Applied Machine Learning

Introduction

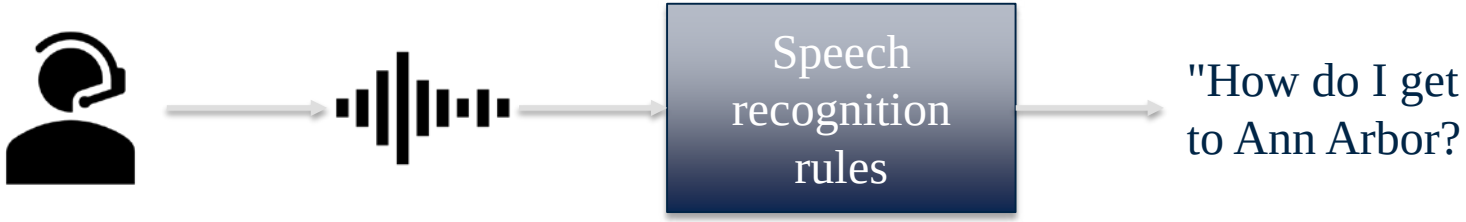
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

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*

Speech Recognition



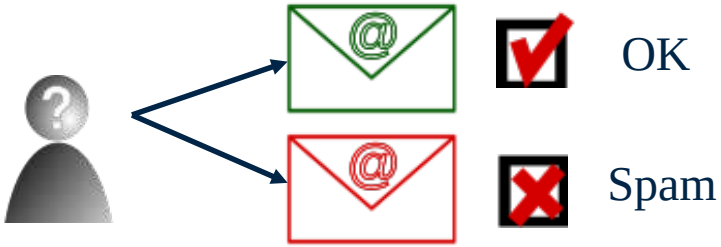
Machine Learning models can learn by example

- **Algorithms learn rules from <u>labelled examples</u>**
- **A set of labelled examples used for learning is called <u>training data.</u>**
- **The learned rules should also be able to <u>generalize</u> to correctly recognize or predict new examples not in the training set.**

Audio signal	Output text
	How do I get to Ann Arbor?
	Hello!
	Please order me a pizza.

Machine Learning models learn from experience

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



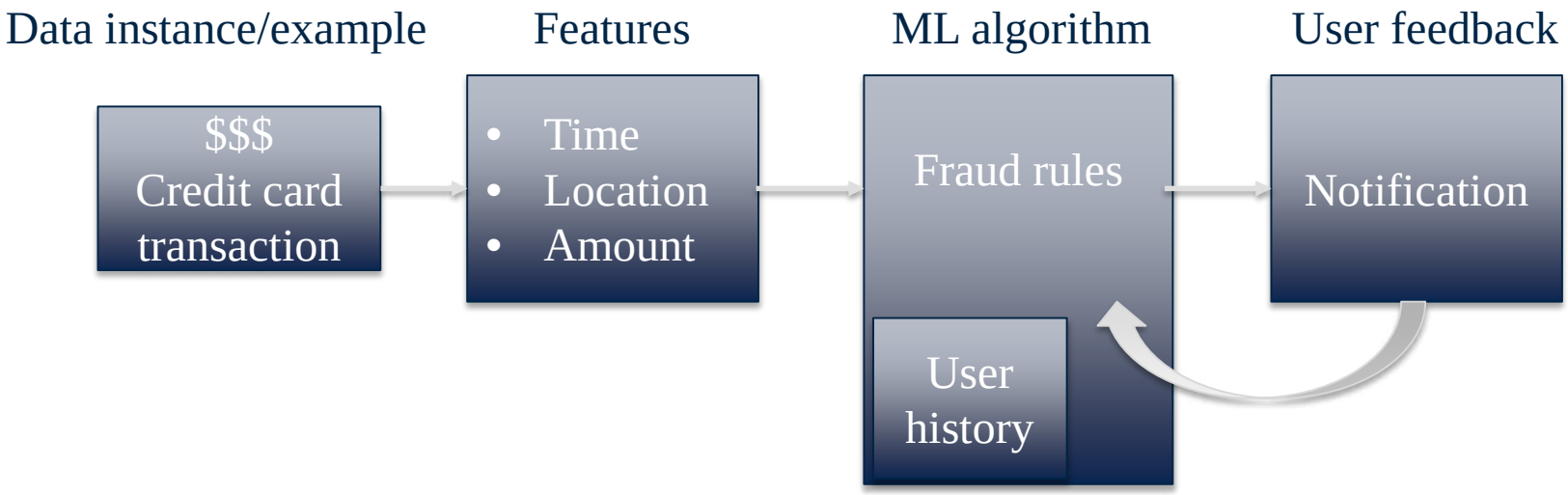
- **User feedback**
  (Clicks on a search page)

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

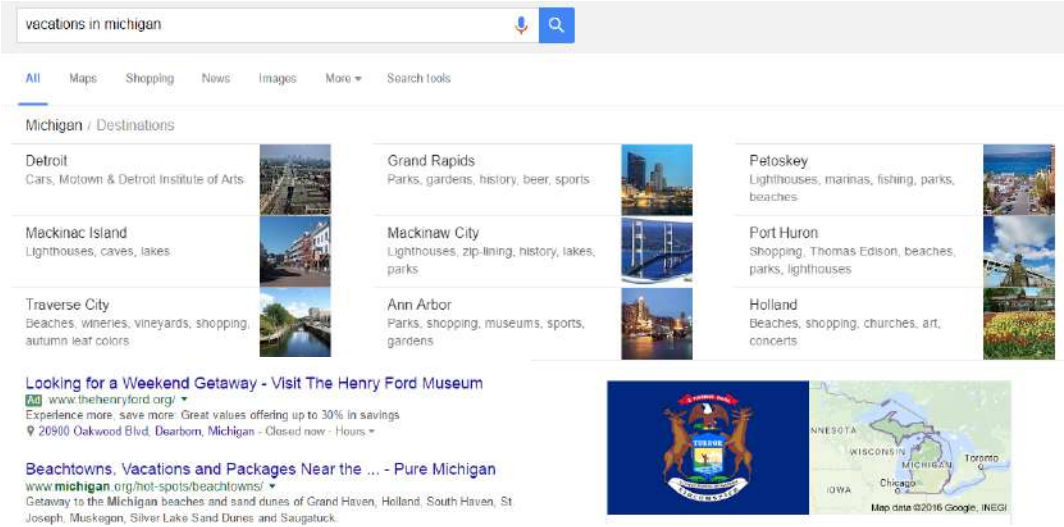
Machine Learning brings together statistics, computer science, and more..

- **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?*

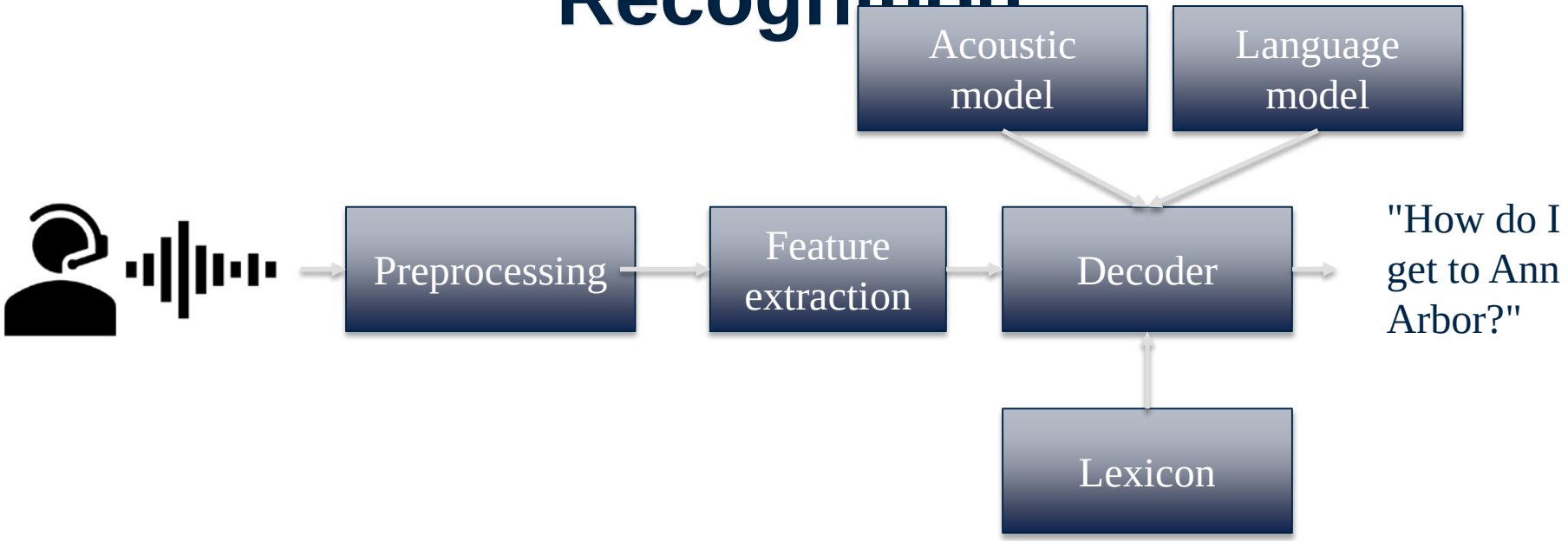
Machine Learning for fraud detection and credit scoring



Web search: query spell-checking, result ranking, content classification and selection, advertising placement



Machine Learning for Speech Recognition



Machine Learning algorithms are at the heart of the information economy

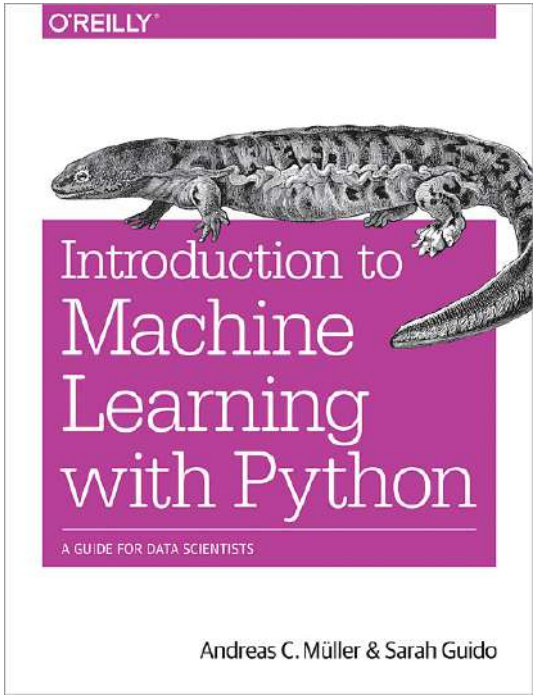
- Finance: fraud detection, credit scoring
- Web search
- Speech recognition
- eCommerce: Product recommendations
- Email spam filtering
- Health applications: drug design and discovery
- Education: Automated essay scoring

What is Applied Machine Learning?

- **Understand basic ML concepts and workflow**
- **How to properly apply 'black-box' machine learning components and features**
- **Learn how to apply machine learning algorithms in Python using the scikit-learn package**

- **What is not covered in this course:**
  - *Underlying theory of statistical machine learning*
  - *Lower-level details of how particular ML components work*
  - *In-depth material on more advanced concepts like deep learning*

Recommended text for this course



Introduction to Machine Learning with Python
A Guide for Data Scientists
By Andreas C. Müller and Sarah Guido

O'Reilly Media

Applied Machine Learning

Key Concepts in Machine Learning

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Key types of Machine Learning problems

<u>Supervised</u> machine learning: Learn to predict <u>target values</u> from labelled data.

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

Supervised Learning (classification example)

Training set

X Sample	Y 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.

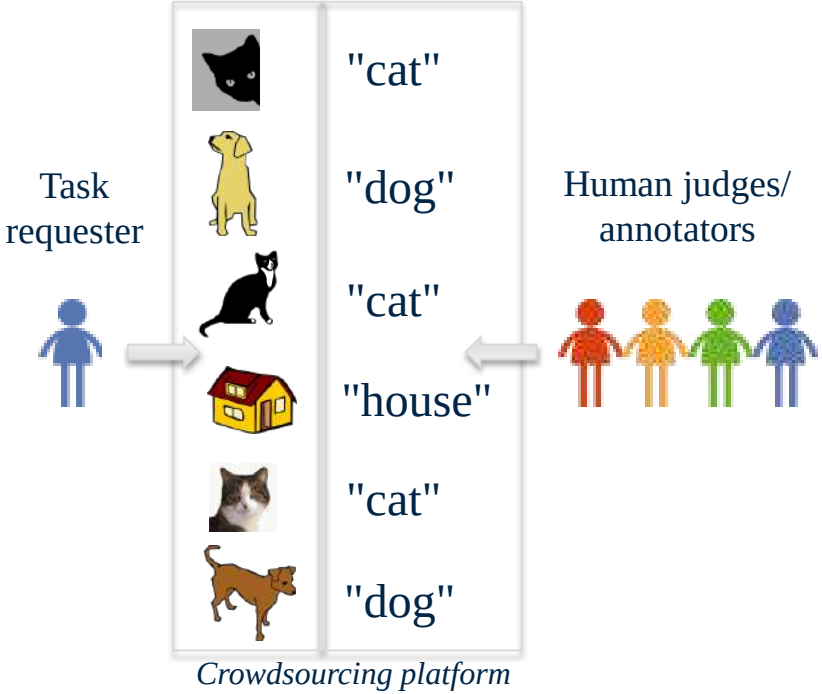
Future sample

Label: Orange

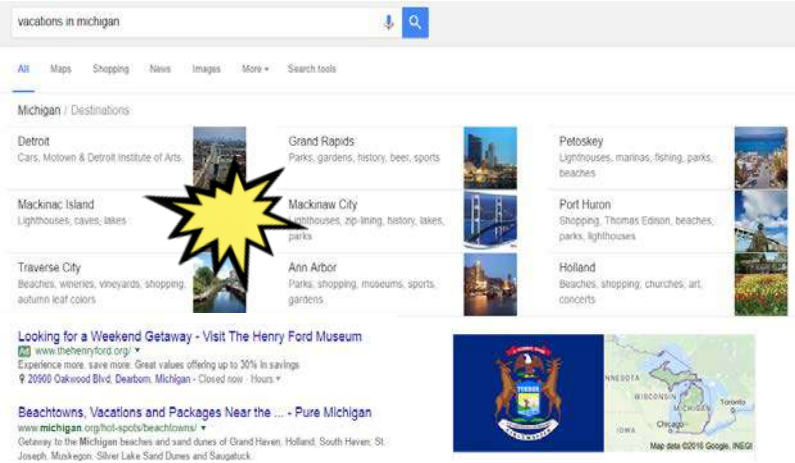
After training, at prediction time, the trained model is used to predict the fruit type for new instances using the learned rules.

Examples of explicit and implicit label sources

Explicit labels



Implicit labels



Clicking and reading the "Mackinac Island" result can be an implicit label for the search engine to learn that "Mackinac Island" is especially relevant for the query `[vacations in michigan]` for that specific user.

Key types of Machine Learning problems

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

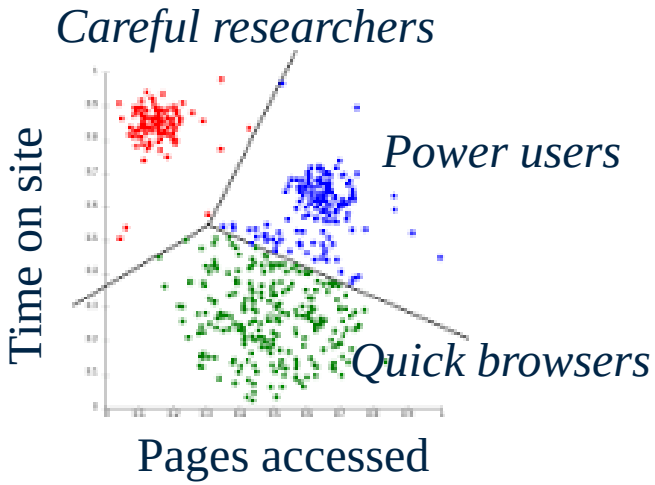
- Classification (target values are discrete classes)
- Regression (target values are continuous values)

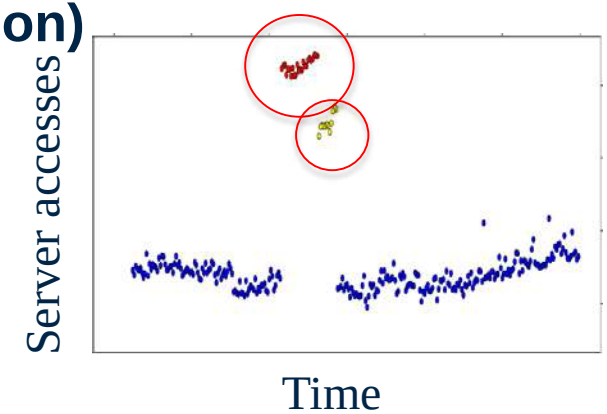
Unsupervised machine learning: Find structure in *unlabeled data*

- **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)**





A Basic Machine Learning Workflow

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

e.g. image pixels, with
k-nearest neighbor classifier

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

e.g. % correct predictions on test set

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

e.g. try a range of values for "k" parameter in k-nearest neighbor classifier

Feature Representations

Feature representation

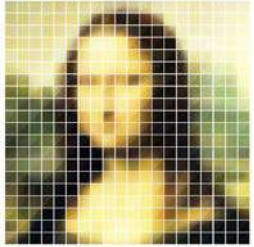
Email

```
To:  Chris Brooks
From:  Daniel Romero
Subject:  Next course offering
Hi Daniel,
Could you please send the outline for the
next course offering?  Thanks! -- Chris
```

Feature	Count
to	1
chris	2
brooks	1
from	1
daniel	2
romero	1
the	2
...	

A list of words with their frequency counts

Picture



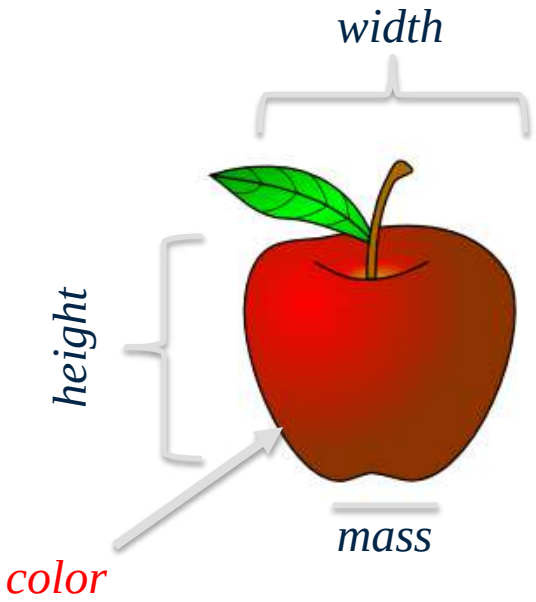
A matrix of color values (pixels)

Sea Creatures

Feature	Value
DorsalFin	Yes
MainColor	Orange
Stripes	Yes
StripeColor1	White
StripeColor2	Black
Length	4.3 cm

A set of attribute values

Representing a piece of fruit as an array of features (plus label information)



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)

Represent / Train / Evaluate / Refine Cycle

Representation:
Extract and select object features

Represent / Train / Evaluate / Refine Cycle

Representation: Extract and select object features

→

Train models: Fit the estimator to the data

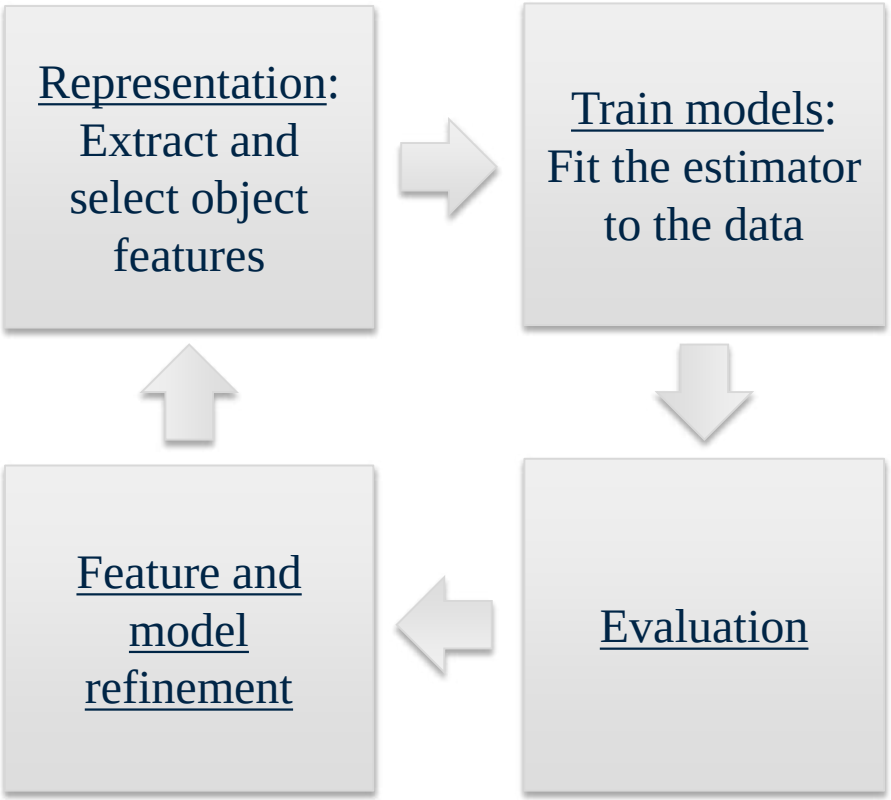
Represent / Train / Evaluate / Refine Cycle

<u>Representation</u>: Extract and select object features

<u>Train models</u>: Fit the estimator to the data

<u>Evaluation</u>

Represent / Train / Evaluate / Refine Cycle



Applied Machine Learning

Python Tools for Machine Learning

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

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
```

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 [illegible]s

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

http://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

matplotlib 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
```

Versions of main libraries used in this course

Library name	Minimum version
scikit-learn	0.18.1
scipy	0.19.0
numpy	1.12.1
pandas	0.19.2
matplotlib	2.0.1
seaborn	0.7.1
graphviz	0.7

It's okay if your versions of these don't match ours exactly, as long as the version of scikit-learn and other libraries you're using is the same or greater than listed here.

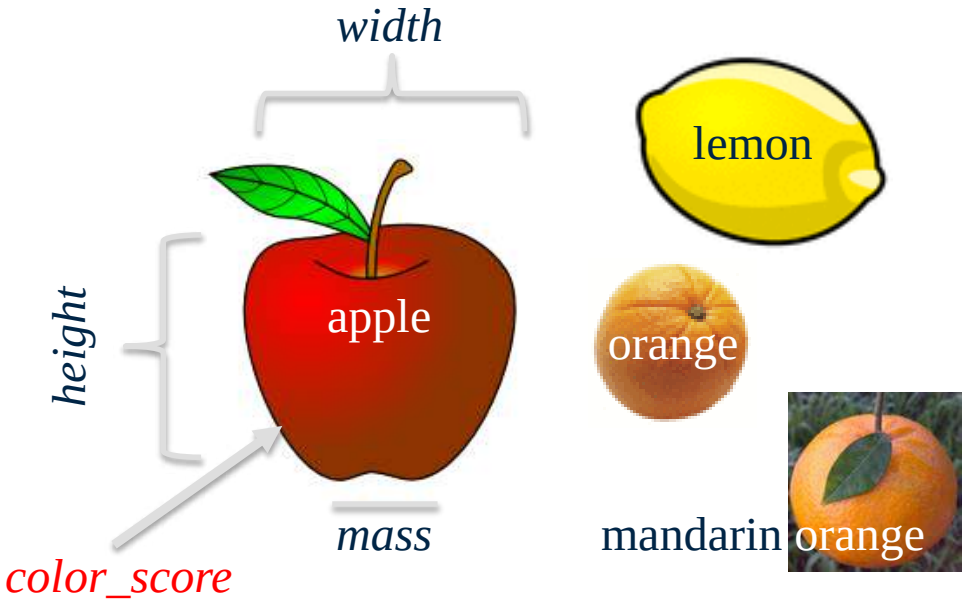
Applied Machine Learning

An Example Machine Learning Problem

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

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

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	andarin	mandarin	86	6.2	4.7	0.80
4	2	andarin	mandarin	84	6.0	4.6	0.79
5	2	andarin	mandarin	80	5.8	4.3	0.77
6	2	andarin	mandarin	80	5.9	4.3	0.81
7	2	andarin	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
1	1	pple	braeburn	166	6.9	7.3	0.93
1	1	pple	braeburn	172	7.1	7.6	0.92
1	1	pple	braeburn	154	7.0	7.1	0.88
1	1	pple	golden_delicious	164	7.3	7.7	0.70
1	1	pple	golden_delicious	152	7.6	7.3	0.69
1	1	pple	golden_delicious	156	7.7	7.1	0.69
1	1	pple	golden_delicious	156	7.6	7.5	0.67
1	1	pple	golden_delicious	168	7.5	7.6	0.73
1	1	pple	cripps_pink	162	7.5	7.1	0.83
1	1	pple	cripps_pink	162	7.4	7.2	0.85
2	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)

The scale for the (simplistic) `color_score` feature used in the fruit dataset

0.00 0.25 0.50 0.75 1.00

Color category	`color_score`
Red	0.85 - 1.00
Orange	0.75 - 0.85
Yellow	0.65 - 0.75
Green	0.45 - 0.65

Creating Training and Testing Sets

X

	height	width	mass	color_score
0	7.3	8.4	192	0.55
1	6.8	8.0	180	0.59
2	7.2	7.4	176	0.60
3	4.7	6.2	86	0.80
4	4.6	6.0	84	0.79
5	4.3	5.8	80	0.77
6	4.3	5.9	80	0.81
7	4.0	5.8	76	0.81
8	7.8	7.1	178	0.92
9	7.0			
10	7.3			
11	7.6			
12	7.1			
13	7.7			
14	7.3	7.6	152	0.69
15	7.1	7.7	156	0.69
16	7.5	7.6	156	0.67
17	7.6	7.5	168	0.73
18	7.1	7.5	162	0.83
19	7.2	7.4	162	0.85

y

0	1
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	1
14	1
15	1
16	1
17	1
18	1
19	1

Original data set

```
X_train, X_test, y_train, y_test
    = train_test_split(X, y)
```

X_train

	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	1[illegible]	0.79
[illegible]	[illegible]	[illegible]	[illegible]	0.73
⋮				⋮
39	7.4	6.8	144	0.75
3	4.7	6.2	86	0.80
0	7.3	8.4	192	0.55
53	8.4	6.0	120	0.74
47	9.7	7.3	196	0.72
44	10.5	7.3	200	0.72

y_train

42	3
48	4
7	2
14	1
32	3
49	4
29	3
37	3
56	4
⋮	
39	3
3	2
0	1
53	4
47	4
44	4

Training set

X_test

	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
[illegible]	[illegible]	[illegible]	[illegible]	0.93
[illegible]	[illegible]	[illegible]	[illegible]	0.79
41	8.2	7.6	180	0.79
33	8.1	7.5	190	0.74

y_test

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

Test set

Applied Machine Learning

Examining the Data

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

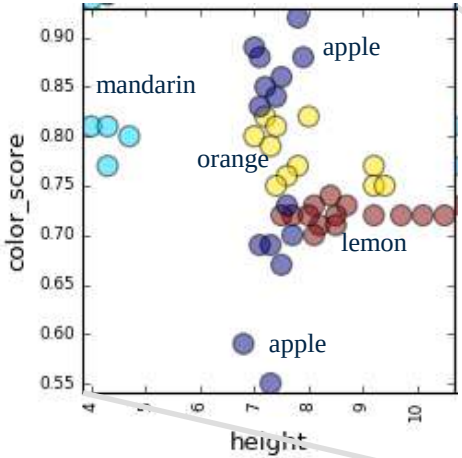
Some reasons why looking at the data initially is important

Examples of incorrect or missing feature values

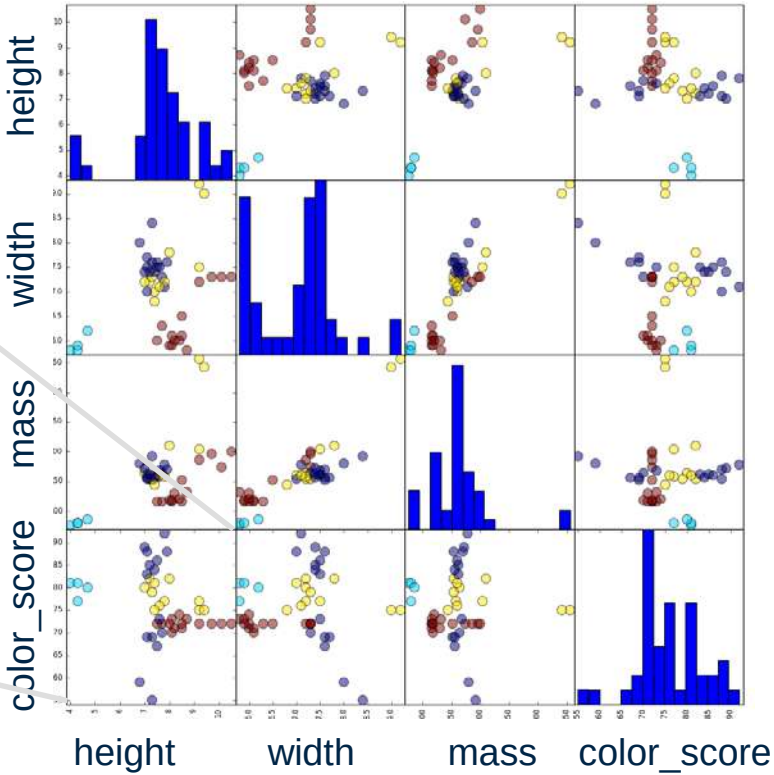
- Inspecting feature values may help identify what cleaning or preprocessing still needs to be done once you can see the range or distribution of values that is typical for each attribute.
- You might notice missing or noisy data, or inconsistencies such as the wrong data type being used for a column, 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	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69

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.



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)
```

```
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]:
```

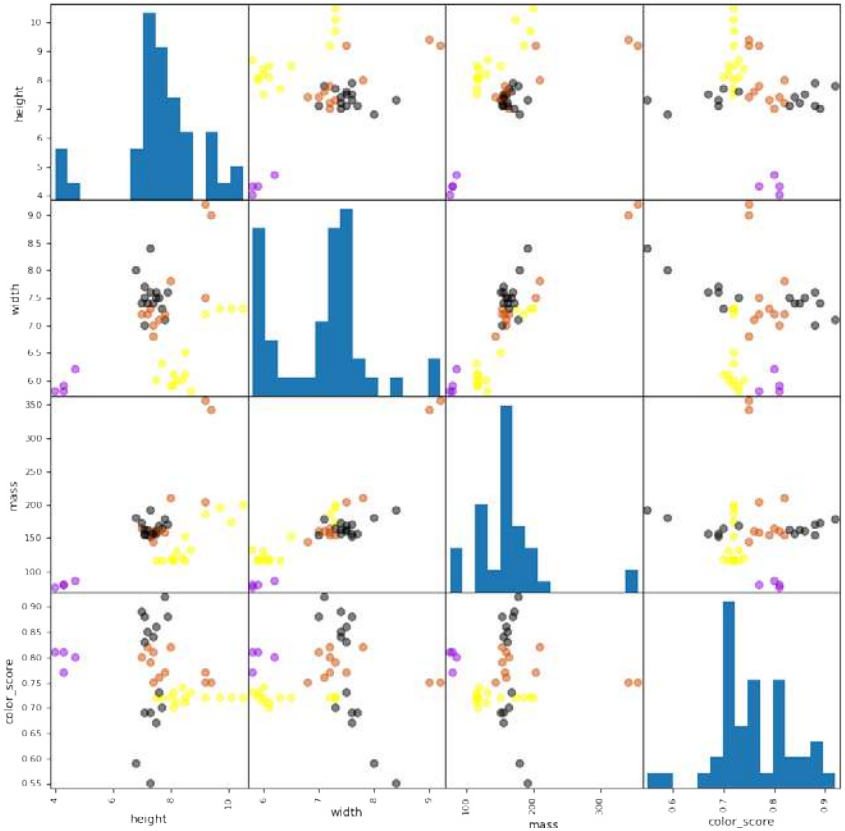
	height	width	mass	color_score
42	7.2	7.2	154	0.82
48	10.1	7.3	174	0.72
7	4.0	5.8	76	0.81
14	7.3	7.6	152	0.69
32	7.0	7.2	164	0.80
49	8.7	5.8	132	0.73
29	7.4	7.0	160	0.81
37	7.3	7.3	154	0.79
56	8.1	5.9	116	0.73
18	7.1	7.5	162	0.83
55	7.7	6.3	116	0.72
27	9.2	7.5	204	0.77
15	7.1	7.7	156	0.69
5	4.3	5.8	80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

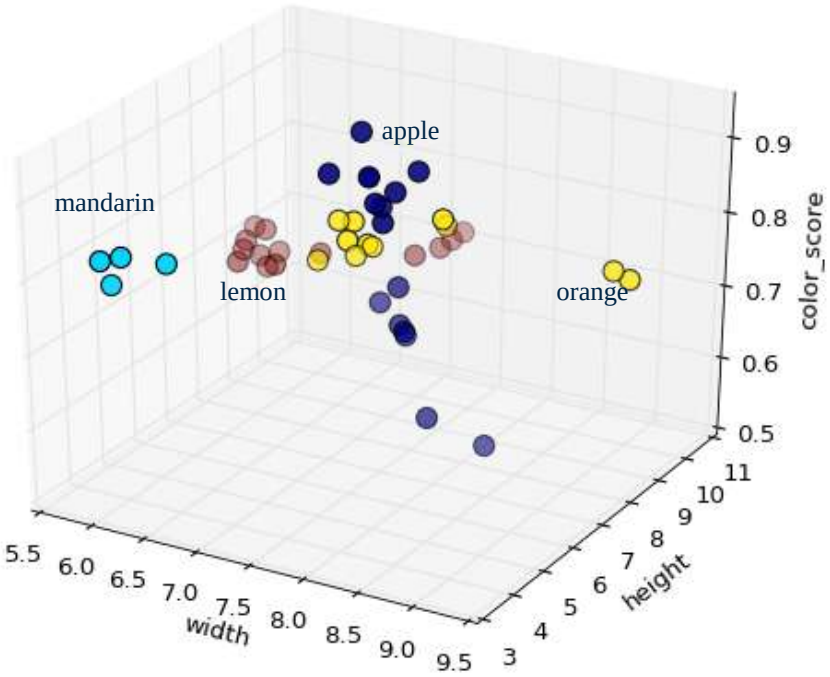
	height	width	mass	color_score
26	9.2	9.6	362	0.74
35	7.9	7.1	150	0.75
43	10.3	7.2	194	0.70
28	7.1	6.7	140	0.72
11	7.6	7.1	172	0.92
2	7.2	7.4	176	0.60
34	7.8	7.6	142	0.75
46	10.2	7.3	216	0.71
40	7.5	7.1	154	0.78
22	7.1	7.3	140	0.87
4	4.6	6.0	84	0.79
10	7.3	6.9	166	0.93
30	7.5	7.1	158	0.79
41	8.2	7.6	180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

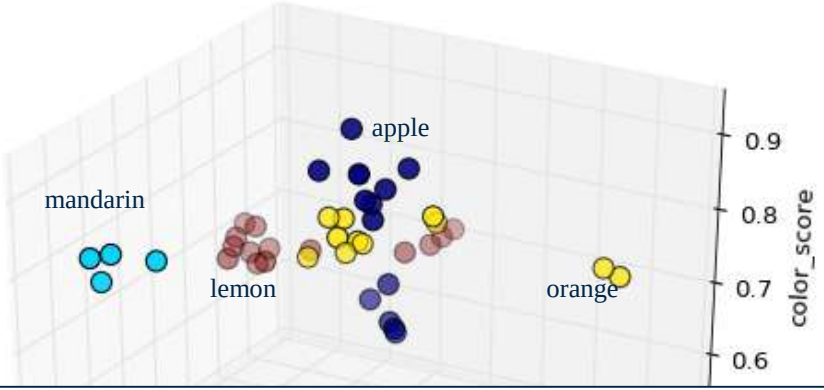



```
from matplotlib import cm
cmap = cm.get_cmap('gnuplot')
scatter = pd.scatter_matrix(X_train, c= y_train, marker = 'o', s=40, hist_kwds={'bins':15}, figsize=(12,12), cmap=cmap)
```

A three-dimensional feature scatterplot



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()
```

Applied Machine Learning

K-Nearest Neighbors Classification

Kevyn Collins-Thompson

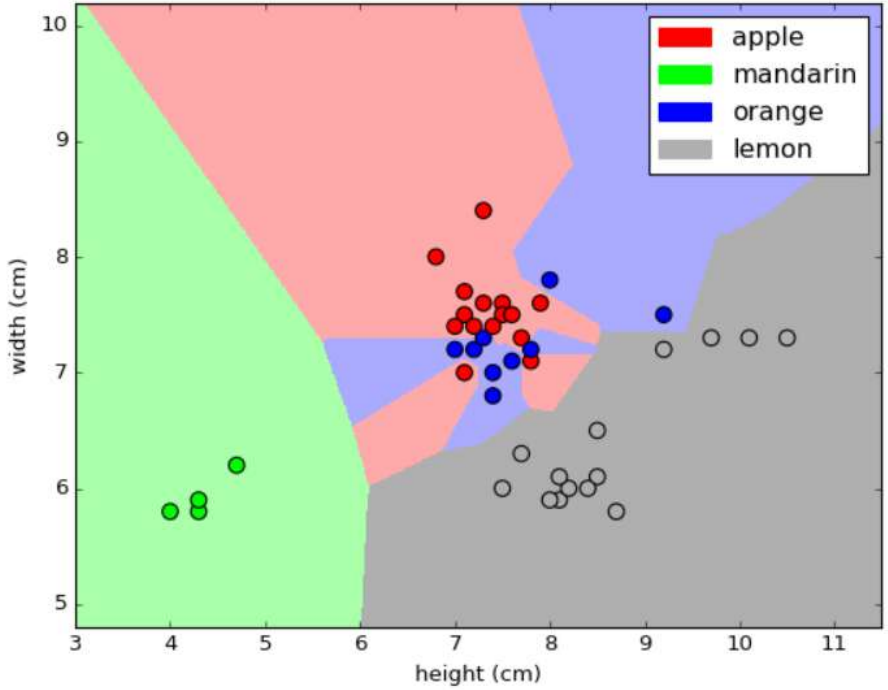
Associate Professor of Information & Computer Science
University of Michigan

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



Fruit dataset
Decision boundaries
with k = 1

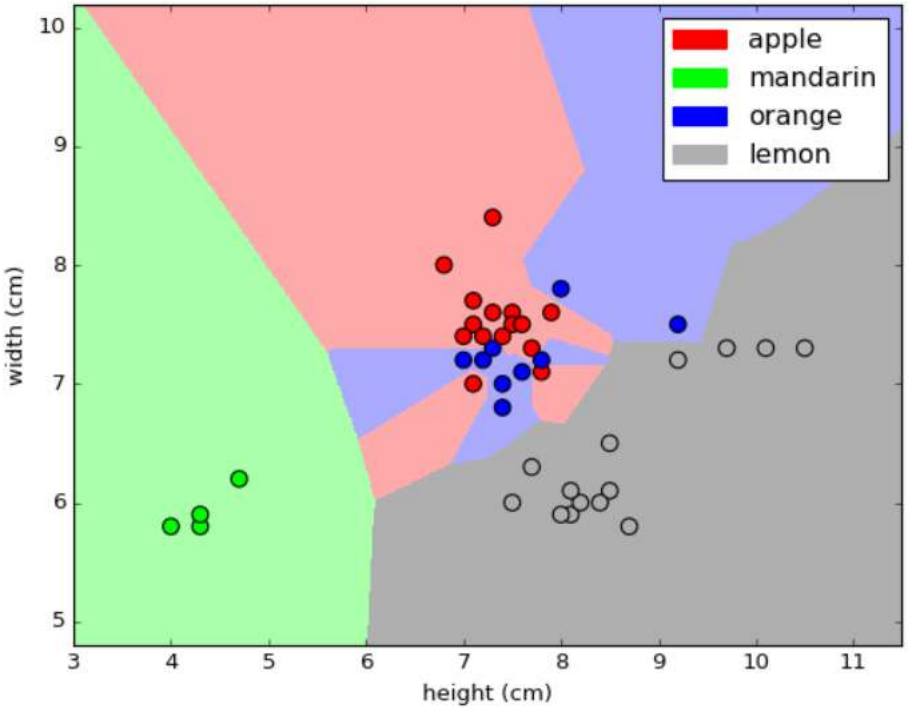
A nearest neighbor algorithm needs four things specified

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

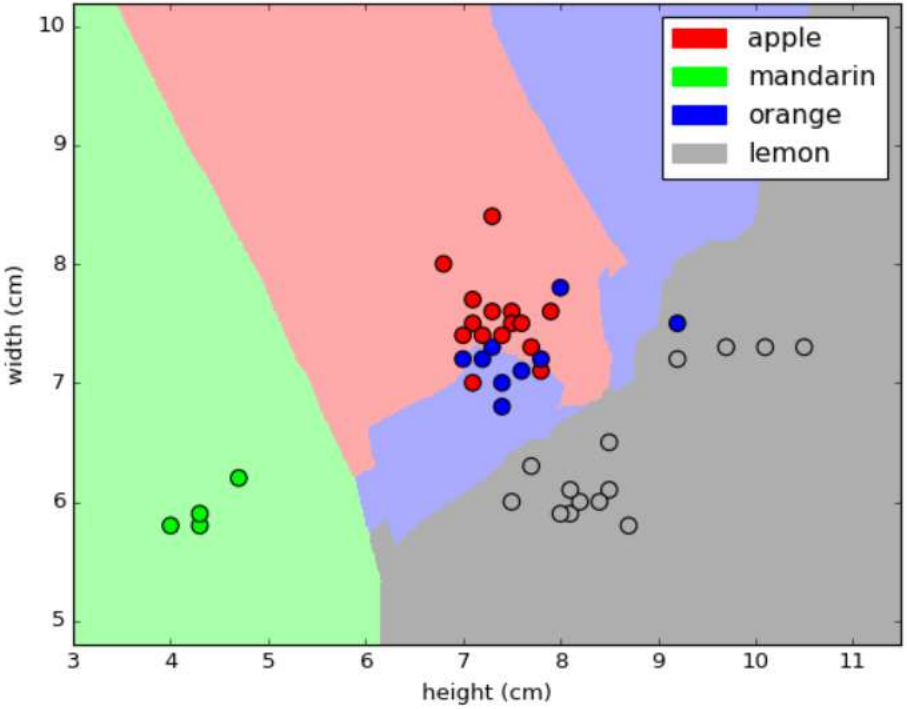
A nearest neighbor algorithm needs four things specified

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)

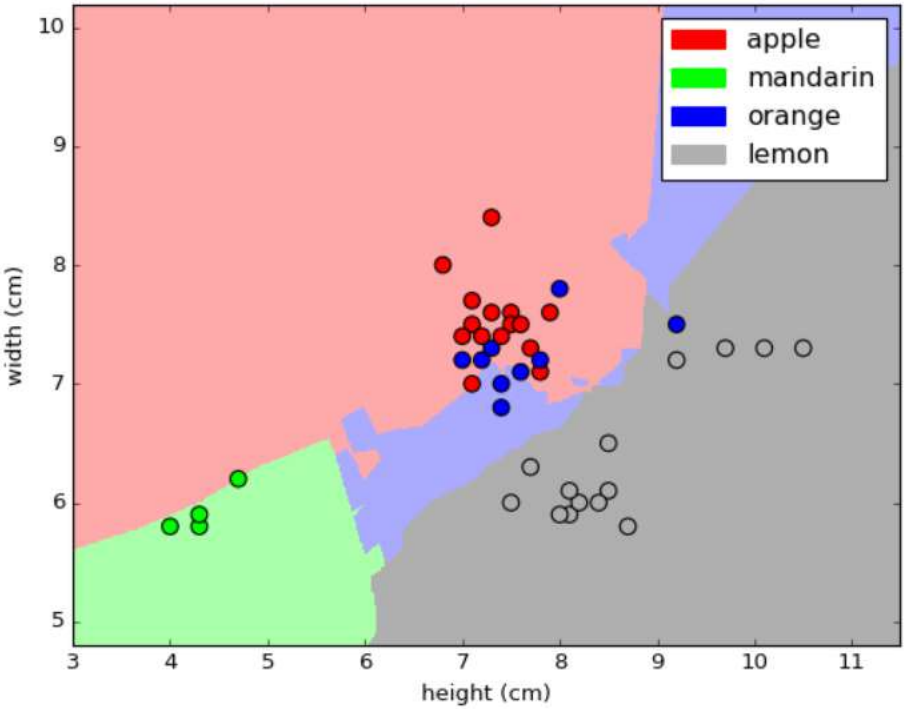
K-nearest neighbors (k=1) for fruit dataset

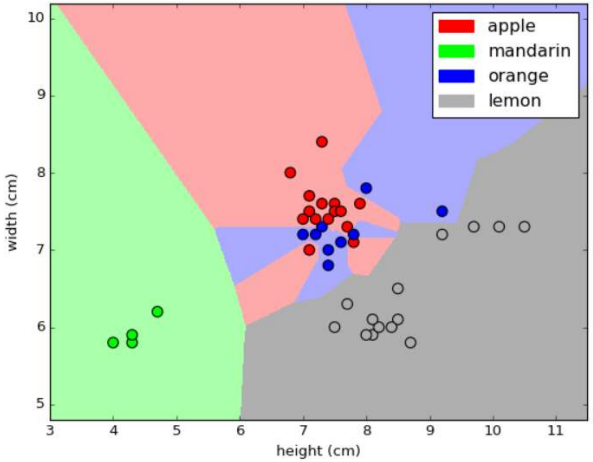


K-nearest neighbors (k=5) for fruit dataset

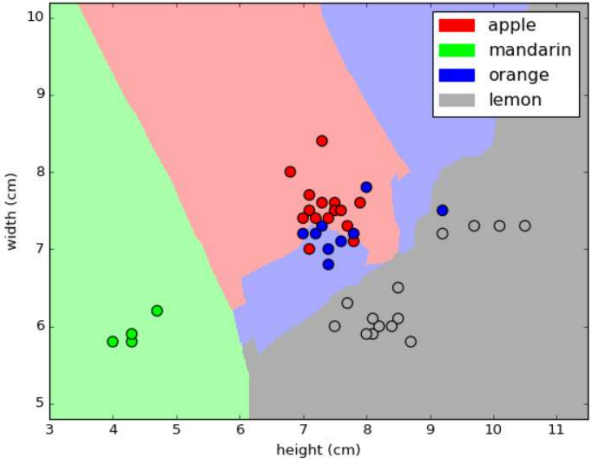


K-nearest neighbors (k=10) for fruit dataset

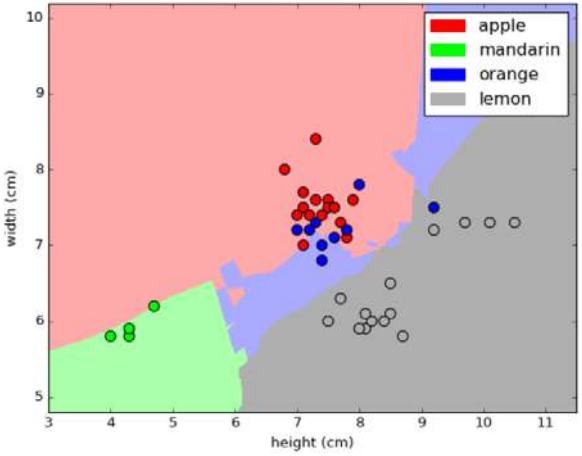



apple
mandarin
orange
lemon
width (cm)
height (cm)

K=1

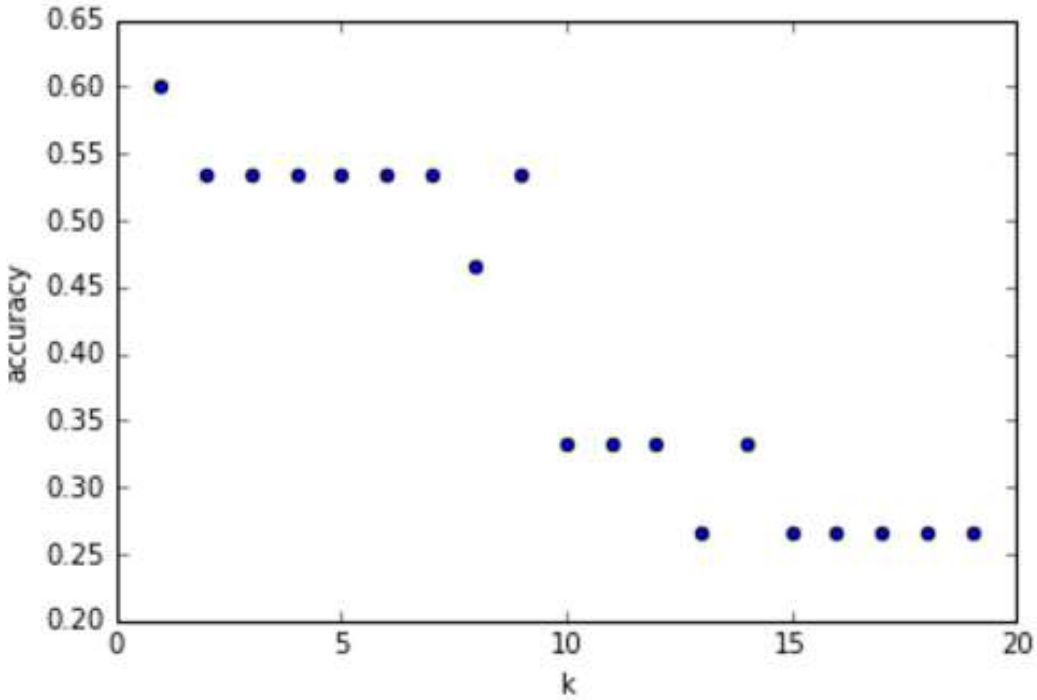

apple
mandarin
orange
lemon
width (cm)
height (cm)

K=5


apple
mandarin
orange
lemon
width (cm)
height (cm)

K=10

How sensitive is k-NN classifier accuracy to the choice of 'k' parameter?



Fruit dataset with 75%/25% train-test split

Module 1

September 22, 2021

---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

0.1 Applied Machine Learning, Module 1: A simple classification task

0.1.1 Import required modules and load data file

In [1]:
```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split

fruits = pd.read_table('readonly/fruit_data_with_colors.txt')
fruits
```

Out[1]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_sc
0	1	apple	granny_smith	192	8.4	7.3	0
1	1	apple	granny_smith	180	8.0	6.8	0
2	1	apple	granny_smith	176	7.4	7.2	0
3	2	mandarin	mandarin	86	6.2	4.7	0
4	2	mandarin	mandarin	84	6.0	4.6	0
5	2	mandarin	mandarin	80	5.8	4.3	0
6	2	mandarin	mandarin	80	5.9	4.3	0
7	2	mandarin	mandarin	76	5.8	4.0	0
8	1	apple	braeburn	178	7.1	7.8	0
9	1	apple	braeburn	172	7.4	7.0	0
10	1	apple	braeburn	166	6.9	7.3	0
11	1	apple	braeburn	172	7.1	7.6	0
12	1	apple	braeburn	154	7.0	7.1	0
13	1	apple	golden_delicious	164	7.3	7.7	0
14	1	apple	golden_delicious	152	7.6	7.3	0
15	1	apple	golden_delicious	156	7.7	7.1	0
16	1	apple	golden_delicious	156	7.6	7.5	0
17	1	apple	golden_delicious	168	7.5	7.6	0

18	1	apple	cripps_pink	162	7.5	7.1	0
19	1	apple	cripps_pink	162	7.4	7.2	0
20	1	apple	cripps_pink	160	7.5	7.5	0
21	1	apple	cripps_pink	156	7.4	7.4	0
22	1	apple	cripps_pink	140	7.3	7.1	0
23	1	apple	cripps_pink	170	7.6	7.9	0
24	3	orange	spanish_jumbo	342	9.0	9.4	0
25	3	orange	spanish_jumbo	356	9.2	9.2	0
26	3	orange	spanish_jumbo	362	9.6	9.2	0
27	3	orange	selected_seconds	204	7.5	9.2	0
28	3	orange	selected_seconds	140	6.7	7.1	0
29	3	orange	selected_seconds	160	7.0	7.4	0
30	3	orange	selected_seconds	158	7.1	7.5	0
31	3	orange	selected_seconds	210	7.8	8.0	0
32	3	orange	selected_seconds	164	7.2	7.0	0
33	3	orange	turkey_navel	190	7.5	8.1	0
34	3	orange	turkey_navel	142	7.6	7.8	0
35	3	orange	turkey_navel	150	7.1	7.9	0
36	3	orange	turkey_navel	160	7.1	7.6	0
37	3	orange	turkey_navel	154	7.3	7.3	0
38	3	orange	turkey_navel	158	7.2	7.8	0
39	3	orange	turkey_navel	144	6.8	7.4	0
40	3	orange	turkey_navel	154	7.1	7.5	0
41	3	orange	turkey_navel	180	7.6	8.2	0
42	3	orange	turkey_navel	154	7.2	7.2	0
43	4	lemon	spanish_belsan	194	7.2	10.3	0
44	4	lemon	spanish_belsan	200	7.3	10.5	0
45	4	lemon	spanish_belsan	186	7.2	9.2	0
46	4	lemon	spanish_belsan	216	7.3	10.2	0
47	4	lemon	spanish_belsan	196	7.3	9.7	0
48	4	lemon	spanish_belsan	174	7.3	10.1	0
49	4	lemon	unknown	132	5.8	8.7	0
50	4	lemon	unknown	130	6.0	8.2	0
51	4	lemon	unknown	116	6.0	7.5	0
52	4	lemon	unknown	118	5.9	8.0	0
53	4	lemon	unknown	120	6.0	8.4	0
54	4	lemon	unknown	116	6.1	8.5	0
55	4	lemon	unknown	116	6.3	7.7	0
56	4	lemon	unknown	116	5.9	8.1	0
57	4	lemon	unknown	152	6.5	8.5	0
58	4	lemon	unknown	118	6.1	8.1	0

In [2]:
```
fruits.head()
```

Out[2]:

	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
```

1 seems like converting a zip to dict, for better understanding

```
In [3]: # create a mapping from fruit label value to fruit name to make results eas
        lookup_fruit_name = dict(zip(fruits.fruit_label.unique(), fruits.fruit_name
        #lookup_fruit_name = zip(fruits.fruit_label.unique(), fruits.fruit_name.uni
        lookup_fruit_name
```

```
Out[3]: {1: 'apple', 2: 'mandarin', 3: 'orange', 4: 'lemon'}
```

The file contains the mass, height, and width of a selection of oranges, lemons and apples. The heights were measured along the core of the fruit. The widths were the widest width perpendicular to the height.

1.0.1 Examining the data

1.0.2 75 25 split hota hai by default

```
In [4]: # plotting a scatter matrix
        from matplotlib import cm

        X = fruits[['height', 'width', 'mass', 'color_score']]
        y = fruits['fruit_label']
        X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
        print('X_train is \n', X_train ,'\n','X_test is \n', X_test ,'\n','y_train
        print('Lengths are X_train ', len(X_train),'X_test ', len(X_test),'y_train

        cmap = cm.get_cmap('gnuplot')
        scatter = pd.scatter_matrix(X_train, c= y_train, marker = 'o', s=40, hist_k
```

```
X_train is
     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
50      8.2    6.0   130         0.71
20      7.5    7.5   160         0.86
51      7.5    6.0   116         0.72
8       7.8    7.1   178         0.92
13      7.7    7.3   164         0.70
25      9.2    9.2   356         0.75
17      7.6    7.5   168         0.73
58      8.1    6.1   118         0.70
57      8.5    6.5   152         0.72
52      8.0    5.9   118         0.72
38      7.8    7.2   158         0.77
1       6.8    8.0   180         0.59
12      7.1    7.0   154         0.88
45      9.2    7.2   186         0.72
24      9.4    9.0   342         0.75
6       4.3    5.9    80         0.81
23      7.9    7.6   170         0.88
36      7.6    7.1   160         0.76
21      7.4    7.4   156         0.84
19      7.2    7.4   162         0.85
9       7.0    7.4   172         0.89
39      7.4    6.8   144         0.75
54      8.5    6.1   116         0.71
3       4.7    6.2    86         0.80
0       7.3    8.4   192         0.55
53      8.4    6.0   120         0.74
47      9.7    7.3   196         0.72
44     10.5    7.3   200         0.72
 X_test is
     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
 y_train is
```

```
 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
23    1
36    3
21    1
19    1
9     1
39    3
54    4
3     2
0     1
53    4
47    4
44    4
Name: fruit_label, dtype: int64
 y_test is
 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
Name: fruit_label, dtype: int64
Lengths are X_train  44 X_test  15 y_train  44 y_test 15
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In [5]:
```
# plotting a 3D scatter plot
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 =
ax.set_xlabel('width')
ax.set_ylabel('height')
ax.set_zlabel('color_score')
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.0.3 Create train-test split

In [6]:
```
# For this example, we use the mass, width, and height features of each fru
X = fruits[['mass', 'width', 'height']]
y = fruits['fruit_label']

# default is 75% / 25% train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

1.0.4 Create classifier object

```
In [7]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors = 5) # we are using 5 closest neighb
```

1.0.5 Train the classifier (fit the estimator) using the training data

```
In [8]: knn.fit(X_train, y_train) #minkowski matrix with p=2 matlab euclidian geome
```

```
Out[8]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                   metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                   weights='uniform')
```

1.0.6 Estimate the accuracy of the classifier on future data, using the test data

```
In [9]: knn.score(X_test, y_test) #kuch khaas nahi aaya
```

```
Out[9]: 0.53333333333333333
```

1.0.7 Use the trained k-NN classifier model to classify new, previously unseen objects

1.0.8 individual examples de rahe hai

```
In [10]: # first example: a small fruit with mass 20g, width 4.3 cm, height 5.5 cm
         fruit_prediction = knn.predict([[20, 4.3, 5.5]])
         print(type(fruit_prediction))
         print(fruit_prediction[0]) #value 2 hai, wo dictionary me map hota hai mar

         lookup_fruit_name[fruit_prediction[0]] #
```

```
<class 'numpy.ndarray'>
2
```

```
Out[10]: 'mandarin'
```

```
In [11]: # second example: a larger, elongated fruit with mass 100g, width 6.3 cm,
         fruit_prediction = knn.predict([[100, 6.3, 8.5]])
         lookup_fruit_name[fruit_prediction[0]]
```

```
Out[11]: 'lemon'
```

1.0.9 Plot the decision boundaries of the k-NN classifier

```
In [12]: from adspy_shared_utilities import plot_fruit_knn

         plot_fruit_knn(X_train, y_train, 5, 'uniform')   # we choose 5 nearest nei
         #plot_fruit_knn(X_train, y_train, 30, 'uniform')
         #plot_fruit_knn(X_train, y_train, 10, 'uniform')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.0.10 How sensitive is k-NN classification accuracy to the choice of the 'k' parameter?

