Cross-Validation

Jaime Davila

3/14/2022

Introduction

Today we will be reusing our last dataset, that is the `1`, `2`, and `7` dataset.

```
digits <- c("1","2","7")
train.127.tbl <- read_csv("~/Mscs 341 S22/Class/Data/train.127.csv") %>%
  mutate(y=factor(y, levels=digits))
test.127.tbl <- read_csv("~/Mscs 341 S22/Class/Data/test.127.csv") %>%
  mutate(y=factor(y, levels=digits))
```

And let's use a KNN model, but this time we will be using the syntax from `tidymodels` and let's encapsulate our model building using a function `build_knn`

```
library(tidymodels)
library(kknn)
## devtools::install_github("KlausVigo/kknn")
tidymodels_prefer()

build_knn <- function (train.tbl, kVal) {
  knn.model <- nearest_neighbor(neighbors = kVal) %>%
    set_engine("kknn") %>%
    set_mode("classification")

  recipe <- recipe(y ~ x_1 + x_2, data=train.tbl)

  knn.wflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn.model)

  knn.fit <- fit(knn.wflow, train.tbl)
}

knn.model <- build_knn(train.127.tbl, 5)
```

We are interested in plotting the boundary of our classifier, so let's create a function that would help us do that:

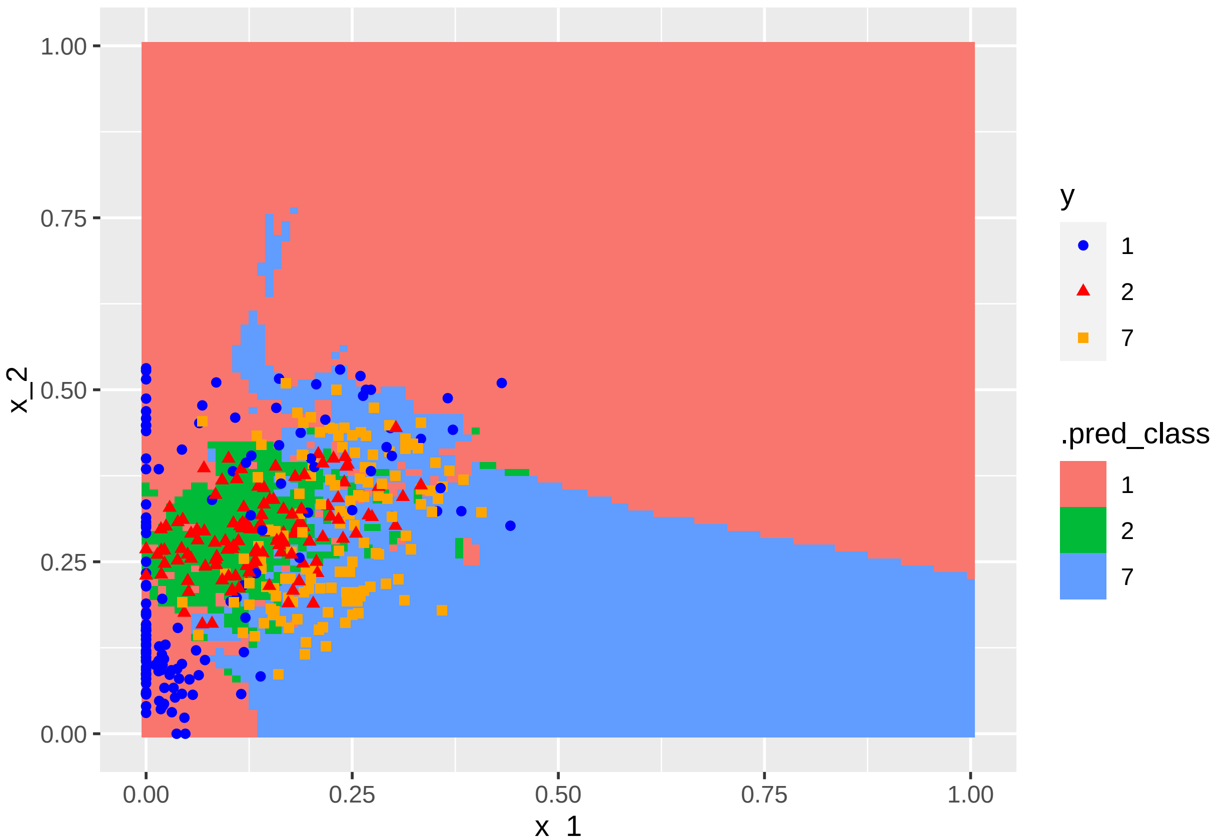
```
plot_boundary <- function(fit, test.tbl, delta){
  grid.tbl <- expand_grid(x_1=seq(0,1, by=delta),
                          x_2=seq(0,1, by=delta))

  augment(fit, grid.tbl)%>%
    ggplot() +
```

