CMSC/LING/STAT 208: Machine Learning

Abhishek Chakraborty [Much of the content in these slides have been adapted from *An Introduction to Statistical Learning: with Applications in R*, James et al.]

CV to Tune Hyperparameter

Auto dataset

```
library(ISLR2)  # load library

data("Auto")   # load dataset
```

We will consider `mpg` as the response and `horsepower` as the predictor.

```
# select the variables to work with

Auto <- Auto %>% select(mpg, horsepower)
```

CV to Tune Hyperparameter

Objective: Find the optimum choice of K in the KNN approach with 5-fold CV repeated 5 times. We will use the following steps.

- Perform EDA (Exploratory Data Analysis)
- Split the data into training and test data (80-20 split).
- Specify CV specifications using **trainControl**.
- Create an object **k_grid** using the following code.

```
k_grid <- expand.grid(k = seq(1, 100, by = 1))  # creates a grid of k values to be used (1 to 100 in this case)
```

- Use the **train** function to run CV. Use **method = "knn"**, **tuneGrid = k_grid**, and **metric = "RMSE"**.
- Obtain the results and plot them. What is the optimum k chosen?
- Create the final model using the optimum k and estimate its prediction error from the test data.

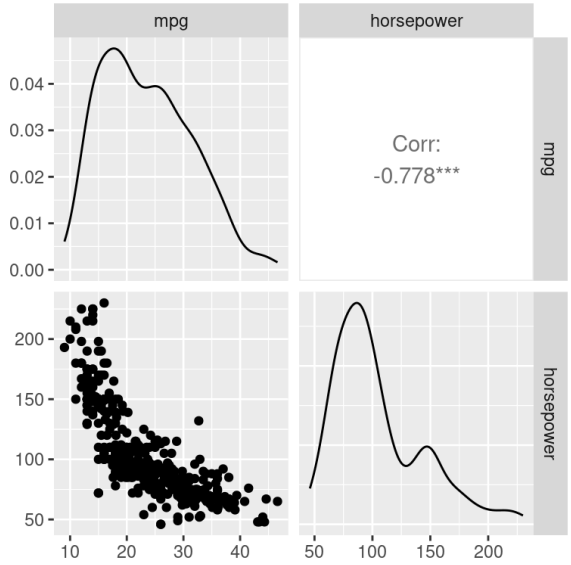
CV to Tune Hyperparameter: EDA

Auto dataset

```
summary(Auto)   # summary of the variables
```

```
##       mpg          horsepower
##  Min.   : 9.00   Min.   : 46.0
##  1st Qu.:17.00   1st Qu.: 75.0
##  Median :22.75   Median : 93.5
##  Mean   :23.45   Mean   :104.5
##  3rd Qu.:29.00   3rd Qu.:126.0
##  Max.   :46.60   Max.   :230.0
```

```
library(GGally)
ggpairs(Auto)    # correlation plot
```



CV to Tune Hyperparameter: Split Data

Auto dataset

```
set.seed(041824)  # fix the random number generator for reproducibility

library(caret)  # load library

train_index <- createDataPartition(y = Auto$mpg, p = 0.8, list = FALSE) # split available data into 80% training and 20% test datasets

Auto_train <- Auto[train_index,]   # training data

Auto_test <- Auto[-train_index,]   # test data
```

CV to Tune Hyperparameter: Perform CV

Auto dataset

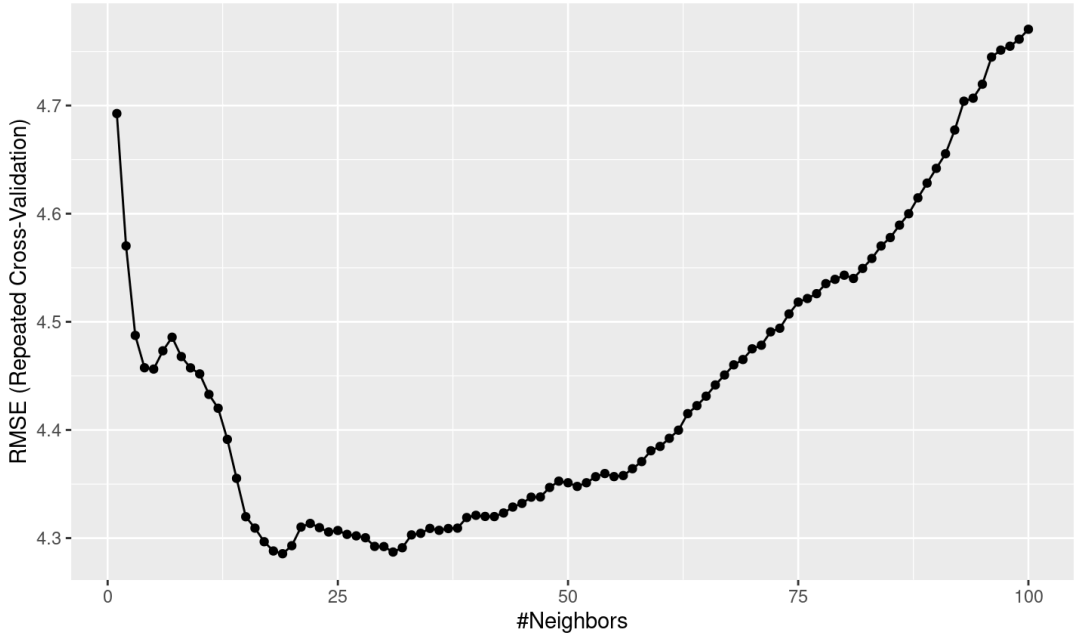
```
set.seed(041824)  # fix the random number generator for reproducibility

# CV specifications
cv_specs <- trainControl(method = "repeatedcv", number = 5, repeats = 5)

# specify grid of 'k' values to search over
k_grid <- expand.grid(k = seq(1, 100, by = 1))

# train the KNN model using CV  to find optimal 'k'
knn_cv <- train(form = mpg ~ horsepower,
                data = Auto_train,
                method = "knn",
                trControl = cv_specs,
                tuneGrid = k_grid,
                metric = "RMSE")
```

