Telefonica

UNIVERSIDAD PONTIFICIA DE SALAMANCA

Experto en Big Data

Machine Learning with Python

Henar Muñoz Frutos
Big Data Developer

Universidad Pontificia de Salamanca 2017 - 2018

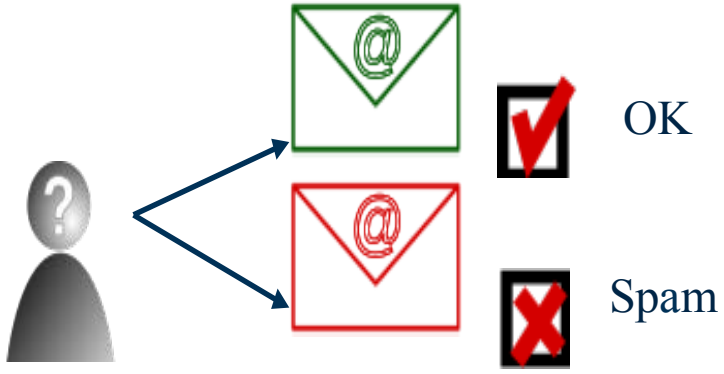
scikit learn

What is Machine Learning (ML)?

- **The study of computer programs (algorithms) that can learn by example**
- **ML algorithms can generalize from existing examples of a task**
  - *e.g. after seeing a training set of labeled images, an image classifier can figure out how to apply labels accurately to new, previously unseen images*

Machine Learning models learn from experience

- **Labeled examples (Email spam detection)**



- **User feedback (Clicks on a search page)**

- **Surrounding environment (self-driving cars)**

Machine Learning brings together statistics, computer science, and more..

- **Statistical methods**
  - *Infer conclusions from data*
  - *Estimate reliability of predictions*
- **Computer science**
  - *Large-scale computing architectures*
  - *Algorithms for capturing, manipulating, indexing, combining, retrieving and performing predictions on data*
  - *Software pipelines that manage the complexity of multiple subtasks*
- **Economics, biology, psychology**
  - *How can an individual or system efficiently improve their performance in a given environment?*
  - *What is learning and how can it be optimized?*

Applied Machine Learning

Key types of Machine Learning problems

Supervised machine learning: Learn to predict target values from labelled data.

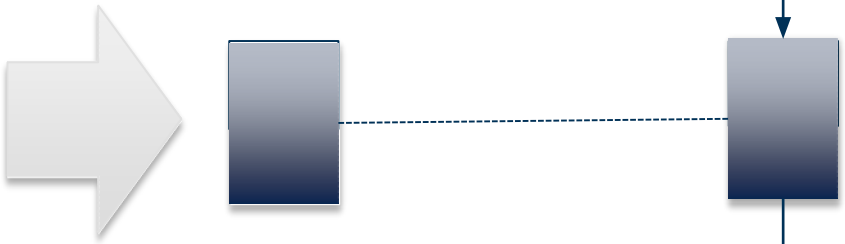
- **Classification (target values are discrete classes)**
- **Regression (target values are continuous values)**

Supervised Learning (classification example)

Training set

Sample	Target Value (Label)
 x_1	Apple y_1
 x_2	Lemon y_2
 x_3	Apple y_3
 x_4	Orange y_4

Classifier
$f : X \rightarrow Y$



At training time, the classifier uses labelled examples to learn rules for recognizing each fruit type.

Future sample

Label: Orange

After training, at prediction time, the trained model is used to predict the fruit type for new instances using the learned rules.

Key types of Machine Learning problems

Supervised machine learning: Learn to predict target values from labelled data.

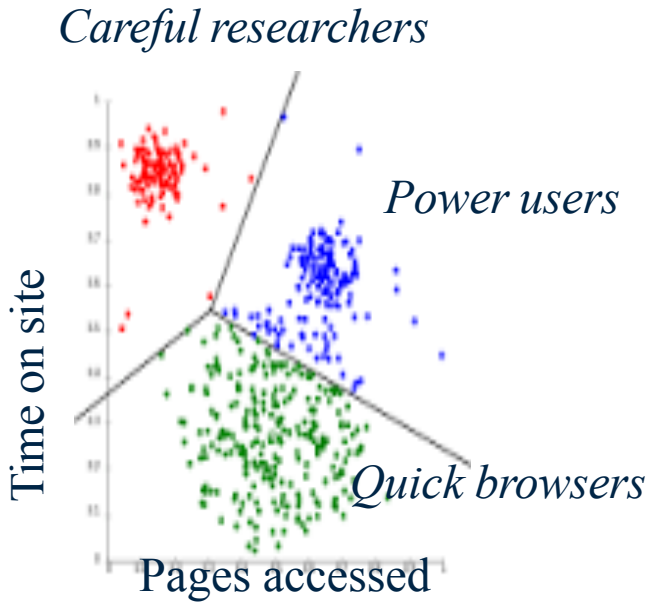
- Classification (target values are discrete classes)
- Regression (target values are continuous values)

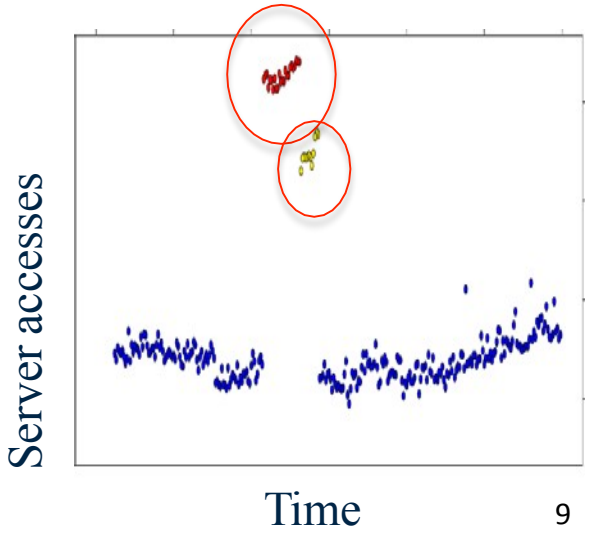
Unsupervised machine learning: Find structure in *unlabeled data*

- **Find groups of similar instances in the data (clustering)**
- **Finding unusual patterns (outlier detection)**

Unsupervised learning: finding useful structure or knowledge in data when no labels are available

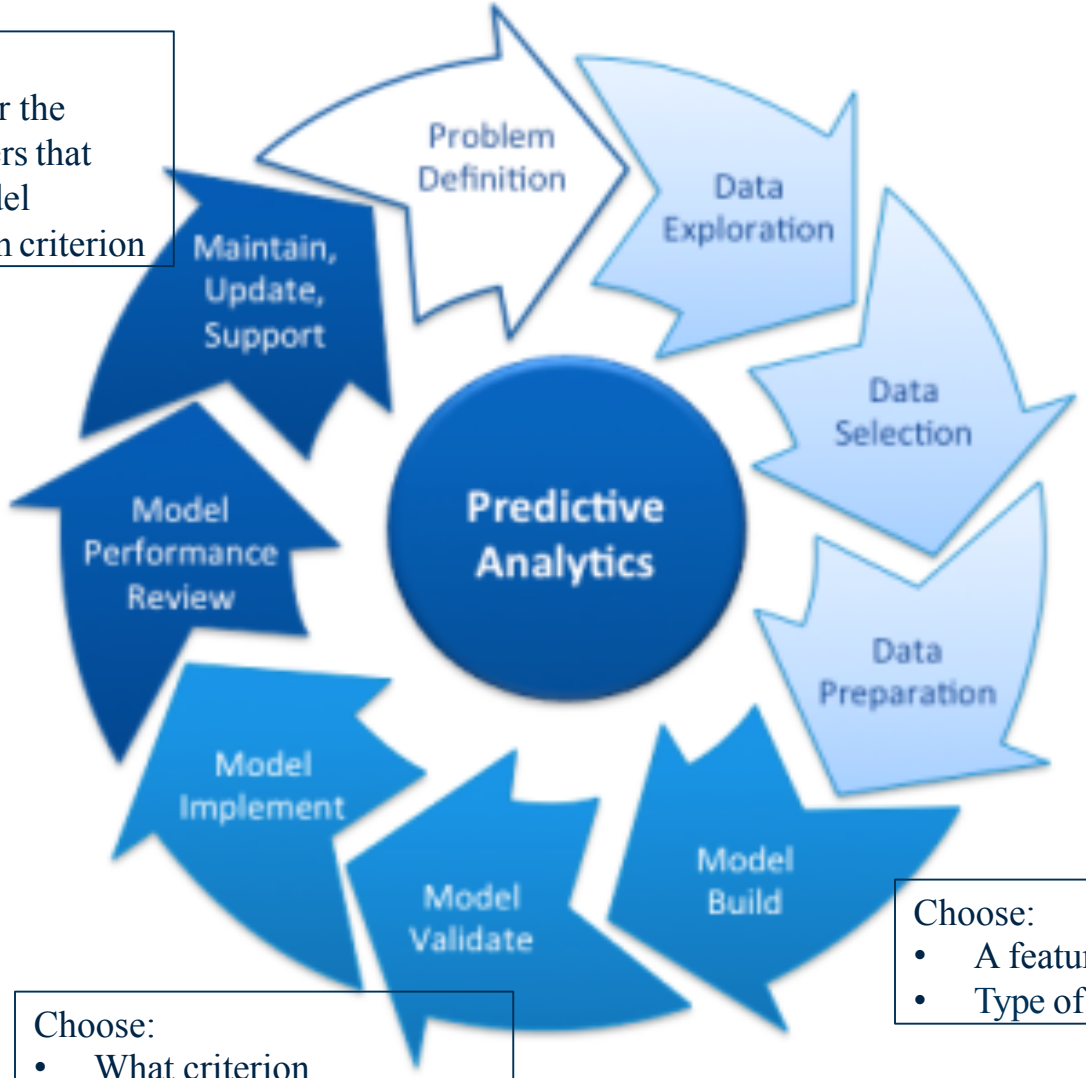
- **Finding clusters of similar users (clustering)**
- **Detecting abnormal server access patterns (unsupervised outlier detection)**





Machine Learning methodology

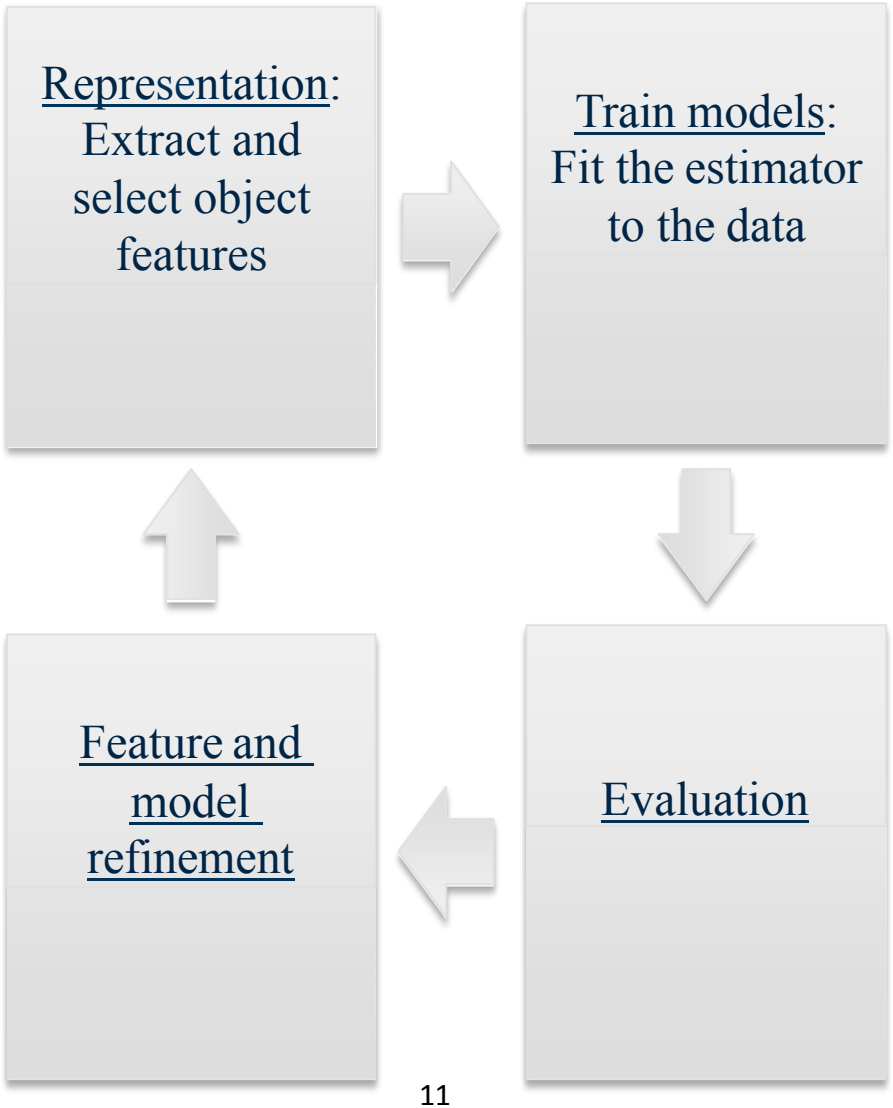
Choose:
- How to search for the settings/parameters that give the best model for this evaluation criterion