```
In [13]: k_range = range(1,20)
         scores = []

         for k in k_range:
             knn = KNeighborsClassifier(n_neighbors = k)
             knn.fit(X_train, y_train)
             scores.append(knn.score(X_test, y_test))

         print(scores)

         plt.figure()
         plt.xlabel('k')
         plt.ylabel('accuracy')
         plt.scatter(k_range, scores)
         plt.xticks([0,5,10,15,20]);
```

```
[0.59999999999999998, 0.53333333333333333, 0.53333333333333333, 0.5333333333333333
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.0.11 How sensitive is k-NN classification accuracy to the train/test split proportion?

```
In [14]: t = [0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2]

         knn = KNeighborsClassifier(n_neighbors = 5)

         plt.figure()

         for s in t:

             scores = []
             for i in range(1,1000):
                 X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
                 knn.fit(X_train, y_train)
                 scores.append(knn.score(X_test, y_test))
             plt.plot(s, np.mean(scores), 'bo')
```

```
        plt.xlabel('Training set proportion (%)')
        plt.ylabel('accuracy');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In [ ]:

Applied Machine Learning

Supervised machine learning (Part 1)

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Learning objectives

- **Understand how a number of different supervised learning algorithms learn by estimating their parameters from data to make new predictions.**
- **Understand the strengths and weaknesses of particular supervised learning methods.**

Learning objectives

- **Learn how to apply specific supervised machine learning algorithms in Python with scikit-learn.**
- **Learn about general principles of supervised machine learning, like overfitting and how to avoid it.**

Review of important terms

- **Feature representation**
- **Data instances/samples/ examples (X)**
- **Target value (y)**

```
fruits
```

	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
17	1	apple	golden_delicious	168	7.5	7.6	0.73
18	1	apple	cripps_pink	162	7.5	7.1	0.83

Review of important terms

- **Training and test sets**
- **Model/Estimator**
  - *Model fitting produces a 'trained model'.*
  - *Training is the process of estimating model parameters.*
- **Evaluation method**

```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

fruits = pd.read_table('fruit_data_with_colors.txt')

X = fruits[['height', 'width', 'mass', 'color_score']]
y = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
print("Accuracy of K-NN classifier on test set: ", knn.score(X_test, y_test))

example_fruit = [[5.5, 2.2, 10, 0.70]]
print("Predicted fruit type for ", example_fruit, " is ", knn.predict(example_fruit))
```

Applied Machine Learning

Overfitting and Underfitting

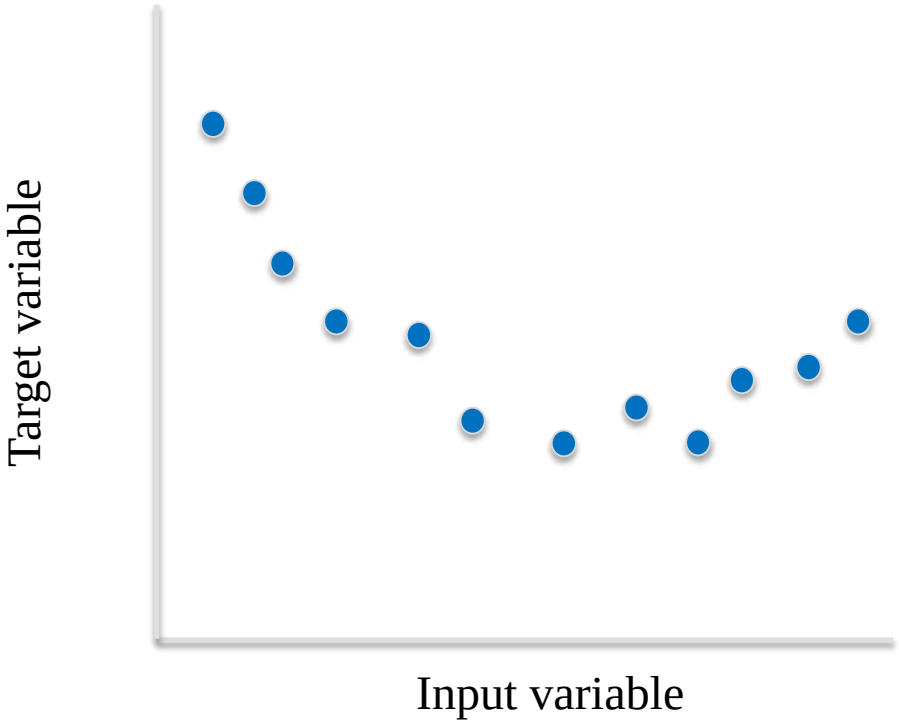
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

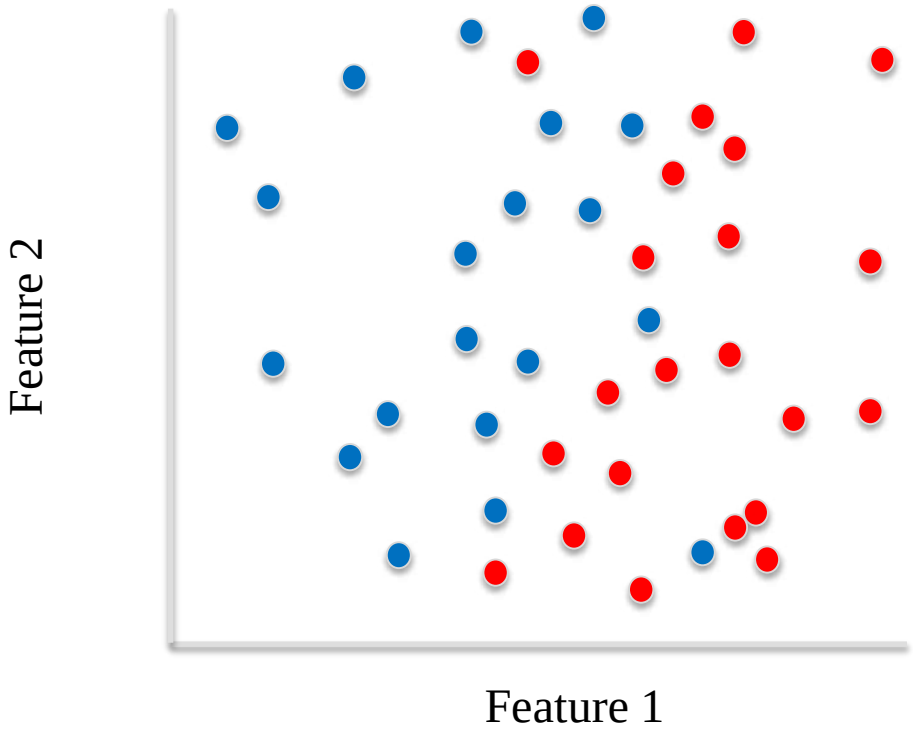
Generalization, Overfitting, and Underfitting

- **<u>Generalization ability</u> 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 <u>overfit</u> 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 <u>underfit</u> and also not likely to generalize well.**

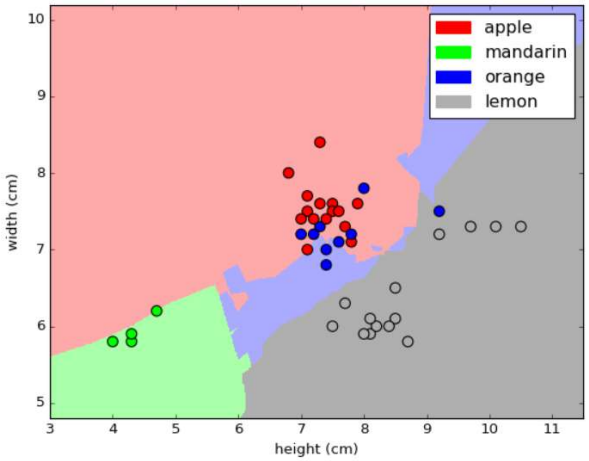
Overfitting in regression



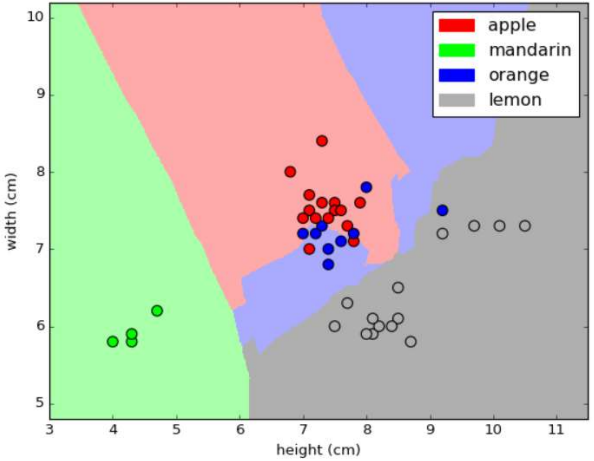
Overfitting in classification



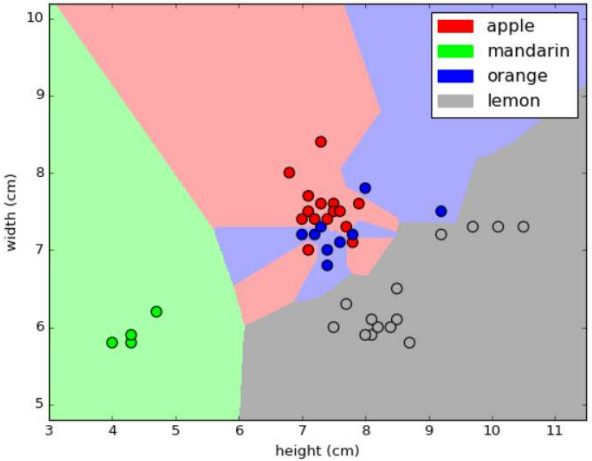
Overfitting with k-NN classifiers



K=10



K=5



K=1

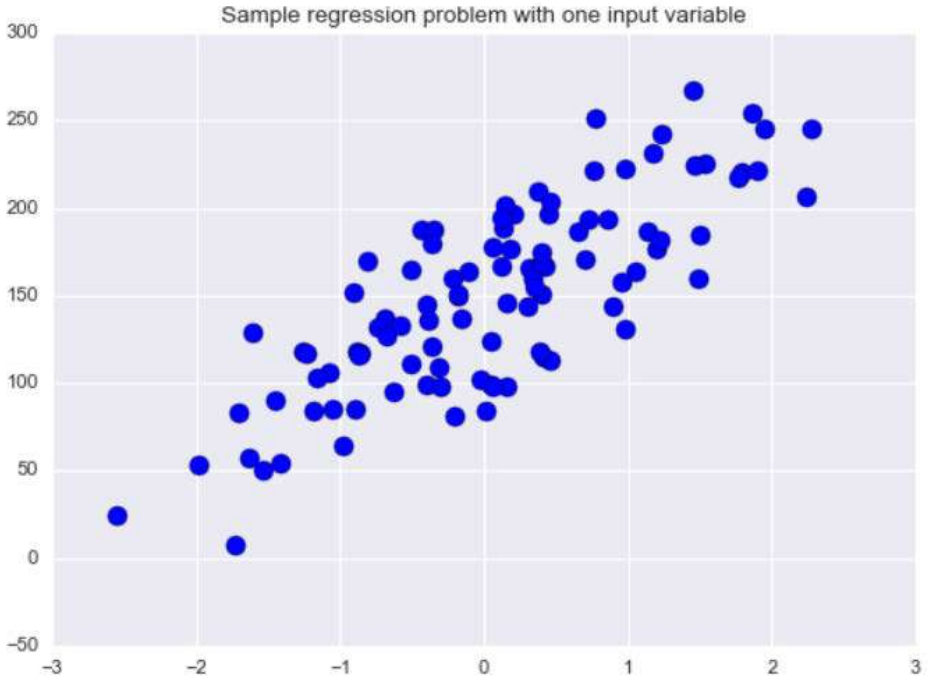
Applied Machine Learning

Supervised learning: Datasets

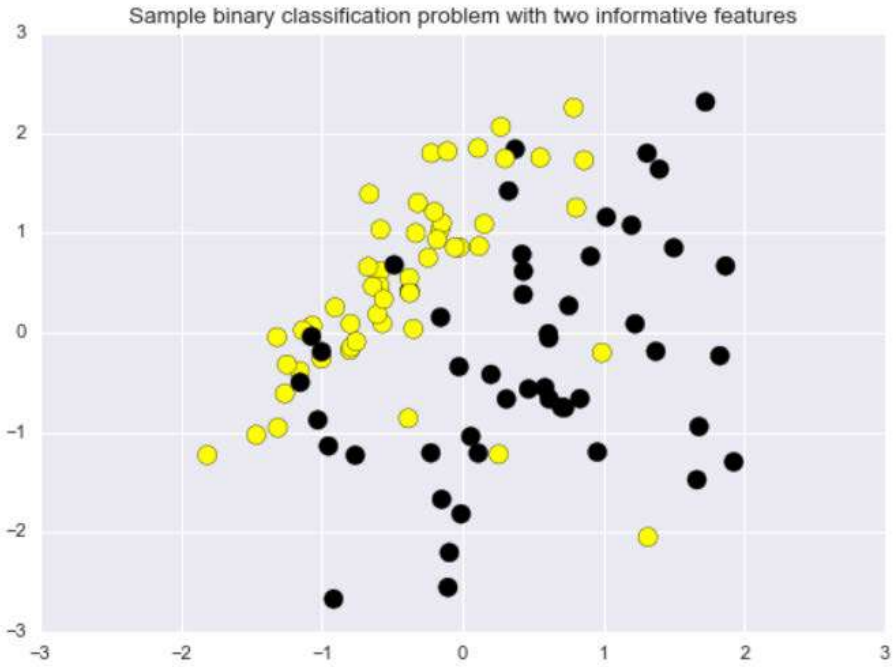
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

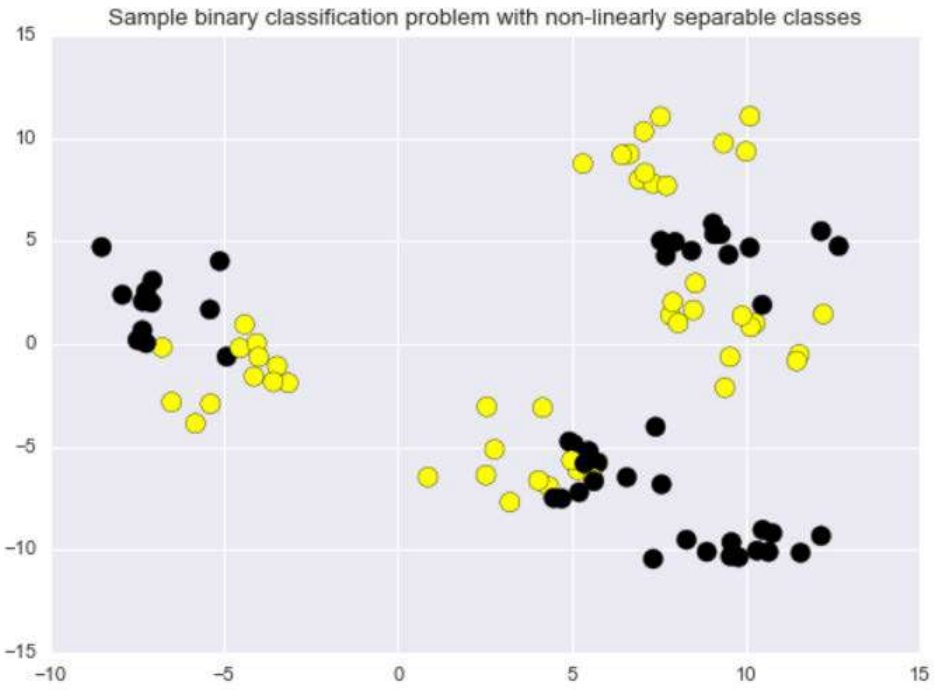
Simple Regression Dataset



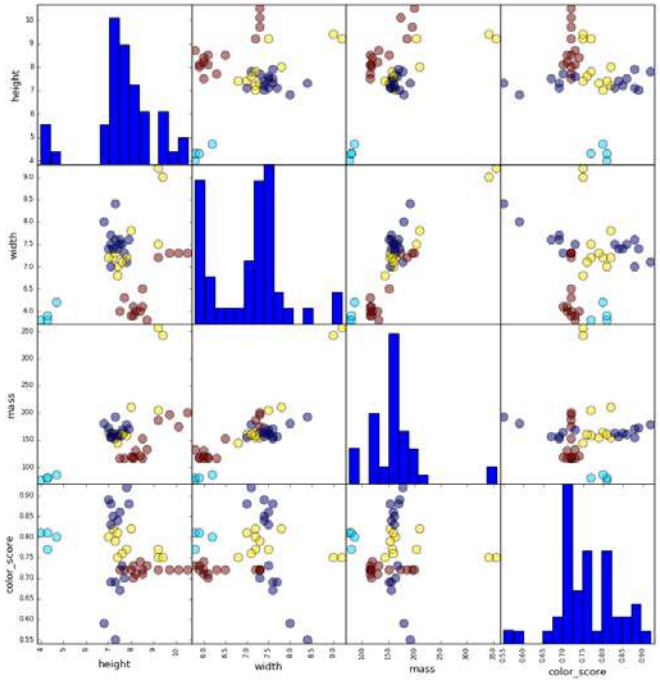
Simple Binary Classification Dataset



Complex Binary Classification Dataset



Fruits Multi-class Classification Dataset



Features

- **`width`**
- **`height`**
- **`mass`**
- **`color_index`**

Classes

0: apple

1: mandarin orange

2: orange

3: lemon

Communities and Crime Dataset

	population	householdsize	agePct12t21	agePct12t29	agePct16t24	agePct65up	numbUrban	pctUrban	medIncome	pctWWage	...	MedRentP
0	11980	3.10	12.47	21.44	10.93	11.33	11980	100.0	75122	89.24	...	23.8
1	23123	2.82	11.01	21.30	10.48	17.18	23123	100.0	47917	78.99	...	27.6
2	29344	2.43	11.36	25.88	11.01	10.28	29344	100.0	35669	82.00	...	24.1
3	16656	2.40	12.55	25.20	12.19	17.57	0	0.0	20580	68.15	...	28.7
5	140494	2.45	18.09	32.89	20.04	13.26	140494	100.0	21577	75.78	...	26.4
6	28700	2.60	11.17	27.41	12.76	14.42	28700	100.0	42805	79.47	...	24.4
7	59459	2.45	15.31	27.93	14.78	14.60	59449	100.0	23221	71.60	...	26.3
8	74111	2.46	16.64	35.16	20.33	8.58	74115	100.0	25326	83.69	...	25.2
9	103590	2.62	19.88	34.55	21.62	13.12	103590	100.0	17852	74.20	...	29.6
10	31601	2.54	15.73	28.57	15.16	14.26	31596	100.0	24763	73.92	...	23.8

Input features:
socio-economic data by location from U.S. Census

Target variable:
Per capita violent crimes

Derived from the original UCI dataset at:
https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unnormalized

```
from adspy_shared_utilities import load_crime_dataset
crime = load_crime_dataset()
```

Applied Machine Learning

K-Nearest Neighbors: Classification and Regression

Kevyn Collins-Thompson

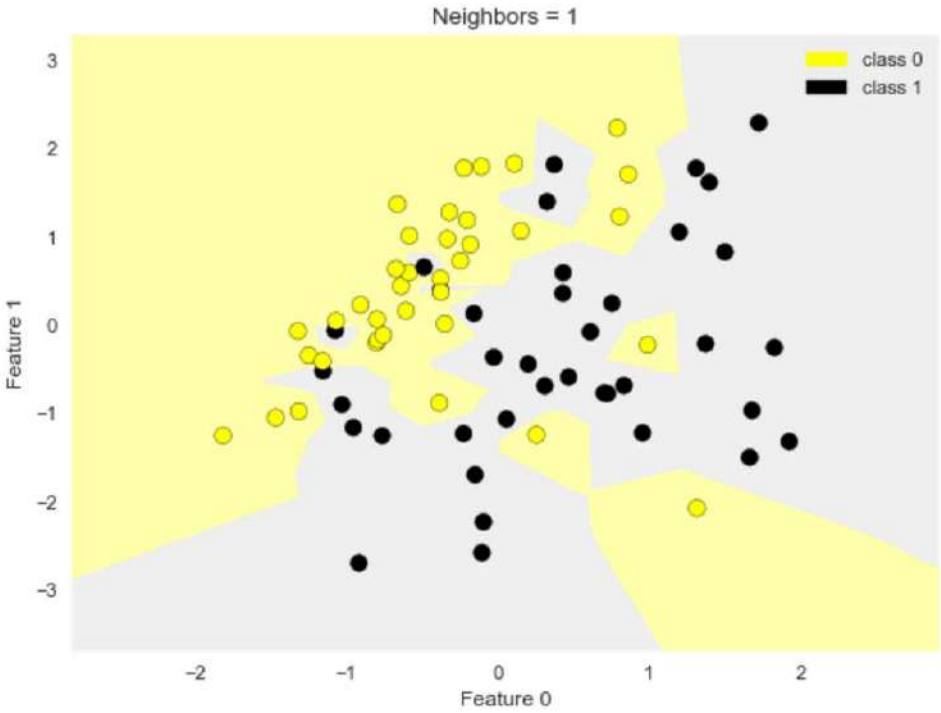
Associate Professor of Information & Computer Science
University of Michigan

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

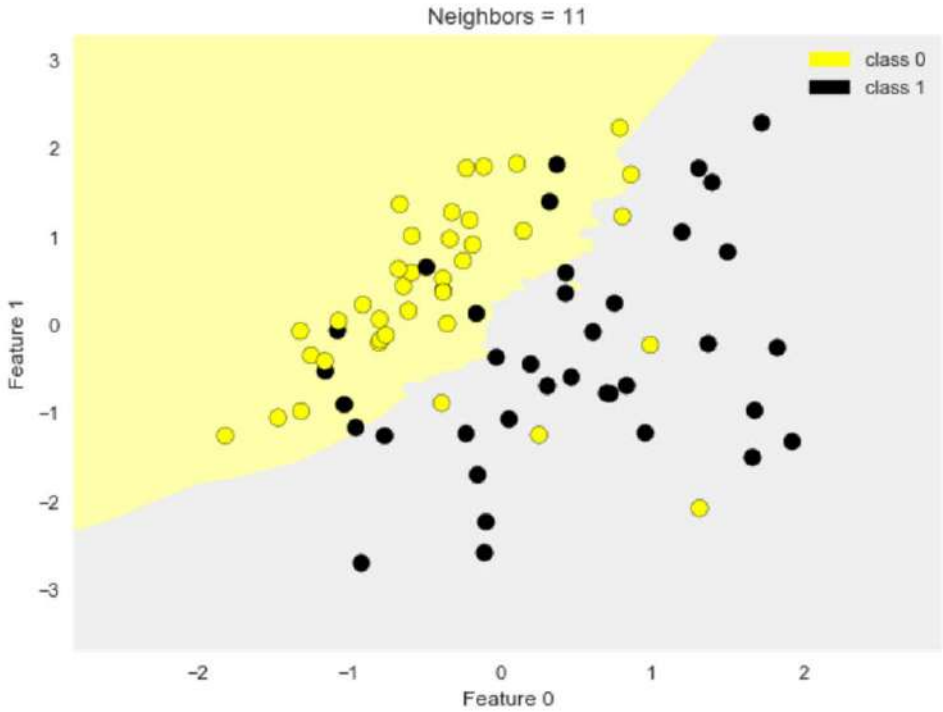
Given a training set X_train with labels y_train, and given a new instance x_test to be classified:

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

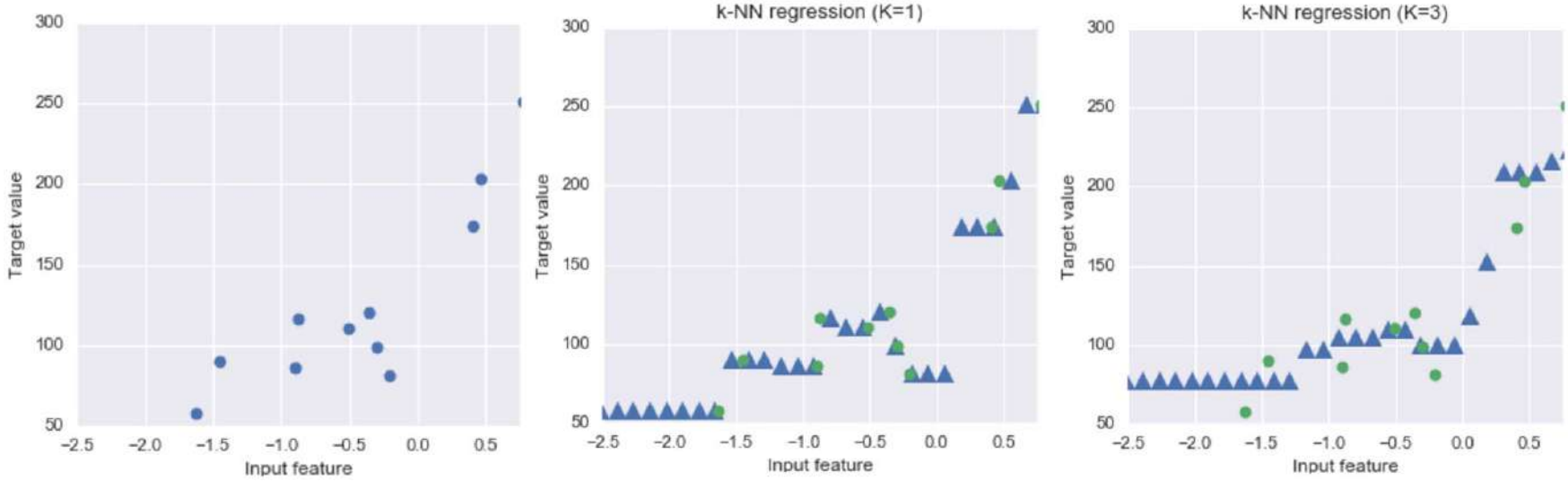
Nearest Neighbors Classification (k=1)



Nearest Neighbors Classification (k=11)



k-Nearest Neighbors Regression



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"**

KNeighborsClassifier and KNeighborsRegressor: important parameters

Model complexity

- ***n_neighbors :*** **number of nearest neighbors (k) to consider**
  - *Default = 5*

Model fitting

- ***metric: distance function between data points***
  - *Default: Minkowski distance with power parameter p = 2 (Euclidean)*

Applied Machine Learning

Linear Regression: Least-Squares

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

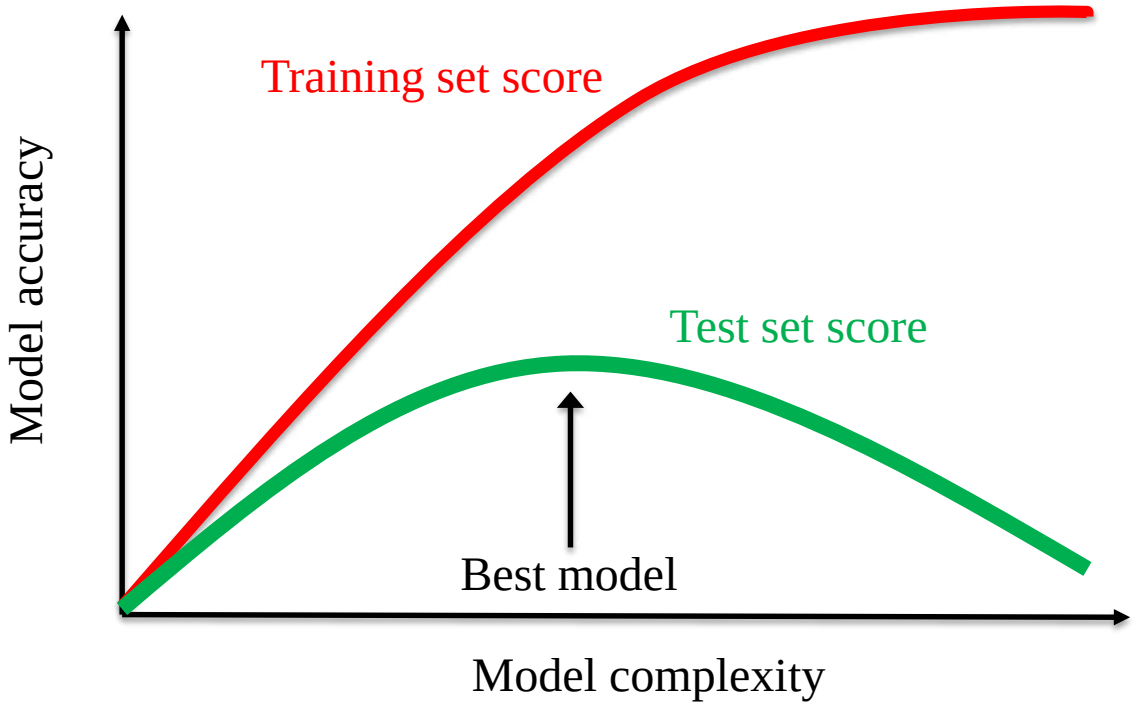
Supervised learning methods: Overview

- **To start with, we'll look at two simple but powerful prediction algorithms:**
  - *K-nearest neighbors (review from week 1, plus regression)*
  - *Linear model fit using least-squares*
- **These represent two complementary approaches to supervised learning:**
  - *K-nearest neighbors makes few assumptions about the structure of the data and gives potentially accurate but sometimes unstable predictions (sensitive to small changes in the training data).*
  - *Linear models make strong assumptions about the structure of the data and give stable but potentially inaccurate predictions.*

Supervised Learning Methods: Overview

- **We'll cover a number of widely-used supervised learning methods for classification and regression.**
- **For each supervised learning method we'll explore:**
  - *How the method works conceptually at a high level.*
  - *What kind of feature preprocessing is typically needed.*
  - *Key parameters that control model complexity, to avoid under- and over-fitting.*
  - *Positives and negatives of the learning method.*

The relationship between model complexity and training/test performance



Linear Models

- **A linear model is a <u>sum of weighted variables</u> that predicts a target output value given an input data instance. <u>Example</u>: *predicting housing prices***
  - *House features: taxes per year (X_{TAX}), age in years (X_{AGE})*

$$\widehat{Y_{PRICE}} = 212000 + 109\, X_{TAX} - 2000\, X_{AGE}$$

  - *A house with feature values (X_{TAX}, X_{AGE}) of (10000, 75) would have a predicted selling price of:*

$$\widehat{Y_{PRICE}} = 212000 + 109 \cdot \mathbf{10000} - 2000 \cdot \mathbf{75} = \mathbf{1{,}152{,}000}$$

Linear Regression is an Example of a Linear Model

Input instance – feature vector: $\boldsymbol{x} = (x_0, x_1, \ldots, x_n)$

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:

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

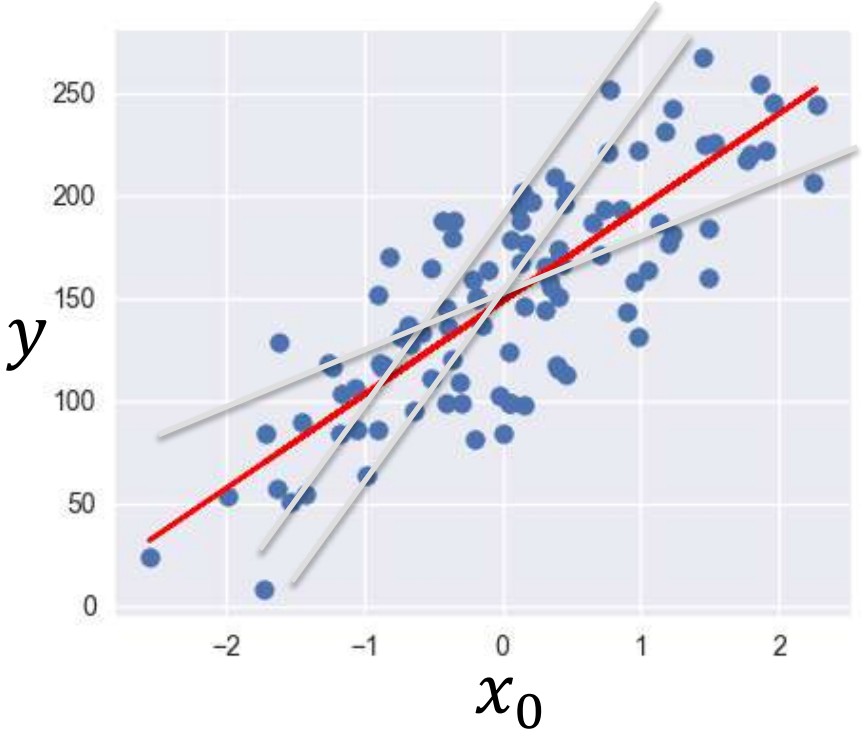
$\widehat{\boldsymbol{b}}$: constant bias term / intercept

A Linear Regression Model with one Variable (Feature)

Input instance: $\boldsymbol{x} = (x_0)$

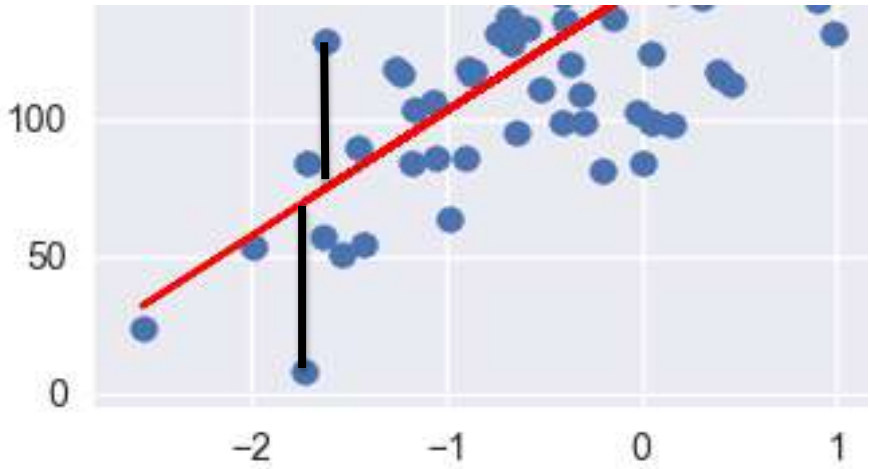
Predicted output: $\hat{y} = \widehat{w_0} x_0 + \hat{b}$

Parameters to estimate: $\widehat{w_0}$ (slope)
$\hat{b}$ (y-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.

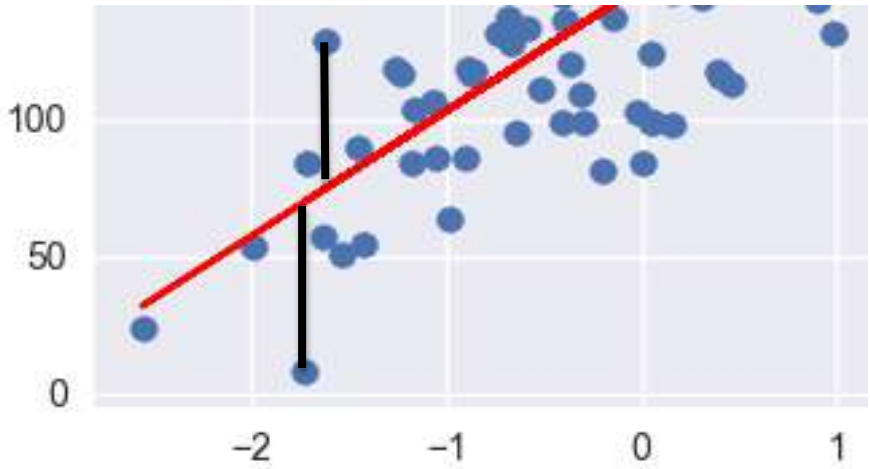


How are Linear Regression Parameters *w*, *b* Estimated?

- **Parameters are estimated from training data.**
- **There are many different ways to estimate *w* and *b*:**
  - *Different methods correspond to different "fit" criteria and goals and ways to control model complexity.*
- **The learning algorithm finds the parameters that optimize an <u>objective function</u>, typically to minimize some kind of <u>loss function</u> of the predicted target values *vs.* actual target values.**

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

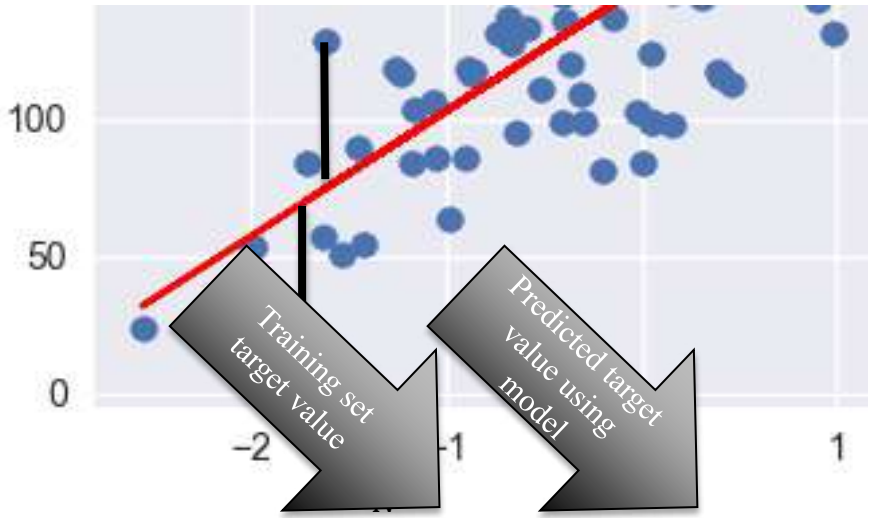
- Finds *w* and *b* that minimizes the <u>sum of squared differences</u> (RSS) over the training data between predicted target and actual target values.
- a.k.a. mean squared error of the linear model
- No parameters to control model complexity.



$$RSS(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2$$

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

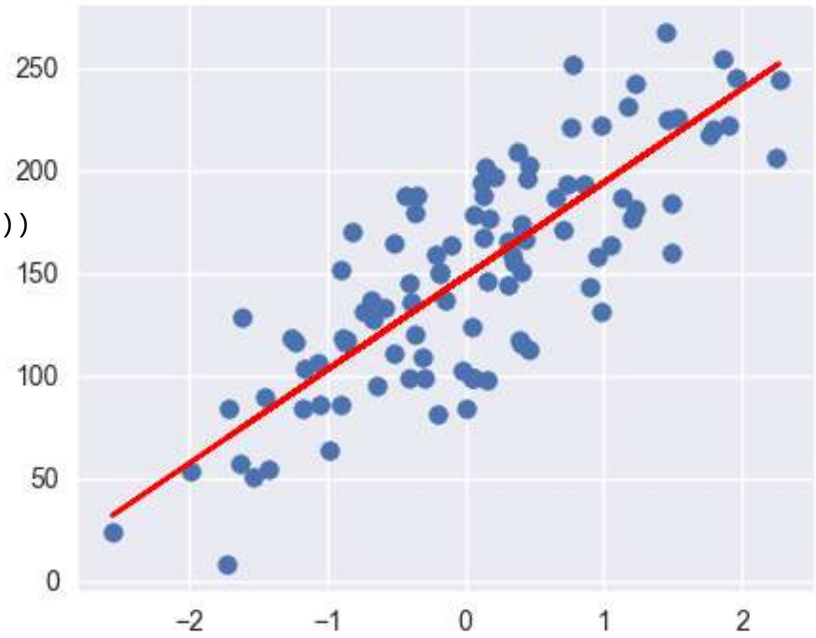
- Finds *w* and *b* that minimizes the <u>sum of squared differences</u> (RSS) over the training data between predicted target and actual target values.
- a.k.a. mean squared error of the linear model
- No parameters to control model complexity.



$$RSS(\boldsymbol{w}, b) = \sum_{\{i=1\}} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2$$

Least-Squares Linear Regression in Scikit-Learn

```
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_))
```



`linreg.coef_`

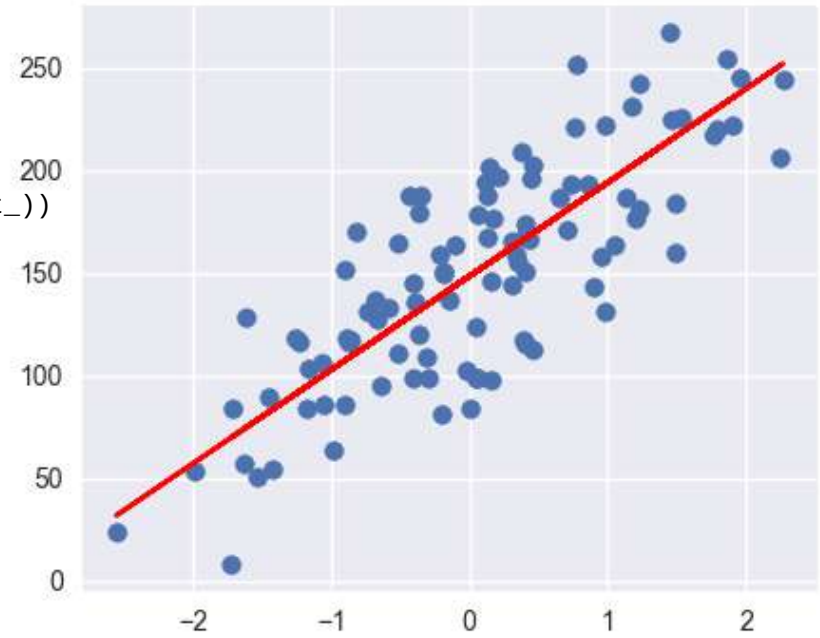
`linreg.intercept_`

$$\hat{y} = \boldsymbol{w_0} x_0 + b$$

Least-Squares Linear Regression in Scikit-Learn

```
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_))
```

Underscore denotes a quantity derived from training data, as opposed to a user setting.



`linreg.coef_`

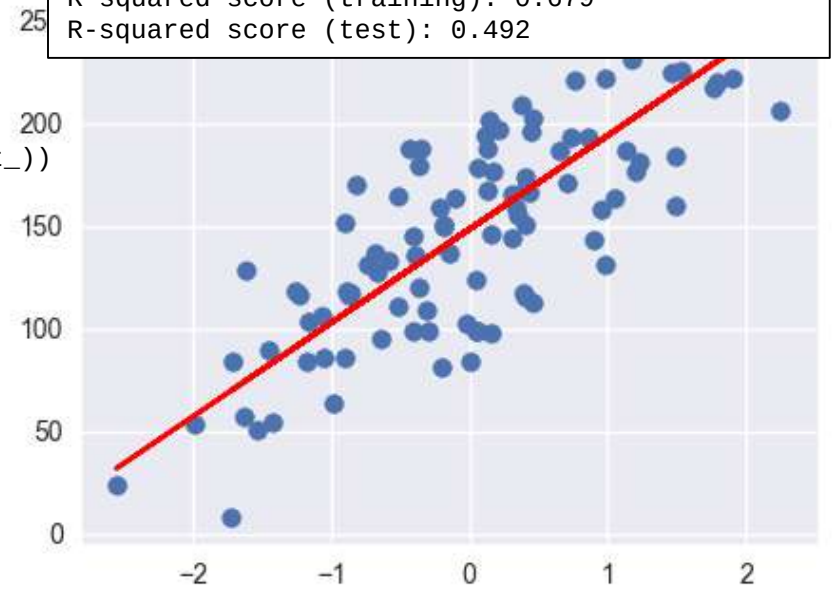
`linreg.intercept_`

$$\hat{y} = \boldsymbol{w_0} x_0 + b$$

Least-Squares Linear Regression in Scikit-Learn

```
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_))
```

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



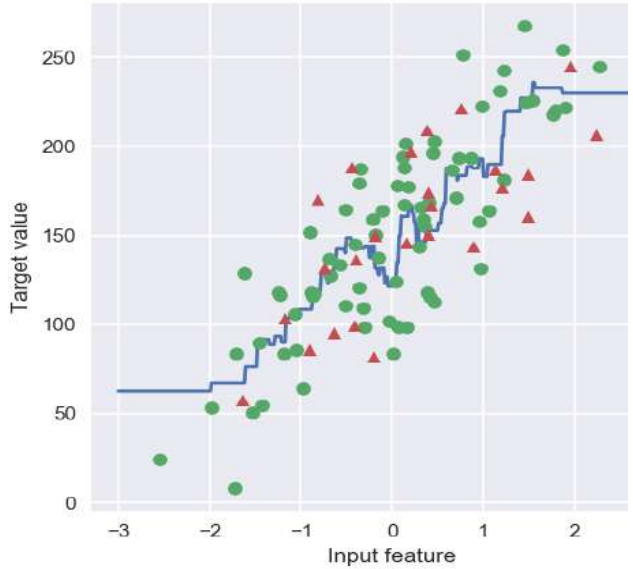
linreg.coef_ linreg.intercept_

$$\hat{y} = \boldsymbol{w_0} x_0 + b$$

K-NN Regression vs Least-Squares Linear Regression

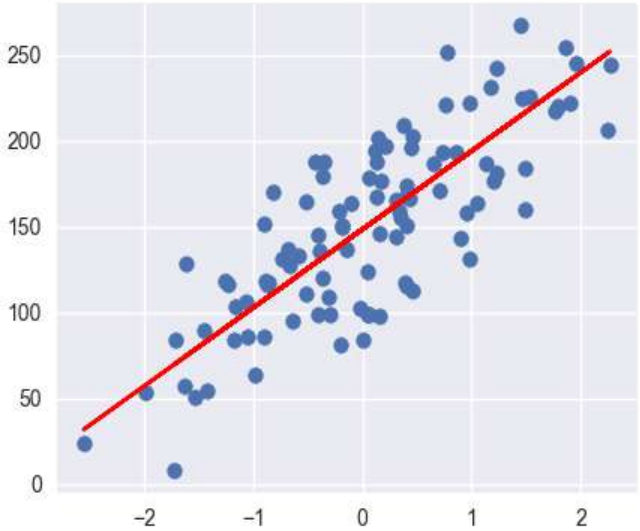
k-NN (k=7)

```
R-squared score (training): 0.720
R-squared score (test): 0.471
```



LS linear

```
R-squared score (training): 0.679
R-squared score (test): 0.492
```



Applied Machine Learning

Linear Regression:
Ridge, Lasso, and Polynomial Regression

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Ridge Regression

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

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

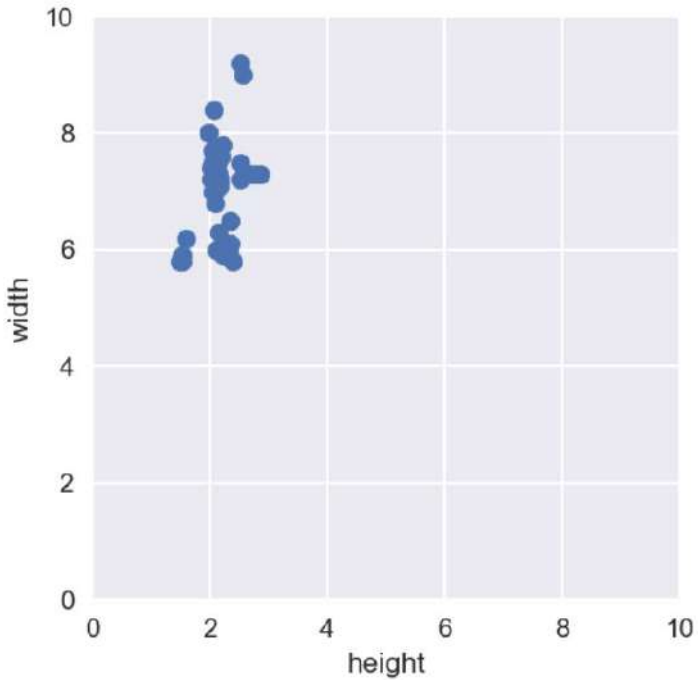
- **Once the parameters are learned, the ridge regression <u>prediction</u> formula is the <u>same</u> as ordinary least-squares.**
- **The addition of a parameter penalty is called <u>regularization</u>. Regularization prevents overfitting by restricting the model, typically to reduce its complexity.**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

The Need for Feature Normalization

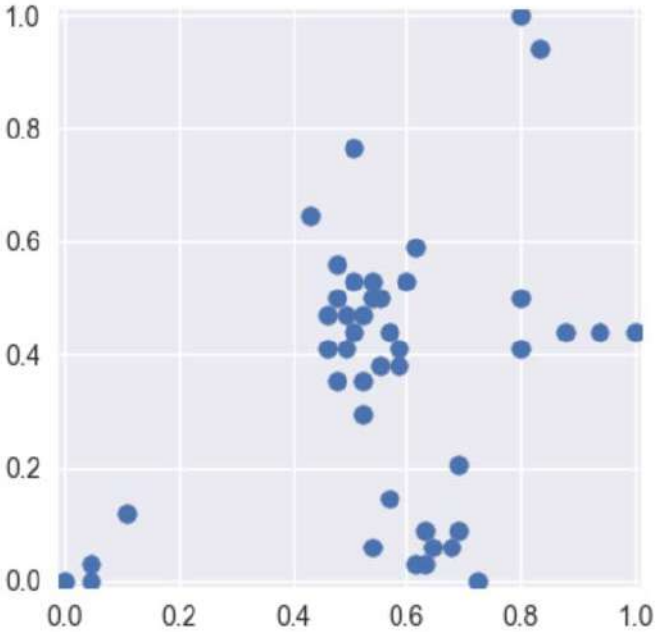
- **Important for some machine learning methods that all features are on the same scale (e.g. faster convergence in learning, more uniform or 'fair' influence for all weights)**
  - *e.g. regularized regression, k-NN, support vector machines, neural networks, …*
- **Can also depend on the data. More on feature engineering later in the course. For now, we do MinMax scaling of the features:**
  - *For each feature x_i : compute the min value x_i^{MIN} and the max value x_i^{MAX} achieved across all instances in the training set.*
  - *For each feature: transform a given feature x_i value to a scaled version x_i' using the formula*

$$x_i' = (x_i - x_i^{MIN})/(x_i^{MAX} - x_i^{MIN})$$

Feature Normalization with MinMaxScaler



Unnormalized data points



Normalized with MinMaxScaler

Using a scaler object: fit and transform methods

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
clf = Ridge().fit(X_train_scaled, y_train)
r2_score = clf.score(X_test_scaled, y_test)
```

Tip: It can be more efficient to do fitting and transforming together on the training set using the `fit_transform` method.

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

Feature Normalization: The test set must use identical scaling to the training set

- **Fit the scaler using the training set, then apply the same scaler to transform the test set.**
- **Do not scale the training and test sets using different scalers: this could lead to random skew in the data.**
- **Do not fit the scaler using any part of the test data: referencing the test data can lead to a form of *data leakage.* More on this issue later in the course.**

Lasso regression is another form of regularized linear regression that uses an <u>L1 regularization</u> penalty for training (instead of ridge's L2 penalty)

- **L1 penalty: Minimize the sum of the <u>absolute values</u> of the coefficients**

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

- **This has the effect of setting parameter weights in *w* to <u>zero</u> for the least influential variables. This is called a <u>sparse</u> solution: a kind of feature selection**
- **The parameter α controls amount of L1 regularization (default = 1.0).**
- **The prediction formula is the same as ordinary least-squares.**
- **When to use ridge vs lasso regression:**
  - *Many small/medium sized effects: use ridge.*
  - *Only a few variables with medium/large effect: use lasso.*

Lasso Regression on the Communities and Crime Dataset

For alpha = 2.0, 20 out of 88 features have non-zero weight.

Top features (sorted by abs. magnitude):

```
PctKidsBornNeverMar, 1488.365 # percentage of kids born to people who never married
PctKids2Par, -1188.740 # percentage of kids in family housing with two parents
HousVacant, 459.538  # number of vacant households
PctPersDenseHous, 339.045 # percent of persons in dense housing (more than 1 person/room)
NumInShelters, 264.932 # number of people in homeless shelters
```

Polynomial Features with Linear Regression

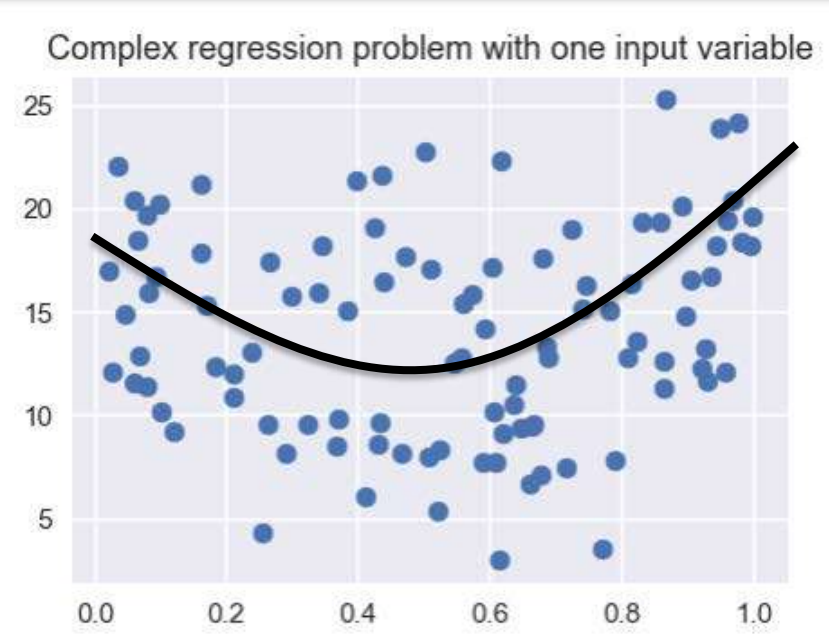
$$\boldsymbol{x} = (x_0, x_1) \Rightarrow \boldsymbol{x}' = (x_0, x_1, x_0^2, x_0 x_1, x_1^2)$$

$$\hat{y} = \hat{w}_0 x_0 + \hat{w}_1 x_1 + \hat{w}_{00}\, x_0^2 + \hat{w}_{01} x_0 x_1 + \hat{w}_{11} x_1^2 + b$$

- **Generate new features consisting of all polynomial combinations of the original two features (x_0, x_1).**
- **The *degree* of the polynomial specifies how many variables participate at a time in each new feature (above example: degree 2)**
- **This is still a weighted linear combination of features, so it's <u>still a linear model</u>, and can use same least-squares estimation method for *w* and *b*.**

Least-Squares Polynomial Regression

Original feature



With polynomial degree 2 (quadratic) feature

Polynomial Features with Linear Regression

- **Why would we want to transform our data this way?**
  - *To capture interactions between the original features by adding them as features to the linear model.*
  - *To make a classification problem easier (we'll see this later).*
- **More generally, we can apply other non-linear transformations to create new features**
  - *(Technically, these are called non-linear basis functions)*
- **Beware of polynomial feature expansion with high as this can lead to complex models that overfit**
  - *Thus, polynomial feature expansion is often combined with a regularized learning method like ridge regression.*

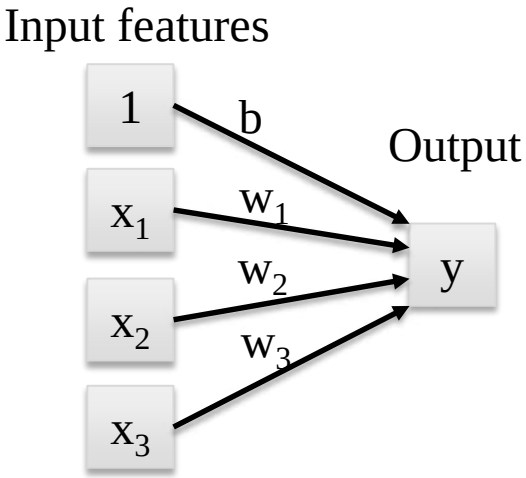
Applied Machine Learning

Logistic regression

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

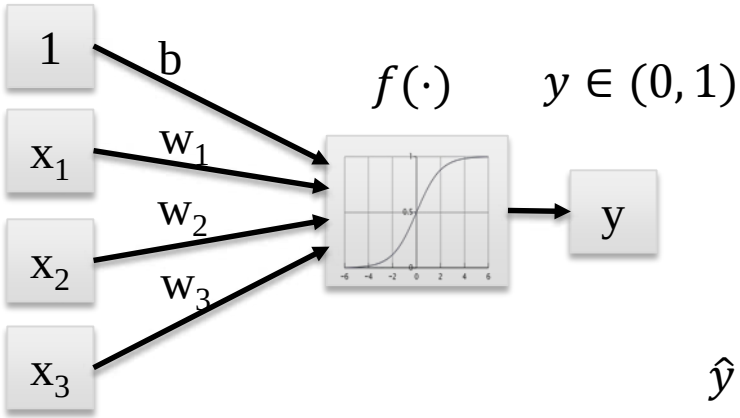
Linear Regression

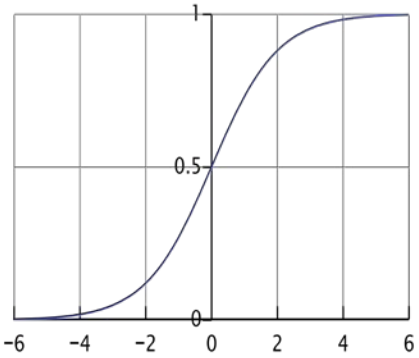


$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

Linear models for classification: Logistic Regression

Input features



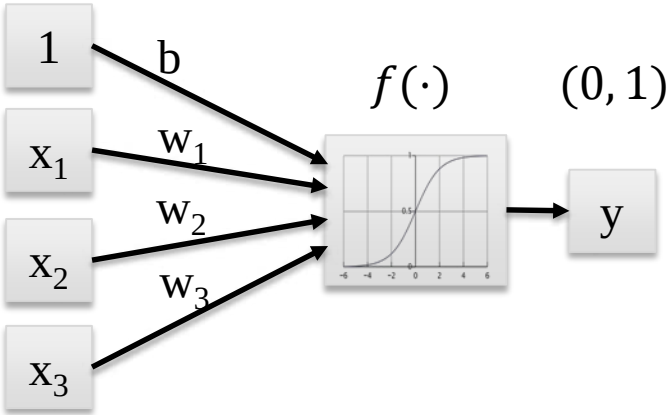


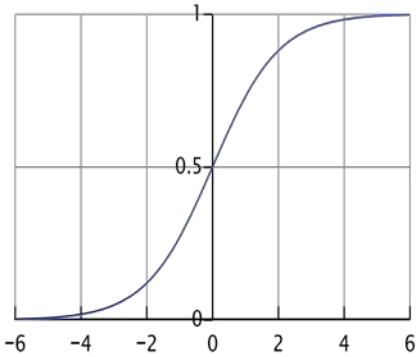
$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

$$= \frac{1}{1 + \exp\left[-\left(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n\right)\right]}$$

Linear models for classification: Logistic Regression

Input features



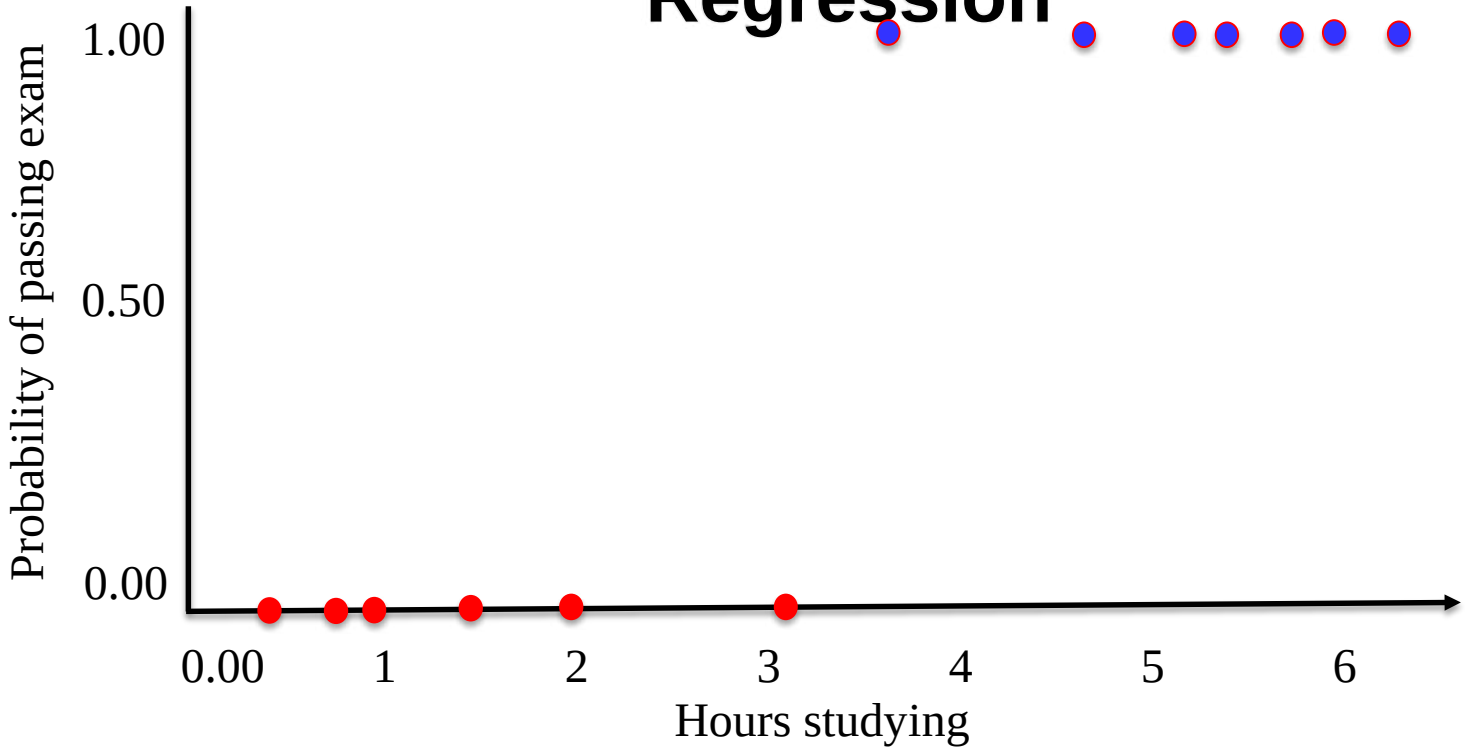


The logistic function transforms real-valued input to an output number y between 0 and 1, interpreted as the <u>probability</u> the input object belongs to the positive class, given its input features $(x_0, x_1, \ldots, x_n)$

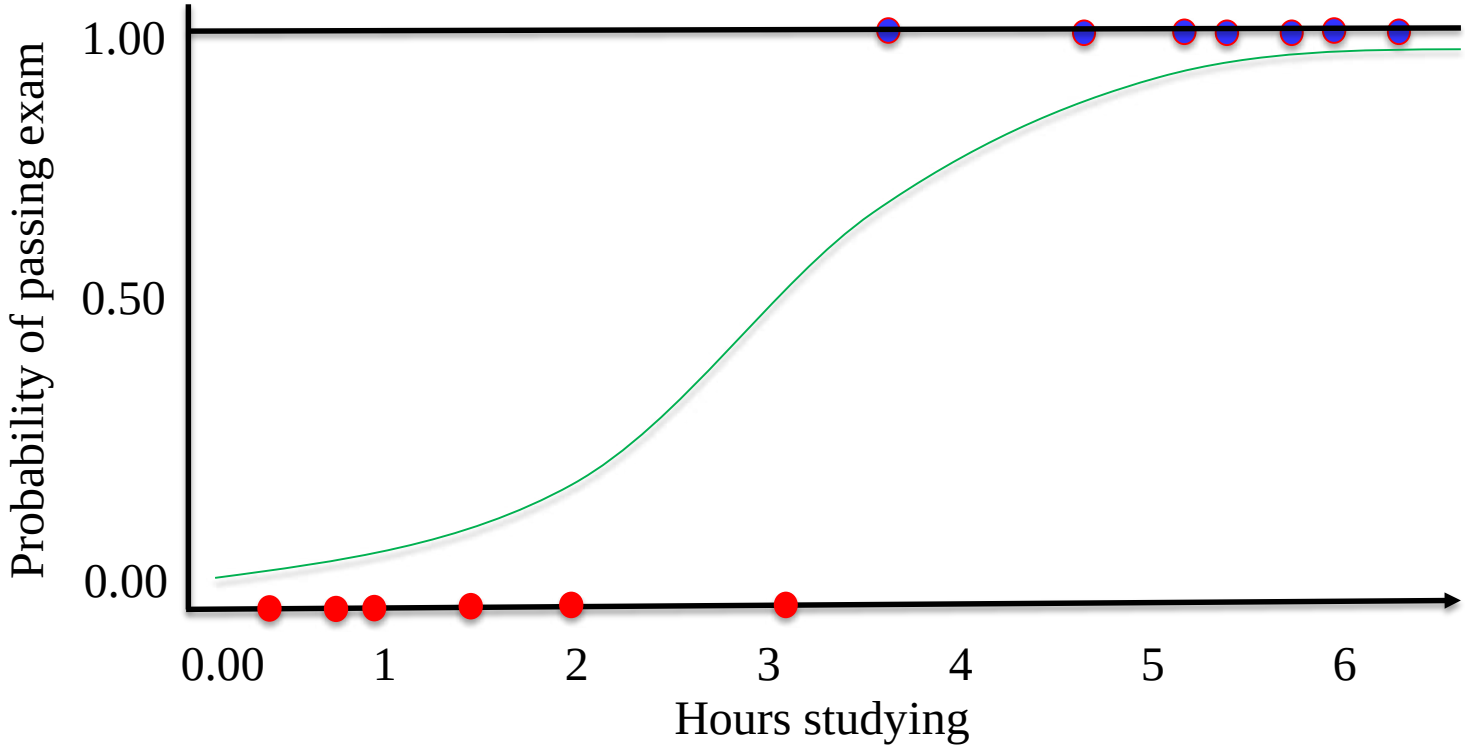
$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

$$= \frac{1}{1 + \exp\left[-(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)\right]}$$

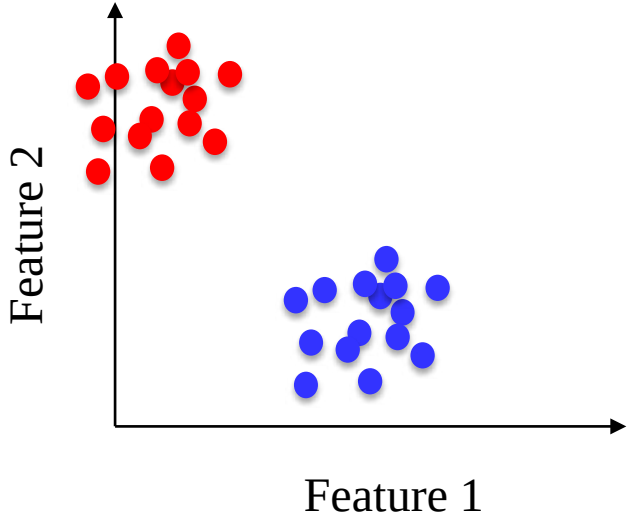
Linear models for classification: Logistic Regression



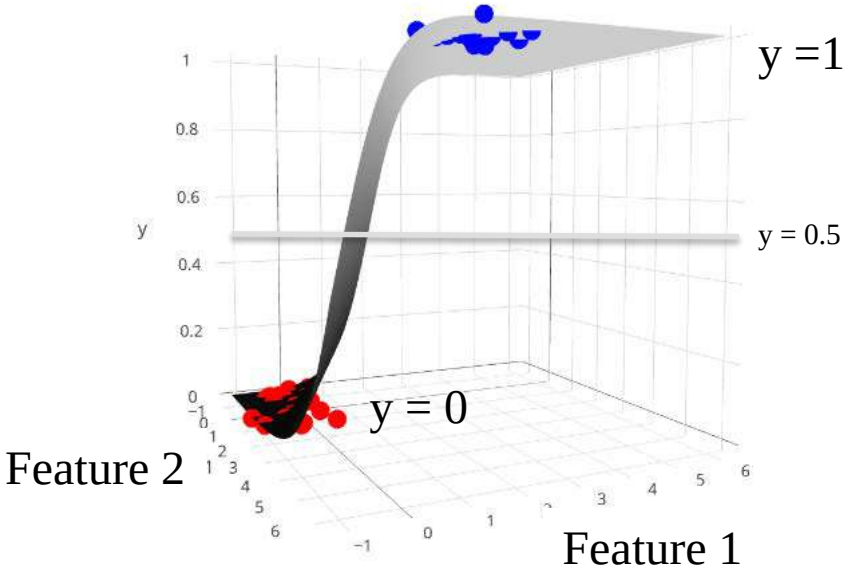
Linear models for classification: Logistic Regression



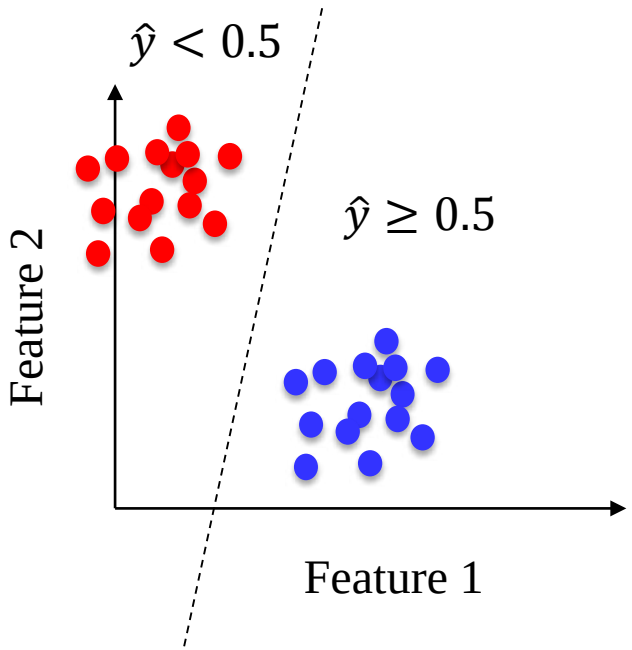
Logistic Regression for binary classification



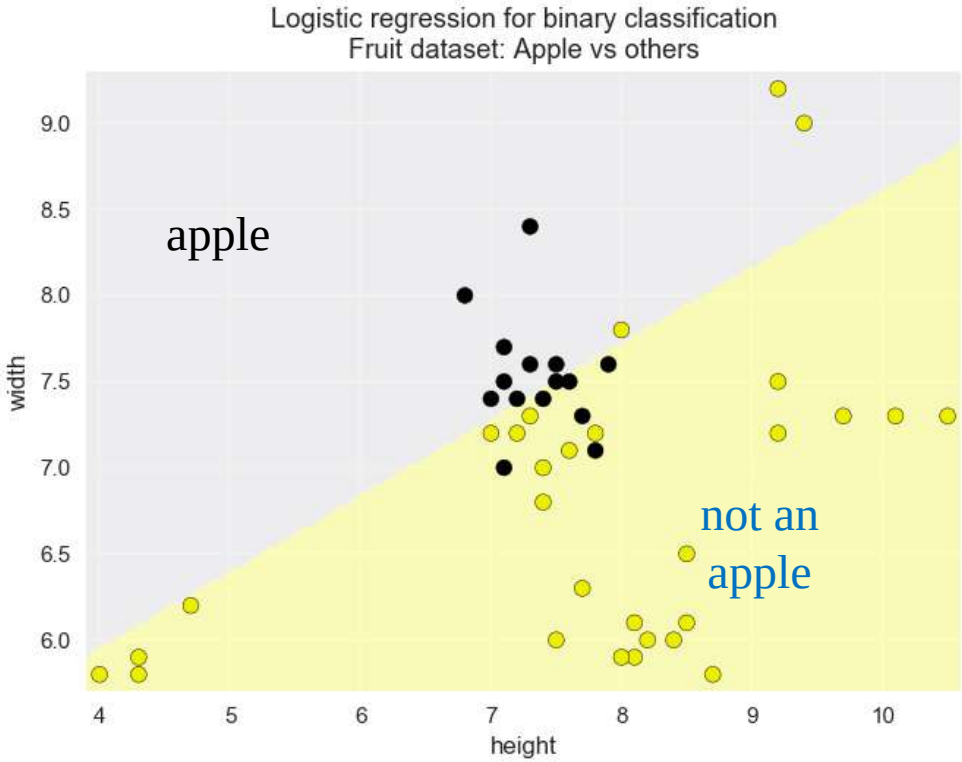
Logistic Regression for binary classification



Logistic Regression for binary classification

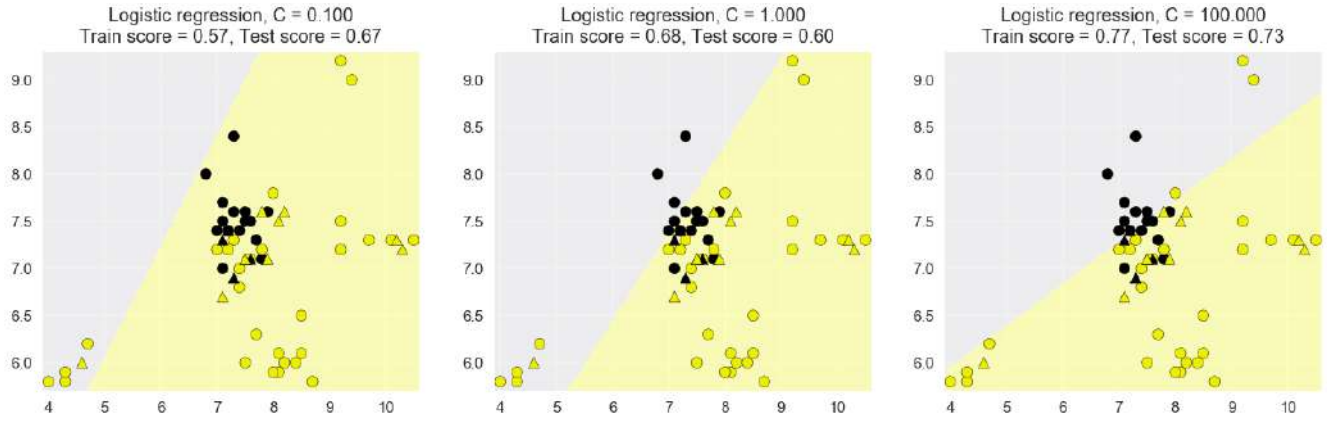


Simple logistic regression problem: two-class, two-feature version of the fruit dataset



Logistic Regression: Regularization

- **L2 regularization is 'on' by default (like ridge regression)**
- **Parameter C controls amount of regularization (default 1.0)**
- **As with regularized linear regression, it can be important to normalize all features so that they are on the same scale.**



Applied Machine Learning

Linear Classifiers:
Support Vector Machines

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Linear classifiers: how would you separate these two groups of training examples with a straight line?

Feature vector

Class value

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

$$= sign\left(\sum w[i]x[i] + \mathrm{b}\right)$$

Linear classifiers: how would you separate these two groups of training examples with a straight line?

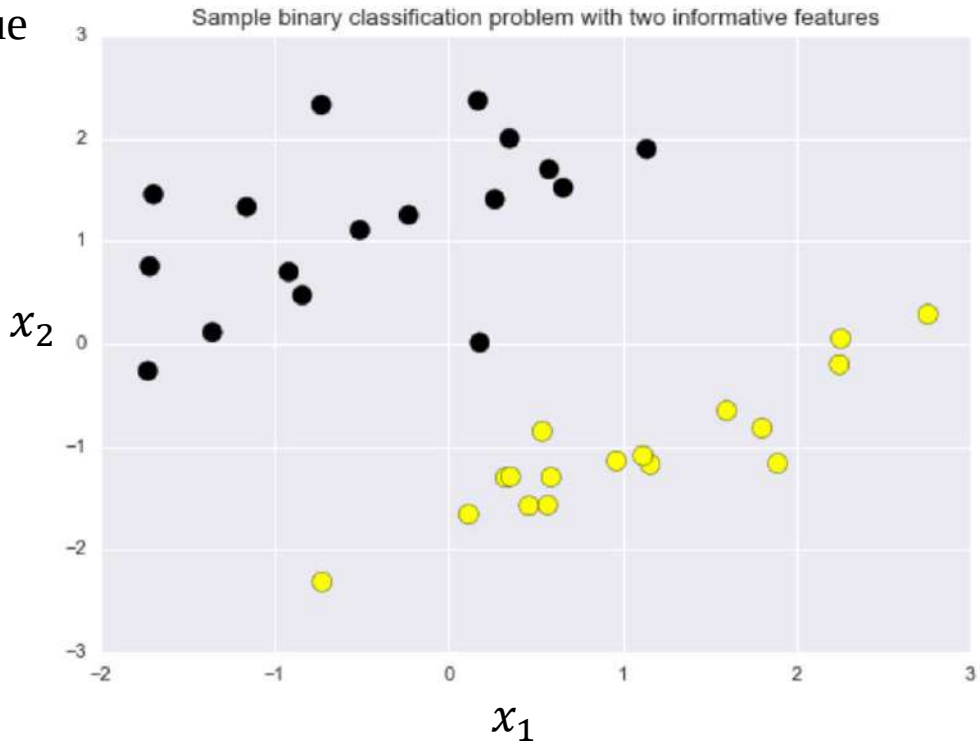
Feature vector

Class value

$$\boldsymbol{x} = (x_1, x_2) \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(\boldsymbol{x}, w, b) = sign(w \circ \boldsymbol{x} + b)$$

A linear classifier is a function that maps an input data point *x* to an output class value *y* (+1 or -1) using a linear function (with weight parameters *w* of the input point's features.



Linear classifiers: how would you separate these two groups of training examples with a line?

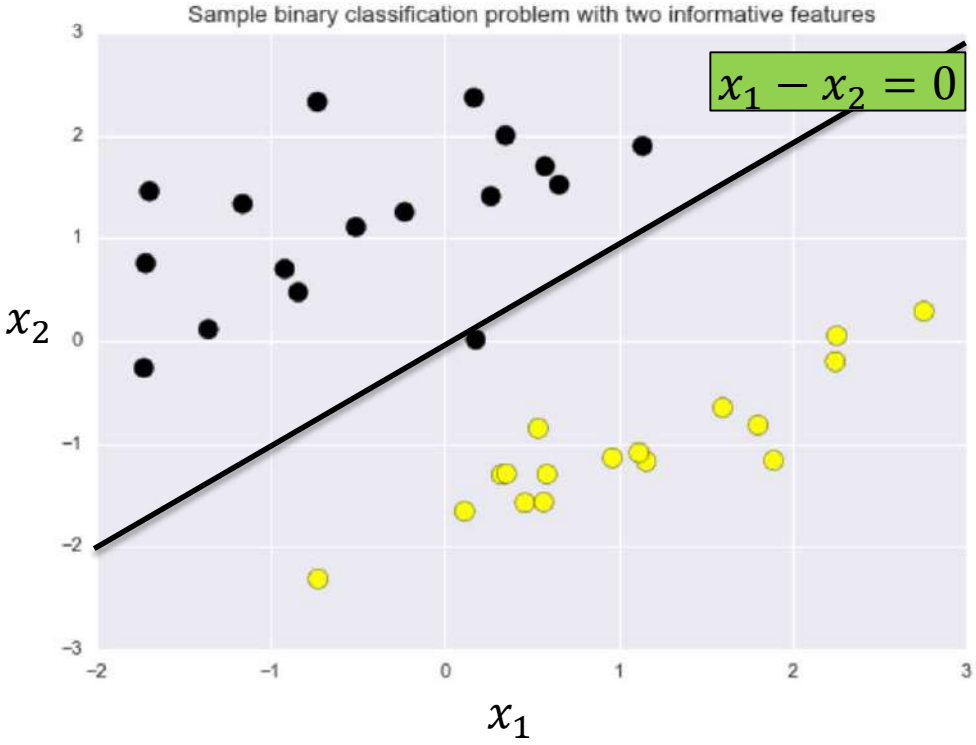
Feature vector

Class value

$$x \rightarrow \boxed{f} \rightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

$$x_1 - x_2 = 0$$
$$w = [1, -1]$$
$$b = 0$$



Linear classifiers: how would you separate these two groups of training examples with a line?

