KNN Model

Fiona Huang

2025-04-25

```r
# import datasets
wine <- read.csv("C:/Users/xinya/Downloads/Cornell Classes/STSCI 3740/final project/wine-quality-white-a
```

Fitting a KNN Model

```r
# normalize the data using z-score
normalize <- function(x) {
  return((x - mean(x)) / sd(x))
}

all_columns <- names(wine)
columns_to_normalize <- all_columns[all_columns != "quality" & sapply(wine, is.numeric)]
wine_norm <- wine
wine_norm[columns_to_normalize] <- lapply(wine[columns_to_normalize], normalize)


# change type of wine to white=1, red=2
wine_norm$type <- as.numeric(factor(wine_norm$type))

# split the dataset into train/test
set.seed(1)
index <- sample(1:nrow(wine_norm), size=nrow(wine_norm)*0.7, rep=FALSE)
training <- wine_norm[index, ]
testing <- wine_norm[-index, ]

training_X <- training %>% select(-quality)
testing_X <- testing %>% select(-quality)
```

```r
# try different values of k from 1 to 20
k.values <- 1:20

knn.errors <- sapply(k.values, function(k) {
  knn.pred <- knn(training_X, testing_X, training$quality, k=k)
  mean(knn.pred != testing$quality)
})

print(knn.errors)
```

```
##  [1] 0.3989744 0.4794872 0.4707692 0.4574359 0.4482051 0.4456410 0.4507692
##  [8] 0.4528205 0.4451282 0.4528205 0.4364103 0.4420513 0.4317949 0.4379487
## [15] 0.4358974 0.4358974 0.4461538 0.4415385 0.4389744 0.4405128
```

The value of k that seems to perform the best on this data is k=1.

Choose the optimal k-value using Cross Validation

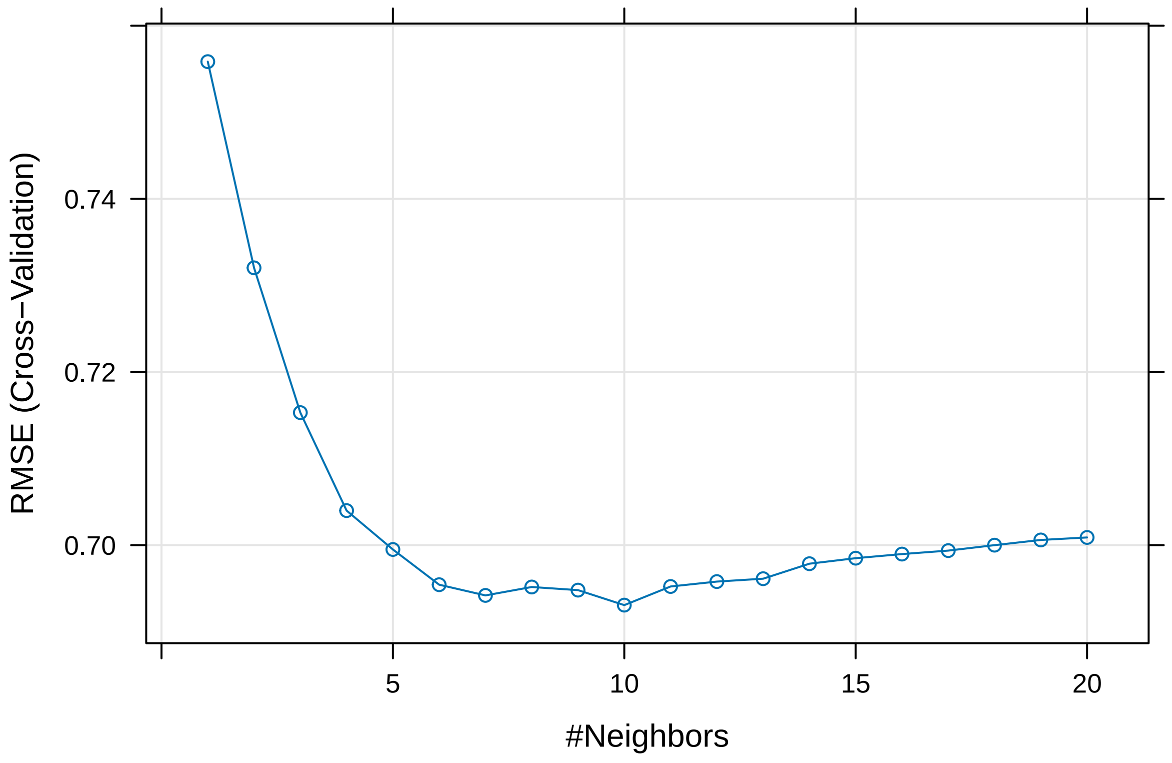
```r
set.seed(1)
# 10-fold cross validation
control <- trainControl(method = "cv", number = 10)

knn_cv <- train(
  quality ~ .,
  data = wine_norm,
  method = "knn",
  trControl = control,
  tuneGrid = expand.grid(k = 1:20)
)

knn_cv
```

```
## k-Nearest Neighbors
##
## 6497 samples
##   12 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5846, 5847, 5847, 5847, 5848, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    1  0.7558513  0.3795189  0.4239693
##    2  0.7320306  0.3612061  0.4939238
##    3  0.7153067  0.3654421  0.5125768
##    4  0.7039897  0.3718375  0.5170525
##    5  0.6994976  0.3727748  0.5246780
##    6  0.6954288  0.3756623  0.5260905
##    7  0.6941926  0.3755005  0.5286038
##    8  0.6951634  0.3725416  0.5314023
##    9  0.6948018  0.3720174  0.5344289
##   10  0.6930709  0.3743660  0.5342596
##   11  0.6952228  0.3699285  0.5365115
##   12  0.6957883  0.3683444  0.5382253
##   13  0.6961228  0.3676329  0.5401778
##   14  0.6978546  0.3643808  0.5417053
##   15  0.6984920  0.3632550  0.5426139
##   16  0.6989681  0.3623560  0.5431027
##   17  0.6993684  0.3614560  0.5446430
##   18  0.6999960  0.3602518  0.5456382
##   19  0.7005973  0.3591466  0.5467203
##   20  0.7008890  0.3587130  0.5473792
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 10.
```

```
plot(knn_cv)
```