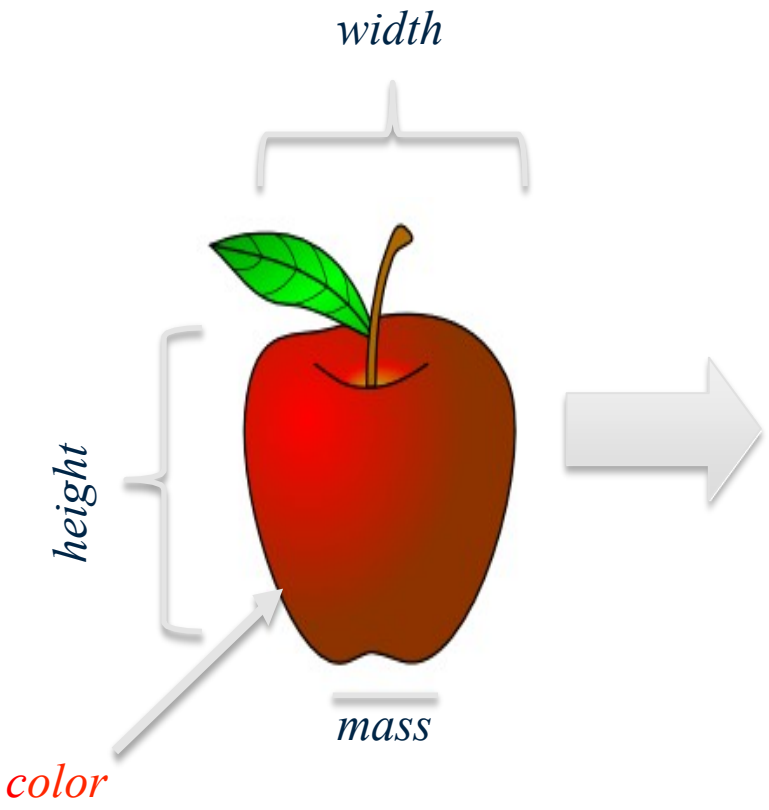
Choose:
- A feature representation
- Type of model to use

Choose:
- What criterion distinguishes good vs. bad models?

Represent / Train / Evaluate / Refine Cycle



Representing a piece of fruit as an array of features



1. Feature representation

Label information

(available in training data only)

Feature representation

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
18	1	apple	cripps_pink	162	7.5	7.1	0.83

2. Learning model

Classifier

Predicted class
(apple)

Python Tools for Machine Learning

scikit-learn: Python Machine Learning Library

- **scikit-learn Homepage**
  http://scikit-learn.org/
- **scikit-learn User Guide**
  http://scikit-learn.org/stable/user_guide.html
- **scikit-learn API reference**
  http://scikit-learn.org/stable/modules/classes.html
- **In Python, we typically import classes and functions we need like this:**

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```


scikit
learn

SciPy Library: Scientific Computing Tools

http://www.scipy.org/

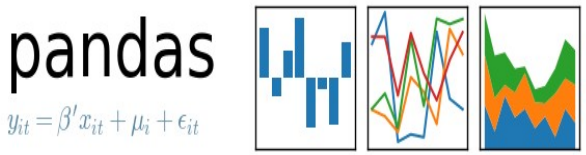
- **Provides a variety of useful scientific computing tools, including statistical distributions, optimization of functions, linear algebra, and a variety of specialized mathematical functions.**
- **With scikit-learn, it provides support for *sparse matrices*, a way to store large tables that consist mostly of zeros.**
- **Example import:** `import scipy as sp`

NumPy: Scientific Computing Library

http://www.numpy.org/

- **Provides fundamental data structures used by scikit-learn, particularly multi-dimensional arrays.**
- **Typically, data that is input to scikit-learn will be in the form of a NumPy array.**
- **Example import:** `import numpy as np`

Pandas: Data Manipulation and Analisys



/pandas.pydata.org/

- **Provides key data structures like `DataFrame`**
- **Also, support for reading/writing data in different formats**
- **Example import: `import pandas as pd`**

matplotlib and other plotting libraries

http://matplotlib.org/

- We typically use matplotlib's **pyplot** module:

```
import matplotlib.pyplot as plt
```

- We also sometimes use the **seaborn** visualization library (http://seaborn.pydata.org/)

```
import seaborn as sn
```

- And sometimes the **graphviz** plotting library:

```
import graphviz
```

Scikit-learn: Datasets

Para favorecer el aprendizaje sklearn viene con varios datasets de muestra con los que podemos jugar:

```
In [29]: from sklearn import datasets
         [x for x in dir(datasets) if 'load' in x]

Out[29]: ['load_boston',
          'load_diabetes',
          'load_digits',
          'load_filenames',
          'load_files',
          'load_iris',
          'load_lfw_pairs',
          'load_lfw_people',
          'load_linnerud',
          'load_mlcomp',
          'load_sample_image',
          'load_sample_images',
          'load_svmlight_file',
          'load_svmlight_files']
```

Scikit-learn: Datasets

Podemos cargar cualquiera de ellos con load_dataset(), y obtendremos un diccionario con distintas entradas, una de ellas DESCR que contiene la descripción del mismo.

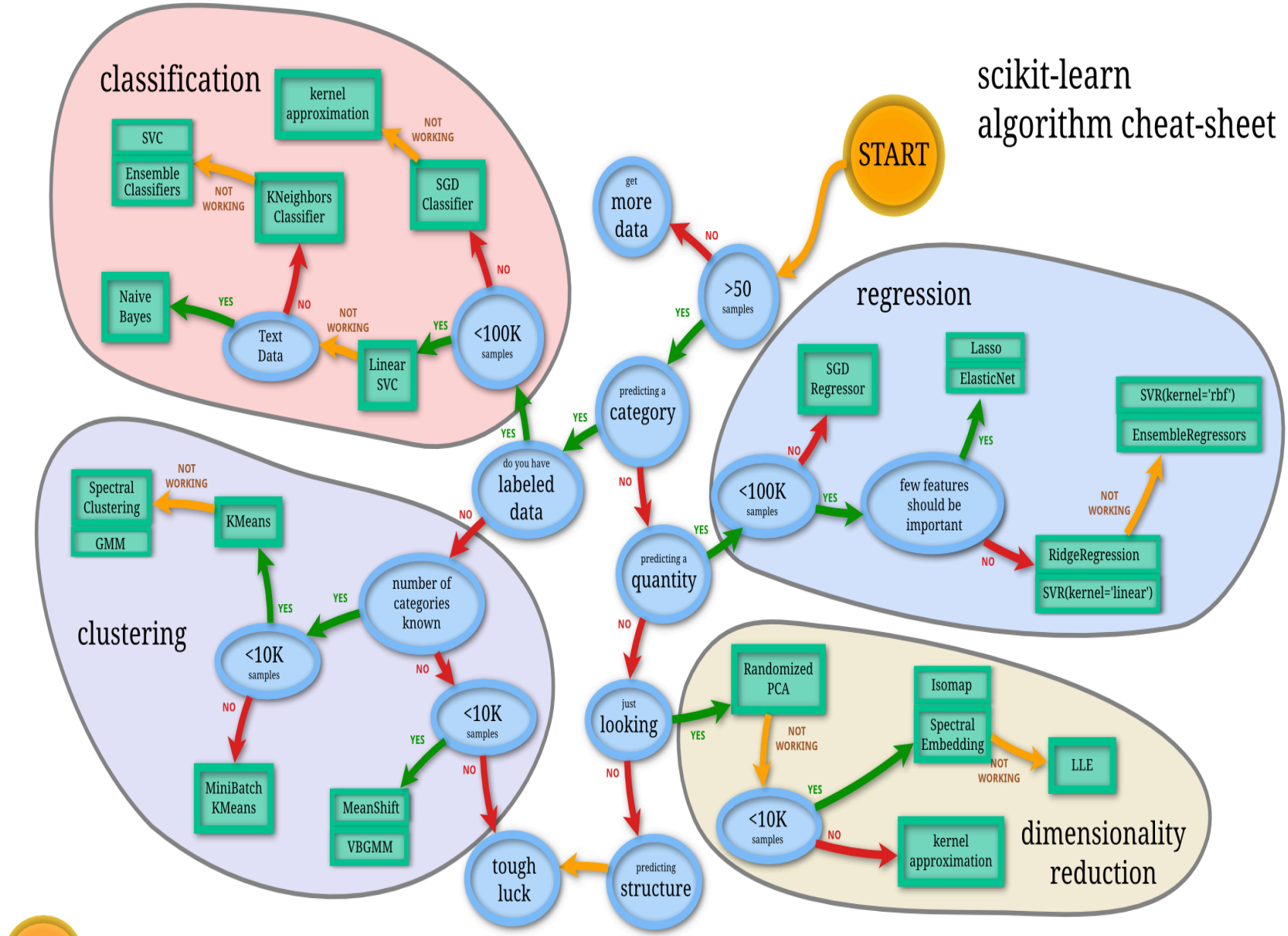
Además los datos suelen venir como “data” y la variable objetivo a predicir en ese dataset como “target”

```
In [7]: from sklearn import datasets
        boston = datasets.load_boston()

In [12]: print boston.DESCR
         Boston House Prices dataset

         Notes
         ------
         Data Set Characteristics:

             :Number of Instances: 506

             :Number of Attributes: 13 numeric/categorical predictive

             :Median Value (attribute 14) is usually the target
```

Scikit-learn: Datasets

Ejercicio:

