Final Project

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
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
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
## Warning: package 'glmnet' was built under R version 4.1.2
```

```
## Warning: package 'VGAM' was built under R version 4.1.2
```

```
## Warning: package 'nnet' was built under R version 4.1.2
```

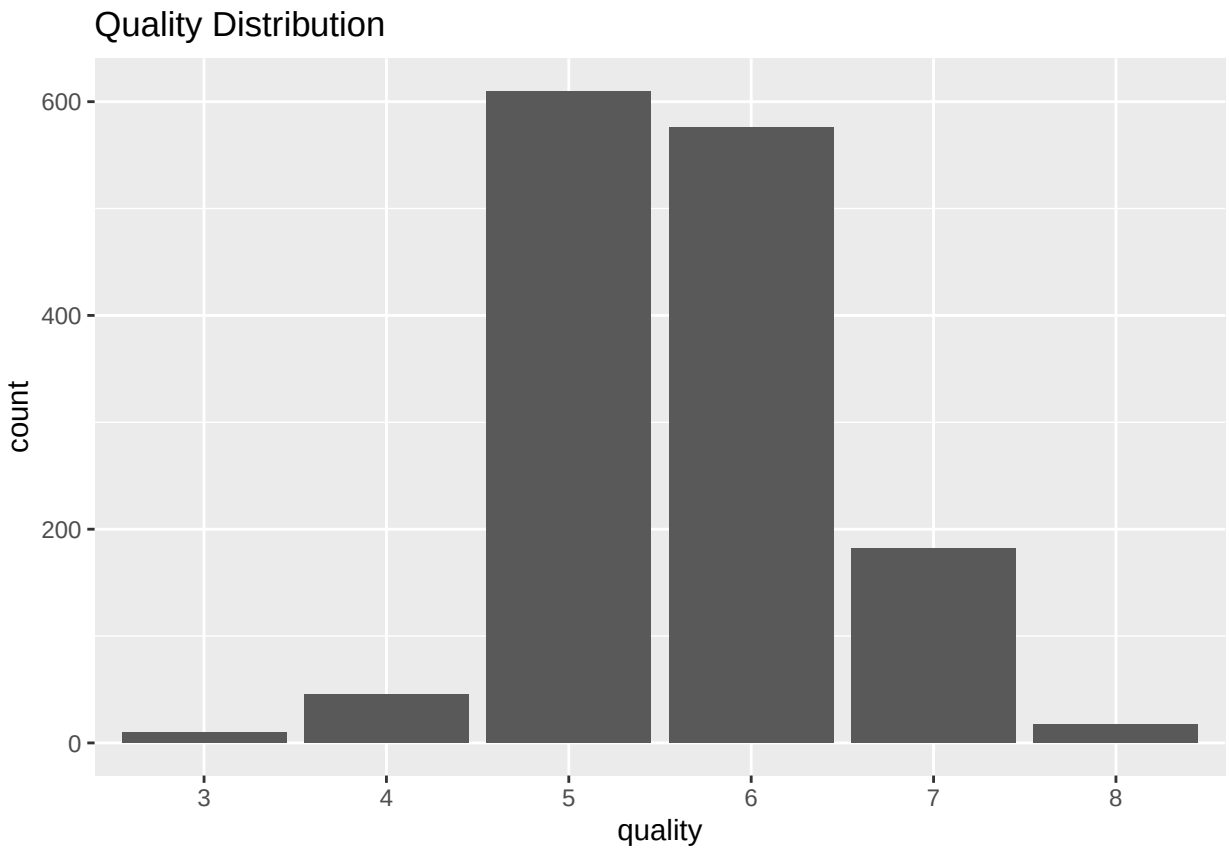
```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
```

Train/Test Split

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

Graphs

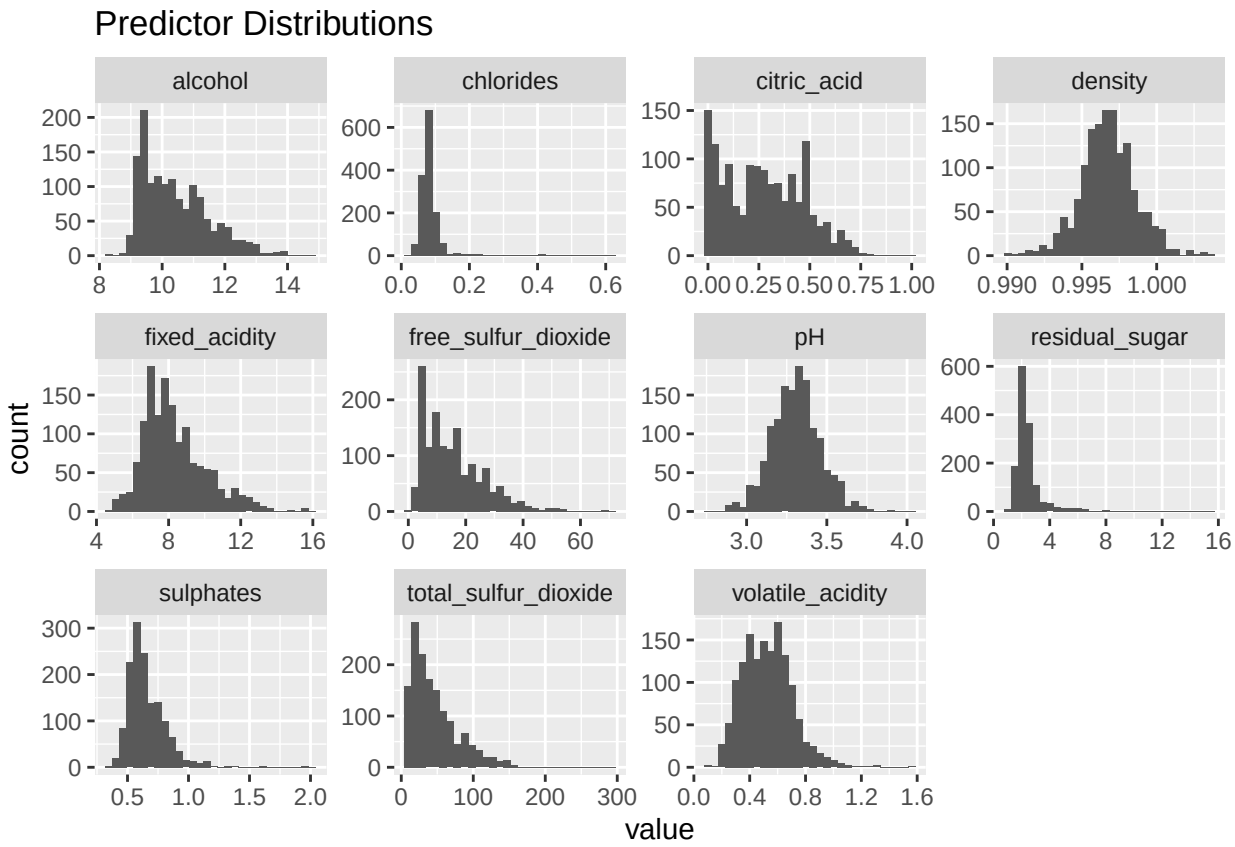
```
ggplot(reds_train) + geom_histogram(aes(x = quality), stat="count") + ggtitle("Quality Distribution")
```



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.

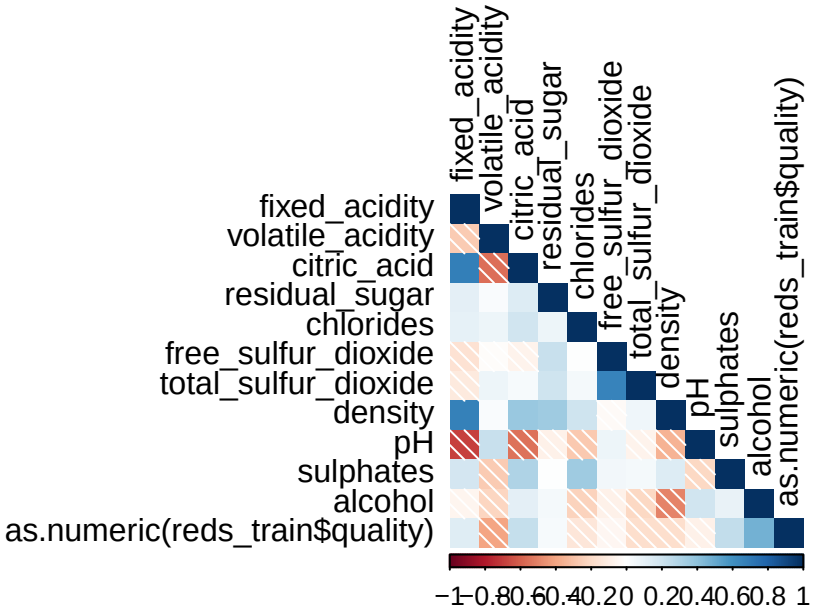
```
reds_long <- reds_train[,1:11] %>%
  pivot_longer(colnames(reds_train)[1:11]) %>% as.data.frame()
ggplot(reds_long) +
  geom_histogram(aes(x = value)) +
  facet_wrap( ~ name, scales = "free") +
  ggtitle("Predictor Distributions")
```

