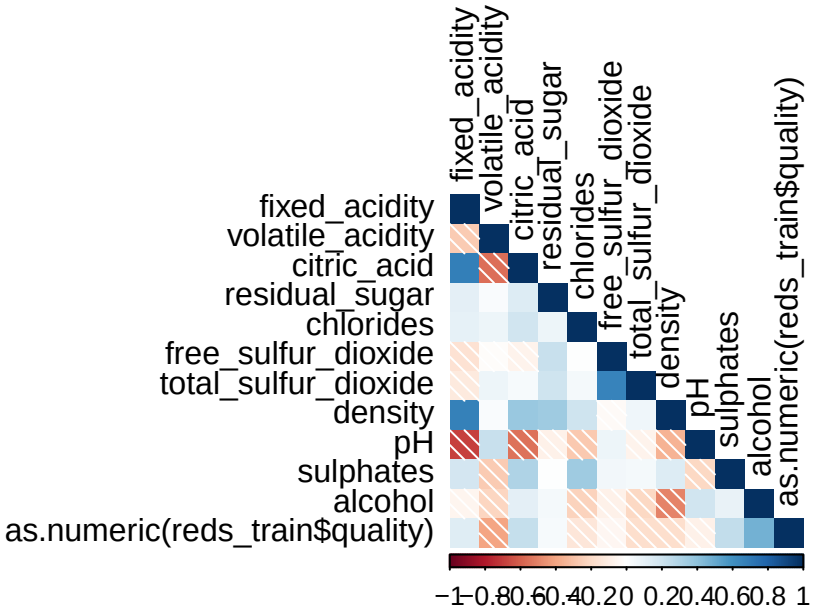
```
reds_long <- reds_train[,1:11] %>%
  pivot_longer(colnames(reds_train)[1:11]) %>% as.data.frame()
ggplot(reds_long) +
  geom_histogram(aes(x = value), fill = "deeppink4") +
  facet_wrap( ~ name, scales = "free") +
  ggtitle("Predictor Distributions")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Few predictors are close to normally distributed (pH and density), and all the variables are on very different scales. This will not be relevant for the methods selected.

```
res <- cor(cbind(reds_train[,1:11], as.numeric(reds_train$quality)))
corrplot::corrplot(res, method = "shade", type="lower", tl.col = "black")
```



The correlation plot shows some potential multicollinearity, as fixed acidity is correlated with citric acid and fixed acidity. None of the variables are strongly correlated with the response variable (when considered to be continuous).

Approach Since the response variable is an ordinal variable, it may make sense to choose a technique that takes this into account.

Machine Learning Method:

Ordinal Logistic Regression (Ordered Logit) and **Conversion to Binary Classification by choosing a threshold value**, both in combination with **Principal Components Analysis**

Since the quality data are ordinal, treating it as a continuous variable is not ideal. Ordered Logit and reframing the problem as a binary classification are potentially good approaches. Because of potential multicollinearity, decomposing the input features into principal components may be advantageous.

It may be the case that someone is interested in a way to identify wine above a certain quality threshold. In this case, we would need to identify a cutoff value (5.5 would be a roughly even split) to use to binarize the data.

Assumptions + Considerations

Ordered Logit

Proportional Odds Assumption:

"No input variable has a disproportionate effect on a specific level of the outcome variable." This implies that the coefficents are the same when determining the odds of an instance belonging to a single response

value, and that the only value that differs between groups is the intercept term.

This assumption may be assessed using the Brant-Wald test, which generates Chi-Squared-distributed values by comparing a multinomial logistic regression model coefficients to those from a proportional odds model.

Brant-Wald Test

