More Classification

Stat 220

Bastola

February 28 2022

KNN (K- Nearest Neighbor)

- Supervised machine learning algorithm i.e., it requires labeled data for training
- Need to tell the algorithm the exact number of neighbors (K) we want to consider

Training and Testing

Training: Fitting a model with certain hyper-parameters on a particular subset of the dataset

Testing: Test the model on a different subset of the dataset to get an estimate of a final, unbiased assessment of the model's performance

Workflows

A machine learning workflow (the “black box”) containing model specification and preprocessing recipe/formula

Basic Structure

```
model_wf <- workflow() %>%
  add_recipe(rec) %>% # add_formula(formula) if no recipe
  add_model(mod_spec)

model_wf %>%
  update_recipe(rec_new)

model_wf %>%
  update_formula(formula_new)

model_wf %>%
  update_model(model_spec_new)
```

Creating a workflow: Splitting the raw data

```r
set.seed(123) # set seed for replicability

fire_split <- initial_split(fire_raw, prop = 0.75)

# Create training data
fire_train <- fire_split %>%
                    training()


# Create testing data
fire_test <- fire_split %>%
                    testing()
```

Make a recipe

```
fire_recipe <- recipe(classes ~ ., data = fire_raw) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors()) %>%
  prep()
```

Specify the model

```
fire_knn_spec <- nearest_neighbor(mode = "classification",
                                  engine = "kknn",
                                  weight_func = "rectangular",
                                  neighbors = 5)
```

Define the workflow object

```
fire_workflow <- workflow() %>%
  add_recipe(fire_recipe) %>%
  add_model(fire_knn_spec)
```

Fit the model

```
fire_fit <- fit(fire_workflow, data = fire_train)
```

```
══ Workflow [trained] ══════════════════════════════════════════
Preprocessor: Recipe
Model: nearest_neighbor()

── Preprocessor ────────────────────────────────────────────────
2 Recipe Steps

• step_scale()
• step_center()

── Model ───────────────────────────────────────────────────────

Call:
kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(5,     data,

Type of response variable: nominal
Minimal misclassification: 0.03296703
Best kernel: rectangular
Best k: 5
```

Evaluate the model on test dataset

```r
test_features <- fire_test %>% select(temperature, isi) %>% data.frame()
fire_pred <- predict(fire_fit, test_features, type = "raw")
fire_results <- fire_test %>%
  select(classes) %>%
  bind_cols(predicted = fire_pred)
```

Compare the known labels and predicted labels

```
fire_results
# A tibble: 61 × 2
   classes  predicted
   <chr>    <fct>
 1 not fire not fire
 2 not fire not fire
 3 fire     fire
 4 not fire not fire
 5 not fire not fire
 6 not fire not fire
 7 fire     fire
 8 fire     fire
 9 not fire not fire
10 not fire not fire
# … with 51 more rows
```

Your Turn 1

05:00

Please clone the repository on classification evaluation to your local folder.

a. Set aside 20% of the cases using the following code.

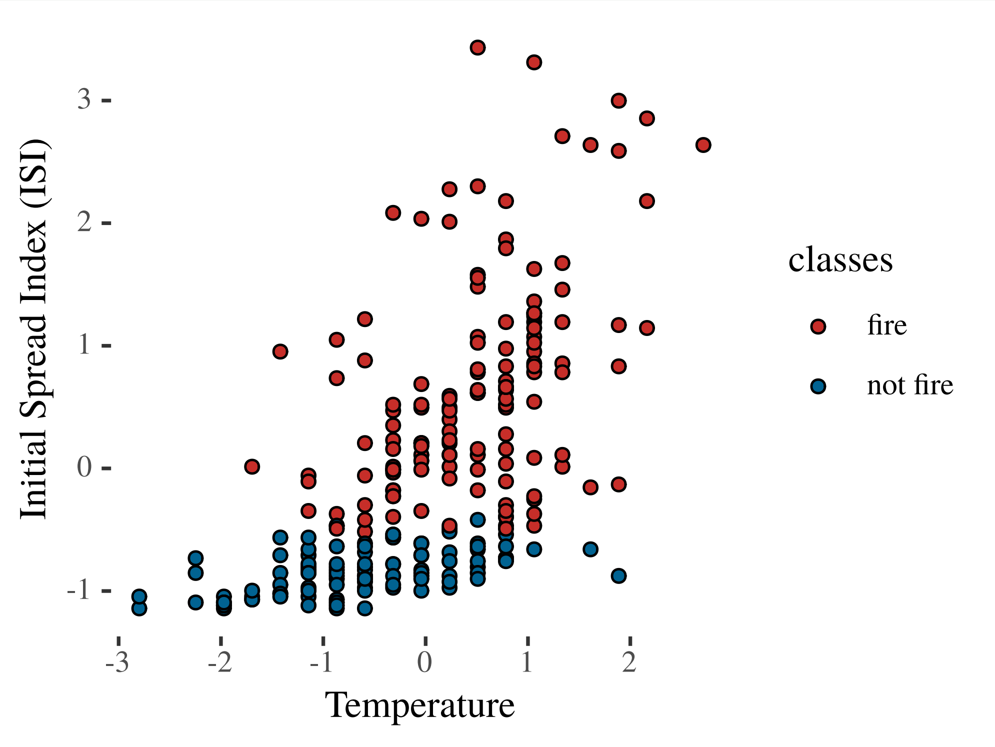
```
set.seed(1234)
db_split <- initial_split(db, prop = 0.80)

# Create training data
db_train <- db_split %>%
                    training()
# Create testing data
db_test <- db_split %>%
                    testing()
```

Using this split, complete the set of questionnaires.