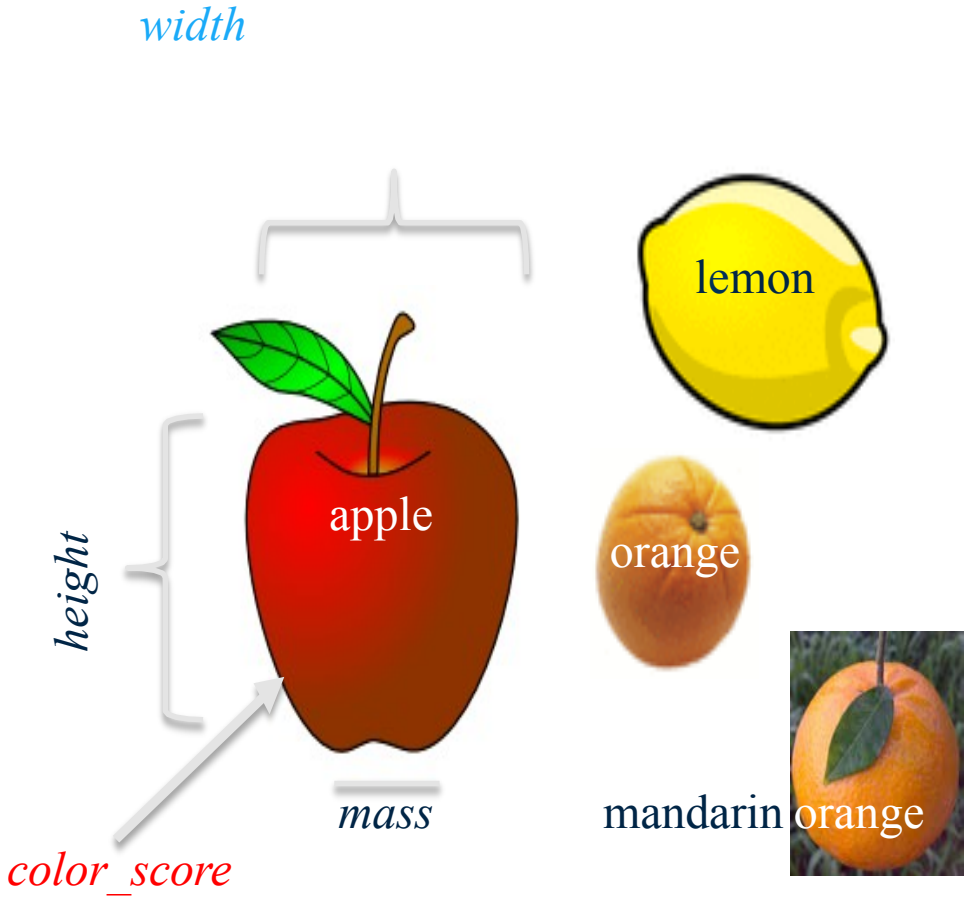
1. Carga el dataset de boston de los precios de las casas
2. Por defecto se habrá cargado tendremos .data como un ndarray, conviértelo en un pandas dataframe y asigna los nombres correctos a las columnas (.DESCR, .feature_names)
3. Crea una nueva columna que sea con el precio de la vivienda (.target)
4. Calcula los descriptivos de las variables, en particular calcula los cuartiles del precio de las viviendas


scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
NO
Linear SVC
NOT WORKING
Text Data
NO
Naive Bayes
YES
KNeighbors Classifier
NOT WORKING
Ensemble Classifiers
SVC
SGD Classifier
NOT WORKING
kernel approximation
classification
NO
predicting a quantity
YES
<100K samples
NO
SGD Regressor
YES
few features should be important
YES
Lasso
ElasticNet
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap
Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
tough luck
NO
number of categories known
YES
<10K samples
NO
MiniBatch KMeans
YES
<10K samples
NO
MeanShift
VBGMM
YES
KMeans
NOT WORKING
Spectral Clustering
GMM
clustering
Back

22

An Example Machine Learning Problem

The Fruit Dataset



	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69
16	1	apple	golden_delicious	156	7.6	7.5	0.67

`fruit_data_with_colors.txt`

Credit: Original version of the fruit dataset created by Dr. Iain Murray, Univ. of Edinburgh

In [27]:
```
fruits
```

Out[27]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69
16	1	apple	golden_delicious	156	7.6	7.5	0.67

In [88]:
```
fruits.shape
```

Out[88]: (59, 7)

The input data as a table

Each row corresponds to a single data instance (sample)

	fruit_label	ruit_name	fruit_subtype	mass	width	height	color_score
0	1	pple	granny_smith	192	8.4	7.3	0.55
1	1	pple	granny_smith	180	8.0	6.8	0.59
2	1	pple	granny_smith	176	7.4	7.2	0.60
3	2	nandarin	mandarin	86	6.2	4.7	0.80
4	2	nandarin	mandarin	84	6.0	4.6	0.79
5	2	nandarin	mandarin	80	5.8	4.3	0.77
6	2	nandarin	mandarin	80	5.9	4.3	0.81
7	2	nandarin	mandarin	76	5.8	4.0	0.81
8	1	pple	braeburn	178	7.1	7.8	0.92
9	1	pple	braeburn	172	7.4	7.0	0.89
10	1	pple	braeburn	166	6.9	7.3	0.93
11	1	pple	braeburn	172	7.1	7.6	0.92
12	1	pple	braeburn	154	7.0	7.1	0.88
13	1	pple	golden_delicious	164	7.3	7.7	0.70
14	1	pple	golden_delicious	152	7.6	7.3	0.69
15	1	pple	golden_delicious	156	7.7	7.1	0.69
16	1	pple	golden_delicious	156	7.6	7.5	0.67
17	1	pple	golden_delicious	168	7.5	7.6	0.73
18	1	pple	cripps_pink	162	7.5	7.1	0.83
19	1	pple	cripps_pink	162	7.4	7.2	0.85
20	1	pple	cripps_pink	160	7.5	7.5	0.86

These four columns contain the features of each data instance (sample)

The `fruit_label` column contains the label for each data instance (sample)

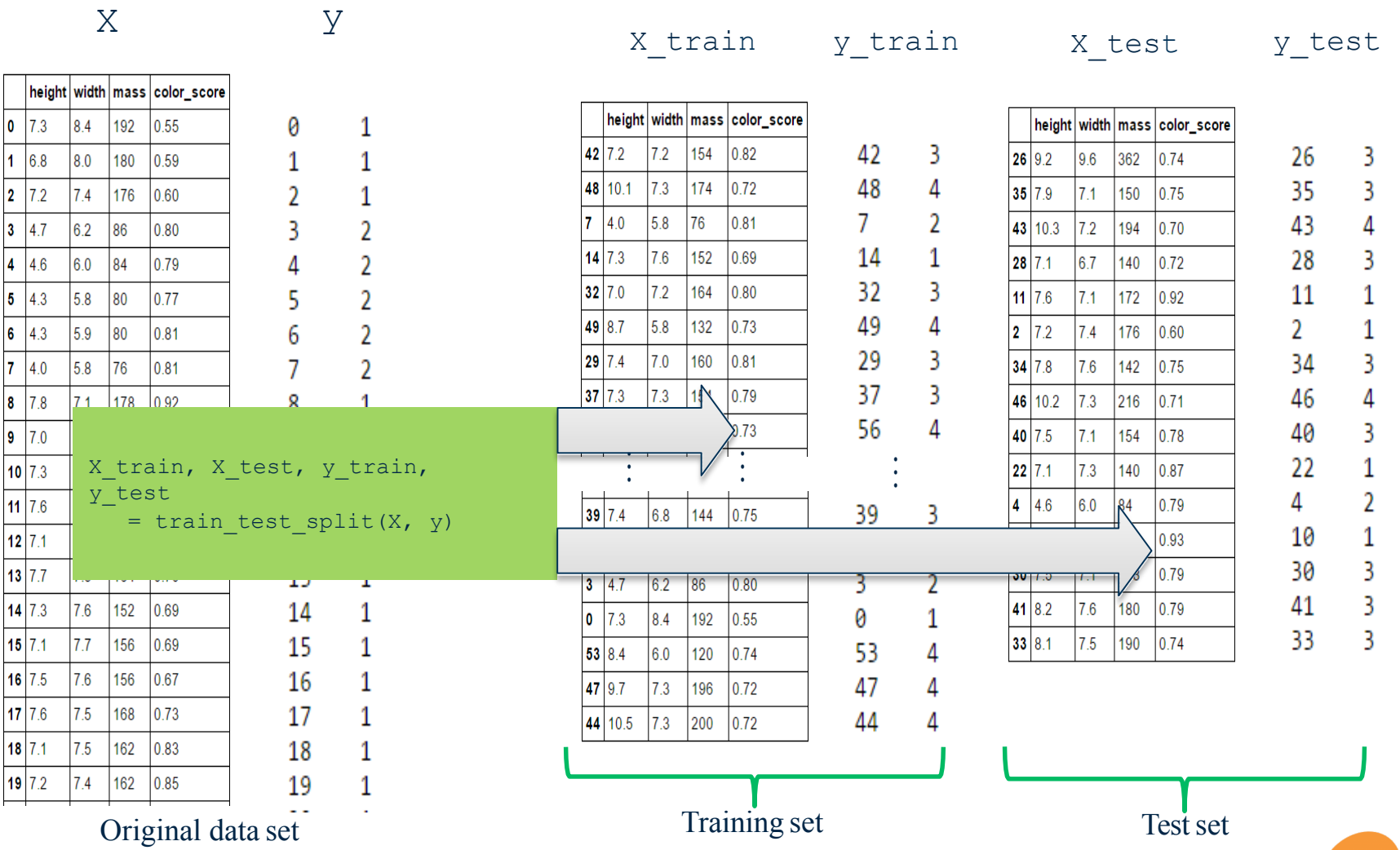
Some reasons why looking at the data initially is important

- Inspecting feature values may help identify what cleaning or preprocessing still needs to be done once you can see the range or distribution of values that is typical for each attribute.
- You might notice missing or noisy data, or inconsistencies such as the wrong data type being used for a column, incorrect units of measurements for a particular column, or that there aren't enough examples of a particular class.
- You may realize that your problem is actually solvable without machine learning.

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	192
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	apple	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn		7.4	7.0	0.89
10	1	apple	braeburn		6.9	7.3	0.93
11	1	apple	braeburn		7.1	7.6	0.92
12	1	apple	braeburn		7.0	7.1	0.88
13	1	apple	golden_delicious	194	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69

Examples of incorrect or missing feature values

Creating Training and Testing Sets



```
In [88]: fruits.shape
Out[88]: (59, 7)

In [92]: X_train.shape
Out[92]: (44, 4)

In [93]: X_test.shape
Out[93]: (15, 4)

In [94]: y_train.shape
Out[94]: (44,)

In [95]: y_test.shape
Out[95]: (15,)
```

```
In [96]: X_train
```

Out[96]:

	height	width	mass	color_score
42	7.2	7.2	154	0.82
48	10.1	7.3	174	0.72
7	4.0	5.8	76	0.81
14	7.3	7.6	152	0.69
32	7.0	7.2	164	0.80
49	8.7	5.8	132	0.73
29	7.4	7.0	160	0.81
37	7.3	7.3	154	0.79
56	8.1	5.9	116	0.73
18	7.1	7.5	162	0.83
55	7.7	6.3	116	0.72
27	9.2	7.5	204	0.77
15	7.1	7.7	156	0.69
5	4.3	5.8	80	0.77
31	8.0	7.8	210	0.82
16	7.5	7.6	156	0.67

```
In [98]: y_train
Out[98]: 42    3
         48    4
         7     2
         14    1
         32    3
         49    4
         29    3
         37    3
         56    4
         18    1
         55    4
         27    3
         15    1
         5     2
         31    3
         16    1
         50    4
         20    1
         51    4
         8     1
         13    1
         25    3
         17    1
         58    4
         57    4
         52    4
         38    3
         1     1
         12    1
         45    4
         24    3
         6     2
```