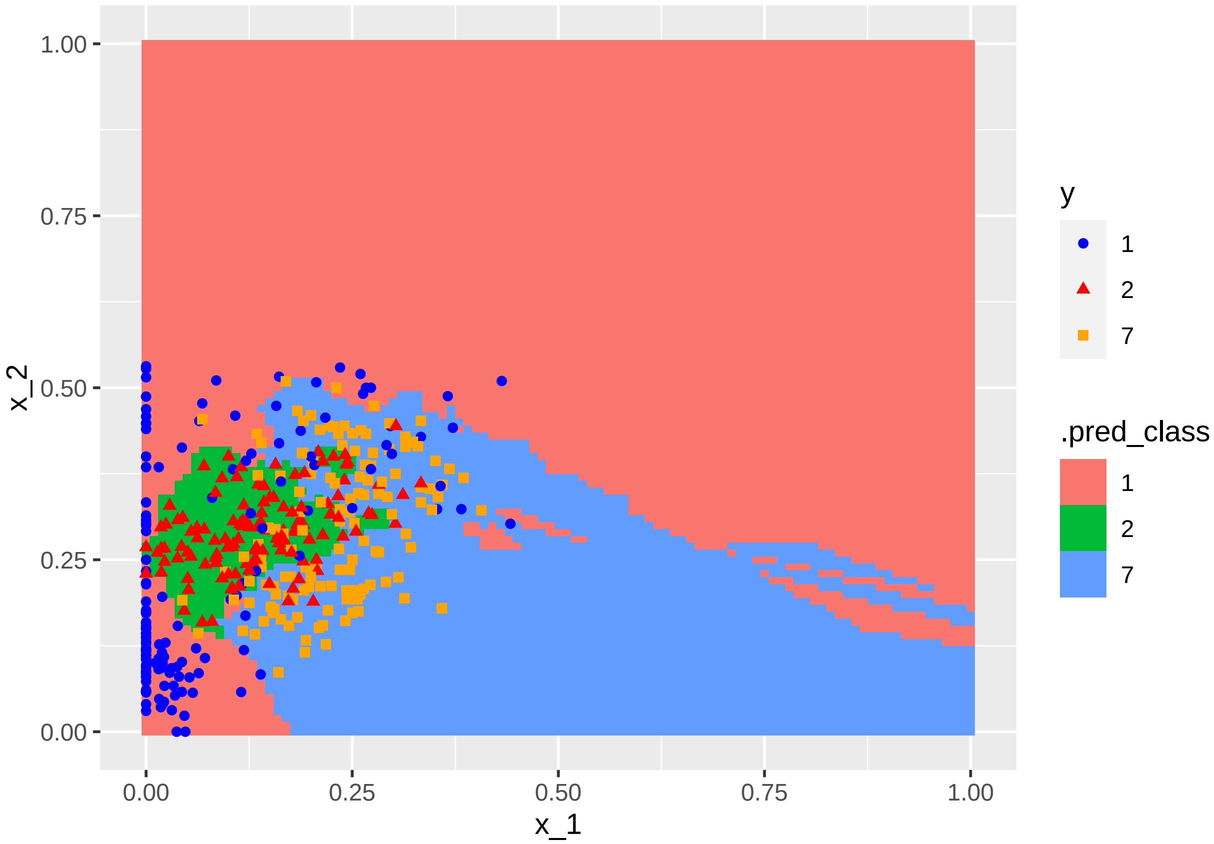
```
    geom_raster(aes(x_1, x_2, fill = .pred_class)) +
    geom_point(data=test.tbl, aes(x=x_1, y=x_2, color=y, shape=y))+
    scale_color_manual(values=c("blue","red","orange"))
}
```

