MA710: Data Mining

Professor Noah Giansiracusa

Bentley University, Department of Mathematical Sciences

Jan 25, 2021

Supervised learning

training set

$$X = \begin{pmatrix} 1.1 & 2.2 \\ 6.7 & 0.5 \\ 2.4 & 9.3 \\ 1.5 & 0.0 \\ 0.5 & 3.5 \\ 5.1 & 9.7 \\ 3.7 & 7.8 \end{pmatrix} \qquad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Target

test set

Categorical target variable (classification)

training set

$$X = \begin{pmatrix} 1.1 & 2.2 \\ 6.7 & 0.5 \\ 2.4 & 9.3 \\ 1.5 & 0.0 \\ 0.5 & 3.5 \\ 5.1 & 9.7 \\ 3.7 & 7.8 \end{pmatrix} \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

test set

Categorical target variable (classification)

training set

$$X = \begin{pmatrix} 1.1 & 2.2 \\ 6.7 & 0.5 \\ 2.4 & 9.3 \\ 1.5 & 0.0 \\ 0.5 & 3.5 \\ 5.1 & 9.7 \\ 3.7 & 7.8 \end{pmatrix} \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

test set

类型

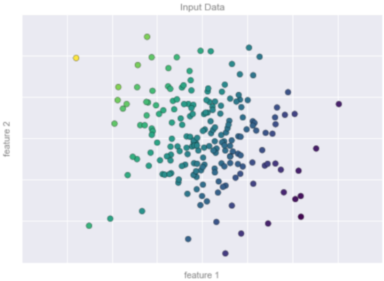
Categorical target variable (classification)

training set

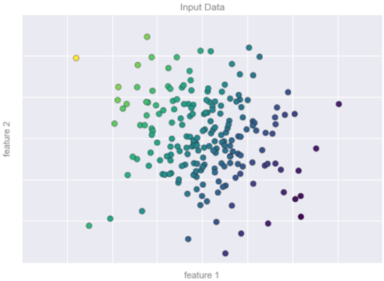
$$X = \begin{pmatrix} 1.1 & 2.2 \\ 6.7 & 0.5 \\ 2.4 & 9.3 \\ 1.5 & 0.0 \\ 0.5 & 3.5 \\ 5.1 & 9.7 \\ 3.7 & 7.8 \end{pmatrix} \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

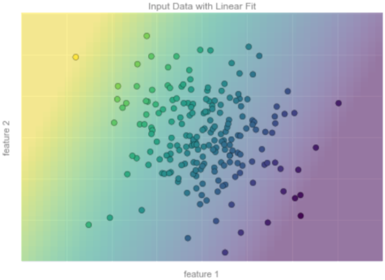
test set

Numerical target variable (regression/prediction)

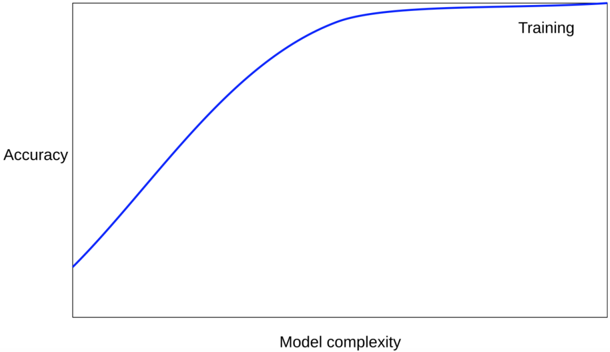


Numerical target variable (regression/prediction)

