Python Data Analytics

OBA 410/510

Lundquist College of Business

Background image from www. givweb.com

Some of the slides and contents are adopted from Statistical Learning course by Dr. Trevor Hastie

Querying DataFrames

UNIVERSITY OF OREGON
Lundquist College of Business

Querying DataFrames

- Boolean mask:
  - We can use masking when we want to query or manipulate values in a DataFrame based on some criteria

```
df5
```

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Ag
p1	6.0	148.0	72.0	0.0	33.6	50.
p3	8.0	183.0	64.0	0.0	23.3	32.
p4	1.0	89.0	66.0	94.0	28.1	21.
p5	0.0	137.0	40.0	168.0	43.1	33.
p6	1.0	162.0	81.0	155.0	4.0	71.
p7	8.0	183.0	64.0	0.0	23.3	32.

```
gl_100=df5['Glucose']>100
gl_100
```

```
p1     True
p3     True
p4    False
p5     True
p6     True
p7     True
Name: Glucose, dtype: bool
```

```
df5[gl_100]
```

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	DiabetesPedigreeFunction	Age	Outcome	HbA1c	test
p1	6.0	148.0	72.0	0.0	33.6	0.627	50.0	1.0	34.0	1.470588
p3	8.0	183.0	64.0	70.0	23.3	0.672	32.0	1.0	41.0	0.780488
p5	0.0	137.0	40.0	168.0	43.1	2.288	33.0	1.0	61.0	0.540984

```
df5[df5['Glucose']>100]
```

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	DiabetesPedigreeFunction	Age	Outcome	HbA1c	test
p1	6.0	148.0	72.0	0.0	33.6	0.627	50.0	1.0	34.0	1.470588
p3	8.0	183.0	64.0	70.0	23.3	0.672	32.0	1.0	41.0	0.780488
p5	0.0	137.0	40.0	168.0	43.1	2.288	33.0	1.0	61.0	0.540984

Querying DataFrames

- More complicated queries using
  - Comparison operators
    - <, <=, >, >=, ==, !=
  - Logical
    - And → &
    - Or → |

```
8]: d=(df5['Glucose']>100) & (df5['Glucose']<150)
    d
```

```
8]: p1     True
    p3    False
    p4    False
    p5     True
    p6    False
    p7    False
    Name: Glucose, dtype: bool
```

```
0]: df5[d]
```

0]:

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Age	Outcome	Hb/
p1	6.0	148.0	72.0	0.0	33.6	50.0	1.0	3
p5	0.0	137.0	40.0	168.0	43.1	33.0	1.0	6

Querying DataFrames

```
0]: df5[df5['Num_of_Preg']==1]
```

0]:

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Age	Outcome	HbA1c	test
p4	1.0	89.0	66.0	94.0	28.1				
p6	1.0	162.0	81.0	155.0	4.0				

```
41]: df5[(df5['Num_of_Preg']==1) & (df5['Body_Mass_Index']>27)]
```

41]:

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Age	Outcome	HbA1c	test
p4	1.0	89.0	66.0	94.0	28.1	21.0	0.0	52.0	0.211538

```
42]: a=(df5['Num_of_Preg']==1) & (df5['Body_Mass_Index']>27)
     a
```

```
42]: p1     False
     p3     False
     p4      True
     p5     False
     p6     False
     p7     False
     dtype: bool
```

```
44]: df5[a]
```

44]:

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Age	Outcome	HbA1c	test
p4	1.0	89.0	66.0	94.0	28.1	21.0	0.0	52.0	0.211538

Querying DataFrames

UNIVERSITY OF OREGON
Lundquist College of Business

- parentheses are important!

54]:
```
# patients (younger than 32 AND BP above 65)  OR  Glucose is 183
y_32=df5['Age']<32
bp_65=df5['BP']>65
gl_2=df5['Glucose']==183
con3=(y_32 & bp_65) | gl_2
df5[con3]
```

54]:

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Age	Outcome	HbA1c	test
p3	8.0	183.0	64.0	70.0	23.3	32.0	1.0	41.0	0.536585
p4	1.0	89.0	66.0	94.0	28.1	21.0	0.0	52.0	0.211538
p7	8.0	183.0	64.0	70.0	23.3	32.0	1.0	41.0	0.536585

58]:
```
# patients younger than 32  AND  (BP above 65 OR Glucose is 183)
y_32=df5['Age']<32
bp_65=df5['BP']>65
gl_2=df5['Glucose']==183
con4=y_32 & (bp_65 | gl_2)
df5[con4]
```

58]:

	Num_of_Preg	Glucose	BP	Insulin	Body_Mass_Index	Age	Outcome	HbA1c	test
p4	1.0	89.0	66.0	94.0	28.1	21.0	0.0	52.0	0.211538

9]:
```
df5[con4][['Num_of_Preg','Glucose']]
```

9]:

	Num_of_Preg	Glucose
p4	1.0	89.0

Summary Statistics

- *df.describe():* Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution
- It ignores missing (NaN) values

```
In [259]: df.describe()
Out[259]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

UNIVERSITY OF OREGON
Lundquist College of Business

Missing Values

- Real world datasets always have missing values
- To develop most predictive models, we need to fix missing values