How do we choose the number of neighbors in a principled way?

Let's start with the scatterplot



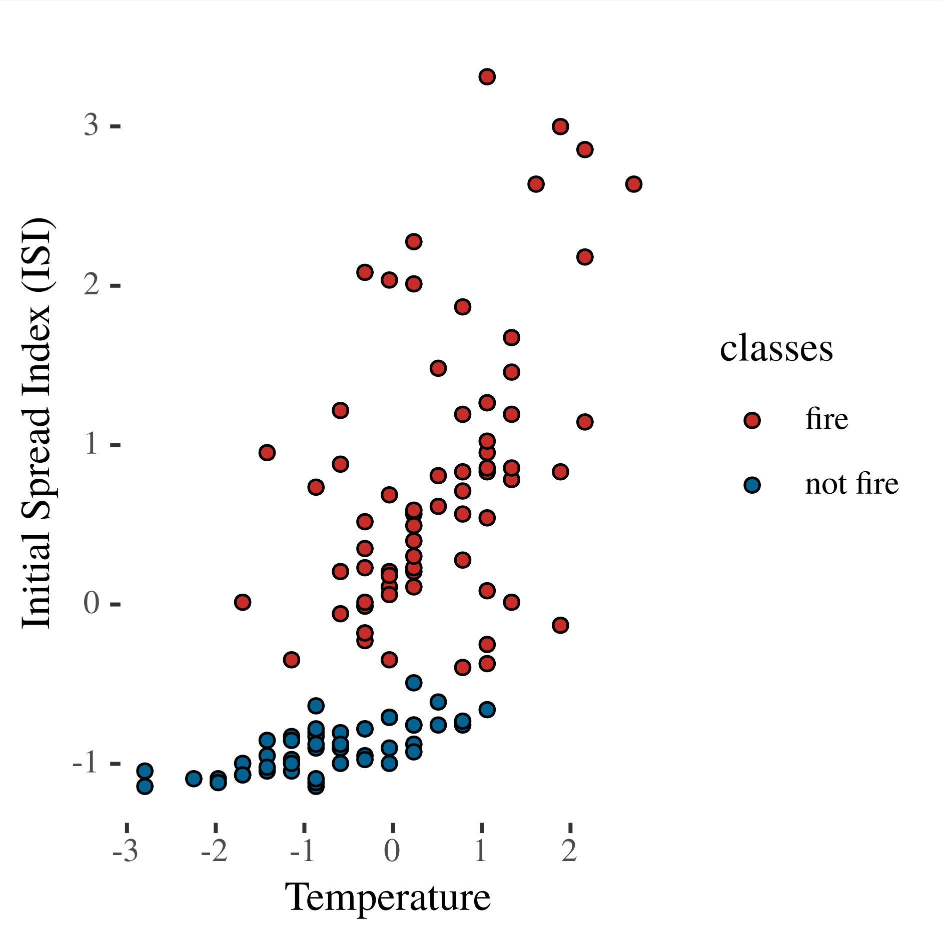
- We normally don't have a clear separation between classes and usually have more than 2 features.
- Eyeballing on a plot to discern the classes is not very helpful in the practical sense