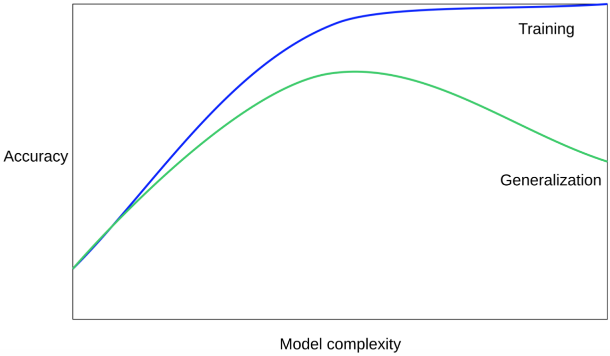




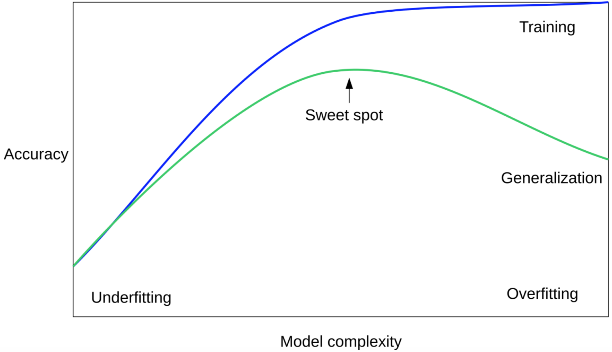
Overfitting and Underfitting

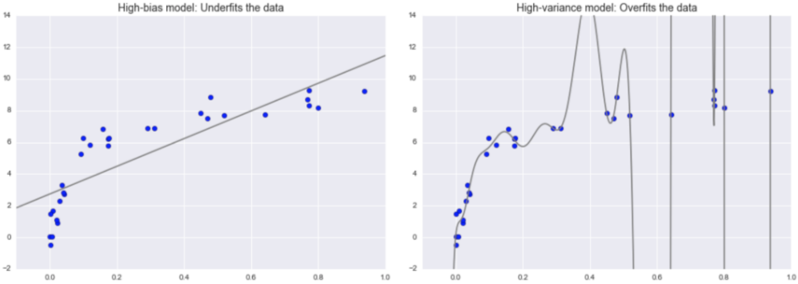


Overfitting and Underfitting



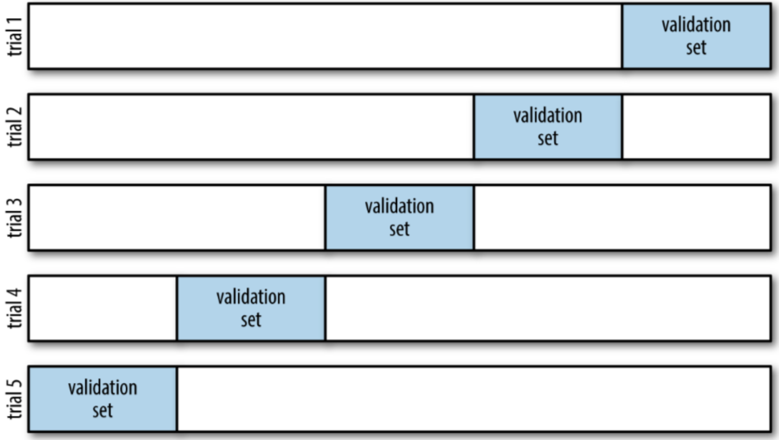
Overfitting and Underfitting

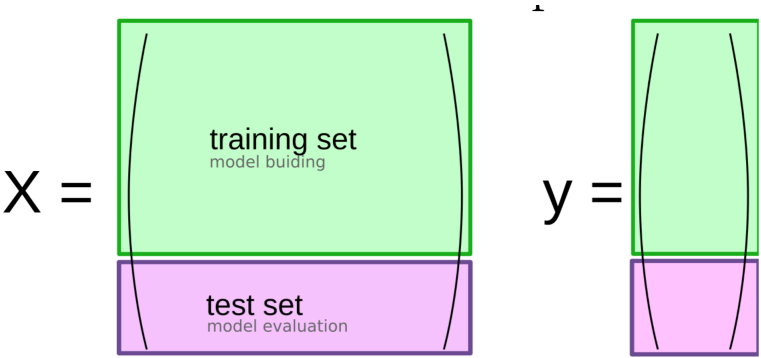


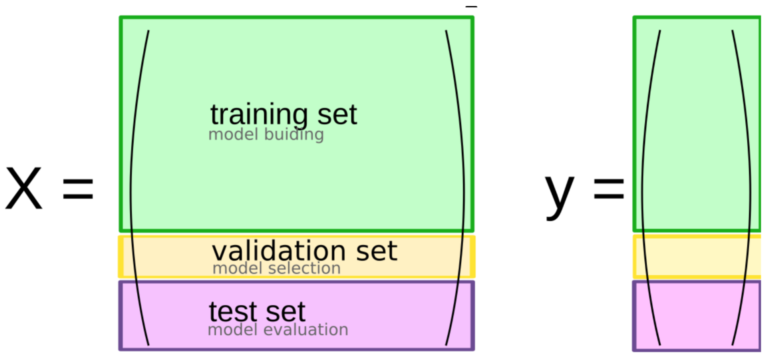

High-bias model: Underfits the data
High-variance model: Overfits the data
14
12
10
8
6
4
2
0
−2
0.0
0.2
0.4
0.6
0.8
1.0


Learning Curve Schematic
training score
validation score
High Variance
Good Fit
model score
training set size

Cross validation




X =
training set
model buiding
test set
model evaluation
y =


X =
training set
model buiding
validation set
model selection
test set
model evaluation
y =

Supervised learning methods

Supervised learning methods

Linear regression:

Supervised learning methods

Linear regression:

- numerical predictors, used for regression (numerical target)

Supervised learning methods

Linear regression:

- numerical predictors, used for regression (numerical target)
- computationally fast

Supervised learning methods

Linear regression:

- numerical predictors, used for regression (numerical target)
- computationally fast
- interpretable model (and indicates feature strengths)

Supervised learning methods

Linear regression:

- numerical predictors, used for regression (numerical target)
- computationally fast
- interpretable model (and indicates feature strengths)
