ITP 449

k-NN
Classification and Regression

Lecture 10

Supervised learning

Determine a function that maps an input to an output based on example input-output pairs

Estimation

Linear Regression
Logistic Regression

Classification

k-Nearest Neighbors (k-NN)
Support Vector Machines (SVMs)
Decision Trees
Random Forests
Neural Networks

Supervised learning

Target variable: Predict the value or category of a target (response) variable based on a group of predictor variables (features, influencers)

Linear regression Used to predict *continuous numeric* outcomes

Logistic regression *Classification* algorithm

k-Nearest Neighbors (k-NN) Classification algorithm that classifies data into two or more *categories*

Support Vector Machines (SVM) Classification algorithm that is used in *image* and *face* detection

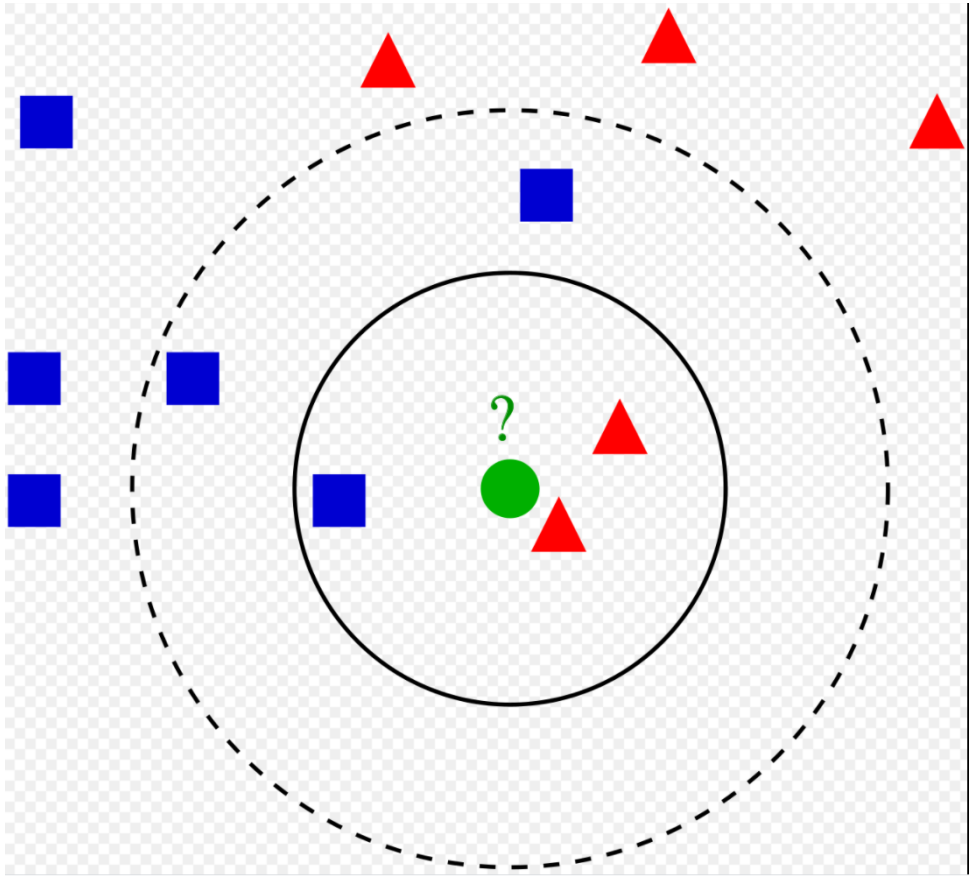
Tree-Based algorithms Tree-based algorithms such as decision trees and Random Forests are used to solve both *classification* and *regression* problems

Näive Bayes Uses the mathematical model of probability to solve classification problems

k-NN for Classification

k Nearest Neighbors

You are likely to be similar to your "neighbors". If I know your neighbors' attributes and response, I can predict your response, given your attributes.



The test sample (green dot) should be classified either to blue squares or to red triangles.

- *If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle.*
- *If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).*

Age	Gender	Income	…	…	Outcome
35	M	$ 40,000	…	…	Buyer
40	F	$ 50,000	…	…	Buyer
30	M	$ 25,000	…	…	Non-buyer
35	M	$ 40,000	…	…	Buyer
40	F	$ 50,000	…	…	Non-buyer
30	M	$ 25,000	…	…	Non-buyer
35	M	$ 40,000	…	…	Buyer
40	F	$ 50,000	…	…	Non-buyer
30	M	$ 25,000	…	…	Non-buyer
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…

…	…	Outcome
…	…	Buyer
…	…	Buyer
…	…	Non-buyer
…	…	Buyer
…	…	Non-buyer
…	…	Non-buyer
…	…	Buyer
…	…	Non-buyer
…	…	Non-buyer
…	…	…
…	…	…
…	…	…
…	…	…
…	…	…

