K Nearest Neighbors Regression

The K Nearest Neighbors Model

KNN

We have existing observations

$$(x_1, y_1), \ldots (x_n, y_n)$$

Given a new observation x_{new}, how do we predict y_{new}?

1. Find the 5 values in $\left(x_1, \ldots, x_n\right)$ that are closest to x_{new}
2. Take the average of the corresponding y_i's to our five closest x_i's.
3. Predict $\hat{y}_{new}$ = average of these 5 y_i's

KNN

To perform **K-Nearest-Neighbors**, we choose the **K** closest observations to our *target*, and we average their *response* values.

The Big Questions:

- What is our definition of **closest**?
- What number should we use for **K**?

KNN

K Nearest Neighbors is a **non-parametric** method.

- We aren't assuming anything about how our observations are **generated** or **distributed**.
- (We don't even assume that the sample of observations is **random**!)
- We don't impose any **structure** on our function f
- Least-squares regression: f = $\beta_0 + \beta X + \epsilon$ for *any* X.
- KNN: f = average of 5 closest x_i's *that we observed*

KNN

Recall:

- **regression** is when we are trying to predict a **numeric** response
- **classification** is when we are trying to predict a **categorical** response

K Nearest Neighbors can be used for both!

(but we'll focus on regression for now)

Example

Example

Recall from Assignment 1:

```
ins <- read_csv("https://www.dropbox.com/s/bocjjyo1ehr5auz/insurance.csv?dl=1")
head(ins)
```

```
## # A tibble: 6 x 6
##     age sex      bmi smoker region    charges
##   <dbl> <chr>  <dbl> <chr>  <chr>       <dbl>
## 1    19 female  27.9 yes    southwest  16885.
## 2    33 male    22.7 no     northwest  21984.
## 3    32 male    28.9 no     northwest   3867.
## 4    31 female  25.7 no     southeast   3757.
## 5    60 female  25.8 no     northwest  28923.
## 6    25 male    26.2 no     northeast   2721.
```

Example

Establish our model:

```
knn_mod <- nearest_neighbor(neighbors = 5) %>%
  set_engine("kknn") %>%
  set_mode("regression")
```

- New *engine* - just take it from here: https://www.tidymodels.org/find/parsnip/.

(You will have to `install.packages("knn")` if you are on your home computer.)

- *mode* now matters a lot - "classification" would be possible too!
- New *model function* `nearest_neighbor`, which requires an **input**, `neighbors`

Example

Fit our model:

```
knn_fit_1 <- knn_mod %>%
  fit(charges ~ age, data = ins)

knn_fit_1$fit %>% summary()
```

```
##
## Call:
## kknn::train.kknn(formula = charges ~ age, data = data, ks = ~5)
##
## Type of response variable: continuous
## minimal mean absolute error: 8370.425
## Minimal mean squared error: 128968111
## Best kernel: optimal
## Best k: 5
```

Choosing K

Check your intuition:

1. What happens if we use **K = 1**?

Not necessarily bad, but we could be thrown off by weird outlier observations!

1. What happens if we use **K = (number of observations)**

We predict the same y-value no matter what!

Try it!

Open Activity-KNN-r.Rmd or equivalent

Use cross validation to choose between a KNN model with 5 neighbors that uses only *age* versus one that uses both *age* and *bmi*.

How do these models compare to the *least-squares regression* approach from Tuesday?

How do these models compare to a KNN model with 10 neighbors?

Dummy variables

Dummy variables

Suppose we now want to include `region` in our KNN model.

```
knn_fit_2 <- knn_mod %>%
  fit(charges ~ age + smoker, data = ins)
```

Dummy variables

We can't calculate a **distance** between **categories**!

Instead, we make **dummy variables**:

- `southwest` = 1 if southwest, 0 if not
- `northwest` = 1 if northwest, 0 if not
- ... etc

Now these are (sort of) **numeric** variables.