- very rigid, doesn't work for modeling complicated relations in data

Logistic regression:

Logistic regression:

- numerical predictors, used for classification (categorical target) usually binary classification (two classes)

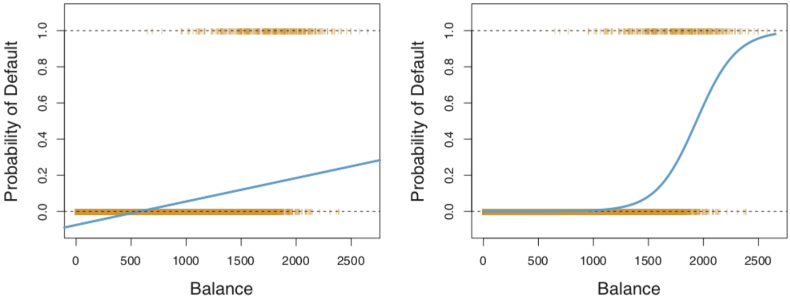
Logistic regression:

- numerical predictors, used for classification (categorical target) usually binary classification (two classes)
- still fairly fast and interpretable, often used as a first approach as a baseline to compare fancier methods against

Logistic regression:

- numerical predictors, used for classification (categorical target) usually binary classification (two classes)
- still fairly fast and interpretable, often used as a first approach as a baseline to compare fancier methods against

Example (linear regression on left, logistic regression on right):



k-nearest neighbor (abbreviated KNN or k-NN)

k-nearest neighbor (abbreviated KNN or k-NN)

- numerical predictors (and need to normalize or standardize them first), used for classification (multi-class) and regression

k-nearest neighbor (abbreviated KNN or k-NN)

- numerical predictors (and need to normalize or standardize them first), used for classification (multi-class) and regression
- non-parametric (and nearly impossible to interpret model) and lazy learner (computational time comes not from training but from evaluating)

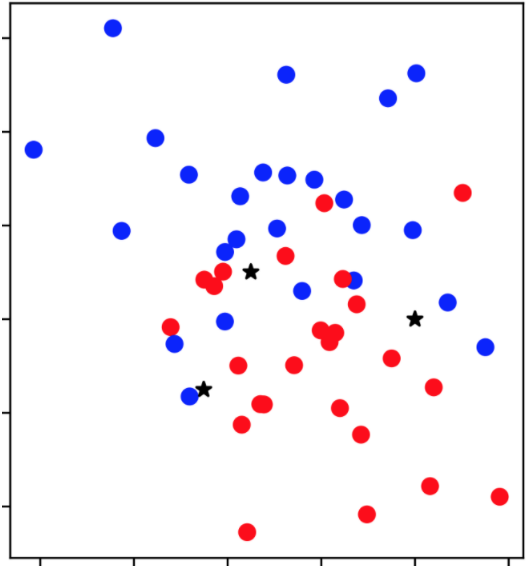
k-nearest neighbor (abbreviated KNN or k-NN)