CV to Tune Hyperparameter: Compare CV Results

Auto dataset

```
knn_cv   # CV results, shows RMSE for all K
```

```
ggplot(knn_cv)   # plot CV results for different 'k'
```



CV to Tune Hyperparameter: Final Model

Auto dataset

```
# final model with optimal 'k' chosen from CV

knn_cv$bestTune     # optimal value of K
```

```
##     k
## 19 19
```

```
knn_cv$finalModel   # final model
```

```
## 19-nearest neighbor regression model
```

```
# obtain predictions on test data
final_model_preds <- predict(knn_cv, newdata = Auto_test)

# estimate test set prediction error
sqrt(mean((Auto_test$mpg - final_model_preds)^2))    # test set RMSE
```

```
## [1] 4.226318
```

Data Preprocessing and Feature Enginnering

Data preprocessing and engineering techniques generally refer to the addition, deletion, or transformation of data.

We will cover several fundamental and common preprocessing tasks that can potentially significantly improve modeling performance.

- Dealing with zero-variance (zv) and/or near-zero variance (nzv) variables
- Imputing missing entries
- Label encoding ordinal categorical variables
- Standardizing (centering and scaling) numeric predictors
- Lumping predictors
- One-hot/dummy encoding categorical predictors

Ames Housing Dataset

```
ames <- readRDS("AmesHousing.rds")   # load dataset
```

```
glimpse(ames)  # check type of features
```

```
## Rows: 881
## Columns: 20
## $ Sale_Price    <int> 244000, 213500, 185000, 394432, 190000, 149000, 149900, …
## $ Gr_Liv_Area   <int> 2110, 1338, 1187, 1856, 1844, NA, NA, 1069, 1940, 1544, …
## $ Garage_Type   <fct> Attchd, Attchd, Attchd, Attchd, Attchd, Attchd, Attchd, …
## $ Garage_Cars   <dbl> 2, 2, 2, 3, 2, 2, 2, 2, 3, 3, 2, 3, 3, 2, 2, 2, 3, 2, 2,…
## $ Garage_Area   <dbl> 522, 582, 420, 834, 546, 480, 500, 440, 606, 868, 532, 7…
## $ Street        <fct> Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pa…
## $ Utilities     <fct> AllPub, AllPub, AllPub, AllPub, AllPub, AllPub, AllPub, …
## $ Pool_Area     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
## $ Neighborhood  <fct> North_Ames, Stone_Brook, Gilbert, Stone_Brook, Northwest…
## $ Screen_Porch  <int> 0, 0, 0, 0, 0, 0, 0, 165, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ Overall_Qual  <fct> Good, Very_Good, Above_Average, Excellent, Above_Average…
## $ Lot_Area      <int> 11160, 4920, 7980, 11394, 11751, 11241, 12537, 4043, 101…
## $ Lot_Frontage  <dbl> 93, 41, 0, 88, 105, 0, 0, 53, 83, 94, 95, 90, 105, 61, 6…
## $ MS_SubClass   <fct> One_Story_1946_and_Newer_All_Styles, One_Story_PUD_1946_…
## $ Misc_Val      <int> 0, 0, 500, 0, 0, 700, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ Open_Porch_SF <int> 0, 0, 21, 0, 122, 0, 0, 55, 95, 35, 70, 74, 130, 82, 48,…
## $ TotRms_AbvGrd <int> 8, 6, 6, 8, 7, 5, 6, 4, 8, 7, 7, 7, 7, 6, 7, 7, 10, 7, 7…
## $ First_Flr_SF  <int> 2110, 1338, 1187, 1856, 1844, 1004, 1078, 1069, 1940, 15…
## $ Second_Flr_SF <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 563, 0, 886, 656, 11…
## $ Year_Built    <int> 1968, 2001, 1992, 2010, 1977, 1970, 1971, 1977, 2009, 20…
```

```
sum(is.na(ames))    # check for missing entries
```

```
## [1] 113
```