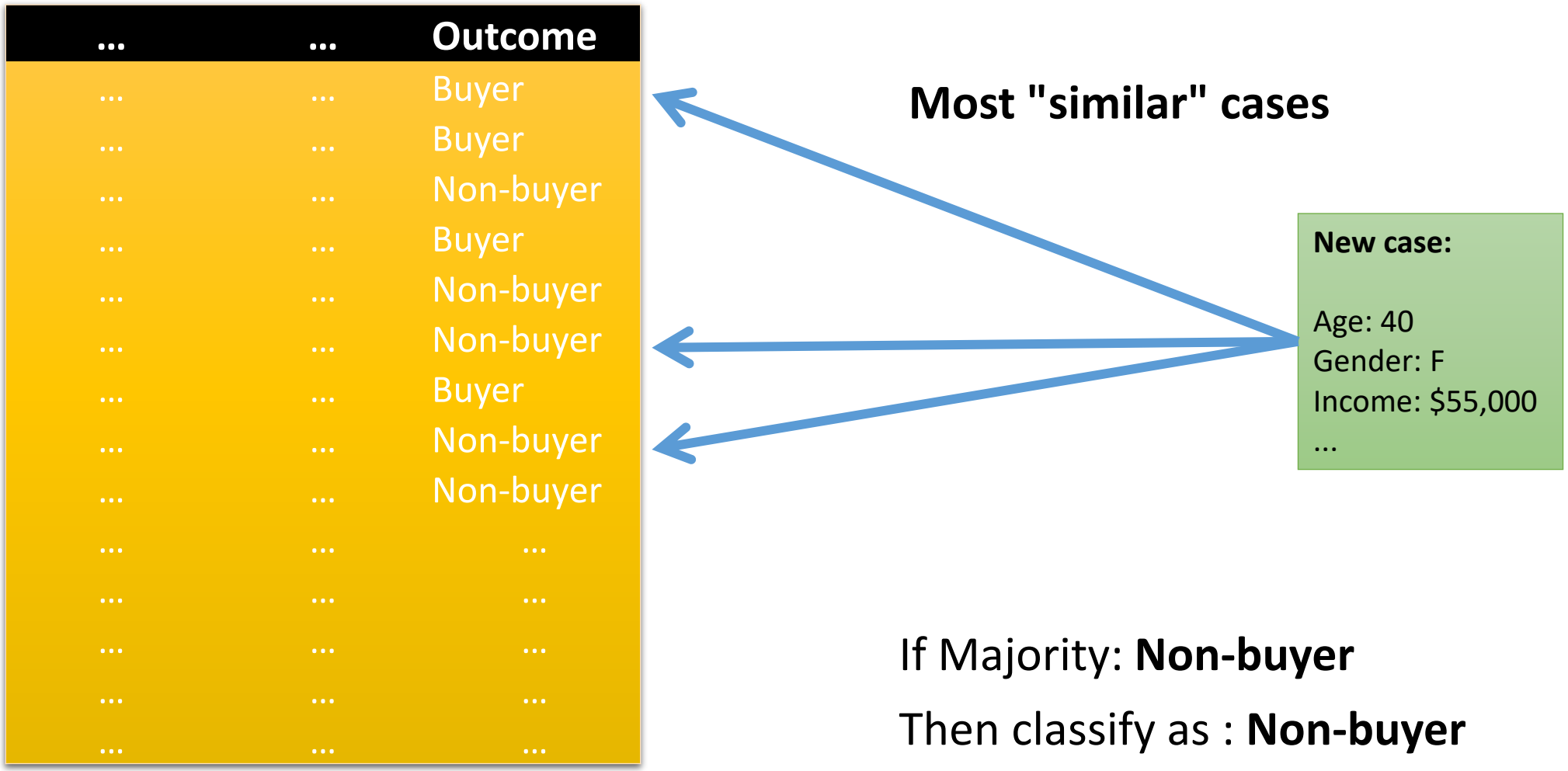
New case:

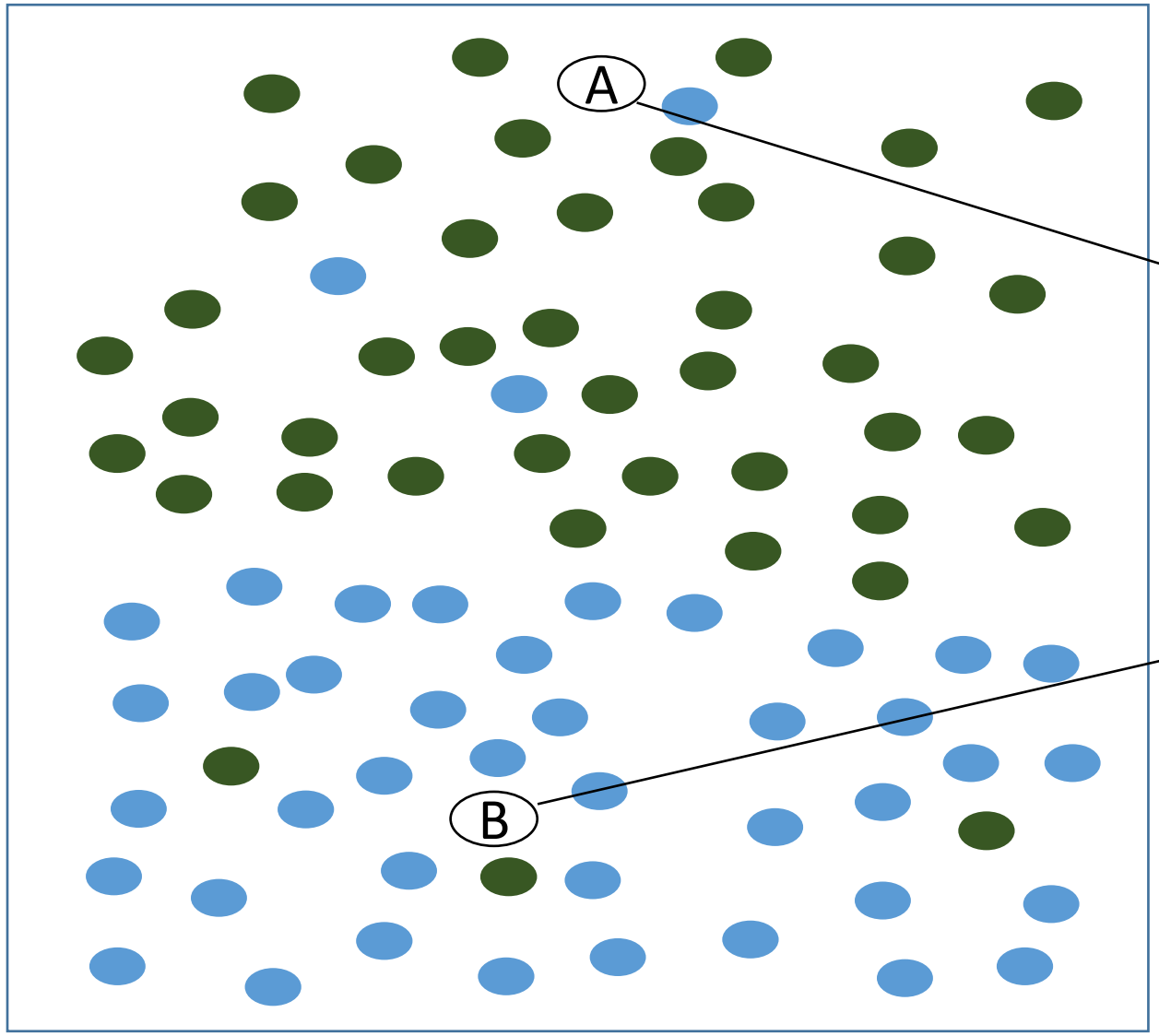
Age: 40
Gender: F
Income: $55,000
…

Buyer or Non-buyer?