```
## `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
```

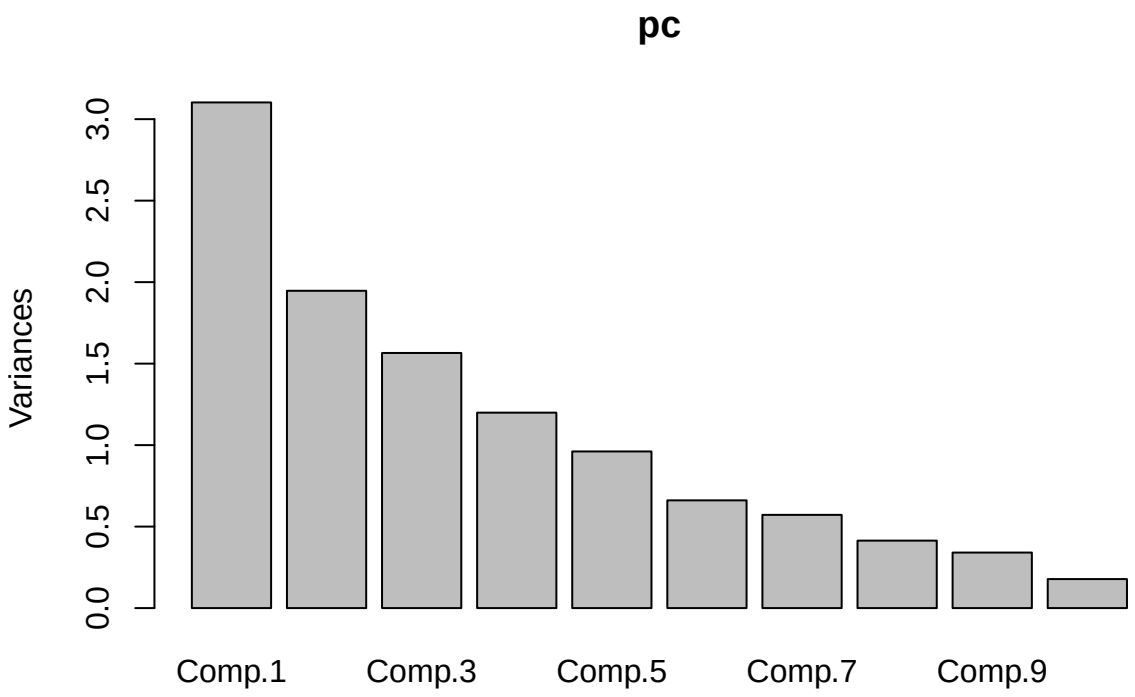
The proportional odds/parallel regression assumption does not hold in this case. Specifically, volatile acidity, free sulfur dioxide, and pH seem to violate this assumption. Since this is the case, we may have to proceed with a multinomial logistic regression.

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

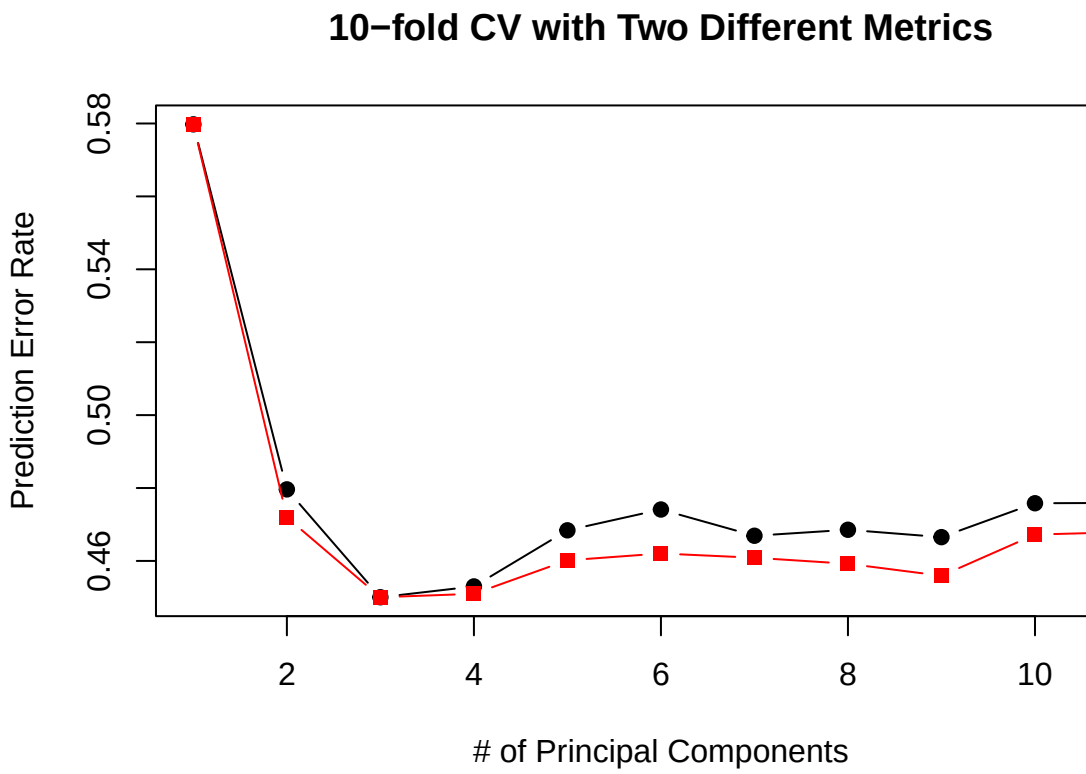
```
p_mat
```