Ames Housing Dataset

```
summary(ames)  # check type of features, which features have missing entries?
```

```
##    Sale_Price      Gr_Liv_Area                Garage_Type   Garage_Cars
##  Min.   : 34900   Min.   : 334   Attchd             :514   Min.   :0.000
##  1st Qu.:129500   1st Qu.:1118   Basment            : 10   1st Qu.:1.000
##  Median :160000   Median :1442   BuiltIn            : 55   Median :2.000
##  Mean   :181115   Mean   :1495   CarPort            :  5   Mean   :1.762
##  3rd Qu.:213500   3rd Qu.:1728   Detchd             :234   3rd Qu.:2.000
##  Max.   :755000   Max.   :5642   More_Than_Two_Types:  9   Max.   :4.000
##                   NA's   :113    No_Garage          : 54
##   Garage_Area      Street     Utilities     Pool_Area
##  Min.   :   0.0   Grvl:  4   AllPub:880   Min.   :  0.00
##  1st Qu.: 324.0   Pave:877   NoSeWa:  0   1st Qu.:  0.00
##  Median : 480.0              NoSewr:  1   Median :  0.00
##  Mean   : 476.5                           Mean   :  2.41
##  3rd Qu.: 592.0                           3rd Qu.:  0.00
##  Max.   :1418.0                           Max.   :576.00
##
##              Neighborhood  Screen_Porch           Overall_Qual    Lot_Area
##  North_Ames        :127   Min.   :  0.00   Average      :243   Min.   :  1300
##  College_Creek     : 86   1st Qu.:  0.00   Above_Average:217   1st Qu.:  7449
##  Old_Town          : 83   Median :  0.00   Good         :177   Median :  9512
##  Northridge_Heights: 52   Mean   : 18.11   Very_Good    : 99   Mean   : 10105
##  Somerset          : 50   3rd Qu.:  0.00   Below_Average: 83   3rd Qu.: 11526
##  Edwards           : 49   Max.   :490.00   Excellent    : 38   Max.   :159000
##  (Other)           :434                    (Other)      : 24
##   Lot_Frontage                                  MS_SubClass     Misc_Val
##  Min.   :  0.00   One_Story_1946_and_Newer_All_Styles :335   Min.   :   0.00
##  1st Qu.: 43.00   Two_Story_1946_and_Newer            :171   1st Qu.:   0.00
##  Median : 63.00   One_and_Half_Story_Finished_All_Ages: 92   Median :   0.00
##  Mean   : 57.78   One_Story_PUD_1946_and_Newer        : 53   Mean   :  37.97
##  3rd Qu.: 78.00   Duplex_All_Styles_and_Ages          : 40   3rd Qu.:   0.00
##  Max.   :313.00   One_Story_1945_and_Older            : 36   Max.   :8300.00
##                   (Other)                             :154
##  Open_Porch_SF    TotRms_AbvGrd     First_Flr_SF  Second_Flr_SF
##  Min.   :  0.00   Min.   : 2.000   Min.   : 334   Min.   :   0.0
##  1st Qu.:  0.00   1st Qu.: 5.000   1st Qu.: 877   1st Qu.:   0.0
##  Median : 27.00   Median : 6.000   Median :1092   Median :   0.0
##  Mean   : 49.93   Mean   : 6.413   Mean   :1171   Mean   : 319.6
##  3rd Qu.: 72.00   3rd Qu.: 7.000   3rd Qu.:1426   3rd Qu.: 682.0
##  Max.   :742.00   Max.   :12.000   Max.   :4692   Max.   :2065.0
##
##    Year_Built
##  Min.   :1875
##  1st Qu.:1954
```

Ames Housing Dataset

```
levels(ames$Overall_Qual)   # the levels are NOT properly ordered
```

```
##  [1] "Above_Average"  "Average"        "Below_Average"  "Excellent"
##  [5] "Fair"           "Good"           "Poor"           "Very_Excellent"
##  [9] "Very_Good"      "Very_Poor"
```

```
# relevel the levels
ames$Overall_Qual <- factor(ames$Overall_Qual, levels = c("Very_Poor", "Poor", "Fair", "Below_Average",
                                                          "Average", "Above_Average", "Good", "Very_Good",
                                                          "Excellent", "Very_Excellent"))

levels(ames$Overall_Qual)   # the levels are properly ordered
```

```
##  [1] "Very_Poor"      "Poor"           "Fair"           "Below_Average"
##  [5] "Average"        "Above_Average"  "Good"           "Very_Good"
##  [9] "Excellent"      "Very_Excellent"
```

Ames Housing Dataset

```
# split the dataset into training and test sets

set.seed(042324)   # set seed

train_index <- createDataPartition(ames$Sale_Price, p = 0.8, list = FALSE)   # 'Sale_Price' is the response

ames_train <- ames[train_index,]   # training data

ames_test <- ames[-train_index,]   # test data
```

Ames Housing Dataset

```
# set up the recipe

library(recipes)

ames_recipe <- recipe(Sale_Price ~ ., data = ames_train)   # sets up the type and role of variables

ames_recipe$var_info
```

```
## # A tibble: 20 × 4
##    variable      type      role      source
##    <chr>         <list>    <chr>     <chr>
##  1 Gr_Liv_Area   <chr [2]> predictor original
##  2 Garage_Type   <chr [3]> predictor original
##  3 Garage_Cars   <chr [2]> predictor original
##  4 Garage_Area   <chr [2]> predictor original
##  5 Street        <chr [3]> predictor original
##  6 Utilities     <chr [3]> predictor original
##  7 Pool_Area     <chr [2]> predictor original
##  8 Neighborhood  <chr [3]> predictor original
##  9 Screen_Porch  <chr [2]> predictor original
## 10 Overall_Qual  <chr [3]> predictor original
## 11 Lot_Area      <chr [2]> predictor original
## 12 Lot_Frontage  <chr [2]> predictor original
## 13 MS_SubClass   <chr [3]> predictor original
## 14 Misc_Val      <chr [2]> predictor original
## 15 Open_Porch_SF <chr [2]> predictor original
## 16 TotRms_AbvGrd <chr [2]> predictor original
## 17 First_Flr_SF  <chr [2]> predictor original
## 18 Second_Flr_SF <chr [2]> predictor original
## 19 Year_Built    <chr [2]> predictor original
## 20 Sale_Price    <chr [2]> outcome   original
```