And let's put things together and try different values of k

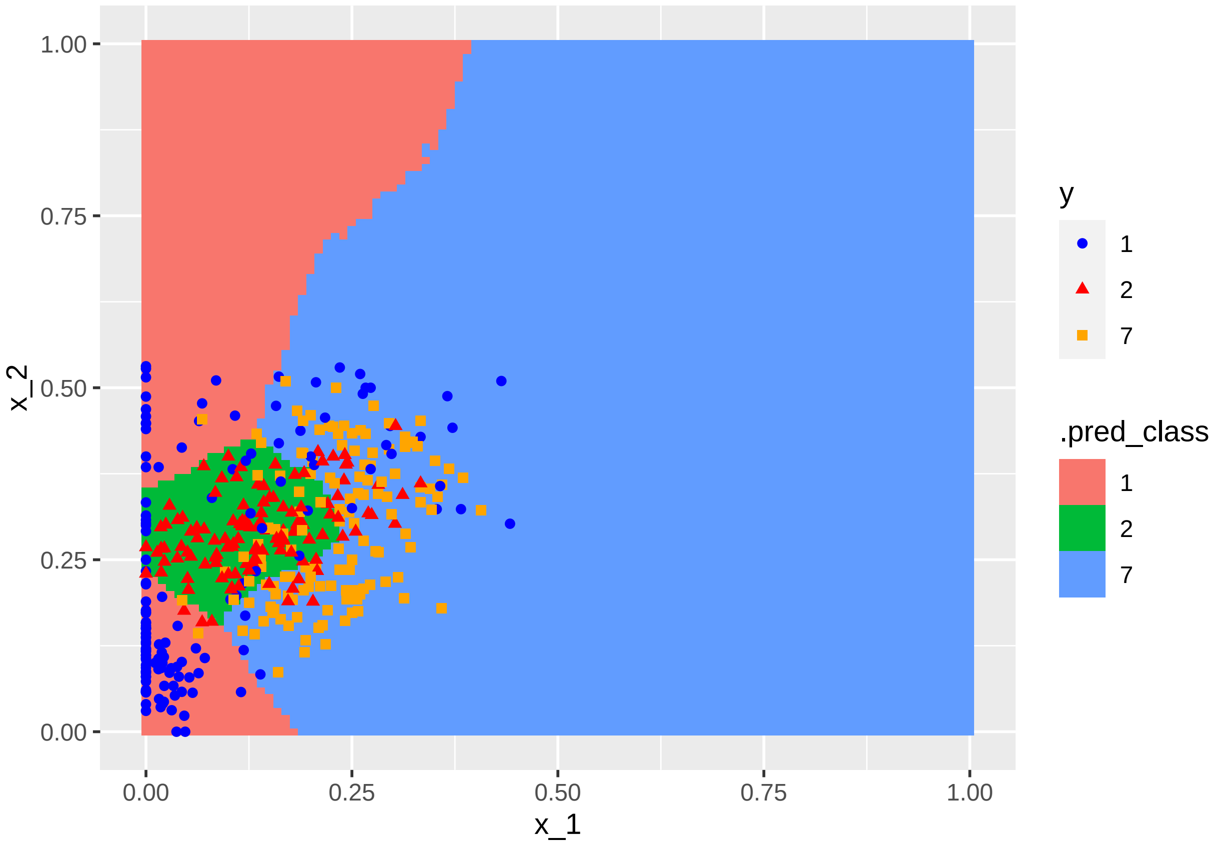
```
plot_boundary(knn.model, test.127.tbl, 0.01)
```




```
plot_boundary(build_knn(test.127.tbl, 10),
              test.127.tbl, 0.01)
```




```
plot_boundary(build_knn(test.127.tbl, 50),
              test.127.tbl, 0.01)
```



We are interested in finding the best parameter of k, but notice to find this parameter we are peeking repeatedly at the testing dataset which might result in some overfitting. Is there a better approach that we can use to do that?

Cross-validation

The answer to our question is to use k-fold cross-validation which allows us to reuse our training dataset without having to look at our testing dataset. More details on how this approach works in https://rafalab.github.io/dsbook/cross-validation.html#k-fold-cross-validation

K-fold validation in `tidymodels`

The details of how K-fold cross validation can be implemented in `tidymodels` are available from:

https://emilhvitfeldt.github.io/ISLR-tidymodels-labs/resampling-methods.html#k-fold-cross-validation

These steps can be summarized as follows:

- Create a `parsnip` workflow where the models parameters are **marked** for tuning
- Create a `vfold_cv` `rsample` object with the cross-validation resamples
- Create a `tibble` denoting the parameters denoted to be explored
- Use `tune_grid()` using the 3 objects defined before.