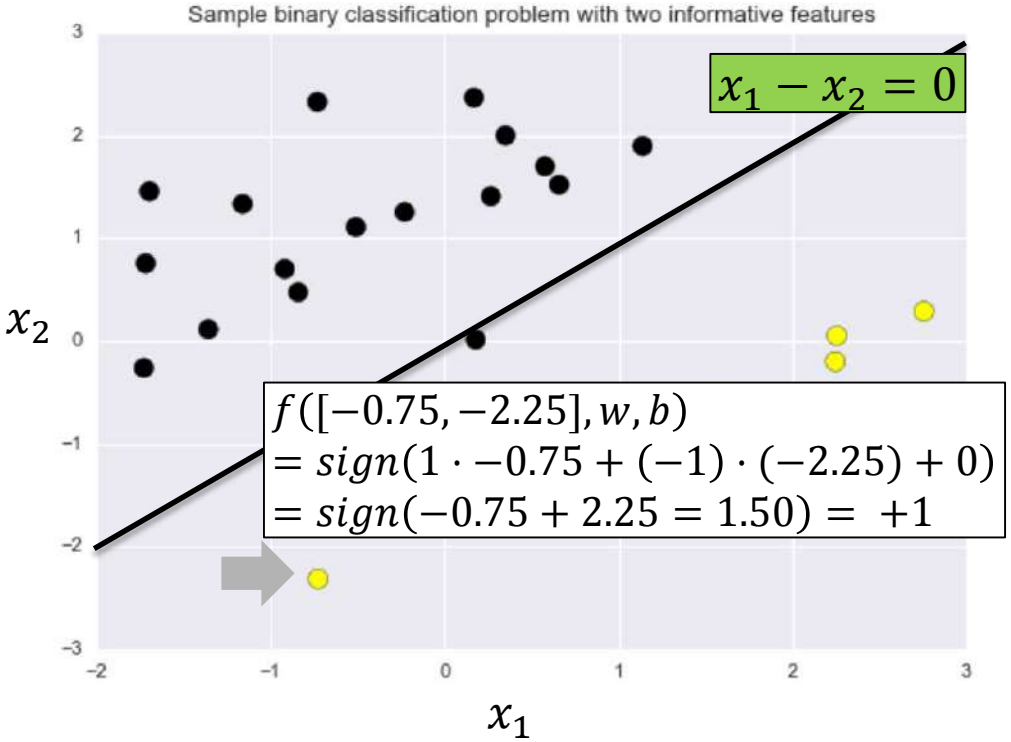
Feature vector

Class value

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

$$x_1 - x_2 = 0$$
$$w = [1, -1]$$
$$b = 0$$



Linear classifiers: how would you separate these two groups of training examples with a line?

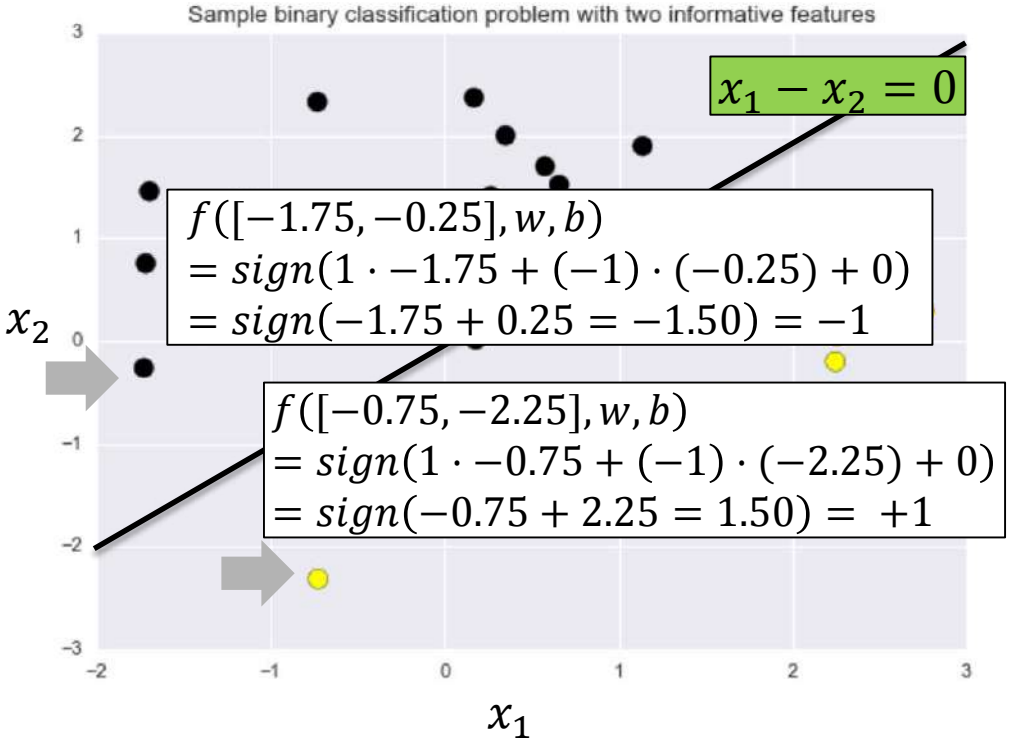
Feature vector

Class value

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

$$x_1 - x_2 = 0$$
$$w = [1, -1]$$
$$b = 0$$

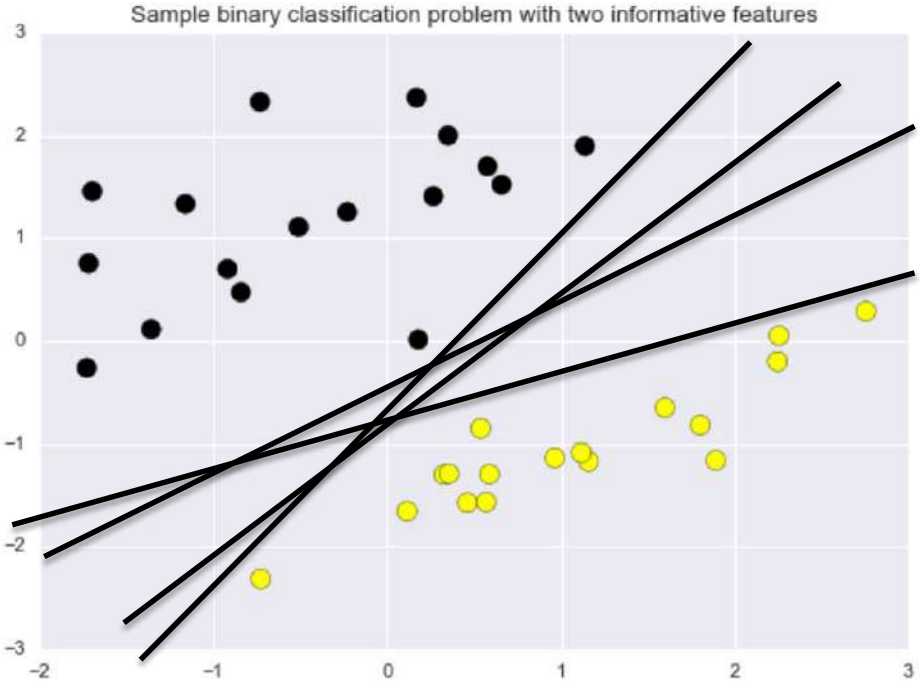


Linear Classifiers

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

There are many possible linear classifiers that could separate the two classes. **Which one is best?**



Classifier Margin

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

<u>Classifier margin</u>

Defined as the maximum width the decision boundary area can be increased before hitting a data point.

Maximum margin linear classifier: Linear Support Vector Machines

$$x \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$f(x, w, b) = sign(w \circ x + b)$$

<u>Maximum margin classifier</u>

The linear classifier with maximum margin is a linear Support Vector Machine (LSVM).

Regularization for SVMs: the C parameter

- **The strength of regularization is determined by C**
- **Larger values of C: less regularization**
  - *Fit the training data as well as possible*
  - *Each individual data point is important to classify correctly*
- **Smaller values of C: more regularization**
  - *More tolerant of errors on individual data points*

Linear Models: Pros and Cons

Pros:

- **Simple and easy to train.**
- **Fast prediction.**
- **Scales well to very large datasets.**
- **Works well with sparse data.**
- **Reasons for prediction are relatively easy to interpret.**

Cons:

- **For lower-dimensional data, other models may have superior generalization performance.**
- **For classification, data may not be linearly separable (more on this in SVMs with non-linear kernels)**

linear_model: Important Parameters

Model complexity

- **alpha: weight given to the L1 or L2 regularization term in regression models**
  - *default = 1.0*
- **C: regularization weight for LinearSVC and LogisticRegression classification models**
  - *default = 1.0*

Applied Machine Learning

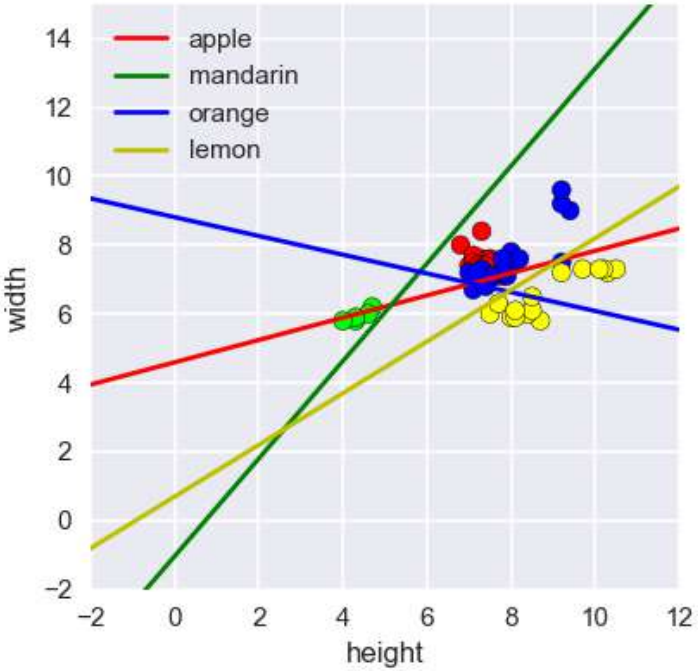
Multi-Class Classification

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

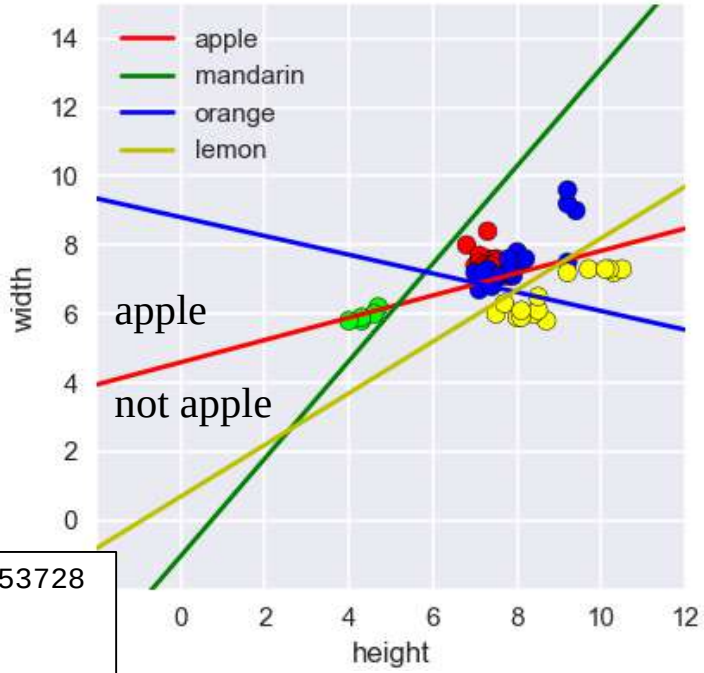
Multi-Class classification with linear models

```
clf = LinearSVC(C=5, random_state = 67)
clf.fit(X_train, y_train)

print(clf.coef_)

[[-0.23401135  0.72246132]
 [-1.63231901  1.15222281]
 [ 0.0849835   0.31186707]
 [ 1.26189663 -1.68097   ]]

print(clf.intercept_)
[-3.31753728  1.19645936 -2.7468353  1.16107418]
```



Multi-Class classification with linear models

```
clf = LinearSVC(C=5, random_state = 67)
clf.fit(X_train, y_train)

print(clf.coef_)

[[-0.23401135  0.72246132]
 [-1.63231901  1.15222281]
 [ 0.0849835   0.31186707]
 [ 1.26189663 -1.68097   ]]

print(clf.intercept_)
[-3.31753728  1.19645936 -2.7468353  1.16107418]
```




```
y_apple = -0.23401135 * height + 0.72246132 * width – 3.31753728

height=2, width=6:  y_apple = + 0.549 (>= 0: predict apple)
height=2, width=2:  y_apple = - 2.340 (< 0:  predict other)
```

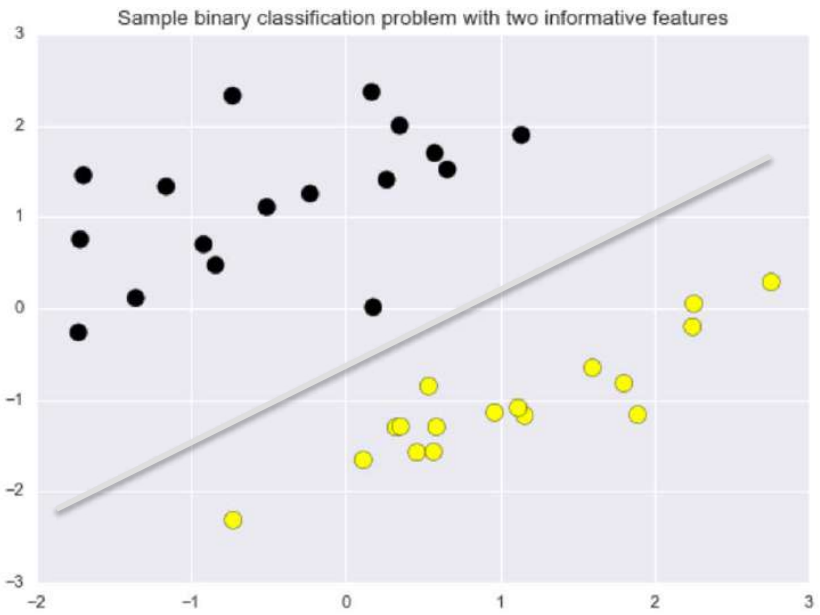
Applied Machine Learning

Kernelized Support Vector Machines

Kevyn Collins-Thompson

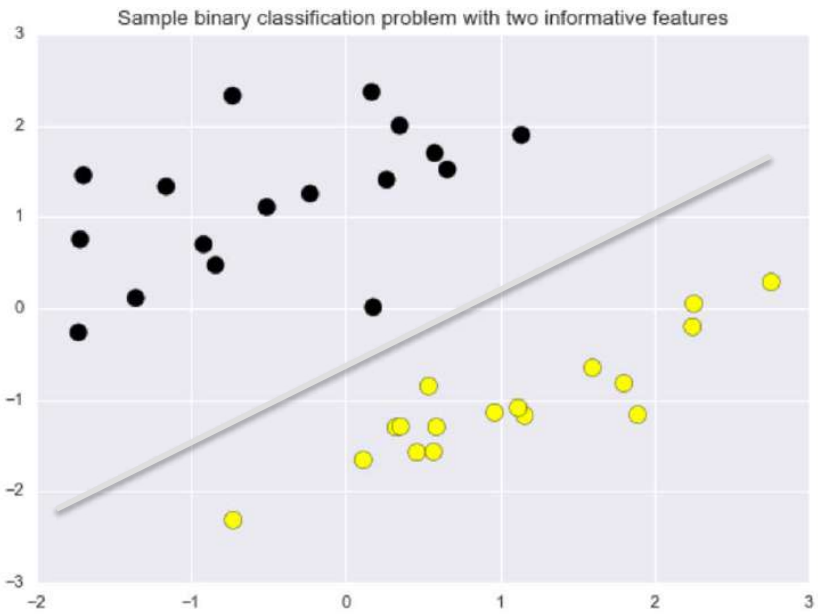
Associate Professor of Information & Computer Science
University of Michigan

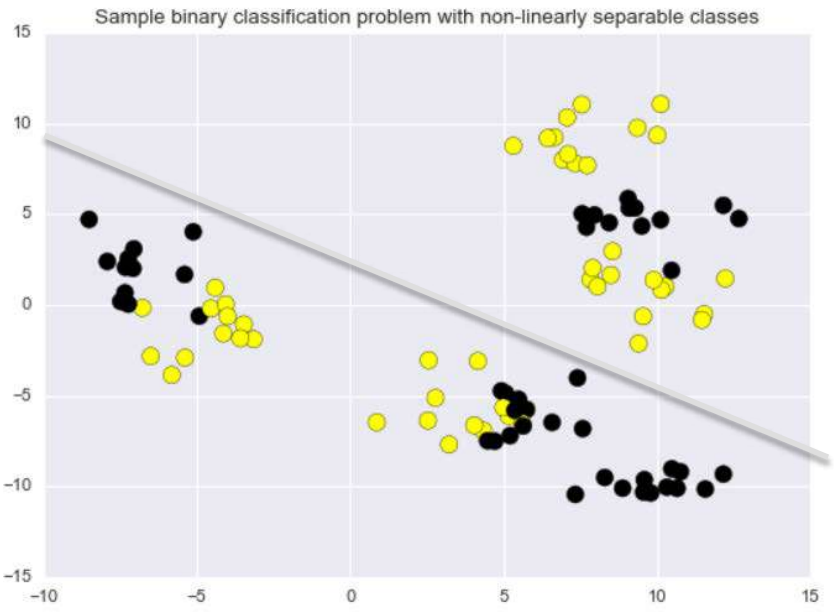
We saw how linear support vector classifiers could effectively find a decision boundary with maximum margin



Easy for a linear classifier

But what about more complex binary classification problems?

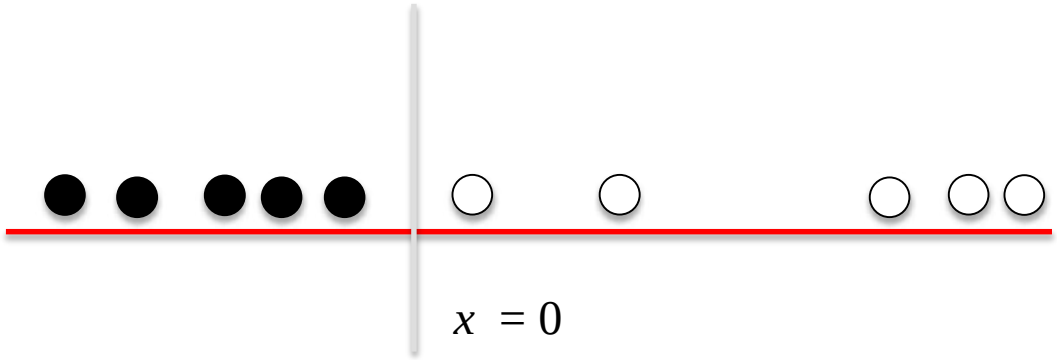




Easy for a linear classifier

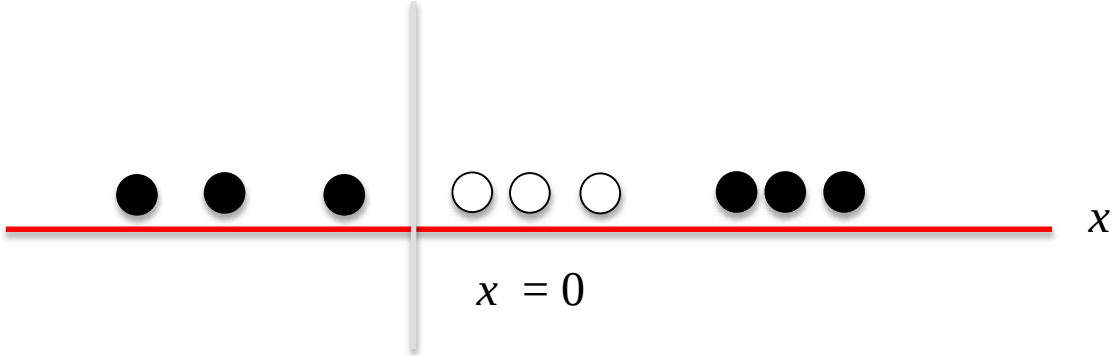
Difficult/impossible for a linear classifier

A simple 1-dimensional classification problem for a linear classifier

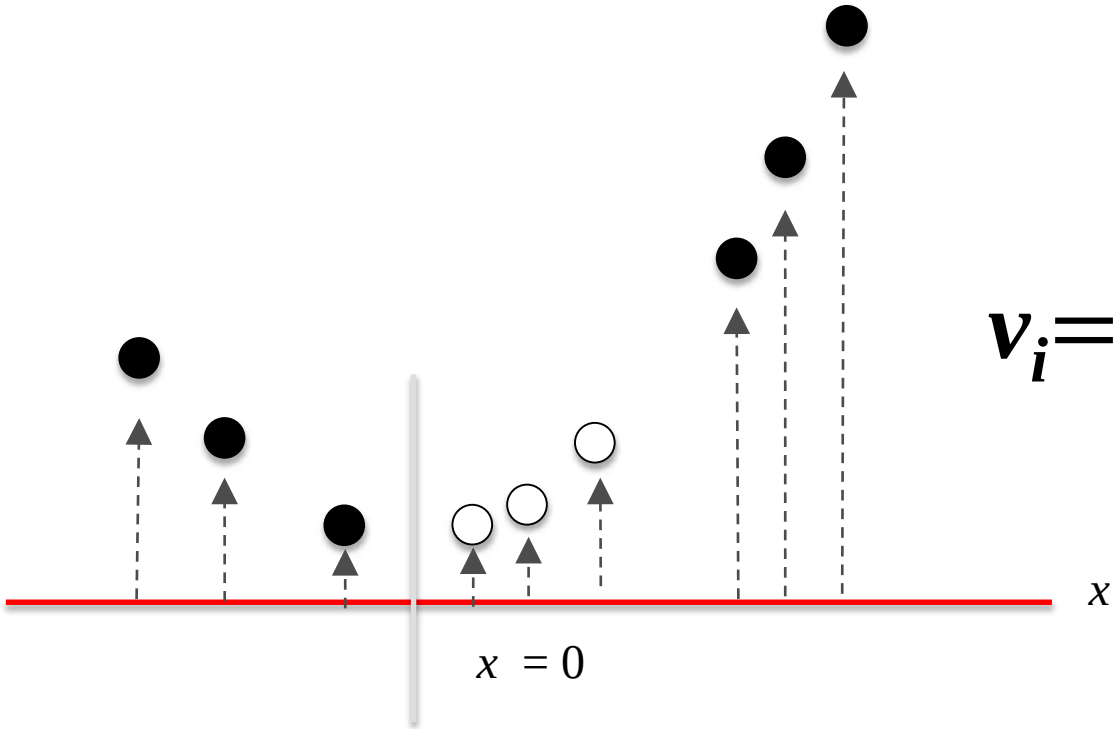


A more perplexing 1-d classification problem for a linear classifier

A more perplexing 1-d classification problem for a linear classifier

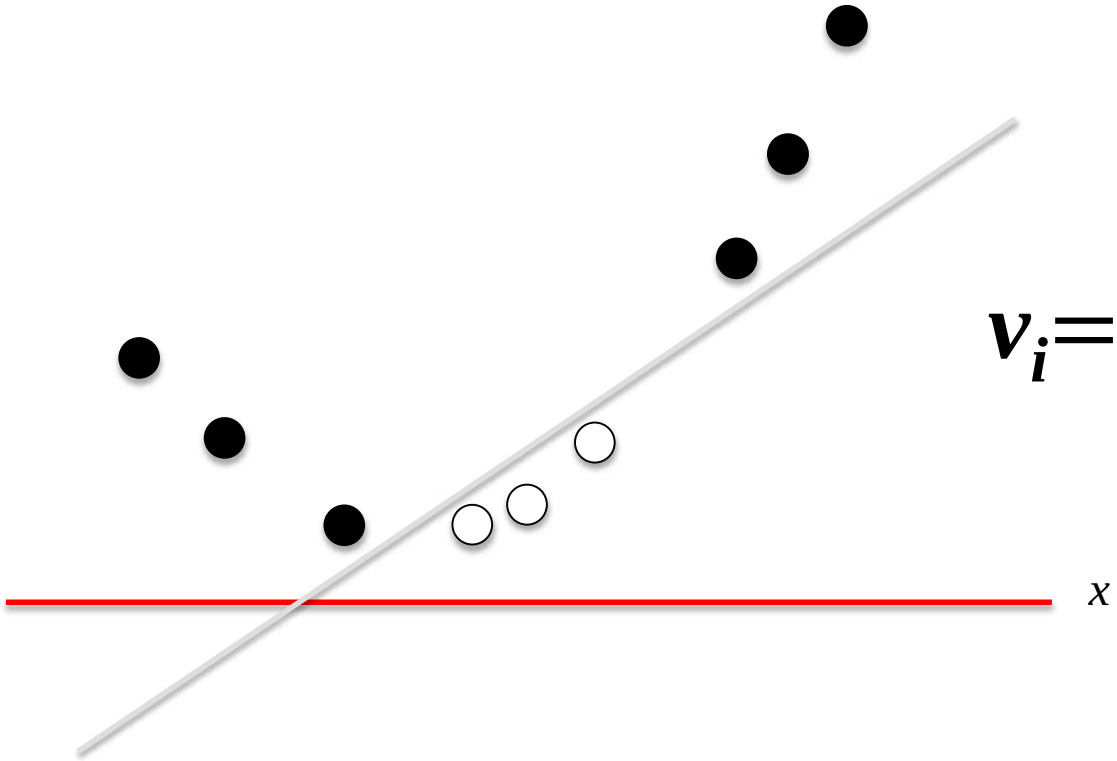


Let's transform the data by adding a second dimension/feature (set to the squared value of the first feature)



$$\boldsymbol{v_i} = (x_i, x_i^2)$$

The data transformation makes it possible to solve this with a linear classifier

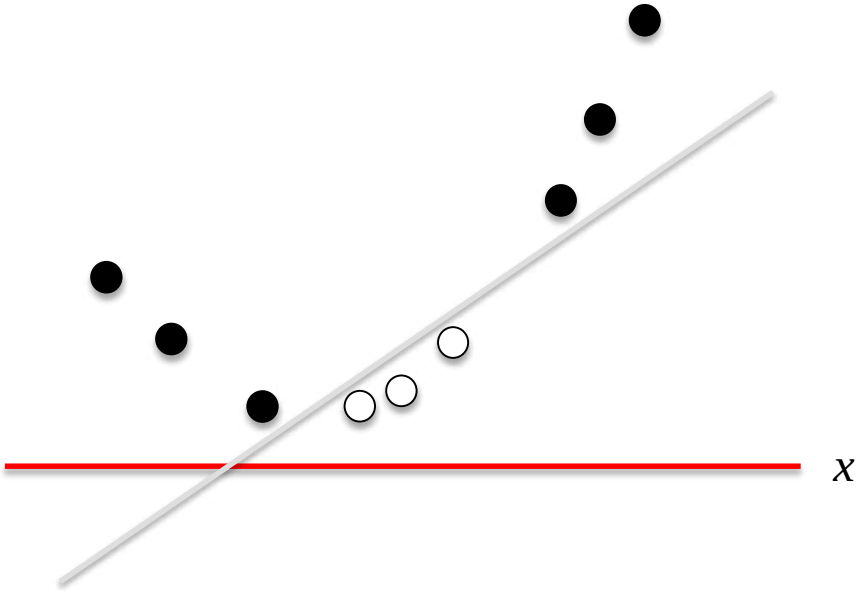


$$\boldsymbol{v_i} = (x_i, x_i^2)$$

What does the linear decision boundary in feature space correspond to in the original input space?

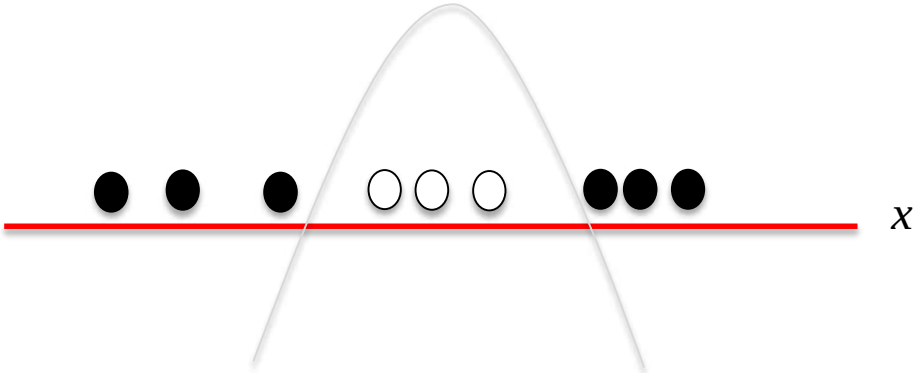
Original input space

Feature space

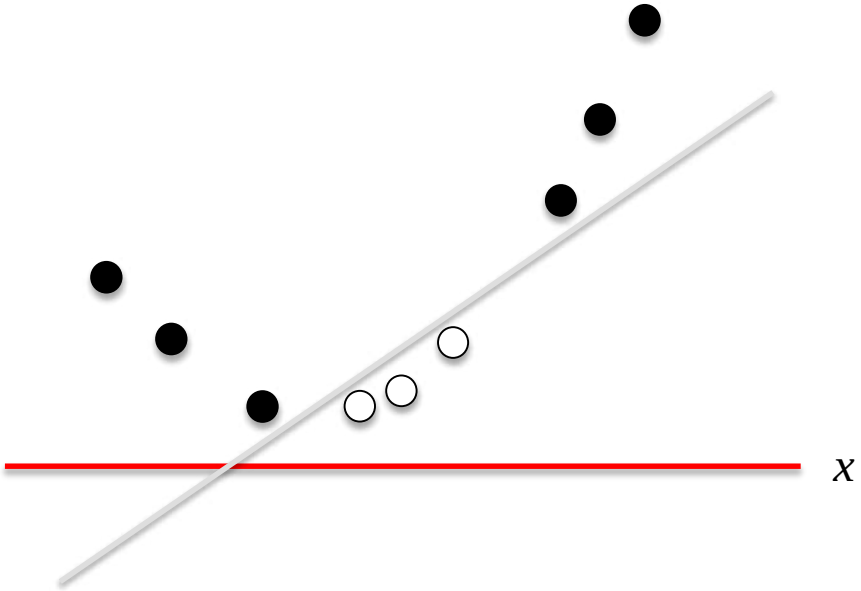


What does the linear decision boundary correspond to in the original input space?

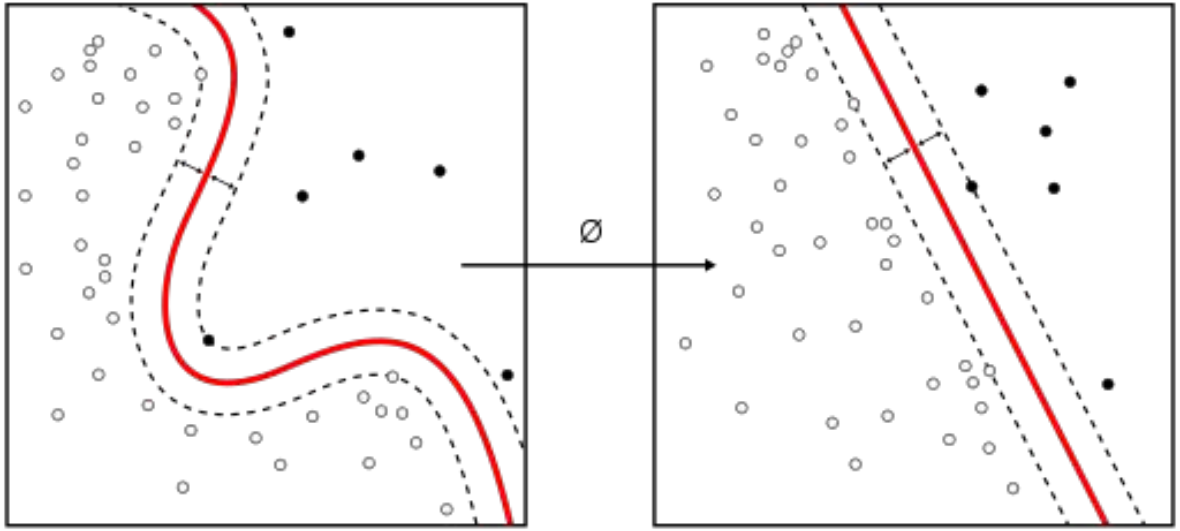
Original input space



Feature space



Transforming the data can make it much easier for a linear classifier.

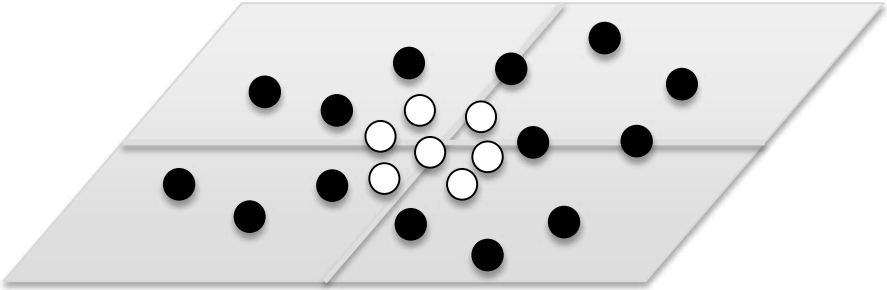


Original input space **Feature space**

Source: Wikipedia "Kernel Machine" article.
https://commons.wikimedia.org/w/index.php?curid=47868867

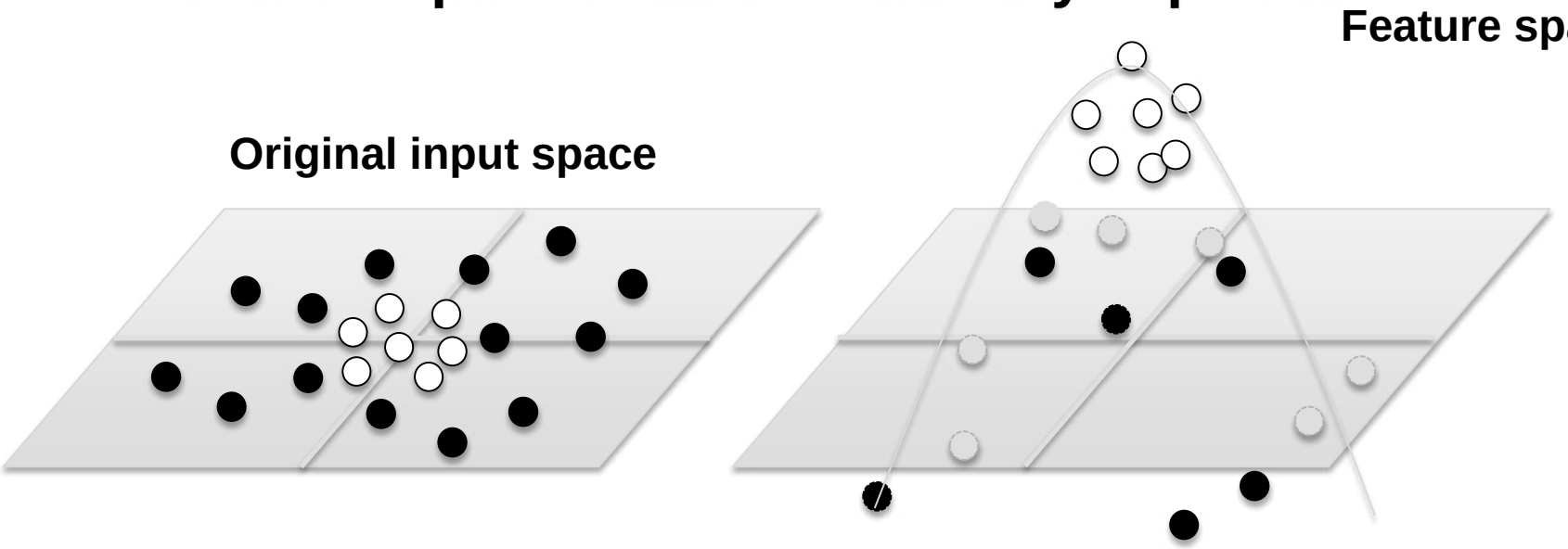
Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

Original input space



$$\boldsymbol{x_i} = (x_0, x_1)$$

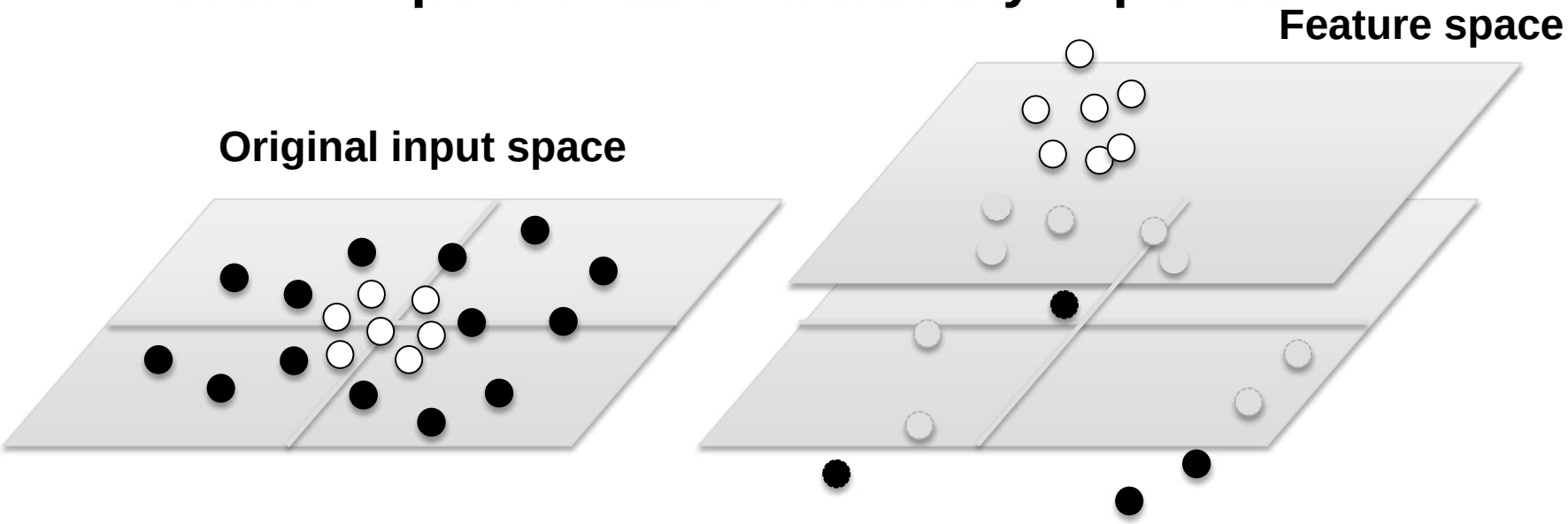
Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable



$$\boldsymbol{x_i} = (x_0, x_1)$$

$$\boldsymbol{v_i} = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

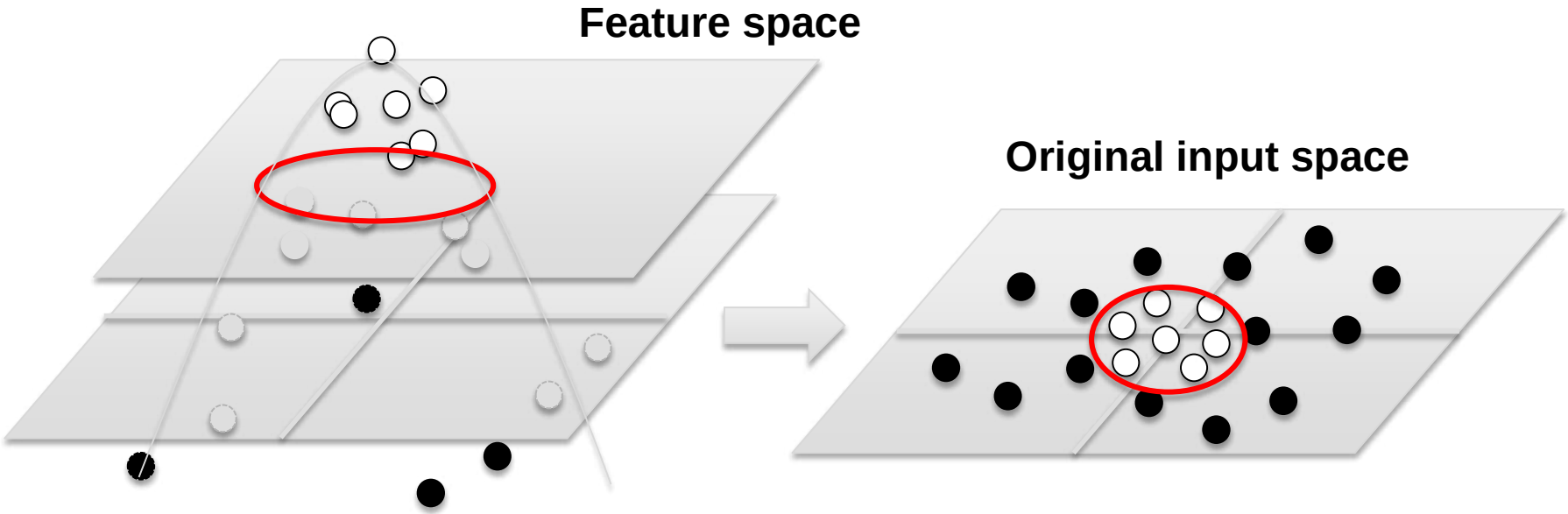
Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable



$$\boldsymbol{x_i} = (x_0, x_1)$$

$$\boldsymbol{v_i} = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

Example of mapping a 2D classification problem to a 3D feature space to make it linearly separable

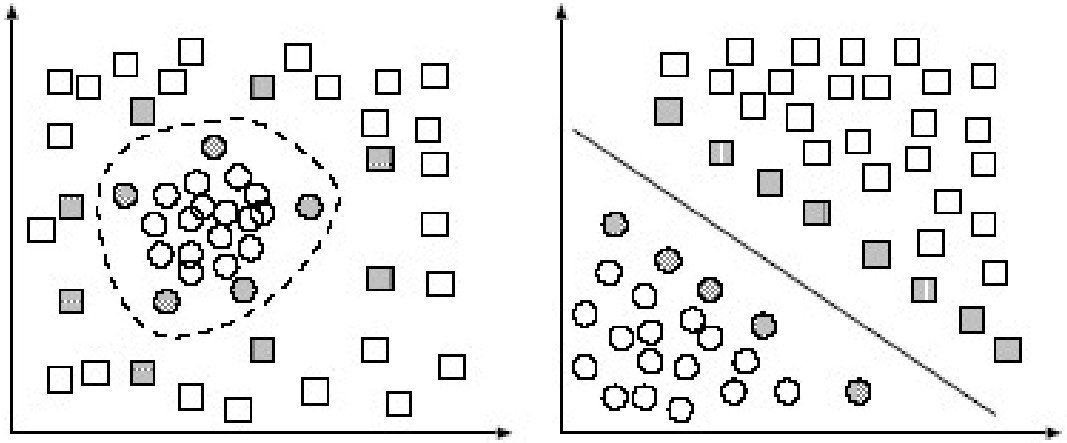


$$\boldsymbol{v_i} = (x_0, x_1, 1 - (x_0^2 + x_1^2))$$

$$\boldsymbol{x_i} = (x_0, x_1)$$

Radial Basis Function Kernel

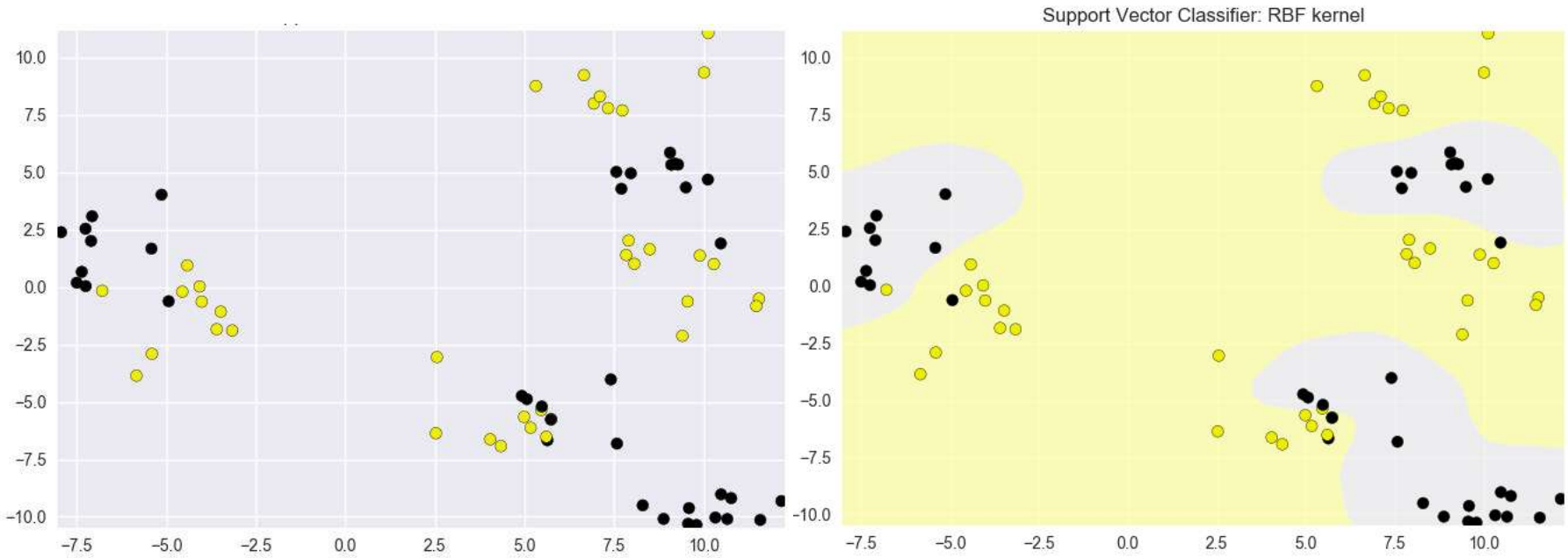
$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\ [-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2]$$



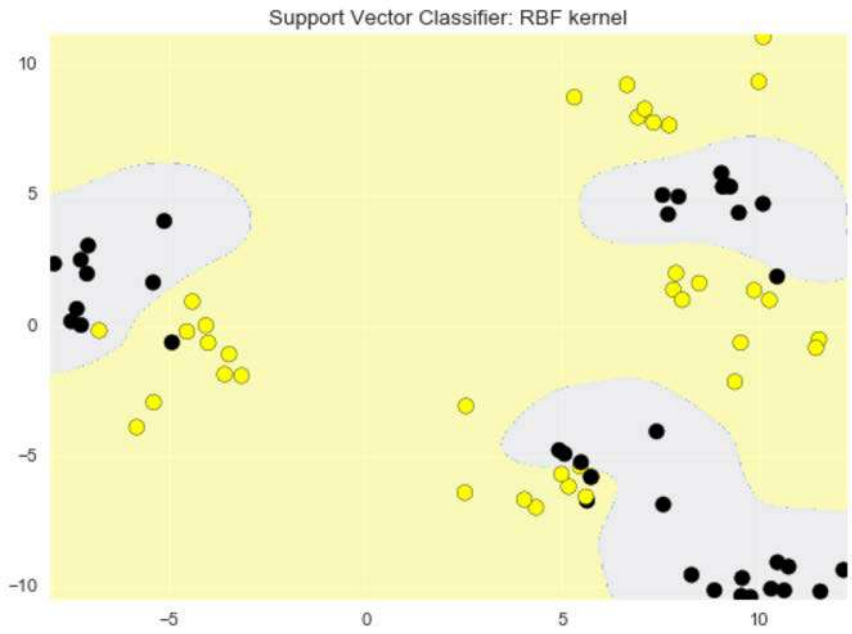
Original input space **Feature space**

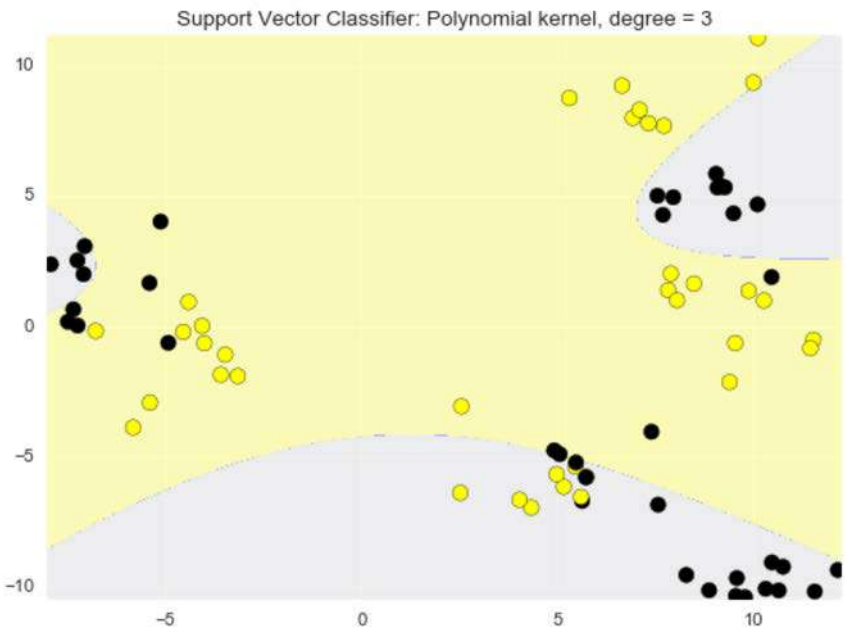
A kernel is a similarity measure (modified dot product) between data points

Applying the SVM with RBF kernel



Radial Basis Kernel vs Polynomial Kernel



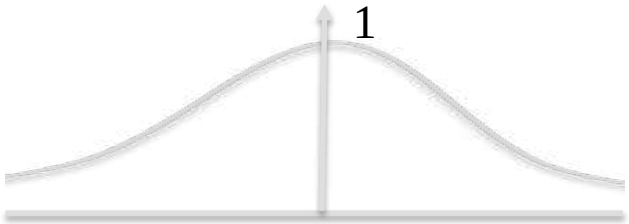


Radial Basis Function kernel: Gamma Parameter

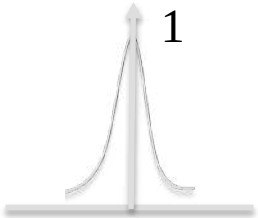
$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\ [-\gamma \cdot \|\boldsymbol{x} - \boldsymbol{x}'\|^2]$$

gamma (γ): kernel width parameter

small gamma (0.01)



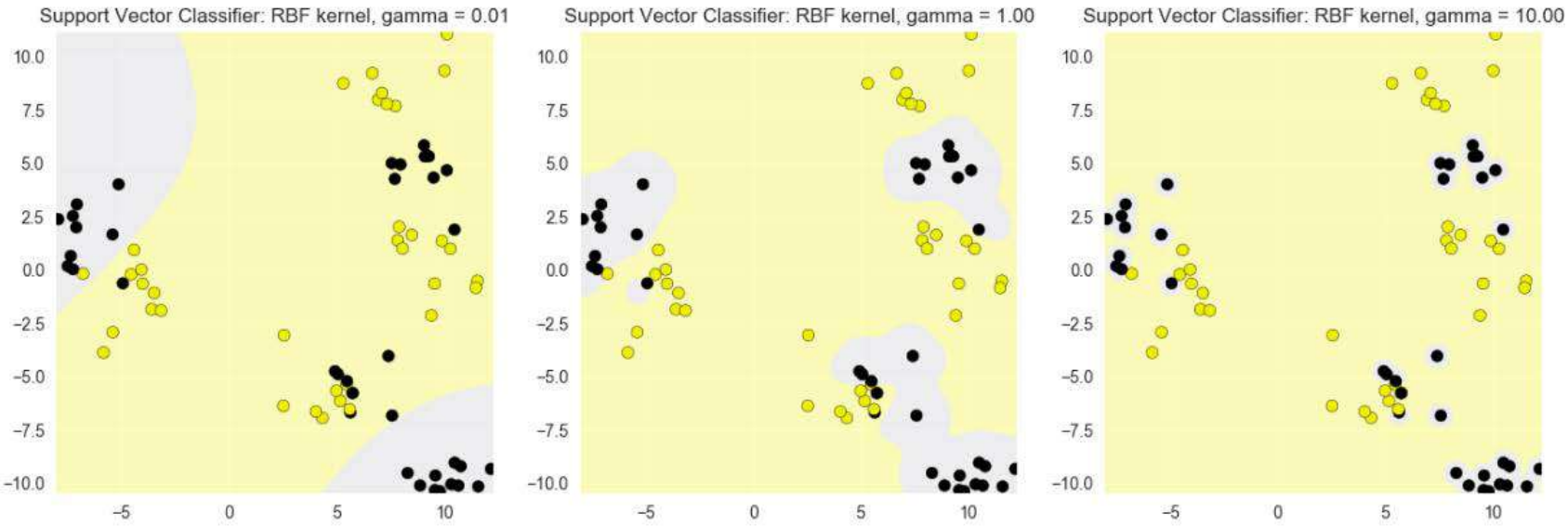
large gamma (10)



$\|\boldsymbol{x} - \boldsymbol{x}'\|^2$

Squared distance between x and x'

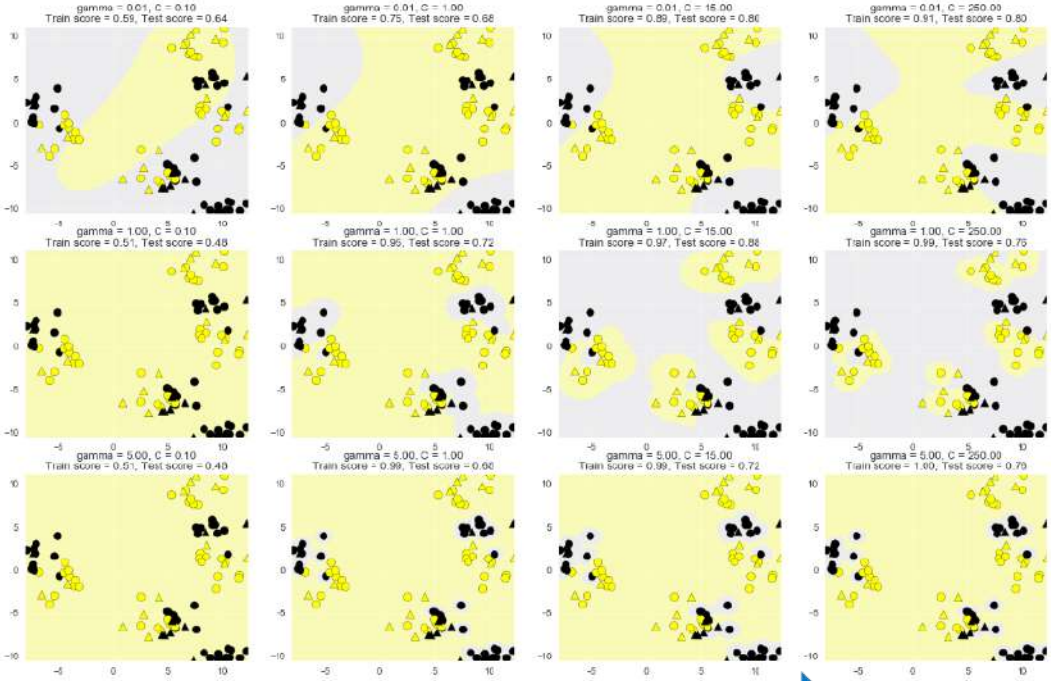
The effect of the RBF gamma parameter on decision boundaries



gamma = 0.01

gamma = 1.0

gamma = 10



Increasing gamma

Increasing C

Reminder: Using a scaler object: fit and transform methods

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
clf = SVC().fit(X_train_scaled, y_train)
accuracy = clf.score(X_test_scaled, y_test)
```

Tip: It can be more efficient to do fitting and transforming together on the training set using the `fit_transform` method.

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

Kernelized Support Vector Machines: pros and cons

Pros:

- Can perform well on a range of datasets.
- Versatile: different kernel functions can be specified, or custom kernels can be defined for specific data types.
- Works well for both low- and high-dimensional data.

Cons:

- Efficiency (runtime speed and memory usage) decreases as training set size increases (e.g. over 50000 samples).
- Needs careful normalization of input data and parameter tuning.
- Does not provide direct probability estimates (but can be estimated using e.g. Platt scaling).
- Difficult to interpret why a prediction was made.

Kernelized Support Vector Machines (SVC): Important parameters

Model complexity

- **`kernel`: Type of kernel function to be used**
  - *Default = 'rbf' for radial basis function*
  - *Other types include 'polynomial'*
- **kernel parameters**
  - `gamma` *(γ): RBF kernel width*
- **`C`: regularization parameter**
- **Typically `C` and `gamma` are tuned at the same time.**

Applied Machine Learning

Cross-validation

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

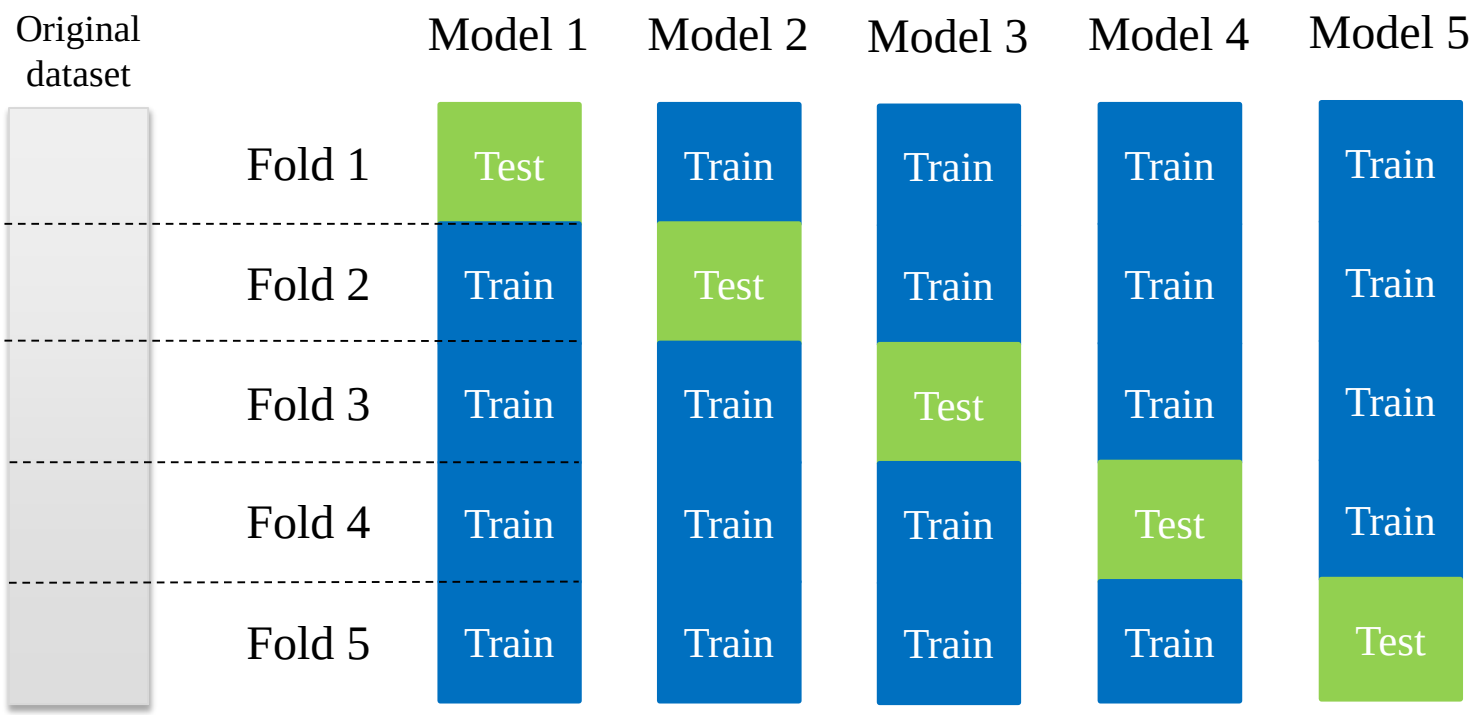
Cross-validation

- **Uses multiple train-test splits, not just a single one**
- **Each split used to train & evaluate a separate model**
- **Why is this better?**
  - *The accuracy score of a supervised learning method can vary, depending on which samples happen to end up in the training set.*
  - *Using multiple train-test splits gives more stable and reliable estimates for how the classifier is likely to perform on average.*
  - *Results are averaged over multiple different training sets instead of relying on a single model trained on a particular training set.*

random_state	Test set accuracy
0	1.00
1	0.93
5	0.93
7	0.67
10	0.87

Accuracy of k-NN classifier (k=5) on fruit data test set for different random_state values in train_test_split.

Cross-validation Example (5-fold)



Stratified Cross-validation

(Folds and dataset shortened for illustration purposes.)

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Example has 20 data samples
= 4 classes with 5 samples each.

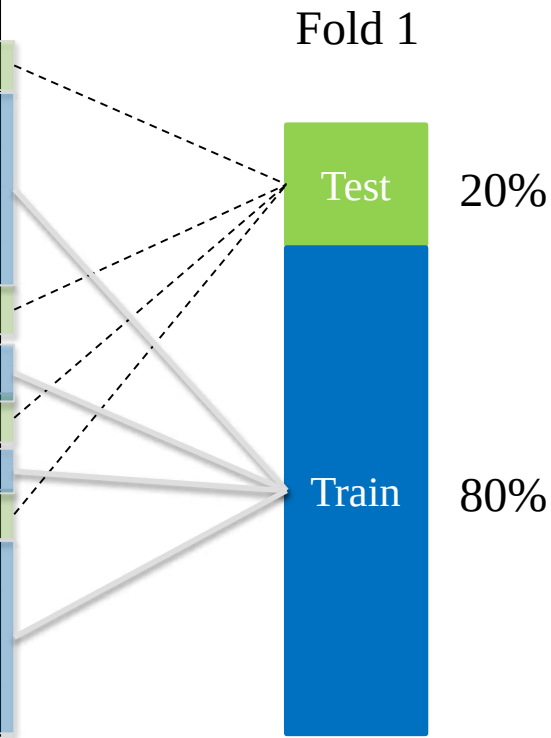
5-fold CV: 5 folds of 4 samples each.

Fold 1 uses the first 20% of the dataset as the test set, which only contains samples from class 1.

Classes 2, 3, 4 are missing entirely from test set and so will be missing from the evaluation.

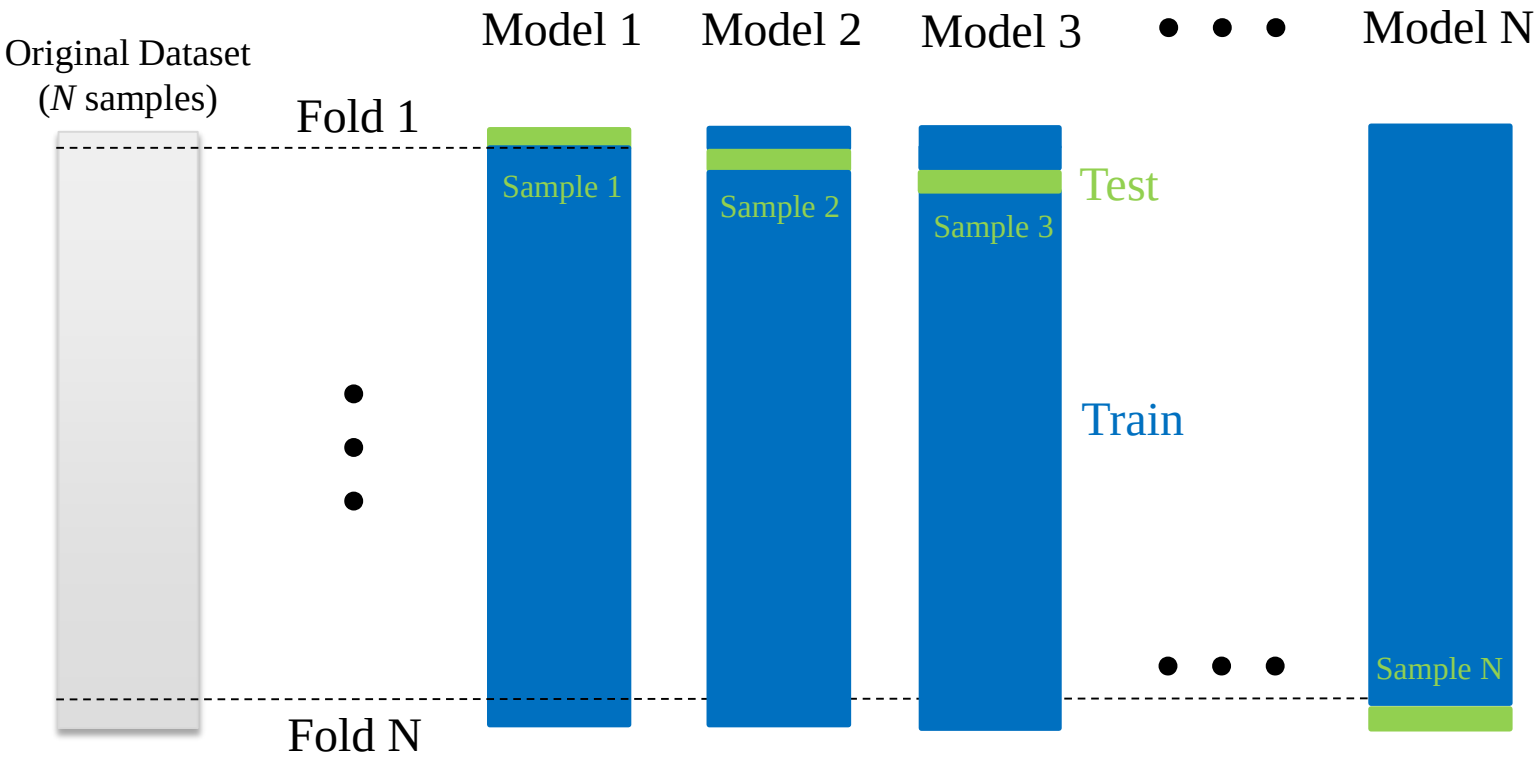
Stratified Cross-validation

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon



Stratified folds each contain a proportion of classes that matches the overall dataset. Now, all classes will be fairly represented in the test set.

Leave-one-out cross-validation (with *N* samples in dataset)



Validation curves show sensitivity to changes in an important parameter

```
from sklearn.svm import SVC
from sklearn.model_selection import validation_curve

param_range = np.logspace(-3, 3, 4)
train_scores, test_scores
   = validation_curve(SVC(), X, y, param_name="gamma",
       param_range=param_range, cv=5)
```

```
: print(train_scores)
```

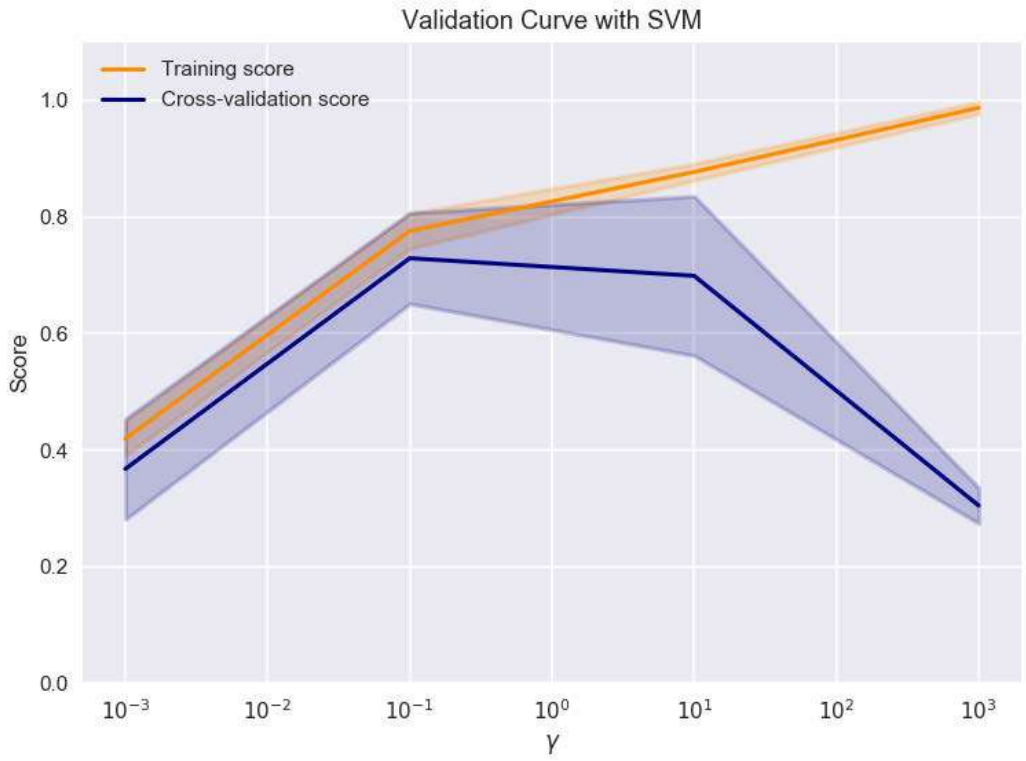
```
[[ 0.48648649  0.425       0.41463415]
 [ 0.83783784  0.725       0.75609756]
 [ 0.91891892  0.9         0.92682927]
 [ 1.          1.          0.97560976]]
```