Classify a case:

Based on k most "similar" cases in the historical data


... ... Outcome
... ... Buyer
... ... Buyer
... ... Non-buyer
... ... Buyer
... ... Non-buyer
... ... Non-buyer
... ... Buyer
... ... Non-buyer
... ... Non-buyer
... ...
... ...
... ...
... ...
... ...
Most "similar" cases
New case:
Age: 40
Gender: F
Income: $55,000
...
If Majority: Non-buyer
Then classify as : Non-buyer



Predominantly “Red” neighbors -- classify as “Red”

Predominantly “Blue” neighbors -- classify as “Blue”

kNN

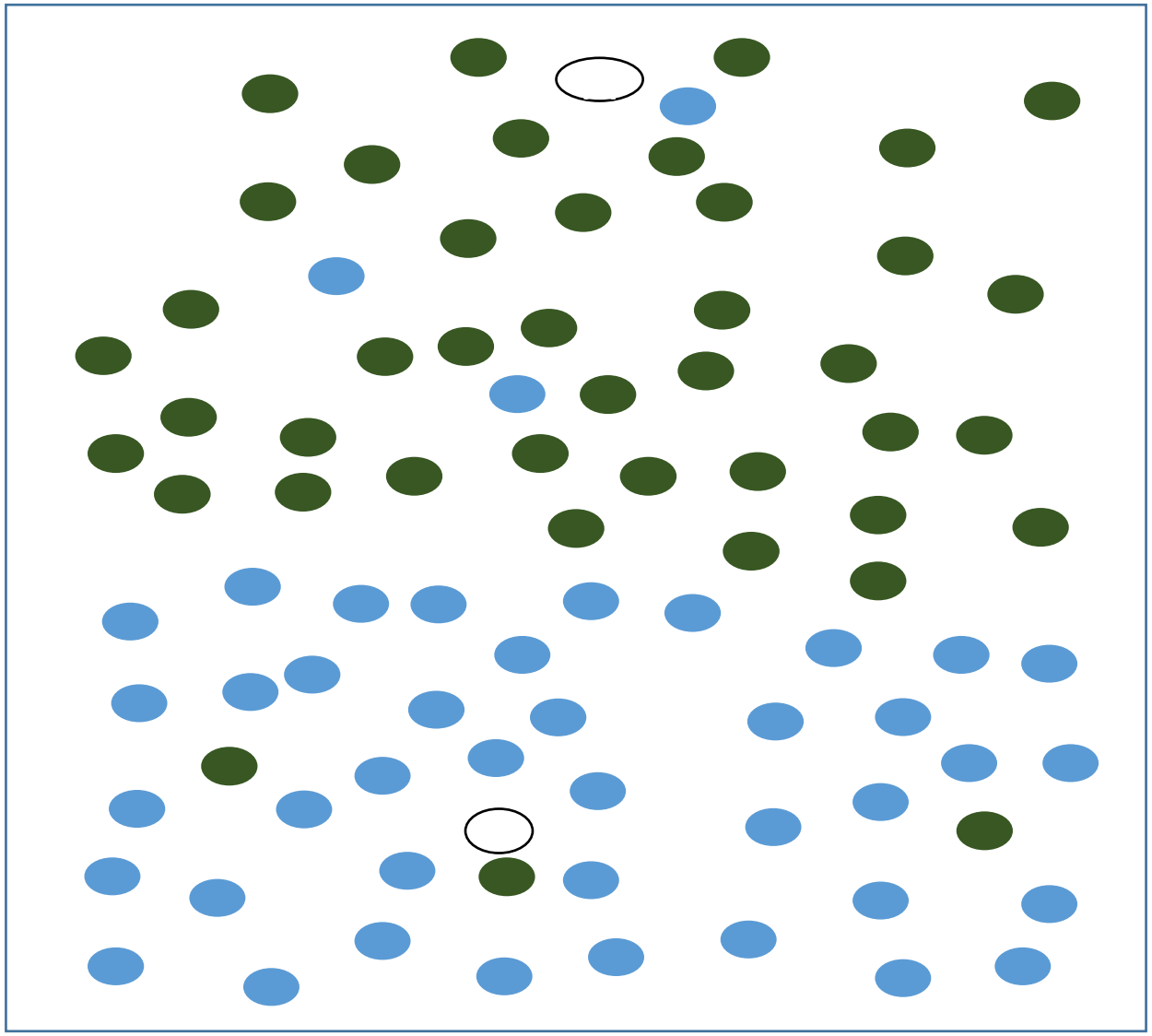
- The figure shows the political affiliations of a set of people in a certain neighborhood. Suppose also that the locations of the dots also represent the actual locations of the households.

- Now, we are asked to guess for the two households that are neither red nor blue and marked as “A” and “B”, what our best guess would be for how they would vote. What would your guess be for the “A”? How about for “B”?

- It would be safe to say that the top half represents a red neighborhood and the bottom half a blue one, and we would therefore guess accordingly for each of the two unknown cases “A” and “B”.

- The k-Nearest Neighbor approach simply formalizes this intuition. Given a set of cases whose classifications are known and given a new case to be classified, we simply look at the “neighbors” of the new case within the known cases and then classify the new case depending on the majority class of the neighbors.

- In the above example the neighbors of “A” are predominantly red and those of B, predominantly blue.



How many neighbors to consider?

Small
Latching
onto noise

Too little
specificity

Number of neighbors: k

Two important questions arise:

1. How many neighbors to consider?
2. In the present example, the idea of “neighbor” seems clear. How about in a general case – what would “neighbor” mean for some of the data sets we have looked at earlier?

Let us look at the easy question (the first one) first:

- If we consider too few neighbors, then we might end up classifying a case based on accidental occurrences that do not reflect the real situation. In the example, if we consider only one neighbor, we can see that in either case, the closest point is of the opposite color of the general pattern of the region. So too small a value for the number of neighbors could end up with noise determining our classification.
- On the other hand, too high a value would cause a loss of sharpness. In the above case, suppose we considered a value like 50 then we would classify every new case based simply on the overall majority, without considering “local” characteristics at all and this would be the equivalent of using the Naïve Rule.