- numerical predictors (and need to normalize or standardize them first), used for classification (multi-class) and regression
- non-parametric (and nearly impossible to interpret model) and lazy learner (computational time comes not from training but from evaluating)
- sensitive to local structure of data (this is both good and bad)

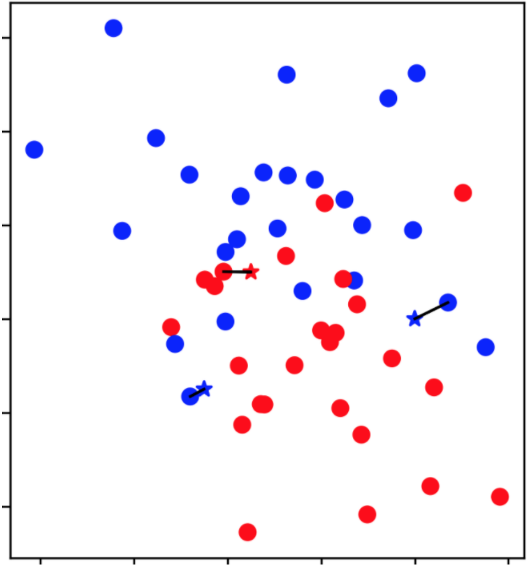
k-nearest neighbor (abbreviated KNN or k-NN)

- numerical predictors (and need to normalize or standardize them first), used for classification (multi-class) and regression
- non-parametric (and nearly impossible to interpret model) and lazy learner (computational time comes not from training but from evaluating)
- sensitive to local structure of data (this is both good and bad)
- depends on hyperparameter k, an integer

k-nearest neighbor (abbreviated KNN or k-NN)

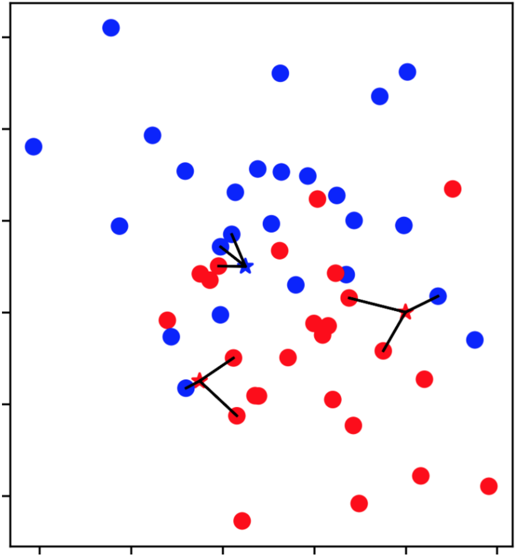


k-nearest neighbor (abbreviated KNN or k-NN)



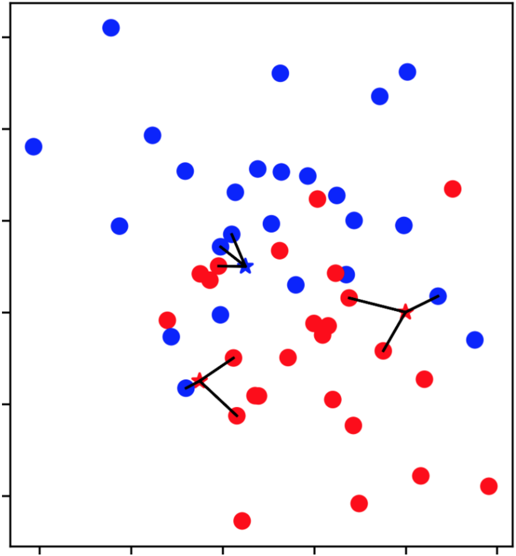
k=1

k-nearest neighbor (abbreviated KNN or k-NN)



k=3

k-nearest neighbor (abbreviated KNN or *k*-NN)



k=3

Smaller *k* value means more local-influenced and complex model (large *k* averages things out more, essentially blurring the data)

Unsupervised learning

This is any form of machine learning where the goal is to find patterns and structure in the feature matrix X without relying on a target vector y.