```
##               [,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$quality) / nrow(vdf)
    CV_tmp_b[k] <- sum((as.numeric(predict(model_multi, newdata = vdf)) - as.numeric(vdf$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.

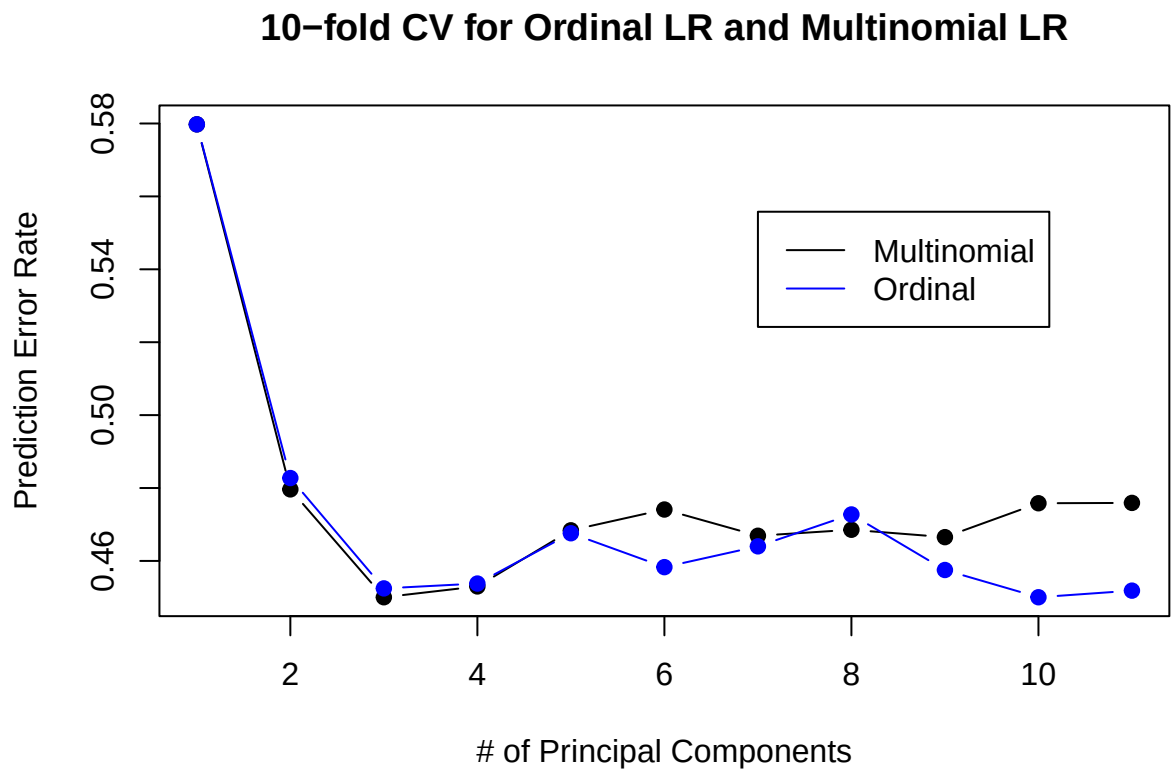
Comparison Back to True Ordinal Logistic Regression