We need to strike a balance:

We will build models with different values for k and choose the one that performs best on the validation partition

Income	# Cars	Ownership of Boat
$ 95,000.00	1	No
$120,000.00	2	Yes
$ 30,000.00	2	No
$ 60,000.00	1	Yes
$ 80,000.00	0	Yes
$ 75,000.00	2	No
$120,000.00	1	No
$ 85,000.00	1	Yes
$110,000.00	3	Yes
$200,000.00	2	Yes
$130,000.00	1	
$ 78,000.00	2	

Income: $85,000
 # Cars: 2

Neighbors?

Numeric attributes

- Suppose our data is not inherently geographical, how would the concept of *neighbor* apply?

- For example, suppose we have data like that shown on the left and have a person whose income is $85,000 and who owns 2 cars. How do we find the *neighbors* of this person?

- We can use the notion of “distance” that we talked about in the previous lectures.

- That is, we could consider each case as a set of x-y co-ordinates and then calculate the so-called Euclidian distance that results from applying Pythagoras’ theorem.

Income	# Cars	Ownership of Boat
$ 95,000.00	1	No

(85000, 2)

$$\sqrt{(95000-85000)^2+(1-2)^2}$$

Applying the formula for Euclidian distance we get the above.

Income	# Cars	Ownership of Boat
$ 95,000.00	1	No
$120,000.00	2	Yes
$ 30,000.00	2	No
$ 60,000.00	1	Yes
$ 80,000.00	0	Yes
$ 75,000.00	2	No
$120,000.00	1	No
$ 85,000.00	1	Yes
$110,000.00	3	Yes
$200,000.00	2	Yes
$130,000.00	1	
$ 78,000.00	2	

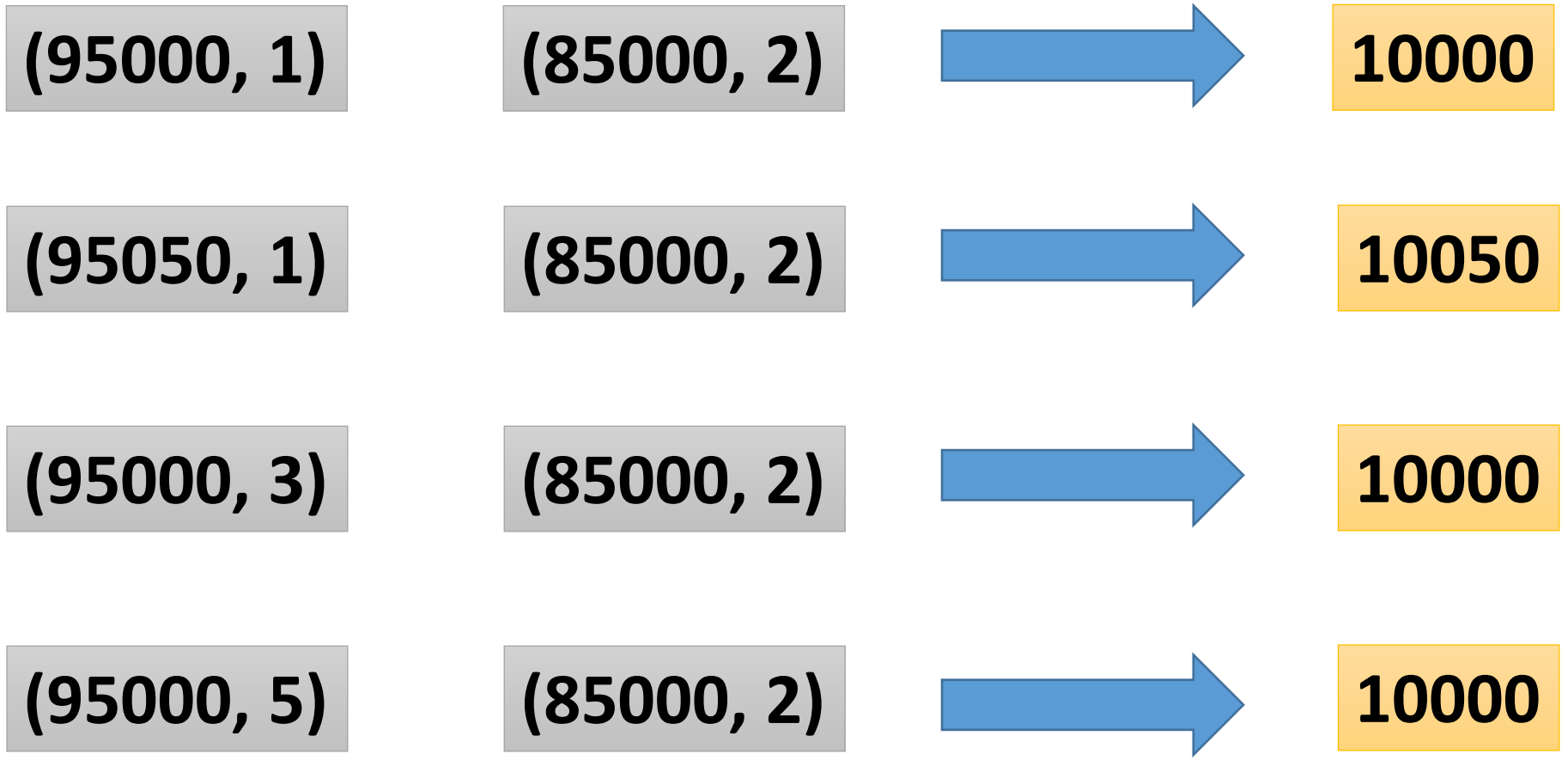
kNN: Classification

Categorical target

Numerical predictors

Prediction also possible … discuss later

Since kNN relies on the notion of distance, the predictor attributes must be numeric. Categorical predictors will not work.
kNN can be used for both classification and regression. In this lecture we will look at its use for classification. When used for classification, kNN will treat the target attribute as categorical, even if it is encoded as a number.



Distance is insensitive to number of cars …

Because of the relative magnitudes of the two variables, one dominates the distance measure.

What can we do?

We can use a ***normalizer*** or a ***standard scaler.*** That way no attribute will dominate the distance measure.