```
model_olr <- MASS::polr(quality ~ ., data = reds_train)
brant(model_olr)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## ------------------------------------------------------
## Test for     X2  df  probability
## ------------------------------------------------------
## Omnibus          111.78  44  0
## fixed_acidity        5.53    4   0.24
## volatile_acidity 11.73   4   0.02
## citric_acid      7.44    4   0.11
## residual_sugar       11.59   4   0.02
## chlorides        5.9 4   0.21
## free_sulfur_dioxide  4.97    4   0.29
## total_sulfur_dioxide 18.88   4   0
## density          7.87    4   0.1
## pH           11.84   4   0.02
## sulphates        5.56    4   0.23
## alcohol          1.91    4   0.75
## ------------------------------------------------------
##
## H0: Parallel Regression Assumption holds
```

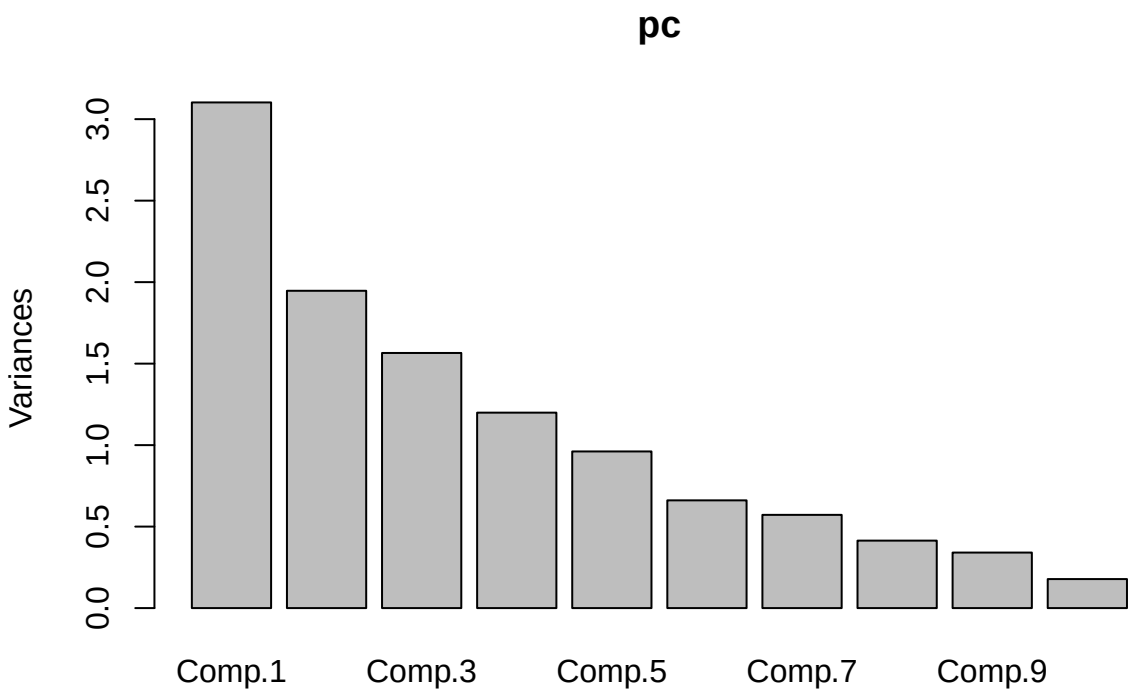
The proportional odds/parallel regression assumption does not hold in this case. Specifically, volatile acidity, free sulfur dioxide, and pH seem to violate this assumption. Since this is the case, we may have to proceed with a multinomial logistic regression.

Tuning Parameters

The tuning parameter in PCR is the **number of Principal Components**. A higher number of principal components reduces bias but increases variance.

Implementation and tuning: PCR

```
pc <- princomp(reds_train[,1:11],cor=T)
plot(pc)
```



The first few principal components do not capture much of the variance in the predictors. This may indicate that PCR is not well-suited for this data.

Repeating the Brant-Wald Test on the Principal Components

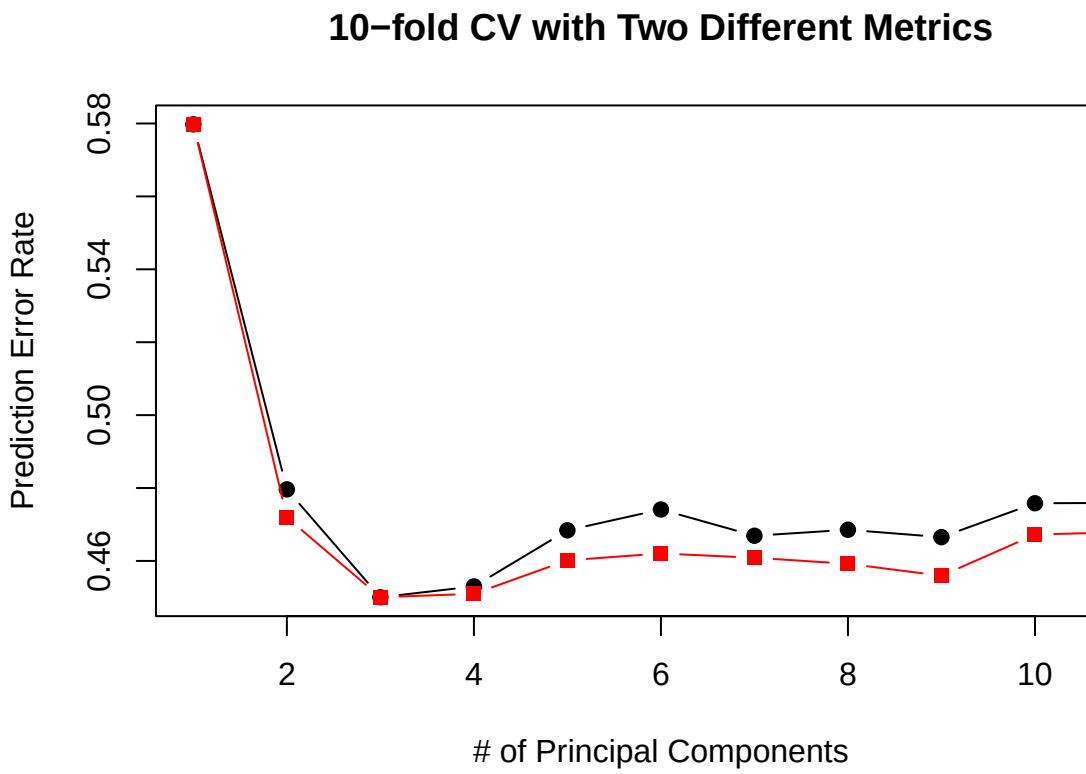
```
p_mat
```