```
In [377]: df_miss=pd.read_csv('diabetes_w_missing.csv')
```

```
In [378]: df_miss
```

Out[378]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Gender
0	6	148.0	72.0	35.0	0	33.6	0.627	50	1	Male
1	1	85.0	66.0	29.0	0	26.6	0.351	31	0	Male
2	8	183.0	64.0	0.0	0	23.3	0.672	32	1	Female
3	1	89.0	66.0	23.0	94	NaN	0.167	21	0	NaN
4	0	137.0	40.0	35.0	168	43.1	2.288	33	1	Male
5	5	116.0	74.0	0.0	0	25.6	0.201	30	0	Female
6	3	NaN	50.0	32.0	88	31.0	0.248	26	1	NaN
7	10	115.0	0.0	0.0	0	35.3	0.134	29	0	Female

```
In [379]: df_miss.shape

Out[379]: (48, 10)
```

```
In [382]: df_miss['Gender'].describe()

Out[382]: count         46
          unique         2
          top       Female
          freq          35
          Name: Gender, dtype: object
```

```
In [381]: df_miss.describe()
```

Out[381]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	48.000000	43.000000	47.000000	46.000000	48.000000	42.000000	48.000000	48.000000	48.000000
mean	5.333333	129.255814	70.276596	19.195652	84.937500	31.564286	0.523354	37.708333	0.500000

UNIVERSITY OF OREGON
Lundquist College of Business

Removing missing values

In [475]:
```
# by default dropps rows with missing values
df_miss.dropna()
```

Out[475]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Gender
0	6	148.0	72.0	35.0	0	33.6	0.627	50	1	Male
1	1	85.0	66.0	29.0	0	26.6	0.351	31	0	Male
2	8	183.0	64.0	0.0	0	23.3	0.672	32	1	Female
4	0	137.0	40.0	35.0	168	43.1	2.288	33	1	Male
5	5	116.0	74.0	0.0	0	25.6	0.201	30	0	Female
7	10	115.0	0.0	0.0	0	35.3	0.134	29	0	Female
8	2	197.0	70.0	45.0	543	30.5	0.158	53	1	Female
11	10	168.0	74.0	0.0	0	38.0	0.537	34	1	Female
13	1	189.0	60.0	23.0	846	30.1	0.398	59	1	Female
14	5	166.0	72.0	19.0	175	25.8	0.587	51	1	Female
15	7	100.0	0.0	0.0	0	30.0	0.484	32	1	Female

In [477]:
```
# dropps column with missing values
df_miss.dropna(axis=1)
```

Out[477]:

	Pregnancies	Insulin	DiabetesPedigreeFunction	Age	Outcome
0	6	0	0.627	50	1
1	1	0	0.351	31	0
2	8	0	0.672	32	1
3	1	94	0.167	21	0
4	0	168	2.288	33	1
5	5	0	0.201	30	0

UNIVERSITY OF OREGON
Lundquist College of Business

Imputing Missing Values

In [398]:
```
Glucose_mean=np.mean(df_miss['Glucose'])
SkinThickness_mean=np.mean(df_miss['SkinThickness'])
Gender_mode=df_miss['Gender'].mode()
imp_values={'Glucose':Glucose_mean,'SkinThickness':SkinThickness_mean,'BMI':20, 'Gender':Gender_mode[0]}
df_miss.fillna(value=imp_values)
```

Out[398]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Gender
0	6	148.000000	72.0	35.000000	0	33.6	0.627	50	1	Male
1	1	85.000000	66.0	29.000000	0	26.6	0.351	31	0	Male
2	8	183.000000	64.0	0.000000	0	23.3	0.672	32	1	Female
3	1	89.000000	66.0	23.000000	94	20.0	0.167	21	0	Female
4	0	137.000000	40.0	35.000000	168	43.1	2.288	33	1	Male
5	5	116.000000	74.0	0.000000	0	25.6	0.201	30	0	Female
6	3	129.255814	50.0	32.000000	88	31.0	0.248	26	1	Female
7	10	115.000000	0.0	0.000000	0	35.3	0.134	29	0	Female
8	2	197.000000	70.0	45.000000	543	30.5	0.158	53	1	Female
9	8	125.000000	96.0	19.195652	0	0.0	0.232	54	1	Female
10	4	110.000000	92.0	0.000000	0	20.0	0.191	30	0	Female

Imputing Missing Values

Imputing all numeric columns with their mean

```
In [9]:  1  df_miss.mean()

Out[9]: Pregnancies          5.333333
        Glucose            129.255814
        BloodPressure       70.276596
        SkinThickness       19.195652
        Insulin             84.937500
        BMI                 31.564286
```

```
In [8]:  1  df_miss.fillna(df_miss.mean())
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Gende
0	6	148.000000	72.000000	35.000000	0	33.600000	0.627	50	1	Mal
1	1	85.000000	66.000000	29.000000	0	26.600000	0.351	31	0	Mal
2	8	183.000000	64.000000	0.000000	0	23.300000	0.672	32	1	Femal
3	1	89.000000	66.000000	23.000000	94	31.564286	0.167	21	0	Nal
4	0	137.000000	40.000000	35.000000	168	43.100000	2.288	33	1	Mal
5	5	116.000000	74.000000	0.000000	0	25.600000	0.201	30	0	Femal
6	3	129.255814	50.000000	32.000000	88	31.000000	0.248	26	1	Nal
7	10	115.000000	0.000000	0.000000	0	35.300000	0.134	29	0	Femal
8	2	197.000000	70.000000	45.000000	543	30.500000	0.158	53	1	Femal
9	8	125.000000	96.000000	19.195652	0	0.000000	0.232	54	1	Femal
10	4	110.000000	92.000000	0.000000	0	31.564286	0.191	30	0	Femal