Distance  **Numerical Predictors**

What to do about categorical attributes?

Since distance calculations require numerical predictors, what can we do if we have categorical attributes?

We can use the dummy attribute technique.

kNN for classification needs:

Categorical target

If target attributes not categorical:

- Consider “binning” or
- Use kNN for regression instead

Numerical predictor

If any predictor attribute is categorical – convert to dummy attributes

Binning

Original
60
14
81
1
164
107
181
17
114
119
171
58
238
267
35

Bins	
0	A
50	B
100	C
150	D
200	E
250	F

Binned
B
A
B
A
D
C
D
A
C
C
D
B
E
F
A

Binning

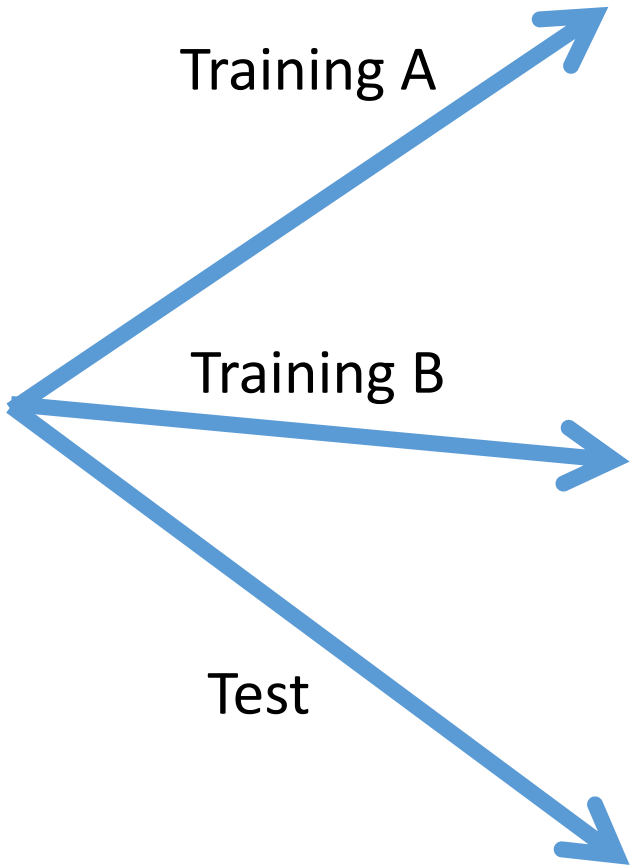
When we have a target attribute that is numeric, but we want it to be categorical so that we can use a classification method, we can resort to binning, which is just a way to say that we will create a discrete number of bins into which we will slot the values.

In the previous slide, we have broken up the whole range of the variable into six bins of interval size 50 units. The resulting values are shown on the right.

Income	Lot Size	Ownership
85.5	16.8	owner
47.4	16.4	non-owner
52.8	20.8	non-owner
49.2	17.6	non-owner
82.8	22.4	owner
63	14.8	non-owner
66	18.4	non-owner
69	20	owner
64.8	17.2	non-owner
64.8	21.6	owner
87	23.6	owner
43.2	20.4	non-owner
59.4	16	non-owner
84	17.6	non-owner
60	18.4	owner
110.1	19.2	owner
51	22	owner
81	20	owner
108	17.6	owner
75	19.6	non-owner
61.5	20.8	owner
33	18.8	non-owner
93	20.8	owner
51	14	non-owner

Norm_income	Norm_lot_size	Ownership
0.862	-0.885	owner
-1.063	-1.050	non-owner
-0.790	0.762	non-owner
-0.972	-0.556	non-owner
0.726	1.421	owner
-0.275	-1.709	non-owner
-0.123	-0.226	non-owner
0.028	0.432	owner
-0.184	-0.721	non-owner
-0.184	1.091	owner
0.938	1.915	owner
-1.275	0.597	non-owner
-0.457	-1.215	non-owner
0.786	-0.556	non-owner
-0.426	-0.226	owner
2.105	0.103	owner
-0.881	1.256	owner
0.635	0.432	owner
1.999	-0.556	owner
0.332	0.268	non-owner
-0.351	0.762	owner
-1.790	-0.062	non-owner
1.241	0.762	owner
-0.881	-2.038	non-owner

Norm_income	Norm_lot_size	Ownership
0.862	-0.885	owner
-1.063	-1.050	non-owner
-0.790	0.762	non-owner
-0.972	-0.556	non-owner
0.726	1.421	owner
-0.275	-1.709	non-owner
-0.123	-0.226	non-owner
0.028	0.432	owner
-0.184	-0.721	non-owner
-0.184	1.091	owner
0.938	1.915	owner
-1.275	0.597	non-owner
-0.457	-1.215	non-owner
0.786	-0.556	non-owner
-0.426	-0.226	owner
2.105	0.103	owner
-0.881	1.256	owner
0.635	0.432	owner
1.999	-0.556	owner
0.332	0.268	non-owner
-0.351	0.762	owner
-1.790	-0.062	non-owner
1.241	0.762	owner
-0.881	-2.038	non-owner



Norm_income	Norm_lot_size	Ownership
0.862	-0.885	owner
-1.063	-1.050	non-owner
-0.790	0.762	non-owner
-0.972	-0.556	non-owner
0.726	1.421	owner
-0.275	-1.709	non-owner
-0.123	-0.226	non-owner
0.028	0.432	owner
-0.184	-0.721	non-owner
-0.184	1.091	owner
0.938	1.915	owner
-1.275	0.597	non-owner

Norm_income	Norm_lot_size	Ownership
-0.457	-1.215	non-owner
0.786	-0.556	non-owner
-0.426	-0.226	owner
2.105	0.103	owner
-0.881	1.256	owner
0.635	0.432	owner

Norm_income	Norm_lot_size	Ownership
1.999	-0.556	owner
0.332	0.268	non-owner
-0.351	0.762	owner
-1.790	-0.062	non-owner
1.241	0.762	owner
-0.881	-2.038	non-owner

Unlike almost any other predictive analytics technique, kNN requires three partitions. It uses two partitions just to build the model and uses the third one for testing.

We call the first two partitions as Training A and Training B partitions. The third is called Test.

In this example, we have too little data to partition and build a model. However, we do the steps for demonstration.