Zero-Variance (zv) and/or Near-Zero Variance (nzv) Variables

A rule of thumb for detecting near-zero variance features is:

- The fraction of unique values over the sample size is low (say $\leq 10\%$).
- The ratio of the frequency of the most prevalent value to the frequency of the second most prevalent value is large (say $\geq 20\%$).

Zero-Variance (zv) and/or Near-Zero Variance (nzv) Variables

```
# investigate zv/nzv predictors

nearZeroVar(ames, saveMetrics = TRUE)   # check which predictors are zv/nzv
```

```
##                freqRatio percentUnique zeroVar   nzv
## Sale_Price      1.000000    55.7321226   FALSE FALSE
## Gr_Liv_Area     1.333333    62.9965948   FALSE FALSE
## Garage_Type     2.196581     0.7945516   FALSE FALSE
## Garage_Cars     1.970213     0.5675369   FALSE FALSE
## Garage_Area     2.250000    38.0249716   FALSE FALSE
## Street        219.250000     0.2270148   FALSE  TRUE
## Utilities     880.000000     0.2270148   FALSE  TRUE
## Pool_Area     876.000000     0.6810443   FALSE  TRUE
## Neighborhood    1.476744     2.9511918   FALSE FALSE
## Screen_Porch  199.750000     6.6969353   FALSE  TRUE
## Overall_Qual    1.119816     1.1350738   FALSE FALSE
## Lot_Area        1.071429    79.7956867   FALSE FALSE
## Lot_Frontage    1.617021    11.5777526   FALSE FALSE
## MS_SubClass     1.959064     1.7026107   FALSE FALSE
## Misc_Val      141.833333     1.9296254   FALSE  TRUE
## Open_Porch_SF  23.176471    19.2962543   FALSE FALSE
## TotRms_AbvGrd   1.311224     1.2485812   FALSE FALSE
## First_Flr_SF    1.777778    63.7911464   FALSE FALSE
## Second_Flr_SF  64.250000    31.3280363   FALSE FALSE
## Year_Built      1.116279    12.0317821   FALSE FALSE
```

Zero-Variance (zv) and/or Near-Zero Variance (nzv) Variables

```
# investigate zv/nzv predictors

nearZeroVar(ames_train, saveMetrics = TRUE)   # check which predictors are zv/nzv
```

```
##                freqRatio percentUnique zeroVar   nzv
## Sale_Price      1.090909    59.2067989   FALSE FALSE
## Gr_Liv_Area     2.000000    66.9971671   FALSE FALSE
## Garage_Type     2.081218     0.9915014   FALSE FALSE
## Garage_Cars     1.933333     0.7082153   FALSE FALSE
## Garage_Area     1.714286    41.9263456   FALSE FALSE
## Street        234.333333     0.2832861   FALSE  TRUE
## Utilities     705.000000     0.2832861   FALSE  TRUE
## Pool_Area     702.000000     0.7082153   FALSE  TRUE
## Neighborhood    1.500000     3.6827195   FALSE FALSE
## Screen_Porch  159.250000     7.2237960   FALSE  TRUE
## Overall_Qual    1.072626     1.4164306   FALSE FALSE
## Lot_Area        1.166667    82.4362606   FALSE FALSE
## Lot_Frontage    1.623377    13.8810198   FALSE FALSE
## MS_SubClass     1.920290     2.1246459   FALSE FALSE
## Misc_Val      136.200000     2.4079320   FALSE  TRUE
## Open_Porch_SF  21.200000    21.6713881   FALSE FALSE
## TotRms_AbvGrd   1.324841     1.4164306   FALSE FALSE
## First_Flr_SF    1.857143    67.1388102   FALSE FALSE
## Second_Flr_SF  69.000000    33.0028329   FALSE FALSE
## Year_Built      1.083333    14.4475921   FALSE FALSE
```

Zero-Variance (zv) and/or Near-Zero Variance (nzv) Variables

```
blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val)    # filter out zv/nzv predictors
```

Imputing Missing Entries

Possible imputation techniques:

- `step_impute_median`: used for numeric (especially discrete) variables
- `step_impute_mean`: used for numeric variables
- `step_impute_knn`: used for both numeric and categorical variables (computationally expensive)
- `step_impute_mode`: used for nominal (having no order) categorical variables