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

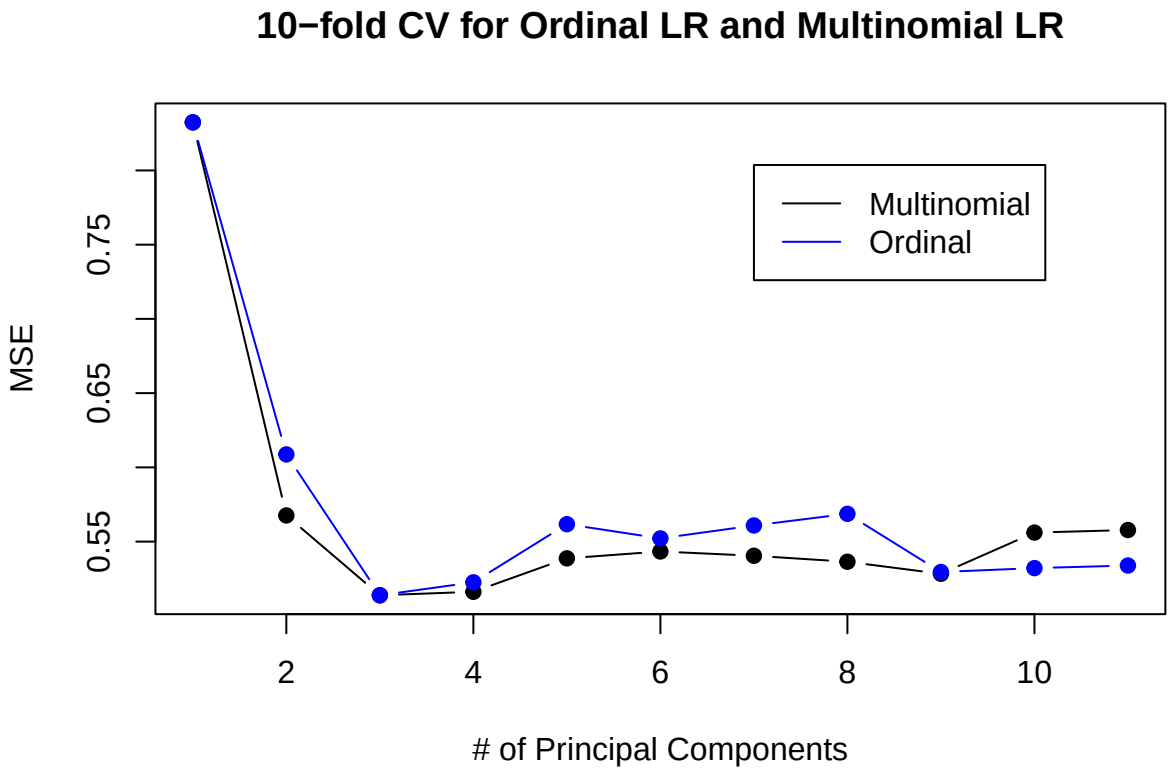
```
Q=11
CV_plot_a_olr = {}
CV_plot_b_olr = {}
for(i in 1:Q){
  CV_tmp_a_olr <- {}
  CV_tmp_b_olr <- {}
  for(k in 1:10){
    reds_train_train = reds_train[!(ids == k),]
    reds_train_val = reds_train[ids == k,]
    pc_t <- princomp(reds_train_train[,1:11],cor=T)
    pct_scores <- as.data.frame(pc_t$scores[,1:i])
    pc_v <- princomp(reds_train_val[,1:11],cor=T)
    pcv_scores <- as.data.frame(pc_v$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$quality) / nrow(vdf)
    CV_tmp_b_olr[k] <- sum((as.numeric(predict(model_olr, newdata = vdf)) - as.numeric(vdf$quality))^2)

  }
  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)
```