Training A

	Norm_income	Norm_lot_size	Ownership
	0.862	-0.885	owner
3	-1.063	-1.050	non-owner
	-0.790	0.762	non-owner
	-0.972	-0.556	non-owner
	0.726	1.421	owner
1	-0.275	-1.709	non-owner
	-0.123	-0.226	non-owner
	0.028	0.432	owner
2	-0.184	-0.721	non-owner
	-0.184	1.091	owner
	0.938	1.915	owner
	-1.275	0.597	non-owner

Training B

Norm_income	Norm_lot_size	Ownership
-0.457	-1.215	non-owner
0.786	-0.556	non-owner
-0.426	-0.226	owner
2.105	0.103	owner
-0.881	1.256	owner
0.635	0.432	owner

non-owner

k = 3

Majority neighbors non-owner. Therefore …

With the Training A and Training B partitions, let us see how the process would work for k=3.
*Note that we are using the **standardized** attributes.*

To classify a row of the Training B partition, we do the following:

1. Find its closest three neighbors (since k = 3) in the Training A partition using the usual distance formula.
2. Find which *ownership* status has a majority among the three neighbors
3. Classify the case from the Training B partition as belonging to the majority case.

Classify all cases in the Training B partition set using the same method.

At the end of the process, we will have the model's classifications for k = 3. For each case, we also know the actual classification. Based on this, we can calculate the error matrix.

We repeat the same process for many other values of k and choose the k that performs best on the Training B partition.

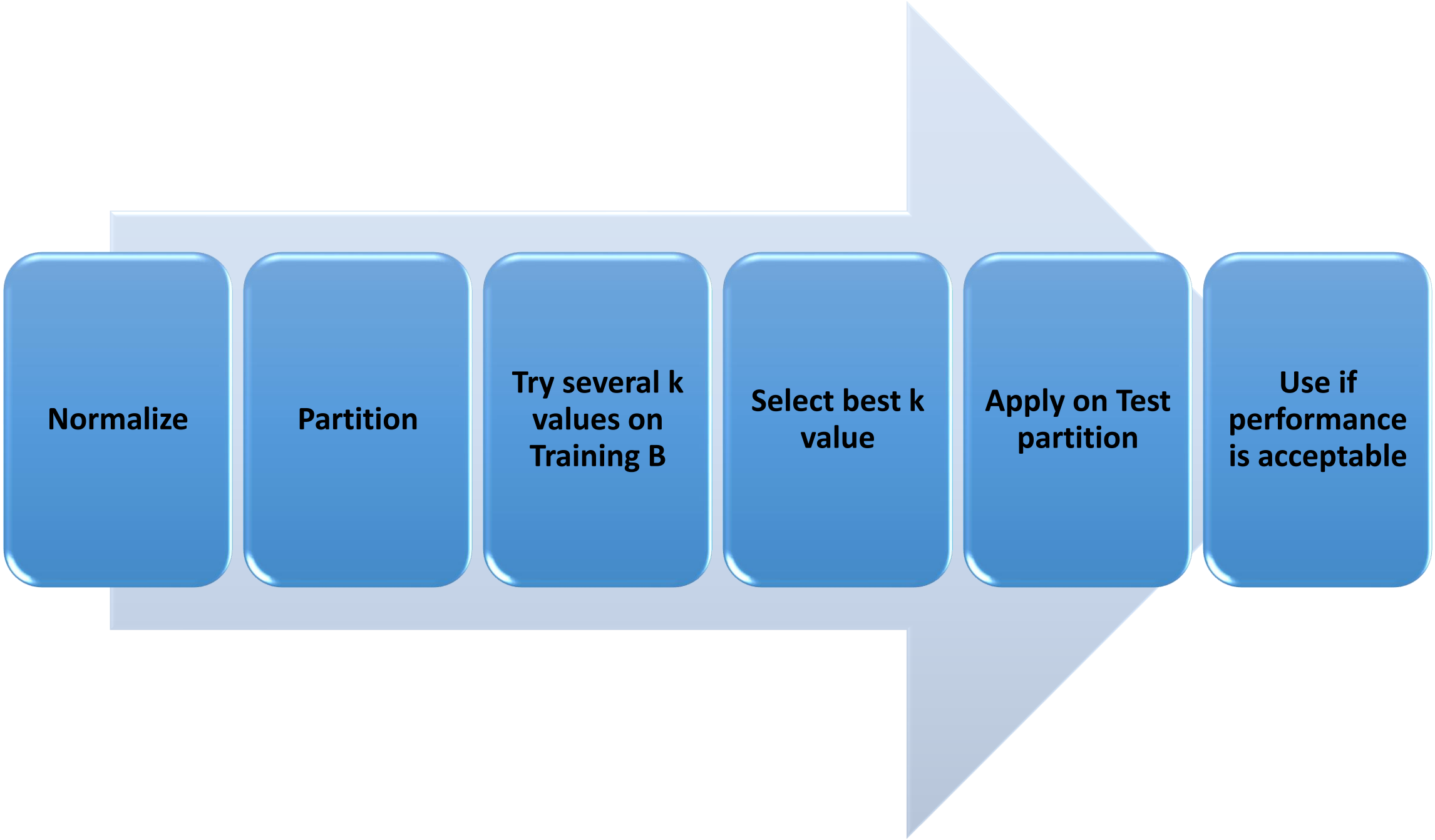
We then apply the best value of k to the cases in the Test partition and see how the model performs.

Training B

Norm_income	Norm_lot_size	Ownership	k=1 pred	k=3 pred	…
-0.457	-1.215	non-owner	…	…	…
0.786	-0.556	non-owner	…	…	…
-0.426	-0.226	owner	…	…	…
2.105	0.103	owner	…	…	…
-0.881	1.256	owner	…	…	…
0.635	0.432	owner	…	…	…

Model prediction on Training B

		k=1		k=3		k=5	
		Owner	Non-owner	Owner	Non-owner	Owner	Non-owner
Actual	Owner	10	5	12	3	9	6
	Non-Owner	5	30	4	31	11	24


Normalize
Partition
Try several k values on Training B
Select best k value
Apply on Test partition
Use if performance is acceptable

Model prediction on Training B

		k=1		k=3		k=5	
		Owner	Non-owner	Owner	Non-owner	Owner	Non-owner
Actual	Owner	10	5	12	3	9	6
	Non-Owner	5	30	4	31	11	24

k=1	k=3	k=5
40/50	43/50	33/50
80%	86%	66%

86% correctness Good?