The first step is to create a knn model/workflow, making sure to use the function `tune()` as the `neighbors` option inside the `nearest_neighbor()` function

```
knn.model <- nearest_neighbor(neighbors = tune()) %>%
    set_engine("kknn") %>%
    set_mode("classification")

recipe <- recipe(y ~ x_1 + x_2, data=train.127.tbl)

knn.wf <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn.model)
```

1. Look at the documentation of `vfold_cv` and use it to create a 10-fold cross-validation dataset called `digits.folds` using your training dataset. What is the type of `digits.folds`? Display the training/testing dataset with id `Fold02` by using the functions `testing()` and `training()`.

```
set.seed(12345)
digits.folds <- vfold_cv(train.127.tbl, v = 10)
digits.folds
```

```
## #  10-fold cross-validation
## # A tibble: 10 x 2
##    splits             id
##    <list>             <chr>
##  1 <split [1440/161]> Fold01
##  2 <split [1441/160]> Fold02
##  3 <split [1441/160]> Fold03
##  4 <split [1441/160]> Fold04
##  5 <split [1441/160]> Fold05
##  6 <split [1441/160]> Fold06
##  7 <split [1441/160]> Fold07
##  8 <split [1441/160]> Fold08
##  9 <split [1441/160]> Fold09
## 10 <split [1441/160]> Fold10
```

```
training(digits.folds$splits[[2]])
```

```
## # A tibble: 1,441 x 3
##    y        x_1    x_2
##    <fct>  <dbl>  <dbl>
##  1 1     0      0.556
##  2 7     0.213  0.213
##  3 1     0.0238 0.0714
##  4 7     0.152  0.232
##  5 7     0.216  0.235
##  6 7     0.0485 0.136
##  7 2     0.197  0.370
##  8 7     0.323  0.354
##  9 2     0.165  0.266
## 10 7     0.274  0.218
## # ... with 1,431 more rows
```

```
testing(digits.folds$splits[[2]])
```

```
## # A tibble: 160 x 3
##    y        x_1    x_2
##    <fct>  <dbl>  <dbl>
##  1 1     0.204  0.429
```

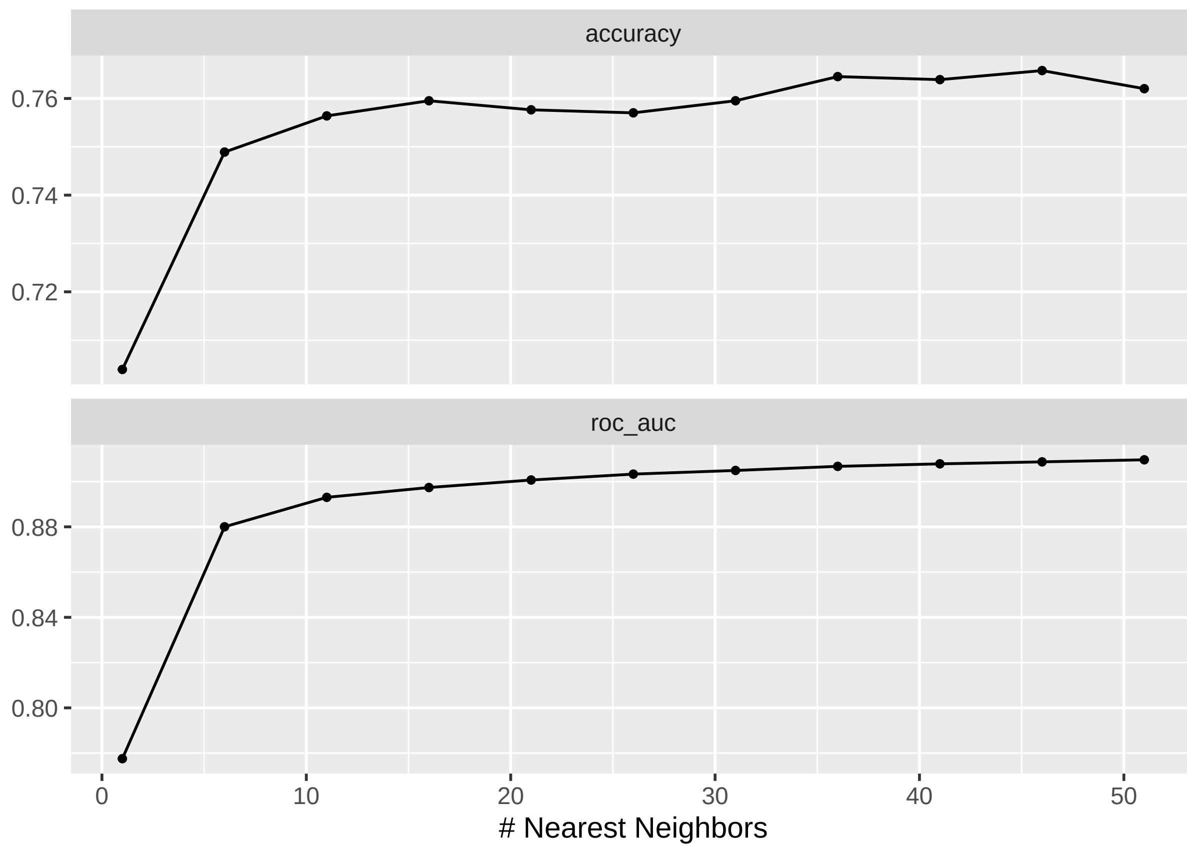
```
##    2 2       0.225   0.296
##    3 7       0.22    0.46
##    4 7       0.270   0.255
##    5 7       0.208   0.286
##    6 2       0.148   0.393
##    7 1       0       0.0909
##    8 7       0.182   0.208
##    9 1       0       0.419
##   10 2       0.0241  0.229
## # ... with 150 more rows
```