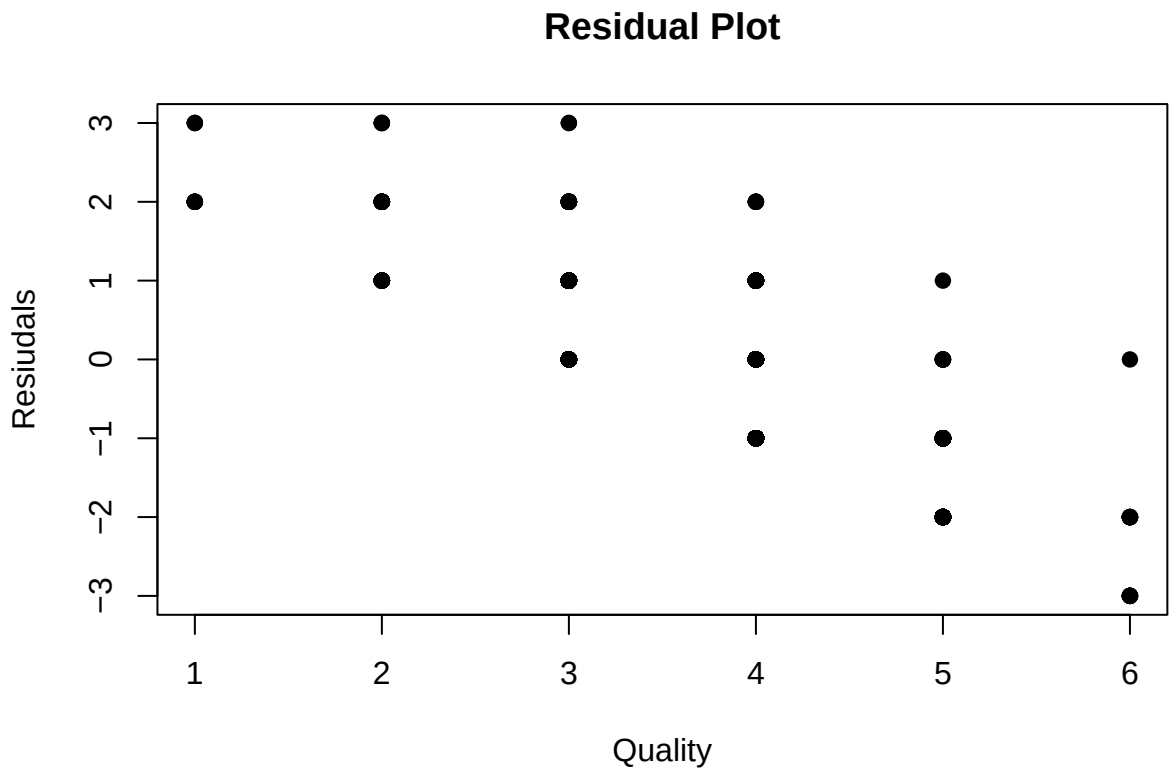
When comparing the ordinal logistic regression model to the multinomial logistic regression model, I found that there is a new principal component number (10) at which the prediction error rate is minimized. Also, the number of principal components that minimizes the MSE is the same (3), but is slightly 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.

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=testdf))
plot(as.numeric(reds_train$quality), train_guesses-as.numeric(reds_train$quality), pch=19,
     main ="Residual Plot", xlab = "Quality", ylab= "Resiudals")
```

```
## Warning in train_guesses - as.numeric(reds_train$quality): longer object length
## is not a multiple of shorter object length
```




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

```
## [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)/nrow(testdf)
```

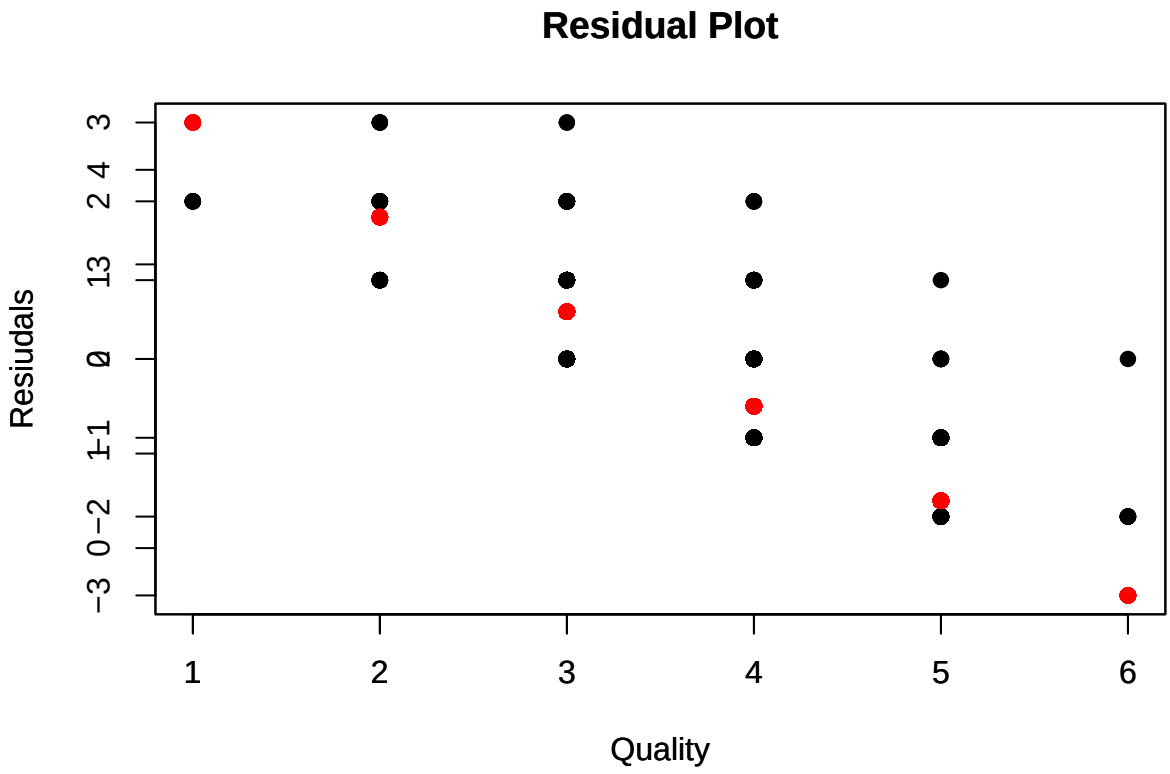
```
## [1] 4.312893
```

Better!