Imputing Missing Entries

```
summary(ames_train)   # check which predictors have missing entries
```

```
##    Sale_Price      Gr_Liv_Area                Garage_Type   Garage_Cars
##  Min.   : 46500   Min.   : 599   Attchd             :410   Min.   :0.000
##  1st Qu.:129425   1st Qu.:1119   Basment            :  9   1st Qu.:1.000
##  Median :160000   Median :1445   BuiltIn            : 45   Median :2.000
##  Mean   :180946   Mean   :1500   CarPort            :  4   Mean   :1.766
##  3rd Qu.:213375   3rd Qu.:1736   Detchd             :197   3rd Qu.:2.000
##  Max.   :755000   Max.   :5642   More_Than_Two_Types:  5   Max.   :4.000
##                   NA's   :87     No_Garage          : 36
##   Garage_Area      Street     Utilities     Pool_Area             Neighborhood
##  Min.   :   0.0   Grvl:  3   AllPub:705   Min.   :  0.000   North_Ames   :111
##  1st Qu.: 324.8   Pave:703   NoSeWa:  0   1st Qu.:  0.000   College_Creek: 74
##  Median : 480.0              NoSewr:  1   Median :  0.000   Old_Town     : 66
##  Mean   : 478.4                           Mean   :  2.191   Gilbert      : 40
##  3rd Qu.: 585.5                           3rd Qu.:  0.000   Sawyer       : 39
##  Max.   :1418.0                           Max.   :555.000   Somerset     : 38
##                                                             (Other)      :338
##   Screen_Porch           Overall_Qual    Lot_Area       Lot_Frontage
##  Min.   :  0.00   Average      :192   Min.   :  1470   Min.   :  0.00
##  1st Qu.:  0.00   Above_Average:179   1st Qu.:  7312   1st Qu.: 42.25
##  Median :  0.00   Good         :147   Median :  9506   Median : 60.00
##  Mean   : 18.56   Very_Good    : 78   Mean   : 10176   Mean   : 57.56
##  3rd Qu.:  0.00   Below_Average: 65   3rd Qu.: 11615   3rd Qu.: 78.00
##  Max.   :490.00   Excellent    : 28   Max.   :159000   Max.   :313.00
##                   (Other)      : 17
##                                MS_SubClass     Misc_Val       Open_Porch_SF
##  One_Story_1946_and_Newer_All_Styles :265   Min.   :   0.00   Min.   :  0.00
##  Two_Story_1946_and_Newer            :138   1st Qu.:   0.00   1st Qu.:  0.00
##  One_and_Half_Story_Finished_All_Ages: 78   Median :   0.00   Median : 26.00
##  One_Story_PUD_1946_and_Newer        : 45   Mean   :  43.91   Mean   : 49.41
##  Duplex_All_Styles_and_Ages          : 32   3rd Qu.:   0.00   3rd Qu.: 73.00
##  One_Story_1945_and_Older            : 31   Max.   :8300.00   Max.   :742.00
##  (Other)                             :117
##  TotRms_AbvGrd     First_Flr_SF  Second_Flr_SF    Year_Built
##  Min.   : 3.000   Min.   : 407   Min.   :   0   Min.   :1875
##  1st Qu.: 5.000   1st Qu.: 874   1st Qu.:   0   1st Qu.:1954
##  Median : 6.000   Median :1092   Median :   0   Median :1972
##  Mean   : 6.408   Mean   :1171   Mean   : 319   Mean   :1971
##  3rd Qu.: 7.000   3rd Qu.:1425   3rd Qu.: 675   3rd Qu.:2000
##  Max.   :12.000   Max.   :4692   Max.   :2065   Max.   :2010
##
```

Imputing Missing Entries

```
blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%      # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area)       # impute missing entries
```

Label Encoding Ordinal Categorical Variables

Label encoding is a pure numeric conversion of the levels of a categorical variable. If a categorical variable is a factor and it has pre-specified levels then the numeric conversion will be in level order. If no levels are specified, the encoding will be based on alphabetical order.

We should be careful with label encoding unordered categorical features because most models will treat them as ordered numeric features

Label Encoding Ordinal Categorical Variables

```
# investigate predictors with possible ordering (label encoding)

ames_train %>% count(Overall_Qual)
```

```
## # A tibble: 10 × 2
##    Overall_Qual       n
##    <fct>          <int>
##  1 Very_Poor          1
##  2 Poor               3
##  3 Fair               6
##  4 Below_Average     65
##  5 Average          192
##  6 Above_Average    179
##  7 Good             147
##  8 Very_Good         78
##  9 Excellent         28
## 10 Very_Excellent     7
```

Label Encoding Ordinal Categorical Variables

```
blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%    # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area) %>%      # impute missing entries
  step_integer(Overall_Qual)        # numeric conversion of levels of the predictors
```

Standardizing (centering and scaling) Numeric Predictors

Standardizing features includes **centering** and **scaling** so that numeric variables have zero mean and unit variance, which provides a common comparable unit of measure across all the variables.

Before centering and scaling, it is better to remove zv/nzv variables, and perform necessary imputation and label encoding.

```
blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%      # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area) %>%       # impute missing entries
  step_integer(Overall_Qual) %>%       # numeric conversion of levels of the predictors
  step_center(all_numeric(), -all_outcomes()) %>%      # center (subtract mean) all numeric predictors
  step_scale(all_numeric(), -all_outcomes())           # scale (divide by standard deviation) all numeric predictors
```