```
##               [,1]
##  [1,] 2.231714e-02
##  [2,] 3.879316e-10
##  [3,] 7.872666e-07
##  [4,] 1.277809e-05
##  [5,] 3.051798e-09
##  [6,] 1.500352e-08
##  [7,] 1.182007e-08
##  [8,] 1.930281e-08
##  [9,] 1.476675e-08
## [10,] 4.013037e-08
## [11,] 8.134802e-08
```

All ordered logit models using 1-10 principal components failed the Brant-Wald test for the proportional odds assumption. Therefore, I will proceed with multinomial logistic regression, which does not account for order among quality groups.

Tuning: Selecting Number of Principal Components

```
# Cross Validation: # of PC's

set.seed(42)
ids <- sample(1:10, nrow(reds_train), replace = TRUE)
CV_plot_a <- {}
CV_plot_b <- {}
Q = 11
for(i in 1:Q){
  CV_tmp_a <- {}
  CV_tmp_b <- {}
  for(k in 1:10){
    reds_train_train = reds_train[!(ids == k),]
    reds_train_val = reds_train[ids == k,]
    pc_t <- princomp(reds_train_train[,1:11],cor=T)
    pct_scores <- as.data.frame(pc_t$scores[,1:i])
    pc_v <- princomp(reds_train_val[,1:11],cor=T)
    pcv_scores <- as.data.frame(pc_v$scores[,1:i])
    train_df <- data.frame(pc_t$scores[,1:i])
    t_names = {}
    for(q in 1:i){
      t_names = append(t_names, paste("pc", q, sep=""))
    }
    names(pct_scores) <- t_names
    names(pcv_scores) <- t_names

    tdf <- cbind(pct_scores, quality = reds_train_train$quality)
    vdf <- cbind(pcv_scores, quality = reds_train_val$quality)

    model_multi <- nnet::multinom(quality ~ . , data = tdf, trace = F)

    CV_tmp_a[k] <- sum(predict(model_multi, newdata= vdf) != reds_train_val$quality) / nrow(vdf)
    CV_tmp_b[k] <- sum(((as.numeric(predict(model_multi, newdata = vdf))+2) - (as.numeric(vdf$quality)+2
  }
  CV_plot_a[i] = mean(CV_tmp_a)
  CV_plot_b[i] = mean(CV_tmp_b)/nrow(vdf)