```
plot(as.numeric(tdf$quality), train_guesses-as.numeric(tdf$quality), pch=19,
     main ="Residual Plot", xlab = "Quality", ylab= "Resiudals")
```

```
## Warning in train_guesses - as.numeric(tdf$quality): longer object length is not
## a multiple of shorter object length
```

```
par(new=T)
plot(as.numeric(tdf$quality), 5.5-as.numeric(tdf$quality), pch=19,
     main ="Residual Plot", xlab = "Quality", ylab= "Resiudals", col="red")
```



Part 2: Hearts

Overview

Dataset

Name: **Heart Disease Data set**

Response Variable:
Angiographic disease status:
0: <50% diameter narrowing
1: >50% diameter narrowing

Predictor Variables (13 total):
Categorical:
1. Sex
0 = female
1 = male

2. Chest Pain Type (cp)
   1 = typical angina 2 = atypical angina 3 = non-anginal pain 4 = asymptomatic

3. Fasting Blood Sugar (fbs)
   0 = fbs $<$ 120 mg/dl 1 = fbs $>$ 120 mg/dl

4. Resting Electrocardiographic results (restecg)
   0 = normal
   1 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV)
   2 = showing probable or definite left ventricular hypertrophy by Estes' criteria

5. Exercise-Induced Angina (exang)
   0 = no
   1 = yes

6. The slope of the peak exercise ST segment (slope)
   1 = upsloping
   2 = flat
   3 = downsloping

7. thal
   3 = normal
   6 = fixed defect
   7 = reversible defect

Quantitative

1. Age

2. resting blood pressure (in mm Hg on admission to the hospital) (trestbps)

3. serum cholestoral in mg/dl (chol)

4. maximum heart rate achieved (thalach)

5. ST depression induced by exercise relative to rest (oldpeak)

6. Number of major vessels (0-3) colored by flourosopy (ca)

Missing Data

```
heart_data %>%
  mutate(ca = as.numeric(ca), thal = as.numeric(thal)) %>%
  filter(!is.na(ca), !is.na(thal)) -> heart_data
dim(heart_data)
```