```
: print(test_scores)
```

```
[[ 0.45454545  0.31578947  0.33333333]
 [ 0.81818182  0.68421053  0.61111111]
 [ 0.40909091  0.84210526  0.66666667]
 [ 0.36363636  0.21052632  0.38888889]]
```

One row per parameter sweep value,
One column per CV fold.

Validation Curve Example



The validation curve shows the mean cross-validation accuracy (solid lines) for training (orange) and test (blue) sets as a function of the SVM parameter (gamma). It also shows the variation around the mean (shaded region) as computed from k-fold cross-validation scores.

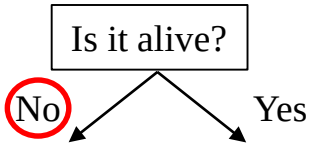
Applied Machine Learning

Decision Trees

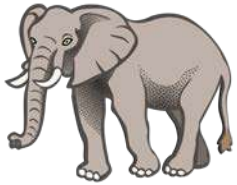
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

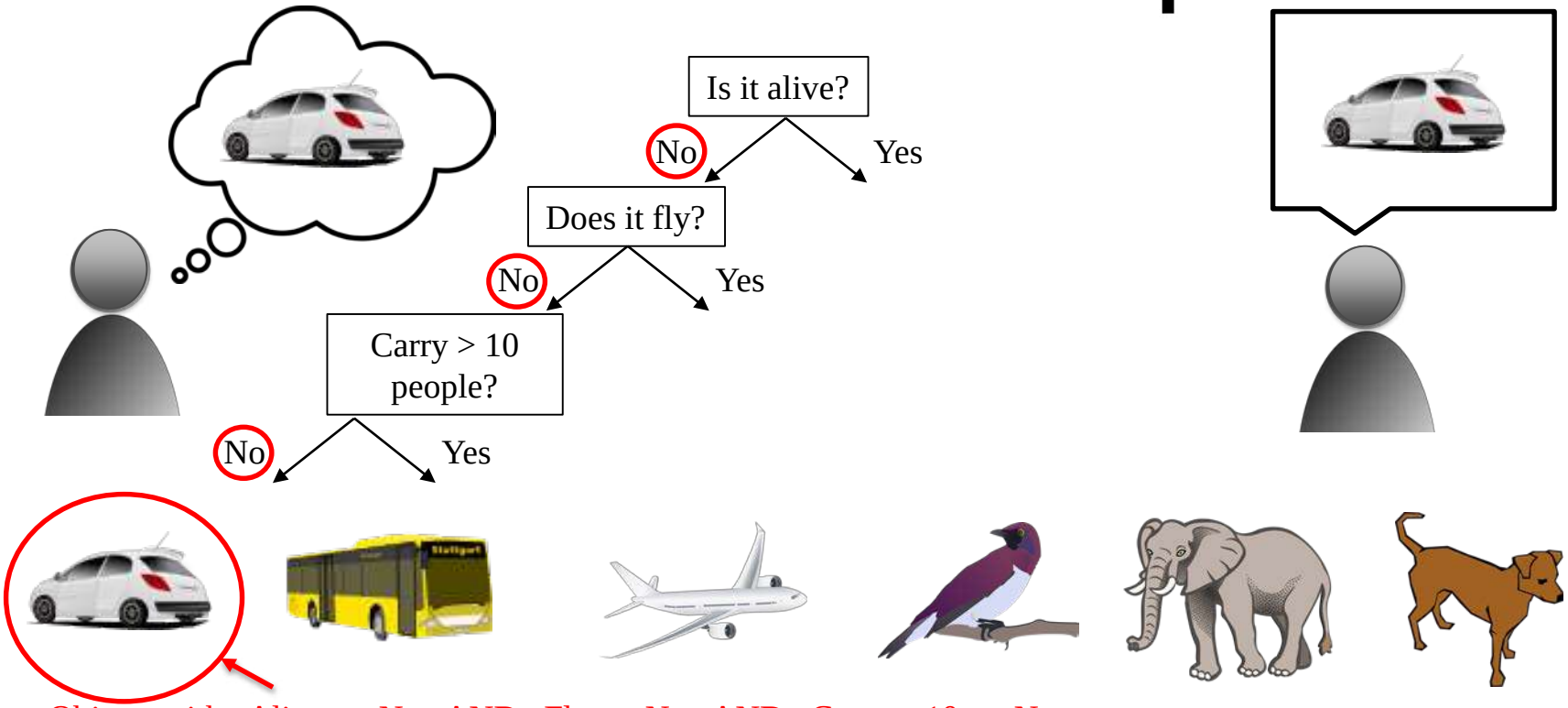
Decision Tree Example



Objects with Alive == No



Decision Tree Example



Objects with: Alive == No <u>AND</u> Fly == No <u>AND</u> Carry > 10 == No

Decision Tree Example

Root node

Is it alive?

No

Yes

Does it fly?

No

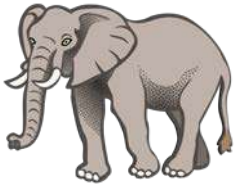
Yes

Does it have a trunk?

No

Yes

Alive == Yes
Fly == No
Trunk == No



Alive == Yes
Fly == No
Trunk == Yes

Leaf node

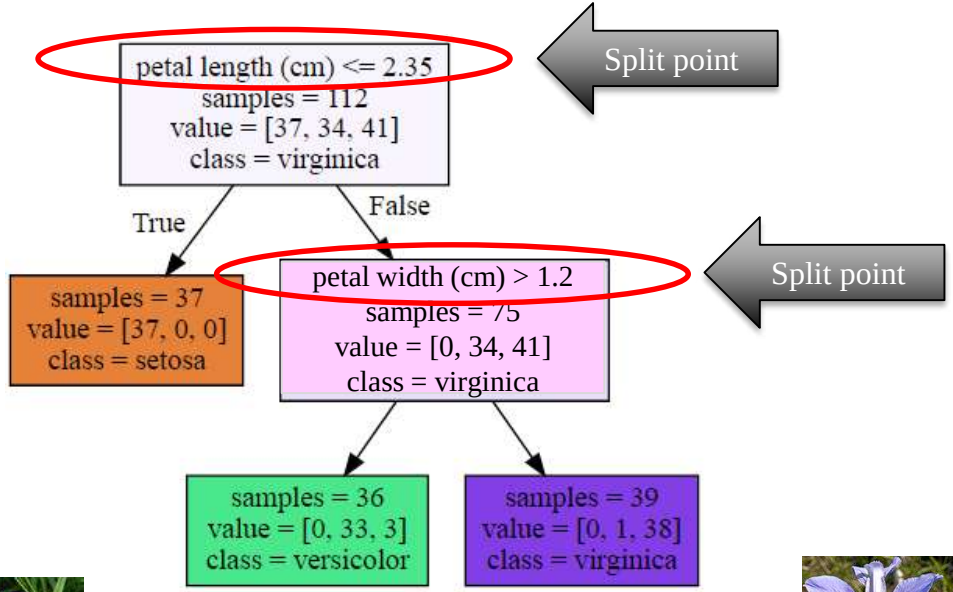
The Iris Dataset

Iris setosa *Iris versicolor* *Iris virginica*

150 flowers
3 species
50 examples/species

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

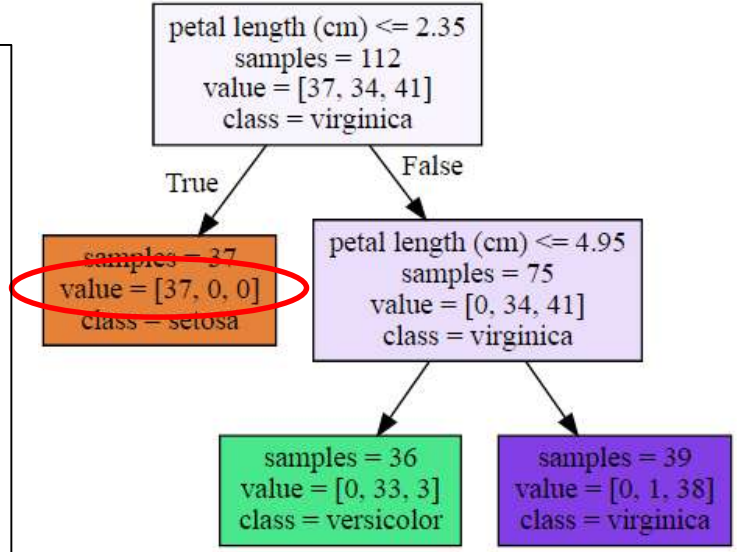
virginica

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.

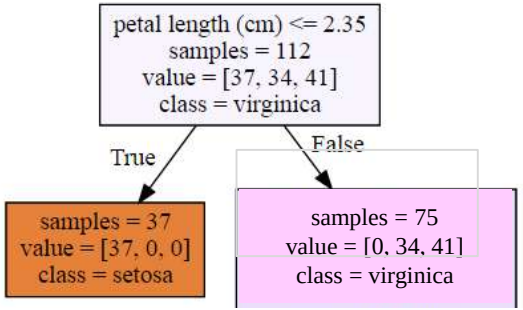
This leaf has 37 setosa samples, zero versicolor, and zero virginica samples.



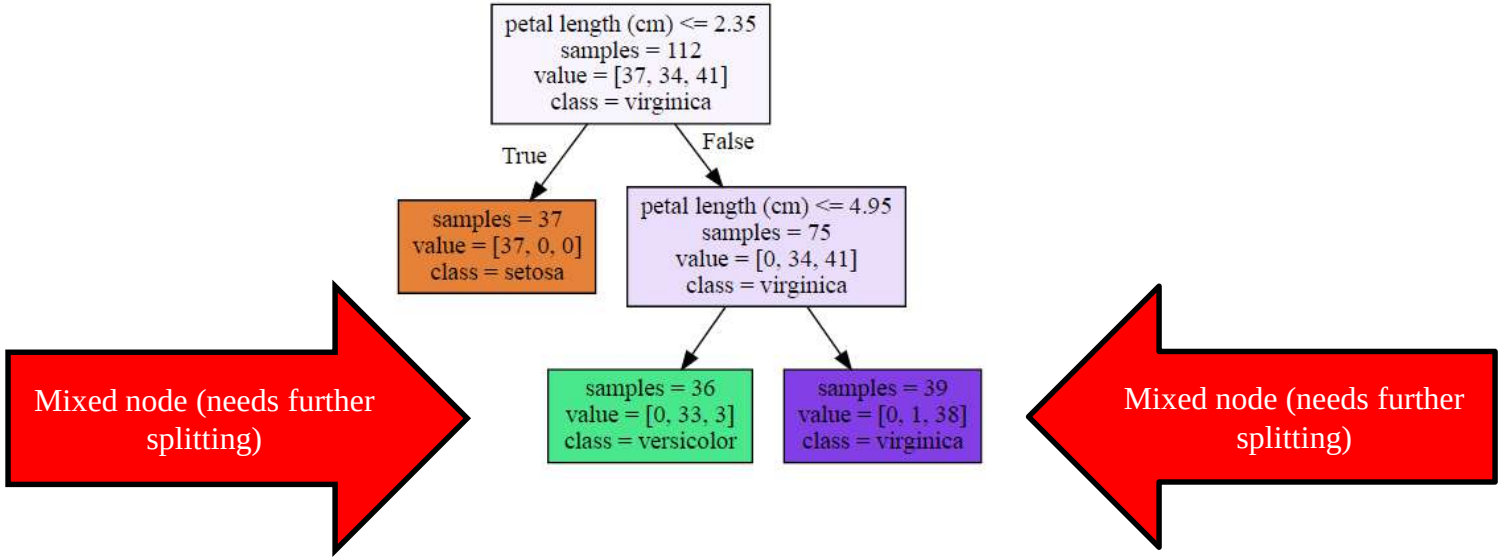
This leaf has 0 setosa, 33 versicolor, and 3 virginica samples.

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

Pure node (all one class: perfect classification)

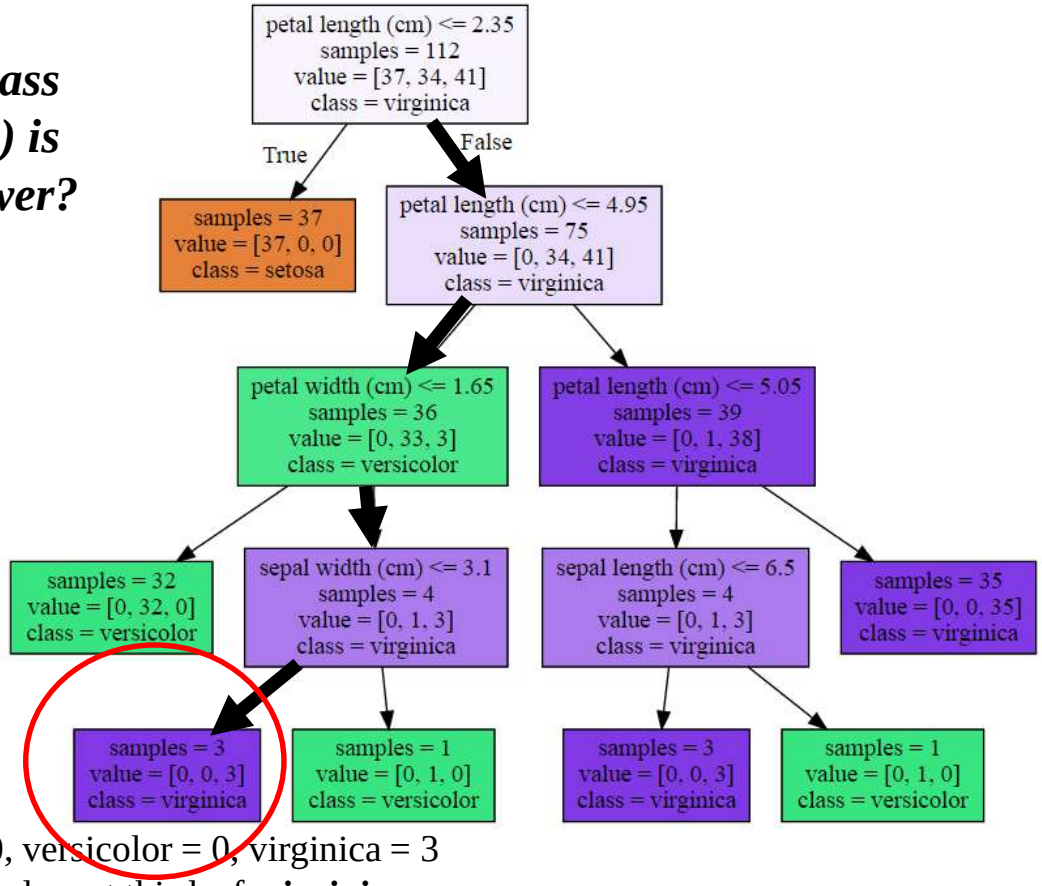


Mixed node (mixture of classes, still needs further splitting)


petal length (cm) <= 2.35
samples = 112
value = [37, 34, 41]
class = virginica
True
False
samples = 37
value = [37, 0, 0]
class = setosa
petal length (cm) <= 4.95
samples = 75
value = [0, 34, 41]
class = virginica
samples = 36
value = [0, 33, 3]
class = versicolor
samples = 39
value = [0, 1, 38]
class = virginica
Mixed node (needs further splitting)
Mixed node (needs further splitting)

What class (species) is this flower?

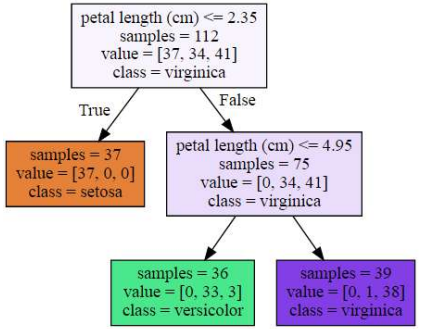
petal length: 3.0
petal width: 2.0
sepal width: 2.0
sepal length: 4.2



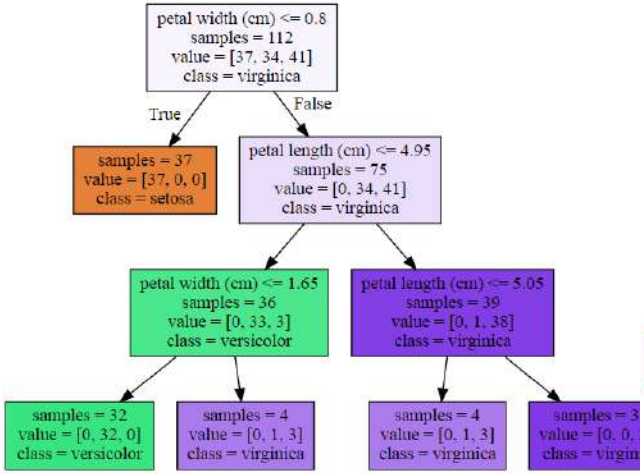
Leaf counts are: setosa = 0, versicolor = 0, virginica = 3
Predicted class is majority class at this leaf: **virginica**

Controlling the Model Complexity of Decision Trees

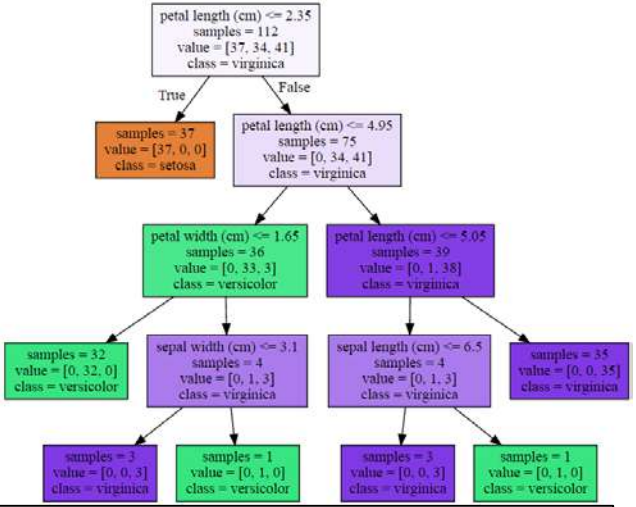
`max_depth = 2`



`max_depth = 3`

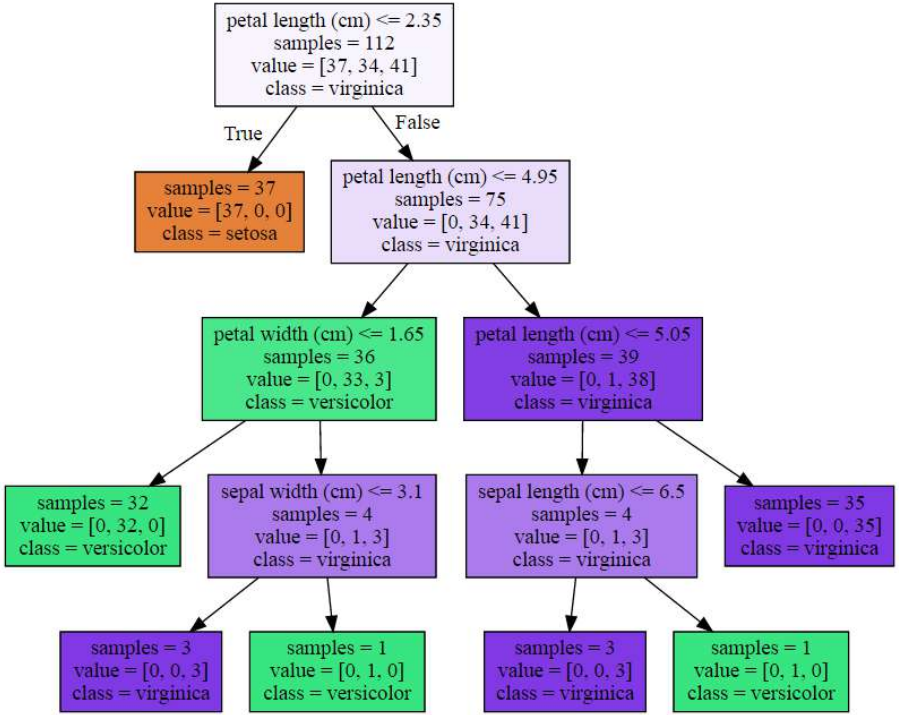


`max_depth = 4`



<u>Other parameters</u>: Max. # of leaf nodes: `max_leaf_nodes` Min. samples to consider splitting: `min_samples_leaf`

Visualizing Decision Trees

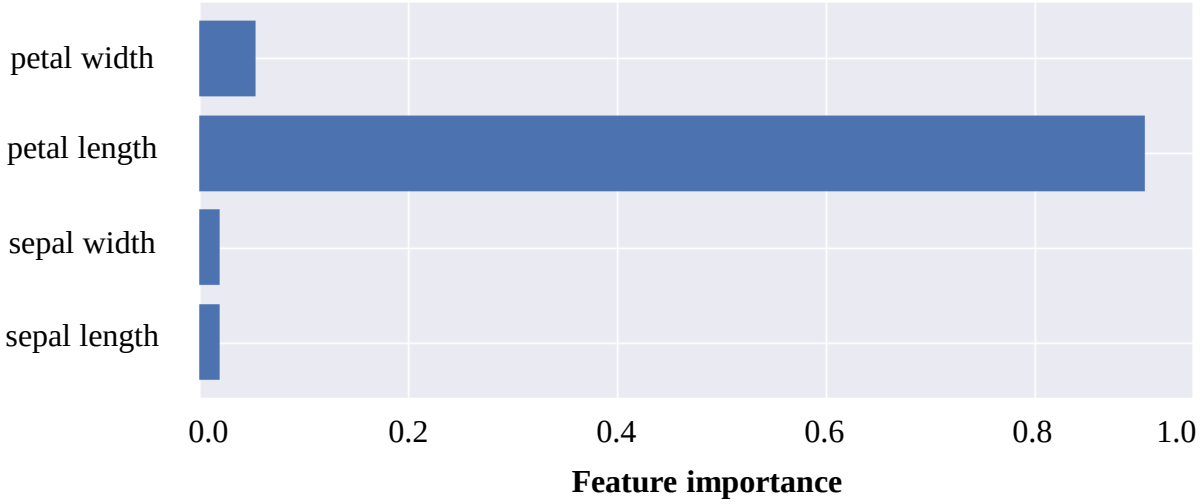


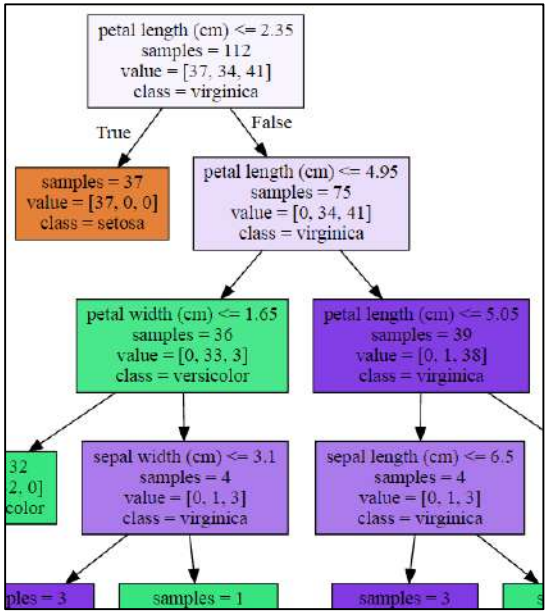
See: `plot_decision_tree()` function in `adspy_shared_utilities.py` code

Feature Importance: How important is a feature to overall prediction accuracy?

- A number between 0 and 1 assigned to each feature.
- Feature importance of 0 → the feature was not used in prediction.
- Feature importance of 1 → the feature predicts the target perfectly.
- All feature importances are normalized to sum to 1.

Feature Importance Chart





Decision tree

See: `plot_feature_importances()` 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.**

Decision Trees: DecisionTreeClassifier Key Parameters

- `max_depth`: controls maximum depth (number of split points). Most common way to reduce tree complexity and overfitting.
- `min_samples_leaf`: threshold for the minimum # of data instances a leaf can have to avoid further splitting.
- `max_leaf_nodes`: limits total number of leaves in the tree.
- In practice, adjusting only one of these (e.g. `max_depth`) is enough to reduce overfitting.

Module 2

September 22, 2021

---

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Applied Machine Learning: Module 2 (Supervised Learning, Part I)

1.1 Preamble and Review

1.1.1 we must apply the scaling to the test set that we computed for the training set

1.1.2 X_train me scaler.fit_transform kiya hai, X_test me scaler.transform hi hai. aage sab kaam ye scaled data se hi kiya hai

In [3]:
```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

np.set_printoptions(precision=2)


fruits = pd.read_table('readonly/fruit_data_with_colors.txt')

feature_names_fruits = ['height', 'width', 'mass', 'color_score']
X_fruits = fruits[feature_names_fruits]  #direct one steo me bhi kar sakte
y_fruits = fruits['fruit_label']
target_names_fruits = ['apple', 'mandarin', 'orange', 'lemon']

X_fruits_2d = fruits[['height', 'width']]
y_fruits_2d = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X_fruits, y_fruits, ran
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
# we must apply the scaling to the test set that we computed for the traini
X_test_scaled = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train_scaled, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(X_train_scaled, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test_scaled, y_test)))

example_fruit = [[5.5, 2.2, 10, 0.70]]
example_fruit_scaled = scaler.transform(example_fruit)
print('Predicted fruit type for ', example_fruit, ' is ',
          target_names_fruits[knn.predict(example_fruit_scaled)[0]-1]) #thi
print(target_names_fruits[knn.predict(example_fruit_scaled)[0]])
```

```
Accuracy of K-NN classifier on training set: 0.95
Accuracy of K-NN classifier on test set: 1.00
Predicted fruit type for  [[5.5, 2.2, 10, 0.7]]  is  mandarin
orange
```

1.2 Datasets

In [4]:
```
from sklearn.datasets import make_classification, make_blobs
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_breast_cancer
from adspy_shared_utilities import load_crime_dataset

cmap_bold = ListedColormap(['#FFFF00', '#00FF00', '#0000FF','#000000'])


# synthetic dataset for simple regression
from sklearn.datasets import make_regression #MAKE REGRESSION GIVES SIMPL C
plt.figure()
plt.title('Sample regression problem with one input variable') #OKAY ONLY,
X_R1, y_R1 = make_regression(n_samples = 100, n_features=1,
                             n_informative=1, bias = 150.0,
                             noise = 30, random_state=0)
print('X_R1 is \n ',  X_R1)
print('y_R1 is \n ',  y_R1)
plt.scatter(X_R1, y_R1, marker= 'o', s=50)
plt.show()
```

```
# synthetic dataset for more complex regression
from sklearn.datasets import make_friedman1 #MAKE FRIEDMAN1 GIVES COMPLEX C
plt.figure()
plt.title('Complex regression problem with one input variable')
X_F1, y_F1 = make_friedman1(n_samples = 100,
                            n_features = 7, random_state=0) #7 FEATURES LIYA

print('X_F1 is \n ',  X_F1)
print('X_F1[:,2] is \n',  X_F1[:,2])
print('y_F1 is \n ',  y_F1)
plt.scatter(X_F1[:, 2], y_F1, marker= 'o', s=50) #ONLY 1 FEATURE LIYA ABHI
plt.show()

# synthetic dataset for classification (binary)
plt.figure()
plt.title('Sample binary classification problem with two informative featur
X_C2, y_C2 = make_classification(n_samples = 100, n_features=2,
                                 n_redundant=0, n_informative=2,
                                 n_clusters_per_class=1, flip_y = 0.1,
                                 class_sep = 0.5, random_state=0) # 2 FEATUR
print('X_C2 is \n ',  X_C2)
print('X_C2[:,0] is \n',  X_C2[:,0])
print('X_C2[:,1] is \n',  X_C2[:,1])
print('y_C2 is \n ',  y_C2) #binary classication
plt.scatter(X_C2[:, 0], X_C2[:, 1], c=y_C2,
            marker= 'o', s=50, cmap=cmap_bold) #PLOT BHI 2 FEATURE WISE KIYA
plt.show()


# more difficult synthetic dataset for classification (binary)
# with classes that are not linearly separable
X_D2, y_D2 = make_blobs(n_samples = 100, n_features = 2, centers = 8,
                        cluster_std = 1.3, random_state = 4) #MAKE BLOBS MAK

print('X_D2 is \n ',  X_D2)
print('X_D2[:,0] is \n',  X_D2[:,0])
print('X_D2[:,1] is \n',  X_D2[:,1])
print('y_D2 is \n ',  y_D2)

y_D2 = y_D2 % 2
print('new y_D2 is \n ',  y_D2) #matlab labels diye hai even ko 0 baki ko 1
plt.figure()
plt.title('Sample binary classification problem with non-linearly separable
plt.scatter(X_D2[:,0], X_D2[:,1], c=y_D2,
            marker= 'o', s=50, cmap=cmap_bold) #PLOT BHI 2 FEATURE WISE KIYA
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
X_R1 is
  [[-0.36]
 [ 0.98]
 [ 0.4 ]
 [-0.81]
 [-0.89]
 [ 0.44]
 [-0.98]
 [ 0.43]
 [ 0.21]
 [-0.31]
 [-0.51]
 [ 0.13]
 [-1.54]
 [-0.4 ]
 [ 0.65]
 [ 1.18]
 [-0.18]
 [ 1.79]
 [ 1.45]
 [-0.68]
 [ 0.98]
 [ 1.9 ]
 [-0.44]
 [ 0.31]
 [ 0.76]
 [ 0.78]
 [ 1.95]
 [ 0.33]
 [-0.35]
 [ 1.53]
 [-0.9 ]
 [-0.58]
 [-1.05]
 [ 0.38]
 [ 0.01]
 [ 0.46]
 [ 0.14]
 [-0.4 ]
 [ 0.05]
 [-1.25]
 [ 1.05]
```

```
[ 0.4 ]
[-1.71]
[ 2.24]
[ 0.18]
[-0.21]
[ 1.23]
[-0.85]
[ 1.76]
[ 0.41]
[-0.91]
[-1.42]
[ 0.47]
[ 0.9 ]
[ 1.14]
[ 0.4 ]
[ 0.39]
[-1.73]
[ 1.47]
[-0.19]
[-0.87]
[ 0.3 ]
[ 1.2 ]
[-1.98]
[-2.55]
[-1.45]
[-0.03]
[ 0.15]
[ 1.22]
[ 1.87]
[ 2.27]
[-0.74]
[ 0.07]
[ 1.49]
[-0.67]
[-1.63]
[ 0.13]
[ 0.95]
[ 1.49]
[ 0.06]
[ 0.86]
[ 0.71]
[-0.15]
[-1.07]
[-0.1 ]
[-0.3 ]
[-0.39]
[ 0.73]
[ 0.12]
```

```
 [-0.51]
 [-1.23]
 [-0.21]
 [-0.63]
 [ 0.16]
 [ 0.05]
 [-1.17]
 [-0.36]
 [ 0.36]
 [-1.61]
 [-1.18]]
y_R1 is
  [ 120.61  131.23  150.56  169.91  118.16  196.36   63.87  166.52  196.7
  109.35  110.9   167.15   50.76   99.06  186.67  231.4   149.87  220.19
  267.55  136.61  222.41  221.7   187.93  165.96  221.    251.48  245.1
  159.38  187.46  225.78   85.72  133.32   85.72  209.37   83.89  203.09
  201.83  145.01   99.18  118.33  163.73  168.56   83.61  206.67  177.06
   81.58  242.47  117.24  217.54  115.11  151.73   54.41  112.72  143.6
  186.77  174.29  117.77    8.01  224.52  150.69  116.34  143.89  176.42
   53.42   24.22   89.85  102.12   98.35  181.29  254.29  244.83  131.6
   98.48  184.41  127.36   57.58  188.15  158.16  160.22  178.13  193.31
  171.07  137.31  106.02  163.76   98.45  136.06  193.64  194.18  164.32
  116.63  159.32   95.27  145.7   123.87  103.44  179.35  155.14  128.52
   83.82]


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


X_F1 is
  [[ 0.55  0.72  0.6   0.54  0.42  0.65  0.44]
 [ 0.89  0.96  0.38  0.79  0.53  0.57  0.93]
 [ 0.07  0.09  0.02  0.83  0.78  0.87  0.98]
 [ 0.8   0.46  0.78  0.12  0.64  0.14  0.94]
 [ 0.52  0.41  0.26  0.77  0.46  0.57  0.02]
 [ 0.62  0.61  0.62  0.94  0.68  0.36  0.44]
 [ 0.7   0.06  0.67  0.67  0.21  0.13  0.32]
 [ 0.36  0.57  0.44  0.99  0.1   0.21  0.16]
 [ 0.65  0.25  0.47  0.24  0.16  0.11  0.66]
 [ 0.14  0.2   0.37  0.82  0.1   0.84  0.1 ]
 [ 0.98  0.47  0.98  0.6   0.74  0.04  0.28]
 [ 0.12  0.3   0.12  0.32  0.41  0.06  0.69]
 [ 0.57  0.27  0.52  0.09  0.58  0.93  0.32]
 [ 0.67  0.13  0.72  0.29  0.18  0.59  0.02]
 [ 0.83  0.    0.68  0.27  0.74  0.96  0.25]
```

```
[ 0.58  0.59  0.57  0.22  0.95  0.45  0.85]
[ 0.7   0.3   0.81  0.4   0.88  0.58  0.88]
[ 0.69  0.73  0.5   0.96  0.64  0.42  0.61]
[ 0.02  0.3   0.66  0.29  0.62  0.43  0.14]
[ 0.3   0.57  0.59  0.57  0.65  0.65  0.43]
[ 0.9   0.37  0.44  0.89  0.81  0.7   0.1 ]
[ 0.92  0.71  1.    0.15  0.87  0.16  0.62]
[ 0.12  0.85  0.81  0.57  0.41  0.07  0.7 ]
[ 0.45  0.72  0.87  0.98  0.86  0.01  0.36]
[ 0.73  0.17  0.52  0.05  0.2   0.02  0.79]
[ 0.22  0.35  0.93  0.7   0.03  0.16  0.62]
[ 0.58  0.24  0.93  0.61  0.54  0.59  0.73]
[ 0.31  0.4   0.21  0.19  0.94  0.74  0.49]
[ 0.23  0.25  0.06  0.43  0.31  0.7   0.38]
[ 0.18  0.02  0.07  0.68  0.45  0.54  0.9 ]
[ 0.99  0.22  0.66  0.26  0.02  0.76  0.32]
[ 0.38  0.59  0.83  0.63  0.87  0.27  0.8 ]
[ 0.19  0.95  0.69  0.22  0.95  0.73  0.25]
[ 0.21  0.52  0.03  0.21  0.42  0.37  0.46]
[ 0.28  0.59  0.86  0.12  0.52  0.13  0.72]
[ 0.4   0.57  0.18  0.14  0.49  0.36  0.94]
[ 0.77  0.75  0.9   0.08  0.55  0.58  0.96]
[ 0.29  0.24  0.1   0.02  0.93  0.67  0.79]
[ 0.28  0.59  0.06  0.49  0.98  0.88  0.34]
[ 0.96  0.23  0.95  0.94  0.8   0.63  0.87]
[ 0.29  0.85  0.62  0.01  0.35  0.15  0.98]
[ 0.48  0.5   0.64  0.37  0.14  0.82  0.19]
[ 0.51  0.22  0.1   0.86  0.97  0.96  0.91]
[ 0.77  0.33  0.08  0.41  0.23  0.13  0.05]
[ 0.73  0.01  0.77  0.15  0.08  0.09  0.67]
[ 0.25  0.42  0.56  0.86  0.73  0.27  0.13]
[ 0.06  0.3   0.26  0.46  0.68  0.7   0.28]
[ 0.38  0.18  0.79  0.06  0.7   0.78  0.78]
[ 0.26  0.37  0.59  0.27  0.37  0.2   0.46]
[ 0.04  0.8   0.08  0.52  0.31  0.58  0.96]
[ 0.65  0.04  0.43  0.51  0.54  0.68  0.28]
[ 0.13  0.39  0.96  0.19  0.9   0.54  0.46]
[ 0.88  0.46  0.72  0.4   0.9   0.69  0.7 ]
[ 0.33  0.76  0.64  0.24  0.16  0.8   0.96]
[ 0.46  0.59  0.86  0.46  0.95  0.58  0.82]
[ 0.91  0.82  0.16  0.63  0.4   0.06  0.42]
[ 0.26  0.85  0.03  0.96  0.36  0.36  0.02]
[ 0.19  0.4   0.93  0.1   0.95  0.87  0.45]
[ 0.33  0.23  0.61  0.03  0.02  0.43  0.07]
[ 0.25  0.22  0.25  0.13  0.01  0.12  0.62]
[ 0.97  0.99  0.41  0.16  0.64  0.49  0.99]
[ 0.07  0.78  0.29  0.24  0.66  0.25  0.67]
[ 0.52  0.42  0.55  0.29  0.71  0.41  0.36]
```

```
 [ 0.83  0.92  0.05  0.23  0.35  0.81  0.99]
 [ 0.97  0.9   0.3   0.99  0.25  0.11  0.95]
 [ 0.23  0.69  0.06  0.73  0.88  0.27  0.38]
 [ 0.37  0.75  0.24  0.17  0.45  0.3   0.84]
 [ 0.24  0.5   0.94  0.63  0.87  0.94  0.75]
 [ 0.7   0.97  0.99  0.45  0.07  0.29  0.15]
 [ 0.42  0.13  0.6   0.38  0.9   0.97  0.55]
 [ 0.27  0.59  0.9   0.41  0.55  0.27  0.46]
 [ 0.4   0.25  0.51  0.31  0.37  0.52  0.75]
 [ 0.33  0.92  0.86  0.05  0.25  0.45  0.1 ]
 [ 0.35  0.74  0.68  0.62  0.71  0.2   0.34]
 [ 0.68  0.88  0.54  0.28  0.03  0.71  0.01]
 [ 0.37  0.53  0.92  0.09  0.41  0.02  0.34]
 [ 0.62  0.28  0.21  0.12  0.58  0.7   0.67]
 [ 0.95  0.    0.65  0.6   0.59  0.96  0.02]
 [ 0.7   0.81  0.51  0.33  0.79  0.1   0.44]
 [ 0.52  0.69  0.09  0.23  0.41  0.62  0.89]
 [ 0.62  0.13  0.98  0.87  0.5   0.92  0.54]
 [ 0.92  0.83  0.97  0.92  0.04  0.17  0.39]
 [ 0.95  0.3   0.16  0.89  0.45  0.91  0.16]
 [ 0.66  0.44  0.08  0.7   0.25  0.04  0.06]
 [ 0.06  0.91  0.74  0.9   0.67  0.53  0.3 ]
 [ 1.    0.36  0.47  0.38  0.98  0.17  0.33]
 [ 0.68  0.06  0.61  0.48  0.28  0.24  0.51]
 [ 0.37  0.46  0.34  0.97  0.13  0.1   0.34]
 [ 0.59  0.66  0.4   1.    0.35  0.72  0.64]
 [ 0.81  0.98  0.89  0.76  0.7   0.34  0.15]
 [ 0.06  0.24  0.43  0.52  0.77  0.96  0.12]
 [ 0.11  0.59  0.75  0.85  0.94  0.98  0.4 ]
 [ 0.38  0.15  0.68  0.66  0.86  0.1   0.5 ]
 [ 0.58  0.24  0.17  0.86  0.06  0.47  0.12]
 [ 0.46  0.98  0.42  0.86  0.12  0.27  0.4 ]
 [ 0.4   0.67  0.34  0.71  0.64  0.4   0.43]
 [ 0.61  0.07  0.82  0.65  0.73  0.54  0.11]
 [ 0.41  0.41  0.32  0.03  0.74  0.11  0.61]
 [ 0.7   0.63  0.96  0.1   0.87  0.03  0.53]
 [ 0.4   0.52  0.37  0.19  0.02  0.52  0.84]]
X_F1[:,2] is
 [ 0.6   0.38  0.02  0.78  0.26  0.62  0.67  0.44  0.47  0.37  0.98  0.12
  0.52  0.72  0.68  0.57  0.81  0.5   0.66  0.59  0.44  1.    0.81  0.87
  0.52  0.93  0.93  0.21  0.06  0.07  0.66  0.83  0.69  0.03  0.86  0.18
  0.9   0.1   0.06  0.95  0.62  0.64  0.1   0.08  0.77  0.56  0.26  0.79
  0.59  0.08  0.43  0.96  0.72  0.64  0.86  0.16  0.03  0.93  0.61  0.25
  0.41  0.29  0.55  0.05  0.3   0.06  0.24  0.94  0.99  0.6   0.9   0.51
  0.86  0.68  0.54  0.92  0.21  0.65  0.51  0.09  0.98  0.97  0.16  0.08
  0.74  0.47  0.61  0.34  0.4   0.89  0.43  0.75  0.68  0.17  0.42  0.34
  0.82  0.32  0.96  0.37]
y_F1 is
```

```
  [ 17.21  15.11  17.02  15.12  17.42  22.4    9.63  16.53   8.23   9.89
  24.2    9.27   8.38   7.47   7.13  15.88  16.42  22.78   6.69  14.27
  21.63  19.62  12.86  25.29   5.39  13.27  16.77  12.07  11.62  12.95
   9.52  19.35  12.87  12.1   11.31  12.36  16.59  10.2   18.51  23.89
   9.19  11.56  20.25  15.99   3.59  15.49   9.63   7.86   7.74  11.42
   8.59  12.14  19.07  10.6   19.41  17.87  22.09  11.72   3.04   4.33
   6.09   8.22  12.82  14.89  15.77  20.46  13.05  18.26  18.25  10.24
  14.87   8.05  12.62  17.67  12.58  12.31  10.92   9.46  17.08  16.74
  18.42  20.46  21.22  19.71  15.23  17.76   7.77  15.94  21.37  20.21
   9.65  16.33  13.32  15.35  19.14  18.28  13.59   9.56  19.44   8.54]


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


X_C2 is
  [[ 0.37 -0.92]
 [-0.16  0.51]
 [-1.64  2.23]
 [ 0.62 -2.19]
 [-0.99  1.93]
 [-0.15  1.31]
 [ 1.62  0.31]
 [-2.01  1.99]
 [ 0.51 -0.13]
 [ 0.41 -0.7 ]
 [ 0.53  1.1 ]
 [ 0.6   1.28]
 [ 0.45  0.85]
 [-1.01  0.77]
 [ 0.48  1.85]
 [ 0.3  -0.79]
 [ 0.33 -0.52]
 [ 0.6  -0.15]
 [ 0.5   1.12]
 [ 0.61  0.43]
 [-1.28  1.03]
 [ 0.31  0.94]
 [ 0.38  1.45]
 [ 0.36 -0.97]
 [ 0.3   0.36]
 [ 0.65 -0.18]
 [-0.15  0.48]
 [ 0.45  1.33]
 [ 0.29  1.48]
```

```
[ 0.44 -0.96]
[-0.28  0.67]
[ 0.73  0.95]
[ 0.44  0.87]
[ 0.66  0.01]
[ 0.55 -1.64]
[-0.98 -0.47]
[-0.71  0.89]
[-1.13  1.98]
[ 0.67  0.22]
[ 0.35 -1.78]
[-1.61  0.59]
[-0.72 -0.09]
[-0.23 -1.02]
[ 1.24 -0.28]
[ 0.26  0.3 ]
[-1.19  0.84]
[ 0.84 -1.61]
[-0.54 -0.12]
[-2.32  3.19]
[ 0.6   0.42]
[ 0.22 -0.19]
[ 0.61 -0.56]
[ 0.45  2.72]
[ 0.8  -1.11]
[ 0.58 -0.57]
[ 0.23 -0.47]
[ 0.6   1.29]
[ 0.24 -0.68]
[-1.25  0.99]
[ 0.38  0.93]
[-0.85  2.81]
[ 0.53  1.54]
[-1.05  1.5 ]
[-2.21  1.03]
[ 0.42  0.68]
[-1.58  1.15]
[ 0.51 -0.79]
[-0.35  0.74]
[ 0.2  -0.16]
[ 0.23  0.77]
[ 0.35  0.37]
[ 0.67  0.69]
[ 0.43 -0.08]
[-0.27  0.54]
[-0.18 -0.59]
[ 0.31 -0.08]
[ 0.7  -0.3 ]
```

```
 [-0.57  0.39]
 [-1.25  1.4 ]
 [ 0.21  0.4 ]
 [ 0.25  0.21]
 [-0.58  0.19]
 [ 0.31  1.38]
 [ 0.37  0.19]
 [ 0.65  1.62]
 [ 0.38  0.77]
 [ 0.67 -1.01]
 [ 0.37  2.28]
 [-0.82  1.89]
 [ 0.79 -1.63]
 [-0.74  0.82]
 [ 1.86 -0.3 ]
 [-2.39  0.69]
 [-2.18  0.7 ]
 [ 0.24  0.63]
 [ 0.07  0.18]
 [ 0.27  0.83]
 [ 0.88 -1.36]
 [ 1.65 -0.48]
 [ 0.4   1.55]]
X_C2[:,0] is
 [ 0.37 -0.16 -1.64  0.62 -0.99 -0.15  1.62 -2.01  0.51  0.41  0.53  0.6
  0.45 -1.01  0.48  0.3   0.33  0.6   0.5   0.61 -1.28  0.31  0.38  0.36
  0.3   0.65 -0.15  0.45  0.29  0.44 -0.28  0.73  0.44  0.66  0.55 -0.98
 -0.71 -1.13  0.67  0.35 -1.61 -0.72 -0.23  1.24  0.26 -1.19  0.84 -0.54
 -2.32  0.6   0.22  0.61  0.45  0.8   0.58  0.23  0.6   0.24 -1.25  0.38
 -0.85  0.53 -1.05 -2.21  0.42 -1.58  0.51 -0.35  0.2   0.23  0.35  0.67
  0.43 -0.27 -0.18  0.31  0.7  -0.57 -1.25  0.21  0.25 -0.58  0.31  0.37
  0.65  0.38  0.67  0.37 -0.82  0.79 -0.74  1.86 -2.39 -2.18  0.24  0.07
  0.27  0.88  1.65  0.4 ]
X_C2[:,1] is
 [-0.92  0.51  2.23 -2.19  1.93  1.31  0.31  1.99 -0.13 -0.7   1.1   1.28
  0.85  0.77  1.85 -0.79 -0.52 -0.15  1.12  0.43  1.03  0.94  1.45 -0.97
  0.36 -0.18  0.48  1.33  1.48 -0.96  0.67  0.95  0.87  0.01 -1.64 -0.47
  0.89  1.98  0.22 -1.78  0.59 -0.09 -1.02 -0.28  0.3   0.84 -1.61 -0.12
  3.19  0.42 -0.19 -0.56  2.72 -1.11 -0.57 -0.47  1.29 -0.68  0.99  0.93
  2.81  1.54  1.5   1.03  0.68  1.15 -0.79  0.74 -0.16  0.77  0.37  0.69
 -0.08  0.54 -0.59 -0.08 -0.3   0.39  1.4   0.4   0.21  0.19  1.38  0.19
  1.62  0.77 -1.01  2.28  1.89 -1.63  0.82 -0.3   0.69  0.7   0.63  0.18
  0.83 -1.36 -0.48  1.55]
y_C2 is
  [0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0
 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 1 1 1 1 0
 0 1 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0]
X_D2 is
```

```
 [[ -7.35   0.67]
[  5.47  -5.36]
[  8.55   2.96]
[ 12.23   1.46]
[  5.75  -5.75]
[  6.66   9.24]
[ -5.41   1.68]
[  6.58  -6.47]
[ 12.67   4.75]
[ -7.49   0.19]
[  7.84   1.4 ]
[ -3.15  -1.89]
[  9.56 -10.31]
[  5.21  -7.2 ]
[ 11.58 -10.14]
[ 11.54  -0.5 ]
[  0.86  -6.47]
[  5.07  -4.88]
[ -7.09   2.01]
[ -4.93  -0.62]
[ -5.83  -3.86]
[ -5.13   4.03]
[ -7.34   2.07]
[  9.28   5.34]
[  7.34 -10.44]
[  7.97   4.96]
[ -4.14  -1.58]
[ -7.06   3.09]
[  7.9    2.03]
[  9.4   -2.12]
[  9.63 -10.19]
[  8.49   1.65]
[ -6.78  -0.16]
[ -8.52   4.71]
[  9.06   5.86]
[  9.2    5.38]
[ 12.17  -9.32]
[  8.06   1.02]
[  7.41  -4.02]
[ -7.25   2.54]
[  6.93   8.  ]
[ -4.41   0.95]
[  5.17  -6.13]
[  9.36   9.76]
[ 11.47  -0.82]
[  3.22  -7.68]
[ -4.55  -0.2 ]
[ -4.06   0.03]
```

```
[  9.55  -0.63]
[ -3.47  -1.07]
[  5.48  -5.2 ]
[  4.34  -6.93]
[  8.3   -9.51]
[  4.48  -7.48]
[  4.71  -7.51]
[  6.44   9.19]
[  4.04  -6.63]
[ 10.77  -9.2 ]
[ 10.49  -9.03]
[  5.74  -5.77]
[  7.58  -6.82]
[  9.09   5.33]
[  7.07  10.33]
[ 10.66 -10.1 ]
[  4.93  -4.74]
[  4.99  -5.65]
[  9.59  -9.63]
[  9.8  -10.36]
[ -7.93   2.39]
[ 10.28   1.  ]
[  5.61  -6.52]
[ 10.14   0.83]
[ -7.24   0.04]
[  4.16  -3.09]
[ 10.32 -10.04]
[  8.44   4.53]
[ -3.59  -1.84]
[ 10.01   9.35]
[ 12.17   5.49]
[ -5.39  -2.91]
[ 10.12  11.07]
[  7.56   5.02]
[  2.56  -3.05]
[  9.89   1.38]
[  5.64  -6.67]
[ -4.01  -0.63]
[ -6.51  -2.82]
[  5.32   8.76]
[  7.33   7.8 ]
[  2.53  -6.36]
[  9.5    4.34]
[  8.88 -10.1 ]
[  7.55  11.04]
[  2.78  -5.12]
[ 10.48   1.9 ]
[  5.37  -5.81]
```

```
 [  7.1    8.31]
 [  7.7    4.28]
 [  7.73   7.7 ]
 [ 10.12   4.69]]
X_D2[:,0] is
 [ -7.35   5.47   8.55  12.23   5.75   6.66  -5.41   6.58  12.67  -7.49
   7.84  -3.15   9.56   5.21  11.58  11.54   0.86   5.07  -7.09  -4.93
  -5.83  -5.13  -7.34   9.28   7.34   7.97  -4.14  -7.06   7.9    9.4
   9.63   8.49  -6.78  -8.52   9.06   9.2   12.17   8.06   7.41  -7.25
   6.93  -4.41   5.17   9.36  11.47   3.22  -4.55  -4.06   9.55  -3.47
   5.48   4.34   8.3    4.48   4.71   6.44   4.04  10.77  10.49   5.74
   7.58   9.09   7.07  10.66   4.93   4.99   9.59   9.8   -7.93  10.28
   5.61  10.14  -7.24   4.16  10.32   8.44  -3.59  10.01  12.17  -5.39
  10.12   7.56   2.56   9.89   5.64  -4.01  -6.51   5.32   7.33   2.53
   9.5    8.88   7.55   2.78  10.48   5.37   7.1    7.7    7.73  10.12]
X_D2[:,1] is
 [  0.67  -5.36   2.96   1.46  -5.75   9.24   1.68  -6.47   4.75   0.19
   1.4   -1.89 -10.31  -7.2  -10.14  -0.5   -6.47  -4.88   2.01  -0.62
  -3.86   4.03   2.07   5.34 -10.44   4.96  -1.58   3.09   2.03  -2.12
 -10.19   1.65  -0.16   4.71   5.86   5.38  -9.32   1.02  -4.02   2.54   8.
   0.95  -6.13   9.76  -0.82  -7.68  -0.2    0.03  -0.63  -1.07  -5.2
  -6.93  -9.51  -7.48  -7.51   9.19  -6.63  -9.2   -9.03  -5.77  -6.82
   5.33  10.33 -10.1   -4.74  -5.65  -9.63 -10.36   2.39   1.    -6.52
   0.83   0.04  -3.09 -10.04   4.53  -1.84   9.35   5.49  -2.91  11.07
   5.02  -3.05   1.38  -6.67  -0.63  -2.82   8.76   7.8   -6.36   4.34
 -10.1   11.04  -5.12   1.9   -5.81   8.31   4.28   7.7    4.69]
y_D2 is
  [7 2 0 0 2 6 7 5 1 7 0 4 3 5 3 0 2 5 7 7 4 7 7 1 3 1 4 7 0 0 3 0 4 7 1 1 3
 0 5 7 6 4 2 6 0 2 4 4 0 4 5 2 3 5 5 6 2 3 3 5 5 1 6 3 5 2 3 3 7 0 2 0 7 2
 3 1 4 6 1 4 6 1 2 0 5 4 4 6 6 2 1 3 6 2 1 5 6 1 6 1]
new y_D2 is
  [1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1
 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 0 0 0 1 0
 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1]


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

```
In [2]: # Breast cancer dataset for classification
        cancer = load_breast_cancer()
        (X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)


        # Communities and Crime dataset
        (X_crime, y_crime) = load_crime_dataset()
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-2-15d60840d5d3> in <module>()
      1 # Breast cancer dataset for classification
----> 2 cancer = load_breast_cancer()
      3 (X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
      4
      5


NameError: name 'load_breast_cancer' is not defined
```

1.3 K-Nearest Neighbors

1.3.1 Classification

```
In [5]: from adspy_shared_utilities import plot_two_class_knn

        X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                            random_state=0) #SIMPLE

        plot_two_class_knn(X_train, y_train, 1, 'uniform', X_test, y_test)
        plot_two_class_knn(X_train, y_train, 3, 'uniform', X_test, y_test)
        plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

1.3.2 Regression

```
In [4]: from sklearn.neighbors import KNeighborsRegressor

        X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1, random_stat

        knnreg = KNeighborsRegressor(n_neighbors = 5).fit(X_train, y_train)

        print('prediction is \n',knnreg.predict(X_test), 'testing is \n',y_test)
        print('R-squared test score: {:.3f}'
             .format(knnreg.score(X_test, y_test)))
```

```
prediction is
 [ 231.71  148.36  150.59  150.59   72.15  166.51  141.91  235.57  208.26
  102.1   191.32  134.5   228.32  148.36  159.17  113.47  144.04  199.23
  143.19  166.51  231.71  208.26  128.02  123.14  141.91] testing is
 [ 245.1   136.06  150.56  174.29   57.58  145.7   149.87  184.41  186.77
  103.44  143.6    95.27  160.22   99.06  166.52   85.72  187.93  221.
  209.37  196.7   206.67  176.42  169.91  131.6    81.58]
R-squared test score: 0.425
```

```
In [5]: fig, subaxes = plt.subplots(1, 2, figsize=(8,4))
        X_predict_input = np.linspace(-3, 3, 50).reshape(-1,1)
        print(X_predict_input)
        X_train, X_test, y_train, y_test = train_test_split(X_R1[0::5], y_R1[0::5],

        for thisaxis, K in zip(subaxes, [1, 3]):
            knnreg = KNeighborsRegressor(n_neighbors = K).fit(X_train, y_train)
            y_predict_output = knnreg.predict(X_predict_input)
            thisaxis.set_xlim([-2.5, 0.75])
            thisaxis.plot(X_predict_input, y_predict_output, '^', markersize = 10,
                          label='Predicted', alpha=0.8)
            thisaxis.plot(X_train, y_train, 'o', label='True Value', alpha=0.8)
            thisaxis.set_xlabel('Input feature')
            thisaxis.set_ylabel('Target value')
            thisaxis.set_title('KNN regression (K={})'.format(K))
            thisaxis.legend()
        plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[[-3.  ]
 [-2.88]
 [-2.76]
```

```
[-2.63]
[-2.51]
[-2.39]
[-2.27]
[-2.14]
[-2.02]
[-1.9 ]
[-1.78]
[-1.65]
[-1.53]
[-1.41]
[-1.29]
[-1.16]
[-1.04]
[-0.92]
[-0.8 ]
[-0.67]
[-0.55]
[-0.43]
[-0.31]
[-0.18]
[-0.06]
[ 0.06]
[ 0.18]
[ 0.31]
[ 0.43]
[ 0.55]
[ 0.67]
[ 0.8 ]
[ 0.92]
[ 1.04]
[ 1.16]
[ 1.29]
[ 1.41]
[ 1.53]
[ 1.65]
[ 1.78]
[ 1.9 ]
[ 2.02]
[ 2.14]
[ 2.27]
[ 2.39]
[ 2.51]
[ 2.63]
[ 2.76]
[ 2.88]
[ 3.  ]]
```

1.3.3 Regression model complexity as a function of K

```
In [6]: # plot k-NN regression on sample dataset for different values of K
        fig, subaxes = plt.subplots(5, 1, figsize=(5,20))
        X_predict_input = np.linspace(-3, 3, 500).reshape(-1,1)
        X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1,
                                                            random_state = 0)

        for thisaxis, K in zip(subaxes, [1, 3, 7, 15, 55]):
            knnreg = KNeighborsRegressor(n_neighbors = K).fit(X_train, y_train)
            y_predict_output = knnreg.predict(X_predict_input)
            train_score = knnreg.score(X_train, y_train)
            test_score = knnreg.score(X_test, y_test)
            thisaxis.plot(X_predict_input, y_predict_output)
            thisaxis.plot(X_train, y_train, 'o', alpha=0.9, label='Train')
            thisaxis.plot(X_test, y_test, '^', alpha=0.9, label='Test')
            thisaxis.set_xlabel('Input feature')
            thisaxis.set_ylabel('Target value')
            thisaxis.set_title('KNN Regression (K={})\n\
        Train $R^2 = {:.3f}$,  Test $R^2 = {:.3f}$'
                                 .format(K, train_score, test_score))
            thisaxis.legend()
            plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.4 Linear models for regression

1.4.1 Linear regression

1.4.2 linreg.coef_ = w, and linreg.intercept_ = b. OBSERVE THE _ AT THE END, yaha print nahi hua, lekin hai

```
In [7]: from sklearn.linear_model import LinearRegression

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

        print(X_train, X_test, y_train, y_test )
        linreg = LinearRegression().fit(X_train, y_train)

        print('linear model coeff (w): {}'
             .format(linreg.coef_))
        print('linear model intercept (b): {:.3f}'
             .format(linreg.intercept_))
        print('R-squared score (training): {:.3f}'
```

```
            .format(linreg.score(X_train, y_train)))
        print('R-squared score (test): {:.3f}'
            .format(linreg.score(X_test, y_test)))
```

```
[[ 1.76]
 [-0.98]
 [-1.18]
 [-0.15]
 [ 0.13]
 [-0.87]
 [ 0.86]
 [-1.23]
 [ 1.22]
 [-1.42]
 [ 0.33]
 [ 1.45]
 [ 0.39]
 [-1.98]
 [-0.67]
 [ 0.98]
 [ 0.3 ]
 [-1.71]
 [ 0.4 ]
 [-0.89]
 [ 1.18]
 [ 1.79]
 [ 1.05]
 [ 0.05]
 [ 0.44]
 [-0.21]
 [-0.19]
 [-0.36]
 [ 0.01]
 [-0.35]
 [-0.91]
 [ 0.13]
 [ 0.46]
 [ 0.31]
 [ 0.47]
 [-0.51]
 [-0.58]
 [-0.03]
 [-1.73]
 [ 0.06]
 [-0.3 ]
 [-1.05]
 [-0.1 ]
 [ 0.65]
```

```
[-0.51]
[-0.68]
[ 1.53]
[ 0.41]
[ 0.36]
[-1.61]
[ 1.87]
[ 0.98]
[ 0.05]
[ 0.07]
[ 0.95]
[ 0.78]
[-0.4 ]
[ 0.71]
[ 1.23]
[-1.25]
[-1.45]
[ 1.47]
[-1.54]
[ 0.12]
[ 2.27]
[ 0.73]
[ 0.14]
[ 1.9 ]
[-1.07]
[-0.31]
[-0.36]
[ 0.15]
[-2.55]
[-0.85]
[ 0.18]] [[ 1.95]
[-0.39]
[ 0.4 ]
[ 0.4 ]
[-1.63]
[ 0.16]
[-0.18]
[ 1.49]
[ 1.14]
[-1.17]
[ 0.9 ]
[-0.63]
[ 1.49]
[-0.4 ]
[ 0.43]
[-0.9 ]
[-0.44]
[ 0.76]
```

```
 [ 0.38]
 [ 0.21]
 [ 2.24]
 [ 1.2 ]
 [-0.81]
 [-0.74]
 [-0.21]] [ 217.54   63.87   83.82  137.31  188.15  116.34  193.31  116.63  181.29
   54.41  159.38  267.55  117.77   53.42  127.36  131.23  143.89   83.61
  168.56  118.16  231.4   220.19  163.73   99.18  196.36  159.32  150.69
  120.61   83.89  187.46  151.73  167.15  203.09  165.96  112.72  110.9
  133.32  102.12    8.01  178.13   98.45   85.72  163.76  186.67  164.32
  136.61  225.78  115.11  155.14  128.52  254.29  222.41  123.87   98.48
  158.16  251.48  145.01  171.07  242.47  118.33   89.85  224.52   50.76
  194.18  244.83  193.64  201.83  221.7   106.02  109.35  179.35   98.35
   24.22  117.24  177.06] [ 245.1   136.06  150.56  174.29   57.58  145.7   149.87
  103.44  143.6    95.27  160.22   99.06  166.52   85.72  187.93  221.
  209.37  196.7   206.67  176.42  169.91  131.6    81.58]
linear model coeff (w): [ 45.71]
linear model intercept (b): 148.446
R-squared score (training): 0.679
R-squared score (test): 0.492
```

1.4.3 Linear regression: example plot

```
In [8]: plt.figure(figsize=(5,4))
        plt.scatter(X_R1, y_R1, marker= 'o', s=50, alpha=0.8)
        plt.plot(X_R1, linreg.coef_ * X_R1 + linreg.intercept_, 'r-') #PLOTTING X A
        plt.title('Least-squares linear regression')
        plt.xlabel('Feature value (x)')
        plt.ylabel('Target value (y)')
        plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.4.4 itne saare linreg.coeff_ kaha se aaye - ye weights hai!!!!

1.4.5 since we have multiple parameters, to weights wala system hoga na

1.4.6 one parameter rehta to line ban gayi hoti

1.4.7 wo {:.3f} ka kya system hai? - only to display 3 decimal places

```
In [9]: X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                            random_state = 0)
        linreg = LinearRegression().fit(X_train, y_train)
```

```
#print(linreg) #kuch bhi information nahi mila yaha se
#print(X_crime, y_crime)
#print('X_train is \n', X_train,'X_test is \n', X_test,'y_train is \n', y_t

print('Crime dataset')
print('linear model intercept: {}'.format(linreg.intercept_))
print('linear model coeff:\n{}'.format(linreg.coef_))
#print('R-squared score (training): {:.3f}'.format(linreg.score(X_train, y_
print('R-squared score (training): {}'.format(linreg.score(X_train, y_train
print('R-squared score (test): {:.3f}'.format(linreg.score(X_test, y_test))
```

```
Crime dataset
linear model intercept: -1728.1306726048451
linear model coeff:
[  1.62e-03  -9.43e+01   1.36e+01  -3.13e+01  -8.15e-02  -1.69e+01
  -2.43e-03   1.53e+00  -1.39e-02  -7.72e+00   2.28e+01  -5.66e+00
   9.35e+00   2.07e-01  -7.43e+00   9.66e-03   4.38e-03   4.80e-03
  -4.46e+00  -1.61e+01   8.83e+00  -5.07e-01  -1.42e+00   8.18e+00
  -3.87e+00  -3.54e+00   4.49e+00   9.31e+00   1.74e+02   1.18e+01
   1.51e+02  -3.30e+02  -1.35e+02   6.95e-01  -2.38e+01   2.77e+00
   3.82e-01   4.39e+00  -1.06e+01  -4.92e-03   4.14e+01  -1.16e-03
   1.19e+00   1.75e+00  -3.68e+00   1.60e+00  -8.42e+00  -3.80e+01
   4.74e+01  -2.51e+01  -2.88e-01  -3.66e+01   1.90e+01  -4.53e+01
   6.83e+02   1.04e+02  -3.29e+02  -3.14e+01   2.74e+01   5.12e+00
   6.92e+01   1.98e-02  -6.12e-01   2.65e+01   1.01e+01  -1.59e+00
   2.24e+00   7.38e+00  -3.14e+01  -9.78e-05   5.02e-05  -3.48e-04
  -2.50e-04  -5.27e-01  -5.17e-01  -4.10e-01   1.16e-01   1.46e+00
  -3.04e-01   2.44e+00  -3.66e+01   1.41e-01   2.89e-01   1.77e+01
   5.97e-01   1.98e+00  -1.36e-01  -1.85e+00]
R-squared score (training): 0.6731208442568583
R-squared score (test): 0.496
```

1.4.8 Ridge regression

```
In [10]: from sklearn.linear_model import Ridge
         X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                             random_state = 0)

         linridge = Ridge(alpha=20.0).fit(X_train, y_train)

         print(X_crime.shape)
         print('Crime dataset')
         print('ridge regression linear model intercept: {}'
               .format(linridge.intercept_))
         print('ridge regression linear model coeff:\n{}'
               .format(linridge.coef_))
         print('R-squared score (training): {:.3f}'
```

```
            .format(linridge.score(X_train, y_train)))
        print('R-squared score (test): {:.3f}'
            .format(linridge.score(X_test, y_test)))
        print('Number of non-zero features: {}'
            .format(np.sum(linridge.coef_ != 0)))
        print('Number of zero features: {}'
            .format(np.sum(linridge.coef_ == 0))) # all features have been assign
```

```
(1994, 88)
Crime dataset
ridge regression linear model intercept: -3352.423035846206
ridge regression linear model coeff:
[  1.95e-03   2.19e+01   9.56e+00  -3.59e+01   6.36e+00  -1.97e+01
  -2.81e-03   1.66e+00  -6.61e-03  -6.95e+00   1.72e+01  -5.63e+00
   8.84e+00   6.79e-01  -7.34e+00   6.70e-03   9.79e-04   5.01e-03
  -4.90e+00  -1.79e+01   9.18e+00  -1.24e+00   1.22e+00   1.03e+01
  -3.78e+00  -3.73e+00   4.75e+00   8.43e+00   3.09e+01   1.19e+01
  -2.05e+00  -3.82e+01   1.85e+01   1.53e+00  -2.20e+01   2.46e+00
   3.29e-01   4.02e+00  -1.13e+01  -4.70e-03   4.27e+01  -1.23e-03
   1.41e+00   9.35e-01  -3.00e+00   1.12e+00  -1.82e+01  -1.55e+01
   2.42e+01  -1.32e+01  -4.20e-01  -3.60e+01   1.30e+01  -2.81e+01
   4.39e+01   3.87e+01  -6.46e+01  -1.64e+01   2.90e+01   4.15e+00
   5.34e+01   1.99e-02  -5.47e-01   1.24e+01   1.04e+01  -1.57e+00
   3.16e+00   8.78e+00  -2.95e+01  -2.33e-04   3.14e-04  -4.14e-04
  -1.80e-04  -5.74e-01  -5.18e-01  -4.21e-01   1.53e-01   1.33e+00
   3.85e+00   3.03e+00  -3.78e+01   1.38e-01   3.08e-01   1.57e+01
   3.31e-01   3.36e+00   1.61e-01  -2.68e+00]
R-squared score (training): 0.671
R-squared score (test): 0.494
Number of non-zero features: 88
Number of zero features: 0
```

Ridge regression with feature normalization

1.4.9 normalisation se score improve hua hai!!

In [11]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                    random_state = 0)

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

linridge = Ridge(alpha=20.0).fit(X_train_scaled, y_train)
```

```
print('Crime dataset')
print('ridge regression linear model intercept: {}'
     .format(linridge.intercept_))
print('ridge regression linear model coeff:\n{}'
     .format(linridge.coef_))
print('R-squared score (training): {:.3f}'
     .format(linridge.score(X_train_scaled, y_train)))
print('R-squared score (test): {:.3f}'
     .format(linridge.score(X_test_scaled, y_test)))
print('Number of non-zero features: {}'
     .format(np.sum(linridge.coef_ != 0)))
```

```
Crime dataset
ridge regression linear model intercept: 933.390638504416
ridge regression linear model coeff:
[  88.69   16.49  -50.3   -82.91  -65.9    -2.28   87.74  150.95   18.88
  -31.06  -43.14 -189.44   -4.53  107.98  -76.53    2.86   34.95   90.14
   52.46  -62.11  115.02    2.67    6.94   -5.67 -101.55  -36.91   -8.71
   29.12  171.26   99.37   75.07  123.64   95.24 -330.61 -442.3  -284.5
 -258.37   17.66 -101.71  110.65  523.14   24.82    4.87  -30.47   -3.52
   50.58   10.85   18.28   44.11   58.34   67.09  -57.94  116.14   53.81
   49.02   -7.62   55.14  -52.09  123.39   77.13   45.5   184.91  -91.36
    1.08  234.09   10.39   94.72  167.92  -25.14   -1.18   14.6    36.77
   53.2   -78.86   -5.9    26.05  115.15   68.74   68.29   16.53  -97.91
  205.2    75.97   61.38  -79.83   67.27   95.67  -11.88]
R-squared score (training): 0.615
R-squared score (test): 0.599
Number of non-zero features: 88
```