86% with model

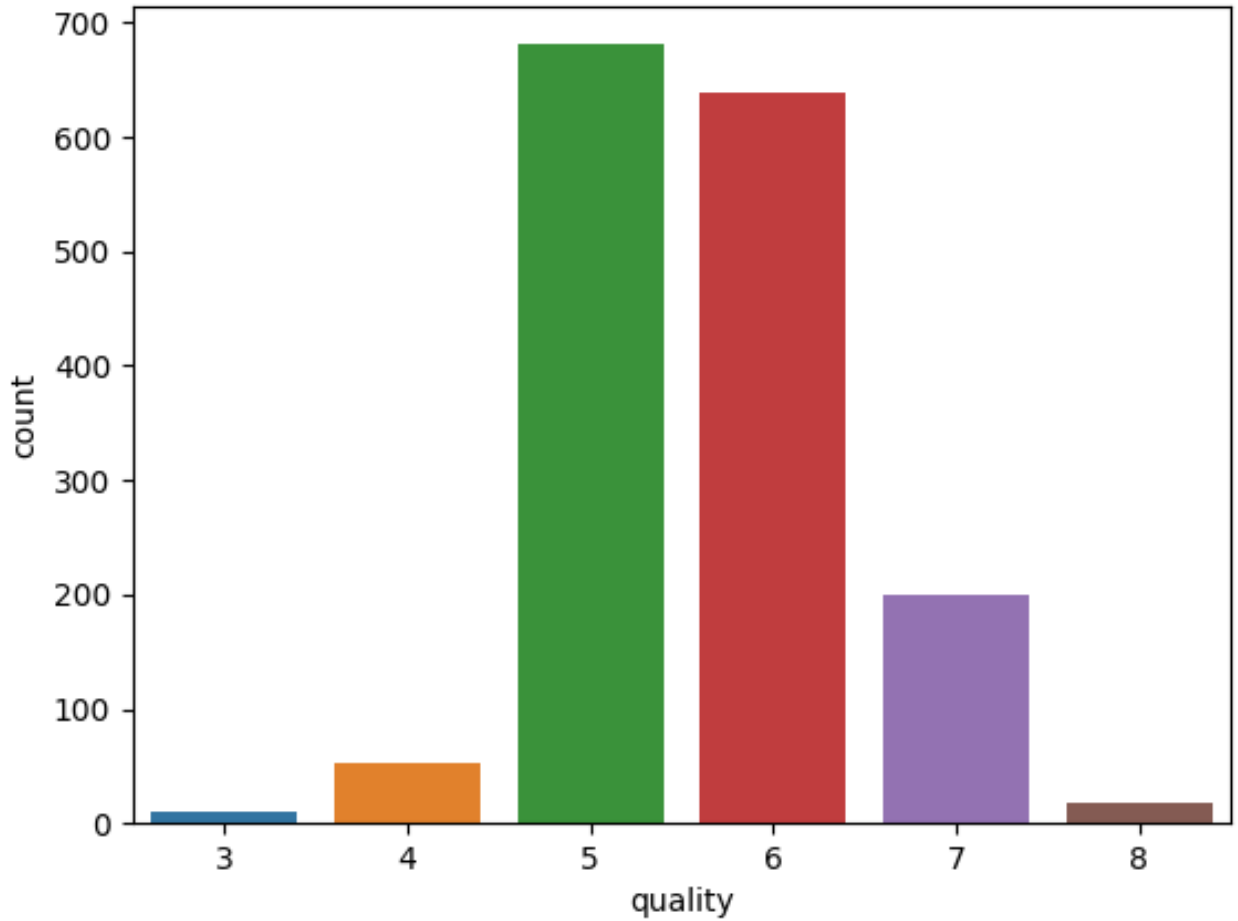
60% without

$$\text{Lift} = \frac{\text{Performance with model}}{\text{Performance without model}}$$

kNN Demo: Wine Quality

Classify wine quality exercise

- Import wine dataset
- Plot count of wine quality
- Import kNN classifier
- Instantiate the classifier
- Set the factors and response arrays
- Normalize the factors
- Partition the dataset
- Fit
- Predict
- Iterate on different value of k
- Plot the accuracy scores (training and testing) of the model for various ks
- Which k is best?


700
600
500
400
300
200
100
0
count
3
4
5
6
7
8
quality

in_class_coding.py

```
1  import numpy as np
2  import pandas as pd
3  import os
4  import matplotlib.pyplot as plt
5  import seaborn as sb
6
7  os.chdir('C:/Users/Reza/Desktop/ITP449_Fall2020/Class/Files')
8  pd.set_option('display.max_columns', None)
9
10 wineData = pd.read_csv('wineQualityReds.csv')
11
12 sb.countplot(x='quality', data=wineData)
13 plt.show()
14
```

1 20 247

Run: in_class_coding

```
C:\Users\Reza\Desktop\ITP449_Fall2020\Class\venv\Scripts\python.exe "C:/Users/Reza/Desktop/ITP449_Fall2020/Class/In Class Coc
```

in_class_coding.py

```
14  # Drop Wine # column
15  wineData.drop(wineData.columns[[0]], axis=1, inplace=True)
16
17  # Features and Target
18  X = wineData.iloc[:, 0:11]
19  y = wineData.iloc[:, 11]
20
21  # Normalize
22  from sklearn.preprocessing import Normalizer
23  myNorm = Normalizer()
24  X = pd.DataFrame(myNorm.transform(X), columns=X.columns)
25  print(X.head())
26
```