Unsupervised learning
This is any form of machine learning where the goal is to find patterns and structure in the feature matrix X without relying on a target vector y.

The most common examples are:

Unsupervised learning
This is any form of machine learning where the goal is to find patterns and structure in the feature matrix X without relying on a target vector y.

The most common examples are:

- Cluster analysis

Unsupervised learning
This is any form of machine learning where the goal is to find patterns and structure in the feature matrix X without relying on a target vector y.

The most common examples are:

- Cluster analysis
- Dimension reduction (such as PCA)

k-means clustering:

k-means clustering algorithm:

1. Start with k initial clusters (user chooses k).
2. At every step, each record is reassigned to the cluster with the "closest" centroid.
3. Recompute the centroids of clusters that lost or gained a record, and repeat Step 2.

k-means clustering:

k-means clustering algorithm:

1. Start with k initial clusters (user chooses k).
2. At every step, each record is reassigned to the cluster with the "closest" centroid.
3. Recompute the centroids of clusters that lost or gained a record, and repeat Step 2.

We usually *start* with a random assignment into clusters and *stop* when the "improvement" (measured by sum of distances of points in each cluster from the cluster centroid) is very small.

k-means clustering:

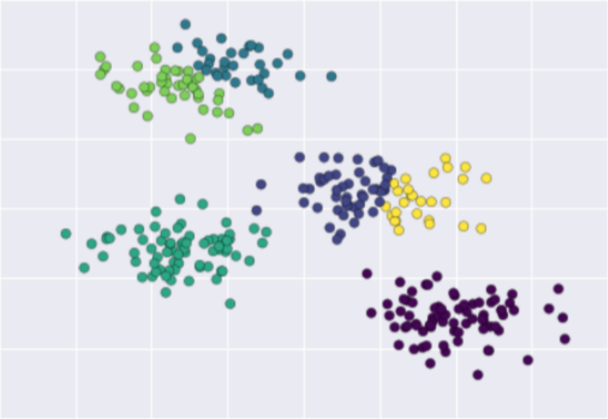
k-means clustering algorithm:

1. Start with k initial clusters (user chooses k).
2. At every step, each record is reassigned to the cluster with the "closest" centroid.
3. Recompute the centroids of clusters that lost or gained a record, and repeat Step 2.

We usually *start* with a random assignment into clusters and *stop* when the "improvement" (measured by sum of distances of points in each cluster from the cluster centroid) is very small.

How to choose k? Maybe contextual knowledge, or you can try to interpret the clusters, or you can try an *elbow chart* (such as overall average within-cluster distance), but often it is hard.

Here the number of k was chosen poorly:



Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.

Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.

Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.

Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.

Chapter 2.3
Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.
5. Determine the data mining task.

Chapter 2.3
Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.
5. Determine the data mining task.
6. Partition the data (for supervised tasks).

Chapter 2.3
Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.
5. Determine the data mining task.
6. Partition the data (for supervised tasks).
7. Choose the data mining techniques to be used.

Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.
5. Determine the data mining task.
6. Partition the data (for supervised tasks).
7. Choose the data mining techniques to be used.
8. Use algorithms to perform the task.

Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.
5. Determine the data mining task.
6. Partition the data (for supervised tasks).
7. Choose the data mining techniques to be used.
8. Use algorithms to perform the task.
9. Interpret the results of the algorithms.

Chapter 2.3

Steps in Data Mining

1. Develop an understanding of the purpose of the project.
2. Obtain the dataset to be used in the analysis.
3. Explore, clean, and preprocess the data.
4. Reduce the data dimension, if necessary.
5. Determine the data mining task.
6. Partition the data (for supervised tasks).
7. Choose the data mining techniques to be used.
8. Use algorithms to perform the task.
9. Interpret the results of the algorithms.
10. Deploy the model.