```
## [1] 299  14
```

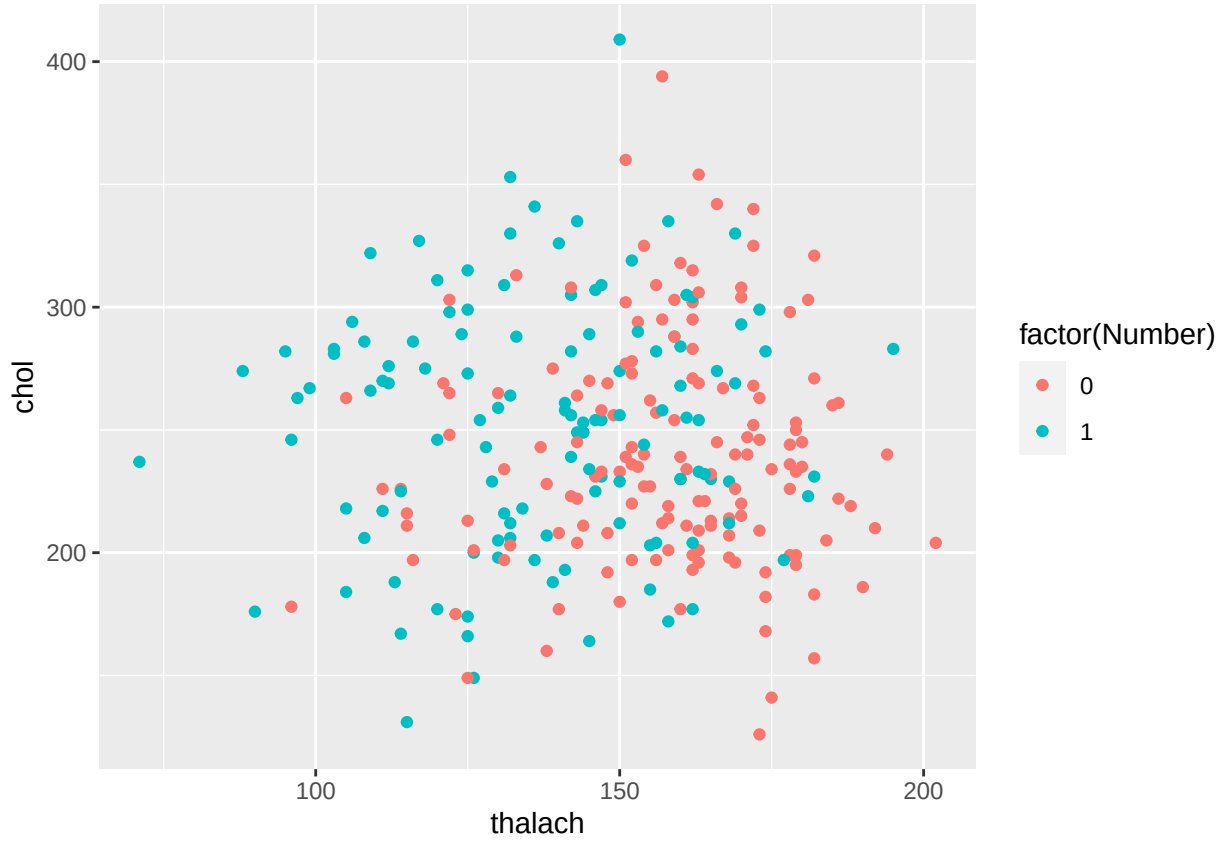
There are 297 instances without NA's.

Train/Test Split

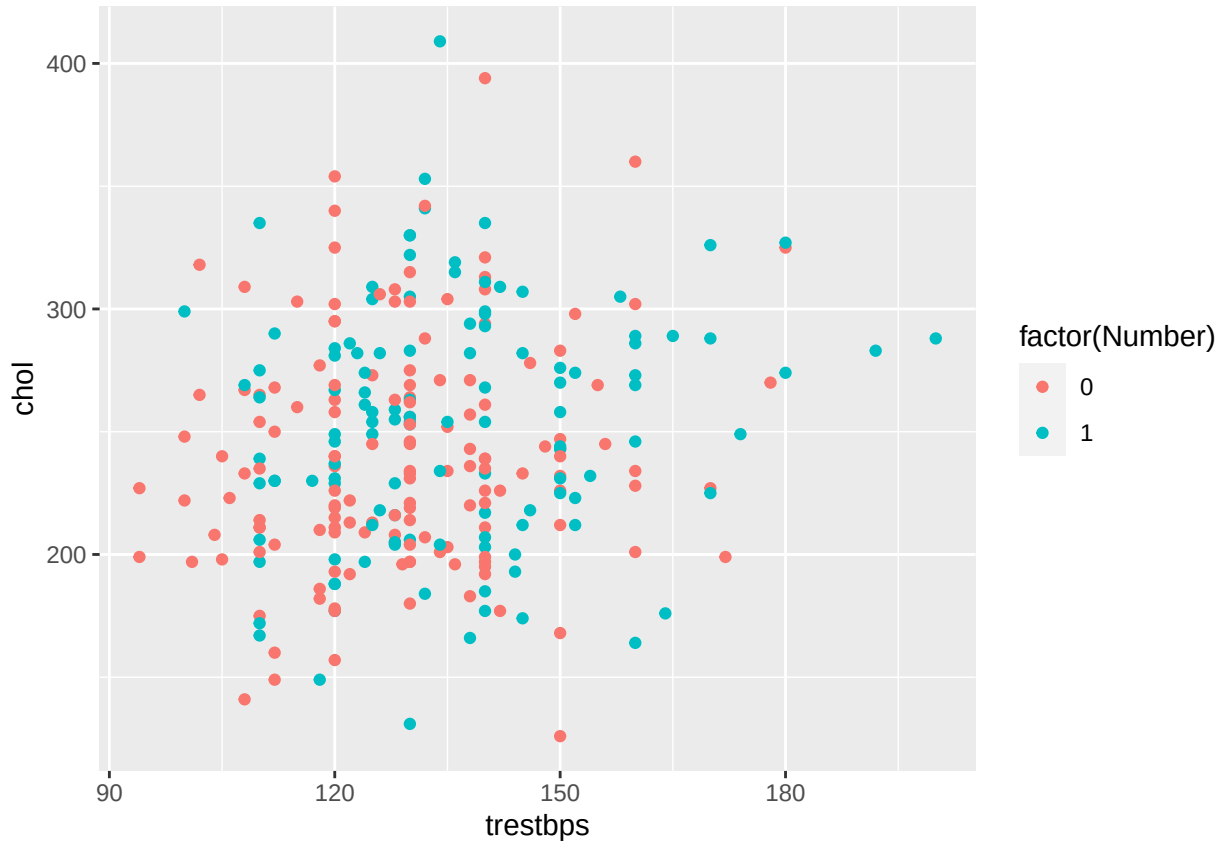
```
n = nrow(heart_data)
set.seed(42)
Z <- sample(n,n/10)
heart_test = heart_data[Z,]
heart_train = heart_data[-Z,]
```

Exploratory Data Analysis

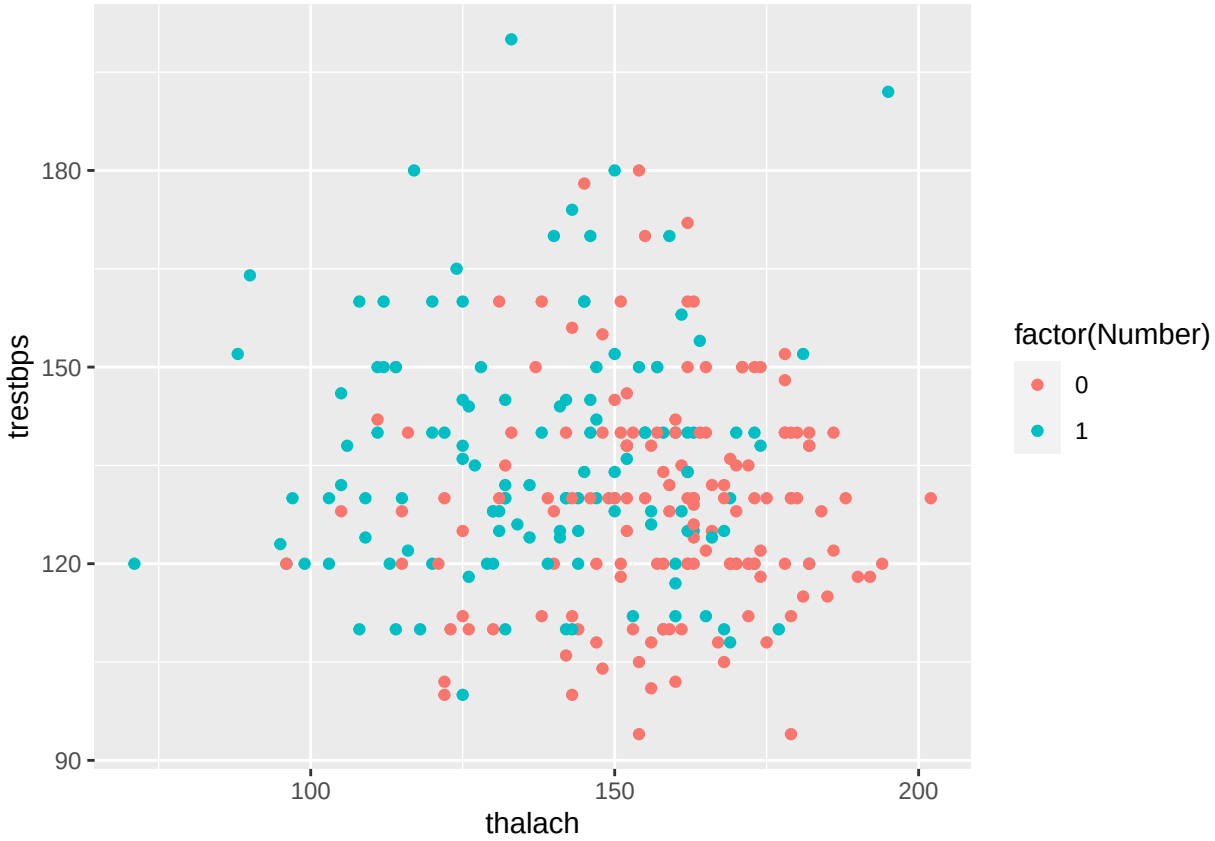
```
ggplot(heart_train) +
  geom_point(aes(x= thalach, y = chol, color=factor(Number)))
```