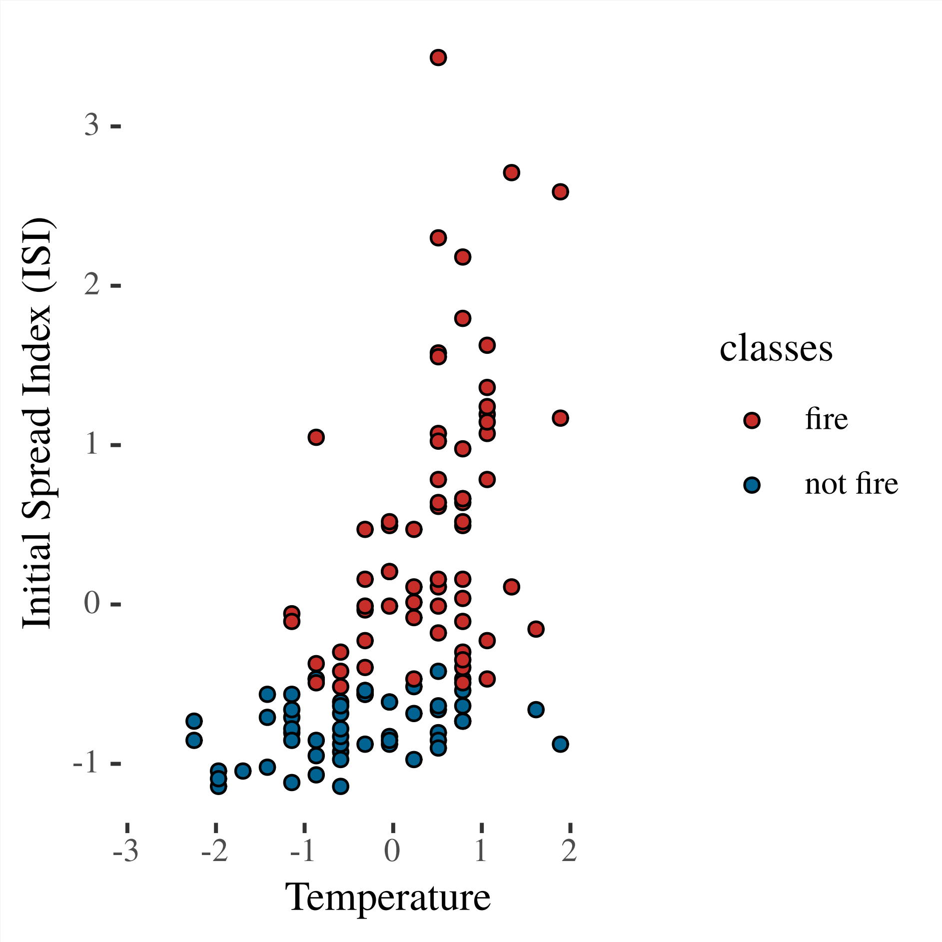
Evaluating accuracy

Idea: We want to evaluate classifiers based on some accuracy metrics.

- Randomly split data set into two pieces: **training set** and **test set**
- Train (i.e. fit) KNN on the training set
- Make predictions on the test set
- See how good those predictions are

Train (left) and test (right) dataset (50-50)





Training 1-NN

Evaluating performance

Confusion matrix

Confusion matrix: tabulation of true (i.e. reference) and predicted class labels

Performance metrics

```
conf_mat(fire_results, truth = classes,
         estimate = predicted)
          Truth
Prediction fire not fire
  fire       61        2
  not fire    6       53
```

Common metrics include:

- accuracy
- sensitivity
- specificity
- positive predictive value (PPV)
- Kappa
- Matthews correlation coefficient (MCC)

Accuracy

Proportion of correctly classified cases

$$\text{Accuracy} = \frac{\text{true positives} + \text{true negatives}}{n}$$

```
          Truth
Prediction fire not fire
  fire       61        2
  not fire    6       53
```

```
accuracy(fire_results, truth = classes,
         estimate = predicted)
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.934
```

Sensitivity

Proportion of positive cases that are predicted to be positive

$$\text{Sensitivity} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Also called... true positive rate or recall

```
           Truth
Prediction fire not fire
  fire       61        2
  not fire    6       53
```

```
sens(fire_results, truth = classes,
         estimate = predicted)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.910
```

Specificity

Proportion of negative cases that are predicted to be negative

$$\text{Specificity} = \frac{\text{true negatives}}{\text{false positives} + \text{true negatives}}$$

Also called... true negative rate

```
            Truth
Prediction fire not fire
  fire       61        2
  not fire    6       53
```

```
spec(fire_results, truth = classes,
         estimate = predicted)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.964
```

Kappa

Cohen Kappa gives information on how much better a model over the random classifier. Kappa can range from −1 to +1

The value < 0 means no agreement while 1.0 shows perfect agreement.

```
kap(fire_results, truth = classes,
    estimate = predicted)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 kap     binary         0.868
```

Positive predictive value (PPV)

Proportion of cases that are predicted to be positives that are truly positives

$$\text{PPV} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Also called... precision

```
           Truth
Prediction fire not fire
  fire       61        2
  not fire    6       53
```

```
ppv(fire_results, truth = classes,
             estimate = predicted)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 ppv     binary         0.968
```

Matthews Correlation Coefficient (MCC)

The Matthews correlation coefficient (MCC) is used as a measure of the quality of a binary classifier. The value ranges from −1 and +1.

- **MCC: -1** indicates total disagreement
- **MCC: 0** indicate no agreement
- **MCC: +1** indicates total agreement

```
mcc(fire_results, truth = classes,
         estimate = predicted)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mcc     binary         0.870
```

Your Turn 2

05:00

Here is the confusion matrix fora hypothetical two-class 7-NN penguin classifier

```
             Truth
Prediction fire not fire
  fire       51        2
  not fire    1       44
```

Calculate the accuracy, sensitivity, specificity, and PPV of this classifier.

So many metrics!!