}


plot(1:Q, CV_plot_a, type = "b", pch=19, main = "10-fold CV with Two Different Metrics",
     ylab = "Prediction Error Rate", xlab = "# of Principal Components")
par(new=T)
plot(1:Q, CV_plot_b, type = "b", col = "red", ylab="", xlab = "", axes=F, pch=15)
axis(4, col="red", col.axis="red")
mtext(side=4, "MSE", col = "red", line = -2)
```



10-Fold Cross-Validation

To assess overfitting in this approach, I used both the prediction error rate (whether or not the correct class was predicted) and the mean squared error that results from treating quality as a ordinal/continuous variable after the fact. Both methods yielded a minimum CV error with 3 principal components, but each the errors do not increase drastically as additional principal components are considered. I will use this graph to tune the principal component number to 3.

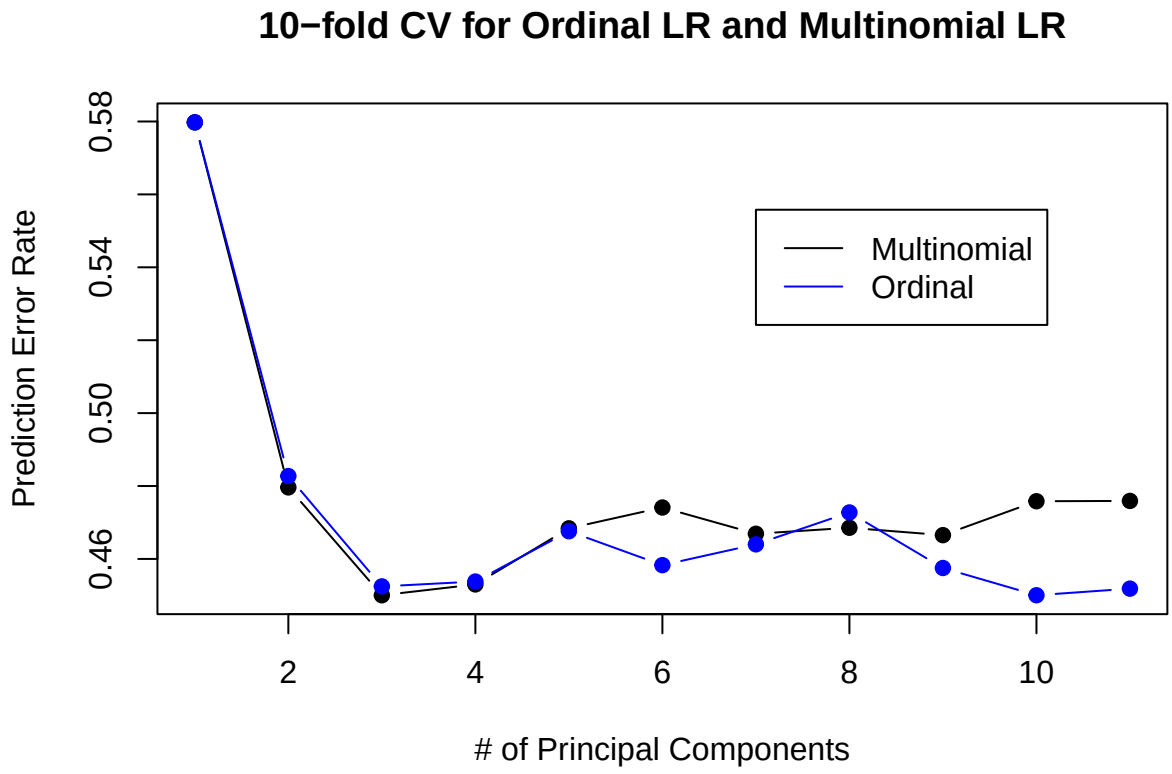
Comparison to True Ordinal Logistic Regression

Some suggest that failing the satisfy the proportional odds requirement does not doom that method entirely, so I want to run through it quickly to see if the cross-validation scores are comparable

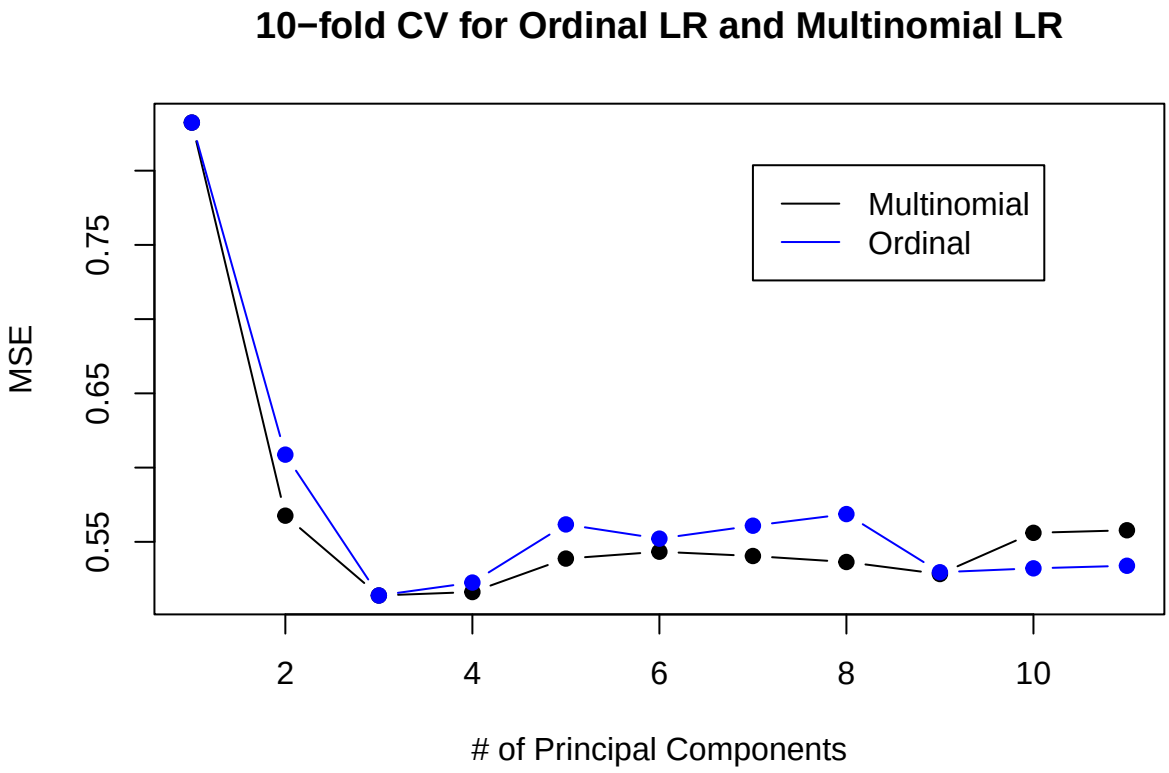
```
Q=11
CV_plot_a_olr = {}
CV_plot_b_olr = {}
for(i in 1:Q){
  CV_tmp_a_olr <- {}
  CV_tmp_b_olr <- {}
  for(k in 1:10){
    reds_train_train = reds_train[!(ids == k),]
    reds_train_val = reds_train[ids == k,]
    pc_t <- princomp(reds_train_train[,1:11],cor=T)
    pct_scores <- as.data.frame(pc_t$scores[,1:i])
    pc_v <- princomp(reds_train_val[,1:11],cor=T)
    pcv_scores <- as.data.frame(pc_v$scores[,1:i])
    train_df <- data.frame(pc_t$scores[,1:i])
    t_names = {}
```