Chapter 2.4

```
> housing.df <- read.csv("DMBA-R-datasets/WestRoxbury.csv", header = TRUE)
> dim(housing.df)  # find the dimension of data frame
[1] 5802   14
> head(housing.df)  # show the first six rows
  TOTAL.VALUE  TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS FULL.BATH HALF.BATH KITCHEN FIREPLACE REMODEL
1       344.2 4330     9965     1880       2436        1352      2     6        3         1         1       1         0    None
2       412.6 5190     6590     1945       3108        1976      2    10        4         2         1       1         0  Recent
3       330.1 4152     7500     1890       2294        1371      2     8        4         1         1       1         0    None
4       498.6 6272    13773     1957       5032        2608      1     9        5         1         1       1         1    None
5       331.5 4170     5000     1910       2370        1438      2     7        3         2         0       1         0    None
6       337.4 4244     5142     1950       2124        1060      1     6        3         1         0       1         1     Old
```

Chapter 2.4

```
> housing.df <- read.csv("DMBA-R-datasets/WestRoxbury.csv", header = TRUE)
> dim(housing.df)  # find the dimension of data frame
[1] 5802   14
> head(housing.df)  # show the first six rows
  TOTAL.VALUE  TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS FULL.BATH HALF.BATH KITCHEN FIREPLACE REMODEL
1       344.2 4330     9965     1880       2436        1352      2     6        3         1         1       1         0    None
2       412.6 5190     6590     1945       3108        1976      2    10        4         2         1       1         0  Recent
3       330.1 4152     7500     1890       2294        1371      2     8        4         1         1       1         0    None
4       498.6 6272    13773     1957       5032        2608      1     9        5         1         1       1         1    None
5       331.5 4170     5000     1910       2370        1438      2     7        3         2         0       1         0    None
6       337.4 4244     5142     1950       2124        1060      1     6        3         1         0       1         1     Old
```

Table 2.7 illustrates some methods for dealing with missing values.

Chapter 2.4

```
> housing.df <- read.csv("DMBA-R-datasets/WestRoxbury.csv", header = TRUE)
> dim(housing.df)  # find the dimension of data frame
[1] 5802   14
> head(housing.df)  # show the first six rows
  TOTAL.VALUE  TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS FULL.BATH HALF.BATH KITCHEN FIREPLACE REMODEL
1       344.2 4330     9965     1880       2436        1352      2     6        3         1         1       1         0    None
2       412.6 5190     6590     1945       3108        1976      2    10        4         2         1       1         0  Recent
3       330.1 4152     7500     1890       2294        1371      2     8        4         1         1       1         0    None
4       498.6 6272    13773     1957       5032        2608      1     9        5         1         1       1         1    None
5       331.5 4170     5000     1910       2370        1438      2     7        3         2         0       1         0    None
6       337.4 4244     5142     1950       2124        1060      1     6        3         1         0       1         1     Old
```

Table 2.7 illustrates some methods for dealing with missing values.

```
> class(housing.df$REMODEL)
[1] "factor"
> levels(housing.df$REMODEL)
[1] "None"   "Old"    "Recent"
```

Chapter 2.4

```
> housing.df <- read.csv("DMBA-R-datasets/WestRoxbury.csv", header = TRUE)
> dim(housing.df)  # find the dimension of data frame
[1] 5802   14
> head(housing.df)  # show the first six rows
  TOTAL.VALUE  TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS FULL.BATH HALF.BATH KITCHEN FIREPLACE REMODEL
1       344.2 4330     9965     1880       2436        1352      2     6        3         1         1       1         0    None
2       412.6 5190     6590     1945       3108        1976      2    10        4         2         1       1         0  Recent
3       330.1 4152     7500     1890       2294        1371      2     8        4         1         1       1         0    None
4       498.6 6272    13773     1957       5032        2608      1     9        5         1         1       1         1    None
5       331.5 4170     5000     1910       2370        1438      2     7        3         2         0       1         0    None
6       337.4 4244     5142     1950       2124        1060      1     6        3         1         0       1         1     Old
```

Table 2.7 illustrates some methods for dealing with missing values.

```
> class(housing.df$REMODEL)
[1] "factor"
> levels(housing.df$REMODEL)
[1] "None"    "Old"     "Recent"
```