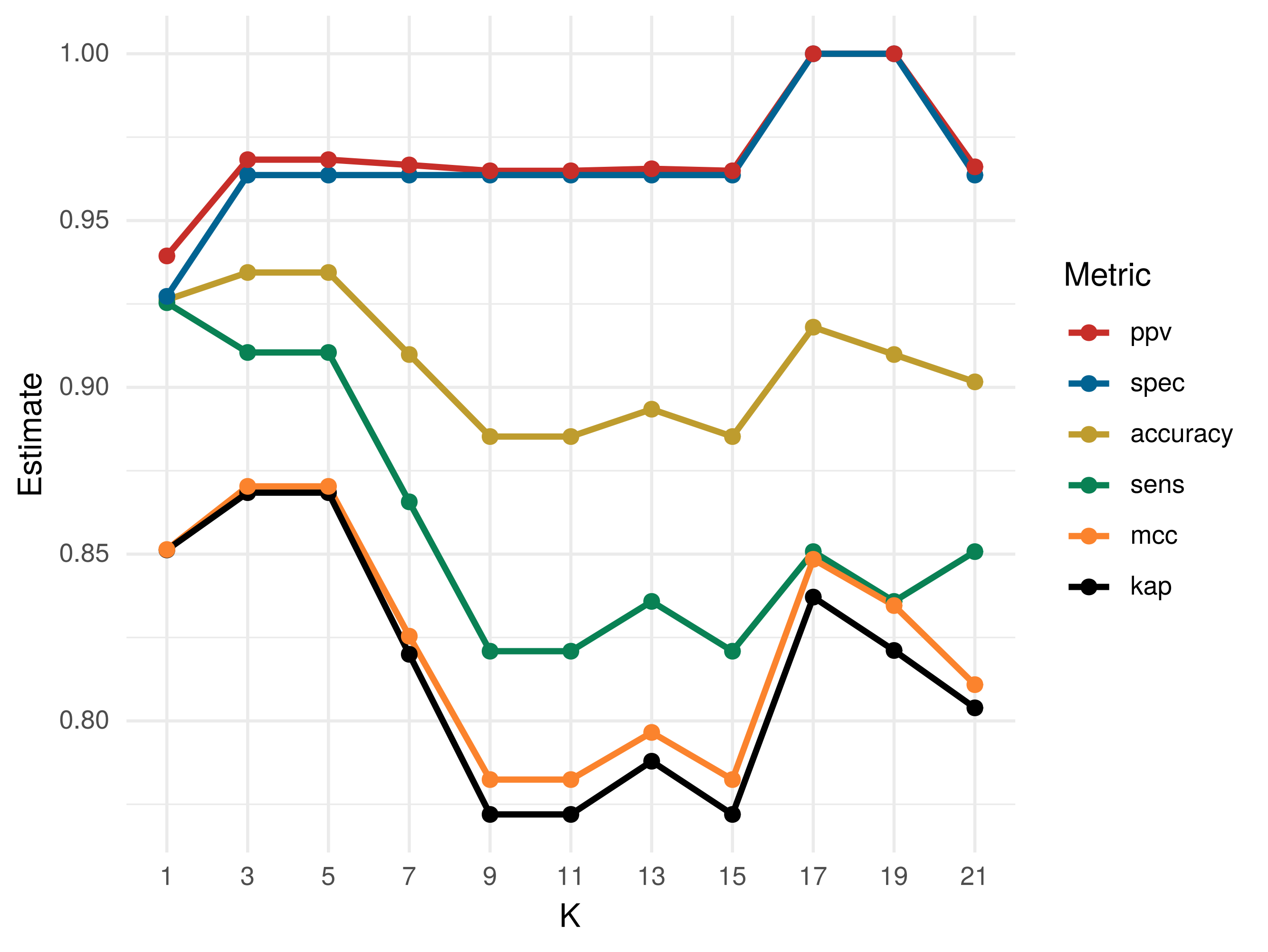
```
custom_metrics <- metric_set(accuracy, sens, spec, ppv, kap, mcc)

metrics <- custom_metrics(fire_results,
               truth = classes,
               estimate = predicted)
metrics <- metrics %>% select(-.estimator)
```

Tabulate the metrics

```
metrics
# A tibble: 6 × 2
  .metric  .estimate
  <chr>        <dbl>
1 accuracy     0.934
2 sens         0.910
3 spec         0.964
4 ppv          0.968
5 kap          0.868
6 mcc          0.870
```