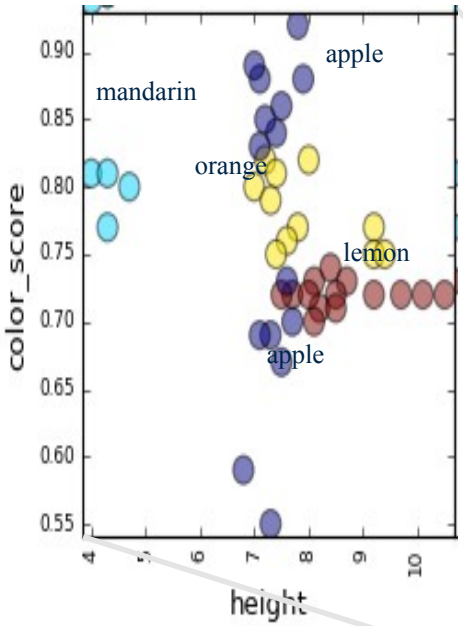
```
In [97]: X_test
```

Out[97]:

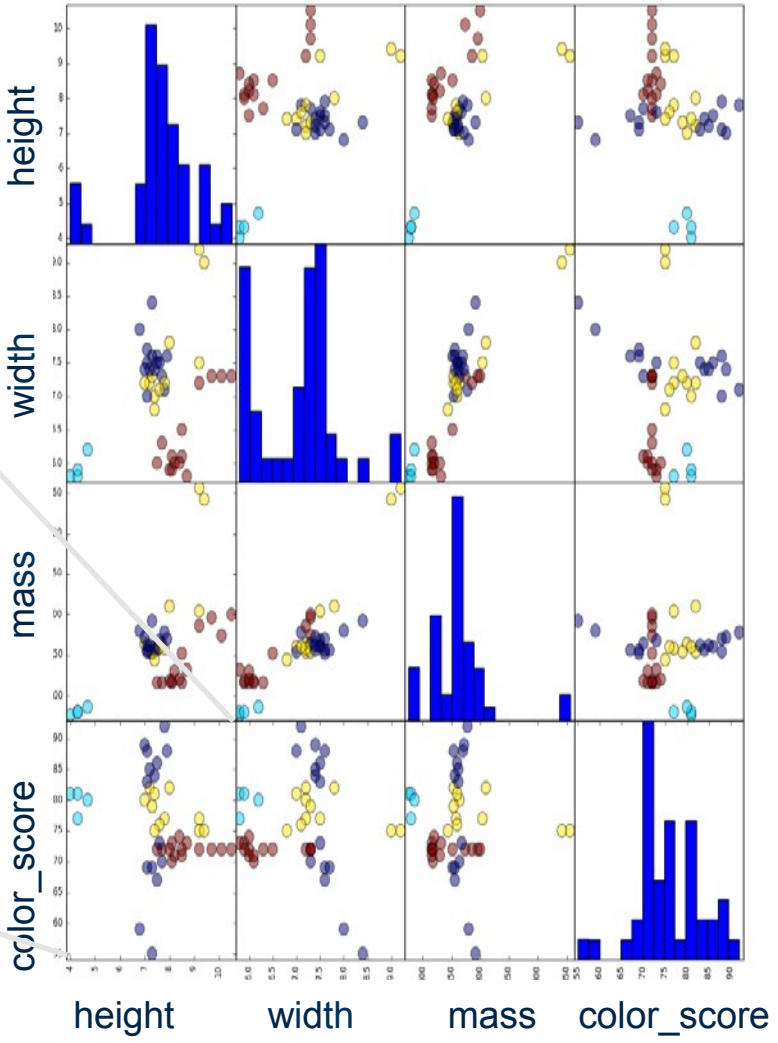
	height	width	mass	color_score
26	9.2	9.6	362	0.74
35	7.9	7.1	150	0.75
43	10.3	7.2	194	0.70
28	7.1	6.7	140	0.72
11	7.6	7.1	172	0.92
2	7.2	7.4	176	0.60
34	7.8	7.6	142	0.75
46	10.2	7.3	216	0.71
40	7.5	7.1	154	0.78
22	7.1	7.3	140	0.87
4	4.6	6.0	84	0.79
10	7.3	6.9	166	0.93
30	7.5	7.1	158	0.79
41	8.2	7.6	180	0.79
33	8.1	7.5	190	0.74

```
In [99]: y_test
Out[99]: 26    3
         35    3
         43    4
         28    3
         11    1
         2     1
         34    3
         46    4
         40    3
         22    1
         4     2
         10    1
         30    3
         41    3
         33    3
```

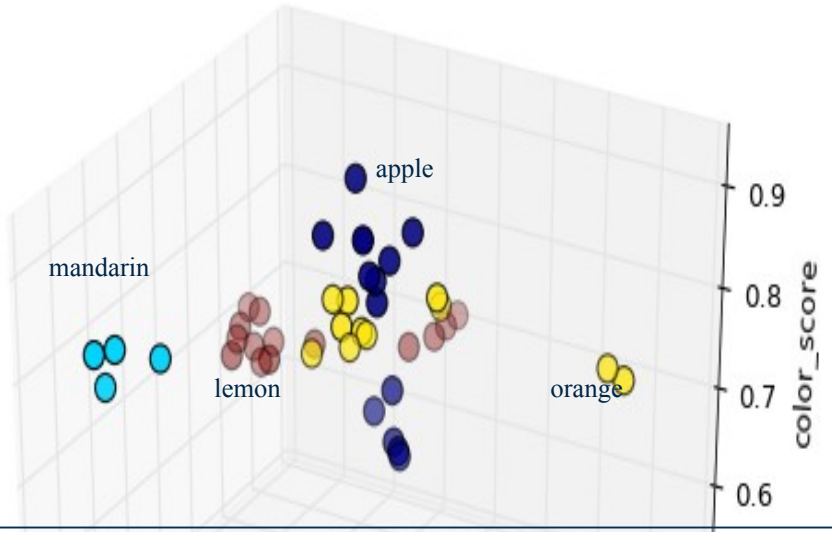
A pairwise feature scatterplot visualizes the data using all possible pairs of features, with one scatterplot per feature pair, and histograms for each feature along the diagonal



Individual scatterplot plotting all fruits by their **height** and **color_score**. Colors represent different fruit classes.