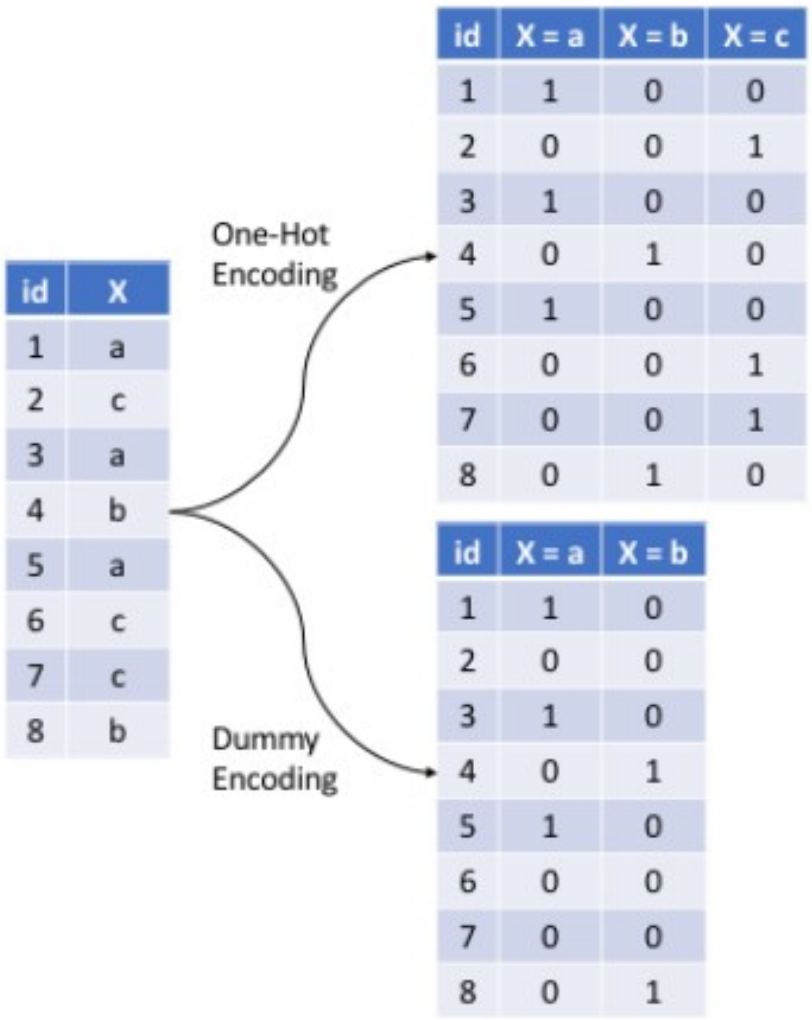
Lumping Predictors

Sometimes features (numerical or categorical) will contain levels that have very few observations (decided by a threshold). It can be beneficial to collapse, or "lump" these into a lesser number of categories.

```
# lumping categorical predictors if need be

ames_train %>% count(Neighborhood) %>% arrange(n)    # check frequency of categories
```

```
blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%  # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area) %>%                                   # impute missing entries
  step_integer(Overall_Qual) %>%                                      # numeric conversion of levels of the predictors
  step_center(all_numeric(), -all_outcomes()) %>%                     # center (subtract mean) all numeric predictors
  step_scale(all_numeric(), -all_outcomes()) %>%                      # scale (divide by standard deviation) all numeric predictors
  step_other(Neighborhood, threshold = 0.01, other = "other")         # lumping required predictors
```

One-hot/dummy Encoding Categorical Predictors



Adapted from Hands-on Machine Learning with R, Bradley Boehmke & Brandon Greenwell

One-hot/dummy Encoding Categorical Predictors

```
blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%  # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area) %>%                                   # impute missing entries
  step_integer(Overall_Qual) %>%                                      # numeric conversion of levels of the predictors
  step_center(all_numeric(), -all_outcomes()) %>%                     # center (subtract mean) all numeric predictors
  step_scale(all_numeric(), -all_outcomes()) %>%                      # scale (divide by standard deviation) all numeric predictors
  step_other(Neighborhood, threshold = 0.01, other = "other") %>%     # lumping required predictors
  step_dummy(all_nominal(), one_hot = FALSE)                          # one-hot/dummy encode nominal categorical predictors
```

Preprocessing Steps

A suggested order of potential steps that should work for most problems:

1. Filter out zero or near-zero variance features.
2. Perform imputation if required.
3. Label encode ordinal categorical features.
4. Normalize/Standardize (center and scale) numeric features.
5. Lump certain features if required.
6. One-hot or dummy encode categorical features.

Data Leakage (A Serious, Common Problem)

Data leakage is when information from outside the training data set is used to create the model. Data leakage often occurs when the data preprocessing task is implemented with CV. To minimize this, feature engineering should be done in isolation of each resampling iteration.

Data Leakage (Example)

KNN Classification: Toy Example

Obs.	X_1	X_2	Y
1	1033	1.7	Red
2	1112	1.5	Red
3	1500	1	Red
4	999	1	Green
5	1012	1.5	Green
6	1013	1	Red
7	1233	1	Green
8	1332	1	Red

Suppose you implement 4-fold CV. Let's say the folds are randomly chosen to be observation pairs (2, 3), (4, 7), (1, 8), and (5, 6).