Often we'll need to convert categorical variables into binary dummy variables (one-hot encoding):

```
> library(fastDummies)
> new.df <- dummy_cols(housing.df,remove_first_dummy=T,remove_selected_columns=T)
> head(new.df)
  TOTAL.VALUE  TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS FULL.BATH HALF.BATH KITCHEN FIREPLACE REMODEL_Old REMODEL_Recent
1       344.2 4330     9965     1880       2436        1352      2     6        3         1         1       1         0           0              0
2       412.6 5190     6590     1945       3108        1976      2    10        4         2         1       1         0           0              1
3       330.1 4152     7500     1890       2294        1371      2     8        4         1         1       1         0           0              0
4       498.6 6272    13773     1957       5032        2608      1     9        5         1         1       1         1           0              0
5       331.5 4170     5000     1910       2370        1438      2     7        3         2         0       1         0           0              0
6       337.4 4244     5142     1950       2124        1060      1     6        3         1         0       1         1           1              0
```

Chapter 2.4

Why use dummy variables instead of encoding categories numerically?

Chapter 2.4

Why use dummy variables instead of encoding categories numerically?

For *ordered* categories it is better to just encode numerically—for example, survey data might have values "bad", "ok", "good", "great", and you could just encode these as 1,2,3,4.

Why use dummy variables instead of encoding categories numerically?

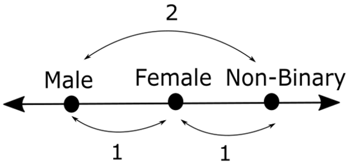
For *ordered* categories it is better to just encode numerically—for example, survey data might have values "bad", "ok", "good", "great", and you could just encode these as 1,2,3,4.

But don't do this for unordered categories—for example, consider a gender variable with values "Male", "Female", "Non-Binary":

Why use dummy variables instead of encoding categories numerically?

For *ordered* categories it is better to just encode numerically—for example, survey data might have values “bad”, “ok”, “good”, “great”, and you could just encode these as 1,2,3,4.

But don’t do this for unordered categories—for example, consider a gender variable with values “Male”, “Female”, “Non-Binary”:



Putting the values in a single dimension forces some to be closer than others.

Chapter 2.4

For a categorical variable with k possible values, should we use k dummy variables or $k-1$?

For a categorical variable with k possible values, should we use k dummy variables or $k - 1$?

For linear and logistic regression use $k - 1$, otherwise you have multicollinearity problems.

For a categorical variable with k possible values, should we use k dummy variables or $k-1$?

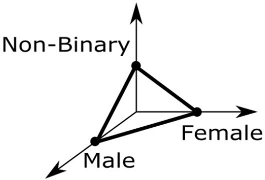
For linear and logistic regression use $k-1$, otherwise you have multicollinearity problems.

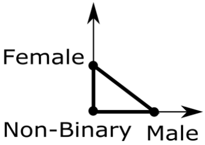
But for methods where distance is crucial (such as KNN), use all k otherwise you will distort the distances:

For a categorical variable with k possible values, should we use k dummy variables or $k-1$?

For linear and logistic regression use $k-1$, otherwise you have multicollinearity problems.

But for methods where distance is crucial (such as KNN), use all k otherwise you will distort the distances:





Here putting the values in a single dimension forces some to be closer than others.

Chapter 2.4

For many data mining procedures you'll want to first **normalize** your data (for each variable, subtract the off the minimum value then divide by the maximum value, so the range turns into $[0, 1]$)

Chapter 2.4

For many data mining procedures you'll want to first **normalize** your data (for each variable, subtract the off the minimum value then divide by the maximum value, so the range turns into $[0, 1]$) or **standardize** your data (for each variable, subtract off the mean then divide by the standard deviation—only do this when you think your data is approximately Gaussian).

Chapter 2.4

For many data mining procedures you'll want to first **normalize** your data (for each variable, subtract the off the minimum value then divide by the maximum value, so the range turns into $[0, 1]$) or **standardize** your data (for each variable, subtract off the mean then divide by the standard deviation—only do this when you think your data is approximately Gaussian).

In R, you normalize with the function `rescale()` from the `scales` package, and you standardize with the function `scale()`.