UNIVERSITY OF OREGON
Lundquist College of Business

Developing Machine Learning

(predictive) Models

Classification and Regression

- Two major types of supervised machine learning problem:
  - Classification
    - The goal is to predict a class label
    - Binary classification (classifying email as spam or not spam, classifying tumors as benign or malignant)
    - Multiclass classification (classifying the language of a text, classifying fruits to different types)
  - Regression
    - The goal is to predicting a continuous or a floating number
    - Predicting a person annual income
    - Predicting a house price

k-Nearest Neighbors

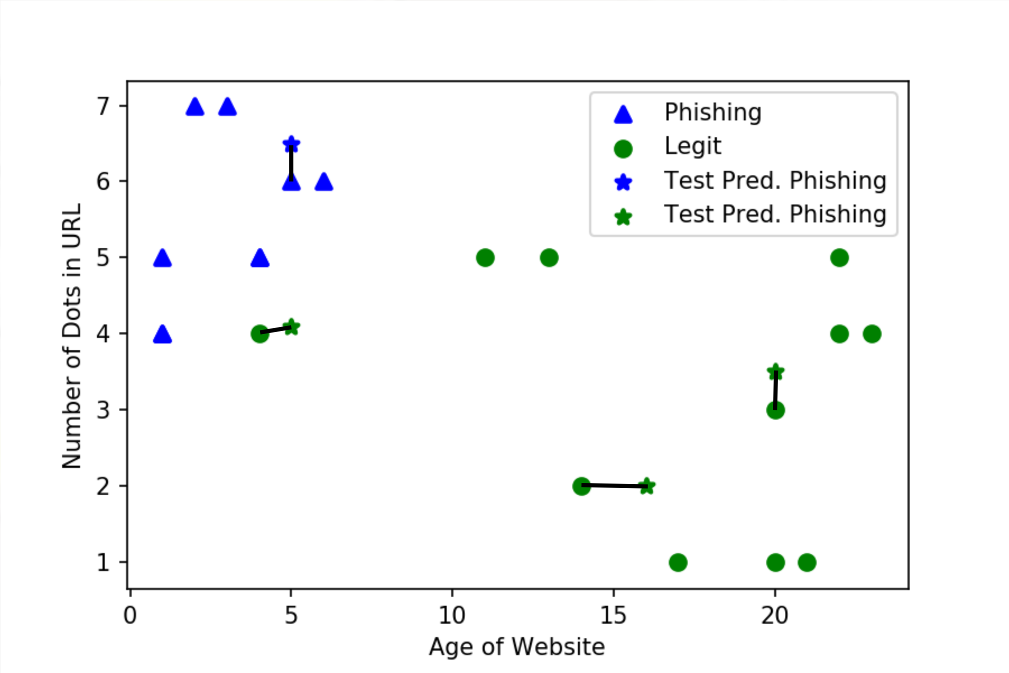
k-Nearest Neighbors

- k-NN is arguably the simplest machine learning algorithm
- k-NN algorithm can be used for both classification and regression
- Building the model consists only of storing the training data
- To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset (its “nearest neighbors”)

k-NN Classification

- In the simplest version, the algorithm only considers exactly one nearest neighbor: the closest training data point
- The prediction is simply the known output of this training data point

- Phishing Website Data:
    - Two features:
        - **Age of website** (phishing websites are usually new)
        - **Number of dots in the URL** (it is unlikely for a legitimate website to have many dots in the URL while most phishing websites have many dots in the URL)
    - Target:
        - **Phishing or Legit**