```
    for(q in 1:i){
      t_names = append(t_names, paste("pc", q, sep=""))
    }

    names(pct_scores) <- t_names
    names(pcv_scores) <- t_names

    tdf <- cbind(pct_scores, quality = reds_train_train$quality)
    vdf <- cbind(pcv_scores, quality = reds_train_val$quality)

    model_olr <- MASS::polr(quality ~ ., data=tdf) # Only difference from above block

    CV_tmp_a_olr[k] <- sum(predict(model_olr, newdata= vdf) != vdf$quality) / nrow(vdf)
    CV_tmp_b_olr[k] <- sum(((as.numeric(predict(model_olr, newdata = vdf))+2) - (as.numeric(vdf$quality)

  }
  CV_plot_a_olr[i] = mean(CV_tmp_a_olr)
  CV_plot_b_olr[i] = mean(CV_tmp_b_olr)/nrow(vdf)
}
```

```
plot(1:Q, CV_plot_a, type = "b", pch=19, main = "10-fold CV for Ordinal LR and Multinomial LR",
     ylab = "Prediction Error Rate", xlab = "# of Principal Components")
par(new=T)
plot(1:Q, CV_plot_a_olr, type = "b", col = "blue", ylab="", xlab = "", axes=F, pch=19)
legend(7,.54,legend = c("Multinomial", "Ordinal"), col = c("black", "blue"), lty = 1:1)
```



```
plot(1:Q, CV_plot_b, type = "b", pch=19, main = "10-fold CV for Ordinal LR and Multinomial LR",
     ylab = "MSE", xlab = "# of Principal Components")
par(new=T)
plot(1:Q, CV_plot_b_olr, type = "b", col = "blue", ylab="", xlab = "", axes=F, pch=19)
legend(7,.7,legend = c("Multinomial", "Ordinal"), col = c("black", "blue"), lty = 1:1)
```



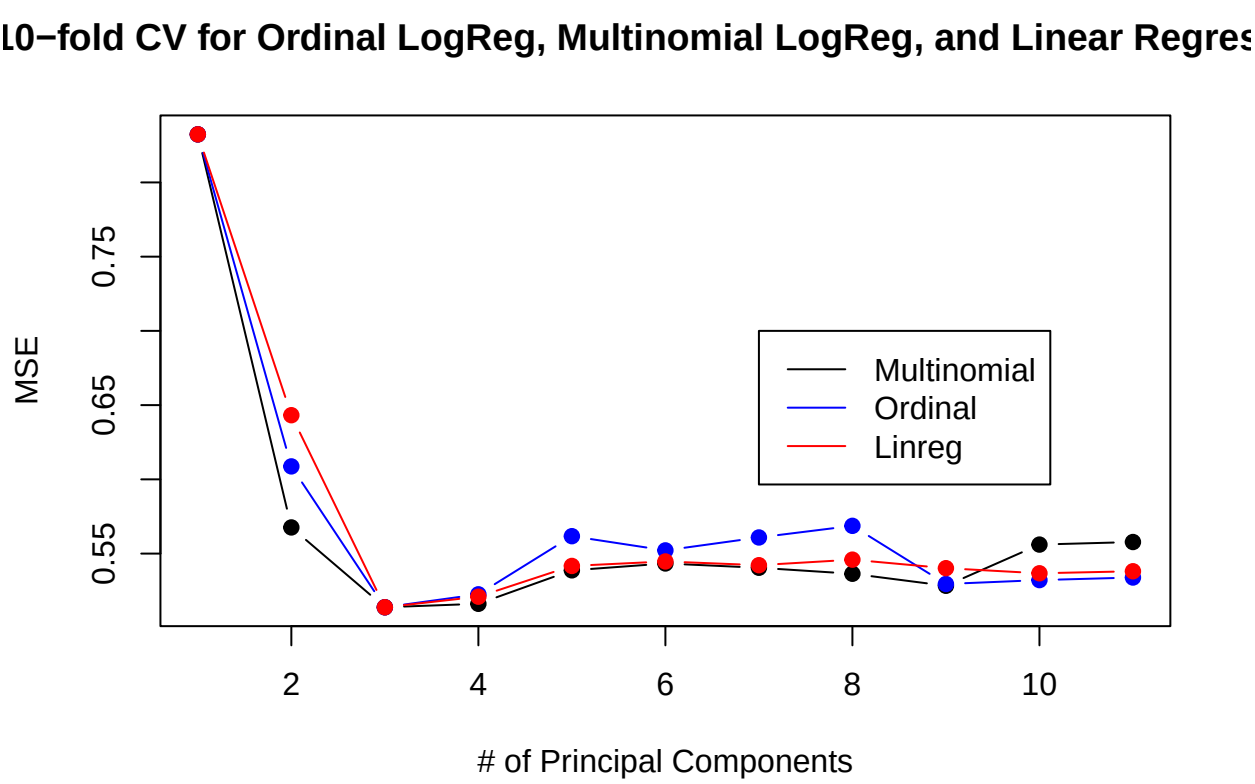
When comparing the ordinal logistic regression model to the multinomial logistic regression model, I found that there is a new principal component number (10) at which the prediction error rate is minimized. Also, the number of principal components that minimizes the MSE is the same (3), but is slightly lower for the ordinal model. This difference of around 0.006 does not support a wide difference in predictive ability, and I believe they would perform similarly on the test dataset.

Comparison to Multivariable Linear Regression