A three-dimensional feature scatterplot



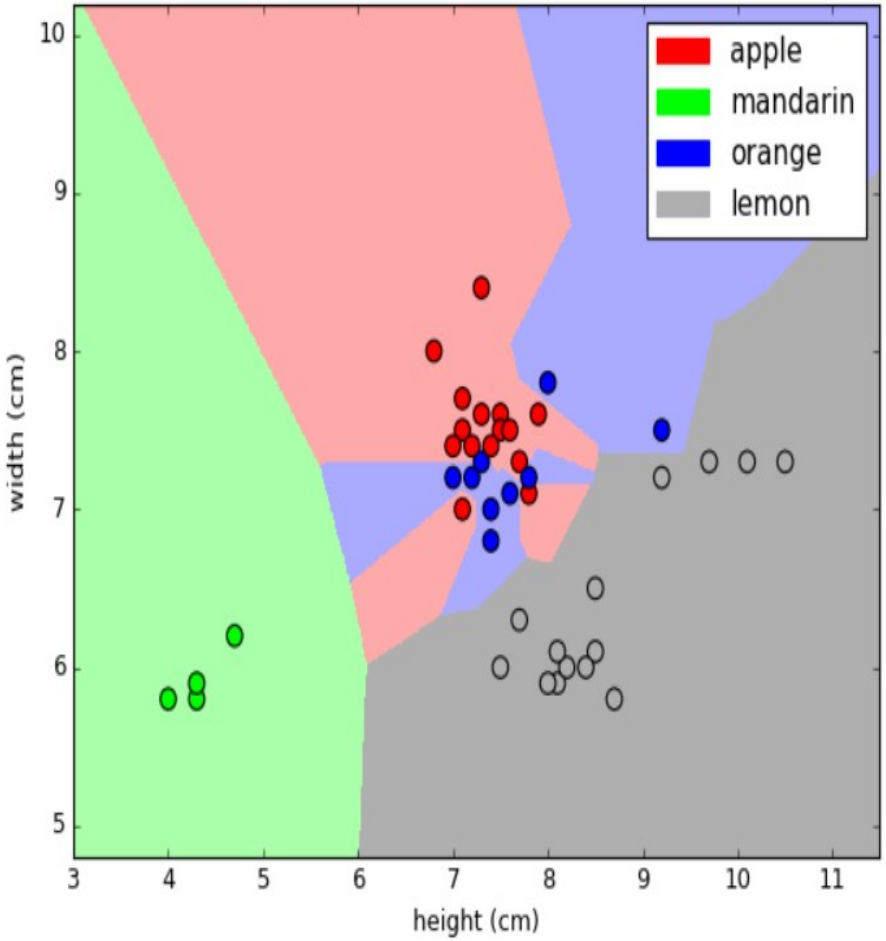

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax.scatter(X_train['width'], X_train['height'], X_train['color_score'], c = y_train, marker = 'o', s=100)
ax.set_xlabel('width')
ax.set_ylabel('height')
ax.set_zlabel('color_score')
plt.show()
```

The k-Nearest Neighbor (k-NN) Classifier Algorithm

Given a training set X_train with labels y_train, and given a new instance x_test to be classified:

1. **Find the most similar instances (let's call them X_NN) to x_test that are in X_train.**
2. **Get the labels y_NN for the instances in X_NN**
3. **Predict the label for x_test by combining the labels y_NN e.g. simple majority vote**

A visual explanation of k-NN classifiers



Fruit dataset
Decision boundaries
with k = 1

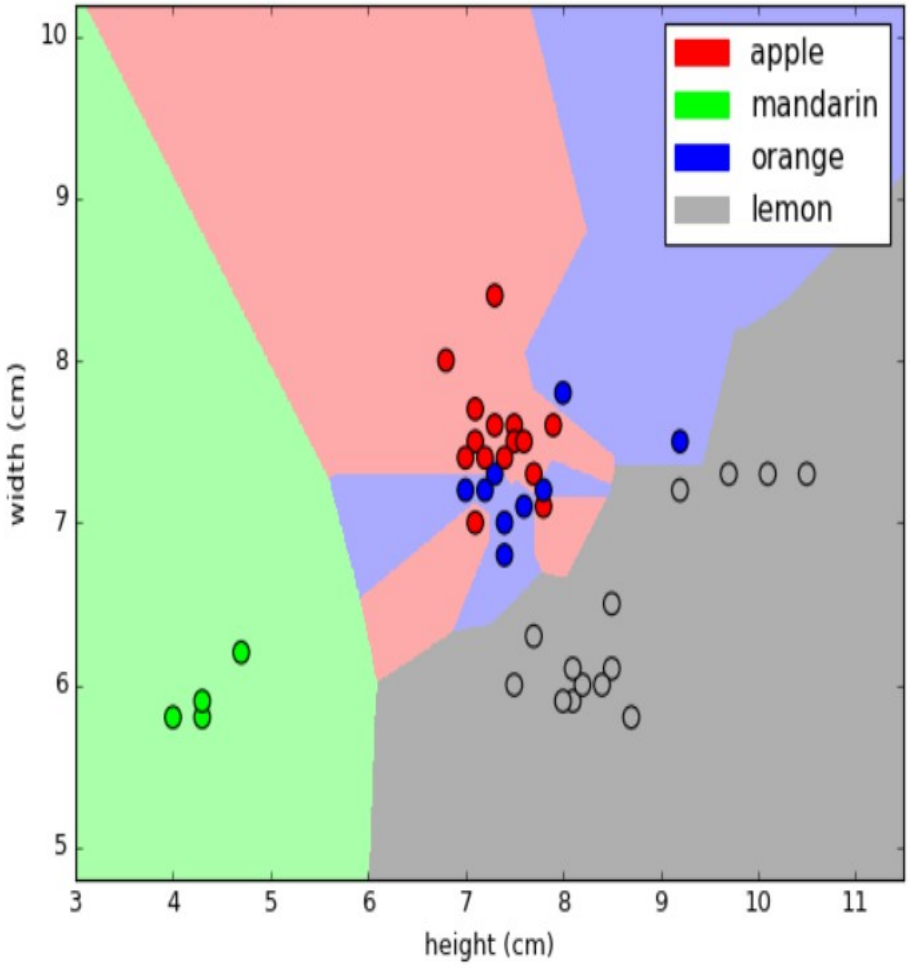
A nearest neighbor algorithm needs four things specified

1. A distance metric
2. How many 'nearest' neighbors to look at?
3. Optional weighting function on the neighbor points
4. Method for aggregating the classes of neighbor points

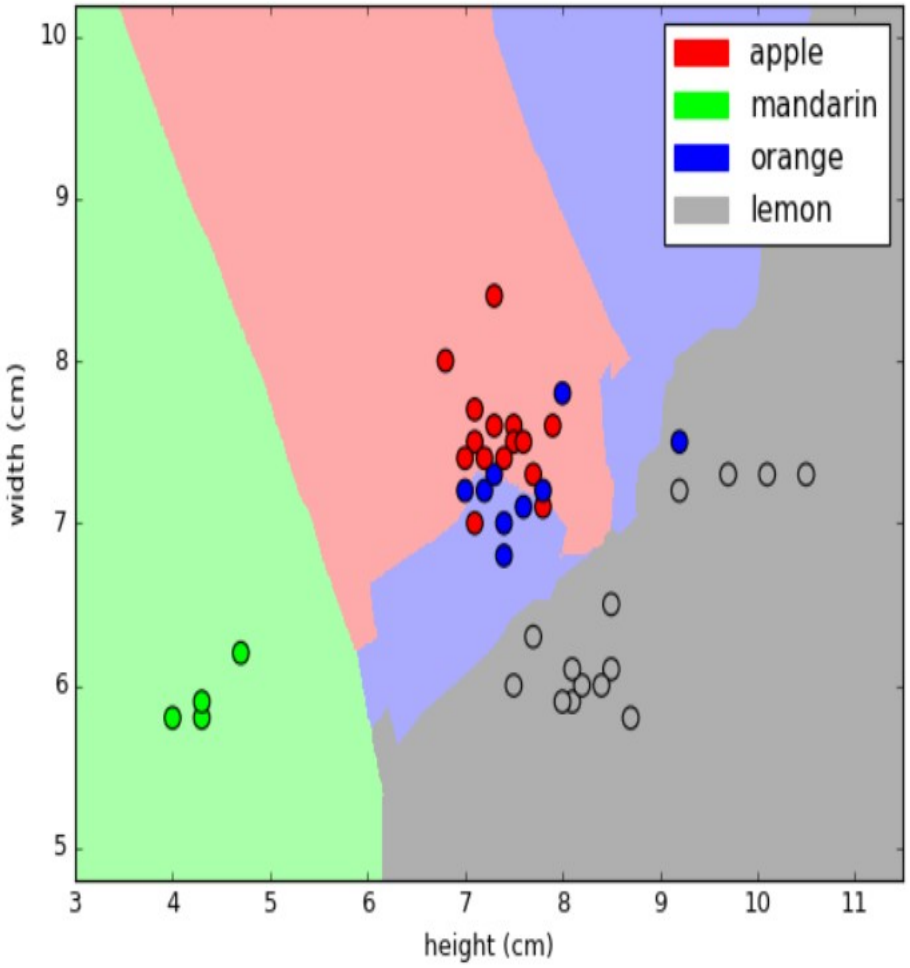
A nearest neighbor algorithm needs four things specified

1. A distance metric
   Typically Euclidean (Minkowski with p = 2)
2. How many 'nearest' neighbors to look at?
   e.g. five
3. Optional weighting function on the neighbor points
   Ignored
4. How to aggregate the classes of neighbor points
   Simple **majority vote**
   (Class with the most representatives among nearest neighbors)

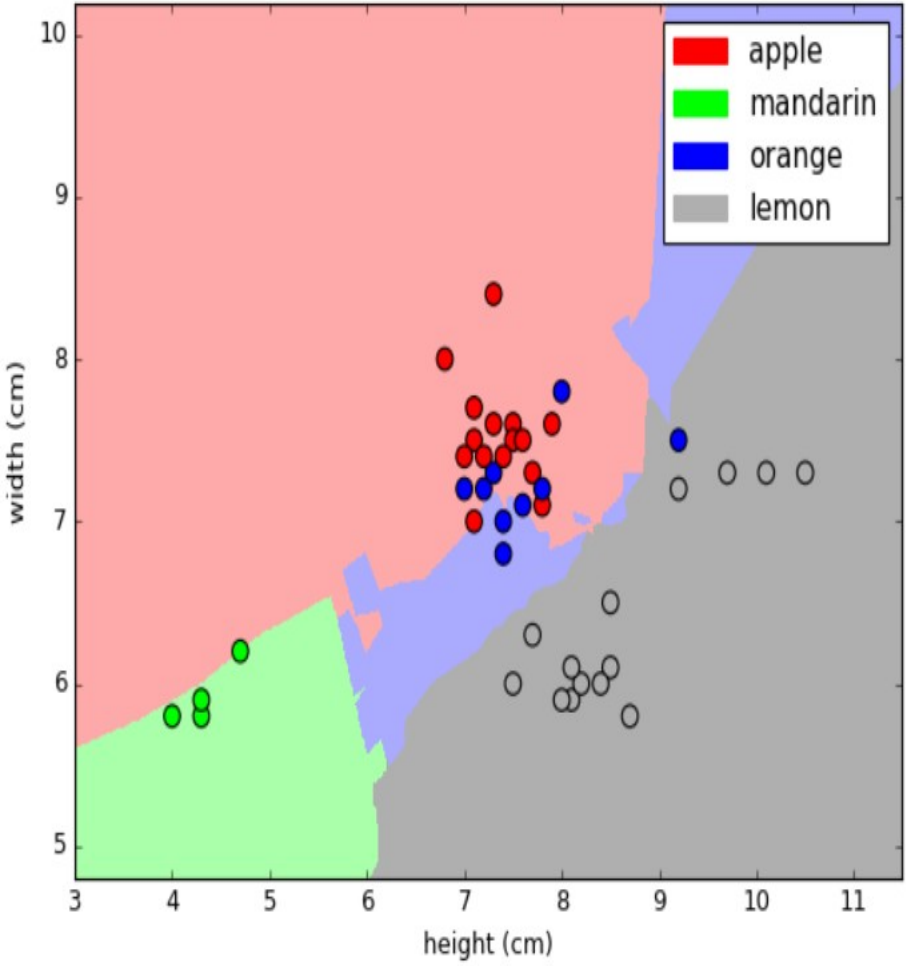
K-nearest neighbors (k=1) for fruit dataset



K-nearest neighbors (k=5) for fruit dataset



K-nearest neighbors (k=10) for fruit dataset

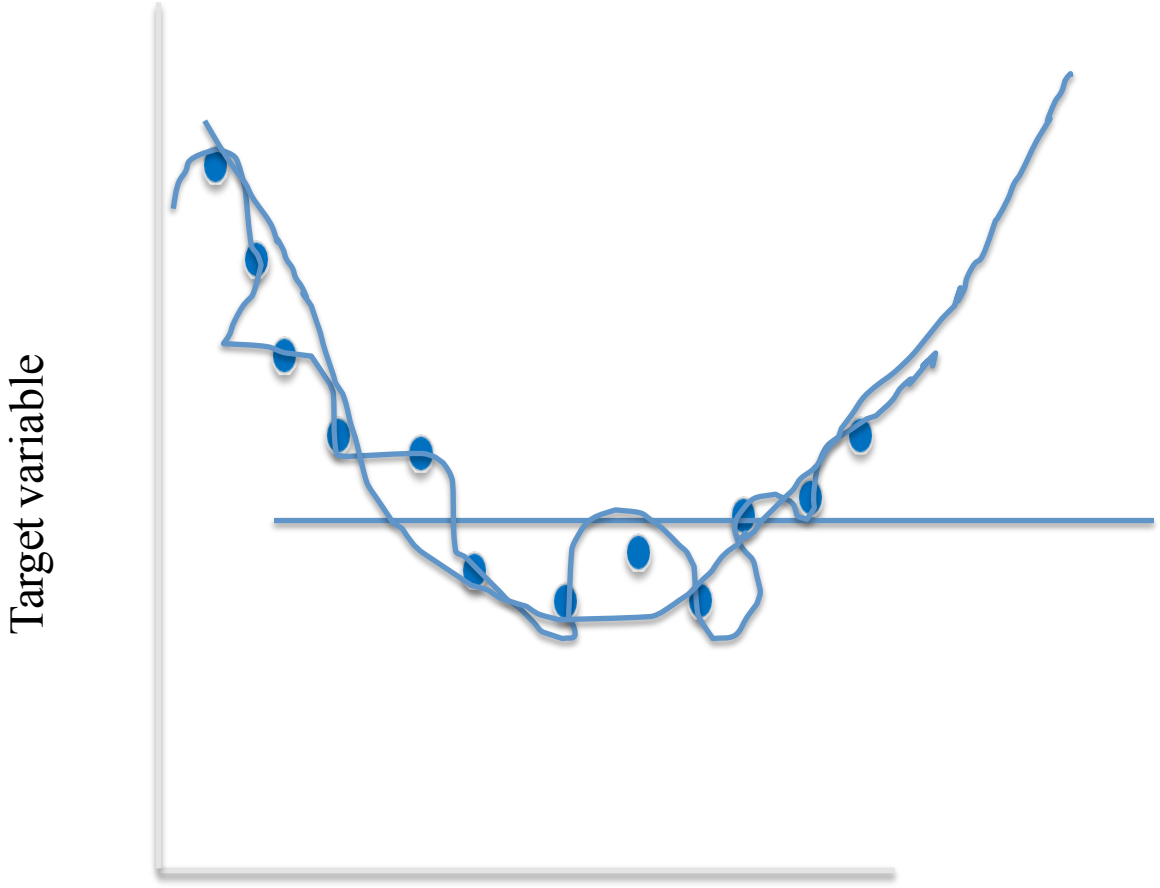


Overfitting

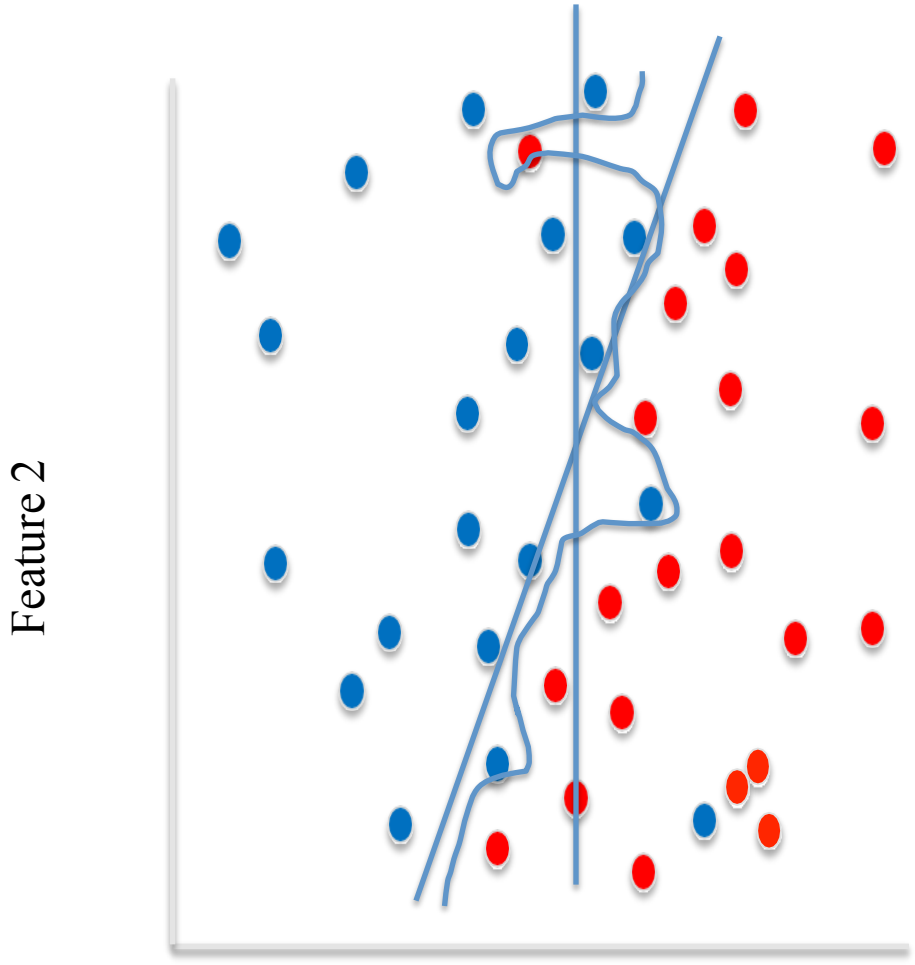
Generalization, Overfitting, and Underfitting

- **Generalization ability refers to an algorithm's ability to give accurate predictions for new, previously unseen data.**
- **Assumptions:**
  - *Future unseen data (test set) will have the same properties as the current training sets.*
  - *Thus, models that are accurate on the training set are expected to be accurate on the test set.*
  - *But that may not happen if the trained model is tuned too specifically to the training set.*
- **Models that are too complex for the amount of training data available are said to overfit and are not likely to generalize well to new examples.**
- **Models that are too simple, that don't even do well on the training data, are said to underfit and also not likely to generalize well.**

Overfitting in regression



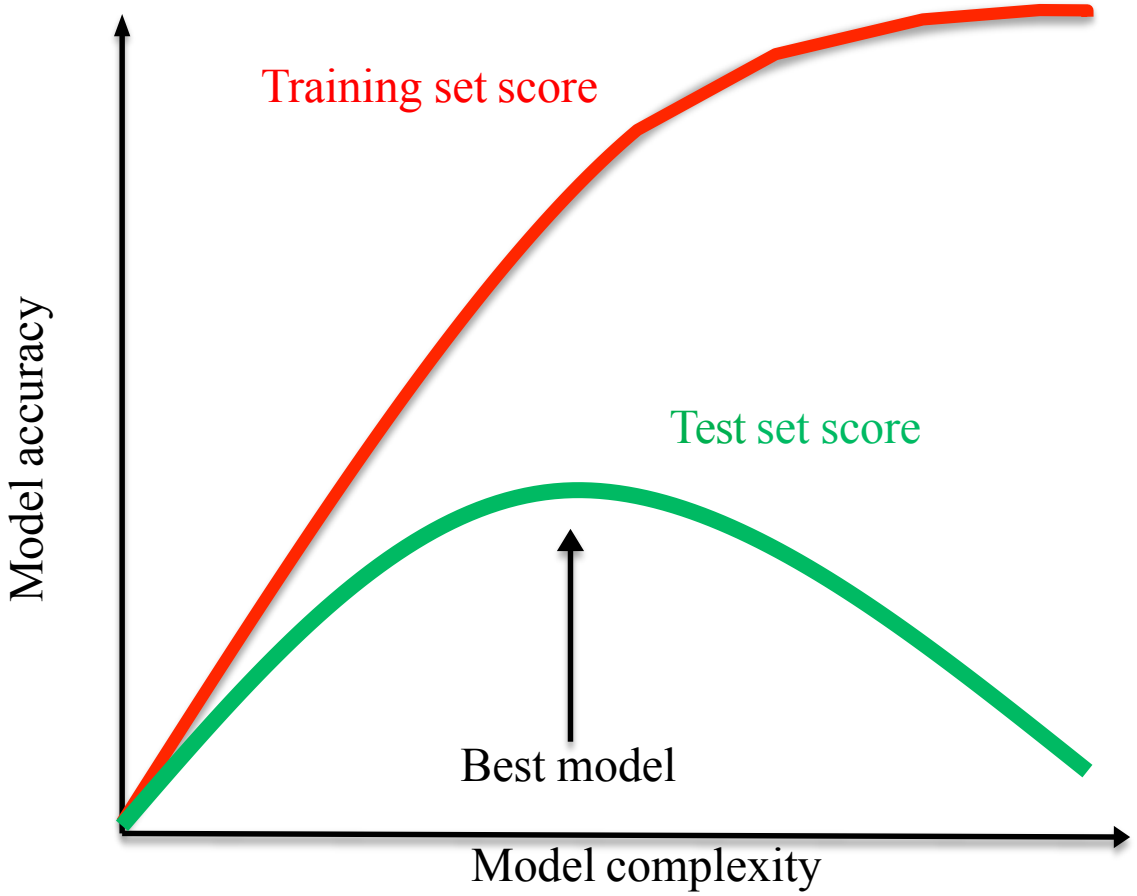
Overfitting in classification



The R^2 ("r-squared") Regression Score

- **Measures how well a prediction model for regression fits the given data.**
- **The score is between 0 and 1:**
  - *A value of 0 corresponds to a constant model that predicts the mean value of all training target values.*
  - *A value of 1 corresponds to perfect prediction*
- **Also known as "coefficient of determination"**

The relationship between model complexity and training/test performance



Regression

Sklearn: Regresión

Las técnicas de regresión nos permiten estimar un valor continuo a partir de una serie de variables predictoras.

En función del algoritmo que utilicemos estaremos construyendo una función lineal o no lineal para minimizar una función de costes sobre el conjunto de datos de entrenamiento (típicamente la función de error acumulado entre nuestras prediciones y los valores.

La mayoría de los algoritmos en sklearn tienen una interfaz similar, vamos a ver un ejemplo con una regresión lineal para estimar el precio de la vivienda con el dataset de boston.

Sklearn: Separar conjuntos de entrenamiento y test

Hasta ahora en los ejemplos hemos estado utilizando todo el mismo dataset para entrenar y para medir sobre él el performance del modelo, en un caso real querremos separar un conjunto de test haremos que no participe en el entrenamiento y evaluaremos el performance sobre este subconjunto.

El módulo cross-validation además de utilidades para evaluar el ajuste de distintos estimadores, proporciona una función para separar de un modo rápido y sencillo un dataset en 2 conjuntos, uno para entrenamiento y otro para validación.