Fitting the best model

```
model <- knn(training_X, testing_X, training$quality, k=10)

confusion_matrix <- table(coPredicted = model, Actual = testing$quality)
print(confusion_matrix)
```

```
##             Actual
## coPredicted   3   4   5   6   7   8   9
##           3   0   0   0   0   0   0   0
##           4   0   2   5   0   0   0   0
##           5   4  32 392 186  16   4   0
##           6   4  33 219 547 155  22   1
##           7   0   0  19 123 144  31   1
##           8   0   0   0   5   2   2   1
##           9   0   0   0   0   0   0   0
```

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
accuracy
```

```
## [1] 0.5574359
```

Apply PCA to reduce diensionality

```
# apply PCA before fitting KNN
new_wine <- wine
new_wine$type <- as.numeric(factor(new_wine$type))

target <- new_wine$quality
predictors <- new_wine %>% select(-quality)

# standardize data
scaled_data <- scale(predictors)

# Perform PCA
pca_result <- prcomp(scaled_data, center = TRUE, scale. = TRUE)
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6    PC7
## Standard deviation     1.9518 1.5902 1.2496 0.9853 0.85077 0.78329 0.7324
## Proportion of Variance 0.3175 0.2107 0.1301 0.0809 0.06032 0.05113 0.0447
## Cumulative Proportion  0.3175 0.5282 0.6583 0.7392 0.79952 0.85065 0.8953
##                            PC8     PC9   PC10    PC11    PC12
## Standard deviation     0.70921 0.59368 0.5068 0.34552 0.15539
## Proportion of Variance 0.04192 0.02937 0.0214 0.00995 0.00201
## Cumulative Proportion  0.93727 0.96664 0.9880 0.99799 1.00000
```

Decided to choose the first eight PCs

```
# reduce dimensionality
# choose the first eight PCs
pca_data <- pca_result$x[, 1:8]
```

Fit KNN Model on PCA-reduced data

```
# apply KNN on PCA-reduced data
set.seed(1)
index_2 <- sample(1:nrow(pca_data), size=nrow(pca_data)*0.7, rep=FALSE)
training_data <- pca_data[index_2, ]
testing_data <- pca_data[-index_2, ]