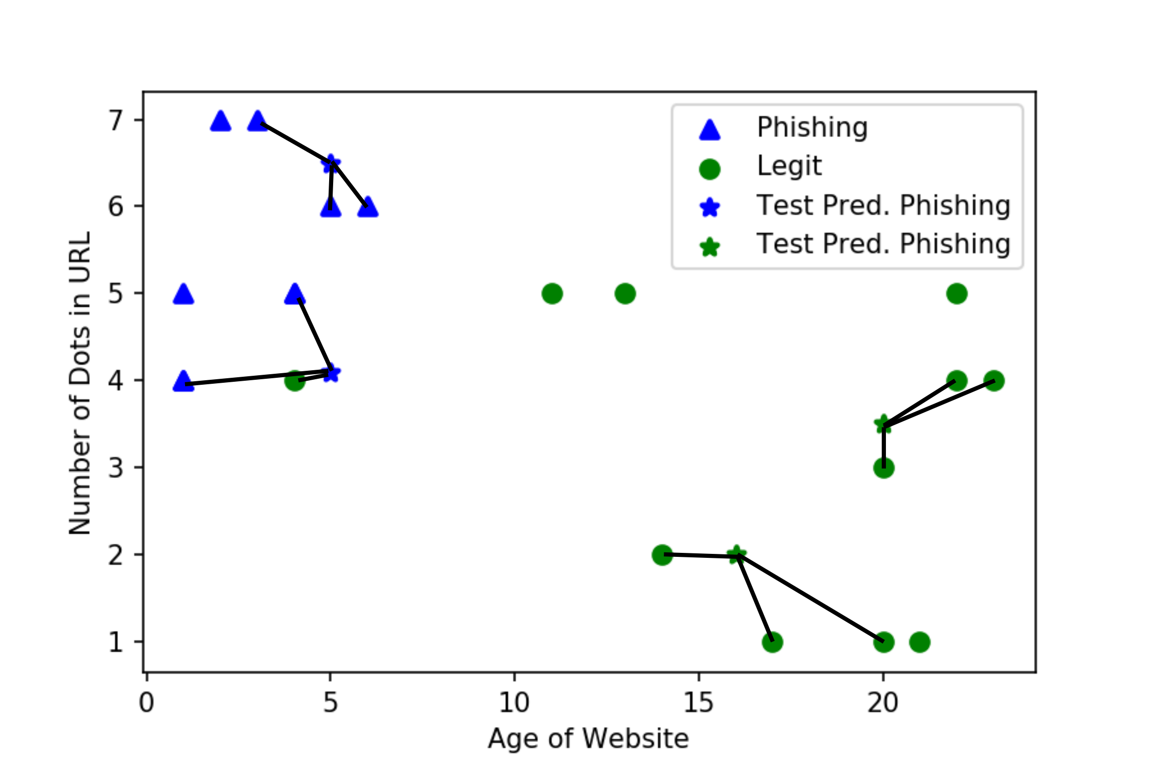


k-NN Classification

- Instead of considering only the closest neighbor, we can also consider any arbitrary number (k) of neighbors
- This is where the name of k-NN comes from
- When considering more than one neighbor, we use *voting* to make the classification
  - We assign the majority class among the k-nearest neighbors to the test data
  - If there is a tie, class is randomly chosen. However, we typically choose an odd value for k.

k-NN Classification

- k-NN example (number of neighbors=3)



- k-NN can be applied to multi class problems as well

k-NN Classification

```
import pandas as pd
phish_data=pd.read_csv('Phishing_websites.csv')
phish_data.head(4)
```

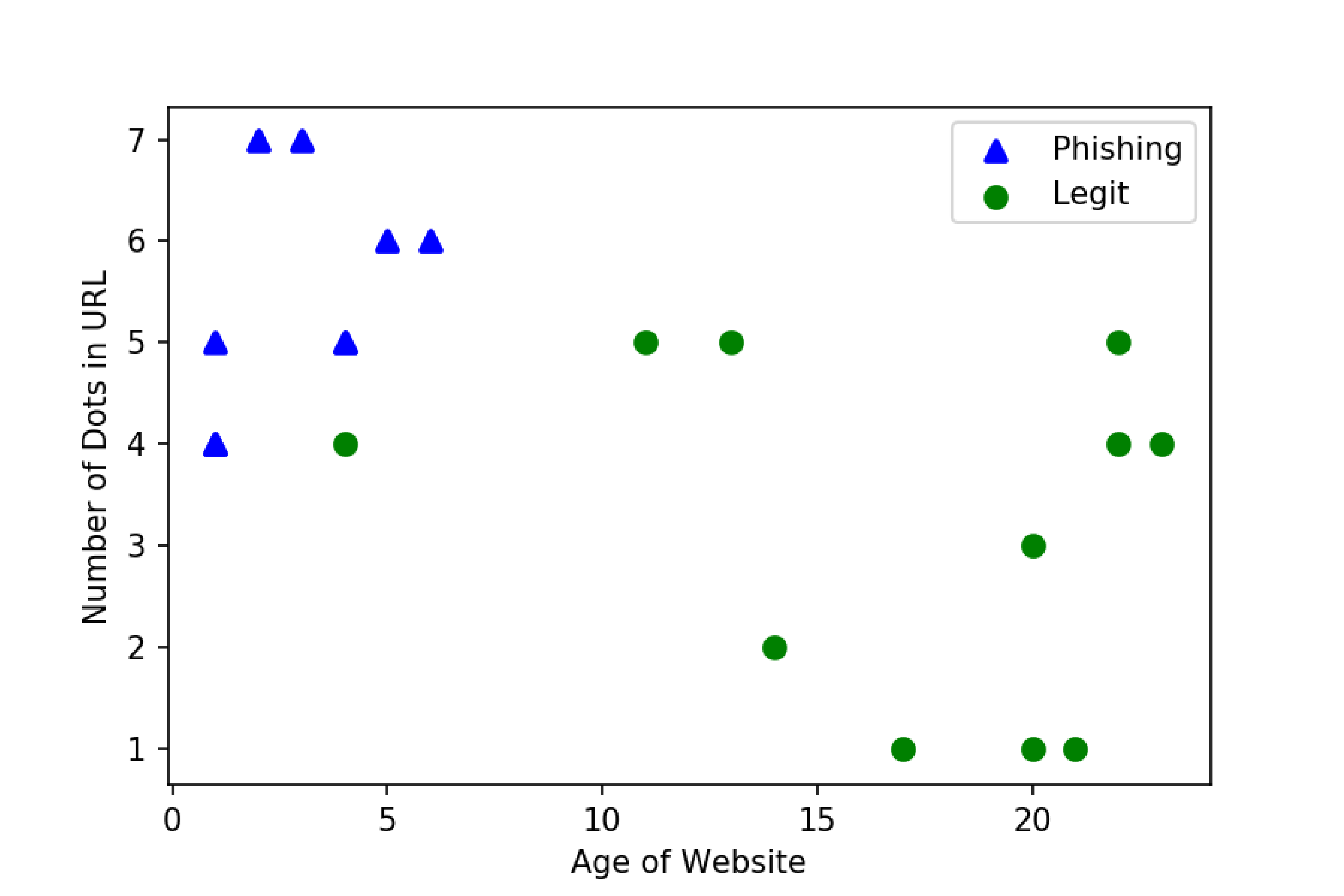
	Age_of_web	num_of_dots_inURL	Phishing
0	1	5	1
1	5	6	1
2	6	6	1
3	2	7	1

```
# creating features and target sets (separte feature and target)
X_phish=phish_data.iloc[: , :2]
y_phish=phish_data['Phishing']
# check my work
X_phish.head(3)
```