Ridge regression with regularization parameter: alpha

```
In [12]: print('Ridge regression: effect of alpha regularization parameter\n')
         for this_alpha in [0, 1, 10, 20, 50, 100, 1000]:
             linridge = Ridge(alpha = this_alpha).fit(X_train_scaled, y_train)
             r2_train = linridge.score(X_train_scaled, y_train)
             r2_test = linridge.score(X_test_scaled, y_test)
             num_coeff_bigger = np.sum(abs(linridge.coef_) > 1.0) #ye kya hai bhai
             print('Alpha = {:.2f}\nnum abs(coeff) > 1.0: {}, \
         r-squared training: {:.2f}, r-squared test: {:.2f}\n'
                  .format(this_alpha, num_coeff_bigger, r2_train, r2_test)) #num ab
```

```
Ridge regression: effect of alpha regularization parameter



/opt/conda/lib/python3.6/site-packages/scipy/linalg/basic.py:223: RuntimeWarning: s
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
```

```
Reciprocal condition number: 6.332952875642905e-19
  ' condition number: {}'.format(rcond), RuntimeWarning)


Alpha = 0.00
num abs(coeff) > 1.0: 88, r-squared training: 0.67, r-squared test: 0.50

Alpha = 1.00
num abs(coeff) > 1.0: 87, r-squared training: 0.66, r-squared test: 0.56

Alpha = 10.00
num abs(coeff) > 1.0: 87, r-squared training: 0.63, r-squared test: 0.59

Alpha = 20.00
num abs(coeff) > 1.0: 88, r-squared training: 0.61, r-squared test: 0.60

Alpha = 50.00
num abs(coeff) > 1.0: 86, r-squared training: 0.58, r-squared test: 0.58

Alpha = 100.00
num abs(coeff) > 1.0: 87, r-squared training: 0.55, r-squared test: 0.55

Alpha = 1000.00
num abs(coeff) > 1.0: 84, r-squared training: 0.31, r-squared test: 0.30
```

1.4.10 Lasso regression

1.4.11 new and nice list comprehension for getting sorted list

```
In [13]: from sklearn.linear_model import Lasso
         from sklearn.preprocessing import MinMaxScaler
         scaler = MinMaxScaler()

         X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                             random_state = 0)

         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         linlasso = Lasso(alpha=2.0, max_iter = 10000).fit(X_train_scaled, y_train)

         print('Crime dataset')
         print('lasso regression linear model intercept: {}'
              .format(linlasso.intercept_))
         print('lasso regression linear model coeff:\n{}'
              .format(linlasso.coef_))
         print('Non-zero features: {}'
```

```
            .format(np.sum(linlasso.coef_ != 0)))
        print('zero features: {}'
            .format(np.sum(linlasso.coef_ == 0))) #big number bro
        print('R-squared score (training): {:.3f}'
            .format(linlasso.score(X_train_scaled, y_train)))
        print('R-squared score (test): {:.3f}\n'
            .format(linlasso.score(X_test_scaled, y_test)))


        print('\n\n')
        print('two lists are\n',list(X_crime),'\n', linlasso.coef_) #list(X_crime)
        print('\n zip is\n',zip(list(X_crime), linlasso.coef_))
        print('\n list of zip is\n',list(zip(list(X_crime), linlasso.coef_)))

        print('Features with non-zero weight (sorted by absolute magnitude):')
        for e in sorted (list(zip(list(X_crime), linlasso.coef_)),
                         key = lambda e: -abs(e[1])):

            #key is needed for sorting the list by values (values are the coeff)
            #negative sign makes it sort in desending order, usual order is positi
            if e[1] != 0:
                print('\t{}, {:.3f}'.format(e[0], e[1]))
```

```
Crime dataset
lasso regression linear model intercept: 1186.6120619985809
lasso regression linear model coeff:
[    0.       0.      -0.    -168.18    -0.      -0.       0.     119.69
     0.      -0.       0.    -169.68    -0.       0.      -0.       0.
     0.       0.      -0.      -0.       0.      -0.       0.       0.
   -57.53    -0.      -0.       0.     259.33    -0.       0.       0.
     0.      -0.   -1188.74    -0.      -0.      -0.    -231.42     0.
  1488.37     0.      -0.      -0.      -0.       0.       0.       0.
     0.       0.      -0.       0.      20.14     0.       0.       0.
     0.       0.     339.04     0.       0.     459.54    -0.       0.
   122.69    -0.      91.41     0.      -0.       0.       0.      73.14
     0.      -0.       0.       0.      86.36     0.       0.       0.
  -104.57   264.93     0.      23.45   -49.39     0.       5.2      0.  ]
Non-zero features: 20
zero features: 68
R-squared score (training): 0.631
R-squared score (test): 0.624




two lists are
 ['population', 'householdsize', 'agePct12t21', 'agePct12t29', 'agePct16t24', 'ageP
 [    0.       0.      -0.    -168.18    -0.      -0.       0.     119.69
```

```
     0.      -0.       0.    -169.68    -0.       0.      -0.       0.
     0.       0.      -0.      -0.       0.      -0.       0.       0.
   -57.53    -0.      -0.       0.     259.33    -0.       0.       0.
     0.      -0.   -1188.74    -0.      -0.      -0.    -231.42     0.
  1488.37     0.      -0.      -0.      -0.       0.       0.       0.
     0.       0.      -0.       0.      20.14     0.       0.       0.
     0.       0.     339.04     0.       0.     459.54    -0.       0.
   122.69    -0.      91.41     0.      -0.       0.       0.      73.14
     0.      -0.       0.       0.      86.36     0.       0.       0.
  -104.57   264.93     0.      23.45   -49.39     0.       5.2      0.  ]

 zip is
 <zip object at 0x7ff5b7203588>

 list of zip is
 [('population', 0.0), ('householdsize', 0.0), ('agePct12t21', -0.0), ('agePct12t29
Features with non-zero weight (sorted by absolute magnitude):
        PctKidsBornNeverMar, 1488.365
        PctKids2Par, -1188.740
        HousVacant, 459.538
        PctPersDenseHous, 339.045
        NumInShelters, 264.932
        MalePctDivorce, 259.329
        PctWorkMom, -231.423
        pctWInvInc, -169.676
        agePct12t29, -168.183
        PctVacantBoarded, 122.692
        pctUrban, 119.694
        MedOwnCostPctIncNoMtg, -104.571
        MedYrHousBuilt, 91.412
        RentQrange, 86.356
        OwnOccHiQuart, 73.144
        PctEmplManu, -57.530
        PctBornSameState, -49.394
        PctForeignBorn, 23.449
        PctLargHouseFam, 20.144
        PctSameCity85, 5.198
```

Lasso regression with regularization parameter: alpha

1.4.12 nice nice, as alpha increases, number of features kept decreases (underfitting increases)

```
In [14]: print('Lasso regression: effect of alpha regularization\n\
         parameter on number of features kept in final model\n')

         for alpha in [0.5, 1, 2, 3, 5, 10, 20, 50]:
             linlasso = Lasso(alpha, max_iter = 10000).fit(X_train_scaled, y_train)
```

```
    r2_train = linlasso.score(X_train_scaled, y_train)
    r2_test = linlasso.score(X_test_scaled, y_test)

    print('Alpha = {:.2f}\nFeatures kept: {}, r-squared training: {:.2f},
r-squared test: {:.2f}\n'
          .format(alpha, np.sum(linlasso.coef_ != 0), r2_train, r2_test))
```

```
Lasso regression: effect of alpha regularization
parameter on number of features kept in final model

Alpha = 0.50
Features kept: 35, r-squared training: 0.65, r-squared test: 0.58

Alpha = 1.00
Features kept: 25, r-squared training: 0.64, r-squared test: 0.60

Alpha = 2.00
Features kept: 20, r-squared training: 0.63, r-squared test: 0.62

Alpha = 3.00
Features kept: 17, r-squared training: 0.62, r-squared test: 0.63

Alpha = 5.00
Features kept: 12, r-squared training: 0.60, r-squared test: 0.61

Alpha = 10.00
Features kept: 6, r-squared training: 0.57, r-squared test: 0.58

Alpha = 20.00
Features kept: 2, r-squared training: 0.51, r-squared test: 0.50

Alpha = 50.00
Features kept: 1, r-squared training: 0.31, r-squared test: 0.30
```

1.4.13 Polynomial regression

1.4.14 linear model has 7 features, so quadratic model should have (linear terms)7 + (square terms) 7 + (7C2) 21 = 35 terms, phir yaha 36 kaise aaye

1.4.15 Ridge transform ke liye sirf ridge lagta hai

```
In [15]: from sklearn.linear_model import LinearRegression
         from sklearn.linear_model import Ridge
         from sklearn.preprocessing import PolynomialFeatures


         X_train, X_test, y_train, y_test = train_test_split(X_F1, y_F1,
```

```
                                                    random_state = 0) #ye c
linreg = LinearRegression().fit(X_train, y_train)
print('running linear regression \n\n')
print('linear model coeff (w): {}'
     .format(linreg.coef_))
print('linear model intercept (b): {:.3f}'
     .format(linreg.intercept_))
print('R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('R-squared score (test): {:.3f}'
     .format(linreg.score(X_test, y_test)))

print('\nNow we transform the original input data to add\n\
polynomial features up to degree 2 (quadratic)\n')
poly = PolynomialFeatures(degree=2)
X_F1_poly = poly.fit_transform(X_F1)

#print('X_F1 is \n',X_F1,'\n X_F1_poly is \n',X_F1_poly)
#print(len(X_F1),len(X_F1_poly))
#print(X_F1[0],X_F1_poly[0])
#print(len(X_F1[0]),len(X_F1_poly[0]))

X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, y_F1,
                                                    random_state = 0)
linreg = LinearRegression().fit(X_train, y_train)

print('(poly deg 2) linear model coeff (w):\n{}'
     .format(linreg.coef_))
print('(poly deg 2) linear model intercept (b): {:.3f}'
     .format(linreg.intercept_))
print('(poly deg 2) R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('(poly deg 2) R-squared score (test): {:.3f}\n'
     .format(linreg.score(X_test, y_test)))

print('\nAddition of many polynomial features often leads to\n\
overfitting, so we often use polynomial features in combination\n\
with regression that has a regularization penalty, like ridge\n\
regression.\n')

X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, y_F1,
                                                    random_state = 0)
linreg = Ridge().fit(X_train, y_train)

print('(poly deg 2 + ridge) linear model coeff (w):\n{}'
     .format(linreg.coef_))
print('(poly deg 2 + ridge) linear model intercept (b): {:.3f}'
     .format(linreg.intercept_))
```

```
        print('(poly deg 2 + ridge) R-squared score (training): {:.3f}'
              .format(linreg.score(X_train, y_train)))
        print('(poly deg 2 + ridge) R-squared score (test): {:.3f}'
              .format(linreg.score(X_test, y_test)))
```

```
running linear regression


linear model coeff (w): [  4.42   6.     0.53  10.24   6.55  -2.02  -0.32]
linear model intercept (b): 1.543
R-squared score (training): 0.722
R-squared score (test): 0.722

Now we transform the original input data to add
polynomial features up to degree 2 (quadratic)

(poly deg 2) linear model coeff (w):
[  3.41e-12   1.66e+01   2.67e+01  -2.21e+01   1.24e+01   6.93e+00
   1.05e+00   3.71e+00  -1.34e+01  -5.73e+00   1.62e+00   3.66e+00
   5.05e+00  -1.46e+00   1.95e+00  -1.51e+01   4.87e+00  -2.97e+00
  -7.78e+00   5.15e+00  -4.65e+00   1.84e+01  -2.22e+00   2.17e+00
  -1.28e+00   1.88e+00   1.53e-01   5.62e-01  -8.92e-01  -2.18e+00
   1.38e+00  -4.90e+00  -2.24e+00   1.38e+00  -5.52e-01  -1.09e+00]
(poly deg 2) linear model intercept (b): -3.206
(poly deg 2) R-squared score (training): 0.969
(poly deg 2) R-squared score (test): 0.805


Addition of many polynomial features often leads to
overfitting, so we often use polynomial features in combination
with regression that has a regularization penalty, like ridge
regression.

(poly deg 2 + ridge) linear model coeff (w):
[ 0.    2.23  4.73 -3.15  3.86  1.61 -0.77 -0.15 -1.75  1.6   1.37  2.52
  2.72  0.49 -1.94 -1.63  1.51  0.89  0.26  2.05 -1.93  3.62 -0.72  0.63
 -3.16  1.29  3.55  1.73  0.94 -0.51  1.7  -1.98  1.81 -0.22  2.88 -0.89]
(poly deg 2 + ridge) linear model intercept (b): 5.418
(poly deg 2 + ridge) R-squared score (training): 0.826
(poly deg 2 + ridge) R-squared score (test): 0.825
```

1.5 Linear models for classification

1.5.1 Logistic regression

Logistic regression for binary classification on fruits dataset using height, width features (positive class: apple, negative class: others)

1.5.2 We need to train in matrix form and not direct df

2 Lines 31, 32 are new!!!!!! ['a','b'][False] gives a, ['a','b'][True] gives b

In [16]:
```
from sklearn.linear_model import LogisticRegression
from adspy_shared_utilities import (
plot_class_regions_for_classifier_subplot)

#print('y_fruits_2d initially is \n',y_fruits_2d)
#print('shapes of X_fruits_2d and y_fruits_2d are \n', X_fruits_2d.shape,y

fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
y_fruits_apple = y_fruits_2d == 1   # make into a binary problem: apples v

#print('X_fruits_2d is \n', X_fruits_2d,
#      '\n y_fruits_apple is \n', y_fruits_apple)
#print('X_fruits_2d.as_matrix() is \n', X_fruits_2d.as_matrix(),
#      '\n y_fruits_apple.as_matrix() is \n', y_fruits_apple.as_matrix())

X_train, X_test, y_train, y_test = (
train_test_split(X_fruits_2d.as_matrix(),
                y_fruits_apple.as_matrix(),
                random_state = 0))

clf = LogisticRegression(C=100).fit(X_train, y_train)
plot_class_regions_for_classifier_subplot(clf, X_train, y_train, None,
                                          None, 'Logistic regression \
for binary classification\nFruit dataset: Apple vs others',
                                          subaxes)

#individual examples liye hai abhi 2
h = 6
w = 8
print('result is', clf.predict([[h,w]]), clf.predict([[h,w]])[0] )#, clf.p
print(['a','b'][True])
print(['a','b'][False])
print('A fruit with height {} and width {} is predicted to be: {}'
     .format(h,w, ['not an apple', 'an apple'][clf.predict([[h,w]])[0]]))

h = 10
w = 7
print('A fruit with height {} and width {} is predicted to be: {}'
     .format(h,w, ['not an apple', 'an apple'][clf.predict([[h,w]])[0]]))
subaxes.set_xlabel('height')
subaxes.set_ylabel('width')

print('Accuracy of Logistic regression classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
```

```
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
result is [ True] True
b
a
A fruit with height 6 and width 8 is predicted to be: an apple
A fruit with height 10 and width 7 is predicted to be: not an apple
Accuracy of Logistic regression classifier on training set: 0.77
Accuracy of Logistic regression classifier on test set: 0.73
```

2.1 Remember logistic regression is used for classification

Logistic regression on simple synthetic dataset

```
In [17]: from sklearn.linear_model import LogisticRegression
         from adspy_shared_utilities import (
         plot_class_regions_for_classifier_subplot)


         X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                             random_state = 0) #simp

         fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
         clf = LogisticRegression().fit(X_train, y_train)
         title = 'Logistic regression, simple synthetic dataset C = {:.3f}'.format
         plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                   None, None, title, subaxes)

         print('Accuracy of Logistic regression classifier on training set: {:.2f}'
              .format(clf.score(X_train, y_train)))
         print('Accuracy of Logistic regression classifier on test set: {:.2f}'
              .format(clf.score(X_test, y_test)))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Accuracy of Logistic regression classifier on training set: 0.80
Accuracy of Logistic regression classifier on test set: 0.80
```

Logistic regression regularization: C parameter

```
In [18]: X_train, X_test, y_train, y_test = (
         train_test_split(X_fruits_2d.as_matrix(),
                          y_fruits_apple.as_matrix(),
                          random_state=0))

         fig, subaxes = plt.subplots(3, 1, figsize=(4, 10))

         for this_C, subplot in zip([0.1, 1, 100], subaxes):
             clf = LogisticRegression(C=this_C).fit(X_train, y_train)
             title ='Logistic regression (apple vs rest), C = {:.3f}'.format(this_C
             #title me train score and test score kaha se aa raha hai?

             plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                       X_test, y_test, title,
                                                       subplot)
         plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Application to real dataset

```
In [19]: from sklearn.linear_model import LogisticRegression

         X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ra

         clf = LogisticRegression().fit(X_train, y_train)
         print('Breast cancer dataset')
         print('Accuracy of Logistic regression classifier on training set: {:.2f}'
              .format(clf.score(X_train, y_train)))
         print('Accuracy of Logistic regression classifier on test set: {:.2f}'
              .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of Logistic regression classifier on training set: 0.96
Accuracy of Logistic regression classifier on test set: 0.96
```

2.1.1 Support Vector Machines

Linear Support Vector Machine

```
In [20]: from sklearn.svm import SVC
         from adspy_shared_utilities import plot_class_regions_for_classifier_subpl
```

```
X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2, random_sta

fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
this_C = 1.0
clf = SVC(kernel = 'linear', C=this_C).fit(X_train, y_train) #interesting,
title = 'Linear SVC, C = {:.3f}'.format(this_C)
plot_class_regions_for_classifier_subplot(clf, X_train, y_train, None, Nor
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Linear Support Vector Machine: C parameter

```
In [21]: from sklearn.svm import LinearSVC
         from adspy_shared_utilities import plot_class_regions_for_classifier

         X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2, random_sta
         fig, subaxes = plt.subplots(1, 2, figsize=(8, 4))

         for this_C, subplot in zip([0.00001, 100], subaxes):
             clf = LinearSVC(C=this_C).fit(X_train, y_train)
             title = 'Linear SVC, C = {:.5f}'.format(this_C)
             plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                       None, None, title, subplot)
         plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Application to real dataset

```
In [22]: from sklearn.svm import LinearSVC
         X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ra

         clf = LinearSVC().fit(X_train, y_train)
         print('Breast cancer dataset')
         print('Accuracy of Linear SVC classifier on training set: {:.2f}'
               .format(clf.score(X_train, y_train)))
         print('Accuracy of Linear SVC classifier on test set: {:.2f}'
               .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of Linear SVC classifier on training set: 0.89
Accuracy of Linear SVC classifier on test set: 0.87
```

2.1.2 Multi-class classification with linear models

LinearSVC with M classes generates M one vs rest classifiers. Right, all are one vs all other classifiers

```
In [23]: from sklearn.svm import LinearSVC

         X_train, X_test, y_train, y_test = train_test_split(X_fruits_2d, y_fruits_

         clf = LinearSVC(C=5, random_state = 67).fit(X_train, y_train)
         print('Coefficients:\n', clf.coef_)
         print('Intercepts:\n', clf.intercept_)
```

```
Coefficients:
 [[-0.26  0.71]
 [-1.63  1.16]
 [ 0.03  0.29]
 [ 1.24 -1.64]]
Intercepts:
 [-3.29  1.2  -2.72  1.16]
```

Multi-class results on the fruit dataset

```
In [24]: plt.figure(figsize=(6,6))
         colors = ['r', 'g', 'b', 'y']
         cmap_fruits = ListedColormap(['#FF0000', '#00FF00', '#0000FF','#FFFF00'])

         plt.scatter(X_fruits_2d[['height']], X_fruits_2d[['width']],
                     c=y_fruits_2d, cmap=cmap_fruits, edgecolor = 'black', alpha=.7)

         x_0_range = np.linspace(-10, 15)

         for w, b, color in zip(clf.coef_, clf.intercept_, ['r', 'g', 'b', 'y']): #
             # Since class prediction with a linear model uses the formula y = w_0
             # and the decision boundary is defined as being all points with y = 0,
             # function of x_0 we just solve w_0 x_0 + w_1 x_1 + b = 0 for x_1:
             plt.plot(x_0_range, -(x_0_range * w[0] + b) / w[1], c=color, alpha=.8)
             #VARIBALE X_1 KAAM KA HAI. HOW???????

         plt.legend(target_names_fruits)
         plt.xlabel('height')
         plt.ylabel('width')
         plt.xlim(-2, 12)
```

```
plt.ylim(-2, 15)
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2.2 Kernelized Support Vector Machines

2.2.1 Classification

```
In [25]: from sklearn.svm import SVC
         from adspy_shared_utilities import plot_class_regions_for_classifier

         X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta

         # The default SVC kernel is radial basis function (RBF)
         #plot_class_regions_for_classifier(SVC().fit(X_train, y_train),
         #                                  X_train, y_train, None, None,
         #                                  'Support Vector Classifier: RBF kernel')

         plot_class_regions_for_classifier(SVC(kernel = 'rbf').fit(X_train, y_train
                                           X_train, y_train, None, None,
                                           'Support Vector Classifier: RBF kernel')

         # Compare decision boundries with polynomial kernel, degree = 3
         plot_class_regions_for_classifier(SVC(kernel = 'poly', degree = 3)
                                           .fit(X_train, y_train), X_train,
                                           y_train, None, None,
                                           'Support Vector Classifier: Polynomial ke
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Support Vector Machine with RBF kernel: gamma parameter

```
In [26]: from adspy_shared_utilities import plot_class_regions_for_classifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta
fig, subaxes = plt.subplots(3, 1, figsize=(4, 11))

#print(subaxes)
for this_gamma, subplot in zip([0.01, 1.0, 10.0], subaxes):
    clf = SVC(kernel = 'rbf', gamma=this_gamma).fit(X_train, y_train)
    title = 'Support Vector Classifier: \nRBF kernel, gamma = {:.2f}'.form
    plot_class_regions_for_classifier_subplot(clf, X_train, y_train, None,
    plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Support Vector Machine with RBF kernel: using both C and gamma parameter

```
In [27]: from sklearn.svm import SVC
         from adspy_shared_utilities import plot_class_regions_for_classifier_subpl

         from sklearn.model_selection import train_test_split


         X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta
         fig, subaxes = plt.subplots(3, 4, figsize=(15, 10), dpi=50)

         for this_gamma, this_axis in zip([0.01, 1, 5], subaxes):

             for this_C, subplot in zip([0.1, 1, 15, 250], this_axis):
                 title = 'gamma = {:.2f}, C = {:.2f}'.format(this_gamma, this_C)
                 clf = SVC(kernel = 'rbf', gamma = this_gamma,
                           C = this_C).fit(X_train, y_train) #IN EFFECT, WO SVC,FIT
                 plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                           X_test, y_test, title,
                                                           subplot)
                 plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2.2.2 Application of SVMs to a real dataset: unnormalized data

```
In [28]: from sklearn.svm import SVC
         X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,
                                                             random_state = 0)
```

```
clf = SVC(C=10).fit(X_train, y_train)
print('Breast cancer dataset (unnormalized features)')
print('Accuracy of RBF-kernel SVC on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of RBF-kernel SVC on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset (unnormalized features)
Accuracy of RBF-kernel SVC on training set: 1.00
Accuracy of RBF-kernel SVC on test set: 0.63
```

2.2.3 Application of SVMs to a real dataset: normalized data with feature preprocessing using minmax scaling

```
In [29]: from sklearn.preprocessing import MinMaxScaler
         scaler = MinMaxScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         clf = SVC(C=10).fit(X_train_scaled, y_train)
         print('Breast cancer dataset (normalized with MinMax scaling)')
         print('RBF-kernel SVC (with MinMax scaling) training set accuracy: {:.2f}'
               .format(clf.score(X_train_scaled, y_train)))
         print('RBF-kernel SVC (with MinMax scaling) test set accuracy: {:.2f}'
               .format(clf.score(X_test_scaled, y_test)))
```

```
Breast cancer dataset (normalized with MinMax scaling)
RBF-kernel SVC (with MinMax scaling) training set accuracy: 0.98
RBF-kernel SVC (with MinMax scaling) test set accuracy: 0.96
```

2.3 Cross-validation

2.3.1 Example based on k-NN classifier with fruit dataset (2 features)

```
In [30]: from sklearn.model_selection import cross_val_score

         clf = KNeighborsClassifier(n_neighbors = 5)
         X = X_fruits_2d.as_matrix()
         y = y_fruits_2d.as_matrix()
         cv_scores = cross_val_score(clf, X, y) #cross val score is the function

         print('Cross-validation scores (3-fold):', cv_scores)
         print('Mean cross-validation score (3-fold): {:.3f}'
               .format(np.mean(cv_scores)))
```

```
Cross-validation scores (3-fold): [ 0.77  0.74  0.83]
Mean cross-validation score (3-fold): 0.781
```

2.3.2 A note on performing cross-validation for more advanced scenarios.

In some cases (e.g. when feature values have very different ranges), we've seen the need to scale or normalize the training and test sets before use with a classifier. The proper way to do cross-validation when you need to scale the data is *not* to scale the entire dataset with a single transform, since this will indirectly leak information into the training data about the whole dataset, including the test data (see the lecture on data leakage later in the course). Instead, scaling/normalizing must be computed and applied for each cross-validation fold separately. To do this, the easiest way in scikit-learn is to use *pipelines*. While these are beyond the scope of this course, further information is available in the scikit-learn documentation here:

http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

or the Pipeline section in the recommended textbook: Introduction to Machine Learning with Python by Andreas C. Müller and Sarah Guido (O'Reilly Media).

2.4 Validation curve example

3 I forgot very quickly, this may be imp!

```
In [31]: from sklearn.svm import SVC
         from sklearn.model_selection import validation_curve

         param_range = np.logspace(-3, 3, 4)
         train_scores, test_scores = validation_curve(SVC(), X, y,
                                                      param_name='gamma',
                                                      param_range=param_range, cv=3)
```

3.0.1 4x3 (4 levels of gamma x 3 fits per level) arrays of the scores for the training and test sets.

```
In [32]: print(train_scores)

[[ 0.49  0.42  0.41]
 [ 0.84  0.72  0.76]
 [ 0.92  0.9   0.93]
 [ 1.    1.    0.98]]


In [33]: print(test_scores)

[[ 0.45  0.32  0.33]
 [ 0.82  0.68  0.61]
 [ 0.41  0.84  0.67]
 [ 0.36  0.21  0.39]]
```

```
In [34]: # This code based on scikit-learn validation_plot example
         #  See:  http://scikit-learn.org/stable/auto_examples/model_selection/plot
         plt.figure()

         train_scores_mean = np.mean(train_scores, axis=1)
         train_scores_std = np.std(train_scores, axis=1)
         test_scores_mean = np.mean(test_scores, axis=1)
         test_scores_std = np.std(test_scores, axis=1)

         plt.title('Validation Curve with SVM')
         plt.xlabel('$\gamma$ (gamma)')
         plt.ylabel('Score')
         plt.ylim(0.0, 1.1)
         lw = 2

         plt.semilogx(param_range, train_scores_mean, label='Training score',
                      color='darkorange', lw=lw)

         plt.fill_between(param_range, train_scores_mean - train_scores_std,
                          train_scores_mean + train_scores_std, alpha=0.2,
                          color='darkorange', lw=lw)

         plt.semilogx(param_range, test_scores_mean, label='Cross-validation score'
                      color='navy', lw=lw)

         plt.fill_between(param_range, test_scores_mean - test_scores_std,
                          test_scores_mean + test_scores_std, alpha=0.2,
                          color='navy', lw=lw)

         plt.legend(loc='best')
         plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py:524: RuntimeWarning: Mo
  max_open_warning, RuntimeWarning)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

3.1 Decision Trees

```
In [1]: from sklearn.datasets import load_iris
        from sklearn.tree import DecisionTreeClassifier
        from adspy_shared_utilities import plot_decision_tree
        from sklearn.model_selection import train_test_split
```

```
iris = load_iris()
print(iris.data,'\n\n',iris.target)

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
clf = DecisionTreeClassifier().fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]
 [ 5.4  3.7  1.5  0.2]
 [ 4.8  3.4  1.6  0.2]
 [ 4.8  3.   1.4  0.1]
 [ 4.3  3.   1.1  0.1]
 [ 5.8  4.   1.2  0.2]
 [ 5.7  4.4  1.5  0.4]
 [ 5.4  3.9  1.3  0.4]
 [ 5.1  3.5  1.4  0.3]
 [ 5.7  3.8  1.7  0.3]
 [ 5.1  3.8  1.5  0.3]
 [ 5.4  3.4  1.7  0.2]
 [ 5.1  3.7  1.5  0.4]
 [ 4.6  3.6  1.   0.2]
 [ 5.1  3.3  1.7  0.5]
 [ 4.8  3.4  1.9  0.2]
 [ 5.   3.   1.6  0.2]
 [ 5.   3.4  1.6  0.4]
 [ 5.2  3.5  1.5  0.2]
 [ 5.2  3.4  1.4  0.2]
 [ 4.7  3.2  1.6  0.2]
 [ 4.8  3.1  1.6  0.2]
 [ 5.4  3.4  1.5  0.4]
 [ 5.2  4.1  1.5  0.1]
 [ 5.5  4.2  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]
 [ 5.   3.2  1.2  0.2]
```

```
[ 5.5  3.5  1.3  0.2]
[ 4.9  3.1  1.5  0.1]
[ 4.4  3.   1.3  0.2]
[ 5.1  3.4  1.5  0.2]
[ 5.   3.5  1.3  0.3]
[ 4.5  2.3  1.3  0.3]
[ 4.4  3.2  1.3  0.2]
[ 5.   3.5  1.6  0.6]
[ 5.1  3.8  1.9  0.4]
[ 4.8  3.   1.4  0.3]
[ 5.1  3.8  1.6  0.2]
[ 4.6  3.2  1.4  0.2]
[ 5.3  3.7  1.5  0.2]
[ 5.   3.3  1.4  0.2]
[ 7.   3.2  4.7  1.4]
[ 6.4  3.2  4.5  1.5]
[ 6.9  3.1  4.9  1.5]
[ 5.5  2.3  4.   1.3]
[ 6.5  2.8  4.6  1.5]
[ 5.7  2.8  4.5  1.3]
[ 6.3  3.3  4.7  1.6]
[ 4.9  2.4  3.3  1. ]
[ 6.6  2.9  4.6  1.3]
[ 5.2  2.7  3.9  1.4]
[ 5.   2.   3.5  1. ]
[ 5.9  3.   4.2  1.5]
[ 6.   2.2  4.   1. ]
[ 6.1  2.9  4.7  1.4]
[ 5.6  2.9  3.6  1.3]
[ 6.7  3.1  4.4  1.4]
[ 5.6  3.   4.5  1.5]
[ 5.8  2.7  4.1  1. ]
[ 6.2  2.2  4.5  1.5]
[ 5.6  2.5  3.9  1.1]
[ 5.9  3.2  4.8  1.8]
[ 6.1  2.8  4.   1.3]
[ 6.3  2.5  4.9  1.5]
[ 6.1  2.8  4.7  1.2]
[ 6.4  2.9  4.3  1.3]
[ 6.6  3.   4.4  1.4]
[ 6.8  2.8  4.8  1.4]
[ 6.7  3.   5.   1.7]
[ 6.   2.9  4.5  1.5]
[ 5.7  2.6  3.5  1. ]
[ 5.5  2.4  3.8  1.1]
[ 5.5  2.4  3.7  1. ]
[ 5.8  2.7  3.9  1.2]
[ 6.   2.7  5.1  1.6]
```

```
[ 5.4  3.   4.5  1.5]
[ 6.   3.4  4.5  1.6]
[ 6.7  3.1  4.7  1.5]
[ 6.3  2.3  4.4  1.3]
[ 5.6  3.   4.1  1.3]
[ 5.5  2.5  4.   1.3]
[ 5.5  2.6  4.4  1.2]
[ 6.1  3.   4.6  1.4]
[ 5.8  2.6  4.   1.2]
[ 5.   2.3  3.3  1. ]
[ 5.6  2.7  4.2  1.3]
[ 5.7  3.   4.2  1.2]
[ 5.7  2.9  4.2  1.3]
[ 6.2  2.9  4.3  1.3]
[ 5.1  2.5  3.   1.1]
[ 5.7  2.8  4.1  1.3]
[ 6.3  3.3  6.   2.5]
[ 5.8  2.7  5.1  1.9]
[ 7.1  3.   5.9  2.1]
[ 6.3  2.9  5.6  1.8]
[ 6.5  3.   5.8  2.2]
[ 7.6  3.   6.6  2.1]
[ 4.9  2.5  4.5  1.7]
[ 7.3  2.9  6.3  1.8]
[ 6.7  2.5  5.8  1.8]
[ 7.2  3.6  6.1  2.5]
[ 6.5  3.2  5.1  2. ]
[ 6.4  2.7  5.3  1.9]
[ 6.8  3.   5.5  2.1]
[ 5.7  2.5  5.   2. ]
[ 5.8  2.8  5.1  2.4]
[ 6.4  3.2  5.3  2.3]
[ 6.5  3.   5.5  1.8]
[ 7.7  3.8  6.7  2.2]
[ 7.7  2.6  6.9  2.3]
[ 6.   2.2  5.   1.5]
[ 6.9  3.2  5.7  2.3]
[ 5.6  2.8  4.9  2. ]
[ 7.7  2.8  6.7  2. ]
[ 6.3  2.7  4.9  1.8]
[ 6.7  3.3  5.7  2.1]
[ 7.2  3.2  6.   1.8]
[ 6.2  2.8  4.8  1.8]
[ 6.1  3.   4.9  1.8]
[ 6.4  2.8  5.6  2.1]
[ 7.2  3.   5.8  1.6]
[ 7.4  2.8  6.1  1.9]
[ 7.9  3.8  6.4  2. ]
```

```
 [ 6.4  2.8  5.6  2.2]
 [ 6.3  2.8  5.1  1.5]
 [ 6.1  2.6  5.6  1.4]
 [ 7.7  3.   6.1  2.3]
 [ 6.3  3.4  5.6  2.4]
 [ 6.4  3.1  5.5  1.8]
 [ 6.   3.   4.8  1.8]
 [ 6.9  3.1  5.4  2.1]
 [ 6.7  3.1  5.6  2.4]
 [ 6.9  3.1  5.1  2.3]
 [ 5.8  2.7  5.1  1.9]
 [ 6.8  3.2  5.9  2.3]
 [ 6.7  3.3  5.7  2.5]
 [ 6.7  3.   5.2  2.3]
 [ 6.3  2.5  5.   1.9]
 [ 6.5  3.   5.2  2. ]
 [ 6.2  3.4  5.4  2.3]
 [ 5.9  3.   5.1  1.8]]

 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.97
```

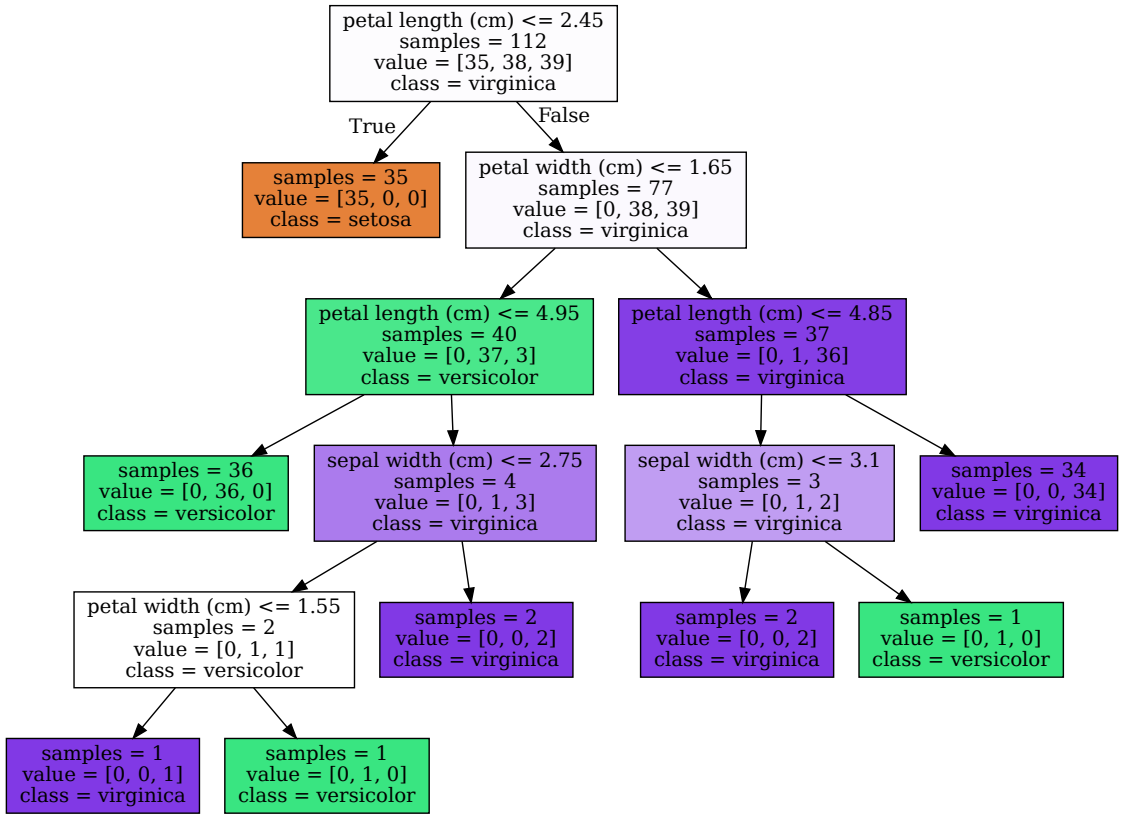
Setting max decision tree depth to help avoid overfitting

```
In [2]: clf2 = DecisionTreeClassifier(max_depth = 3).fit(X_train, y_train)

        print('Accuracy of Decision Tree classifier on training set: {:.2f}'
             .format(clf2.score(X_train, y_train)))
        print('Accuracy of Decision Tree classifier on test set: {:.2f}'
             .format(clf2.score(X_test, y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 0.98
Accuracy of Decision Tree classifier on test set: 0.97
```

Visualizing decision trees

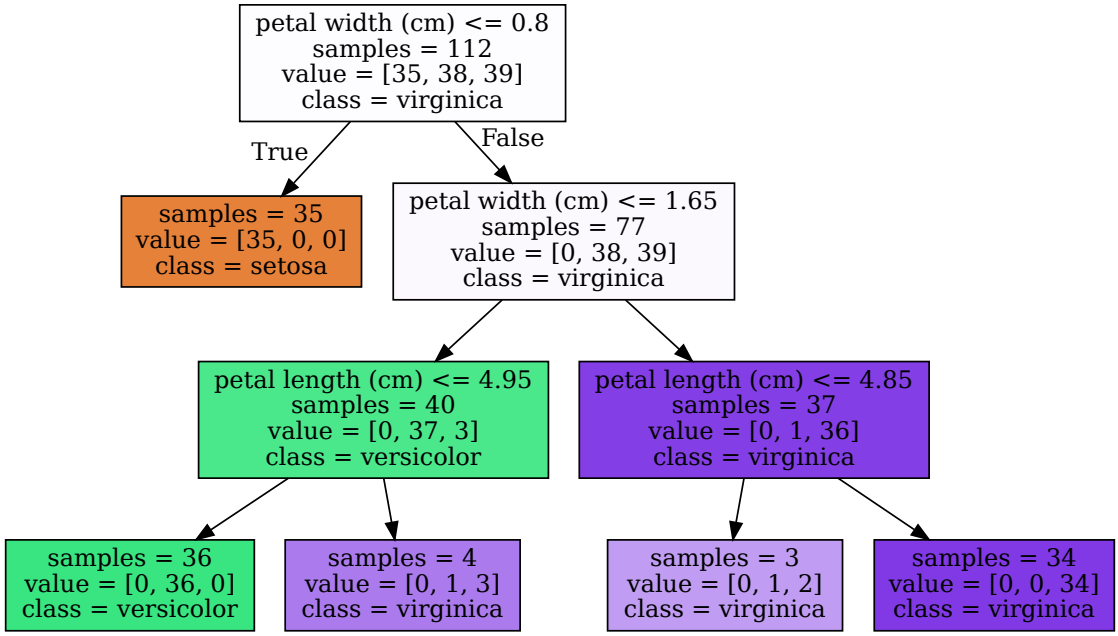
```
In [3]: plot_decision_tree(clf, iris.feature_names, iris.target_names)

Out[3]:
```



Pre-pruned version (max_depth = 3)

```
In [4]: plot_decision_tree(clf2, iris.feature_names, iris.target_names)
```

Out[4]:



Feature importance

```
In [7]: from adspy_shared_utilities import plot_feature_importances

        plt.figure(figsize=(10,4), dpi=80)
        plot_feature_importances(clf, iris.feature_names)
        plt.show()

        print('Feature importances: {}'.format(clf.feature_importances_))
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Feature importances: [ 0.    0.02  0.55  0.43]
```

4 ye kya kiya and kyu kiya samjha hi nahi

```
In [8]: from sklearn.tree import DecisionTreeClassifier
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo

        X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
        fig, subaxes = plt.subplots(6, 1, figsize=(6, 32))
```

```
        pair_list = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]
        tree_max_depth = 4

        for pair, axis in zip(pair_list, subaxes):
            X = X_train[:, pair]
            y = y_train

            clf = DecisionTreeClassifier(max_depth=tree_max_depth).fit(X, y)
            title = 'Decision Tree, max_depth = {:d}'.format(tree_max_depth)
            plot_class_regions_for_classifier_subplot(clf, X, y, None,
                                                      None, title, axis,
                                                      iris.target_names)

            axis.set_xlabel(iris.feature_names[pair[0]])
            axis.set_ylabel(iris.feature_names[pair[1]])

        plt.tight_layout()
        plt.show()
```

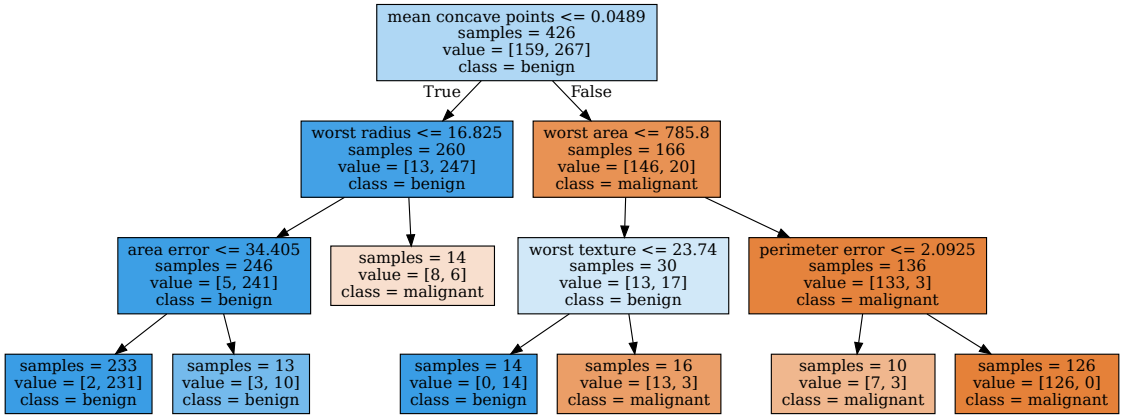
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Decision Trees on a real-world dataset

```
In [13]: from sklearn.tree import DecisionTreeClassifier
         from adspy_shared_utilities import plot_decision_tree
         from adspy_shared_utilities import plot_feature_importances

         X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ra

         clf = DecisionTreeClassifier(max_depth = 3, min_samples_leaf = 10,
                                      random_state = 0).fit(X_train, y_train)

         plot_decision_tree(clf, cancer.feature_names, cancer.target_names)
```

```
Out[13]:
```




```
In [14]: print('Breast cancer dataset: decision tree')
         print('Accuracy of DT classifier on training set: {:.2f}'
               .format(clf.score(X_train, y_train)))
         print('Accuracy of DT classifier on test set: {:.2f}'
               .format(clf.score(X_test, y_test)))

         #plt.figure(figsize=(10,6),dpi=80)
         #plot_feature_importances(clf, cancer.feature_names)
         #plt.tight_layout()

         #plt.show()
```

```
Breast cancer dataset: decision tree
Accuracy of DT classifier on training set: 0.96
Accuracy of DT classifier on test set: 0.94
```

```
In [ ]:
```

Applied Machine Learning

Model Evaluation & Selection

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Learning objectives

- Understand why accuracy only gives a partial picture of a classifier's performance.

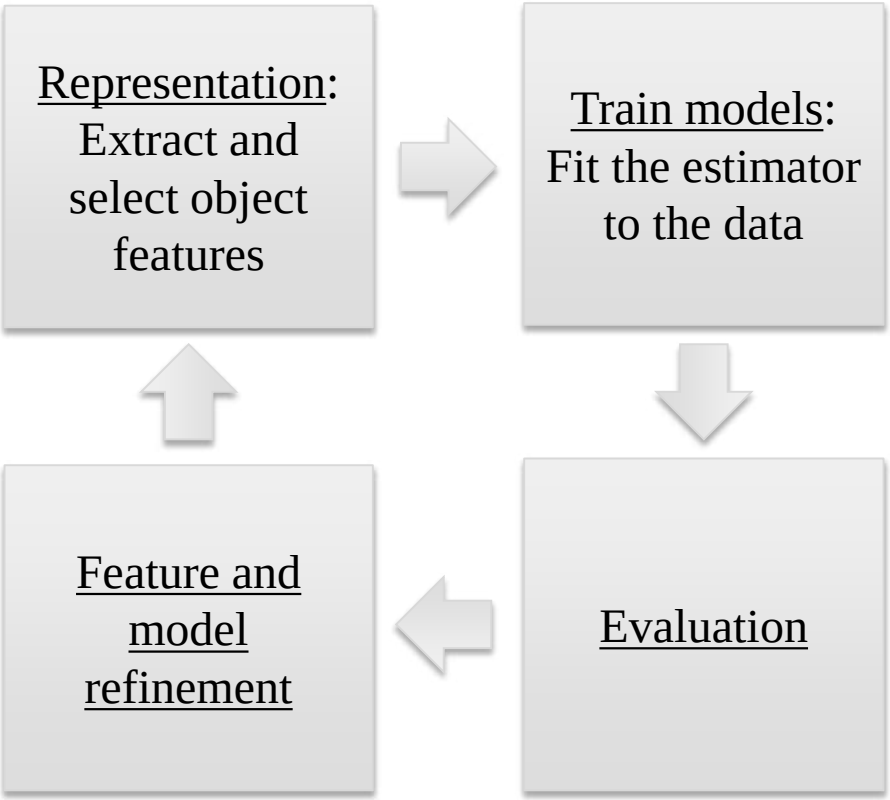
- Understand the motivation and definition of important evaluation metrics in machine learning.

Learning objectives

- Learn how to use a variety of evaluation metrics to evaluate supervised machine learning models.

- Learn about choosing the right metric for selecting between models or for doing parameter tuning.

Represent / Train / Evaluate / Refine Cycle



Evaluation

- Different applications have very different goals
- Accuracy is widely used, but many others are possible, e.g.:
  - User satisfaction (Web search)
  - Amount of revenue (e-commerce)
  - Increase in patient survival rates (medical)

Evaluation

- It's very important to choose evaluation methods that match the goal of your application.
- Compute your selected evaluation metric for multiple different models.
- Then select the model with 'best' value of evaluation metric.

Accuracy with imbalanced classes

- Suppose you have two classes:
  - Relevant (R): the positive class
  - Not_Relevant (N): the negative class
- Out of 1000 randomly selected items, on average
  - One item is relevant and has an R label
  - The rest of the items (999 of them) are not relevant and labelled N.
- Recall that:

$$\text{Accuracy} = \frac{\#\text{correct predictions}}{\#\text{total instances}}$$

Accuracy with imbalanced classes

- You build a classifier to predict relevant items, and see that its accuracy on a test set is **99.9**%.
- Wow! Amazingly good, right?
- For comparison, suppose we had a "dummy" classifier that didn't look at the features at all, and always just blindly predicted the most frequent class (i.e. the negative N class).

Accuracy with imbalanced classes

- Assuming a test set of 1000 instances, what would this dummy classifier's accuracy be?
- Answer:

$$\text{Accuracy}_{\text{DUMMY}} = 999 / 1000 = 99.9\%$$

Dummy classifiers completely ignore the input data!

- Dummy classifiers serve as a sanity check on your classifier's performance.
- They provide a null metric (e.g. null accuracy) baseline.
- Dummy classifiers should not be used for real problems.

Dummy classifiers completely ignore the input data!

- Some commonly-used settings for the `strategy` parameter for DummyClassifier in scikit-learn:
  - ***most_frequent*** : predicts the most frequent label in the training set.
  - ***stratified*** : random predictions based on training set class distribution.
  - ***uniform*** : generates predictions uniformly at random.
  - ***constant*** : always predicts a constant label provided by the user.
    - A major motivation of this method is F1-scoring, when the positive class is in the minority.

What if my classifier accuracy is close to the null accuracy baseline?

This could be a sign of:

- Ineffective, erroneous or missing features
- Poor choice of kernel or hyperparameter
- Large class imbalance

Dummy regressors

`strategy` parameter options:

- *mean* : predicts the mean of the training targets.
- *median* : predicts the median of the training targets.
- *quantile* : predicts a user-provided quantile of the training targets.
- *constant* : predicts a constant user-provided value.

Binary prediction outcomes

True negative	TN	FP
True positive	FN	TP
	Predicted negative	**Predicted** positive

Label 1 = positive class
(class of interest)

Label 0 = negative class
(everything else)

TP = true positive
FP = false positive (Type I error)
TN = true negative
FN = false negative (Type II error)

Confusion matrix for binary prediction task

True negative	TN = 356	FP = 51
True positive	FN = 38	TP = 5
	Predicted negative	Predicted positive

N = 450

- Every test instance is in exactly one box (integer counts).
- Breaks down classifier results by error type.
- Thus, provides more information than simple accuracy.
- Helps you choose an evaluation metric that matches project goals.
- Not a single number like accuracy.. but there are many possible metrics that can be derived from the confusion matrix.

Applied Machine Learning

Confusion Matrices & Basic Evaluation Metrics

Kevyn Collins-Thompson

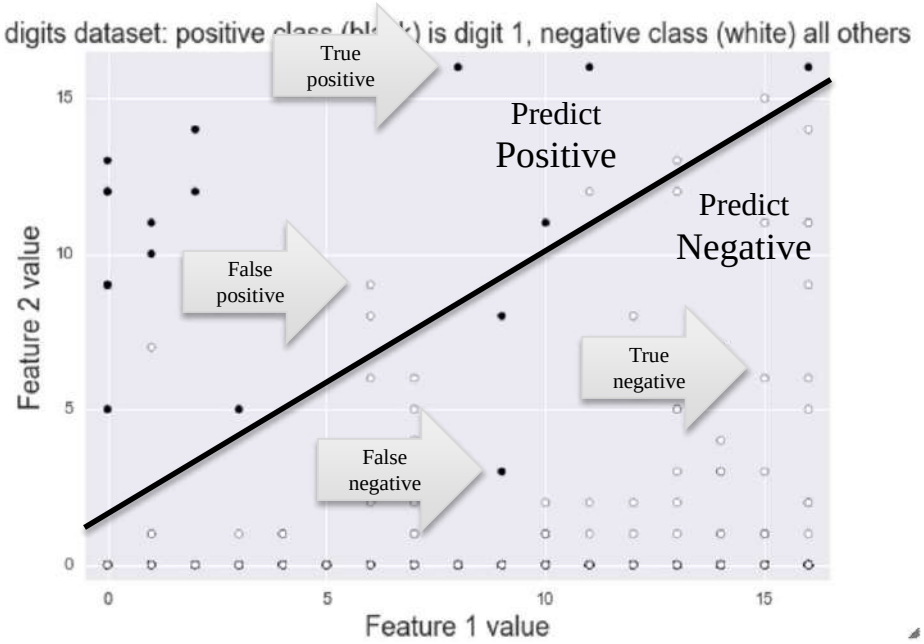
Associate Professor of Information & Computer Science
University of Michigan

Confusion Matrix for Binary Prediction Task

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	N = 450

Always look at the confusion matrix for your classifier.

Visualization of Different Error Types



TN = 429	FP = 6
FN = 2	TP = 13

Accuracy: for what fraction of all instances is the classifier's prediction correct (for either positive or negative class)?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\text{Accuracy} = \frac{TN+TP}{TN+TP+FN+FP}$$

$$= \frac{400+26}{400+26+17+7}$$

$$= 0.95$$

Classification error (1 – Accuracy): for what fraction of all instances is the classifier's prediction incorrect?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\text{ClassificationError} = \frac{FP + FN}{TN + TP + FN + FP}$$

$$= \frac{7+17}{400+26+17+7}$$

$$= 0.060$$

Recall, or True Positive Rate (TPR): what fraction of all positive instances does the classifier correctly identify as positive?

	Predicted negative	Predicted positive	
True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			N = 450

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$= \frac{26}{26+17}$$

$$= 0.60$$

Recall is also known as:

- True Positive Rate (TPR)
- Sensitivity
- Probability of detection

Precision: what fraction of positive predictions are correct?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	N = 450

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$= \frac{26}{26+7}$$

$$= 0.79$$

False positive rate (FPR): what fraction of all negative instances does the classifier <u>incorrectly</u> identify as positive?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$FPR = \frac{FP}{TN+FP}$$

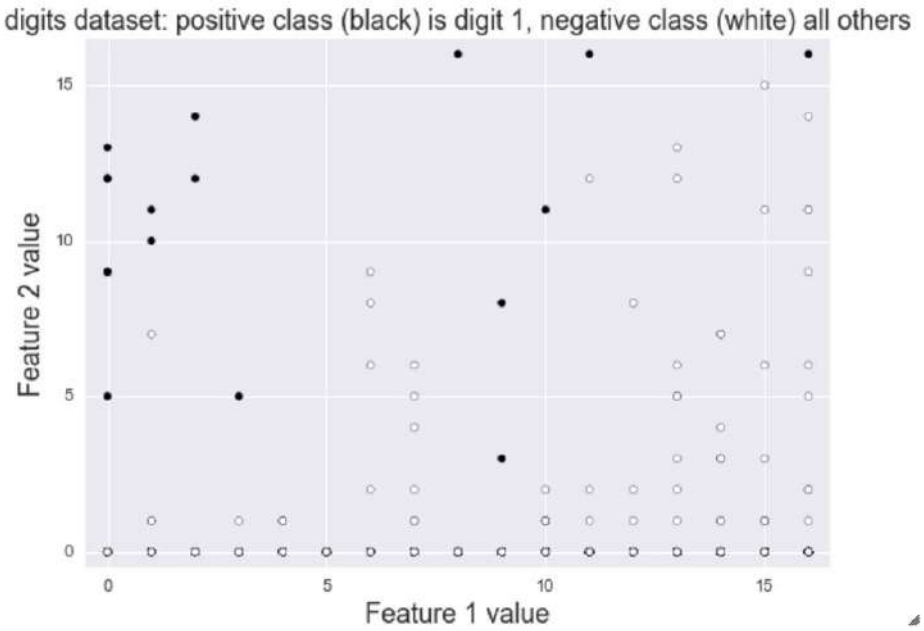
$$= \frac{7}{400+7}$$

$$= 0.02$$

False Positive Rate is also known as:

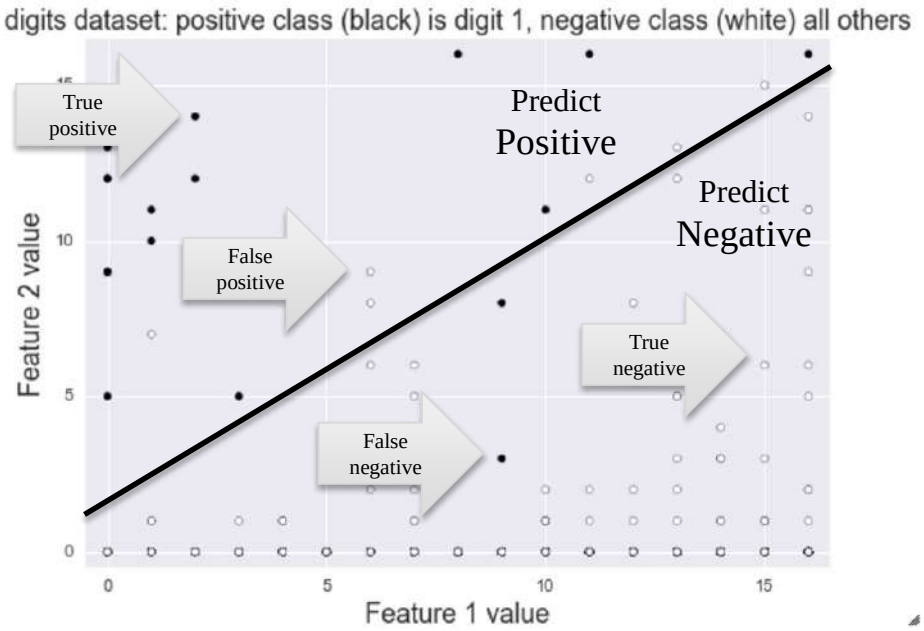
- Specificity

A Graphical Illustration of Precision & Recall



TN =	FP =
FN =	TP =

The Precision-Recall Tradeoff

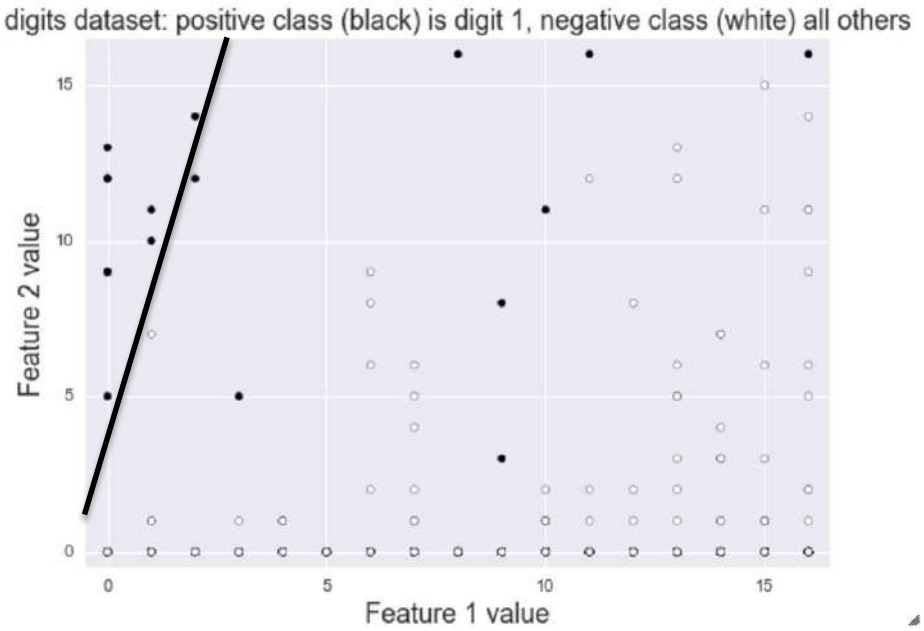


TN = 429	FP = 6
FN = 2	TP = 13

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{13}{19} = 0.68$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{13}{15} = 0.87$$

High Precision, Lower Recall

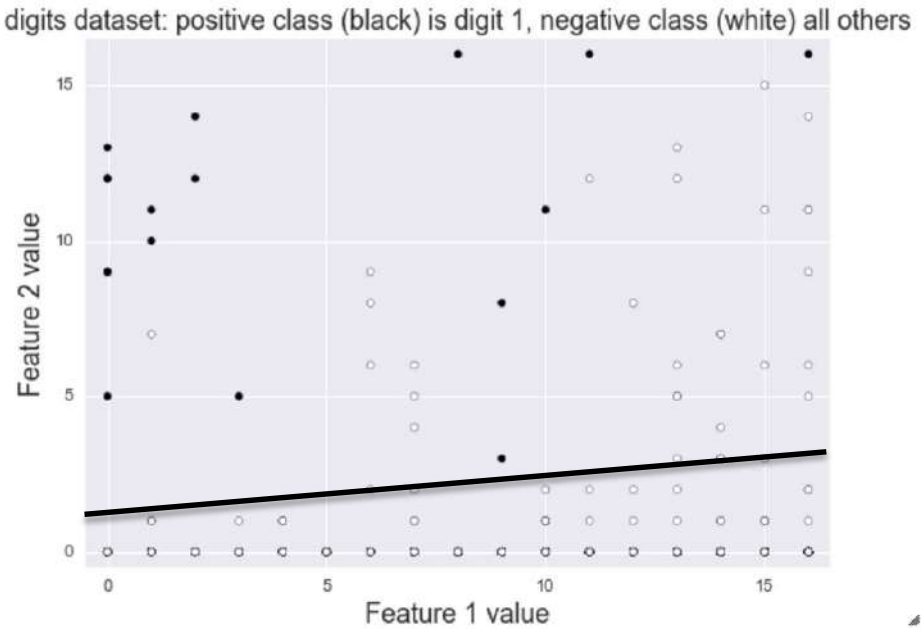


TN = 435	FP = 0
FN = 8	TP = 7

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{7}{7} = 1.00$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{7}{15} = 0.47$$

Low Precision, High Recall



TN = 408	FP = 27
FN = 0	TP = 15

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{15}{42} = 0.36$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{15}{15} = 1.00$$

There is often a tradeoff between precision and recall

- Recall-oriented machine learning tasks:
  - Search and information extraction in legal discovery
  - Tumor detection
  - Often paired with a human expert to filter out false positives
- Precision-oriented machine learning tasks:
  - Search engine ranking, query suggestion
  - Document classification
  - Many customer-facing tasks (users remember failures!)

F1-score: combining precision & recall into a single number

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

F-score: generalizes F1-score for combining precision & recall into a single number

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

$$F_\beta = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} = \frac{(1+\beta^2) \cdot TP}{(1+\beta^2) \cdot TP + \beta \cdot FN + FP}$$

β allows adjustment of the metric to control the emphasis on recall vs precision:

- **Precision-oriented users: $\boldsymbol{\beta}$ = 0.5 (false positives hurt performance more than false negatives)**
- **Recall-oriented users: $\boldsymbol{\beta}$ = 2 (false negatives hurt performance more than false positives)**

Applied Machine Learning

Classifier Decision Functions

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Decision Functions (decision_function)

- Each classifier score value per test point indicates how confidently the classifier predicts the positive class (large-magnitude positive values) or the negative class (large-magnitude negative values).
- Choosing a fixed decision threshold gives a classification rule.
- By sweeping the decision threshold through the entire range of possible score values, we get a series of classification outcomes that form a curve.