Preprocessing With `recipes` Package

There are three main steps in creating and applying feature engineering with `recipes`:

- **recipe:** where you define your feature engineering steps to create your blueprint
- **prepare:** estimate feature engineering parameters based on training data
- **bake:** apply the blueprint to new data

Preprocessing With **recipes** Package

```
# finally, after all preprocessing steps have been decided set up the overall blueprint

ames_recipe <- recipe(Sale_Price ~ ., data = ames_train)

blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%
  step_impute_mean(Gr_Liv_Area) %>%
  step_integer(Overall_Qual) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_other(Neighborhood, threshold = 0.01, other = "other") %>%
  step_dummy(all_nominal(), one_hot = FALSE)


prepare <- prep(blueprint, data = ames_train)    # estimate feature engineering parameters based on training data


baked_train <- bake(prepare, new_data = ames_train)   # apply the blueprint to training data for building final/optimal model

baked_test <- bake(prepare, new_data = ames_test)    # apply the blueprint to test data for future use
```

Training Model

- A KNN model

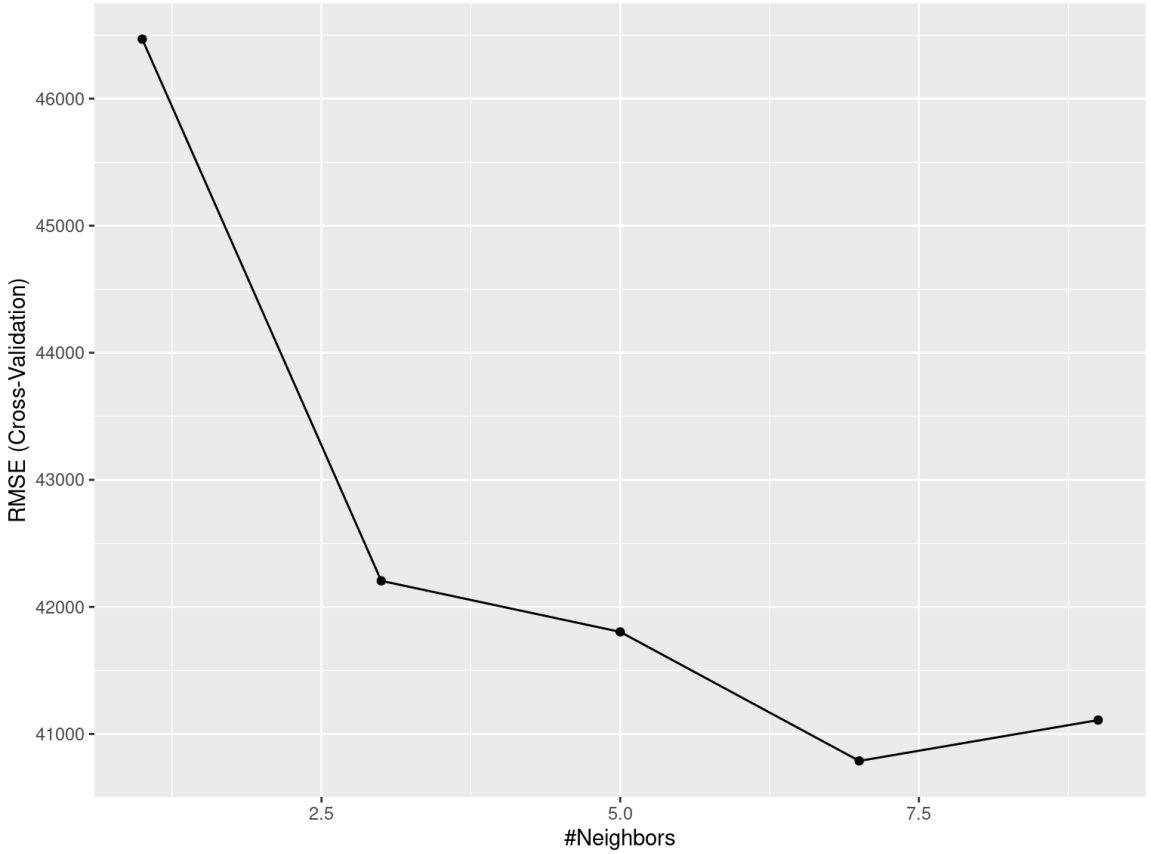
```
# perform CV to tune K

set.seed(042324)

cv_specs <- trainControl(method = "cv", number = 5)   # 5-fold CV (1 repeat)

k_grid <- expand.grid(k = seq(1, 10, by = 2))

knn_fit <- train(blueprint,
                 data = ames_train,
                 method = "knn",
                 trControl = cv_specs,
                 tuneGrid = k_grid,
                 metric = "RMSE")

knn_fit
```

```
## k-Nearest Neighbors
##
## 706 samples
##  19 predictor
##
## Recipe steps: nzv, impute_mean, integer, center, scale, other, dummy
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 565, 564, 566, 565, 564
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared   MAE
##   1  46468.03  0.7073754  26663.25
##   3  42205.11  0.7615049  24057.37
##   5  41803.20  0.7672474  23890.36
##   7  40788.08  0.7880202  23564.76
##   9  41109.35  0.7888861  23708.25
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

Training Model

- A KNN model

```
ggplot(knn_fit)
```



Training Model

- A linear regression model

```
set.seed(042324)

lm_fit <- train(blueprint,
                  data = ames_train,
                  method = "lm",
                  trControl = cv_specs,
                  metric = "RMSE")

lm_fit
```

```
## Linear Regression
##
## 706 samples
##  19 predictor
##
## Recipe steps: nzv, impute_mean, integer, center, scale, other, dummy
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 565, 564, 566, 565, 564
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   42023.63  0.7710352  24446.12
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