```
In [104]: from sklearn.cross_validation import train_test_split

          X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, test_size=0.3)

In [105]: print len(X_train)
          print len(X_test)

          354
          152
```

Linear Models

- **A linear model is a sum of weighted variables that predicts a target output value given an input data instance**

Predicted output:

$$\hat{y} = \widehat{w_0} x_0 + \widehat{w_1} x_1 + \cdots \widehat{w_n} x_n + \hat{b}$$

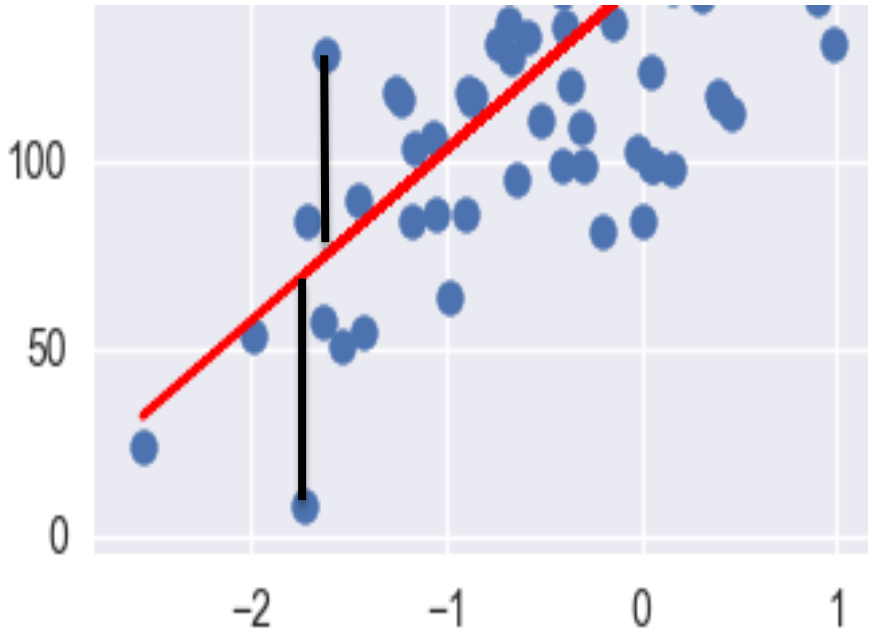
Parameters to estimate:

$\hat{\boldsymbol{w}} = (\widehat{w_0}, \cdots, \widehat{w_n})$: feature weights/ model coefficients

$\hat{\boldsymbol{b}}$: constant bias term / intercept

Least-Squares Linear Regression ("Ordinary least-squares")

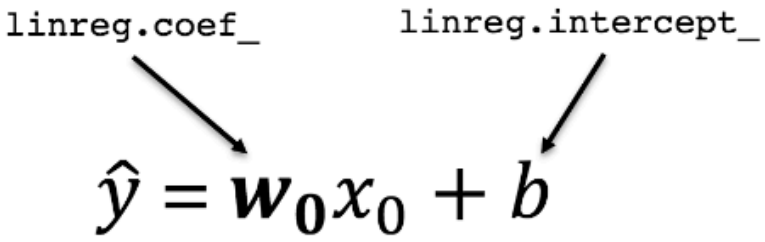
- Finds *w* and *b* that minimizes the mean squared error of the linear model: the sum of squared differences between predicted target and actual target values.
- No parameters to control model complexity.

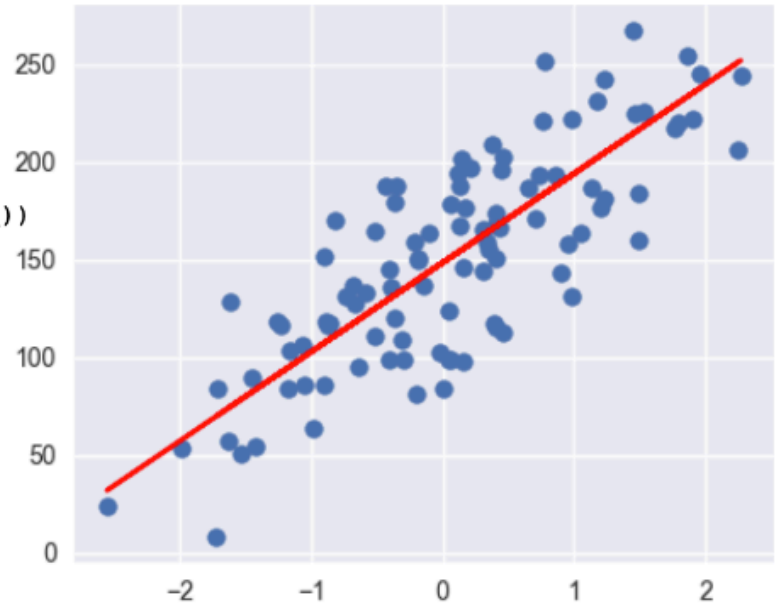


Least-Squares Linear Regression in Scikit-Learn

```
linear model coeff (w): [ 45.70870465]
linear model intercept (b): 148.44575345658873
R-squared score (training): 0.679
R-squared score (test): 0.492
```

```
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test =
      train_test_split(X_R1, y_R1, random_state = 0)

linreg = LinearRegression().fit(X_train, y_train)

print("linear model intercept (b): {}".format(linreg.intercept_))
print("linear model coeff (w): {}".format(linreg.coef_))
```





Sklearn: Regresión

Lo primero es instanciar el modelo que queremos utilizar

```
In [42]: from sklearn.linear_model import LinearRegression

         # Instancio mi regresion
         boston_houseprice_lm = LinearRegression()
```

A continuación ya podemos entrenar nuestro modelo, independientemente del modelo, se utilizará el método modelo.fit(X, y), donde X es el conjunto de variables e y contiene el valor a predecir.