1 19 247

Run: in_class_coding

```
   fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chlorides  \
0       0.195153          0.018460     0.000000        0.050107   0.002004
1       0.107241          0.012099     0.000000        0.035747   0.001347
2       0.135457          0.013198     0.000695        0.039942   0.001598
3       0.174367          0.004359     0.008718        0.029580   0.001168
4       0.195153          0.018460     0.000000        0.050107   0.002004
```

in_class_coding.py

```
13
14  # Drop Wine # column
15  wineData.drop(wineData.columns[[0]], axis=1, inplace=True)
16
17  # Features and Target
18  X = wineData.iloc[:, 0:11]
19  y = wineData.iloc[:, 11]
20
21  # Normalize
22  from sklearn.preprocessing import Normalizer
23  myNorm = Normalizer()
24  X = pd.DataFrame(myNorm.transform(X), columns=X.columns)
25  print(X.head())
26
27  # Partitions
28  from sklearn.model_selection import train_test_split
29  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2020, stratify=y)
30
```

stratify = y means partitions will have the same proportions of values as in vector y

in_class_coding.py

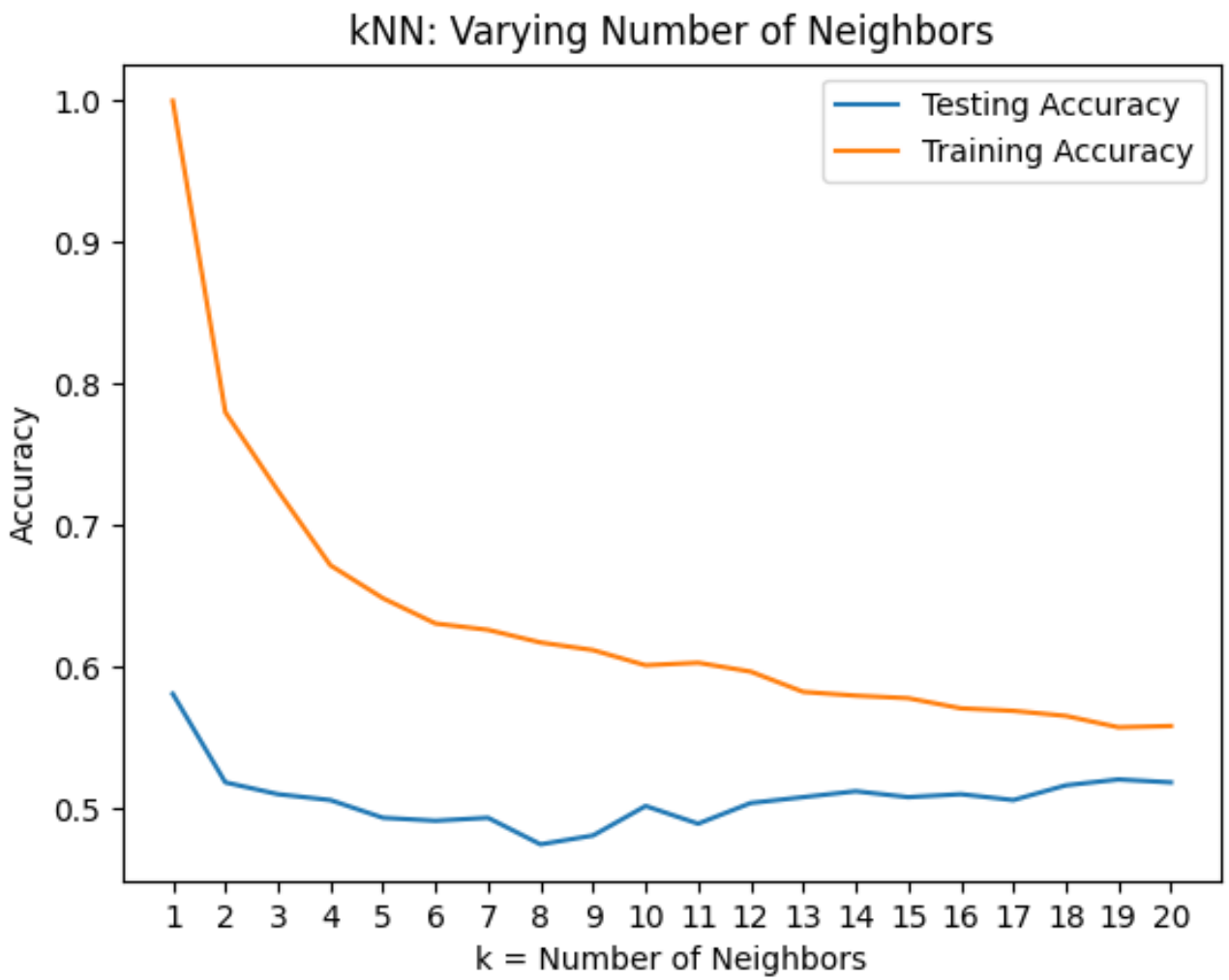
```
29  from sklearn.model_selection import train_test_split
30  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2020, stratify=y)
31
32  # range of k neighbors
33  neighbors = np.arange(1, 21)
34  trainAccuracy = np.empty(20)
35  testAccuracy = np.empty(20)
36
37  from sklearn.neighbors import KNeighborsClassifier
38  from sklearn.metrics import confusion_matrix
39
40  for k in neighbors:
41      kNN = KNeighborsClassifier(n_neighbors=k)
42      kNN.fit(X_train, y_train) # Does not do much since there is no training per se
43      y_pred = kNN.predict(X_test)
44      cf = confusion_matrix(y_test, y_pred)
45      trainAccuracy[k-1] = kNN.score(X_train, y_train)
46      testAccuracy[k-1] = kNN.score(X_test, y_test)
47
```

21 248

in_class_coding.py

21 248

```
39
40  for k in neighbors:
41      kNN = KNeighborsClassifier(n_neighbors=k)
42      kNN.fit(X_train, y_train) # Does not do much since there is no training per se
43      y_pred = kNN.predict(X_test)
44      cf = confusion_matrix(y_test, y_pred)
45      trainAccuracy[k-1] = kNN.score(X_train, y_train)
46      testAccuracy[k-1] = kNN.score(X_test, y_test)
47
48  # Generating the accuracy plots
49  plt.plot(neighbors, testAccuracy, label='Testing Accuracy')
50  plt.plot(neighbors, trainAccuracy, label='Training Accuracy')
51  plt.xticks(neighbors)
52  plt.legend()
53  plt.title('kNN: Varying Number of Neighbors')
54  plt.xlabel('k = Number of Neighbors')
55  plt.ylabel('Accuracy')
56  plt.show()
57
```


kNN: Varying Number of Neighbors
Testing Accuracy
Training Accuracy
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
k = Number of Neighbors