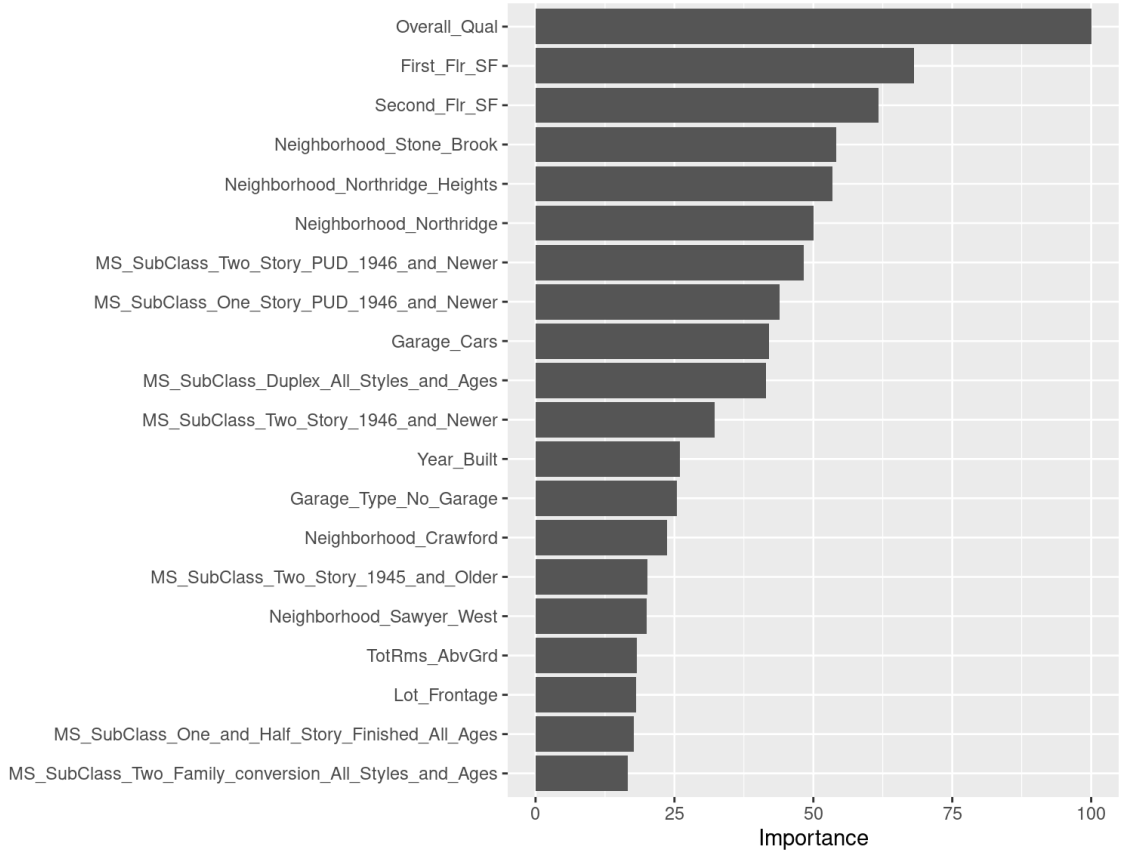
Final Model and Test Set Error

```
# refit the final/optimal model using ALL modified training data, and obtain estimate of prediction error from modified test data

final_model <- lm(Sale_Price ~ ., data = baked_train)    # build final model

final_preds <- predict(final_model, newdata = baked_test)   # obtain predictions on test data

sqrt(mean((baked_test$Sale_Price - final_preds)^2))    # calculate test set RMSE
```

```
## [1] 33947.67
```

Variable Importance

```
# variable importance

library(vip)

vip(object = lm_fit,          # CV object
    num_features = 20,    # maximum number of predictors to show importance for
    method = "model")           # model-specific VI scores
```



Your Turn!!!

You will work with a dataset containing information on employee attrition. Please load the dataset using the code below.

```
attrition <- readRDS("attrition.rds")
```

Objective: The task is to predict `Attrition` (Yes/No) using the rest of the variables in the data (predictors/features).

- **Step 1**: Investigate the dataset
  - What are the types of features? - categorical or numeric
  - If categorical, are they ordinal or nominal? If ordinal, are their levels in appropriate order? You can use the `levels` function to check the ordering.
  - Are there any features with missing entries?
  - Are there any zv/nzv features?
- **Step 2**: Split the data into training and test sets (70-30 split)
- **Step 3**: Perform required data preprocessing and create the blueprint. If using `step_dummy()`, set `one_hot = FALSE`.
- **Step 4**: Implement 5-fold CV (1 repeat) to compare the performance of the following models. Use `metric = "Accuracy"`.
  - a logistic regression model (`method = "glm"` and `family = "binomial"`)
  - a KNN classifier with the optimal `K` chosen by CV (`method = "knn"`). Use a grid of `K` values $1, 2, \ldots, 10$.

What is the optimal `K` chosen? How do the models compare in terms of the CV accuracies?

- **Step 5**: Build your final optimal model. Obtain probability and class label predictions for the test set (use threshold of 0.5). Create the corresponding confusion matrix and report the test set accuracy. Also, create the ROC curve for the optimal model and report the AUC.