Chapter 2.4

For many data mining procedures you'll want to first **normalize** your data (for each variable, subtract the off the minimum value then divide by the maximum value, so the range turns into $[0, 1]$) or **standardize** your data (for each variable, subtract off the mean then divide by the standard deviation—only do this when you think your data is approximately Gaussian).

In R, you normalize with the function `rescale()` from the `scales` package, and you standardize with the function `scale()`.

When normalizing/standardizing, you should compute the translation and scaling factors from the **training** data alone then apply them to the other data sets (to avoid "leakage").

Chapter 2.5

Table 2.9 (Table in 2.10 in Python version) shows how to partition data into train, validation, test sets.

Chapter 2.5

Table 2.9 (Table in 2.10 in Python version) shows how to partition data into train, validation, test sets.

Chapter 2.6

After we preprocess then partition, we can build a model (supervised learning).

Chapter 2.5

Table 2.9 (Table in 2.10 in Python version) shows how to partition data into train, validation, test sets.

Chapter 2.6

After we preprocess then partition, we can build a model (supervised learning). Here's an example with linear regression:

Chapter 2.6

Training data:

```
reg <- lm(TOTAL_VALUE ~ ., data = housing.df, subset = train.rows)
tr.res <- data.frame(train.data$TOTAL_VALUE, reg$fitted.values, reg$residuals)
head(tr.res)
```

Partial Output

```
> head(tr.res)
     train.data.TOTAL_VALUE reg.fitted.values reg.residuals
3651                  371.6          371.5818   0.018235205
359                   299.4          299.4014  -0.001431463
1195                  294.5          294.4762   0.023835688
1024                  249.4          249.4029  -0.002874472
3984                  505.5          505.5246  -0.024612237
2227                  410.5          410.5323  -0.032339156
```

Validation data:

```
pred <- predict(reg, newdata = valid.data)
vl.res <- data.frame(valid.data$TOTAL_VALUE, pred, residuals =
          valid.data$TOTAL_VALUE - pred)
head(vl.res)
```

Partial Output

```
> head(vl.res)
   valid.data.TOTAL_VALUE     pred    residuals
1                   344.2 344.2388 -0.038842075
14                  575.0 575.0025 -0.002509638
16                  298.2 298.2107 -0.010697442
17                  313.1 313.0764  0.023567596
18                  344.9 344.8719  0.028111825
19                  330.7 330.7222 -0.022234883
```

Chapter 2.6 and 5.2

Performance metrics:

```
library(forecast)
# compute accuracy on training set
accuracy(reg$fitted.values, train.data$TOTAL_VALUE)

# compute accuracy on prediction set
pred <- predict(reg, newdata = valid.data)
accuracy(pred, valid.data$TOTAL_VALUE)
```

Partial Output

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE         MAE          MPE       MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

Chapter 2.6 and 5.2

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE       MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE       MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error

RMSE: Root Mean Squared Error

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE        MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error
RMSE: Root Mean Squared Error
MAE: Mean Absolute Error

Chapter 2.6 and 5.2

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE       MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error

RMSE: Root Mean Squared Error

MAE: Mean Absolute Error

MPE: Mean Percentage Error

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE       MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error
RMSE: Root Mean Squared Error
MAE: Mean Absolute Error
MPE: Mean Percentage Error
MAPE: Mean Absolute Percentage Error

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE        MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error
RMSE: Root Mean Squared Error
MAE: Mean Absolute Error
MPE: Mean Percentage Error
MAPE: Mean Absolute Percentage Error

To see if outliers are playing a role, look at the distribution of the residuals.

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                   ME       RMSE        MAE          MPE       MAPE
Test set 1.388101e-16 0.02268016 0.01956465 5.193036e-06 0.00528389

> accuracy(pred, valid.data$TOTAL_VALUE)
               ME     RMSE      MAE      MPE     MAPE
Test set 90.86934 161.5043 118.6455 15.14207 24.45668
```

ME: Mean Error
RMSE: Root Mean Squared Error
MAE: Mean Absolute Error
MPE: Mean Percentage Error
MAPE: Mean Absolute Percentage Error

To see if outliers are playing a role, look at the distribution of the residuals.

Always good to compare a model against the *naive baseline/benchmark*, which for regression is the average value of the target variable across the training data.