```
Q=11
CV_plot_b_linreg = {}
for(i in 1:Q){
  CV_tmp_b_linreg <- {}
  for(k in 1:10){
    reds_train_train = reds_train[!(ids == k),]
    reds_train_val = reds_train[ids == k,]
    pc_t <- princomp(reds_train_train[,1:11],cor=T)
    pct_scores <- as.data.frame(pc_t$scores[,1:i])
    pc_v <- princomp(reds_train_val[,1:11],cor=T)
```

```
    pcv_scores <- as.data.frame(pc_v$scores[,1:i])
    train_df <- data.frame(pc_t$scores[,1:i])
    t_names = {}
    for(q in 1:i){
      t_names = append(t_names, paste("pc", q, sep=""))
    }

    names(pct_scores) <- t_names
    names(pcv_scores) <- t_names

    tdf <- cbind(pct_scores, quality = reds_train_train$quality)
    vdf <- cbind(pcv_scores, quality = reds_train_val$quality)

    model_linreg <- lm(as.numeric(quality) + 2 ~ ., data=tdf)

    CV_tmp_b_linreg[k] <- sum((as.numeric(predict(model_linreg, newdata = vdf)) - (as.numeric(vdf$quali

  }
  CV_plot_b_linreg[i] = mean(CV_tmp_b_linreg)/nrow(vdf)
}
```

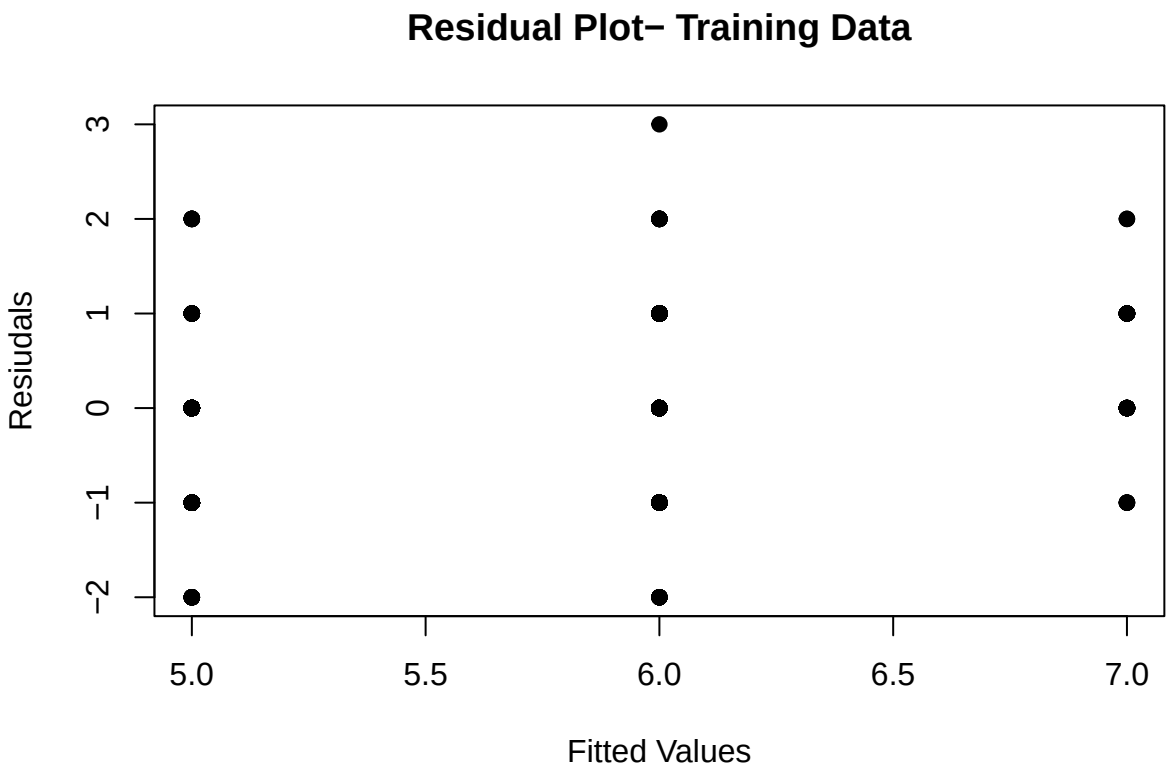
```
plot(1:Q, CV_plot_b, type = "b", pch=19, main = "10-fold CV for Ordinal LogReg, Multinomial LogReg, and
     ylab = "MSE", xlab = "# of Principal Components")
legend(7,.7,legend = c("Multinomial", "Ordinal", "Linreg"), col = c("black", "blue", "red"), lty = 1:1)