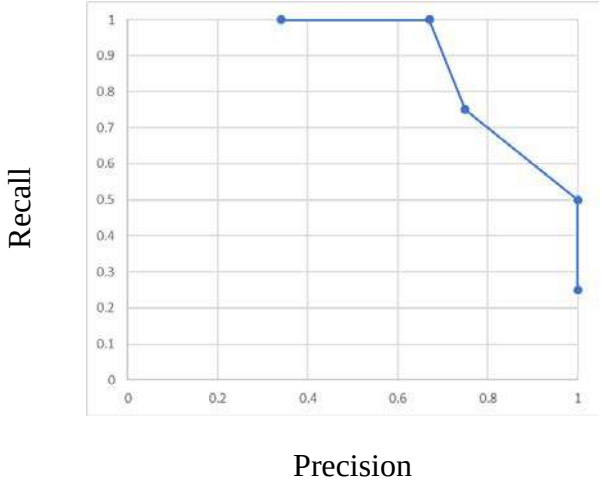
Predicted Probability of Class Membership (predict_proba)

- Typical rule: choose most likely class
  - e.g class 1 if threshold > 0.50.
- Adjusting threshold affects predictions of classifier.
- Higher threshold results in a more conservative classifier
  - e.g. only predict Class 1 if estimated probability of class 1 is above 70%
  - This increases precision. Doesn't predict class 1 as often, but when it does, it gets high proportion of class 1 instances correct.
- Not all models provide realistic probability estimates

Varying the Decision Threshold

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

Classifier score threshold	Precision	Recall
-20	4/12=0.34	4/4=1.00
-10	4/6=0.67	4/4=1.00
0	3/4=0.75	3/4=0.75
10	2/2=1.0	2/4=0.50
20	1/1=1.0	1/4 = 0.25



Applied Machine Learning

Precision-Recall and ROC curves

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Precision-Recall Curves

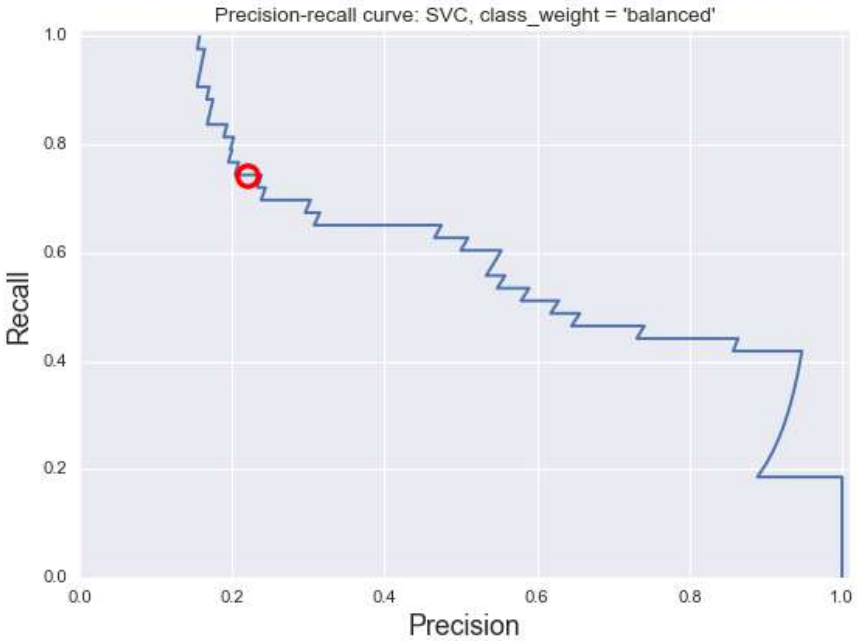
X-axis: Precision
Y-axis: Recall

Top right corner:

- **The "ideal" point**
- **Precision = 1.0**
- **Recall = 1.0**

"Steepness" of P-R curves is important:

- **Maximize precision**
- **while maximizing recall**



ROC Curves

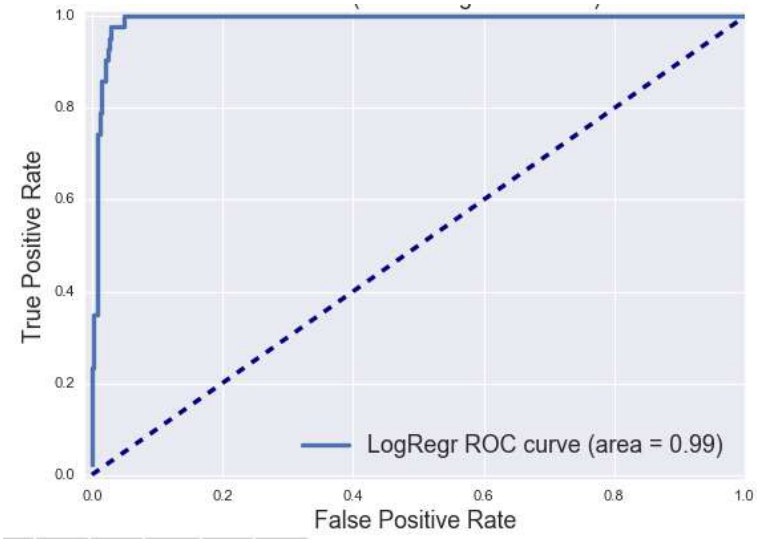
X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:

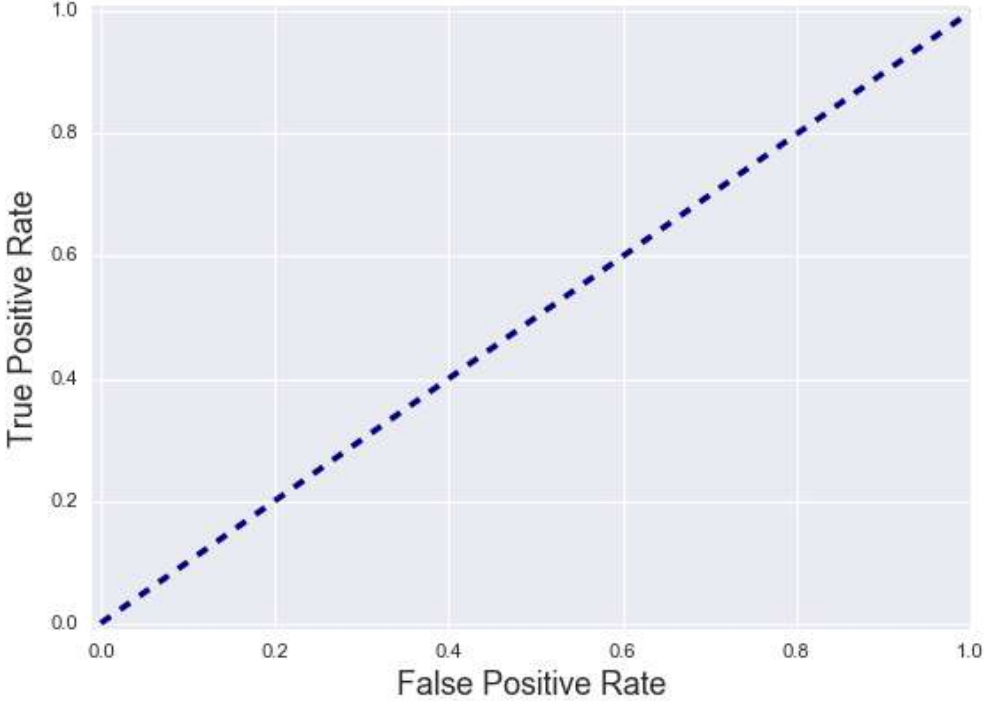
- **The "ideal" point**
- **False positive rate of zero**
- **True positive rate of one**

"Steepness" of ROC curves is important:

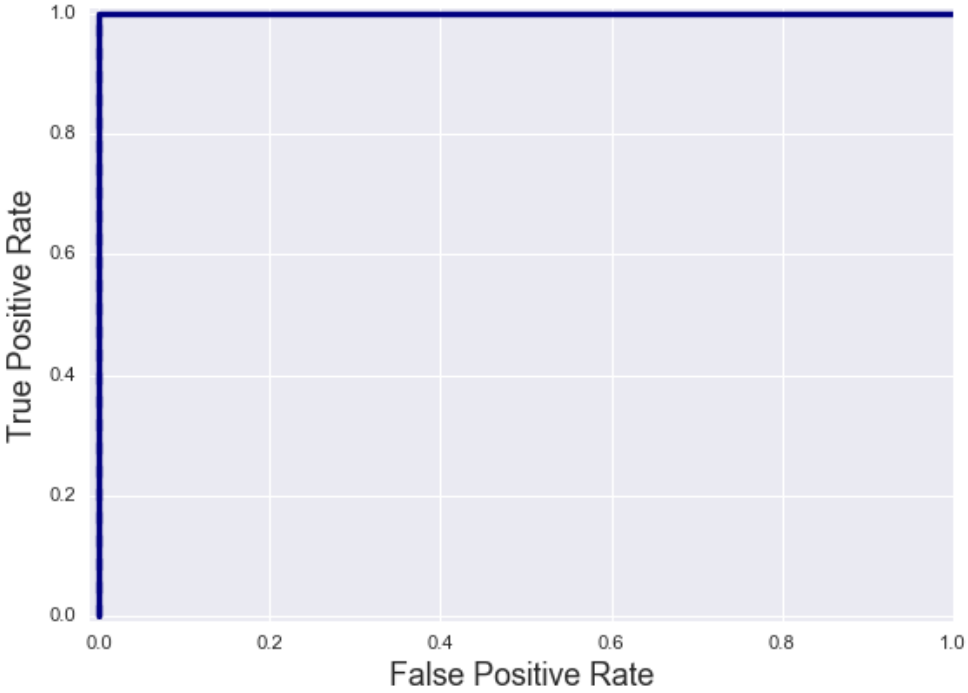
- **Maximize the true positive rate**
- **while minimizing the false positive rate**



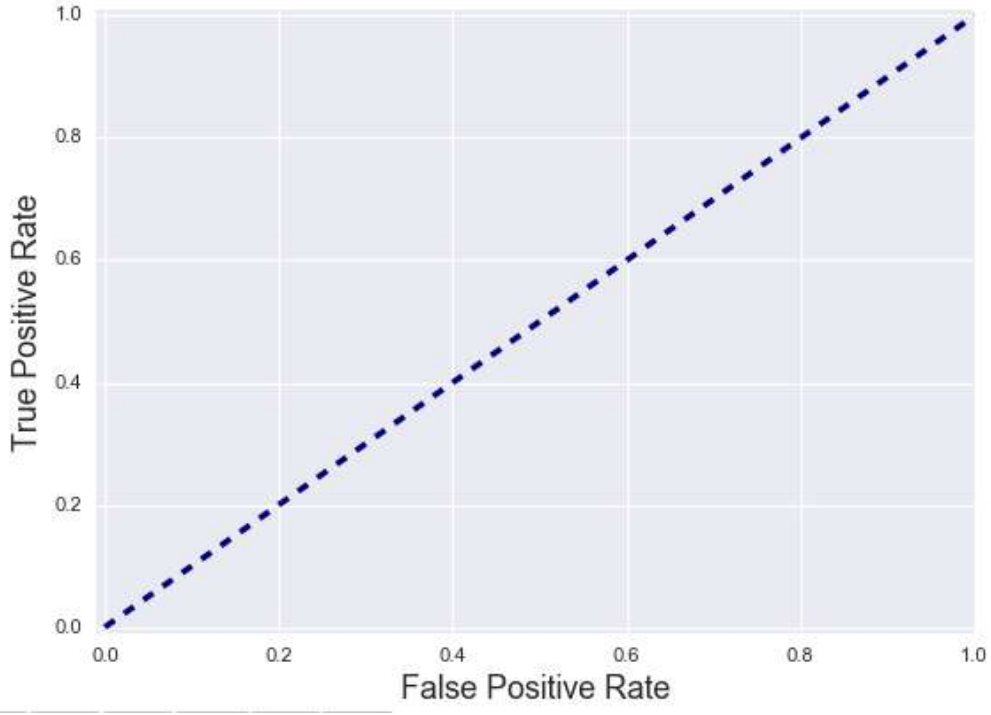
ROC curve examples: random guessing



ROC curve examples: perfect classifier

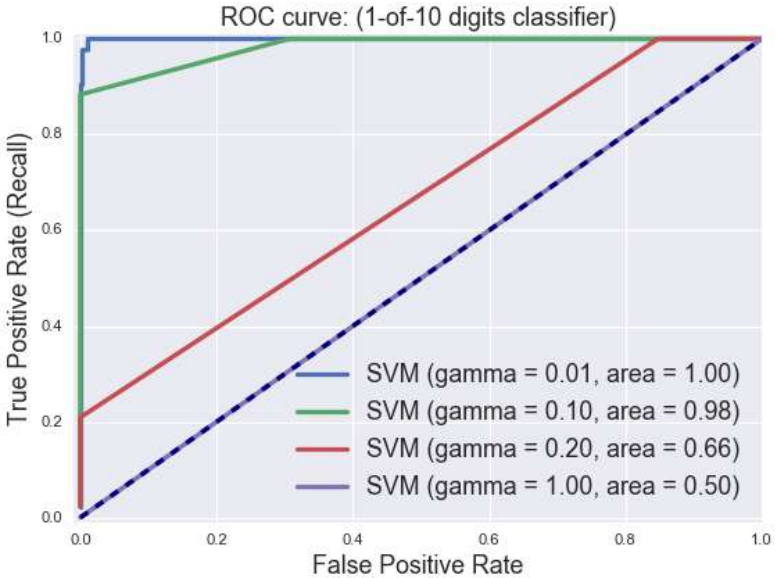


ROC curve examples: bad, okay,



Summarizing an ROC curve in one number: Area Under the Curve (AUC)

- AUC = 0 (worst) AUC = 1 (best)
- AUC can be interpreted as:
  1. The total area under the ROC curve.
  2. The probability that the classifier will assign a higher score to a randomly chosen positive example than to a randomly chosen negative example.
- Advantages:
  - Gives a single number for easy comparison.
  - Does not require specifying a decision threshold.
- Drawbacks:
  - As with other single-number metrics, AUC loses information, e.g. about tradeoffs and the shape of the ROC curve.
  - This may be a factor to consider when e.g. wanting to compare the performance of classifiers with overlapping ROC curves.



Applied Machine Learning

Multi-Class Evaluation

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Multi-Class Evaluation

- Multi-class evaluation is an extension of the binary case.
  - A collection of true vs predicted binary outcomes, one per class
  - Confusion matrices are especially useful
  - Classification report
- Overall evaluation metrics are averages across classes
  - But there are different ways to average multi-class results: we will cover these shortly.
  - The support (number of instances) for each class is important to consider, e.g. in case of imbalanced classes
- Multi-label classification: each instance can have multiple labels (not covered here)

Multi-Class Confusion Matrix

True Digit

	0	1	2	3	4	5	6	7	8	9
0	37	0	0	0	0	0	0	0	0	0
1	0	42	0	0	0	0	0	0	1	0
2	0	0	44	0	0	0	0	0	0	0
3	0	0	0	43	0	0	0	0	1	1
4	0	0	0	0	38	0	0	0	0	0
5	0	0	0	0	0	47	0	0	0	1
6	0	1	0	0	0	0	51	0	0	0
7	0	0	0	0	1	0	0	47	0	0
8	0	3	1	0	0	0	0	0	44	0
9	0	0	0	1	0	1	0	0	1	44

Predicted Digit

Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

Macro-average:

- Each class has equal weight.

1. Compute metric within each class
2. Average resulting metrics across classes

Class	Precision
orange	1/5 = 0.20
lemon	1/2 = 0.50
apple	2/2 = 1.00

Macro-average precision:
(0.20 + 0.50 + 1.00) / 3 = **0.57**

Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

<u>Micro-average</u>:

- Each <u>instance</u> has equal weight.
- Largest classes have most influence

1. Aggregate outcomes across all classes
2. Compute metric with aggregate outcomes

Micro-average precision:
4 / 9 = **0.44**

Macro-Average vs Micro-Average

- If the classes have about the same number of instances, macro- and micro-average will be about the same.
- If some classes are much larger (more instances) than others, and you want to:
  - Weight your metric toward the largest ones, use micro-averaging.
  - Weight your metric toward the smallest ones, use macro-averaging.
- If the micro-average is much lower than the macro-average then examine the larger classes for poor metric performance.
- If the macro-average is much lower than the micro-average then examine the smaller classes for poor metric performance.

Multi-class Evaluation Metrics via the "Average" Parameter for a Scoring Function

- Micro: Metric on aggregated instances
- Macro: Mean per-class metric, classes have equal weight
- Weighted: Mean per-class metric, weighted by support
- Samples: for multi-label problems only

Applied Machine Learning

Regression Evaluation

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

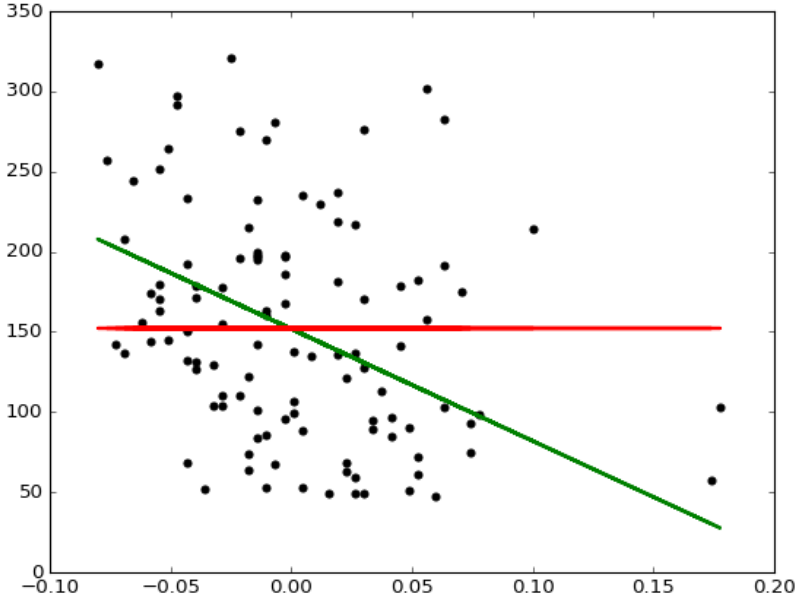
Regression metrics

- Typically r2_score is enough
  - Reminder: computes how well future instances will be predicted
  - Best possible score is 1.0
  - Constant prediction score is 0.0
- Alternative metrics include:
  - mean_absolute_error (absolute difference of target & predicted values)
  - mean_squared_error (squared difference of target & predicted values)
  - median_absolute_error (robust to outliers)

Dummy regressors

As in classification, comparison to a 'dummy' prediction model that uses a fixed rule can be useful.

For this, scikit.learn provides dummy regressors.

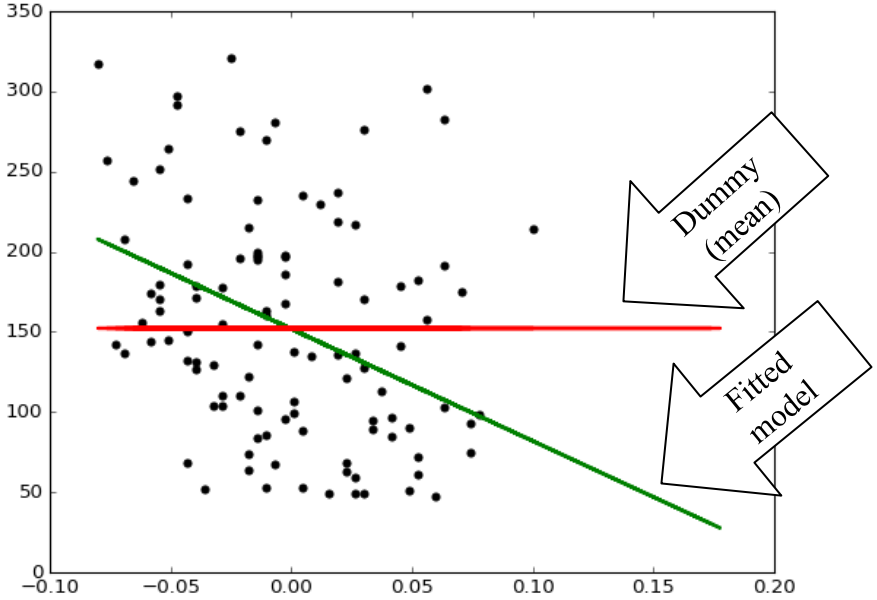


Dummy regressors

As in classification, comparison to a 'dummy' prediction model that uses a fixed rule can be useful.

For this, scikit.learn provides dummy regressors

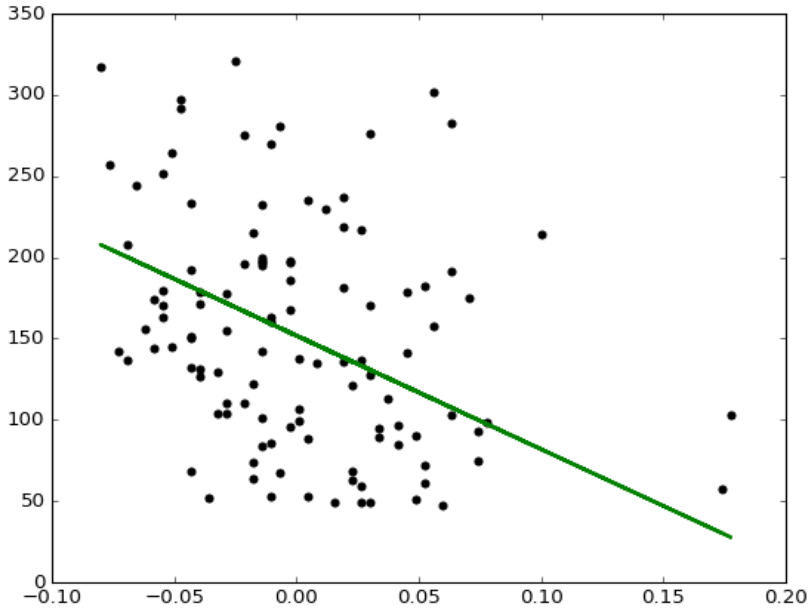
```
Linear model, coefficients: [-698.80206267]
Mean squared error (dummy): 4965.13
Mean squared error (linear model): 4646.74
r2_score (dummy): -0.00
r2_score (linear model): 0.06
```



Dummy regressors

The DummyRegressor class implements four simple baseline rules for regression, using the `strategy` parameter:

- `mean` predicts the mean of the training target values.
- `median` predicts the median of the training target values.
- `quantile` predicts a user-provided quantile of the training target values (e.g. value at the 75th percentile)
- `constant` predicts a custom constant value provided by the user.



Applied Machine Learning

Optimizing Classifiers for Different Evaluation Metrics

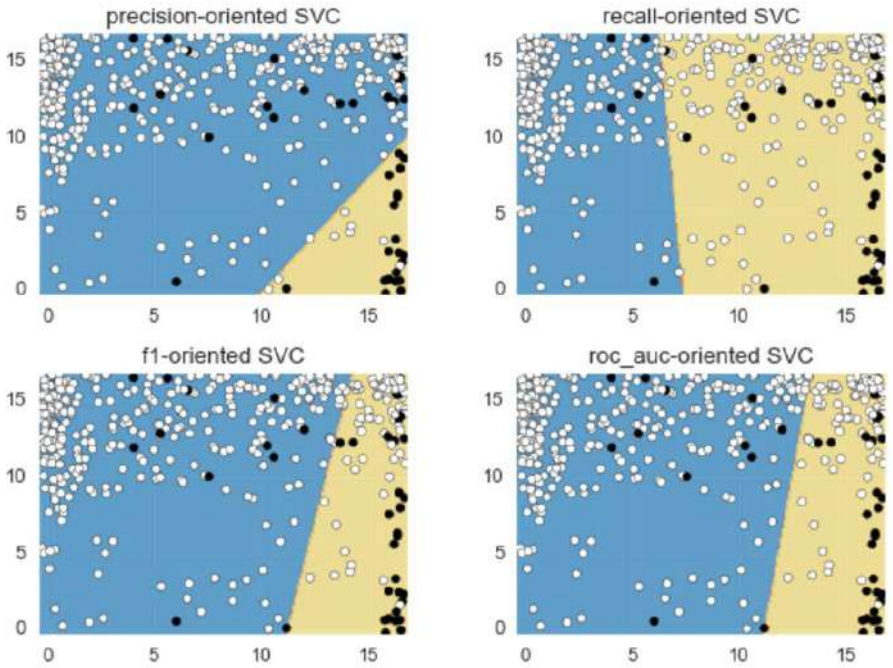
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

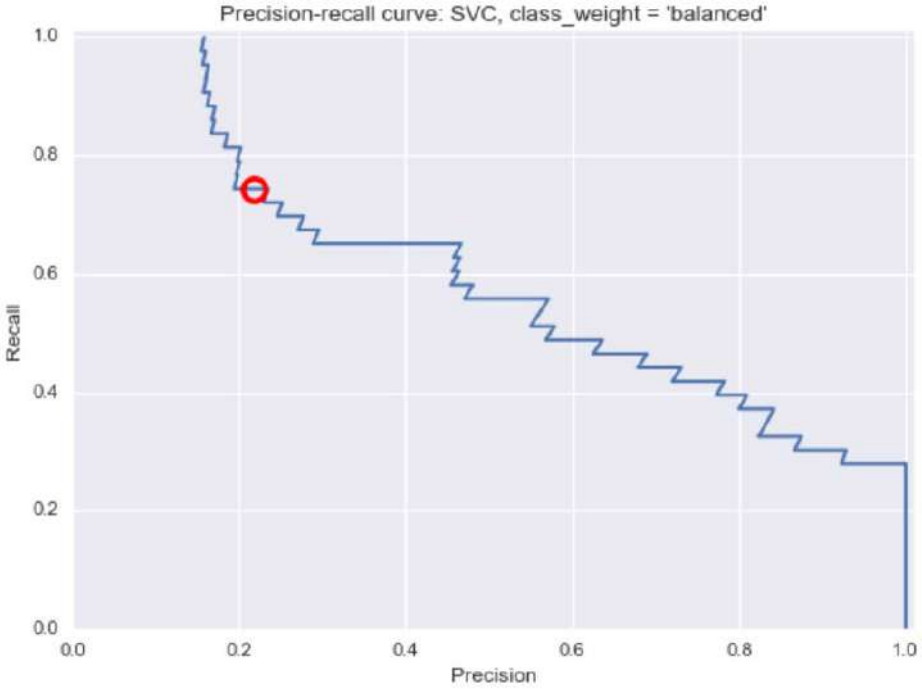
Model Selection Using Evaluation Metrics

- Train/test on same data
  - Single metric.
  - Typically overfits and likely won't generalize well to new data.
  - But can serve as a sanity check: low accuracy on the training set may indicate an implementation problem.
- Single train/test split
  - Single metric.
  - Speed and simplicity.
  - Lack of variance information
- K-fold cross-validation
  - K train-test splits.
  - Average metric over all splits.
  - Can be combined with parameter grid search: GridSearchCV (def. cv = 3)

Example: Optimizing a Classifier Using Different Evaluation Metrics



Example: Precision-Recall Curve of Default Support Vector Classifier



Training, Validation, and Test Framework for Model Selection and Evaluation

- Using only cross-validation or a test set to do model selection may lead to more subtle overfitting / optimistic generalization estimates
- Instead, use three data splits:
  1. Training set (model building)
  2. Validation set (model selection)
  3. Test set (final evaluation)
- In practice:
  - Create an initial training/test split
  - Do cross-validation on the training data for model/parameter selection
  - Save the held-out test set for final model evaluation

Concluding Notes

- Accuracy is often not the right evaluation metric for many real-world machine learning tasks
  - False positives and false negatives may need to be treated very differently
  - Make sure you understand the needs of your application and choose an evaluation metric that matches your application, user, or business goals.
- Examples of additional evaluation methods include:
  - Learning curve: How much does accuracy (or other metric) change as a function of the amount of training data?
  - Sensitivity analysis: How much does accuracy (or other metric) change as a function of key learning parameter values?

Module 3

September 22, 2021

---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Applied Machine Learning: Module 3 (Evaluation)

1.1 Evaluation for Classification

1.1.1 Preamble

```
In [33]: %matplotlib notebook
         import numpy as np
         import pandas as pd
         import seaborn as sns #NEW HAI YE TO
         import matplotlib.pyplot as plt
         from sklearn.model_selection import train_test_split
         from sklearn.datasets import load_digits #matlab ye pehlse se hi built in

         dataset = load_digits()
         print(type(dataset))
         X, y = dataset.data, dataset.target
         print(X, y)

         for class_name, class_count in zip(dataset.target_names, np.bincount(datas
             print(class_name,class_count)
```

```
<class 'sklearn.datasets.base.Bunch'>
[[  0.   0.   5. ...,   0.   0.   0.]
 [  0.   0.   0. ...,  10.   0.   0.]
 [  0.   0.   0. ...,  16.   9.   0.]
 ...,
 [  0.   0.   1. ...,   6.   0.   0.]
 [  0.   0.   2. ...,  12.   0.   0.]
 [  0.   0.  10. ...,  12.   1.   0.]] [0 1 2 ..., 8 9 8]
0 178
```

```
1 182
2 177
3 183
4 181
5 182
6 181
7 179
8 174
9 180
```

In [34]:
```
# Creating a dataset with imbalanced binary classes:
# Negative class (0) is 'not digit 1'
# Positive class (1) is 'digit 1'
y_binary_imbalanced = y.copy()
y_binary_imbalanced[y_binary_imbalanced != 1] = 0

print('Original labels:\t', y[0:30])
print('New binary labels:\t', y_binary_imbalanced[0:30])
```

```
Original labels:         [0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
New binary labels:         [0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
```

In [35]:
```
np.bincount(y_binary_imbalanced)    # Negative class (0) is the most frequ
```

Out[35]:
```
array([1615,  182])
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced

# Accuracy of Support Vector Machine classifier
from sklearn.svm import SVC

svm = SVC(kernel='rbf', C=1).fit(X_train, y_train)
svm.score(X_test, y_test)
```

Out[36]:
```
0.90888888888888886
```

1.1.2 Dummy Classifiers

DummyClassifier is a classifier that makes predictions using simple rules, which can be useful as a baseline for comparison against actual classifiers, especially with imbalanced classes.

In [37]:
```
from sklearn.dummy import DummyClassifier

# Negative class (0) is most frequent
dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_train,
# Therefore the dummy 'most_frequent' classifier always predicts class 0
y_dummy_predictions = dummy_majority.predict(X_test)

y_dummy_predictions
```

```
Out[37]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
In [38]: dummy_majority.score(X_test, y_test)
```

```
Out[38]: 0.9044444444444445
```

```
In [39]: svm = SVC(kernel='linear', C=1).fit(X_train, y_train) #linear is giving be
         svm.score(X_test, y_test)
```

```
Out[39]: 0.97777777777777775
```

1.1.3 Confusion matrices

Binary (two-class) confusion matrix

1.1.4 note that confusion matrix is taken with y_test and y_majority predicted

```
In [40]: from sklearn.metrics import confusion_matrix

         # Negative class (0) is most frequent
         dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_train,
         y_majority_predicted = dummy_majority.predict(X_test)
         confusion = confusion_matrix(y_test, y_majority_predicted)

         print('Most frequent class (dummy classifier)\n', confusion) #DOES NOT PRE
```

```
Most frequent class (dummy classifier)
 [[407   0]
 [ 43   0]]
```

1.1.5 TOTAL correct answers is sum of diagonal entries

```
In [41]: # produces random predictions w/ same class proportion as training set
         dummy_classprop = DummyClassifier(strategy='stratified').fit(X_train, y_tr
         y_classprop_predicted = dummy_classprop.predict(X_test)
         confusion = confusion_matrix(y_test, y_classprop_predicted)

         print('Random class-proportional prediction (dummy classifier)\n', confusi
```

```
Random class-proportional prediction (dummy classifier)
 [[373  34]
 [ 39   4]]
```

```
In [42]: svm = SVC(kernel='linear', C=1).fit(X_train, y_train)
         svm_predicted = svm.predict(X_test)
         confusion = confusion_matrix(y_test, svm_predicted)

         print('Support vector machine classifier (linear kernel, C=1)\n', confusio
```

```
Support vector machine classifier (linear kernel, C=1)
 [[402   5]
 [  5  38]]
```

```
In [43]: from sklearn.linear_model import LogisticRegression

         lr = LogisticRegression().fit(X_train, y_train) #YAHA PAR LR KO FIT KARKE
         lr_predicted = lr.predict(X_test)
         confusion = confusion_matrix(y_test, lr_predicted)

         print('Logistic regression classifier (default settings)\n', confusion)
```

```
Logistic regression classifier (default settings)
 [[401   6]
 [  6  37]]
```

```
In [44]: from sklearn.tree import DecisionTreeClassifier

         dt = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
         tree_predicted = dt.predict(X_test)
         print(tree_predicted)
         confusion = confusion_matrix(y_test, tree_predicted)

         print('Decision tree classifier (max_depth = 2)\n', confusion)
```

```
[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
```

```
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 1 0 1 0 1 0]
Decision tree classifier (max_depth = 2)
 [[400   7]
 [ 17  26]]
```

1.1.6 Evaluation metrics for binary classification

this is for decision trees only

```
In [45]: from sklearn.metrics import accuracy_score, precision_score, recall_score,
         # Accuracy = TP + TN / (TP + TN + FP + FN)
         # Precision = TP / (TP + FP)
         # Recall = TP / (TP + FN)  Also known as sensitivity, or True Positive Rat
         # F1 = 2 * Precision * Recall / (Precision + Recall)
         print('Accuracy: {:.2f}'.format(accuracy_score(y_test, tree_predicted)))
         print('Precision: {:.2f}'.format(precision_score(y_test, tree_predicted)))
         print('Recall: {:.2f}'.format(recall_score(y_test, tree_predicted)))
         print('F1: {:.2f}'.format(f1_score(y_test, tree_predicted)))
```

```
Accuracy: 0.95
Precision: 0.79
Recall: 0.60
F1: 0.68
```

1.1.7 ye avg/total acche se nahi samjha

1.1.8 1 wale row me values wahi hai jo upar thi, note karna ki upar sab 1 ke liye kiya hai

1.1.9 agar 0 ke liye kiya to wo waha not1 wale row se match hota

```
In [46]: # Combined report with all above metrics
         from sklearn.metrics import classification_report

         print(classification_report(y_test, tree_predicted, target_names=['not 1',
```

```
              precision    recall  f1-score   support

       not 1       0.96      0.98      0.97       407
```

```
          1       0.79      0.60      0.68        43

avg / total       0.94      0.95      0.94       450
```

```
In [47]: print('Random class-proportional (dummy)\n',
                 classification_report(y_test, y_classprop_predicted, target_names=['
         print('SVM\n',
                 classification_report(y_test, svm_predicted, target_names = ['not 1'
         print('Logistic regression\n',
                 classification_report(y_test, lr_predicted, target_names = ['not 1',
         print('Decision tree\n',
                 classification_report(y_test, tree_predicted, target_names = ['not 1
```

```
Random class-proportional (dummy)
              precision    recall  f1-score   support

      not 1       0.91      0.92      0.91       407
          1       0.11      0.09      0.10        43

avg / total       0.83      0.84      0.83       450

SVM
              precision    recall  f1-score   support

      not 1       0.99      0.99      0.99       407
          1       0.88      0.88      0.88        43

avg / total       0.98      0.98      0.98       450

Logistic regression
              precision    recall  f1-score   support

      not 1       0.99      0.99      0.99       407
          1       0.86      0.86      0.86        43

avg / total       0.97      0.97      0.97       450

Decision tree
              precision    recall  f1-score   support

      not 1       0.96      0.98      0.97       407
          1       0.79      0.60      0.68        43

avg / total       0.94      0.95      0.94       450
```

1.1.10 Decision functions

1.1.11 phir in scores ka kuch threshold lekar recall and accuracy lete hai, jisse aage dekhenge

1.1.12 Scores are really scores!

decision function directly X_test pe lagate hai

remember ki probablilty wala do columns me asnwer deta hai, prob score second column me baitha rehta hai

In [48]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced
#y_scores_lr = lr.fit(X_train, y_train).decision_function(X_test)
y_scores_lr = lr.decision_function(X_test) #upar pehle se hi lr fitted hai
y_score_list = list(zip(y_test[0:20], y_scores_lr[0:20]))

# show the decision_function scores for first 20 instances
#print(y_scores_lr, '\n\n')
y_score_list
```

Out[48]:
```
[(0, -23.172292973469546),
 (0, -13.542576515500063),
 (0, -21.717588760007867),
 (0, -18.903065133316439),
 (0, -19.733169947138638),
 (0, -9.7463217496747667),
 (1, 5.2327155658831135),
 (0, -19.308012306288916),
 (0, -25.099330209728528),
 (0, -21.824312362996),
 (0, -24.14378275072049),
 (0, -19.578811099762508),
 (0, -22.568371393280199),
 (0, -10.822590225240777),
 (0, -11.907918741521932),
 (0, -10.977026853802803),
 (1, 11.206811164226373),
 (0, -27.64415761980748),
 (0, -12.857692102545409),
 (0, -25.848149140240199)]
```

1.1.13 PROBABILITY ME DECISION FUNCTION NAHI CHAIYYE for precision recl curve

In [49]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced
y_proba_lr = lr.fit(X_train, y_train).predict_proba(X_test) #matlab second
y_proba_list = list(zip(y_test[0:20], y_proba_lr[0:20,1]))
print(y_proba_lr, '\n\n')

# show the probability of positive class for first 20 instances
y_proba_list
```

```
[[  1.00000000e+00   8.63775792e-11]
 [  9.99998686e-01   1.31381186e-06]
 [  1.00000000e+00   3.69973860e-10]
 [  9.99999994e-01   6.17309725e-09]
 [  9.99999997e-01   2.69149254e-09]
 [  9.99941494e-01   5.85060578e-05]
 [  5.31065356e-03   9.94689346e-01]
 [  9.99999996e-01   4.11753024e-09]
 [  1.00000000e+00   1.25747509e-11]
 [  1.00000000e+00   3.32522908e-10]
 [  1.00000000e+00   3.26955298e-11]
 [  9.99999997e-01   3.14072836e-09]
 [  1.00000000e+00   1.58008641e-10]
 [  9.99980057e-01   1.99434424e-05]
 [  9.99993263e-01   6.73680030e-06]
 [  9.99982910e-01   1.70895406e-05]
 [  1.35811909e-05   9.99986419e-01]
 [  1.00000000e+00   9.86949403e-13]
 [  9.99997394e-01   2.60599836e-06]
 [  1.00000000e+00   5.94691130e-12]
 [  1.00000000e+00   3.95650058e-10]
 [  9.99994985e-01   5.01460864e-06]
 [  1.00000000e+00   4.35976181e-11]
 [  9.99910823e-01   8.91767980e-05]
 [  1.00000000e+00   3.71929871e-14]
 [  9.99999970e-01   3.04813627e-08]
 [  9.99999954e-01   4.60607755e-08]
 [  1.00000000e+00   1.23196382e-16]
 [  1.00000000e+00   2.32702795e-14]
 [  1.00000000e+00   1.33428804e-12]
 [  9.99999945e-01   5.53484900e-08]
 [  1.00000000e+00   3.12464070e-12]
 [  9.99996499e-01   3.50066991e-06]
 [  9.97575629e-01   2.42437119e-03]
 [  9.99997615e-01   2.38534562e-06]
 [  9.99999999e-01   1.46730378e-09]
 [  1.00000000e+00   2.35507474e-10]
 [  8.66610346e-01   1.33389654e-01]
 [  9.99999985e-01   1.47850035e-08]
 [  3.97399638e-07   9.99999603e-01]
 [  9.99485754e-01   5.14246337e-04]
 [  9.99990713e-01   9.28693670e-06]
 [  1.00000000e+00   5.87713473e-12]
 [  1.00000000e+00   1.80875400e-13]
 [  7.86693077e-01   2.13306923e-01]
 [  1.00000000e+00   3.23035786e-13]
 [  1.00000000e+00   4.74699464e-12]
 [  9.99999992e-01   7.88701447e-09]
```

```
[  9.99999999e-01   1.07353678e-09]
[  3.22438474e-03   9.96775615e-01]
[  1.00000000e+00   5.32721236e-12]
[  9.99649130e-01   3.50869683e-04]
[  1.00000000e+00   3.74906363e-11]
[  1.00000000e+00   3.03491498e-11]
[  1.00000000e+00   7.95796014e-14]
[  1.00000000e+00   3.86058370e-13]
[  9.99889925e-01   1.10074868e-04]
[  8.49017420e-01   1.50982580e-01]
[  1.00000000e+00   2.01785458e-12]
[  1.00000000e+00   1.04811395e-12]
[  9.99999228e-01   7.72425715e-07]
[  9.99019428e-01   9.80572400e-04]
[  9.99987162e-01   1.28383738e-05]
[  9.99999815e-01   1.85204464e-07]
[  2.17495931e-03   9.97825041e-01]
[  1.00000000e+00   5.53113224e-12]
[  9.99987387e-01   1.26134291e-05]
[  9.99796362e-01   2.03637959e-04]
[  9.99997820e-01   2.17966868e-06]
[  2.37178411e-02   9.76282159e-01]
[  9.83264066e-01   1.67359342e-02]
[  9.99999915e-01   8.46261570e-08]
[  1.00000000e+00   5.36235621e-14]
[  1.09414900e-07   9.99999891e-01]
[  1.00000000e+00   2.15742932e-10]
[  7.46557466e-01   2.53442534e-01]
[  1.00000000e+00   2.62781179e-11]
[  9.99965143e-01   3.48571490e-05]
[  1.00000000e+00   3.27269467e-14]
[  1.00000000e+00   7.62546168e-13]
[  2.12027787e-04   9.99787972e-01]
[  1.00000000e+00   1.02964625e-12]
[  9.99951655e-01   4.83447248e-05]
[  1.00000000e+00   1.82106336e-15]
[  9.99999999e-01   7.96230152e-10]
[  9.99999873e-01   1.26991831e-07]
[  9.99968462e-01   3.15377954e-05]
[  1.00000000e+00   1.66669436e-13]
[  2.68110964e-01   7.31889036e-01]
[  1.00000000e+00   1.44804203e-13]
[  9.99999582e-01   4.17758192e-07]
[  9.99999998e-01   2.35032559e-09]
[  9.99999967e-01   3.31389721e-08]
[  5.15230724e-05   9.99948477e-01]
[  1.56825948e-01   8.43174052e-01]
[  9.99997392e-01   2.60843452e-06]
```

```
[  9.99994390e-01   5.60976015e-06]
[  9.99998465e-01   1.53507129e-06]
[  9.99996148e-01   3.85248794e-06]
[  9.99795375e-01   2.04624729e-04]
[  1.00000000e+00   3.69919737e-14]
[  9.99999999e-01   5.01752298e-10]
[  9.99999191e-01   8.08756032e-07]
[  9.99999997e-01   2.56688114e-09]
[  1.00000000e+00   1.23319456e-10]
[  1.00000000e+00   1.77313909e-11]
[  9.99999907e-01   9.27545085e-08]
[  5.29705851e-05   9.99947029e-01]
[  1.00000000e+00   3.30087230e-13]
[  6.38993501e-01   3.61006499e-01]
[  9.99941271e-01   5.87290628e-05]
[  1.00000000e+00   3.92795462e-11]
[  9.99999967e-01   3.25394863e-08]
[  1.00000000e+00   1.06921194e-11]
[  9.99998701e-01   1.29896270e-06]
[  9.99999074e-01   9.25898713e-07]
[  1.00000000e+00   3.20725796e-10]
[  1.00000000e+00   6.82952030e-13]
[  9.94877556e-01   5.12244434e-03]
[  9.99816105e-01   1.83894950e-04]
[  1.00000000e+00   1.18014127e-13]
[  9.99954378e-01   4.56215949e-05]
[  9.99957117e-01   4.28825528e-05]
[  9.99999647e-01   3.52748925e-07]
[  8.14071675e-01   1.85928325e-01]
[  9.99999998e-01   2.35160183e-09]
[  1.00000000e+00   2.23619986e-13]
[  9.99999985e-01   1.51301042e-08]
[  3.83121897e-01   6.16878103e-01]
[  9.99999966e-01   3.42275432e-08]
[  9.99999371e-01   6.29083358e-07]
[  9.65740327e-01   3.42596734e-02]
[  9.99954410e-01   4.55899898e-05]
[  9.99999969e-01   3.07035812e-08]
[  9.99999815e-01   1.84790207e-07]
[  1.00000000e+00   1.41855567e-12]
[  9.99999250e-01   7.50191092e-07]
[  9.99999998e-01   1.99229866e-09]
[  9.97357449e-01   2.64255147e-03]
[  9.99997175e-01   2.82493955e-06]
[  9.12710245e-01   8.72897550e-02]
[  9.99999999e-01   1.23749258e-09]
[  9.99981839e-01   1.81612914e-05]
[  9.99999976e-01   2.38685440e-08]
```

```
[  1.00000000e+00   1.70657048e-11]
[  1.74116888e-06   9.99998259e-01]
[  1.00000000e+00   2.08598144e-12]
[  2.17488479e-02   9.78251152e-01]
[  9.99526412e-01   4.73587721e-04]
[  1.00000000e+00   3.69554745e-12]
[  9.99999999e-01   5.61437302e-10]
[  1.32598147e-02   9.86740185e-01]
[  9.99999981e-01   1.91488534e-08]
[  1.00000000e+00   1.78459346e-13]
[  9.99999997e-01   3.19758697e-09]
[  1.00661359e-01   8.99338641e-01]
[  9.99999999e-01   1.00132431e-09]
[  9.99930144e-01   6.98563717e-05]
[  9.99456144e-01   5.43856365e-04]
[  9.99999997e-01   2.79107065e-09]
[  9.99998165e-01   1.83510808e-06]
[  9.99629883e-01   3.70117136e-04]
[  1.00000000e+00   1.29061575e-10]
[  1.00000000e+00   1.47816704e-10]
[  9.99983890e-01   1.61104283e-05]
[  1.00000000e+00   1.04853276e-10]
[  9.09246428e-01   9.07535723e-02]
[  9.98765663e-01   1.23433708e-03]
[  9.99986968e-01   1.30319604e-05]
[  9.99997150e-01   2.85008774e-06]
[  9.99983920e-01   1.60795545e-05]
[  4.05657937e-04   9.99594342e-01]
[  1.00000000e+00   3.80800542e-10]
[  9.99999568e-01   4.32382918e-07]
[  6.26936619e-04   9.99373063e-01]
[  9.99999979e-01   2.05414378e-08]
[  1.00000000e+00   3.67774406e-14]
[  9.99999998e-01   1.50752678e-09]
[  1.00000000e+00   1.06781945e-12]
[  1.00000000e+00   1.62720732e-11]
[  9.99999818e-01   1.81566198e-07]
[  8.51900375e-01   1.48099625e-01]
[  1.00000000e+00   5.30422836e-11]
[  1.00000000e+00   5.14921739e-14]
[  1.00000000e+00   2.20648356e-18]
[  1.00000000e+00   2.18694114e-10]
[  1.00000000e+00   6.09689037e-12]
[  1.00000000e+00   1.52036021e-15]
[  9.99999853e-01   1.46503539e-07]
[  1.00000000e+00   9.21404406e-11]
[  1.00000000e+00   4.87267517e-11]
[  9.99999592e-01   4.08437854e-07]
```

```
[  1.00000000e+00   6.51089407e-11]
[  9.99999921e-01   7.88573860e-08]
[  1.00000000e+00   2.50494501e-16]
[  9.99999998e-01   1.69273985e-09]
[  9.82917675e-04   9.99017082e-01]
[  1.00000000e+00   1.19068418e-15]
[  1.00000000e+00   2.10389709e-11]
[  1.00000000e+00   6.15354760e-12]
[  9.96083311e-01   3.91668871e-03]
[  1.00000000e+00   1.79363873e-13]
[  2.58042871e-04   9.99741957e-01]
[  9.99999889e-01   1.11180883e-07]
[  9.99630501e-01   3.69498895e-04]
[  9.91621353e-01   8.37864661e-03]
[  1.00000000e+00   6.63150570e-12]
[  1.00000000e+00   5.42515810e-11]
[  1.00000000e+00   1.61079735e-12]
[  1.00000000e+00   4.04839490e-10]
[  1.00000000e+00   2.11110545e-10]
[  9.93092627e-01   6.90737289e-03]
[  9.99997319e-01   2.68060809e-06]
[  7.92456064e-01   2.07543936e-01]
[  9.99999999e-01   6.96978487e-10]
[  9.99980621e-01   1.93793908e-05]
[  9.59015233e-06   9.99990410e-01]
[  8.43845751e-01   1.56154249e-01]
[  9.99597391e-01   4.02608574e-04]
[  1.00000000e+00   6.74202487e-14]
[  9.99999998e-01   2.45805989e-09]
[  9.99999586e-01   4.14196175e-07]
[  9.99991996e-01   8.00353170e-06]
[  9.99999675e-01   3.25108155e-07]
[  9.93556974e-01   6.44302564e-03]
[  9.99999996e-01   4.16554802e-09]
[  9.99979637e-01   2.03632795e-05]
[  1.00000000e+00   7.03210728e-11]
[  9.99990870e-01   9.13021169e-06]
[  1.04235185e-02   9.89576482e-01]
[  9.99999917e-01   8.34533400e-08]
[  1.00000000e+00   2.82619267e-15]
[  1.00000000e+00   2.51144133e-18]
[  1.00000000e+00   5.90881496e-15]
[  9.99999997e-01   2.89510222e-09]
[  1.00000000e+00   1.43721433e-12]
[  9.99999948e-01   5.20328818e-08]
[  1.00000000e+00   4.11102839e-15]
[  1.00000000e+00   2.30878088e-17]
[  9.99999994e-01   6.07829998e-09]
```

```
[  1.02372398e-02   9.89762760e-01]
[  1.42321822e-03   9.98576782e-01]
[  9.99999204e-01   7.96169506e-07]
[  1.00000000e+00   4.14274499e-10]
[  9.99999234e-01   7.66418465e-07]
[  9.99999400e-01   5.99578995e-07]
[  9.69164402e-01   3.08355982e-02]
[  9.99999979e-01   2.11869221e-08]
[  1.00000000e+00   1.77972198e-11]
[  9.99997518e-01   2.48217902e-06]
[  9.99996563e-01   3.43696341e-06]
[  1.00000000e+00   1.09048619e-16]
[  9.99999685e-01   3.14582938e-07]
[  1.00000000e+00   1.82111753e-10]
[  9.99998066e-01   1.93434376e-06]
[  1.00000000e+00   1.92683905e-11]
[  9.99999414e-01   5.85728292e-07]
[  9.99999894e-01   1.06456129e-07]
[  1.00000000e+00   1.11300754e-10]
[  9.99999996e-01   4.01820432e-09]
[  9.99997353e-01   2.64720603e-06]
[  1.32494302e-01   8.67505698e-01]
[  9.99994321e-01   5.67870630e-06]
[  1.54646888e-01   8.45353112e-01]
[  1.00000000e+00   3.50036846e-13]
[  9.99999997e-01   2.92684412e-09]
[  9.99996758e-01   3.24174304e-06]
[  9.99859422e-01   1.40577971e-04]
[  9.99955867e-01   4.41333791e-05]
[  1.00000000e+00   2.66104540e-12]
[  9.99999990e-01   9.88517721e-09]
[  9.99999966e-01   3.42637589e-08]
[  9.99999832e-01   1.67622164e-07]
[  9.99992887e-01   7.11339461e-06]
[  1.00000000e+00   1.16505126e-17]
[  8.84307275e-01   1.15692725e-01]
[  9.99999975e-01   2.46801547e-08]
[  9.95139086e-01   4.86091358e-03]
[  9.99899739e-01   1.00261325e-04]
[  1.00000000e+00   2.09926445e-10]
[  1.00000000e+00   1.59326992e-10]
[  9.99999988e-01   1.15293908e-08]
[  9.99999988e-01   1.23701197e-08]
[  9.99974565e-01   2.54346554e-05]
[  9.99999666e-01   3.34352464e-07]
[  9.83608226e-01   1.63917735e-02]
[  9.99999952e-01   4.77260768e-08]
[  1.00000000e+00   4.16716100e-11]
```

```
[  9.99999999e-01   6.47126929e-10]
[  9.99999961e-01   3.86517729e-08]
[  9.99999999e-01   1.30077151e-09]
[  1.00000000e+00   7.74401424e-11]
[  9.99999104e-01   8.96193538e-07]
[  1.00000000e+00   9.32515448e-14]
[  1.00000000e+00   3.03896481e-12]
[  9.99906137e-01   9.38629157e-05]
[  9.99999997e-01   2.76243157e-09]
[  9.99997509e-01   2.49059259e-06]
[  9.99700602e-01   2.99397573e-04]
[  9.99999598e-01   4.01909740e-07]
[  1.00000000e+00   2.38126820e-15]
[  8.96407377e-01   1.03592623e-01]
[  9.99985017e-01   1.49827750e-05]
[  9.99999997e-01   2.53540466e-09]
[  1.00000000e+00   1.94320238e-11]
[  9.99994309e-01   5.69089573e-06]
[  9.99999995e-01   4.86501573e-09]
[  1.00000000e+00   1.30593790e-13]
[  1.00000000e+00   3.94586849e-11]
[  9.99938544e-01   6.14559900e-05]
[  1.00000000e+00   1.29435653e-15]
[  9.99999852e-01   1.48007358e-07]
[  9.99998902e-01   1.09826704e-06]
[  9.99999999e-01   1.00750716e-09]
[  9.99999959e-01   4.06388795e-08]
[  9.99994977e-01   5.02322761e-06]
[  1.00000000e+00   1.07482997e-11]
[  9.99997405e-01   2.59548291e-06]
[  1.00000000e+00   8.30450001e-13]
[  9.99999843e-01   1.56763354e-07]
[  9.99999896e-01   1.04076941e-07]
[  1.00000000e+00   5.70099838e-13]
[  9.99999998e-01   2.29970355e-09]
[  9.99999999e-01   9.55940100e-10]
[  9.99910151e-01   8.98485120e-05]
[  4.56154724e-01   5.43845276e-01]
[  9.99991332e-01   8.66787371e-06]
[  1.03119936e-05   9.99989688e-01]
[  1.00000000e+00   4.39362913e-14]
[  9.99860528e-01   1.39472386e-04]
[  9.99999989e-01   1.14531551e-08]
[  9.82850308e-01   1.71496919e-02]
[  9.99999999e-01   1.34801040e-09]
[  9.99999988e-01   1.18623275e-08]
[  1.00000000e+00   9.49921631e-13]
[  9.99997276e-01   2.72410403e-06]
```

```
[  9.99994416e-01   5.58386650e-06]
[  1.00000000e+00   8.27415141e-11]
[  1.00000000e+00   1.41803083e-11]
[  1.31826430e-02   9.86817357e-01]
[  9.15092395e-01   8.49076049e-02]
[  8.46053093e-06   9.99991539e-01]
[  9.99999998e-01   2.49806953e-09]
[  9.99500576e-01   4.99424139e-04]
[  9.80889004e-01   1.91109957e-02]
[  9.99999999e-01   5.01212343e-10]
[  1.00000000e+00   7.14733616e-11]
[  9.99998825e-01   1.17457669e-06]
[  2.08052071e-01   7.91947929e-01]
[  1.40506770e-02   9.85949323e-01]
[  9.99999998e-01   1.68849637e-09]
[  1.00000000e+00   7.14881399e-12]
[  9.99999902e-01   9.83986876e-08]
[  1.00000000e+00   2.45971435e-13]
[  9.99999996e-01   4.33714041e-09]
[  1.00000000e+00   8.03633646e-11]
[  1.00000000e+00   9.64639922e-12]
[  9.99999988e-01   1.21110204e-08]
[  9.94386163e-01   5.61383720e-03]
[  9.99999788e-01   2.12001626e-07]
[  1.00000000e+00   2.34168699e-10]
[  9.99999996e-01   3.81818342e-09]
[  1.00000000e+00   1.26270117e-14]
[  9.99996604e-01   3.39612297e-06]
[  1.00000000e+00   1.58993946e-10]
[  9.99947525e-01   5.24752595e-05]
[  1.00000000e+00   3.72247836e-11]
[  9.99996170e-01   3.83035100e-06]
[  1.00000000e+00   4.03932220e-10]
[  1.00000000e+00   2.94793143e-12]
[  9.99999746e-01   2.53854094e-07]
[  9.99999998e-01   1.81907075e-09]
[  1.00000000e+00   1.97891485e-13]
[  1.00000000e+00   1.08442419e-11]
[  9.99993995e-01   6.00456557e-06]
[  9.99999219e-01   7.81305840e-07]
[  1.00000000e+00   3.42023128e-12]
[  1.00000000e+00   1.50763937e-12]
[  1.00000000e+00   4.48725687e-15]
[  9.99997264e-01   2.73610088e-06]
[  7.23135543e-02   9.27686446e-01]
[  9.99788535e-01   2.11465487e-04]
[  9.99997703e-01   2.29692069e-06]
[  1.00000000e+00   8.86692320e-13]
```

```
[  1.00000000e+00   5.77669586e-13]
[  9.99851786e-01   1.48213948e-04]
[  9.99999994e-01   5.82005967e-09]
[  1.00000000e+00   5.03335396e-13]
[  9.99958537e-01   4.14628861e-05]
[  1.00000000e+00   2.35244547e-10]
[  1.00000000e+00   8.25414031e-14]
[  9.99996522e-01   3.47841222e-06]
[  1.00000000e+00   9.38378722e-13]
[  9.99999999e-01   1.46973137e-09]
[  9.99999904e-01   9.63266515e-08]
[  9.99972687e-01   2.73129444e-05]
[  1.00000000e+00   1.91004822e-15]
[  9.99999996e-01   3.74448346e-09]
[  1.04359290e-03   9.98956407e-01]
[  1.00000000e+00   3.88016487e-10]
[  9.29454214e-01   7.05457862e-02]
[  9.99999965e-01   3.51762977e-08]
[  1.00000000e+00   2.57187900e-11]
[  1.00000000e+00   2.56244258e-10]
[  9.99999469e-01   5.30539460e-07]
[  9.99270825e-01   7.29174569e-04]
[  1.00000000e+00   9.81473128e-11]
[  9.99155158e-01   8.44841633e-04]
[  1.46190742e-02   9.85380926e-01]
[  1.00000000e+00   1.52483645e-10]
[  1.00000000e+00   8.22081950e-17]
[  9.99972012e-01   2.79875155e-05]
[  1.00000000e+00   1.91310789e-13]
[  9.99999110e-01   8.89981604e-07]
[  1.00000000e+00   1.45094590e-11]
[  9.70295043e-01   2.97049566e-02]
[  8.06009779e-02   9.19399022e-01]
[  9.99998952e-01   1.04828868e-06]
[  9.99999768e-01   2.31915159e-07]
[  1.00000000e+00   7.36620830e-15]
[  7.17444554e-03   9.92825554e-01]
[  1.00000000e+00   2.24883212e-10]
[  9.99999972e-01   2.84906479e-08]
[  3.44831513e-01   6.55168487e-01]
[  1.00000000e+00   1.50539208e-10]
[  9.99999845e-01   1.55334420e-07]
[  1.00000000e+00   2.00607960e-12]
[  1.00000000e+00   5.49182969e-11]
[  1.00000000e+00   2.93485257e-11]
[  1.00000000e+00   4.15586902e-17]
[  1.00000000e+00   9.07407020e-12]
[  1.00000000e+00   1.73824391e-19]
```

```
 [  9.99999999e-01   1.09145239e-09]
 [  9.99999787e-01   2.12574862e-07]
 [  3.71044702e-05   9.99962896e-01]
 [  9.99999999e-01   1.12247040e-09]
 [  1.00000000e+00   2.26968078e-12]
 [  9.99041156e-01   9.58844088e-04]
 [  1.00000000e+00   1.70509524e-13]
 [  9.99999919e-01   8.11405752e-08]
 [  9.99999993e-01   6.99666465e-09]
 [  4.67325774e-01   5.32674226e-01]
 [  9.99999998e-01   1.74976592e-09]
 [  9.99548166e-01   4.51834389e-04]
 [  7.29919061e-04   9.99270081e-01]
 [  9.99999997e-01   3.16566212e-09]
 [  9.99995178e-01   4.82183377e-06]
 [  9.99997736e-01   2.26441080e-06]
 [  2.26390831e-04   9.99773609e-01]
 [  1.00000000e+00   1.11825067e-15]]
```

```
Out[49]: [(0, 8.6377579220606777e-11),
          (0, 1.3138118599563783e-06),
          (0, 3.6997386039099529e-10),
          (0, 6.1730972504865465e-09),
          (0, 2.6914925394345074e-09),
          (0, 5.8506057771143608e-05),
          (1, 0.99468934644404694),
          (0, 4.1175302368500096e-09),
          (0, 1.2574750894253029e-11),
          (0, 3.3252290754668869e-10),
          (0, 3.2695529799373086e-11),
          (0, 3.1407283576084884e-09),
          (0, 1.5800864117150149e-10),
          (0, 1.9943442430612578e-05),
          (0, 6.7368003023860014e-06),
          (0, 1.7089540581641637e-05),
          (1, 0.9999864188091131),
          (0, 9.8694940340195476e-13),
          (0, 2.6059983600823893e-06),
          (0, 5.9469113009063784e-12)]
```

1.1.14 Precision-recall curves, simply precision vs recall

1.1.15 Precision recall curve direct hi deta hai precision, recall and thresholds!

```
In [50]: #np.argmin? #Returns the indices of the minimum values along an axis.
```

wo point pe kya tha exactly?

```
In [51]: from sklearn.metrics import precision_recall_curve

         precision, recall, thresholds = precision_recall_curve(y_test, y_scores_lr

         closest_zero = np.argmin(np.abs(thresholds)) #this is an index
         print(closest_zero)#, '\n\n', np.abs(thresholds))
         print(precision,'\n', recall,'\n', thresholds)
         closest_zero_p = precision[closest_zero]
         closest_zero_r = recall[closest_zero]

         plt.figure()
         plt.xlim([0.0, 1.01])
         plt.ylim([0.0, 1.01])
         plt.plot(precision, recall, label='Precision-Recall Curve')
         plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle =
         plt.xlabel('Precision', fontsize=16)
         plt.ylabel('Recall', fontsize=16)
         plt.axes().set_aspect('equal')
         plt.show()
```

```
20
[ 0.68253968  0.67741935  0.68852459  0.7         0.71186441  0.72413793
  0.73684211  0.75        0.76363636  0.77777778  0.77358491  0.78846154
  0.78431373  0.8         0.79591837  0.8125      0.82978723  0.82608696
  0.82222222  0.84090909  0.86046512  0.85714286  0.85365854  0.85
  0.87179487  0.86842105  0.86486486  0.88888889  0.91428571  0.91176471
  0.90909091  0.90625     0.90322581  0.9         0.89655172  0.89285714
  0.88888889  0.88461538  0.88        0.875       0.86956522  0.86363636
  0.85714286  0.85        0.84210526  0.83333333  0.88235294  0.9375
  0.93333333  0.92857143  0.92307692  0.91666667  0.90909091  1.          1.
  1.          1.          1.          1.          1.          1.          1.
  1.          1.        ]
 [ 1.          0.97674419  0.97674419  0.97674419  0.97674419  0.97674419
  0.97674419  0.97674419  0.97674419  0.97674419  0.95348837  0.95348837
  0.93023256  0.93023256  0.90697674  0.90697674  0.90697674  0.88372093
  0.86046512  0.86046512  0.86046512  0.8372093   0.81395349  0.79069767
  0.79069767  0.76744186  0.74418605  0.74418605  0.74418605  0.72093023
  0.69767442  0.6744186   0.65116279  0.62790698  0.60465116  0.58139535
  0.55813953  0.53488372  0.51162791  0.48837209  0.46511628  0.44186047
  0.41860465  0.39534884  0.37209302  0.34883721  0.34883721  0.34883721
  0.3255814   0.30232558  0.27906977  0.25581395  0.23255814  0.23255814
  0.20930233  0.18604651  0.1627907   0.13953488  0.11627907  0.09302326
  0.06976744  0.04651163  0.02325581  0.        ]
 [ -4.04847662  -3.93819545  -3.48628627  -3.44776445  -3.33892603
  -2.5783356   -2.37746137  -2.34718536  -2.30446832  -2.15792885
  -2.03386685  -1.87131487  -1.7495844   -1.72691524  -1.68712543
  -1.47668716  -1.33979401  -1.3051061   -1.08033549  -0.57099832
```

```
  0.13088342   0.17583273   0.47631823   0.6418365    1.00422797
  1.33670725   1.68203683   1.69861005   1.87908244   2.18989765
  2.43420944   2.55168221   3.71752409   3.80620565   4.21070117
  4.25093438   4.30966876   4.31558393   4.55321241   4.57143325
  4.93002949   5.23271557   5.73378353   6.12856799   6.55341039
  6.86404167   6.92400179   7.22184672   7.37403798   7.80959453
  8.26212674   8.3930213    8.45858117   9.84572083   9.87342932
 10.201736    11.20681116  11.4821926   11.55476419  11.68009017
 13.26095216  14.73832302  16.02811865]


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

1.1.16 ROC curves, Area-Under-Curve (AUC)

roc curve will return fpr, tpr, threshold

```
In [52]: from sklearn.metrics import roc_curve, auc

         X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced

         y_score_lr = lr.fit(X_train, y_train).decision_function(X_test)
         #print(y_scores_lr, '\n\n') huge array!
         fpr_lr, tpr_lr, thresholds = roc_curve(y_test, y_score_lr) #FALSE POSITIVE
         print(fpr_lr, '\n\n', tpr_lr, '\n\n', thresholds)
         roc_auc_lr = auc(fpr_lr, tpr_lr)

         plt.figure()
         plt.xlim([-0.01, 1.00])
         plt.ylim([-0.01, 1.01])
         plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})'.
         plt.xlabel('False Positive Rate', fontsize=16)
         plt.ylabel('True Positive Rate', fontsize=16)
         plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
         plt.legend(loc='lower right', fontsize=13)
         plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
         plt.axes().set_aspect('equal')
         plt.show()
```

```
[ 0.          0.          0.002457    0.002457    0.00737101  0.00737101
  0.01228501  0.01228501  0.01474201  0.01474201  0.01965602  0.01965602
  0.02457002  0.02457002  0.02702703  0.02702703  0.02948403  0.02948403
  0.04914005  0.04914005  1.        ]

 [ 0.02325581  0.23255814  0.23255814  0.34883721  0.34883721  0.74418605
```

```
  0.74418605  0.79069767  0.79069767  0.86046512  0.86046512  0.90697674
  0.90697674  0.93023256  0.93023256  0.95348837  0.95348837  0.97674419
  0.97674419  1.          1.        ]

 [ 16.02811865   9.84572083   8.45858117   7.22184672   6.86404167
   1.87908244   1.68203683   1.00422797   0.6418365    0.13088342
  -1.08033549  -1.33979401  -1.68712543  -1.72691524  -1.7495844
  -1.87131487  -2.03386685  -2.15792885  -3.93819545  -4.04847662
 -43.19624141]


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

1.1.17 AREA should be 1 or close to 1 as much as possible

In [ ]:

In [53]:
```
from matplotlib import cm

X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])

#ABHI GAMMA PARAMETER KO CHANGE KARKE SVC USE KIYA HAI
for g in [0.01, 0.1, 0.20, 1]:
    svm = SVC(gamma=g).fit(X_train, y_train)
    y_score_svm = svm.decision_function(X_test)
    fpr_svm, tpr_svm, _ = roc_curve(y_test, y_score_svm) # _ me roc curve
    roc_auc_svm = auc(fpr_svm, tpr_svm)
    accuracy_svm = svm.score(X_test, y_test)
    print("gamma = {:.2f}  accuracy = {:.2f}   AUC = {:.2f}".format(g, acc
                                                                    roc_au
    plt.plot(fpr_svm, tpr_svm, lw=3, alpha=0.7,
             label='SVM (gamma = {:0.2f}, area = {:0.2f})'.format(g, roc_a

plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate (Recall)', fontsize=16)
plt.plot([0, 1], [0, 1], color='k', lw=0.5, linestyle='--')
plt.legend(loc="lower right", fontsize=11)
plt.title('ROC curve: (1-of-10 digits classifier)', fontsize=16)
plt.axes().set_aspect('equal')

plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
gamma = 0.01  accuracy = 0.91   AUC = 1.00
gamma = 0.10  accuracy = 0.90   AUC = 0.98
gamma = 0.20  accuracy = 0.90   AUC = 0.66
gamma = 1.00  accuracy = 0.90   AUC = 0.50
```

1.1.18 Evaluation measures for multi-class classification

Multi-class confusion matrix

```
In [54]: print(i for i in range(0,10))
         print([i for i in range(0,10)])
         print(range(0,10))
         print([range(0,10)])
```

```
<generator object <genexpr> at 0x7f5e21ab4620>
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(0, 10)
[range(0, 10)]
```

```
In [55]: dataset = load_digits()
         X, y = dataset.data, dataset.target
         X_train_mc, X_test_mc, y_train_mc, y_test_mc = train_test_split(X, y, rand


         svm = SVC(kernel = 'linear').fit(X_train_mc, y_train_mc)  #training on tra
         svm_predicted_mc = svm.predict(X_test_mc) #testing on test data
         confusion_mc = confusion_matrix(y_test_mc, svm_predicted_mc) #confusion ma
         print(confusion_mc)
         df_cm = pd.DataFrame(confusion_mc,
                              index = [i for i in range(0,10)], columns = [i for i
         #CONFUSION MATRIX IS CONVERTED TO DF BECAUSE HEATMAP IS TO BE MADE FROM IT

         plt.figure(figsize=(5.5,4))
         sns.heatmap(df_cm, annot=True)
         plt.title('SVM Linear Kernel \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
                                                                                svm
         plt.ylabel('True label')
         plt.xlabel('Predicted label')


         #same but for rbf kernel
```

```
svm = SVC(kernel = 'rbf').fit(X_train_mc, y_train_mc)
svm_predicted_mc = svm.predict(X_test_mc)
confusion_mc = confusion_matrix(y_test_mc, svm_predicted_mc)
print(confusion_mc)
df_cm = pd.DataFrame(confusion_mc, index = [i for i in range(0,10)],
                     columns = [i for i in range(0,10)])

plt.figure(figsize = (5.5,4))
sns.heatmap(df_cm, annot=True)
plt.title('SVM RBF Kernel \nAccuracy:{0:.3f}'.format(accuracy_score(y_test
                                                                    svm_pr
plt.ylabel('True label')
plt.xlabel('Predicted label');
```

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 42  0  0  0  0  0  0  1  0]
 [ 0  0 44  0  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  1  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  0  0  0  0 47  0  0  0  1]
 [ 0  1  0  0  0  0 51  0  0  0]
 [ 0  0  0  0  1  0  0 47  0  0]
 [ 0  3  1  0  0  0  0  0 44  0]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[[24  0  0  0 13  0  0  0  0  0]
 [ 0 10  0  0 33  0  0  0  0  0]
 [ 0  0 17  0 27  0  0  0  0  0]
 [ 0  0  0 42  3  0  0  0  0  0]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  0  0  0 32 16  0  0  0  0]
 [ 0  0  0  0 24  0 28  0  0  0]
 [ 0  0  0  0 31  0  0 17  0  0]
 [ 0  0  0  0 47  0  0  0  1  0]
 [ 0  0  0  0 21  0  0  0  0 26]]
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.1.19 second wale graph se pata chala ki rbf kernel unncessarily 4 value predict karta hai, jo ki unexpected hai

Multi-class classification report

1.1.20 clearly 4 wale ka precision kam hai!

```
In [56]: print(classification_report(y_test_mc, svm_predicted_mc))
```

```
             precision    recall  f1-score   support

          0       1.00      0.65      0.79        37
          1       1.00      0.23      0.38        43
          2       1.00      0.39      0.56        44
          3       1.00      0.93      0.97        45
          4       0.14      1.00      0.25        38
          5       1.00      0.33      0.50        48
          6       1.00      0.54      0.70        52
          7       1.00      0.35      0.52        48
          8       1.00      0.02      0.04        48
          9       1.00      0.55      0.71        47

avg / total       0.93      0.49      0.54       450
```

Micro- vs. macro-averaged metrics

See the slides, they are nice!