2. Create a tibble `neigbors.tbl` with a column called `neighbors` with values `1`, `6`, `11`, . . . , `51`. Create the same tibble using the function `grid_regular()`.

```
neighbors.tbl <-  tibble(neighbors=seq(1,51, by=5))
neighbors.grid.tbl <- grid_regular(neighbors(range = c(1, 51)),
                                   levels = 11)
```

3. Use the function `tune_grid()` to optimize the neigbors parameter from your knn model. Plot the results of your optimization using `autoplot()`. How do you interpret this plot?

```
tune.results <- tune_grid(object = knn.wf,
                          resamples = digits.folds,
                          grid = neighbors.tbl)
autoplot(tune.results)
```



We can finalize our model by selecting the best K which maximizes accuracy and fit our model using the training dataset.

```r
show_best(tune.results, metric = "accuracy")
```

```
## # A tibble: 5 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>
## 1        46 accuracy multiclass 0.766    10 0.00859 Preprocessor1_Model10
## 2        36 accuracy multiclass 0.765    10 0.00934 Preprocessor1_Model08
## 3        41 accuracy multiclass 0.764    10 0.00875 Preprocessor1_Model09
## 4        51 accuracy multiclass 0.762    10 0.00859 Preprocessor1_Model11
## 5        31 accuracy multiclass 0.760    10 0.00925 Preprocessor1_Model07
```

```r
best.neighbor <- select_best(tune.results, metric = "accuracy")
knn.final.wf <- finalize_workflow(knn.wf, best.neighbor)
knn.final.fit <- fit(knn.final.wf, train.127.tbl)
```

4. Calculate the confusion matrix of `knn.final.fit` on the testing dataset. Calculate the accuracy of `knn.final.fit` on the testing dataset and compare it to the values you obtained using cross validation. Plot the boundary of the model for the optimal k