par(new=T)
plot(1:Q, CV_plot_b_olr, type = "b", col = "blue", ylab="", xlab = "", axes=F, pch=19)
par(new=T)
plot(1:Q, CV_plot_b_linreg, type = "b", col = "red", ylab="", xlab = "", axes=F, pch=19)
```



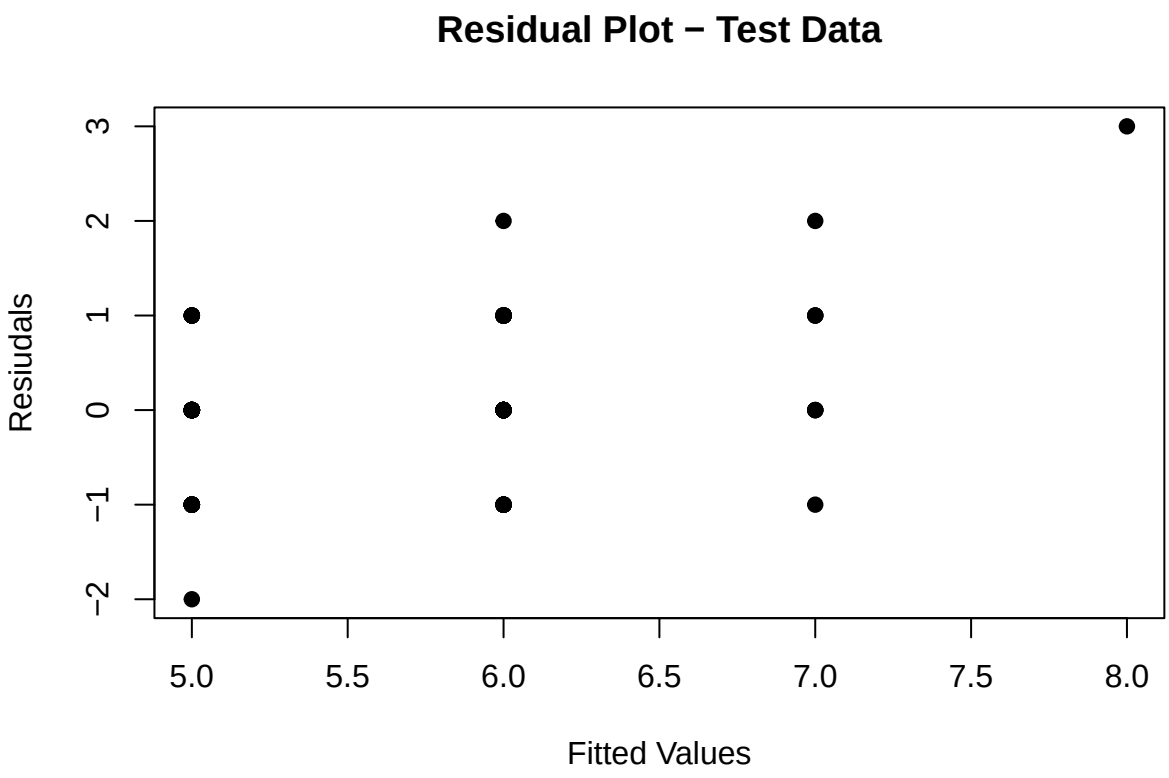
At 3 principal components, the ordinallogistic regression, multinomial logistic regression, and linear regression perform almost exactly the same with regards to MSE, and 3 principal components is the minimum of this graph.

Testing

```
# Train
pc_t <- princomp(reds_train[,1:11],cor=T)
pct_scores <- as.data.frame(pc_t$scores[,1:3])
names(pct_scores) <- c("pc1","pc2", "pc3")

tdf <- cbind(pct_scores, quality=reds_train$quality)
model_olr_final <- MASS::polr(quality ~ ., data=tdf)

# Test
pc_test <- princomp(reds_test[,1:11],cor=T)
pctest_scores <- as.data.frame(pc_test$scores[,1:3])
names(pctest_scores) <- c("pc1","pc2", "pc3")

testdf <- cbind(pctest_scores, quality=reds_test$quality)

train_guesses <- as.numeric(predict(model_olr_final, newdata=tdf)) + 2
plot(train_guesses, train_guesses-(as.numeric(reds_train$quality)+2), pch=19,
     main ="Residual Plot- Training Data", xlab = "Fitted Values", ylab= "Resiudals")
```




```
test_guesses <- as.numeric(predict(model_olr_final, newdata=testdf)) + 2
plot(test_guesses, test_guesses-(as.numeric(reds_test$quality) + 2), pch=19,
     main ="Residual Plot - Test Data", xlab = "Fitted Values", ylab= "Resiudals")
```




```
sum(((as.numeric(predict(model_olr_final, newdata = testdf)) + 2) - (as.numeric(testdf$quality) + 2) )^
```

```
## [1] 0.509434
```

On average, the squared residual is around 0.46, interestingly lower than the average 10-fold CV error from earlier.

How is this compared to guessing the median every time?