Plot them over the hyperparamter, K



Tuning

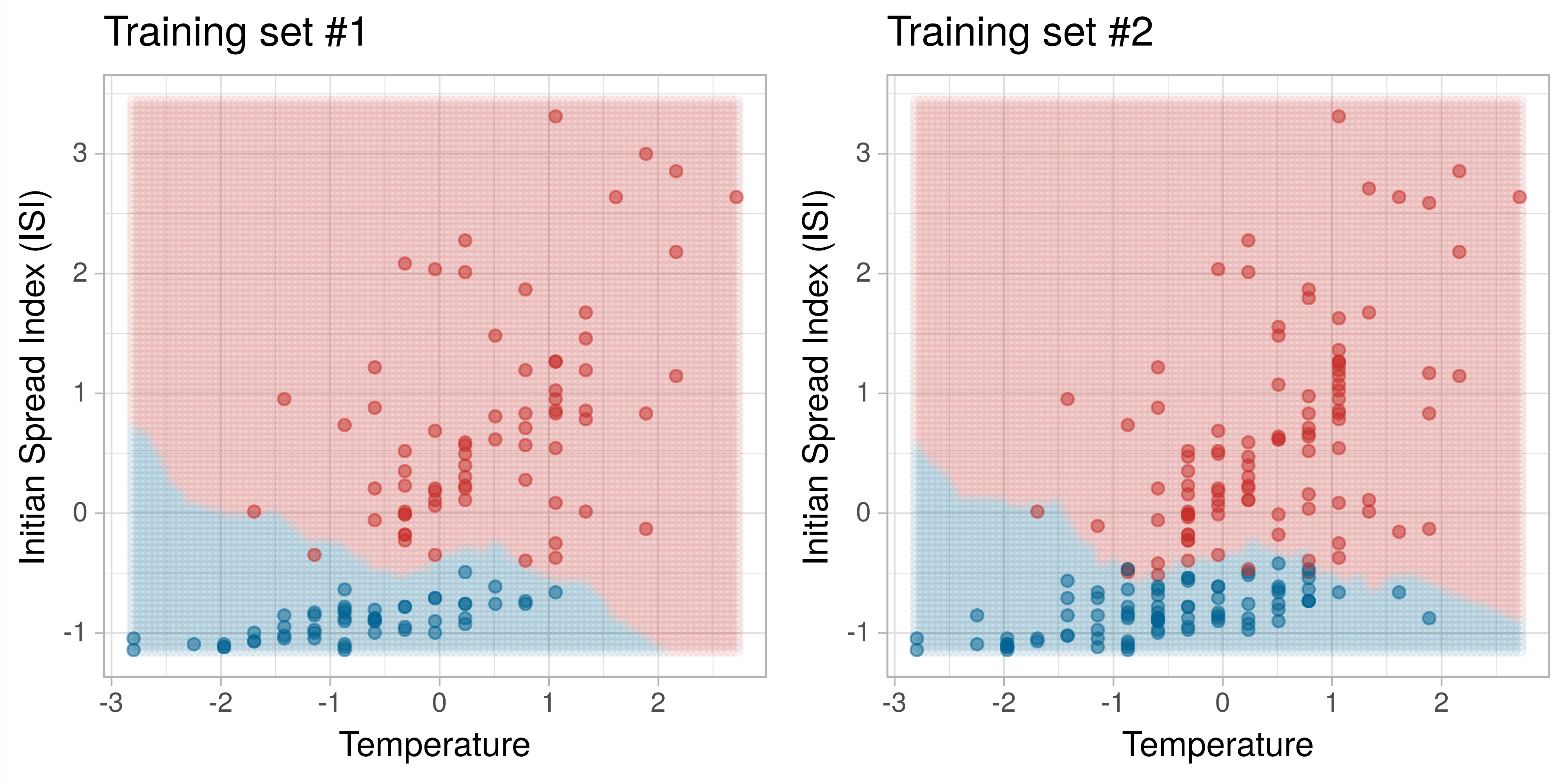
Usually a trial-and-error process by which you

- change some model parameters,
- train the model/algorithm on the data again, then
- compare its performance on a validation set to determine which set of hyper parameters results in the most accurate model.

KNN tuning: find the value of K that creates the best classifier

⚠ **Don't touch the test data set during model tuning!**

Why not to use single (training) test set



Cross validation

Idea: Split the training data up into multiple training-validation pairs, evaluate the classifier on each split and average the performance metrics