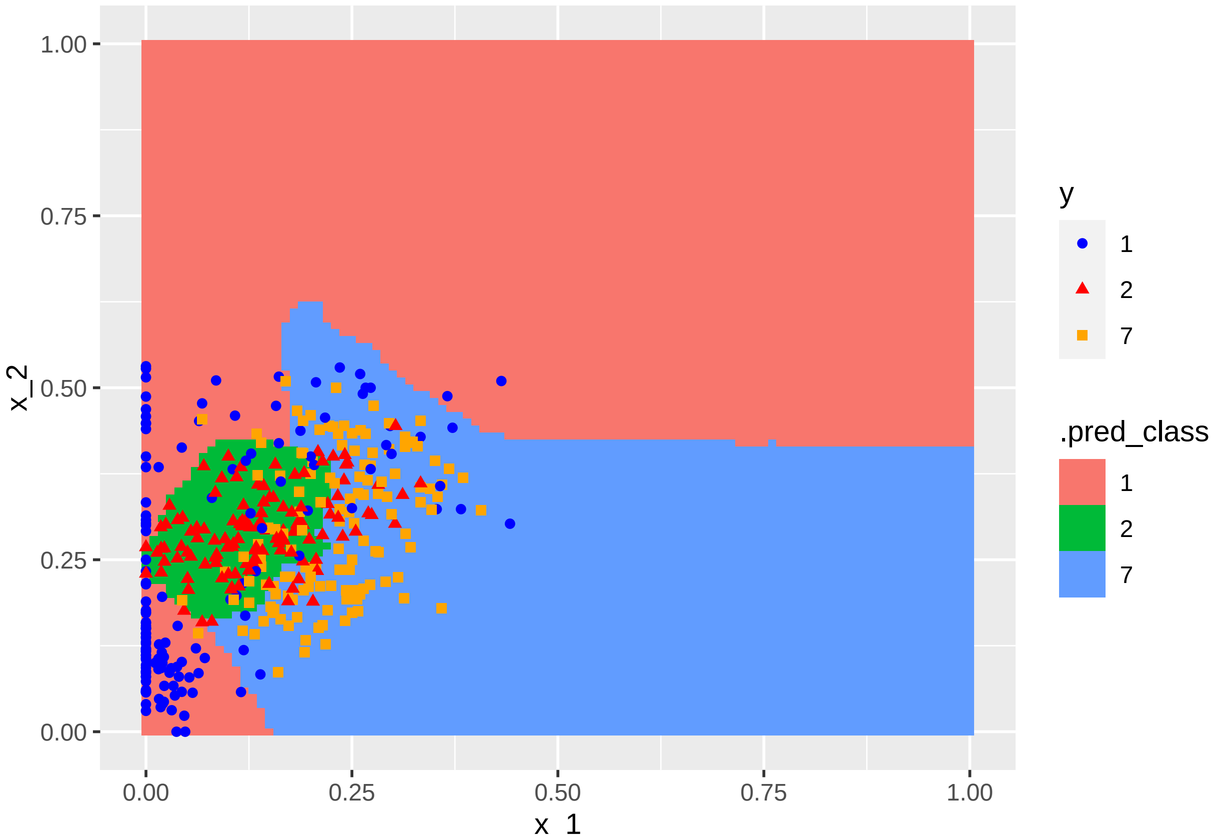
```r
augment(knn.final.fit, test.127.tbl) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction   1   2   7
##          1 102   4   4
##          2  17  91  26
##          7  23  28 104
```

```r
augment(knn.final.fit, test.127.tbl) %>%
  accuracy(truth = y, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.744
```

```r
plot_boundary(knn.final.fit, test.127.tbl, 0.01)
```



Back to the future

Remember the Minneapolis police incident dataset?