```
sum((5.5 - as.numeric(testdf$quality)+2)^2)/nrow(testdf)
```

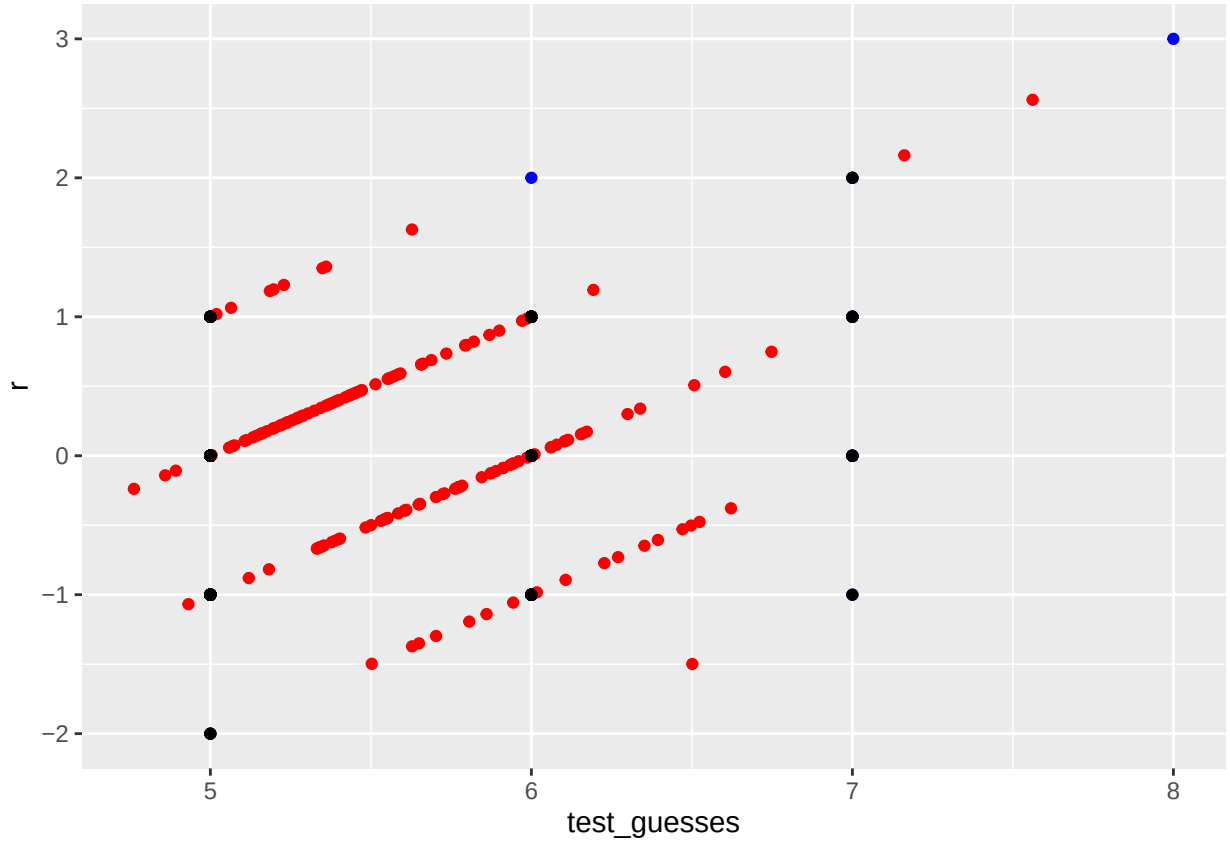
```
## [1] 16.02358
```

Better!

Repeat and with Multinomial and Linear Regression

```
# Train
model_olr_final <- MASS::polr(quality ~ ., data=tdf)
model_linreg_final <- lm(as.numeric(quality) + 2 ~ ., data = tdf)
model_multinom_final <- multinom(quality ~ ., data = tdf, trace = F)
# Test
```

```
test_guesses_linreg <- predict(model_linreg_final, newdata=testdf)
test_guesses_multinom <- as.numeric(predict(model_multinom_final, newdata=testdf)) + 2

plotdf <- tibble(test_guesses, test_guesses_linreg, test_guesses_multinom,
                 r = test_guesses - (as.numeric(reds_test$quality) + 2),
                 r_linreg = test_guesses_linreg - (as.numeric(reds_test$quality) + 2),
                 r_multinom = test_guesses_multinom - (as.numeric(reds_test$quality) + 2))
ggplot(plotdf) +
  geom_point(aes(x=test_guesses, y = r), color = "blue") +
  geom_point(aes(x=test_guesses_linreg, y = r_linreg), color = "red") +
  geom_point(aes(x=test_guesses_multinom, y = r_multinom), color = "black")
```




```
results = data.frame(ordinal_reg_mse = sum((test_guesses - (as.numeric(testdf$quality) + 2) )^2)/nrow(te
                     linear_reg_mse = sum((test_guesses_linreg - (as.numeric(testdf$quality) + 2) )^2)/n
                     multinom_reg_mse = sum((test_guesses_multinom - (as.numeric(testdf$quality) + 2) )
results
```

```
##   ordinal_reg_mse linear_reg_mse multinom_reg_mse
## 1        0.509434      0.4453573        0.4654088
```

Conclusion

Though these methods produce very similar results, ordinal regression has the worst test MSE and extreme values in the residual plot.