1.1.21 precision score

```
In [57]: print('Micro-averaged precision = {:.2f} (treat instances equally)'
                .format(precision_score(y_test_mc, svm_predicted_mc, average = 'micr
         print('Macro-averaged precision = {:.2f} (treat classes equally)'
                .format(precision_score(y_test_mc, svm_predicted_mc, average = 'macr
```

```
Micro-averaged precision = 0.49 (treat instances equally)
Macro-averaged precision = 0.91 (treat classes equally)
```

1.1.22 f1 score

```
In [58]: print('Micro-averaged f1 = {:.2f} (treat instances equally)'
                .format(f1_score(y_test_mc, svm_predicted_mc, average = 'micro')))
         print('Macro-averaged f1 = {:.2f} (treat classes equally)'
                .format(f1_score(y_test_mc, svm_predicted_mc, average = 'macro')))
```

```
Micro-averaged f1 = 0.49 (treat instances equally)
Macro-averaged f1 = 0.54 (treat classes equally)
```

1.1.23 Regression evaluation metrics

In [27]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.dummy import DummyRegressor

diabetes = datasets.load_diabetes()
#print(diabetes)
X = diabetes.data[:, None, 6]
y = diabetes.target


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

lm = LinearRegression().fit(X_train, y_train)
lm_dummy_mean = DummyRegressor(strategy = 'mean').fit(X_train, y_train)

y_predict = lm.predict(X_test)
y_predict_dummy_mean = lm_dummy_mean.predict(X_test)

print('Linear model, coefficients: ', lm.coef_)
print("Mean squared error (dummy): {:.2f}".format(mean_squared_error(y_tes
print("Mean squared error (linear model): {:.2f}".format(mean_squared_erro
print("r2_score (dummy): {:.2f}".format(r2_score(y_test, y_predict_dummy_m
print("r2_score (linear model): {:.2f}".format(r2_score(y_test, y_predict)

# Plot outputs
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_predict, color='green', linewidth=2)
plt.plot(X_test, y_predict_dummy_mean, color='red', linestyle = 'dashed',
         linewidth=2, label = 'dummy')

plt.show()
```

```
Linear model, coefficients:  [-698.80206267]
Mean squared error (dummy): 4965.13
Mean squared error (linear model): 4646.74
r2_score (dummy): -0.00
r2_score (linear model): 0.06
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.1.24 remember ki jab sab output prediction same rehta hai, tab r2 score 0 aane chahiye

1.1.25 Model selection using evaluation metrics

Cross-validation example

```
In [28]: from sklearn.model_selection import cross_val_score
         from sklearn.svm import SVC

         dataset = load_digits()
         # again, making this a binary problem with 'digit 1' as positive class
         # and 'not 1' as negative class
         X, y = dataset.data, dataset.target == 1
         print(X, '\n\n',y)
         clf = SVC(kernel='linear', C=1)

         # accuracy is the default scoring metric
         print('Cross-validation (accuracy)', cross_val_score(clf, X, y, cv=5))
         # use AUC as scoring metric
         print('Cross-validation (AUC)', cross_val_score(clf, X, y, cv=5, scoring =
         # use recall as scoring metric
         print('Cross-validation (recall)', cross_val_score(clf, X, y, cv=5, scorin
```

```
[[  0.   0.   5. ...,   0.   0.   0.]
 [  0.   0.   0. ...,  10.   0.   0.]
 [  0.   0.   0. ...,  16.   9.   0.]
 ...,
 [  0.   0.   1. ...,   6.   0.   0.]
 [  0.   0.   2. ...,  12.   0.   0.]
 [  0.   0.  10. ...,  12.   1.   0.]]

 [False  True False ..., False False False]
Cross-validation (accuracy) [ 0.91944444  0.98611111  0.97214485  0.97493036  0.969
Cross-validation (AUC) [ 0.9641871   0.9976571   0.99372205  0.99699002  0.98675611
Cross-validation (recall) [ 0.81081081  0.89189189  0.83333333  0.83333333  0.83333
```

Grid search example

1.1.26 best params and best score are kaam ke built in

gridsearchcv ek naya type hai

```
In [29]: from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import roc_auc_score

         dataset = load_digits()
         X, y = dataset.data, dataset.target == 1
         # again, making this a binary problem with 'digit 1' as positive class
         # and 'not 1' as negative class
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

         clf = SVC(kernel='rbf')
         grid_values = {'gamma': [0.001, 0.01, 0.05, 0.1, 1, 10, 100]}

         # default metric to optimize over grid parameters: accuracy #YE TO KUCH IN
         grid_clf_acc = GridSearchCV(clf, param_grid = grid_values)  #BY DEAFULT AC
         grid_clf_acc.fit(X_train, y_train)
         y_decision_fn_scores_acc = grid_clf_acc.decision_function(X_test)
         #DECISION FUNCTION UPAR BHI KIYA HAI, DECISION SCORE ME BAHOT SE SCORE MIL

         print('Grid best parameter (max. accuracy): ', grid_clf_acc.best_params_)
         print('Grid best score (accuracy): ', grid_clf_acc.best_score_)


         # alternative metric to optimize over grid parameters: AUC
         grid_clf_auc = GridSearchCV(clf, param_grid = grid_values, scoring = 'roc_
         grid_clf_auc.fit(X_train, y_train)
         y_decision_fn_scores_auc = grid_clf_auc.decision_function(X_test)


         print('Test set AUC: ', roc_auc_score(y_test, y_decision_fn_scores_auc))
         print('Grid best parameter (max. AUC): ', grid_clf_auc.best_params_)
         print('Grid best score (AUC): ', grid_clf_auc.best_score_)
```

```
Grid best parameter (max. accuracy):  {'gamma': 0.001}
Grid best score (accuracy):  0.996288047513
Test set AUC:  0.999828581224
Grid best parameter (max. AUC):  {'gamma': 0.001}
Grid best score (AUC):  0.99987412783
```

Evaluation metrics supported for model selection

```
In [30]: from sklearn.metrics.scorer import SCORERS

         print(sorted(list(SCORERS.keys())))
         #print(sorted(list(SCORERS.values()))) #INVALID HAI YE STATEMENT
```

```
['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'f1_macro', 'f1_micr
```

1.1.27 Two-feature classification example using the digits dataset

Optimizing a classifier using different evaluation metrics

We are trying to get the best results using different evaluation metrix

```
In [31]: from sklearn.datasets import load_digits
         from sklearn.model_selection import train_test_split
         from adspy_shared_utilities import plot_class_regions_for_classifier_subpl
         from sklearn.svm import SVC
         from sklearn.model_selection import GridSearchCV


         dataset = load_digits()
         X, y = dataset.data, dataset.target == 1 #AGAIN BINARY ME SET KIYA
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

         # Create a two-feature input vector matching the example plot above
         # We jitter the points (add a small amount of random noise) in case there
         # in feature space where many instances have the same features.

         jitter_delta = 0.25
         X_twovar_train = X_train[:,[20,59]]+ np.random.rand(X_train.shape[0], 2) -
         X_twovar_test  = X_test[:,[20,59]] + np.random.rand(X_test.shape[0], 2) -

         clf = SVC(kernel = 'linear').fit(X_twovar_train, y_train)
         grid_values = {'class_weight':['balanced', {1:2},{1:3},{1:4},{1:5},{1:10},
         plt.figure(figsize=(9,6))


         for i, eval_metric in enumerate(('precision','recall', 'f1','roc_auc')):
             grid_clf_custom = GridSearchCV(clf, param_grid=grid_values, scoring=ev
             grid_clf_custom.fit(X_twovar_train, y_train) #simple training
             print('Grid best parameter (max. {0}): {1}'
                   .format(eval_metric, grid_clf_custom.best_params_)) #best_params
             print('Grid best score ({0}): {1}'
                   .format(eval_metric, grid_clf_custom.best_score_)) #best_score_
             plt.subplots_adjust(wspace=0.3, hspace=0.3)
             plot_class_regions_for_classifier_subplot(grid_clf_custom, X_twovar_te
                                                       None, None,  plt.subplot(2, 2

             plt.title(eval_metric+'-oriented SVC')
         plt.tight_layout()
         plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Grid best parameter (max. precision): {'class_weight': {1: 2}}
Grid best score (precision): 0.5286027424563252
Grid best parameter (max. recall): {'class_weight': {1: 50}}
Grid best score (recall): 0.935661321592438
Grid best parameter (max. f1): {'class_weight': {1: 3}}
Grid best score (f1): 0.5042831436045673
Grid best parameter (max. roc_auc): {'class_weight': {1: 5}}
Grid best score (roc_auc): 0.8879494460252957
```

Precision-recall curve for the default SVC classifier (with balanced class weights)

ye bhi theek hai, sab upar kiya hua hai

In [32]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from adspy_shared_utilities import plot_class_regions_for_classifier
from sklearn.svm import SVC

dataset = load_digits()
X, y = dataset.data, dataset.target == 1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# create a two-feature input vector matching the example plot above
jitter_delta = 0.25
X_twovar_train = X_train[:,[20,59]]+ np.random.rand(X_train.shape[0], 2) -
X_twovar_test  = X_test[:,[20,59]] + np.random.rand(X_test.shape[0], 2) -

clf = SVC(kernel='linear', class_weight='balanced').fit(X_twovar_train, y_

y_scores = clf.decision_function(X_twovar_test) #as usual, decision functi

precision, recall, thresholds = precision_recall_curve(y_test, y_scores) #
closest_zero = np.argmin(np.abs(thresholds)) #returns a number
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]

plot_class_regions_for_classifier(clf, X_twovar_test, y_test)
plt.title("SVC, class_weight = 'balanced', optimized for accuracy")
plt.show()

plt.figure()
plt.xlim([0.0, 1.01])
```

```
plt.ylim([0.0, 1.01])
plt.title ("Precision-recall curve: SVC, class_weight = 'balanced'")
plt.plot(precision, recall, label = 'Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize=12, fillstyle='no
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
print('At zero threshold, precision: {:.2f}, recall: {:.2f}'
      .format(closest_zero_p, closest_zero_r))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
At zero threshold, precision: 0.21, recall: 0.74
```

In [ ]:

Applied Machine Learning

Supervised machine learning (Part 2)

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Applied Machine Learning

Naïve Bayes Classifiers

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Naïve Bayes Classifiers: a simple, probabilistic classifier family

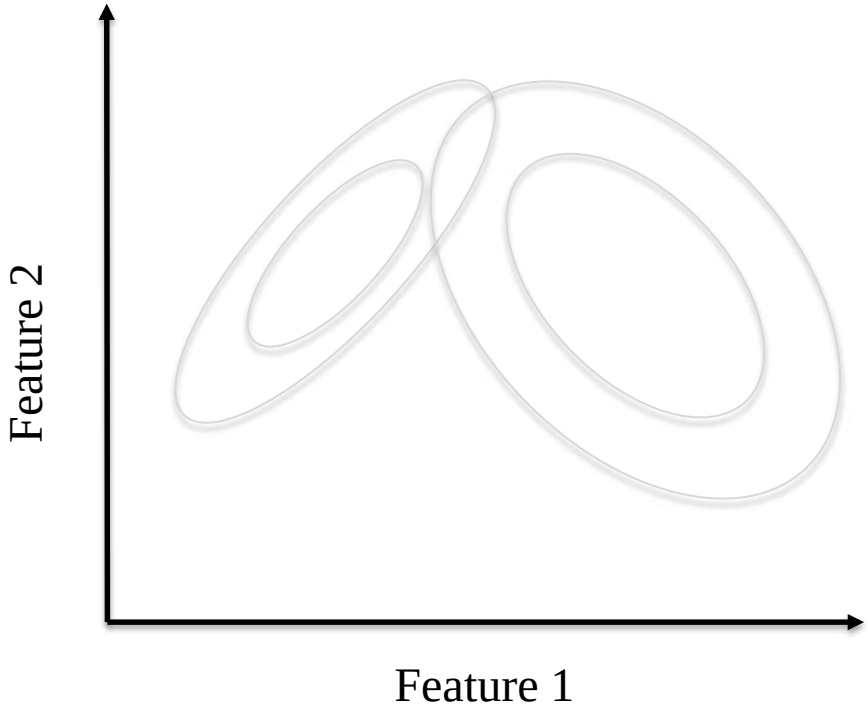
- These classifiers are called 'Naïve' because they assume that features are conditionally independent, given the class.
- In other words: they assume that, for all instances of a given class, the features have little/no correlation with each other.
- Highly efficient learning and prediction.
- But generalization performance may worse than more sophisticated learning methods.
- Can be competitive for some tasks.

Naïve Bayes classifier types

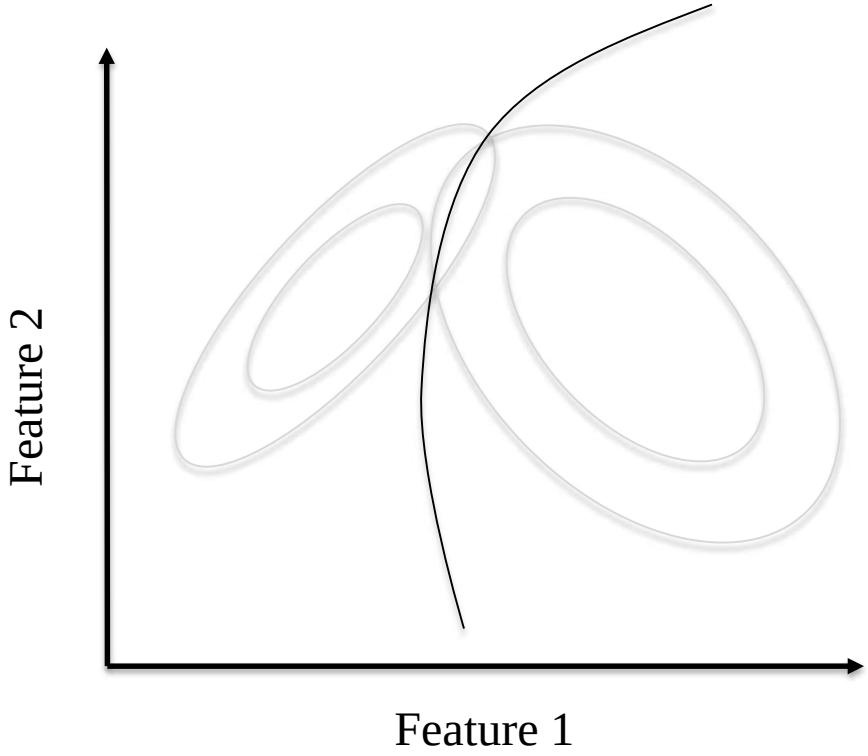
- **Bernoulli: binary features (e.g. word presence/absence)**
- **Multinomial: discrete features (e.g. word counts)**
- **Gaussian: continuous/real-valued features**
  - *Statistics computed for each class:*
    - *For each feature: mean, standard deviation*

- **See the Applied Text Mining course for more details on the Bernoulli and Multinomial Naïve Bayes models**

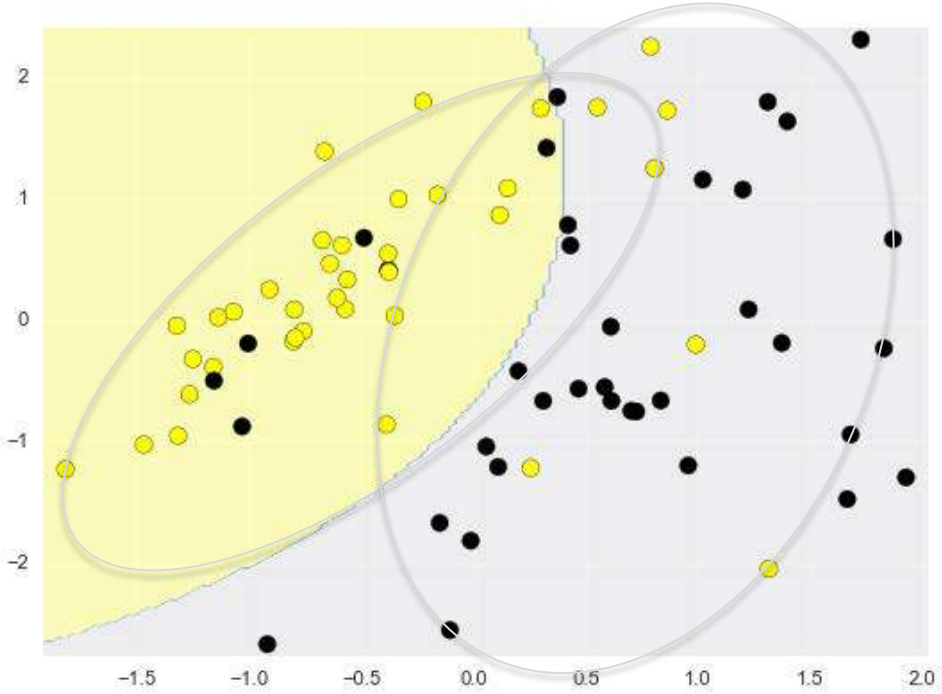
Gaussian Naïve Bayes classifier



Gaussian Naïve Bayes classifier



Gaussian Naïve Bayes classifier



Naïve Bayes classifiers: Pros and Cons

Pros:

- Easy to understand
- Simple, efficient parameter estimation
- Works well with high-dimensional data
- Often useful as a baseline comparison against more sophisticated methods

Cons:

- Assumption that features are conditionally independent given the class is not realistic.
- As a result, other classifier types often have better generalization performance.
- Their confidence estimates for predictions are not very accurate.

Applied Machine Learning

Random Forests

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Random Forests

- **An ensemble of trees, not just one tree.**
- **Widely used, very good results on many problems.**
- **`sklearn.ensemble` module:**
  - *Classification:* `RandomForestClassifier`
  - *Regression:* `RandomForestRegressor`
- **One decision tree → Prone to overfitting.**
- **Many decision trees → More stable, better generalization**
- **Ensemble of trees should be diverse: introduce random variation into tree-building.**

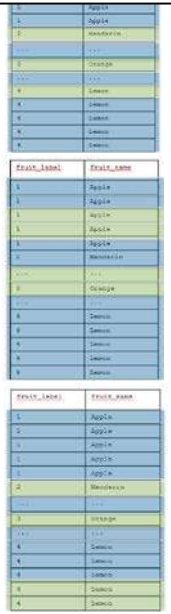
Random Forest Process

Original dataset

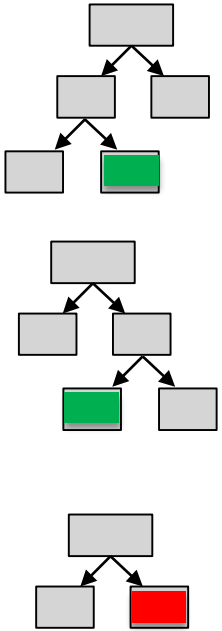
fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Randomized bootstrap copies

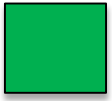
n_estimator



Randomized feature splits



Ensemble prediction



Random Forest Process: Bootstrap Samples

Bootstrap sample 1

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Bootstrap sample 2

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Bootstrap sample 3

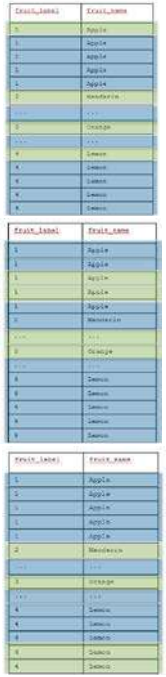
fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Random Forest Process

Original dataset

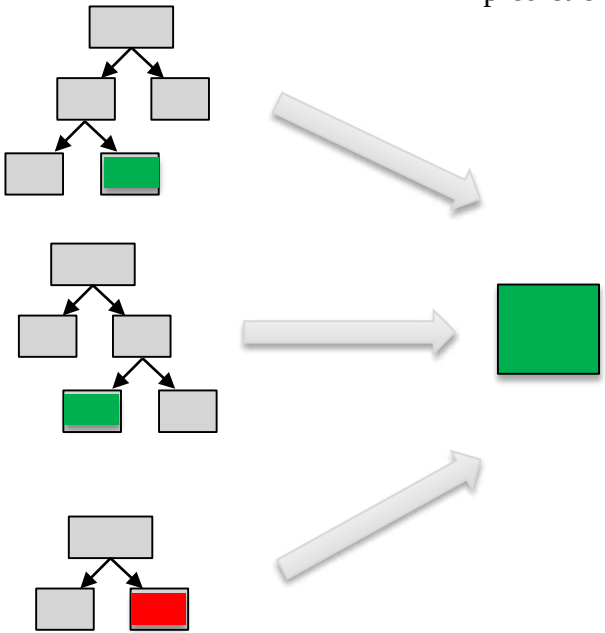
fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

Randomized bootstrap copies



Randomized feature splits

max_features



Ensemble prediction

Random Forest `max_features` Parameter

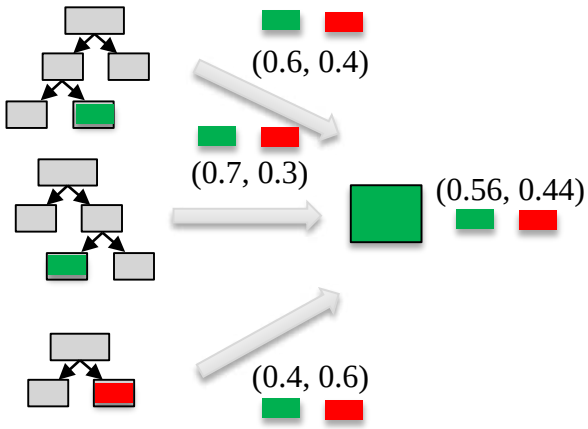
- Learning is quite sensitive to `max_features`.
- Setting `max_features = 1` leads to forests with diverse, more complex trees.
- Setting `max_features = <close to number of features>` will lead to similar forests with simpler trees.

Prediction Using Random Forests

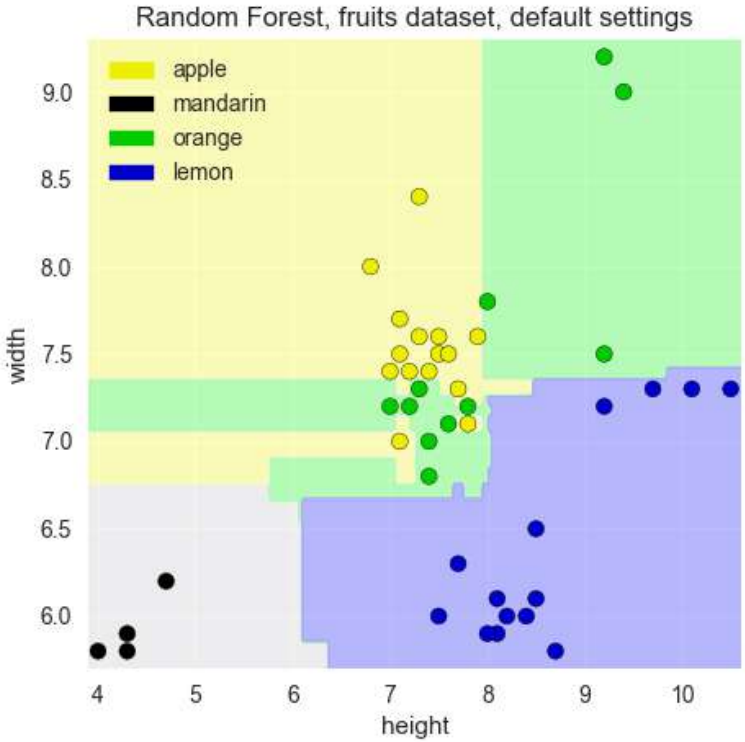
1. Make a prediction for every tree in the forest.

2. Combine individual predictions

- *Regression: mean of individual tree predictions.*
- *Classification:*
  - *Each tree gives probability for each class.*
  - *Probabilities averaged across trees.*
  - *Predict the class with highest probability.*



Random Forest: Fruit Dataset



Random Forest: Pros and Cons

Pros:

- Widely used, excellent prediction performance on many problems.
- Doesn't require careful normalization of features or extensive parameter tuning.
- Like decision trees, handles a mixture of feature types.
- Easily parallelized across multiple CPUs.

Cons:

- The resulting models are often difficult for humans to interpret.
- Like decision trees, random forests may not be a good choice for very high-dimensional tasks (e.g. text classifiers) compared to fast, accurate linear models.

Random Forests: RandomForestClassifier Key Parameters

- **`n_estimators`: number of trees to use in ensemble (default: 10).**
  - *Should be larger for larger datasets to reduce overfitting (but uses more computation).*
- **`max_features`: has a strong effect on performance. Influences the diversity of trees in the forest.**
  - *Default works well in practice, but adjusting may lead to some further gains.*
- **`max_depth`: controls the depth of each tree (default: None. Splits until all leaves are pure).**
- **`n_jobs`: How many cores to use in parallel during training.**

- **Choose a fixed setting for the random_state parameter if you need reproducible results.**

Applied Machine Learning

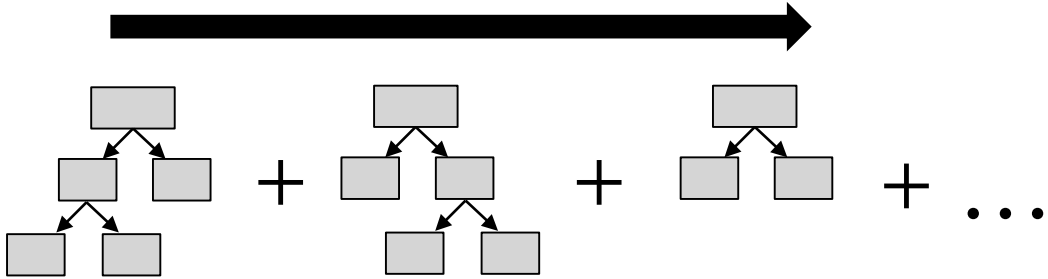
Gradient Boosted Decision Trees

Kevyn Collins-Thompson

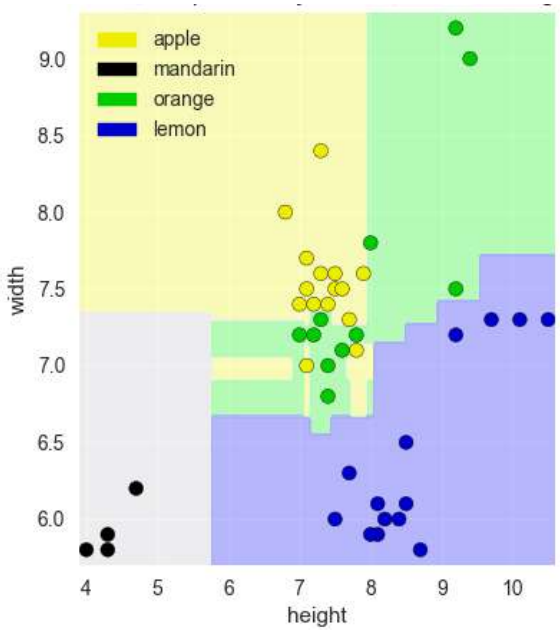
Associate Professor of Information & Computer Science
University of Michigan

Gradient Boosted Decision Trees (GBDT)

- Training builds a series of small decision trees.
- Each tree attempts to correct errors from the previous stage.



- The <u>learning rate</u> controls how hard each new tree tries to correct remaining mistakes from previous round.
  - High learning rate: more complex trees
  - Low learning rate: simpler trees



GBDT decision regions on two-feature fruits dataset

GBDT: Pros and Cons

Pros:

- Often best off-the-shelf accuracy on many problems.
- Using model for prediction requires only modest memory and is fast.
- Doesn't require careful normalization of features to perform well.
- Like decision trees, handles a mixture of feature types.

Cons:

- Like random forests, the models are often difficult for humans to interpret.
- Requires careful tuning of the learning rate and other parameters.
- Training can require significant computation.
- Like decision trees, not recommended for text classification and other problems with very high dimensional sparse features, for accuracy and computational cost reasons.

GBDT: GradientBoostingClassifier

Key Parameters

- `n_estimators`: sets # of small decision trees to use (weak learners) in the ensemble.
- `learning_rate`: controls emphasis on fixing errors from previous iteration.
- The above two are typically tuned together.
- `n_estimators` is adjusted first, to best exploit memory and CPUs during training, then other parameters.
- `max_depth` is typically set to a small value (e.g. 3-5) for most applications.

Applied Machine Learning

Neural networks

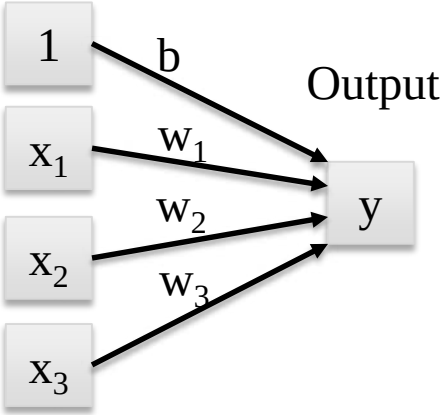
Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Review: Linear and Logistic Regression

Linear regression

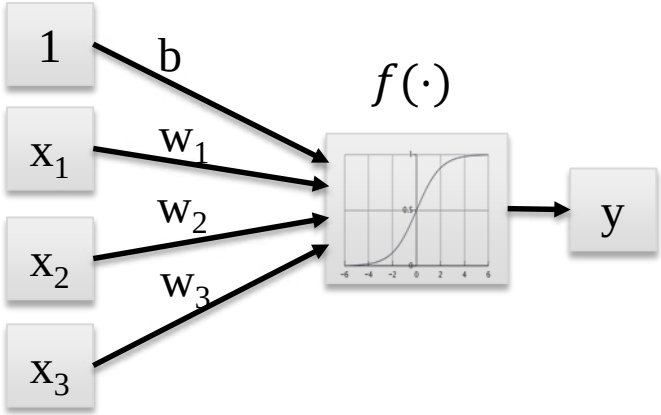
Input features



$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

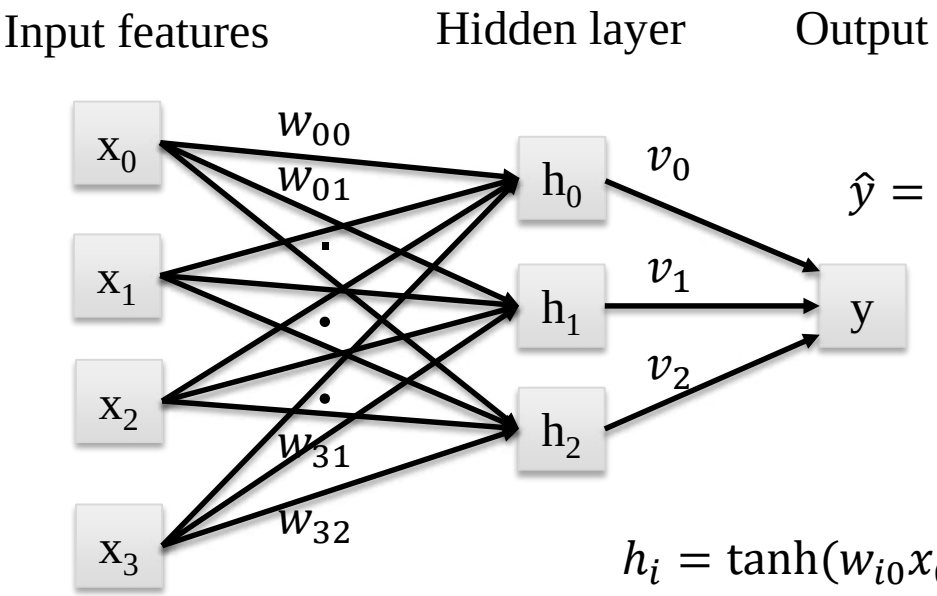
Logistic regression

Input features



$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

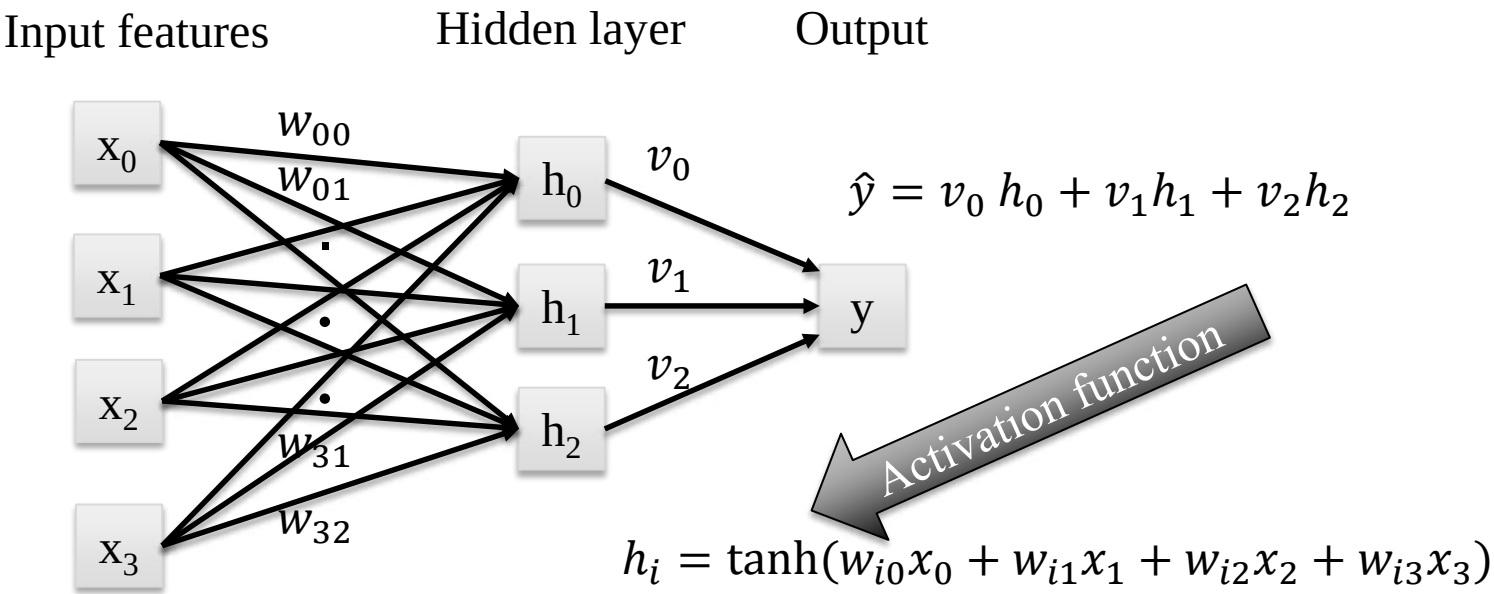
Multi-layer Perceptron with One Hidden Layer (and tanh activation function)



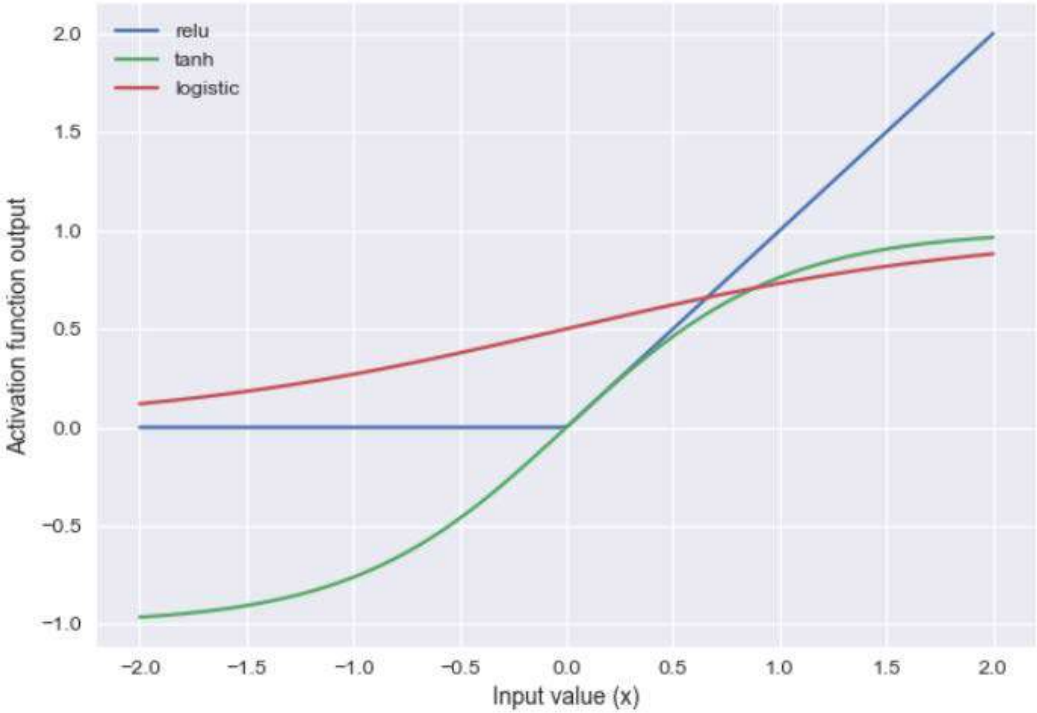
$$\hat{y} = v_0\, h_0 + v_1 h_1 + v_2 h_2$$

$$h_i = \tanh(w_{i0}x_0 + w_{i1}x_1 + w_{i2}x_2 + w_{i3}x_3)$$

Multi-layer Perceptron with One Hidden Layer (and tanh activation function)

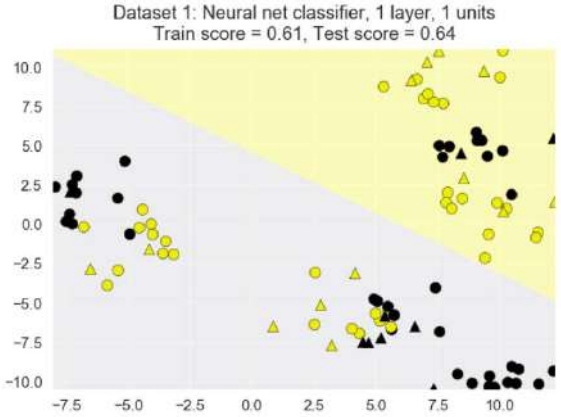


Activation Functions

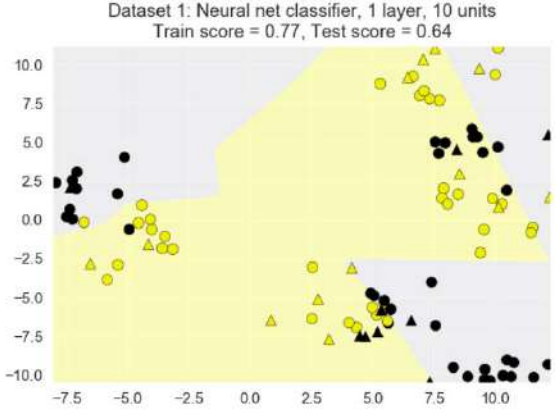


$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

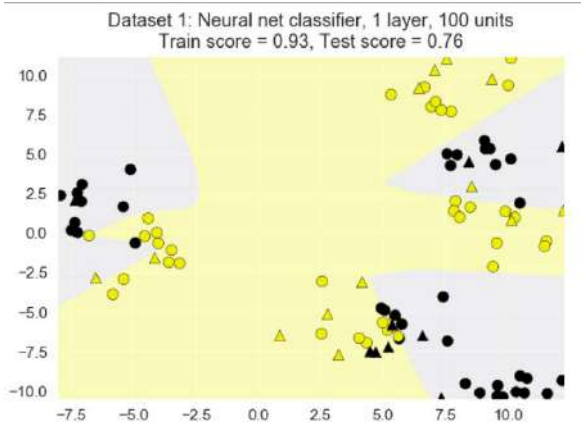
A single hidden layer network using 1, 10, or 100 units



1 layer, 1 unit

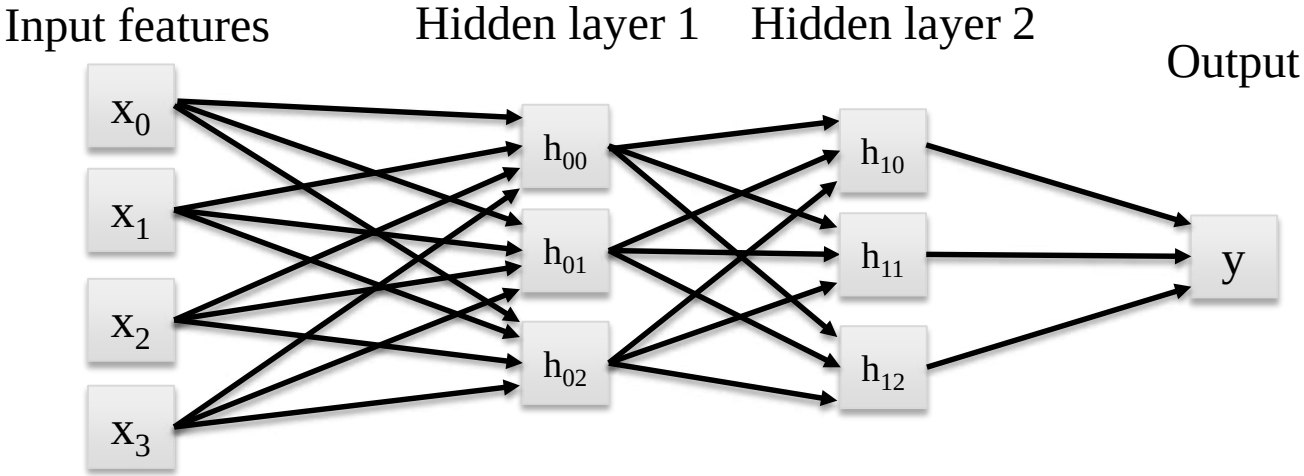


1 layer, 10 units

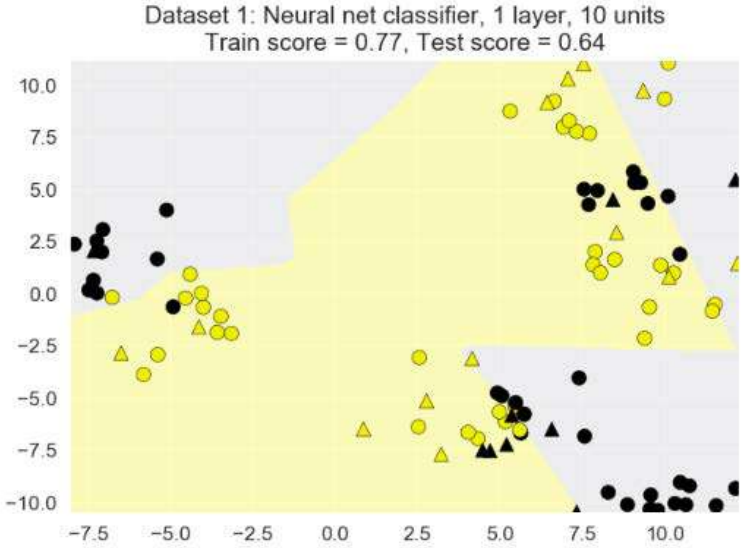


1 layer, 100 units

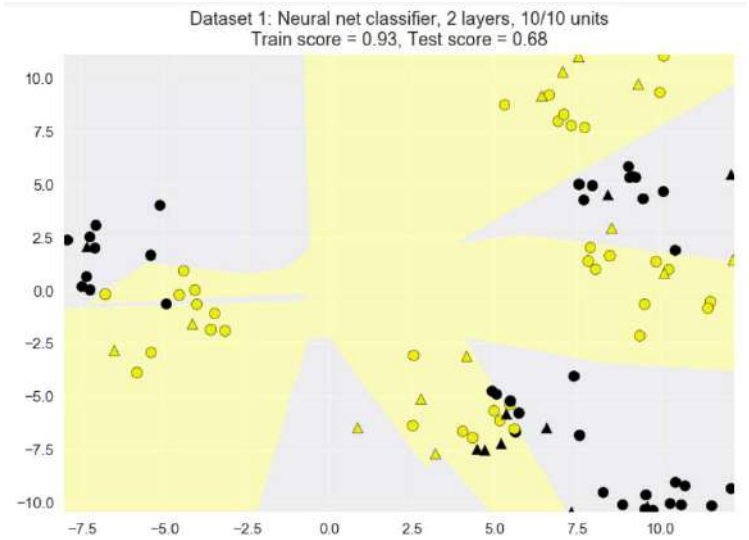
Multi-layer Perceptron with Two Hidden Layers



One vs Two Hidden Layers

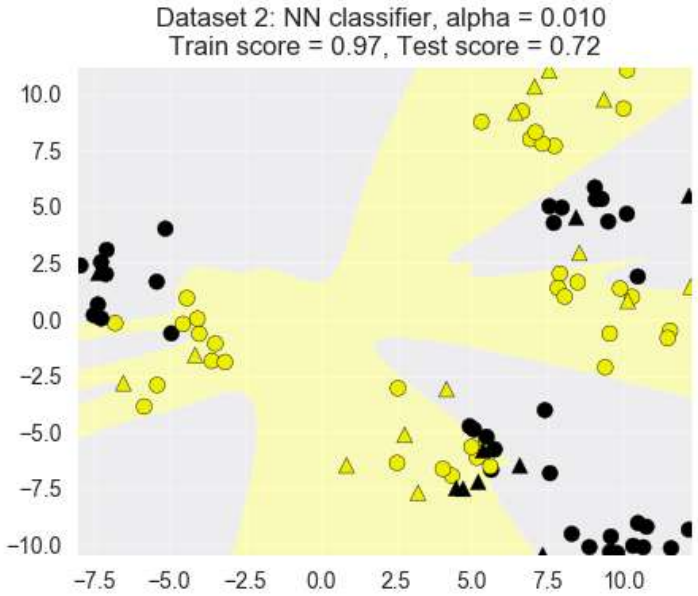


1 layer, 10 units

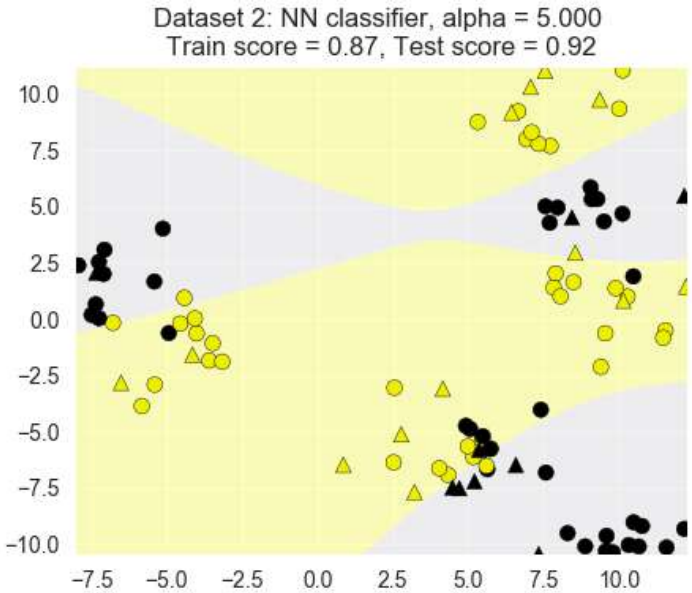


2 layers, (10, 10) units

L2 Regularization with the Alpha Parameter



alpha = 0.01

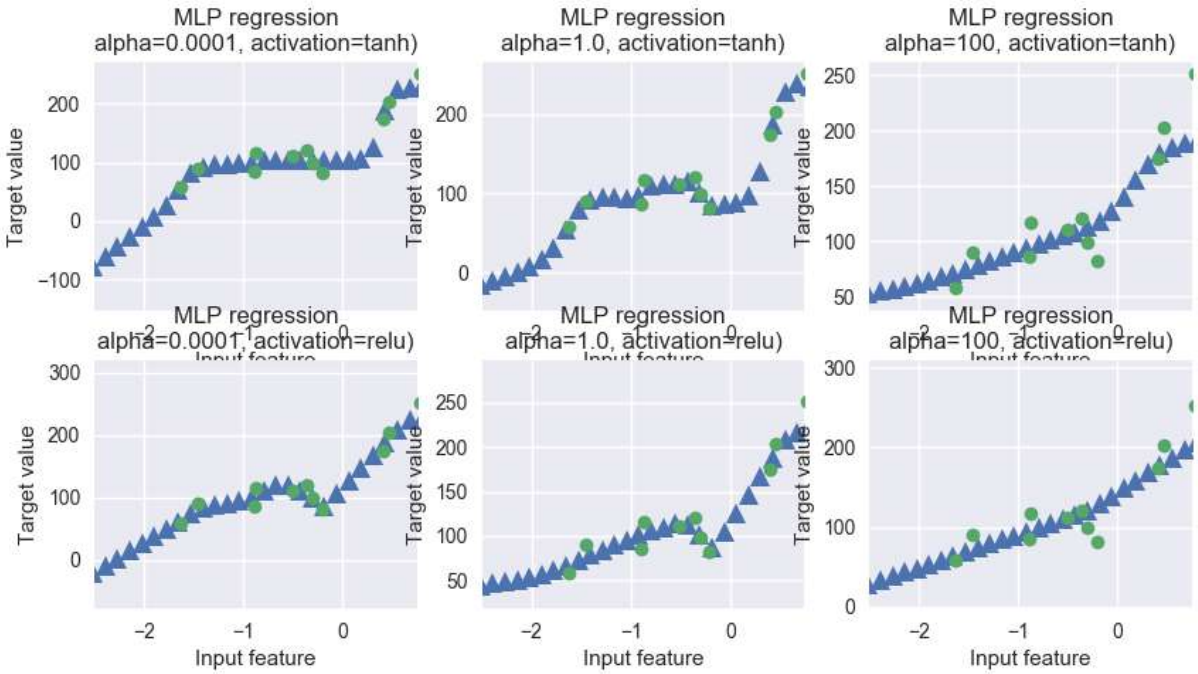


alpha = 5.0

Neural Network Regression with MLPRegressor

activation = 'tanh'

activation = 'relu'



alpha = 0.0001 alpha = 1.0 alpha = 100.0

Increasing regularization

Neural Networks: Pros and Cons

Pros:

- **They form the basis of state-of-the-art models and can be formed into advanced architectures that effectively capture complex features given enough data and computation.**

Cons:

- **Larger, more complex models require significant training time, data, and customization.**
- **Careful preprocessing of the data is needed.**
- **A good choice when the features are of similar types, but less so when features of very different types.**

Neural Nets: MLPClassifier and MLPRegressor Important pParameters

- **`hidden_layer_sizes`: sets the number of hidden layers (number of elements in list), and number of hidden units per layer (each list element).** *Default: (100).*
- **`alpha`: controls weight on the regularization penalty that shrinks weights to zero.** *Default: alpha = 0.0001.*
- **`activation`: controls the nonlinear function used for the activation function, including: `'relu'` (default), `'logistic'`, `'tanh'`.**

Applied Machine Learning

Deep Learning

(Optional overview)

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Deep Learning Summary

- **Deep learning architectures combine a sophisticated automatic feature extraction phase with a supervised learning phase.**
- **The feature extraction phase uses a hierarchy of multiple feature extraction layers.**
- **Starting from primitive, low-level features in the initial layer, each feature layer's output provides the input features to the next higher feature layer.**
- **All features are used in the final supervised learning model.**

An example of a simple deep learning architecture

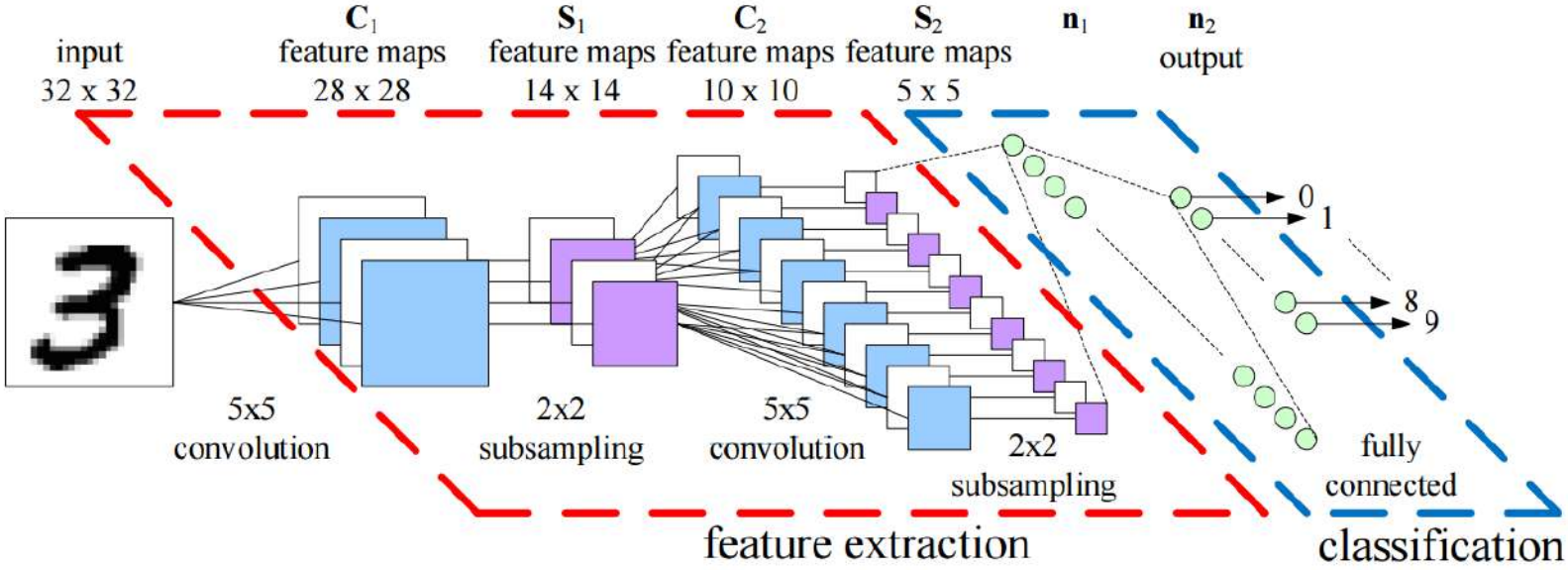
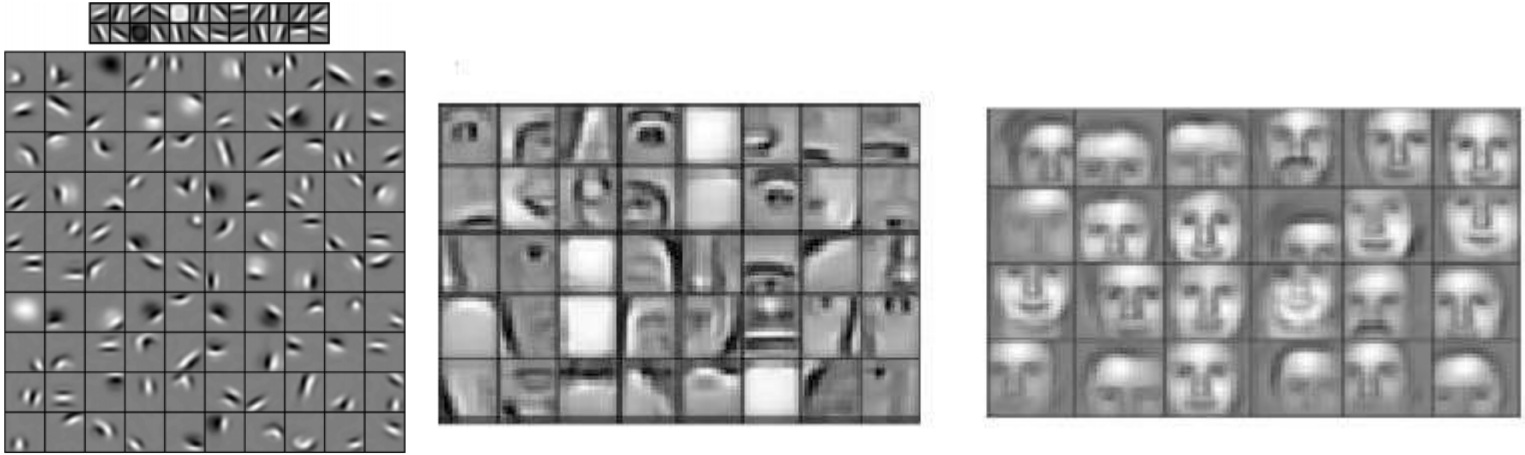


Image: M. Peemen, B. Mesman, and H. Corporaal. Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform. Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems, 2011.

Deep Learning



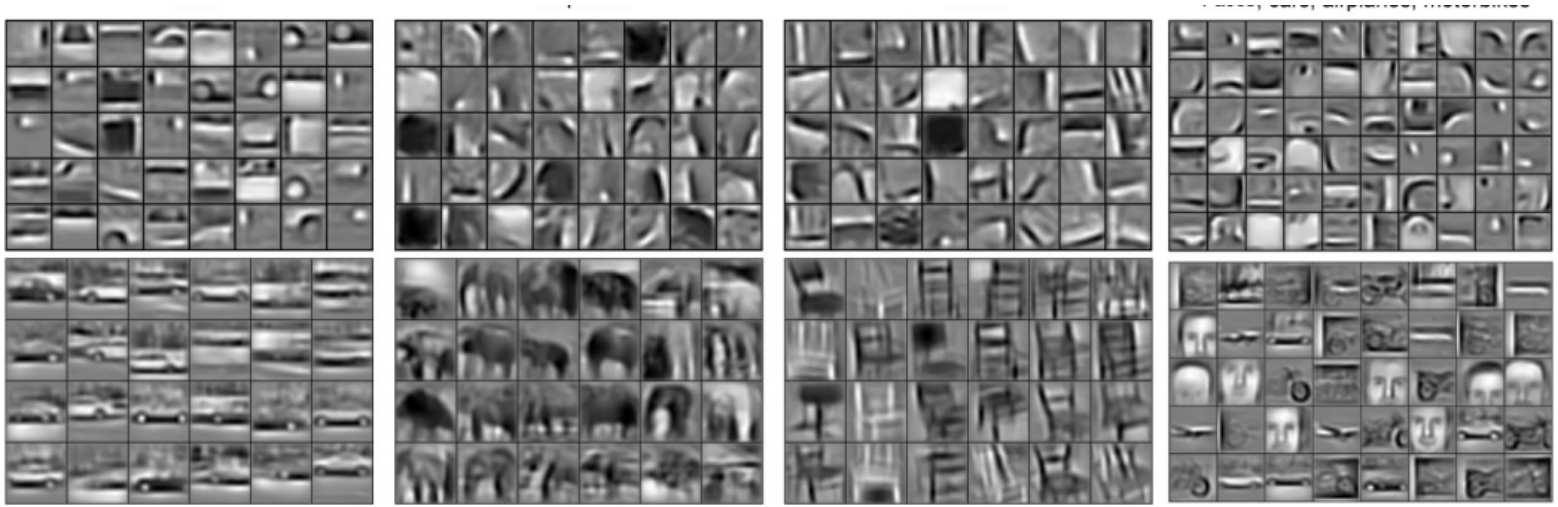
First, second, and third feature layer bases learned for faces

Image: Honglak Lee and colleagues (2011) from “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Deep Learning

Second layer

Third layer



Cars | Elephants | Chairs | Faces, cars, airplanes, motorbikes

Image: Honglak Lee and colleagues (2011) from “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Pros and Cons of Deep Learning

- **Pros:**
  - *Powerful: deep learning has achieved significant gains over other machine learning approaches on many difficult learning tasks, leading to state-of-the-art performance across many different domains.*
  - *Does effective automatic feature extraction, reducing the need for guesswork and heuristics on this key problem.*
  - *Current software provides flexible architectures that can be adapted for new domains fairly easily.*
- **Cons:**
  - *Can require huge amounts of training data.*
  - *Can require huge amounts of computing power.*
  - *Architectures can be complex and often must be highly tailored to a specific application.*
  - *The resulting models may not be easily interpretable.*

Deep learning software for Python

- **Keras** https://keras.io/
- **Lasagne** https://lasagne.readthedocs.io/en/latest/

- **TensorFlow** https://www.tensorflow.org/
- **Theano** http://deeplearning.net/software/theano/

- **Libraries support high-performance computation via GPUs.**

Applied Machine Learning

Data Leakage

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

Data Leakage

- **When the data you're using to train contains information about what you're trying to predict.**
- **Introducing information about the target during training that would not legitimately be available during actual use.**
- **Obvious examples:**
  - *Including the label to be predicted as a feature*
  - *Including test data with training data*
- **If your model performance is too good to be true, it probably is and likely due to "giveaway" features.**

More subtle examples of data leakage

- **Prediction target: will user stay on a site, or leave?**
  - *Giveaway feature: total session length, based on information about <u>future</u> page visits*
- **Predicting if a user on a financial site is likely to open an account**
  - *An account number field that's only filled in once the user does open an account.*
- **Diagnostic test to predict a medical condition**
  - *The existing patient dataset contains a binary variable that happens to mark whether they had surgery for that condition.*
  - *Combinations of missing diagnosis codes that are not be available while the patient's condition was still being studied.*
  - *The patient ID could contain information about specific diagnosis paths (e.g. for routine visit vs specialist).*
- **Any of these leaked features is highly predictive of the target, but not legitimately available at the time prediction needs to be done.**

Other examples of data leakage

Leakage in training data:

- Performing data preprocessing using parameters or results from analyzing the entire dataset: Normalizing and rescaling, detecting and removing outliers, estimating missing values, feature selection.
- Time-series datasets: using records from the future when computing features for the current prediction.
- Errors in data values/gathering or missing variable indicators (e.g. the special value 999) can encode information about missing data that reveals information about the future.

Leakage in features:

- Removing variables that are not legitimate without also removing variables that encode the same or related information (e.g. diagnosis info may still exist in patient ID).
- Reversing of intentional randomization or anonymization that reveals specific information about e.g. users not legitimately available in actual use.

Any of the above could be present in any external data joined to the training set.

Detecting data leakage

- **Before building the model**
  - *Exploratory data analysis to find surprises in the data*
  - *Are there features very highly correlated with the target value?*
- **After building the model**
  - *Look for surprising feature behavior in the fitted model.*
  - *Are there features with very high weights, or high information gain?*
  - *Simple rule-based models like decision trees can help with features like account numbers, patient IDs*
  - *Is overall model performance surprisingly good compared to known results on the same dataset, or for similar problems on similar datasets?*
- **Limited real-world deployment of the trained model**
  - *Potentially expensive in terms of development time, but more realistic*
  - *Is the trained model generalizing well to new data?*

Minimizing Data Leakage

- **Perform data preparation within each cross-validation fold separately**
  - *Scale/normalize data, perform feature selection, etc. within each fold separately, not using the entire dataset.*
  - *For any such parameters estimated on the training data, you must use those same parameters to prepare data on the corresponding held-out test fold.*
- **With time series data, use a timestamp cutoff**
  - *The cutoff value is set to the specific time point where prediction is to occur using current and past records.*
  - *Using a cutoff time will make sure you aren't accessing any data records that were gathered after the prediction time, i.e. in the future.*
- **Before any work with a new dataset, split off a final test validation dataset**
  - *… if you have enough data*
  - *Use this final test dataset as the very last step in your validation*
  - *Helps to check the true generalization performance of any trained models*

Module 4

September 22, 2021

---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Applied Machine Learning: Module 4 (Supervised Learning, Part II)

1.1 Preamble and Datasets

1.1.1 ye sab pehle kiya hai

In [1]:
```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification, make_blobs
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_breast_cancer
from adspy_shared_utilities import load_crime_dataset


cmap_bold = ListedColormap(['#FFFF00', '#00FF00', '#0000FF','#000000'])

# fruits dataset
fruits = pd.read_table('readonly/fruit_data_with_colors.txt')

feature_names_fruits = ['height', 'width', 'mass', 'color_score']
X_fruits = fruits[feature_names_fruits] #filter mara hai
y_fruits = fruits['fruit_label']
target_names_fruits = ['apple', 'mandarin', 'orange', 'lemon']

X_fruits_2d = fruits[['height', 'width']]
y_fruits_2d = fruits['fruit_label']
```

```
# synthetic dataset for simple regression
from sklearn.datasets import make_regression
plt.figure()
plt.title('Sample regression problem with one input variable')
X_R1, y_R1 = make_regression(n_samples = 100, n_features=1, n_informative=1
                             noise = 30, random_state=0)
plt.scatter(X_R1, y_R1, marker= 'o', s=50)
plt.show()

# synthetic dataset for more complex regression
from sklearn.datasets import make_friedman1
plt.figure()
plt.title('Complex regression problem with one input variable')
X_F1, y_F1 = make_friedman1(n_samples = 100, n_features = 7,
                            random_state=0)

plt.scatter(X_F1[:, 2], y_F1, marker= 'o', s=50)
plt.show()

# synthetic dataset for classification (binary)
plt.figure()
plt.title('Sample binary classification problem with two informative featur
X_C2, y_C2 = make_classification(n_samples = 100, n_features=2,
                                 n_redundant=0, n_informative=2,
                                 n_clusters_per_class=1, flip_y = 0.1,
                                 class_sep = 0.5, random_state=0)
plt.scatter(X_C2[:, 0], X_C2[:, 1], marker= 'o',
           c=y_C2, s=50, cmap=cmap_bold)
plt.show()

# more difficult synthetic dataset for classification (binary)
# with classes that are not linearly separable
X_D2, y_D2 = make_blobs(n_samples = 100, n_features = 2,
                        centers = 8, cluster_std = 1.3,
                        random_state = 4)
y_D2 = y_D2 % 2
plt.figure()
plt.title('Sample binary classification problem with non-linearly separable
plt.scatter(X_D2[:,0], X_D2[:,1], c=y_D2,
           marker= 'o', s=50, cmap=cmap_bold)
plt.show()

# Breast cancer dataset for classification
cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# Communities and Crime dataset
```

```
(X_crime, y_crime) = load_crime_dataset()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.2 Naive Bayes classifiers

1.2.1 note that naicve bayers has a feature to train partially if the entire model does not fit in memory

```
In [2]: from sklearn.naive_bayes import GaussianNB
        from adspy_shared_utilities import plot_class_regions_for_classifier

        X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2, random_stat

        nbclf = GaussianNB().fit(X_train, y_train)
        plot_class_regions_for_classifier(nbclf, X_train, y_train, X_test, y_test,
                                          'Gaussian Naive Bayes classifier: Dataset
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
In [3]: X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                            random_state=0)

        nbclf = GaussianNB().fit(X_train, y_train)
        plot_class_regions_for_classifier(nbclf, X_train, y_train, X_test, y_test,
                                          'Gaussian Naive Bayes classifier: Dataset
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.2.2 Application to a real-world dataset

1.2.3 competitive OP!

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ran

        nbclf = GaussianNB().fit(X_train, y_train)
        print('Breast cancer dataset')
        print('Accuracy of GaussianNB classifier on training set: {:.2f}'
             .format(nbclf.score(X_train, y_train)))
        print('Accuracy of GaussianNB classifier on test set: {:.2f}'
             .format(nbclf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of GaussianNB classifier on training set: 0.95
Accuracy of GaussianNB classifier on test set: 0.94
```

1.3 Ensembles of Decision Trees

1.3.1 Random forests

```
In [5]: from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import train_test_split
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo

        X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2,
                                                            random_state = 0)
        # more difficult synthetic dataset for classification (binary)
        # with classes that are not linearly separable


        fig, subaxes = plt.subplots(1, 1, figsize=(6, 6))

        clf = RandomForestClassifier().fit(X_train, y_train)
        title = 'Random Forest Classifier, complex binary dataset, default settings
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train, X_test,
                                                  y_test, title, subaxes)

        plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.3.2 Random forest: Fruit dataset

```
In [6]: from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import train_test_split
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo

        X_train, X_test, y_train, y_test = train_test_split(X_fruits.as_matrix(),
                                                            y_fruits.as_matrix(),
                                                            random_state = 0)
        fig, subaxes = plt.subplots(6, 1, figsize=(6, 32))

        title = 'Random Forest, fruits dataset, default settings'
        pair_list = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]

        for pair, axis in zip(pair_list, subaxes):
            X = X_train[:, pair]
            y = y_train

            clf = RandomForestClassifier().fit(X, y)
            plot_class_regions_for_classifier_subplot(clf, X, y, None,
                                                      None, title, axis,
                                                      target_names_fruits)

            axis.set_xlabel(feature_names_fruits[pair[0]])
            axis.set_ylabel(feature_names_fruits[pair[1]])

        plt.tight_layout()
        plt.show()

        clf = RandomForestClassifier(n_estimators = 10,
                                     random_state=0).fit(X_train, y_train)

        print('Random Forest, Fruit dataset, default settings')
        print('Accuracy of RF classifier on training set: {:.2f}'
             .format(clf.score(X_train, y_train)))
        print('Accuracy of RF classifier on test set: {:.2f}'
             .format(clf.score(X_test, y_test)))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Random Forest, Fruit dataset, default settings
Accuracy of RF classifier on training set: 1.00
```

```
Accuracy of RF classifier on test set: 0.80
```

Random Forests on a real-world dataset

```
In [7]: from sklearn.ensemble import RandomForestClassifier

        X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ran

        clf = RandomForestClassifier(max_features = 8, random_state = 0)
        #observe that max_features have been set to 8 soecific to this exapmle
        clf.fit(X_train, y_train)

        print('Breast cancer dataset')
        print('Accuracy of RF classifier on training set: {:.2f}'
             .format(clf.score(X_train, y_train)))
        print('Accuracy of RF classifier on test set: {:.2f}'
             .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of RF classifier on training set: 1.00
Accuracy of RF classifier on test set: 0.99
```

1.3.3 Gradient-boosted decision trees

```
In [8]: from sklearn.ensemble import GradientBoostingClassifier
        from sklearn.model_selection import train_test_split
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo

        X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_stat
        fig, subaxes = plt.subplots(1, 1, figsize=(6, 6))

        clf = GradientBoostingClassifier().fit(X_train, y_train)
        title = 'GBDT, complex binary dataset, default settings'
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train, X_test,
                                                  y_test, title, subaxes)

        plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Gradient boosted decision trees on the fruit dataset

```
In [9]: X_train, X_test, y_train, y_test = train_test_split(X_fruits.as_matrix(),
                                                            y_fruits.as_matrix(),
                                                            random_state = 0)
        fig, subaxes = plt.subplots(6, 1, figsize=(6, 32))

        pair_list = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]

        for pair, axis in zip(pair_list, subaxes):
            X = X_train[:, pair]
            y = y_train

            clf = GradientBoostingClassifier().fit(X, y)
            plot_class_regions_for_classifier_subplot(clf, X, y, None,
                                                      None, title, axis,
                                                      target_names_fruits)

            axis.set_xlabel(feature_names_fruits[pair[0]])
            axis.set_ylabel(feature_names_fruits[pair[1]])

        plt.tight_layout()
        plt.show()
        clf = GradientBoostingClassifier().fit(X_train, y_train)

        print('GBDT, Fruit dataset, default settings')
        print('Accuracy of GBDT classifier on training set: {:.2f}'
             .format(clf.score(X_train, y_train)))
        print('Accuracy of GBDT classifier on test set: {:.2f}'
             .format(clf.score(X_test, y_test)))
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


GBDT, Fruit dataset, default settings
Accuracy of GBDT classifier on training set: 1.00
Accuracy of GBDT classifier on test set: 0.80
```

Gradient-boosted decision trees on a real-world dataset

```
In [10]: from sklearn.ensemble import GradientBoostingClassifier

         X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ra

         clf = GradientBoostingClassifier(random_state = 0)
         clf.fit(X_train, y_train)
```

```
print('Breast cancer dataset (learning_rate=0.1, max_depth=3)')
print('Accuracy of GBDT classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}\n'
     .format(clf.score(X_test, y_test)))

clf = GradientBoostingClassifier(learning_rate = 0.01, max_depth = 2, ranc
clf.fit(X_train, y_train)

print('Breast cancer dataset (learning_rate=0.01, max_depth=2)')
print('Accuracy of GBDT classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of GBDT classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset (learning_rate=0.1, max_depth=3)
Accuracy of GBDT classifier on training set: 1.00
Accuracy of GBDT classifier on test set: 0.96

Breast cancer dataset (learning_rate=0.01, max_depth=2)
Accuracy of GBDT classifier on training set: 0.97
Accuracy of GBDT classifier on test set: 0.97
```

1.4 Neural networks

Activation functions

```
In [11]: xrange = np.linspace(-2, 2, 200)

         plt.figure(figsize=(7,6))

         plt.plot(xrange, np.maximum(xrange, 0), label = 'relu')
         plt.plot(xrange, np.tanh(xrange), label = 'tanh')
         plt.plot(xrange, 1 / (1 + np.exp(-xrange)), label = 'logistic')
         plt.legend()
         plt.title('Neural network activation functions')
         plt.xlabel('Input value (x)')
         plt.ylabel('Activation function output')

         plt.show()
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

1.4.1 Neural networks: Classification

Synthetic dataset 1: single hidden layer