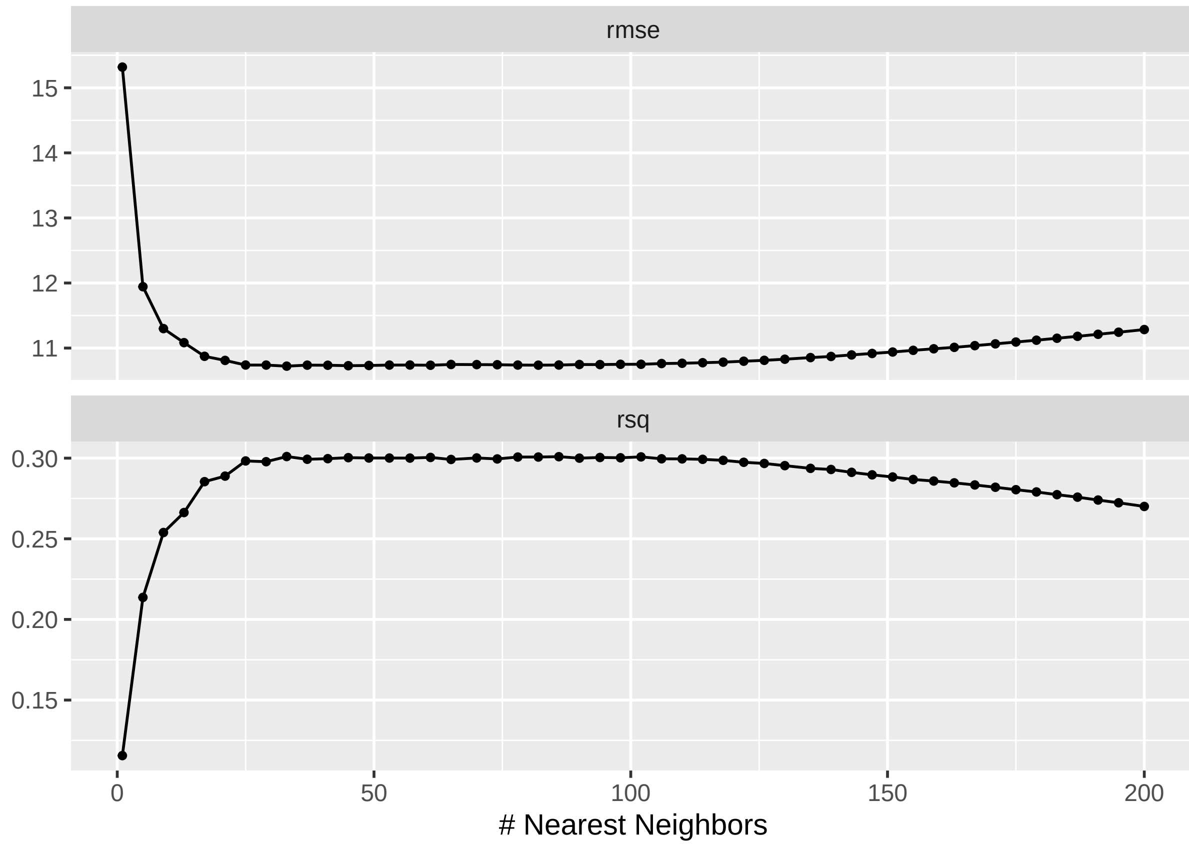
```
mn.police.tbl <- read_csv("~/Mscs 341 S22/Class/Data/police_incidents.mn.csv")
```

5. Using the `tidymodels` library construct a KNN model for predicting `tot` as a function of `week`. Remember to create a training/testing dataset with equal number of observations. Find the optimal k in your KNN model by using cross validation and the function `select_by_one_std_err`.

```
#Training/Testing dataset
set.seed(12345)
mn.split <- initial_split(mn.police.tbl, prop=0.5)
mn.train.tbl <- training(mn.split)
mn.test.tbl <- testing(mn.split)

# Model specification
knn.model <- nearest_neighbor(neighbors = tune()) %>%
    set_engine("kknn") %>%
    set_mode("regression")
recipe <- recipe(tot ~ week, data=mn.train.tbl)
knn.wf <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn.model)
```

```r
# Neighbors optimization using CV
mn.police.folds <- vfold_cv(mn.train.tbl, v = 10)
neighbors.grid.tbl <- grid_regular(neighbors(range = c(1, 200)),
                                   levels = 50)
tune.results <- tune_grid(object = knn.wf,
                          resamples = mn.police.folds,
                          grid = neighbors.grid.tbl)
autoplot(tune.results)
```




```r
show_best(tune.results, metric = "rmse")
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
## 1        33 rmse    standard    10.7    10   0.264 Preprocessor1_Model09
## 2        45 rmse    standard    10.7    10   0.287 Preprocessor1_Model12
## 3        49 rmse    standard    10.7    10   0.300 Preprocessor1_Model13
## 4        61 rmse    standard    10.7    10   0.313 Preprocessor1_Model16
## 5        41 rmse    standard    10.7    10   0.280 Preprocessor1_Model11
```

```r
best.neighbor <- select_by_one_std_err(tune.results,
                                       neighbors, metric = "rmse")
knn.final.wf <- finalize_workflow(knn.wf, best.neighbor)
knn.final.fit <- fit(knn.final.wf, mn.train.tbl)

# Evaluation of model on the testing dataset
augment(knn.final.fit, mn.test.tbl) %>%
```

```r
  rmse(truth = tot, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        10.4
```