	Age_of_web	num_of_dots_inURL
0	1	5
1	5	6
2	6	6

Classification Problem Datasets

- Phishing Website Data



k-NN Classification

- Applying k-NN algorithm using *scikit-learn* package.
  - First step, importing the class (algorithm)
  - Specifying the algorithm parameters
  - Fitting the classifier using the training dataset (including the features and target labels) by calling ***fit*** method
    - All classifiers have a fit method that takes the training data, and then changes the state of the classifier, to essentially enable prediction once the training is finished
  - To make prediction, we call the ***predict*** method.
    - For each test data point, this computes its nearest neighbors in the training set, and finds the most common class among them

k-NN Classification

- To evaluate the model accuracy, we call the ***score*** method
  - We have to pass both features and target of data
  - *score* method, returns the accuracy of our model on a test dataset
  - Accuracy is calculated as:
    - $Accuracy = \frac{\#\ of\ correct\ predictions}{total\ \#\ of\ predictions}$

k-NN Classification

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn_cls=KNeighborsClassifier(n_neighbors=3)
knn_cls.fit(X_phish,y_phish)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
           weights='uniform')
```

```
knn_cls.predict([[3,8]])
```

```
array([1], dtype=int64)
```

```
knn_cls.predict([[3,8],[40,3],[23,4]])
```

```
array([1, 0, 0], dtype=int64)
```

```
knn_cls.score(X_phish,y_phish)
```

```
0.95
```

k-NN Algorithm Parameters

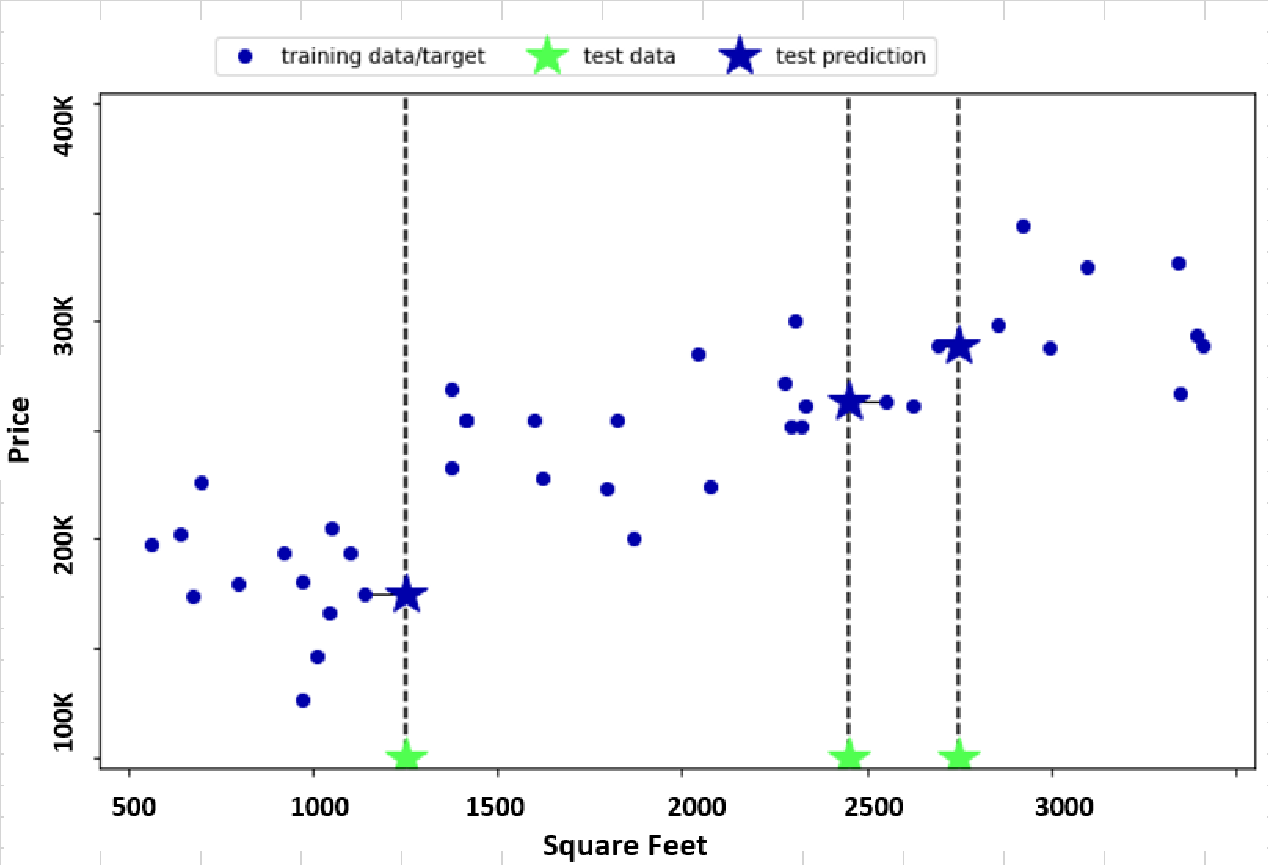
- **n_neighbors** : int, optional (default = 5)
  - Number of neighbors to use. In practice, small numbers like 3 to 5 work well
- weights : str or callable, optional (default = 'uniform')
  - Weight function used in prediction. Possible values:
    - 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
    - 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- metric : string or callable, default 'minkowski'
  - The distance metric to use. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric

```
In [ ]: KNeighborsClassifier()

In [ ]:  Init signature: KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto',
          leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
         Docstring:
         Classifier implementing the k-nearest neighbors vote.
```

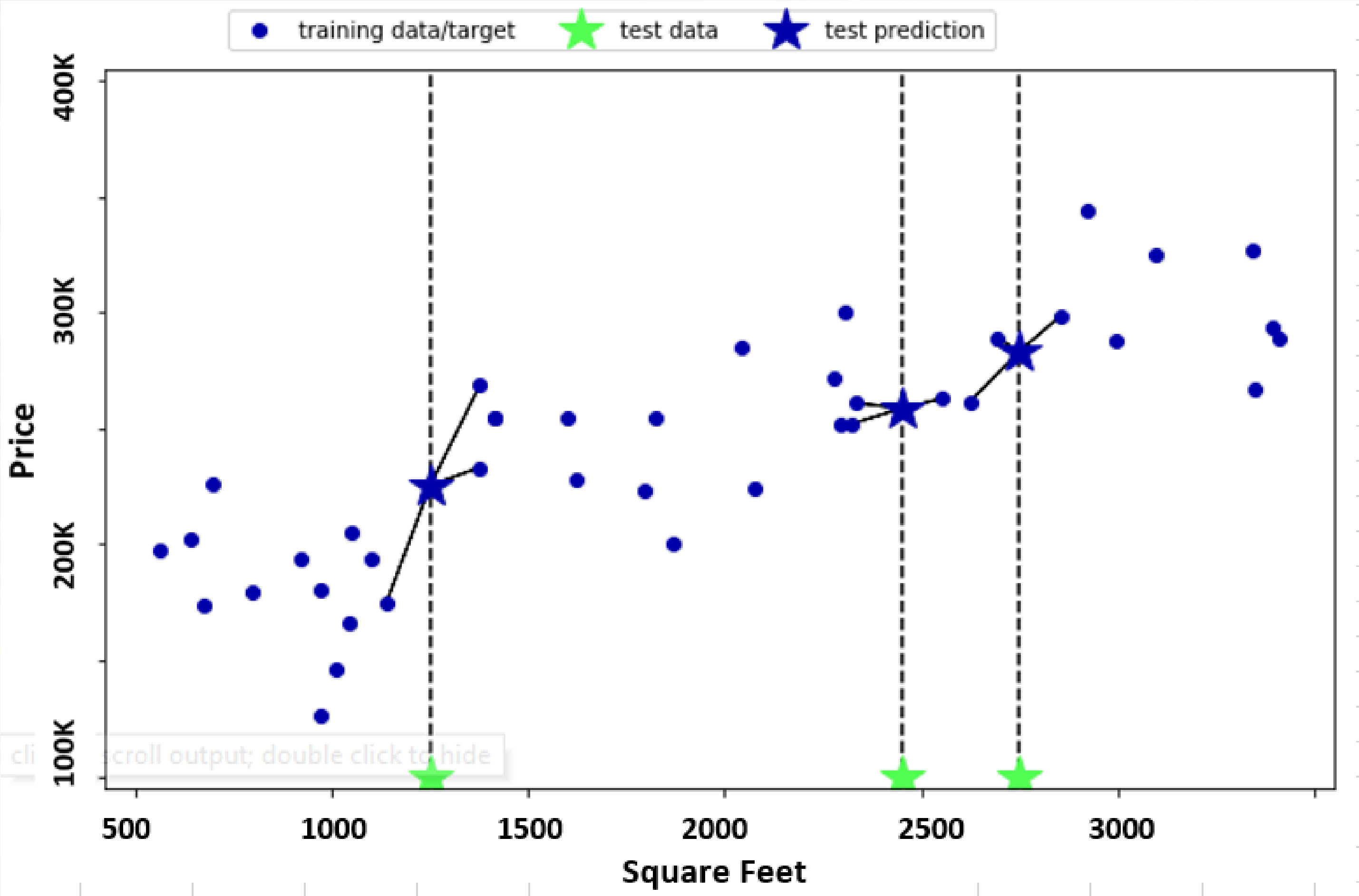
k-NN Regression

- There is also a regression version of k-NN algorithm
- Let's start with single nearest neighbor (using house_1feature dataset)
- We have added 3 test data points (Green stars on x-axis)
- The prediction using a single neighbor is just the target value of nearest neighbor (blue stars)



k-NN Regression

- We can use more than just one nearest neighbor for k-NN regression
- When using multiple nearest neighbors, the prediction is the average or mean of the neighbors target values



k-NN Regression

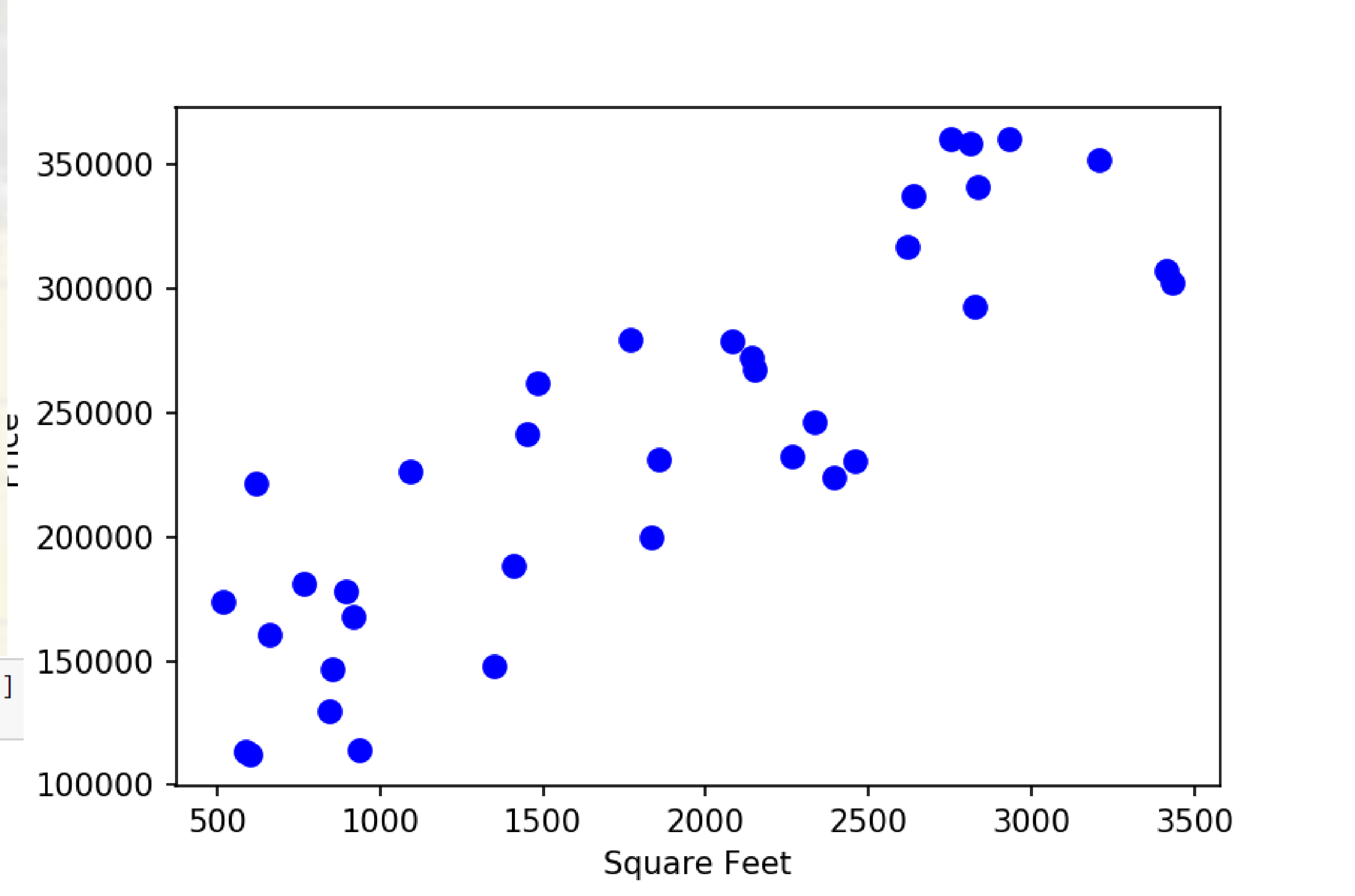
- Applying k-NN algorithm using *scikit-learn* package.
  - First step, importing the class (algorithm)
  - Specifying the algorithm parameters
  - Fitting the model using the training dataset (including the features and target labels) by calling *fit* method
    - All models have a fit method that takes the training data, and then changes the state of the model, to essentially enable prediction once the training is finished
  - To make prediction, we call the *predict* method.
    - For each test data point, this computes its nearest neighbors in the training set, and finds the mean target value among the nearest neighbors

k-NN Regression

- To evaluate the model performance, we call the *score* method
  - We have to pass both features and target of the data
  - score method returns R^2 for regressors
  - R^2 also known as coefficient of determination, is a measure of goodness of prediction for a regression model
  - R^2 is the proportion of variability in the target explained by features
  - R^2 could be between 0 and 1
  - A value of 1 corresponds to perfect prediction
  - A value of 0 corresponds to a constant model that just predicts the mean of the training target value

Regression Problem Datasets

- House_1feature Data
  - One features:
    - **Size of the house**
  - Target:
    - **Price**




```
X_house, y_house=house1[['Square_feet']], house1['Price']
X_house.head()
```

	Square_feet
0	1349
1	897
2	660

k-NN Regression

```
#importing the algorithm
from sklearn.neighbors import KNeighborsRegressor
```