Recipes

Instead of manually changing the whole dataset, we can "teach" our model workflow what it needs to do to the data.

```
ins_rec <- recipe(charges ~ age + region, data = ins) %>%
  step_dummy(region)

ins_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          2
##
## Operations:
##
## Dummy variables from region
```

Workflows

Now, we can combine our **recipe** (data processing instructions) and our **model choice** into a **workflow**:

```
ins_wflow <- workflow() %>%
  add_recipe(ins_rec) %>%
  add_model(knn_mod)

ins_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_dummy()
##
## -- Model -----------------------------------------------------------------------
## K-Nearest Neighbor Model Specification (regression)
##
## Main Arguments:
##   neighbors = 5
##
## Computational engine: kknn
```

Workflow

```r
ins_fit <- ins_wflow %>% fit(ins)

ins_fit %>% pull_workflow_fit()
```

```
## parsnip model object
##
## Fit time:  20ms
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = ~5)
##
## Type of response variable: continuous
## minimal mean absolute error: 7585.732
## Minimal mean squared error: 120283530
## Best kernel: optimal
## Best k: 5
```

Compare

```
knn_fit_1$fit %>% summary()
```

```
##
## Call:
## kknn::train.kknn(formula = charges ~ age, data = data, ks = ~5)
##
## Type of response variable: continuous
## minimal mean absolute error: 8370.425
## Minimal mean squared error: 128968111
## Best kernel: optimal
## Best k: 5
```

Think about it:

We didn't get much benefit from adding *region*.

But *region* does matter to the response variable!

Why?

Standardizing

Standardizing

- What is the **largest** and **smallest** value of a *dummy variable*?
- What is the **largest** and **smallest** value of *age*?

```
summary(ins$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   22.00   34.00   37.96   55.00   64.00
```

Standardizing

- What is the distance between:
  - Person A: 20 years old, from the southwest
  - Person B: 20 years old, from the northeast

$$\sqrt{(20-20)^2 + (1-0)^2 + (1-0)^2} = 1.41$$

- What is the distance between:
  - Person A: 20 years old, from the southwest
  - Person B: 23 years old, from the southwest

$$\sqrt{(20-23)^2 + (0-0)^2 + (0-0)^2} = 3$$

Normalizing

Let's put `age` on a scale that is comparable to the dummy variables.

How about: mean of 0, standard deviation of 1

(i.e., a z-score)

This is called **normalizing** a variable.

Normalizing

Add it to the workflow!

```
ins_rec <- recipe(charges ~ age + region, data = ins) %>%
  step_dummy(region) %>%
  step_normalize(age)

ins_wflow <- workflow() %>%
  add_recipe(ins_rec) %>%
  add_model(knn_mod)

ins_wflow
```

```
## == Workflow ================================================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor ------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -------------------------------------------------------------------
## K-Nearest Neighbor Model Specification (regression)
```

Normalizing

```
ins_fit <- ins_wflow %>% fit(ins)

ins_fit %>% pull_workflow_fit()
```

```
## parsnip model object
##
## Fit time:  0ms
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = ~5)
##
## Type of response variable: continuous
## minimal mean absolute error: 7508.632
## Minimal mean squared error: 118115709
## Best kernel: optimal
## Best k: 5
```

Try it!

Open *Activity-KNN-r.Rmd* again

Make a KNN model with K = 5, using *age*, *bmi*, *smoker*, and *sex*

Compare the model with non-normalized variables to one with normalized variables. Which is better?

Tuning

How do we choose K?

Tuning

K is what is called a **tuning parameter**.

This is a feature of a model that we have to chose *before* fitting the model.

Ideally, we'd try many values of the **tuning parameter** and find the best one.


Fender


PONTIAC

Automatic tuning

```
knn_mod <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("regression")

k_grid <- grid_regular(neighbors())

k_grid
```

```
## # A tibble: 3 x 1
##   neighbors
##       <int>
## 1         1
## 2         5
## 3        10
```

Automatic tuning

```
knn_mod_tune <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("regression")

k_grid <- grid_regular(neighbors(c(1,50)),
                       levels = 25)

k_grid
```

```
## # A tibble: 25 x 1
##    neighbors
##        <int>
##  1         1
##  2         3
##  3         5
##  4         7
##  5         9
##  6        11
##  7        13
##  8        15
##  9        17
## 10        19
```

Tuning

```r
ins_rec <- recipe(charges ~ age + bmi + sex + smoker, data = ins) %>%
  step_dummy(all_nominal()) %>%
  step_normalize(all_numeric())

ins_wflow <- workflow() %>%
  add_recipe(ins_rec) %>%
  add_model(knn_mod_tune)
```

```r
ins_cv <- vfold_cv(ins, v = 10)


knn_grid_search <-
  tune_grid(
    ins_wflow,
    resamples = ins_cv,
    grid = k_grid
  )
```