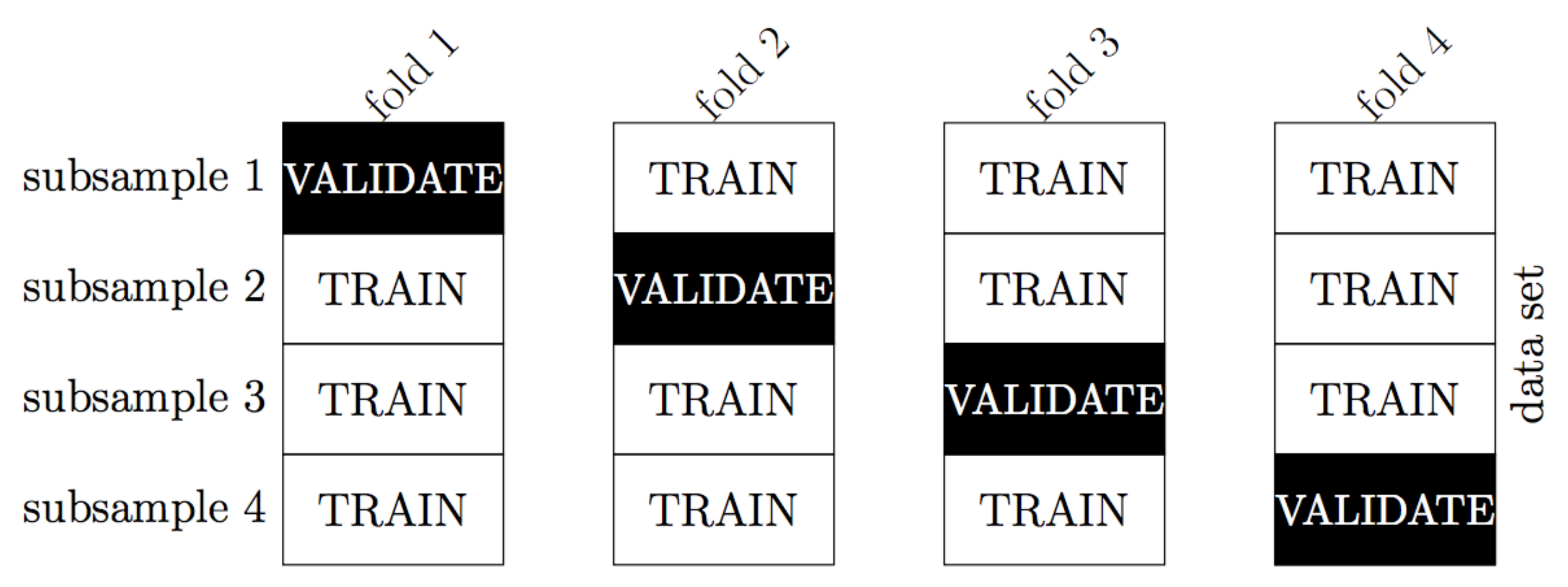
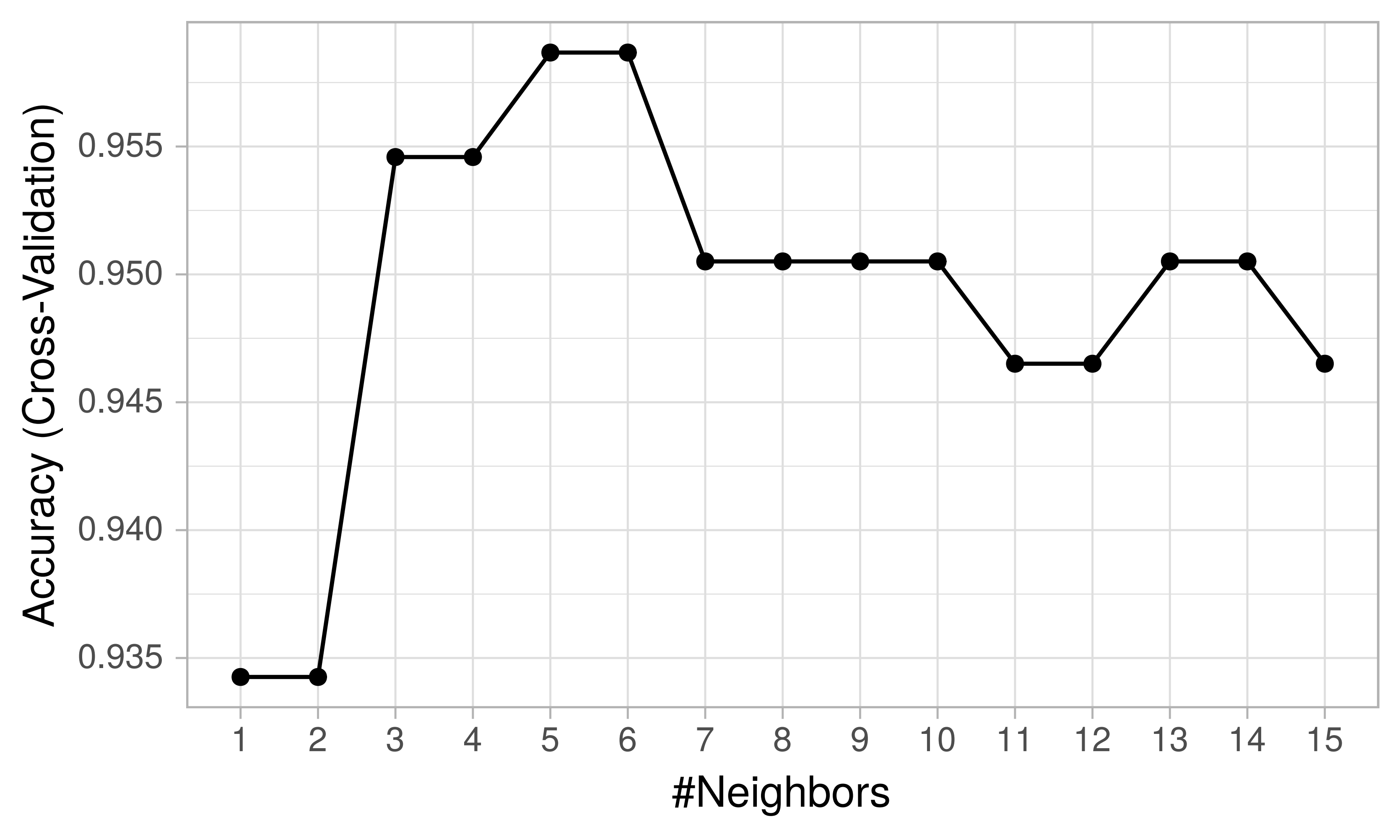


Image courtesy of Dennis Sun

k-fold cross validation

1. split the data into k subsets
2. combine the first $k - 1$ subsets into a training set and train the classifier
3. evaluate the model predictions on the last (i.e. kth) held-out subset
4. repeat steps 2-3 k times (i.e. k "folds"), each time holding out a different one of the k subsets
5. calculate performance metrics from each validation set
6. average each metric over the k folds to come up with a single estimate of that metric



- Based on accuracy, $k = 1$ appears best
- Can look at other metrics
- Accuracy doesn't always decrease with k

5-fold cross validation

Creating the recipe

```
fire_recipe <- recipe(
  classes ~  temperature + isi,
  data = train_complete
) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors())
```

5-fold cross validation

Create your model specification and use `tune()` as a placeholder for the number of neighbors

```
knn_spec <- nearest_neighbor(
  weight_func = "rectangular",
  neighbors = tune()
) %>%
  set_engine("kknn") %>%
  set_mode("classification")
```

Split the `fire_train` data set into `v = 5` folds, stratified by `classes`

```
fire_vfold <- vfold_cv(fire_train, v = 5, strata = classes)
```

5-fold cross validation

Create a grid of K values, the number of neighbors

```
k_vals <- tibble(neighbors = seq(from = 1, to = 15, by = 1))
```