```
# defining and fitting the model
knn_reg=KNeighborsRegressor(n_neighbors=3)
knn_reg.fit(X_house,y_house)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=3, p=2,
          weights='uniform')
```

```
# predicting some test examples
knn_reg.predict([[3000]])
```

```
array([331343.])
```

```
# predicting some test examples
knn_reg.predict([[1000],[1800],[2350],[3000]])
```

```
array([169333.33333333, 236720.33333333, 234297.66666667, 331343.        ])
```

```
# evaluating the performance of the model
knn_reg.score(X_house,y_house)
```

```
0.8574636192620329
```

k-NN Characteristics

- Very easy to understand
- Often has reasonable performance without a lot of adjustments
- Very fast on small datasets
- It makes few assumptions about the structure of the data
- Slow prediction on large datasets
- Often does not perform well on datasets with many features
- Any small change to the training data will impact the model

Breast cancer data

- Data on breast cancer tumor
- Columns

Target: Diagnosis (M = malignant, B = benign)

Ten real-valued features are computed for each cell nucleus. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

Boston housing data

- This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston
- The Boston data frame has 506 rows and 14 columns. This data frame contains the following columns:

Target: *target_medv*

<u>median value </u>of owner-occupied homes in $1000s.

Features:

Crim: per capita crime rate by town.
Zn: proportion of residential land zoned for lots over 25,000 sq.ft.
Indus: proportion of non-retail business acres per town.
Chas: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
nox: nitrogen oxides concentration (parts per 10 million).
rm: average number of rooms per dwelling.
age: proportion of owner-occupied units built prior to 1940.
dis: weighted mean of distances to five Boston employment centres.
rad: index of accessibility to radial highways.
tax: full-value property-tax rate per $10,000.
ptratio: pupil-teacher ratio by town.
lstat: lower status of the population (percent).

UNIVERSITY OF OREGON
Lundquist College of Business

Your turn

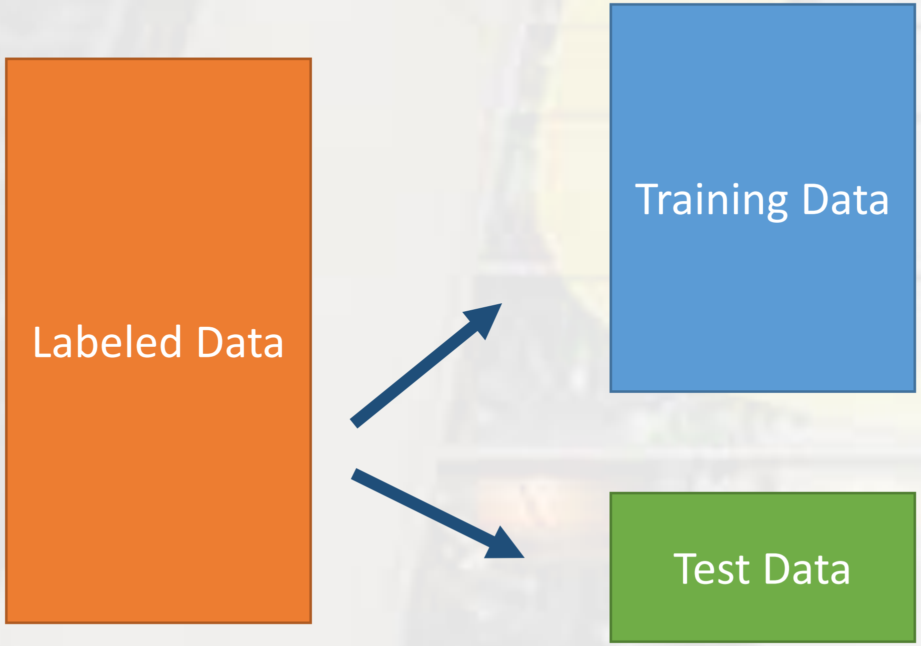
- k-NN Classification on:
  - Breast cancer data
- k-NN regression on:
  - Boston housing data

Fair Evaluation

- The objective of developing predictive models is to predict the outcome of future unseen data points
- But before we apply our models, we need to make sure they actually work!
- We cannot use the training data (data we used to fit the model) to fairly evaluate a model
- Because our model has already seen that data and remembers it
- To assess our model performance, we have to use unseen data for which we have the true target values

Fair Evaluation

- We split our labeled data into two parts:
  - Training data, which we use to develop and fit the model
  - Test data, which we use to assess our developed model



Fair Evaluation

- In practice, usually 70-75% of data is used for training and 25-30% for test
- *Scikit-learn* function for splitting the data: *train-test-split*
- It shuffles and split the data (default 25%/75%)
- Since, this function splits the data randomly, if we want to always get the same results, we have to use the *random_state* parameter

UNIVERSITY OF OREGON
Lundquist College of Business

train-test-split function

- Splitting the Boston Housing Data

```
In [137]: from sklearn.model_selection import train_test_split

In [138]: X_train, X_test, y_train, y_test=train_test_split(X_house,y_house,random_state=0)

In [141]: X_train.shape

Out[141]: (249, 12)

In [142]: X_test.shape

Out[142]: (84, 12)
```

Re-Developing k-NN Models

- k-NN Classification on:
  - Phishing website data
  - Breast cancer data
- k-NN regression on:
  - House_1f data
  - Boston housing data