```
In [57]: # Entreno con mi conjunto de entrenamiento (imaginemos que todo el dataset)
         boston_houseprice_lm.fit(boston_df[boston_df.columns[:-1]], boston_df['VAL'])

Out[57]: LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
```

Sklearn: Regresión

Podemos ver los coeficientes resultado del entrenamiento:

```
In [64]: pd.DataFrame({'columns': boston_df.columns[:-1], 'coefs': boston_houseprice_lm.coef_})
```

Out[64]:

	coefs	columns
0	-0.107171	CRIM
1	0.046395	ZN
2	0.020860	INDUS
3	2.688561	CHAS
4	-17.795759	NOX
5	3.804752	RM
6	0.000751	AGE
7	-1.475759	DIS
8	0.305655	RAD
9	-0.012329	TAX
10	-0.953464	PTRATIO
11	0.009393	B
12	-0.525467	LSTAT

Sklearn: Regresión

Y utilizar el modelo para predecir valores mediante el metodo predict(X)

```
In [67]: boston_df['VAL PRED'] = boston_houseprice_lm.predict(boston_df[boston_df.columns[:-1]])

In [69]: boston_df.head(3)
```

Out[69]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	VAL	VAL PRED
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	30.008213
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	25.029861
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	30.570232

Ejercicio:

1. Calcula con el dataset de Boston el error (absoluto) para cada predicción
2. Calcula el error absoluto máximo, mínimo y el medio
3. Dibuja un gráfico de dispersión con el valor real en el eje X y la predicción en el Y (pista: kind='scatter', x='COL_X', y='COL_Y'
4. Dibuja un gráfico de dispersión ERR vs CRIM
5. Repite el ejercicio con una regresión de tipo Lasso (tb en linearmodels de sklearn)

Sklearn: Regresión

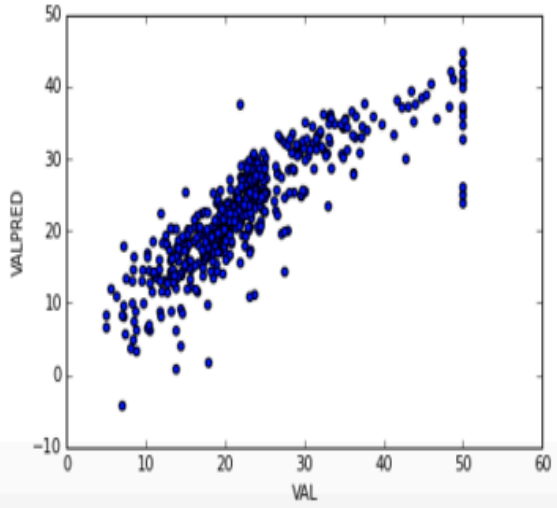
```
In [81]: boston_df['ERRORVAL']= abs(boston_df.VAL - boston_df.VALPRED)
```

```
In [93]: boston_df.head(2)
```

Out[93]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	VAL	VALPRED	ERRORVAL
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.9	4.98	24.0	30.008213	6.008213
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	25.029861	3.429861

```
In [90]: %matplotlib inline
         import matplotlib.pyplot as plt

         boston_df.plot(kind='scatter', x='VAL', y='VALPRED')
         plt.show()
```



54

Ridge Regression

- **Ridge regression learns *w*, *b* using the same least-squares criterion but adds a penalty for large variations in *w* parameters**

$$RSS_{RIDGE}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} w_j^2$$

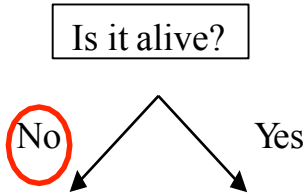
- **Once the parameters are learned, the ridge regression prediction formula is the same as ordinary least-squares.**
- **The addition of a parameter penalty is called regularization. Regularization prevents overfitting by restricting the model, typically to reduce its complexity.**
- **Ridge regression uses L2 regularization: minimize sum of squares of *w* entries**
- **The influence of the regularization term is controlled by the αparameter.**
- **Higher alpha means more regularization and simpler models.**

Ejercicio

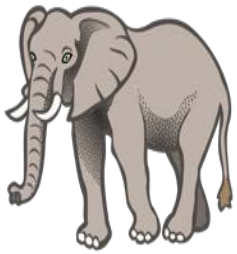
Utilizar una regresion Ridge, Lasso y comparar con la lineal

Decision Tree

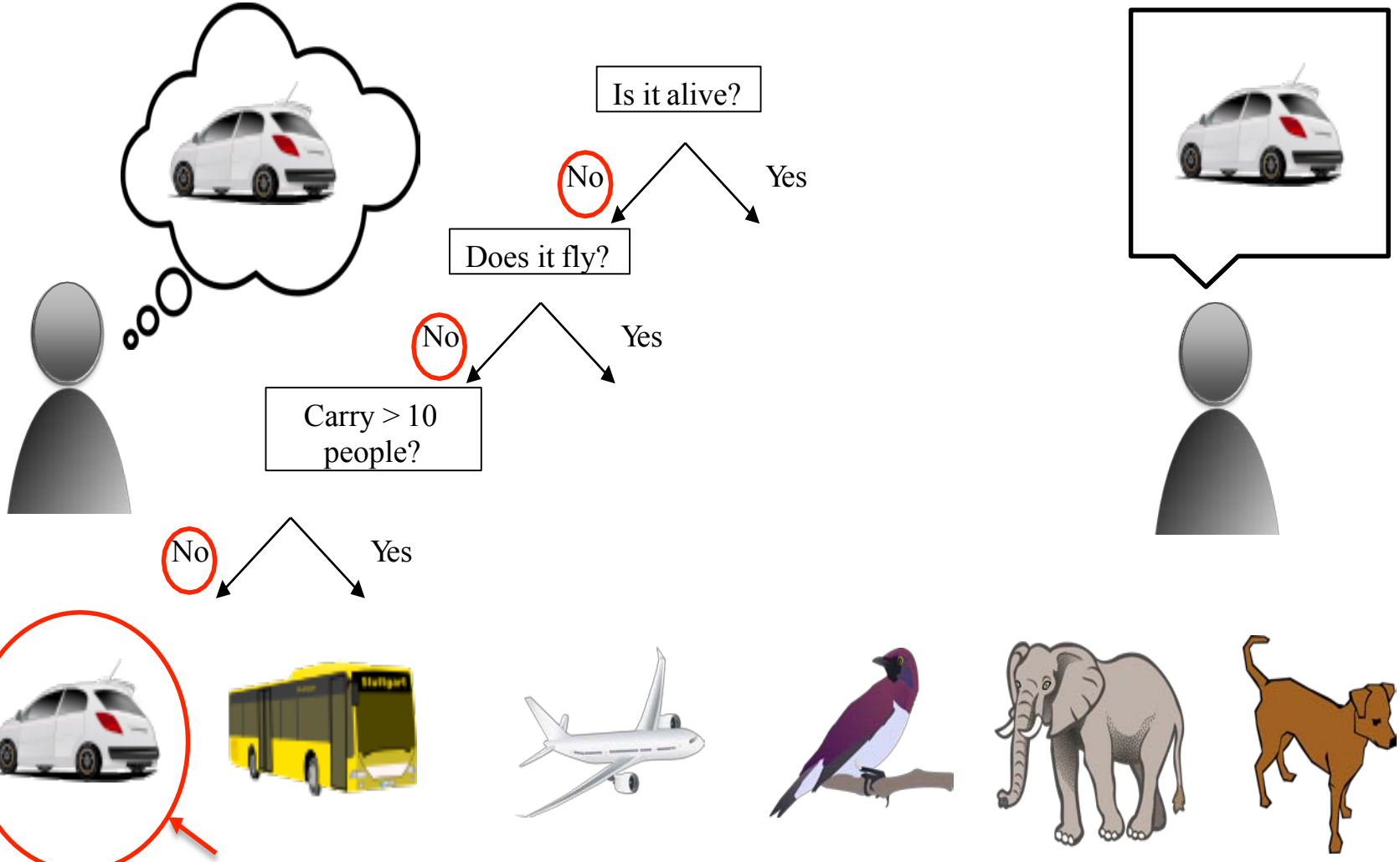
Decision Tree Example



Objects with Alive == No



Decision Tree Example



Objects with: Alive == No AND Fly == No AND Carry > 10 == No

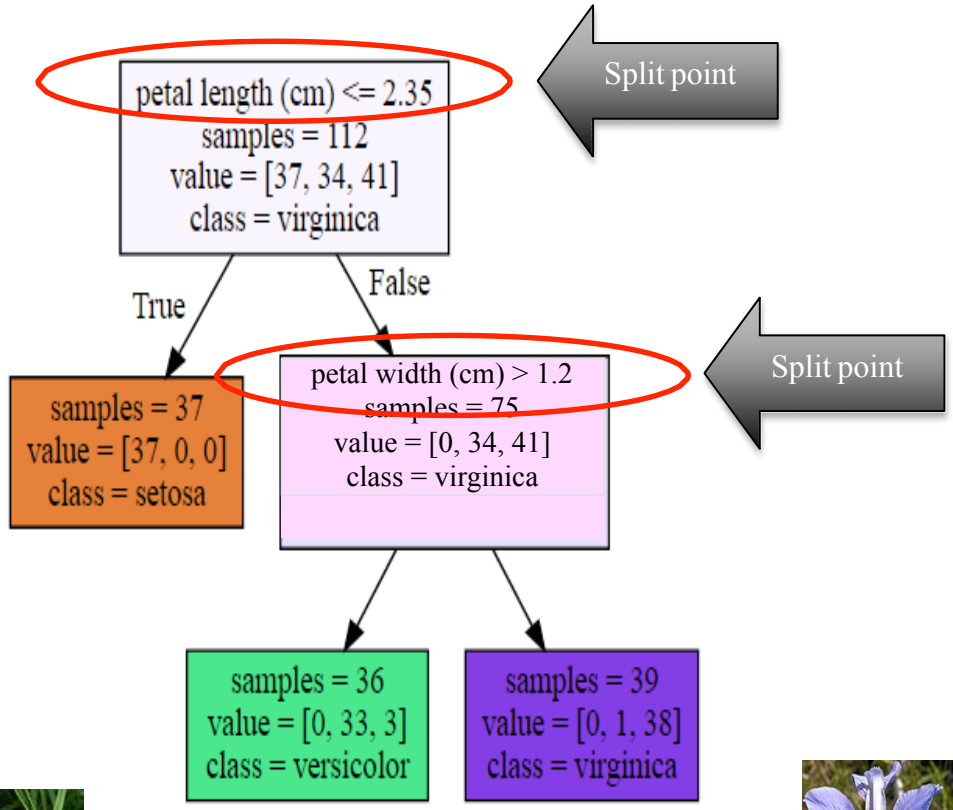
The Iris Dataset

150 flowers
3 species
50 examples/species

Iris setosa *Iris versicolor* *Iris virginica*

Photo credit: Radomił Binek via https://en.wikipedia.org/wiki/Iris_flower_data_set

Decision Tree Splits



samples at this leaf have:
petal length <= 2.35

setosa

versicolor

samples at this leaf have:
petal length > 2.35
AND petal width <= 1.2

samples at this leaf have:
petal length > 2.35
AND petal width > 1.2

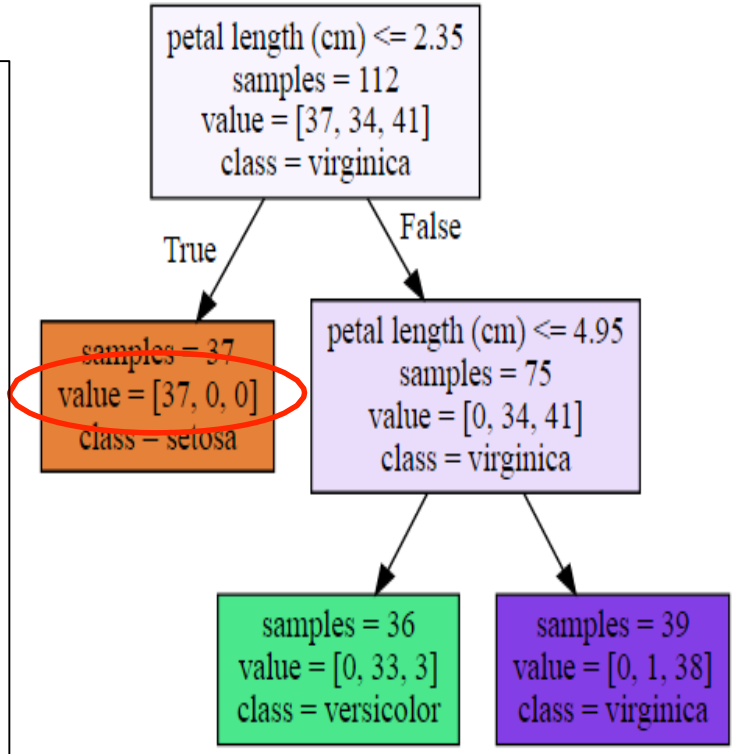
virginica

Informativeness of Splits

The *value* list gives the number of samples of each class that end up at this leaf node during training.

The iris dataset has 3 classes, so there are three counts.

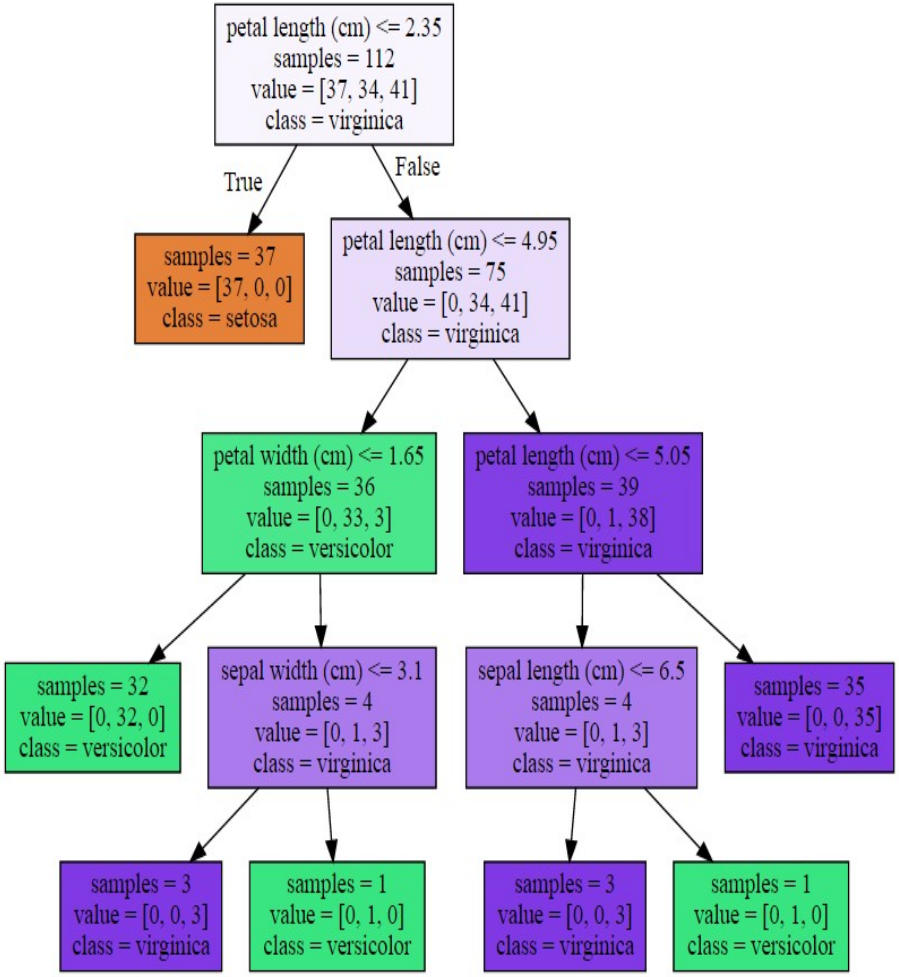
This leaf has 37 setosa samples, zero versicolor, and zero virginica samples.



This leaf has 0 setosa, 33 versicolor, and 3 virginica samples.

This leaf has 0 setosa, 1 versicolor, and 38 virginica samples.

Visualizing Decision Trees



See: `plot_decision_tree()` function in `adspy_shared_utilities.py` code

Decision Trees: Pros and Cons

Pros:

- **Easily visualized and interpreted.**
- **No feature normalization or scaling typically needed.**
- **Work well with datasets using a mixture of feature types (continuous, categorical, binary)**

Cons:

- **Even after tuning, decision trees can often still overfit.**
- **Usually need an ensemble of trees for better generalization performance.**

Classification

Sklearn: Clasificación

La clasificación también es un método de aprendizaje supervisado, en clasificación el objetivo es determinar a que clase o categoría pertenece una observación, nuestra salida es un valor categórico, desde los problemas binarios (0 o 1) hasta los problemas multiclase.

Pandas dispone de distintos clasificadores, desde los más básicos como árboles o sgd classifier hasta los más complejos métodos de boosting.

Vamos a estudiar la clasificación con otros de los datasets de ejemplo, en este caso el iris, se trata de la caracterización de una serie de plantas de 3 especies distintas.

Ejercicio:

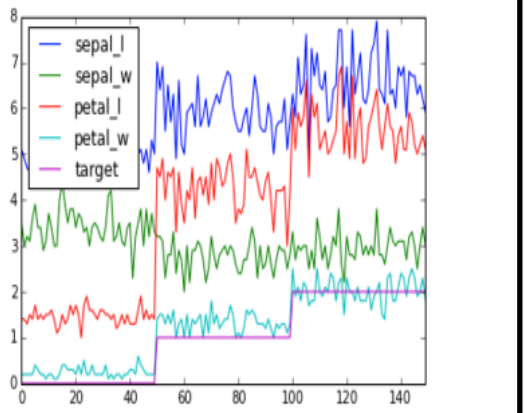
1. Carga el dataset de iris
2. Crea un DataFrame con los datos del iris, añadiéndole el "target" (0, 1, 2) y el "classname" (setosa, versicolor, virginica)
3. Crea un conjunto de entrenamiento (X_train, y_train) y un conjunto de test (X_test, y_test) a partir del dataset completo

66

Sklearn: Clasificación

Para entender el dataset con el que trabajamos, tenemos diferentes opciones.