training_targt <- target[index_2]
testing_target <- target[-index_2]
```

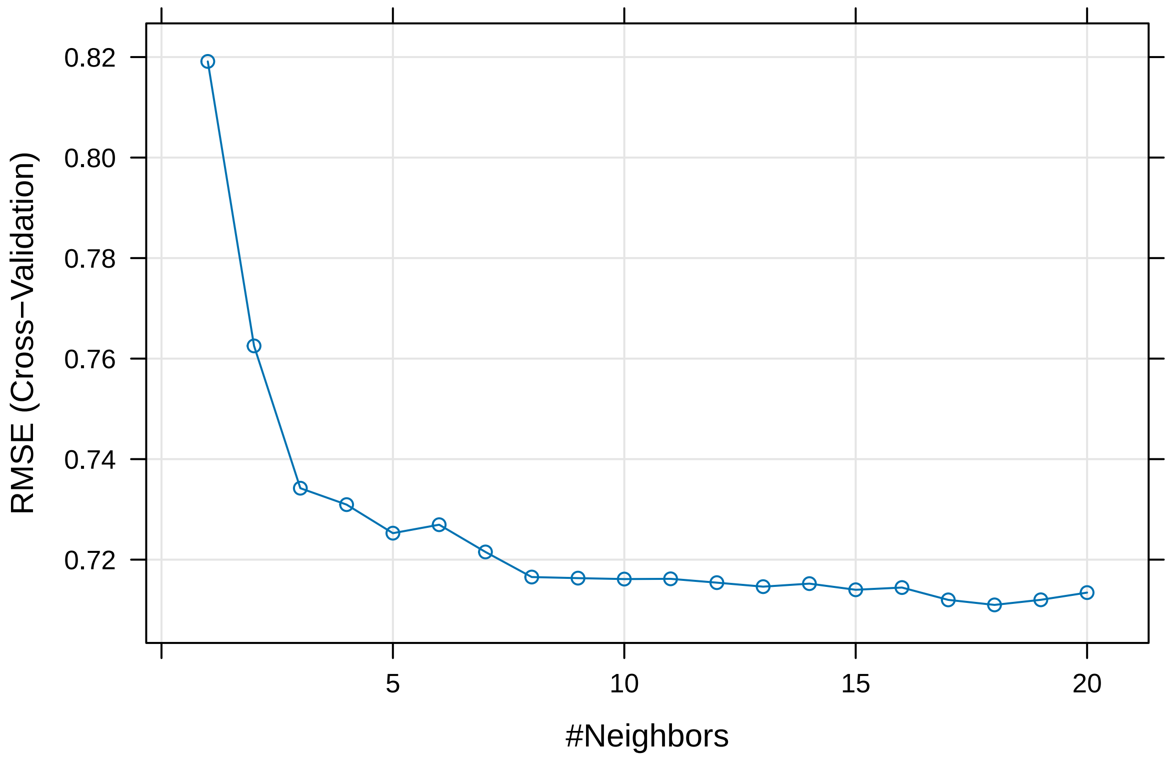
```r
# 10-fold cross validation to find the best k
control_new <- trainControl(method = "cv", number = 10)

knn_cv_2 <- train(
  training_data,
  training_targt,
  method = "knn",
  trControl = control_new,
  tuneGrid = expand.grid(k = 1:20)
)

knn_cv_2
```

```
## k-Nearest Neighbors
##
## 4547 samples
##    8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4091, 4091, 4093, 4092, 4092, 4091, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    1  0.8191349  0.2974479  0.4807663
##    2  0.7625294  0.3141066  0.5264729
##    3  0.7342127  0.3306869  0.5353884
##    4  0.7309495  0.3219460  0.5462665
##    5  0.7252672  0.3256003  0.5506567
##    6  0.7269512  0.3199774  0.5561480
##    7  0.7215278  0.3253284  0.5557852
##    8  0.7165359  0.3322161  0.5540389
##    9  0.7163207  0.3311995  0.5561248
##   10  0.7161339  0.3301913  0.5570248
##   11  0.7161817  0.3296976  0.5592316
##   12  0.7154242  0.3309020  0.5601571
##   13  0.7146273  0.3320483  0.5608825
##   14  0.7152282  0.3306918  0.5617482
##   15  0.7140067  0.3327959  0.5607729
##   16  0.7144465  0.3320431  0.5621654
##   17  0.7119925  0.3367552  0.5608877
##   18  0.7109959  0.3386110  0.5605507
##   19  0.7120115  0.3367728  0.5614468
##   20  0.7134455  0.3339659  0.5633644
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 18.
```

```r
plot(knn_cv_2)
```



```
knn_model <- knn(train = training_data, test = testing_data, cl = training_targt, k = 17)

confusion_matrix <- table(Predicted = knn_model, Actual = testing_target)
print(confusion_matrix)
```

```
##          Actual
## Predicted   3   4   5   6   7   8   9
##         3   0   0   0   0   0   0   0
##         4   0   1   2   1   0   0   0
##         5   5  34 367 200  14   3   0
##         6   3  30 247 566 176  25   2
##         7   0   2  19  94 126  31   1
##         8   0   0   0   0   1   0   0
##         9   0   0   0   0   0   0   0
```

```
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
accuracy
```

```
## [1] 0.5435897
```