Run 5-fold CV on the `k_vals` grid, storing four performance metrics

```
knn_fit <- workflow() %>%
  add_recipe(fire_recipe) %>%
  add_model(knn_spec) %>%
  tune_grid(
    resamples = fire_vfold,
    grid = k_vals,
    metrics = metric_set(accuracy, sensitivity, specificity, ppv, kap, mcc)
    )
```

Choosing K

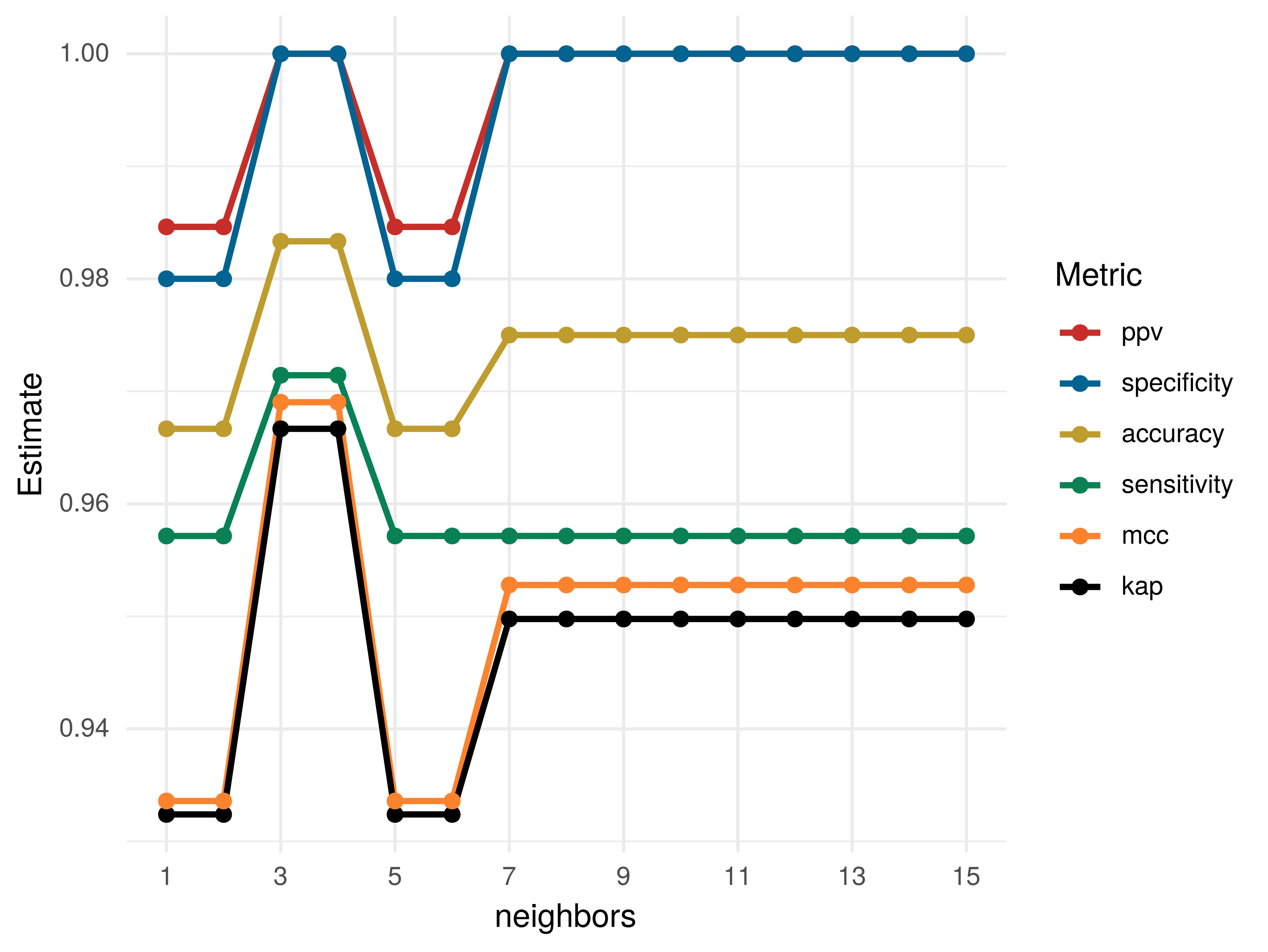
Collect the performance metrics and find the best model

```
cv_metrics <- collect_metrics(knn_fit)
cv_metrics %>% head(6)
# A tibble: 6 × 7
  neighbors .metric     .estimator  mean     n std_err .config
      <dbl> <chr>       <chr>      <dbl> <int>   <dbl> <chr>
1         1 accuracy    binary     0.967     5  0.0243 Preprocessor1_Model0
2         1 kap         binary     0.932     5  0.0493 Preprocessor1_Model0
3         1 mcc         binary     0.934     5  0.0487 Preprocessor1_Model0
4         1 ppv         binary     0.985     5  0.0154 Preprocessor1_Model0
5         1 sensitivity binary     0.957     5  0.0286 Preprocessor1_Model0
6         1 specificity binary     0.98      5  0.02   Preprocessor1_Model0
```