```
In [11]: %matplotlib inline

         import matplotlib.pyplot as plt
         iris_df.plot()

         plt.show()
```




```
In [14]: iris_df.corr()
```

Out[14]:

	sepal_l	sepal_w	petal_l	petal_w	target
sepal_l	1.000000	-0.109369	0.871754	0.817954	0.782561
sepal_w	-0.109369	1.000000	-0.420516	-0.356544	-0.419446
petal_l	0.871754	-0.420516	1.000000	0.962757	0.949043
petal_w	0.817954	-0.356544	0.962757	1.000000	0.956464
target	0.782561	-0.419446	0.949043	0.956464	1.000000

```
In [13]: from pandas.tools.plotting import scatter_matrix
         scatter_matrix(iris_df, alpha=0.5, figsize=(6, 6), diagonal='kde')
```


67

Sklearn: Clasificación

Pese a tratarse de un problema distinto, el tipo de interfaz que ofrecen en sklearn los clasificadores es similar al de los regresores, en general, lo que estaremos haciendo una vez ya hemos preparado, limpiado y entendido nuestros datos será:

1. Elegir el algoritmo a emplear
2. Instanciar un clasificador del tipo deseado
3. Entrenar el clasificador mediante el método .fit()
4. Utilizar el clasificador para estimar las clases de los elementos de nuestro conjunto de test mediante predict(). En clasificadores también tenemos un método que nos da la confianza o probabilidad de pertenencia a la clase: predict_proba()
5. Evaluar el rendimiento del modelo

68

Sklearn: Clasificación

Utilizaremos un árbol de decisión, del módulo tree:

```
In [30]: from sklearn import tree


         my_tree = tree.DecisionTreeClassifier()
         my_tree.fit(X_train, y_train)

Out[30]: DecisionTreeClassifier(compute_importances=False, criterion='gini',
                     max_depth=None, max_features=None, min_density=0.1,
                     min_samples_leaf=1, min_samples_split=2, random_state=None)
```


69

Sklearn: Clasificación

Utilizaremos un árbol de decisión, del módulo tree:

```
In [30]: from sklearn import tree


         my_tree = tree.DecisionTreeClassifier()
         my_tree.fit(X_train, y_train)

Out[30]: DecisionTreeClassifier(compute_importances=False, criterion='gini',
                     max_depth=None, max_features=None, min_density=0.1,
                     min_samples_leaf=1, min_samples_split=2, random_state=None)
```

Ejercicio:

1. Entrena el árbol con el conjunto train
2. Utiliza el método predict() para calcular las prediciones sobre el conjunto de test
3. Calcula el error de clasificacion: errores/total
4. Crea una matriz de confusión: clase_real, clase_predictor, numero_casos
5. Repite el ejercicio con otro clasificador de Sklearn, ¿cuál da mejor resultado? http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
6. Empaqueta la parte de entrenamiento, scoring y calculo del performance como una función

Sklearn: Clasificación

Podemos utilizar directamente el método .score() para tener el rendimiento del modelo con un conjunto de datos etiquetados de test:

```
In [73]: print my_tree.score(X_train, y_train)
         print my_tree.score(X_test, y_test)

         1.0
         0.955555555556
```

En el caso del árbol, podemos exportar las reglas del mismo:

```
In [64]: from sklearn.tree import export_graphviz
         with open('my_tree.dot', 'w') as dotf:
             export_graphviz(
                 my_tree,
                 dotf,
                 feature_names=iris.feature_names)
         # http://sandbox.kidstrythisathome.com/erdos/
```

Sklearn: Ejercicio Final

1. Descarga el dataset titanic.csv. Puedes encontrar una explicación de los campos en https://www.kaggle.com/c/titanic/data

2. Carga el dataset en un DataFrame, el índice del DataFrame debe ser el nombre del pasajero.

3. Elimina las columnas ticket number y cabina

4. Analiza las columnas, en aquellas que aparezcan valores nulos, reemplaza esos valores por la media o la moda (valor más frecuente)

5. Separa el dataset en train y test.

6. Entrena al menos los siguientes clasificadores para predecir la variable "survived": DecisionTreeClassifier, SGDClassifier, RidgeClassifier, KNeighborsClassifier, RandomForestClassifier (*)

7. Rankea los clasificadores por su performance, medido como % de aciertos. Comparalos contra el modelo naive (i.e. asociar a todas las observaciones el resultado más frecuente)

(*) La mayoría de los clasificadores funcionan solo con variables numéricas, si tenemos una categórica como 'hombre'-'mujer', es habitual convertirlo en dos variables tipo: es_hombre (0/1), es_mujer (0/1). Ver pd.get_dummies()