```
ggplot(heart_train) +
  geom_point(aes(x= trestbps, y = chol, color=factor(Number)))
```




```
ggplot(heart_train) +
  geom_point(aes(x= thalach, y = trestbps, color=factor(Number)))
```



The data do not seem to be easily separable when looking at a low number of predictor variables. Above are a subset of three such combinations of quantitative variables. It is possible that the data is still closer to linearly separable using more of the predictor variables at a time.

Machine Learning Methods: Support Vector Machine

Formal Representation of Method:

Support Vector Machine classifiers rely on the idea of a hyperplane that divides the p-dimensional feature space. This hyperplane is chosen to maximize the margin (the distance between the hyperplane and the training instances in the feature space). In the case of Support Vector Machines, this hyperplane contains soft margins, and the cost function to be minimzed is a sum of all the violations of the soft margins. The maximum sum of the cost function itself is a value that can be tuned to produce stricter or more lenient boundaries.

Assumptions/Considerations

The textbook doesn't mention any assumptions made about the data that are necessary for SVM.

Tuning Parameter

The tuning parameter for Support Vector Machines is C, which is the maximum allowed sum of violations to the margin. A lower value of C allows for fewer violations, and a higher value of C allows for more violations. The type of kernel is another way we can tune the model. If we transform the original data using a kernel,

the SVM can create a boundary that is linear in the new feature space but nonlinear in the original feature space.

Implementation: SVM

Finding the optimal kernel and C combination:

Using 10-fold Cross-Validation on the training set

```
set.seed(42)
rg <- list(kernel=c("linear","polynomial","radial","sigmoid"), cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100))
svmt <- tune(svm, factor(Number) ~ .,data=heart_train, ranges = rg)
svmt
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  kernel cost
##  linear 0.01
##
## - best performance: 0.1407407
```

The optimal C in this grid was 10, with a linear kernel.

```
rg2 <- list(kernel=c("linear","polynomial","radial","sigmoid"), cost=seq(5, 15, 0.5))
svmt <- tune(svm, factor(Number) ~ .,data=heart_train, ranges = rg2)
svmt
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   kernel cost
##  sigmoid   11
##
## - best performance: 0.1518519
```

The finer grid identified an optimal C of 5.5.

Testing on the Holdout Set

```
svm.tuned <- svm(factor(Number) ~ ., kernel = "linear", cost = 5.5, data=heart_train)
yhat = predict(svm.tuned, newdata = heart_test[,1:13])
class <- data.frame(yhat = yhat,y = heart_test$Number)
table(class)
```

```
##     y
## yhat  0  1
##    0 11  4
##    1  1 13
```

```
pred <- prediction(as.numeric(yhat==1),as.numeric(heart_test$Number==1))
perf <- performance(pred,"auc")@y.values
perf
```

```
## [[1]]
## [1] 0.8406863
```

Summary Statistics: Total error rate = 17.2% AUC: 0.844

Through both the total error rate and the Area Under the Curve score, we can see that the SVM classifier is doing an okay job correctly identifying test set patients' heart disease status. However, there are more false negatives in the dataset than false positives, and this is not the kind of classifier you want in a disease-detection setting. Below I will compare the results to a simple logistic regression model.

```
log.model <- glm(factor(Number) ~ ., data=heart_train, family = "binomial")
yhat = predict(log.model, newdata= heart_test[,1:13], type = "response")
class <- data.frame(yhat = as.numeric(yhat > 0.5),y = heart_test$Number)
table(class)
```

```
##     y
## yhat  0  1
##    0 11  3
##    1  1 14
```

The logistic regression model has exactly the same distribution of predictions as the SVM model, which makes sense given the explanation of the two methods' relationship to each other detailed in the textbook.