Choosing K

```
cv_metrics %>%
  group_by(.metric) %>%
  slice_max(mean)
# A tibble: 30 × 7
# Groups:   .metric [6]
   neighbors .metric  .estimator  mean     n std_err .config
       <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
 1         3 accuracy binary     0.983     5  0.0167 Preprocessor1_Model03
 2         4 accuracy binary     0.983     5  0.0167 Preprocessor1_Model04
 3         3 kap      binary     0.967     5  0.0333 Preprocessor1_Model03
 4         4 kap      binary     0.967     5  0.0333 Preprocessor1_Model04
 5         3 mcc      binary     0.969     5  0.0310 Preprocessor1_Model03
 6         4 mcc      binary     0.969     5  0.0310 Preprocessor1_Model04
 7         3 ppv      binary     1         5  0      Preprocessor1_Model03
 8         4 ppv      binary     1         5  0      Preprocessor1_Model04
 9         7 ppv      binary     1         5  0      Preprocessor1_Model07
10         8 ppv      binary     1         5  0      Preprocessor1_Model08
# … with 20 more rows
```

Choosing K


1.00
0.98
0.96
0.94
Estimate
1
3
5
7
9
11
13
15
neighbors
Metric
ppv
specificity
accuracy
sensitivity
mcc
kap

The full process

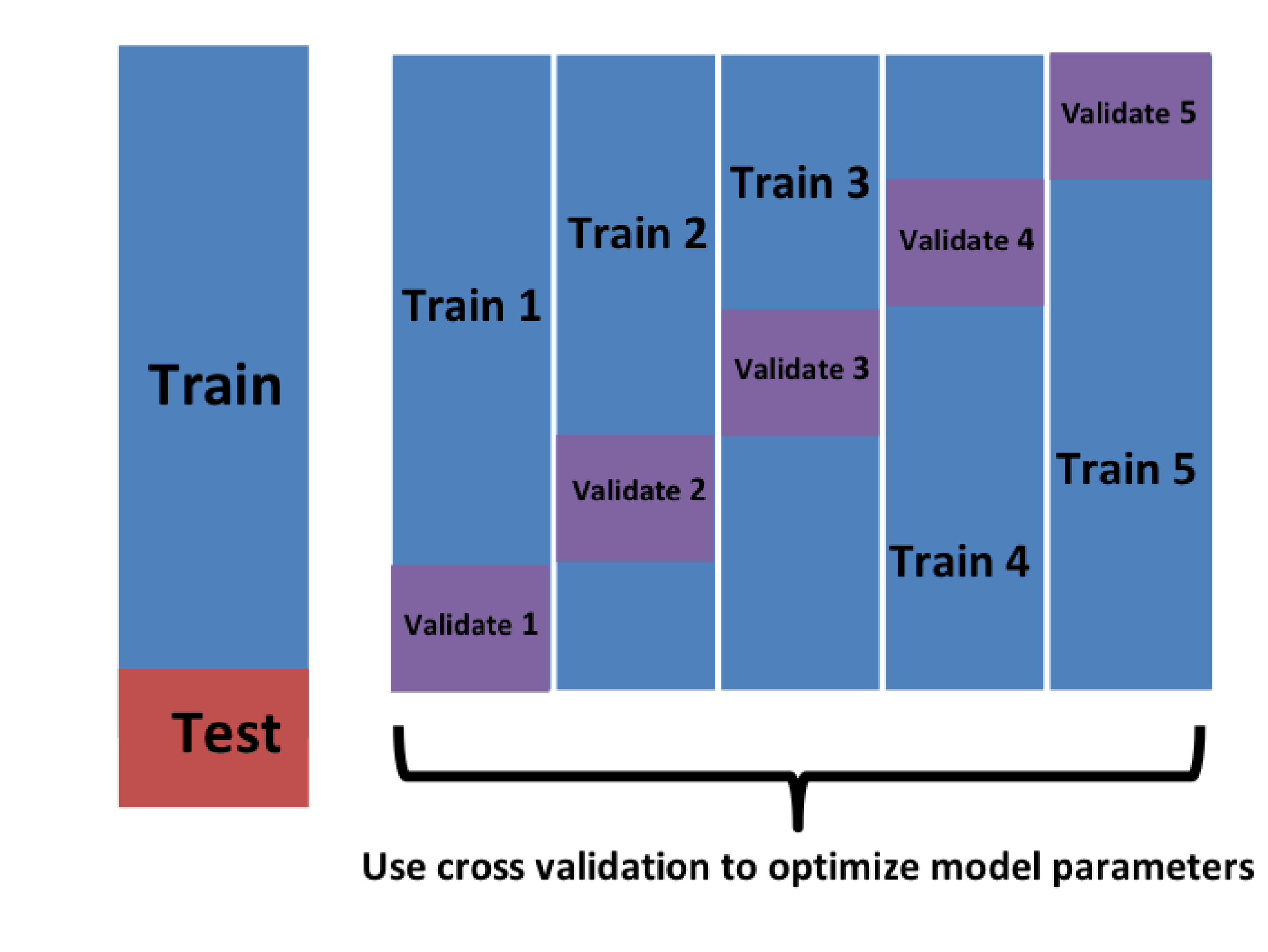


Image source: rafalab.github.io/dsbook/

Your Turn 3

05:00

Follow the steps to run a 5-fold cross validation to find the best value of number of neighbors in the `diabetes` dataset.

```
db_recipe <- recipe(
  diabetes ~  glucose + insulin,
  data = db_raw
) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors())
```