Tuning

```
knn_grid_search
```

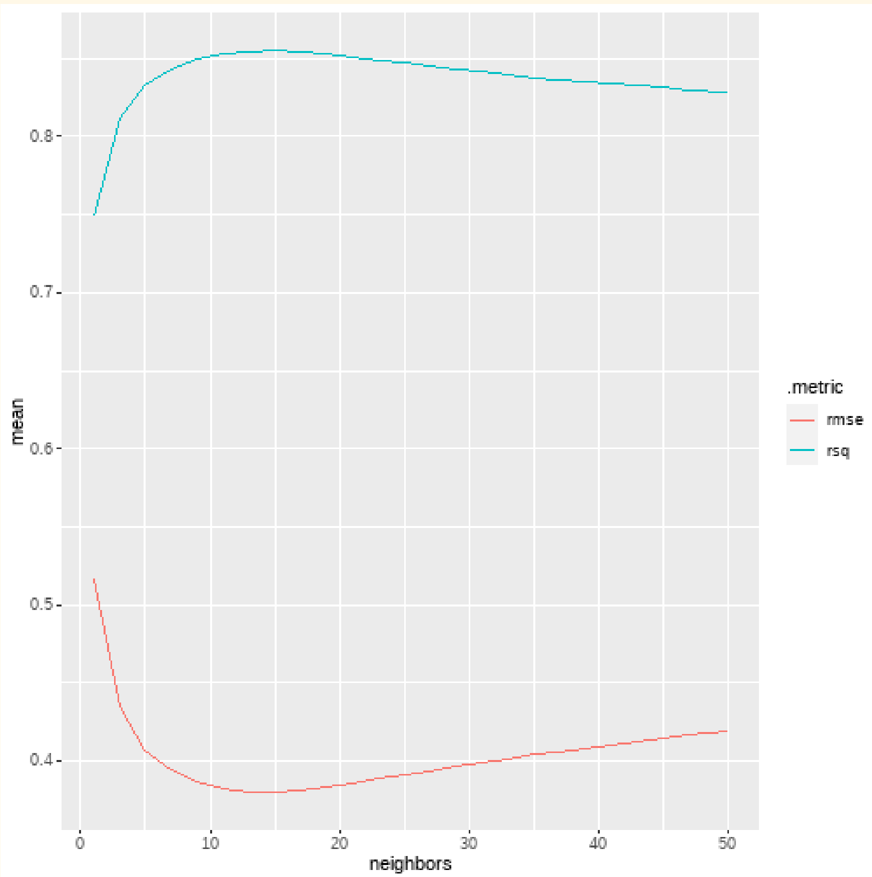
```
## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits           id     .metrics              .notes
##    <list>           <chr>  <list>                <list>
##  1 <rsplit [387/44]> Fold01 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  2 <rsplit [388/43]> Fold02 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  3 <rsplit [388/43]> Fold03 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  4 <rsplit [388/43]> Fold04 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  5 <rsplit [388/43]> Fold05 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  6 <rsplit [388/43]> Fold06 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  7 <rsplit [388/43]> Fold07 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  8 <rsplit [388/43]> Fold08 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
##  9 <rsplit [388/43]> Fold09 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
## 10 <rsplit [388/43]> Fold10 <tibble[,5] [50 x 5]> <tibble[,1] [0 x 1]>
```

Tuning

```
knn_grid_search %>% collect_metrics()
```

```
## # A tibble: 50 x 7
##    neighbors .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 rmse    standard   0.516    10  0.0510 Model01
##  2         1 rsq     standard   0.749    10  0.0322 Model01
##  3         3 rmse    standard   0.434    10  0.0379 Model02
##  4         3 rsq     standard   0.812    10  0.0215 Model02
##  5         5 rmse    standard   0.406    10  0.0346 Model03
##  6         5 rsq     standard   0.833    10  0.0196 Model03
##  7         7 rmse    standard   0.394    10  0.0329 Model04
##  8         7 rsq     standard   0.843    10  0.0187 Model04
##  9         9 rmse    standard   0.386    10  0.0316 Model05
## 10         9 rsq     standard   0.849    10  0.0178 Model05
## # ... with 40 more rows
```

Tuning



Tuning

What if we had only looked at k from 1 to 10?

Tuning

What if we had only looked at k from 20 to 50?

Tuning

```
knn_grid_search %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  slice_min(mean)
```

```
## # A tibble: 1 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        15 rmse    standard   0.379    10  0.0292 Model08
```

Try it!

Open *Activity-KNN-r.Rmd* again

Decide on a best final KNN model to predict insurance charges

Plot the residuals from this model