```
In [12]: from sklearn.neural_network import MLPClassifier
         from adspy_shared_utilities import plot_class_regions_for_classifier_subpl

         X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta

         fig, subaxes = plt.subplots(3, 1, figsize=(6,18))

         for units, axis in zip([1, 10, 100], subaxes):
             nnclf = MLPClassifier(hidden_layer_sizes = [units], solver='lbfgs',
                                   random_state = 0).fit(X_train, y_train)

             title = 'Dataset 1: Neural net classifier, 1 layer, {} units'.format(u

             plot_class_regions_for_classifier_subplot(nnclf, X_train, y_train,
                                                       X_test, y_test, title, axis)
             plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Synthetic dataset 1: two hidden layers

```
In [13]: from adspy_shared_utilities import plot_class_regions_for_classifier

         X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta

         nnclf = MLPClassifier(hidden_layer_sizes = [10, 10], solver='lbfgs',
                               random_state = 0).fit(X_train, y_train)

         plot_class_regions_for_classifier(nnclf, X_train, y_train, X_test, y_test,
                                           'Dataset 1: Neural net classifier, 2 laye
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Regularization parameter: alpha

1.4.2 5 wale ne best kaam kiya

```
In [14]: X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta

         fig, subaxes = plt.subplots(4, 1, figsize=(6, 23))

         for this_alpha, axis in zip([0.01, 0.1, 1.0, 5.0], subaxes):
             nnclf = MLPClassifier(solver='lbfgs', activation = 'tanh',
                                   alpha = this_alpha,
                                   hidden_layer_sizes = [100, 100],
                                   random_state = 0).fit(X_train, y_train) #ALL THES

             title = 'Dataset 2: NN classifier, alpha = {:.3f} '.format(this_alpha)

             plot_class_regions_for_classifier_subplot(nnclf, X_train, y_train,
                                                       X_test, y_test, title, axis)
             plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

The effect of different choices of activation function

```
In [15]: X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_sta

         fig, subaxes = plt.subplots(3, 1, figsize=(6,18))

         for this_activation, axis in zip(['logistic', 'tanh', 'relu'], subaxes):
             nnclf = MLPClassifier(solver='lbfgs', activation = this_activation,
                                   alpha = 0.1, hidden_layer_sizes = [10, 10],
                                   random_state = 0).fit(X_train, y_train)

             title = 'Dataset 2: NN classifier, 2 layers 10/10, {} \
         activation function'.format(this_activation)

             plot_class_regions_for_classifier_subplot(nnclf, X_train, y_train,
                                                       X_test, y_test, title, axis)
             plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.4.3 Neural networks: Regression

1.4.4 CLEARLY MORE SMOOTHER WITH hIGHER ALPHA

In [16]:
```
from sklearn.neural_network import MLPRegressor

fig, subaxes = plt.subplots(2, 3, figsize=(11,8), dpi=70)

X_predict_input = np.linspace(-3, 3, 50).reshape(-1,1)

X_train, X_test, y_train, y_test = train_test_split(X_R1[0::5], y_R1[0::5]


#ACTIVATION FUNCTION AND ALPHA CHANGE KAR RAHE HAI
for thisaxisrow, thisactivation in zip(subaxes, ['tanh', 'relu']):
    for thisalpha, thisaxis in zip([0.0001, 1.0, 100], thisaxisrow):
        mlpreg = MLPRegressor(hidden_layer_sizes = [100,100], activation =
                              alpha = thisalpha, solver = 'lbfgs').fit(X_tr
        y_predict_output = mlpreg.predict(X_predict_input)
        thisaxis.set_xlim([-2.5, 0.75])
        thisaxis.plot(X_predict_input, y_predict_output, '^', markersize =
        thisaxis.plot(X_train, y_train, 'o')
        thisaxis.set_xlabel('Input feature')
        thisaxis.set_ylabel('Target value')
        thisaxis.set_title('MLP regression\nalpha={}, activation={})'
                           .format(thisalpha, thisactivation))
        plt.tight_layout()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Application to real-world dataset for classification

yaha scaling kar rahe hai

In [17]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()

X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ra
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = MLPClassifier(hidden_layer_sizes = [100, 100], alpha = 5.0,
```

```
                                random_state = 0, solver='lbfgs').fit(X_train_scaled, y

print('Breast cancer dataset')
print('Accuracy of NN classifier on training set: {:.2f}'
     .format(clf.score(X_train_scaled, y_train)))
print('Accuracy of NN classifier on test set: {:.2f}'
     .format(clf.score(X_test_scaled, y_test)))
```

```
Breast cancer dataset
Accuracy of NN classifier on training set: 0.98
Accuracy of NN classifier on test set: 0.97
```

In [ ]:

Applied Machine Learning

Unsupervised machine learning

Kevyn Collins-Thompson

Associate Professor of Information & Computer Science
University of Michigan

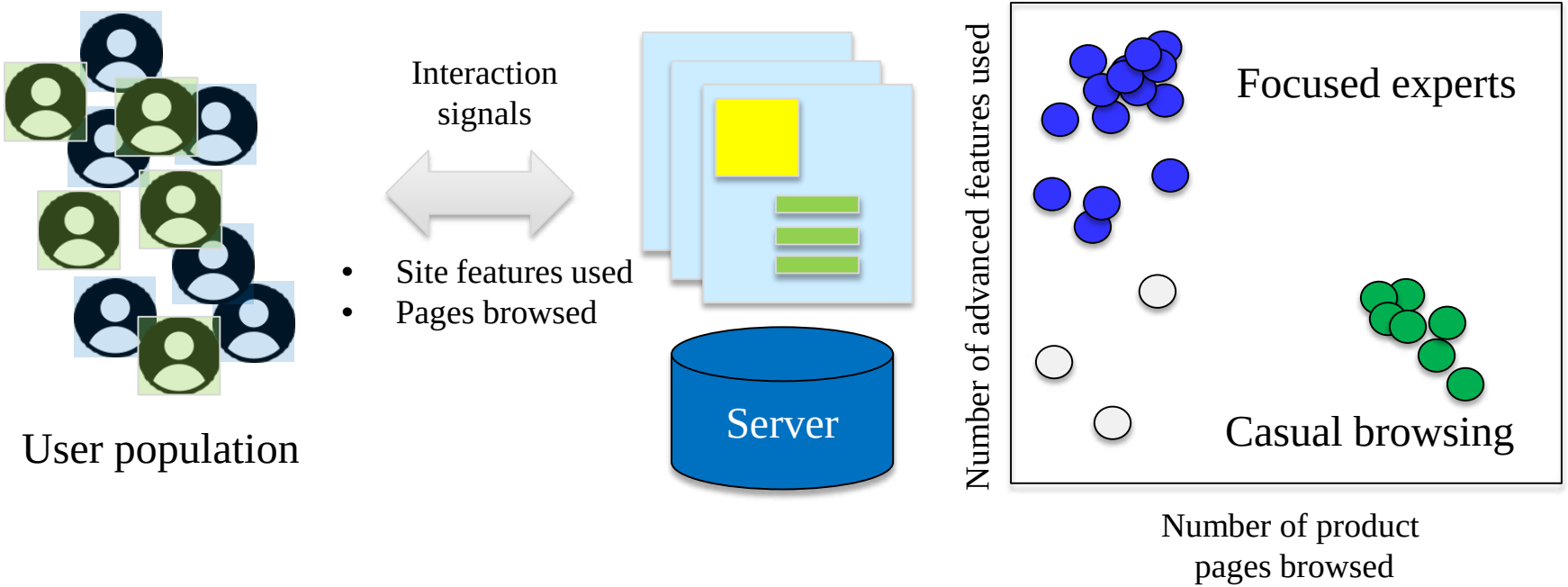
Introduction: Unsupervised Learning

- **Unsupervised learning involves tasks that operate on datasets <u>without</u> labeled responses or target values.**
- **Instead, the goal is to capture interesting structure or information.**

Applications of unsupervised learning:

- **Visualize structure of a complex dataset.**
- **Density estimation to predict probabilities of events.**
- **Compress and summarize the data.**
- **Extract features for supervised learning.**
- **Discover important clusters or outliers.**

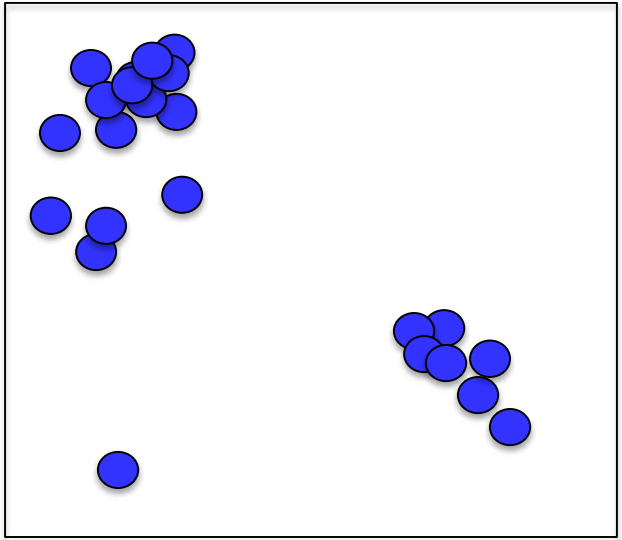
Web Clustering Example



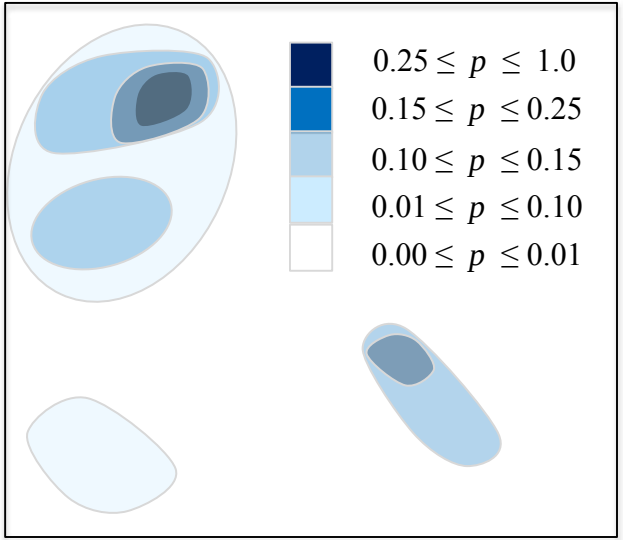
Two major types of unsupervised learning methods

- **Transformations**
  - *Processes that extract or compute information*
- **Clustering**
  - *Find groups in the data*
  - *Assign every point in the dataset to one of the groups*

Transformations: Density Estimation

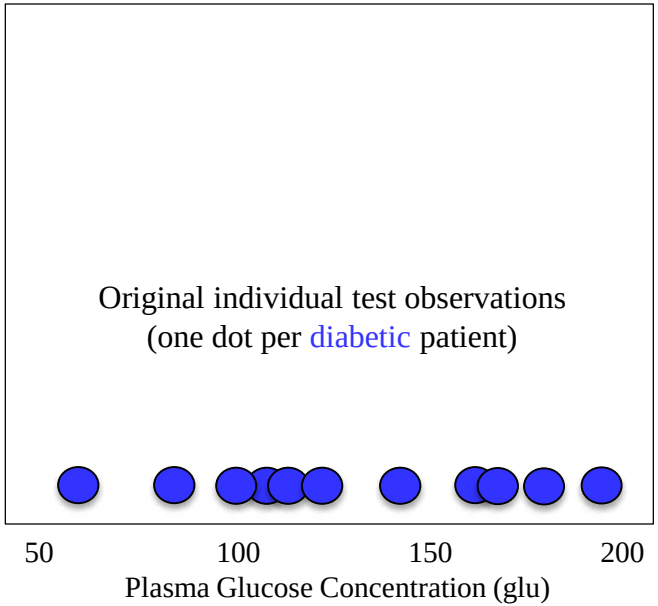


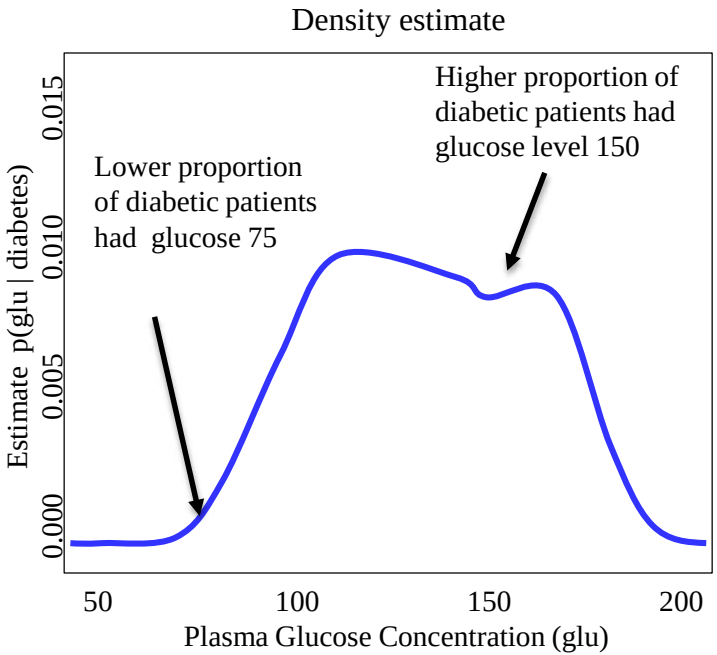
Individual measurements



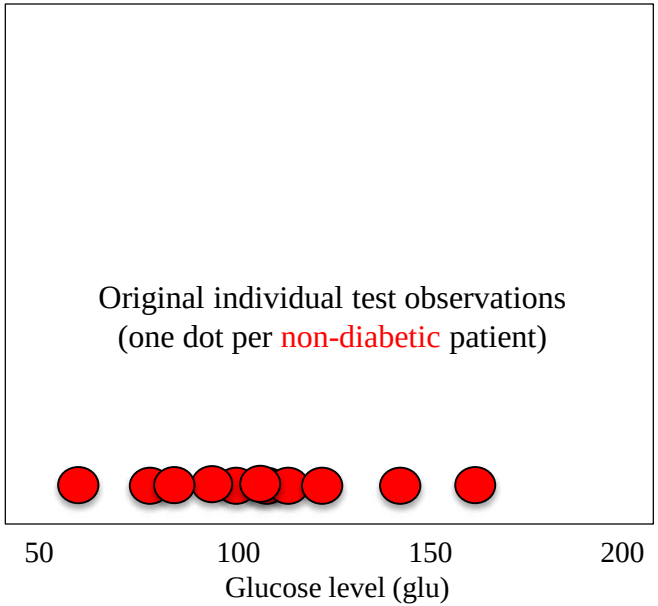
Density estimate
(Estimated probability *p* of observing a measurement at a given location)

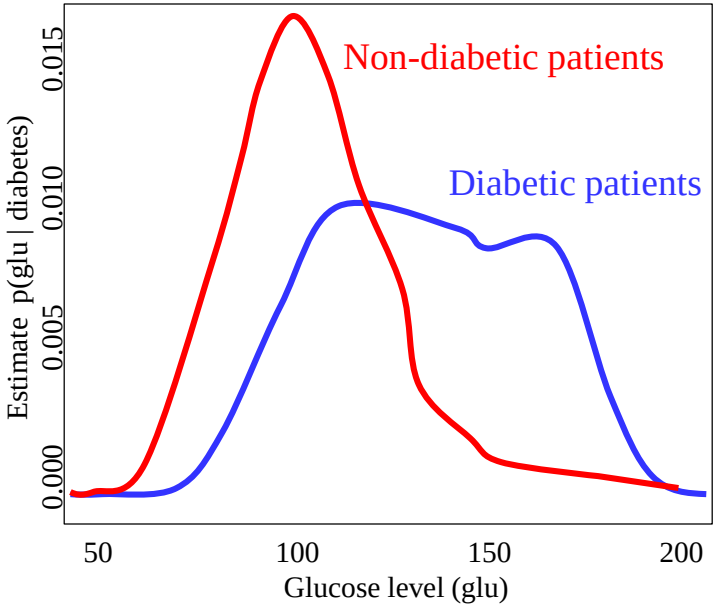
Density Estimation Example



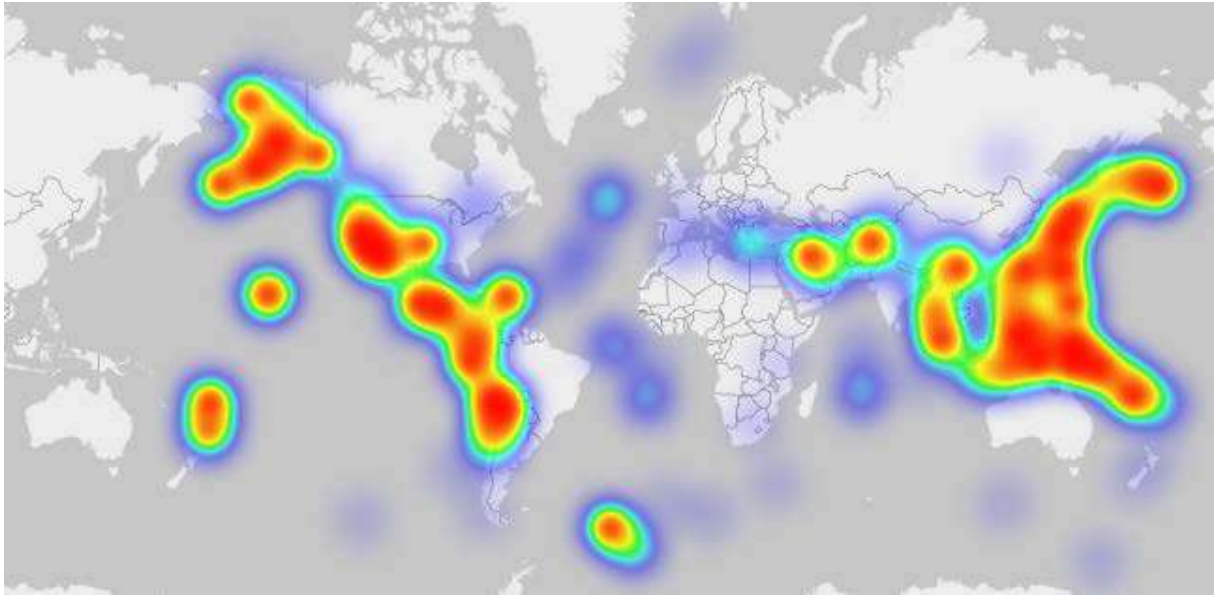


Density Estimation Example





Kernel Density Example

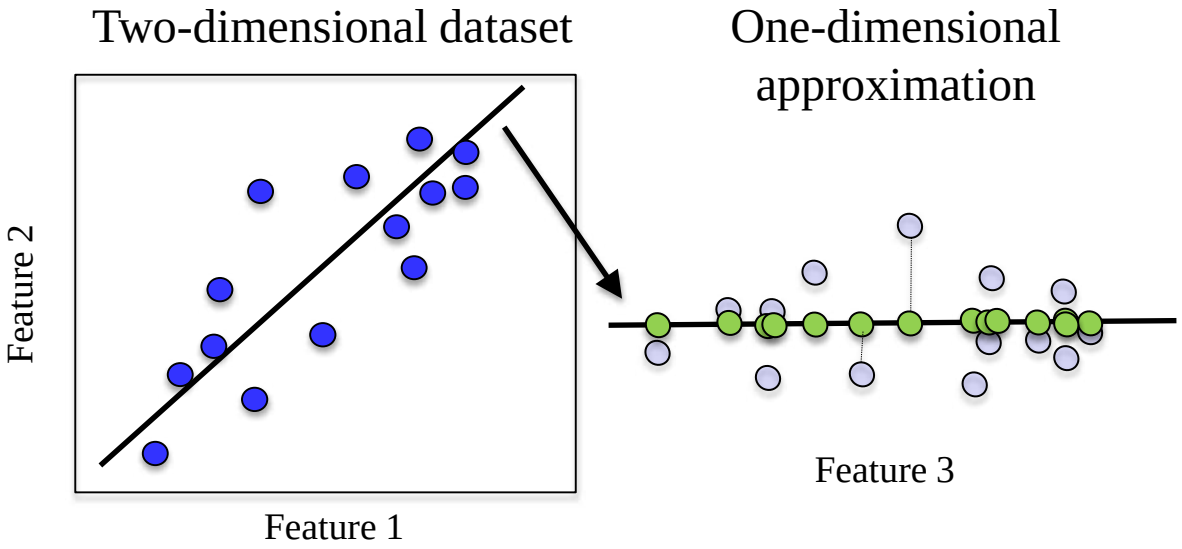


Recent global earthquake activity (U.S. Geological Survey data)

Source: http://www.digital-geography.com/csv-heatmap-leaflet/

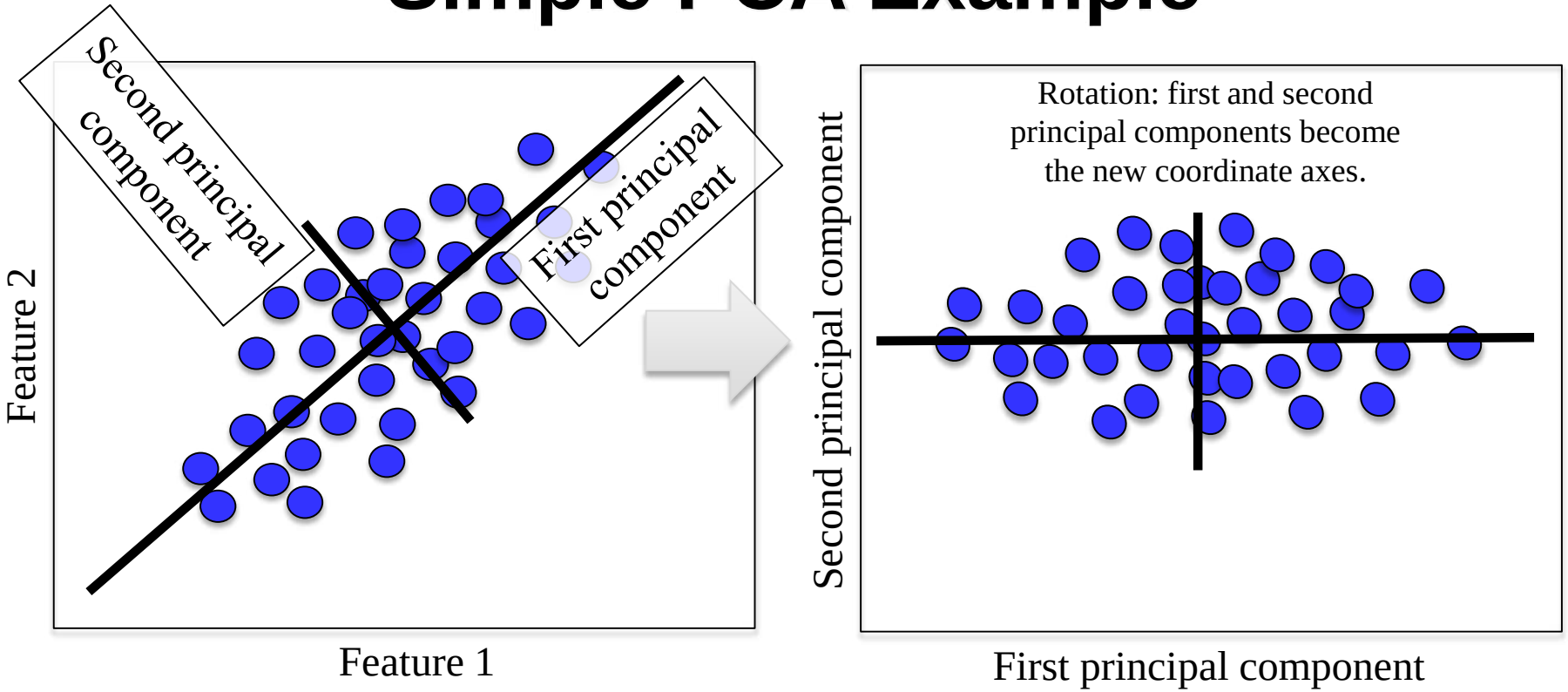
Dimensionality Reduction

- **Finds an approximate version of your dataset using fewer features**
- **Used for exploring and visualizing a dataset to understand grouping or relationships**
- **Often visualized using a 2-dimensional scatterplot**
- **Also used for compression, finding features for supervised learning**



The one-dimensional approximation is obtained by projecting the original points onto the diagonal line and using their position on that line as the new single feature.

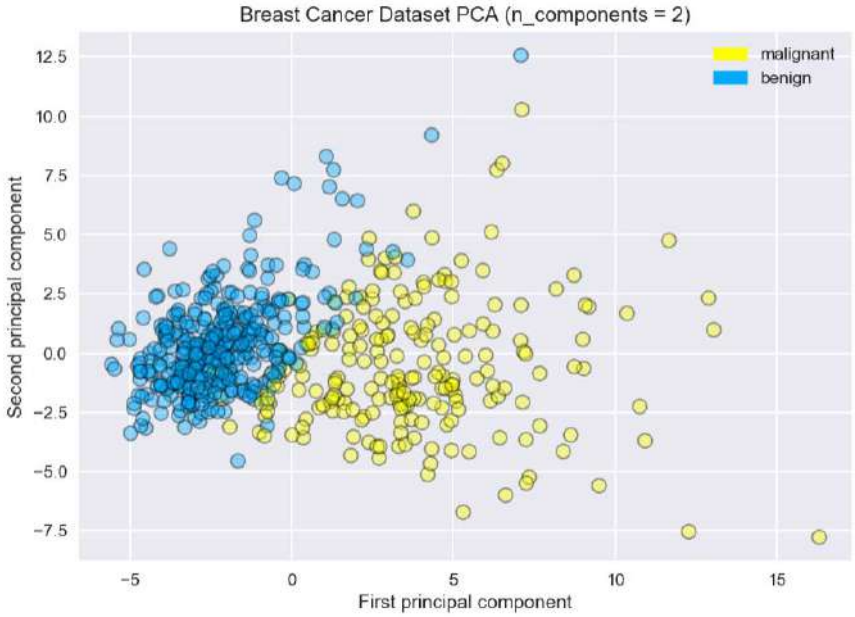
Simple PCA Example



Dimensionality Reduction with PCA in scikit-learn

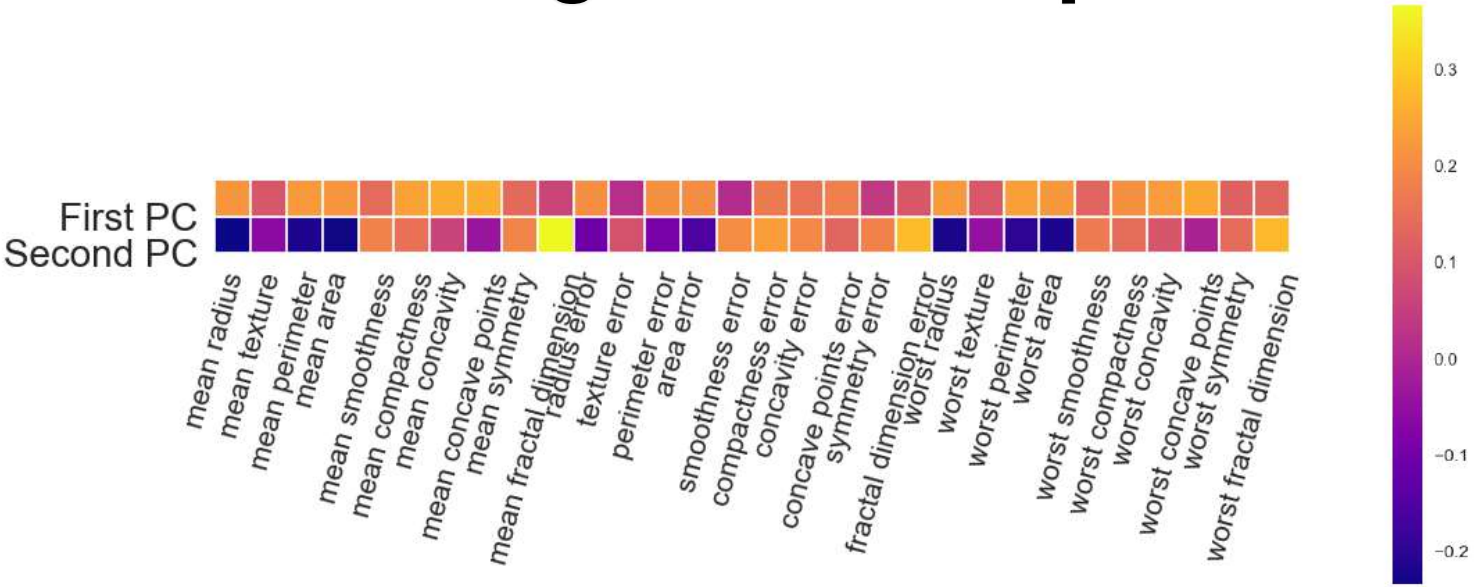
```
# PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# each feature should be centered (zero mean) and with unit variance
X_normalized = StandardScaler().fit(X_cancer).transform(X_cancer)

pca = PCA(n_components = 2).fit(X_normalized)

X_pca = pca.transform(X_normalized)
print(X_cancer.shape, X_pca.shape)
```

```
(569, 30) (569, 2)
```

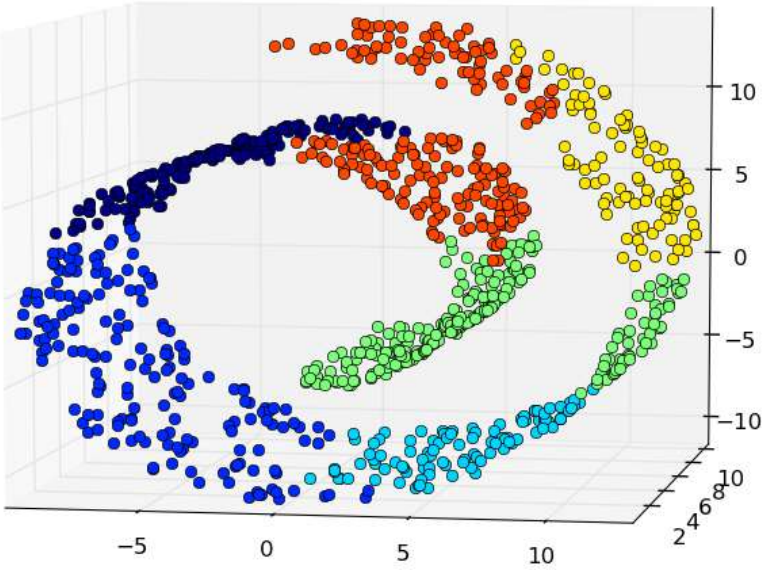



```
from adspy_shared_utilities import plot_labelled_scatter
plot_labelled_scatter(X_pca, y_cancer, ['malignant', 'benign'])

plt.xlabel('First principal component')
plt.ylabel('Second principal component')
plt.title("Breast Cancer Dataset PCA (n_components = 2)")
```

Visualizing PCA Components



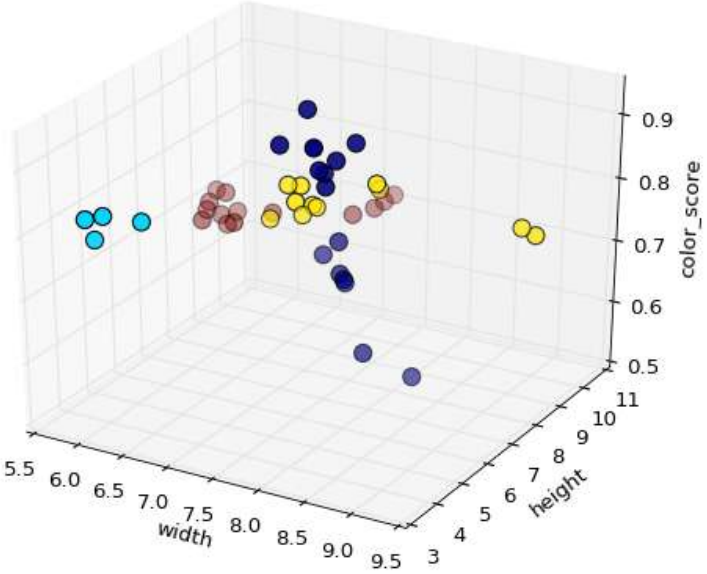
The ”Swiss Roll" Dataset



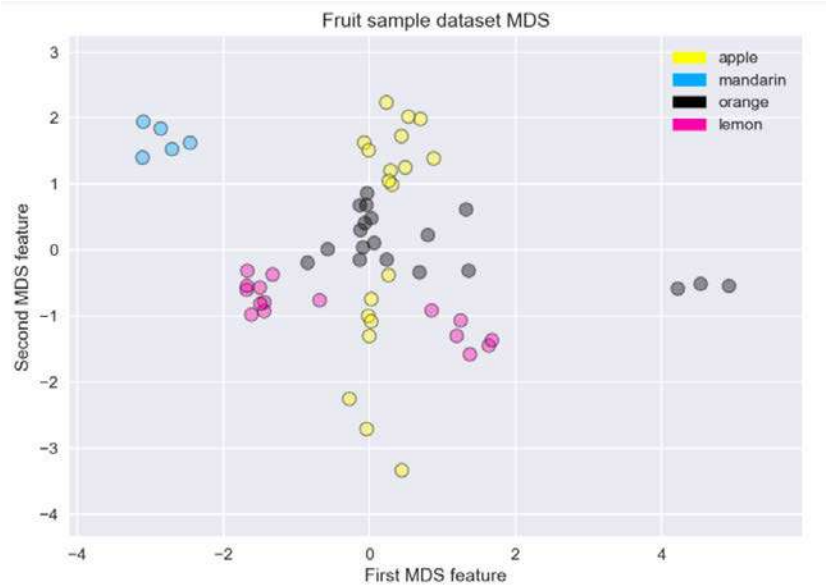
sklearn.datasets.make_swiss_roll(n_samples=1500, noise=0.05)

See: http://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering

Multidimensional scaling (MDS) attempts to find a distance-preserving low-dimensional projection



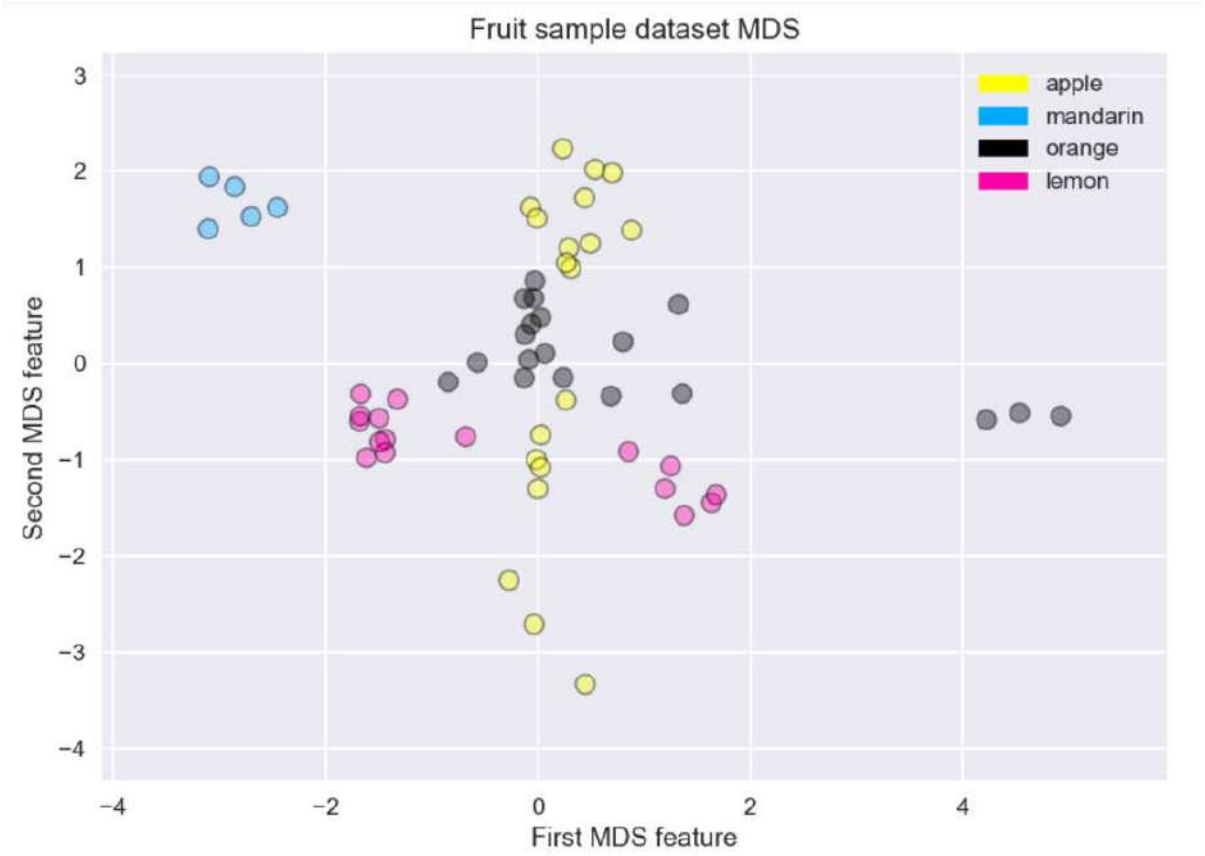
High-dimensional dataset



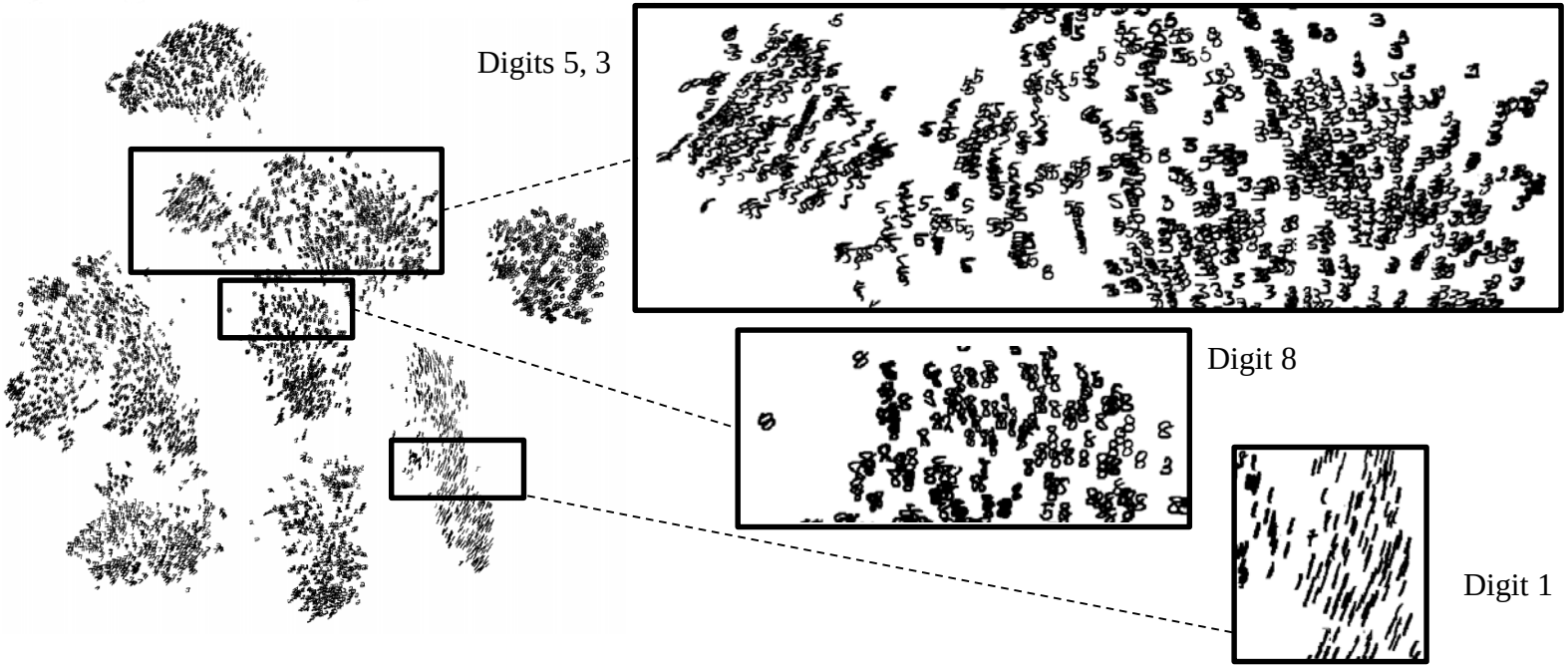
Two-dimensional MDS projection

Notebook: MDS on the Fruit Dataset

```
# Multidimensional scaling
from adspy_shared_utilities import plot_labelled_scatter
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS

# each feature should be centered (zero mean) and with unit variance
X_fruits_normalized = StandardScaler().fit(X_fruits).transform(X_fruits)

mds = MDS(n_components = 2)

X_fruits_mds = mds.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_fruits_mds, y_fruits, ['apple', 'mandarin', 'orange', 'lemon'])
plt.xlabel('First MDS feature')
plt.ylabel('Second MDS feature')
plt.title("Fruit sample dataset MDS")
```


Fruit sample dataset MDS
apple
mandarin
orange
lemon
Second MDS feature
First MDS feature
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

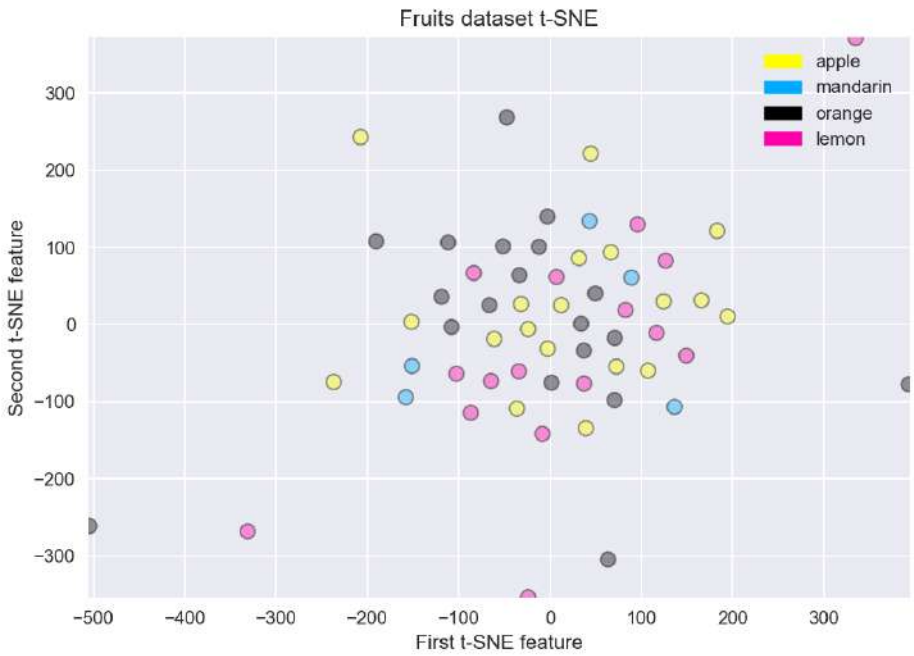
t-SNE: a powerful manifold learning method that finds a 2D projection preserving information about neighbors



Source: van der Maaten & Hinton https://lvdmaaten.github.io/tsne/

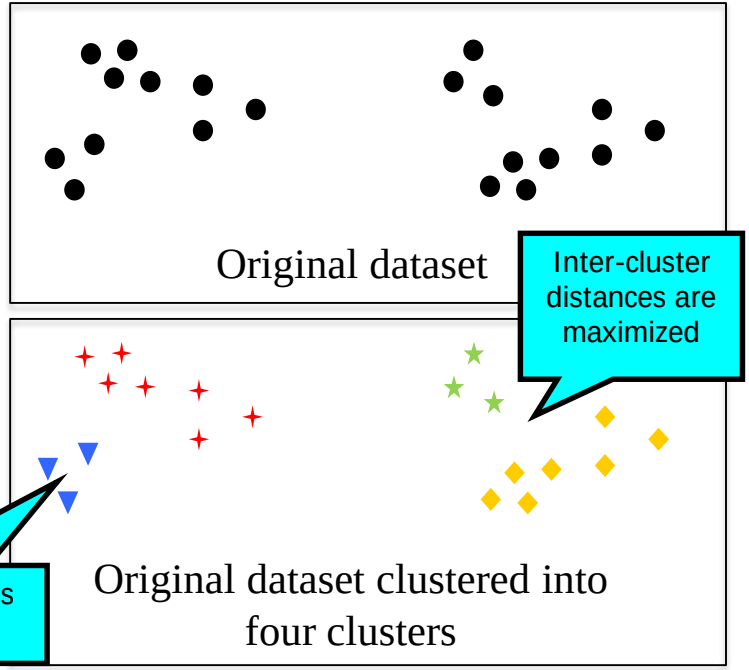
Notebook: t-SNE on the Fruit Dataset

```
from sklearn.manifold import TSNE

tsne = TSNE(random_state = 0)

X_tsne = tsne.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_tsne, y_fruits,
    ['apple', 'mandarin', 'orange', 'lemon'])
plt.xlabel('First t-SNE feature')
plt.ylabel('Second t-SNE feature')
plt.title("Fruits dataset t-SNE")
```



Clustering:
Finding a way to divide a dataset into groups ('clusters')

- **Data points within the same cluster should be 'close' or 'similar' in some way.**
- **Data points in different clusters should be 'far apart' or 'different'**
- **Clustering algorithms output a cluster membership index for each data point:**
  - *Hard clustering: each data point belongs to exactly one cluster*
  - *Soft (or fuzzy) clustering: each data point is assigned a weight, score of membership for each clu*



K-means Clustering

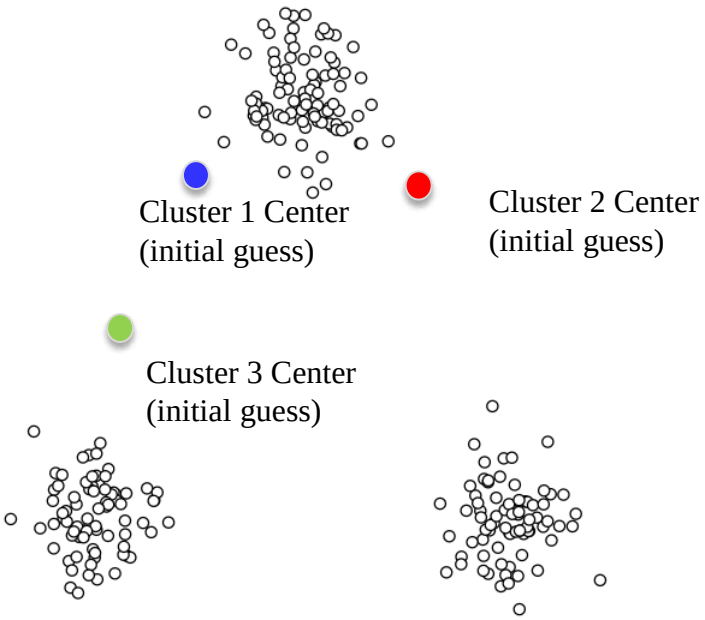
The k-means algorithm

Initialization Pick the number of clusters *k* you want to find. Then pick *k random* points to serve as an initial guess for the cluster centers.

Step A Assign each data point to the nearest cluster center.

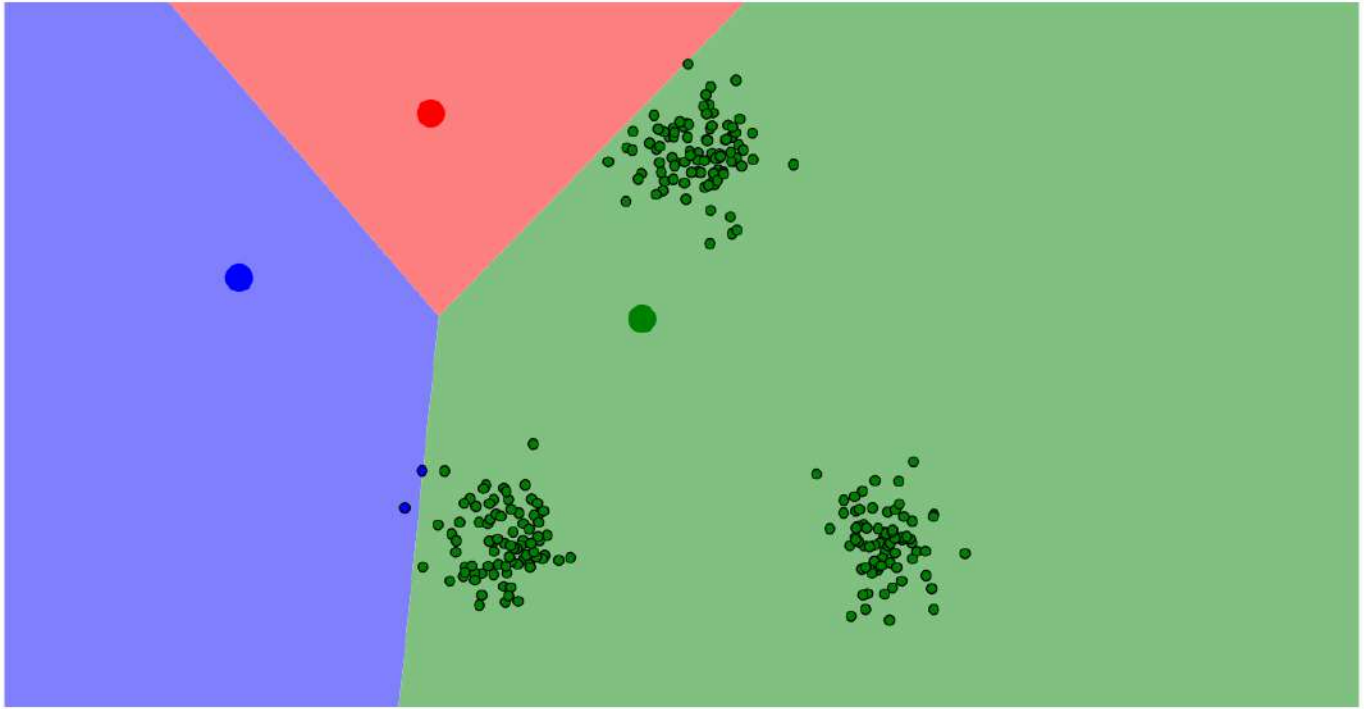
Step B Update each cluster center by replacing it with the mean of all points assigned to that cluster (in step A).

Repeat steps A and B until the centers converge to a stable solution.



Demo: https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

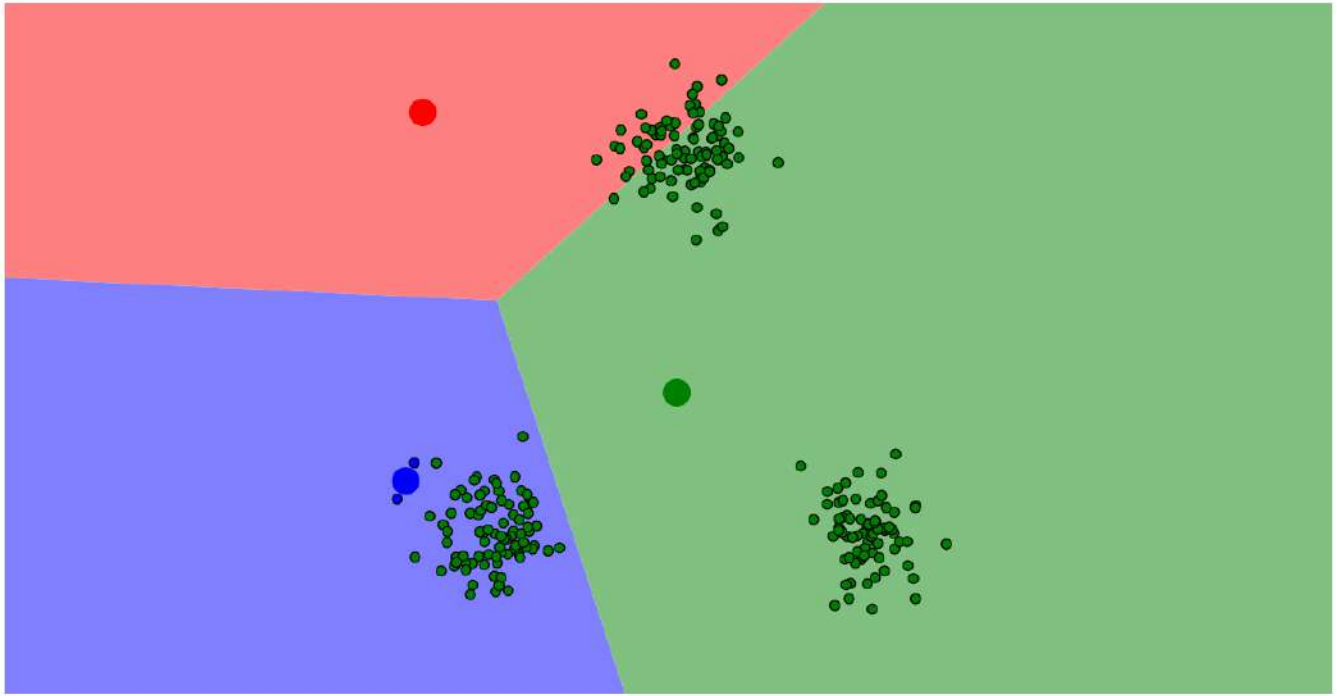
K-means Example: Step 1A



We want three clusters, so three centers are chosen randomly.

Data points are colored according to the closest center.

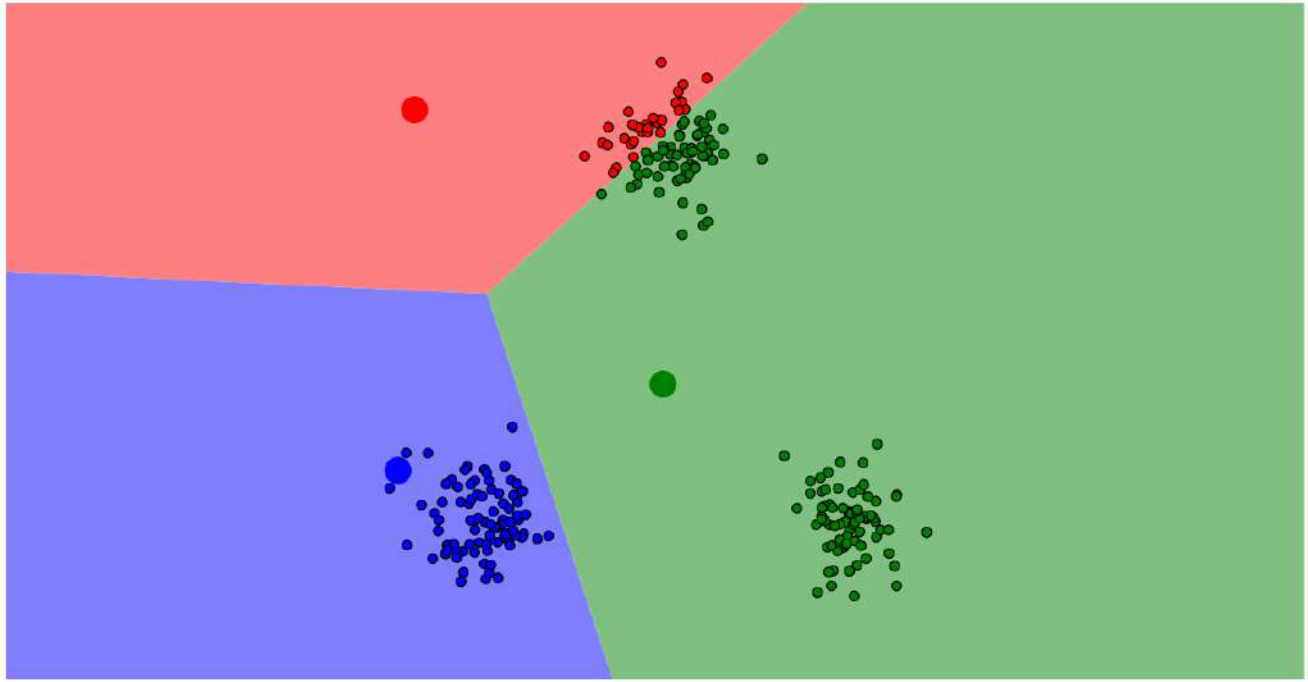
K-means Example: Step 1B



Each center is then updated…

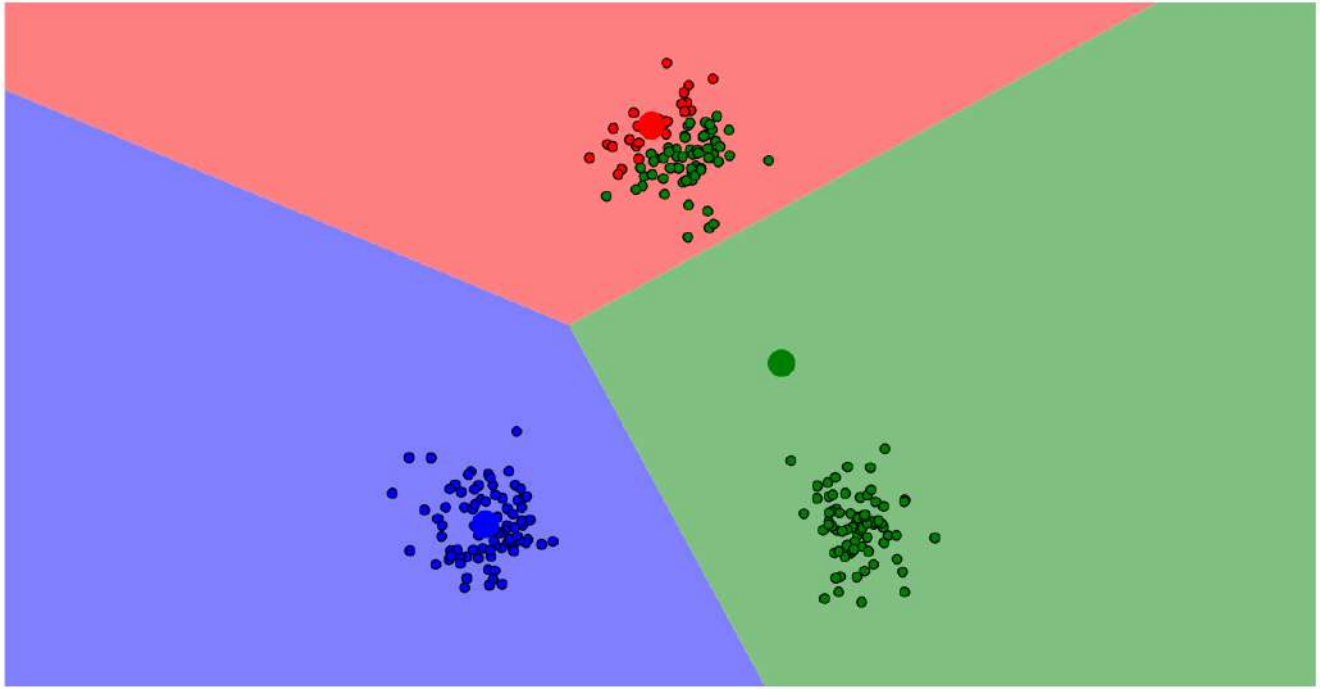
… using the mean of all points assigned to that cluster.

K-means Example: Step 2A



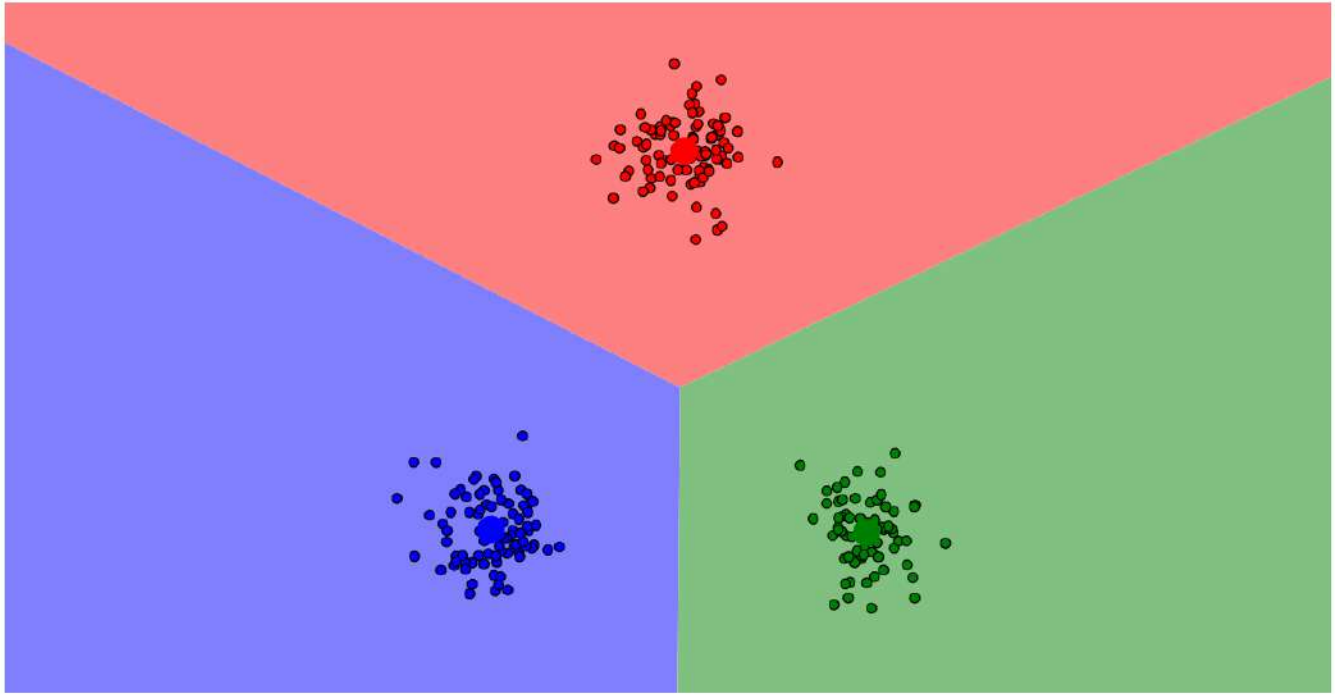
Data points are colored (again) according to the closest center.

K-means Example: Step 2B



Re-calculate all cluster centers.

K-means Example: Converged



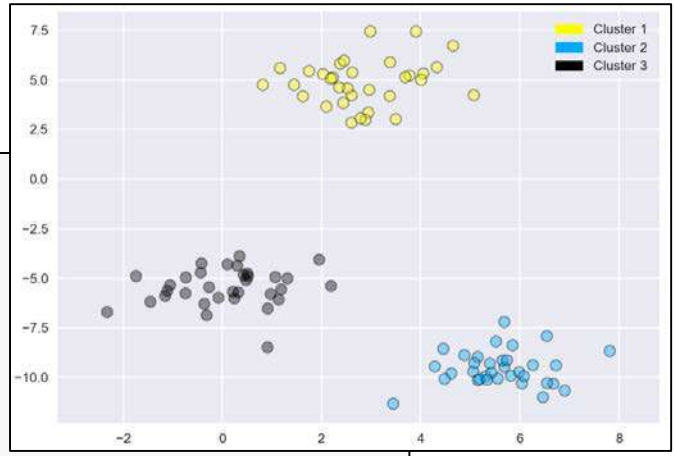
After repeating these steps for several more iterations…

The centers converge to a stable solution!

These centers define the final clusters.

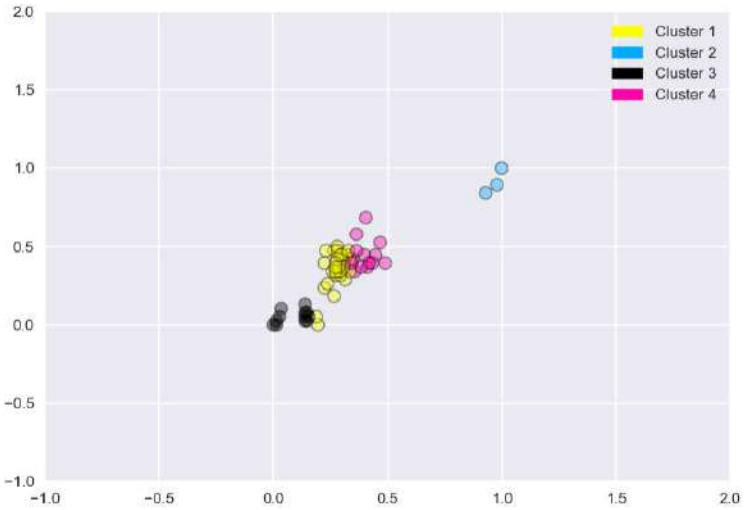
k-means Example in Scikit-Learn

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from adspy_shared_utilities import plot_labelled_scatter

X, y = make_blobs(random_state = 10)

kmeans = KMeans(n_clusters = 3)
kmeans.fit(X)

plot_labelled_scatter(X, kmeans.labels_, ['Cluster 1', 'Cluster 2', 'Cluster 3'])
```



k-means Output on the Fruits Dataset

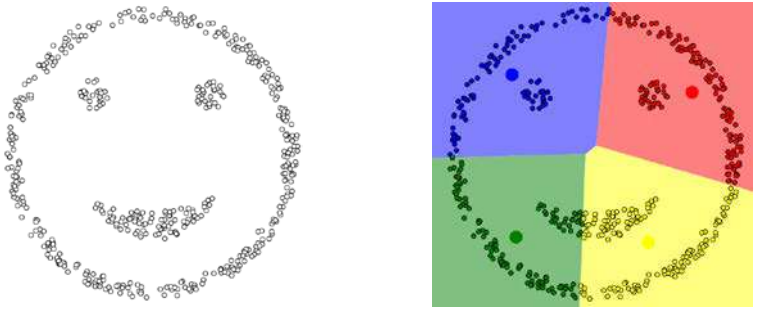
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from adspy_shared_utilities import plot_labelled_scatter
from sklearn.preprocessing import MinMaxScaler

fruits = pd.read_table('fruit_data_with_colors.txt')
X_fruits = fruits[['mass','width','height', 'color_score']].as_matrix()
y_fruits = fruits[['fruit_label']] - 1

X_fruits_normalized = MinMaxScaler().fit(X_fruits).transform(X_fruits)

kmeans = KMeans(n_clusters = 4, random_state = 0)
kmeans.fit(X_fruits)

plot_labelled_scatter(X_fruits_normalized, kmeans.labels_,
                      ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4'])
```



Can you interpret how these clusters correspond with the true fruit labels?

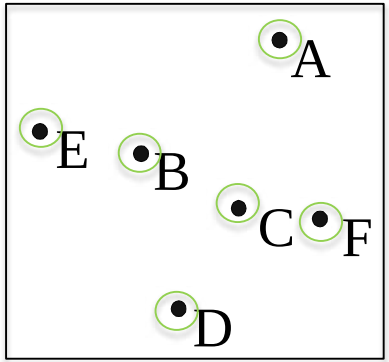
Limitations of k-means

- **Works well for simple clusters that are same size, well-separated, globular shapes.**
- **Does not do well with irregular, complex clusters.**
- **Variants of k-means like k-medoids can work with categorical features.**

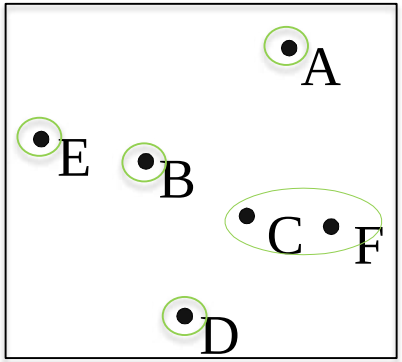


K-means typically performs poorly with data having complex, irregular clusters.

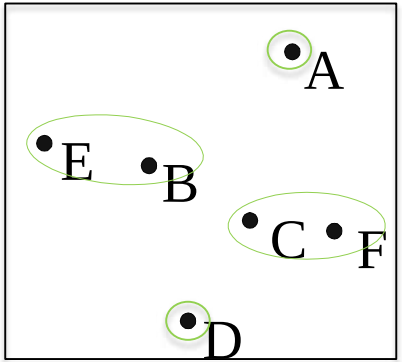
Agglomerative Clustering Example



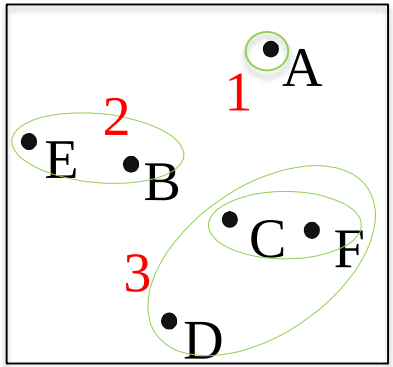
Stage 1



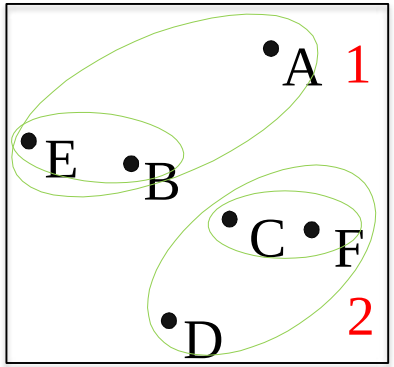
Stage 2



Stage 3



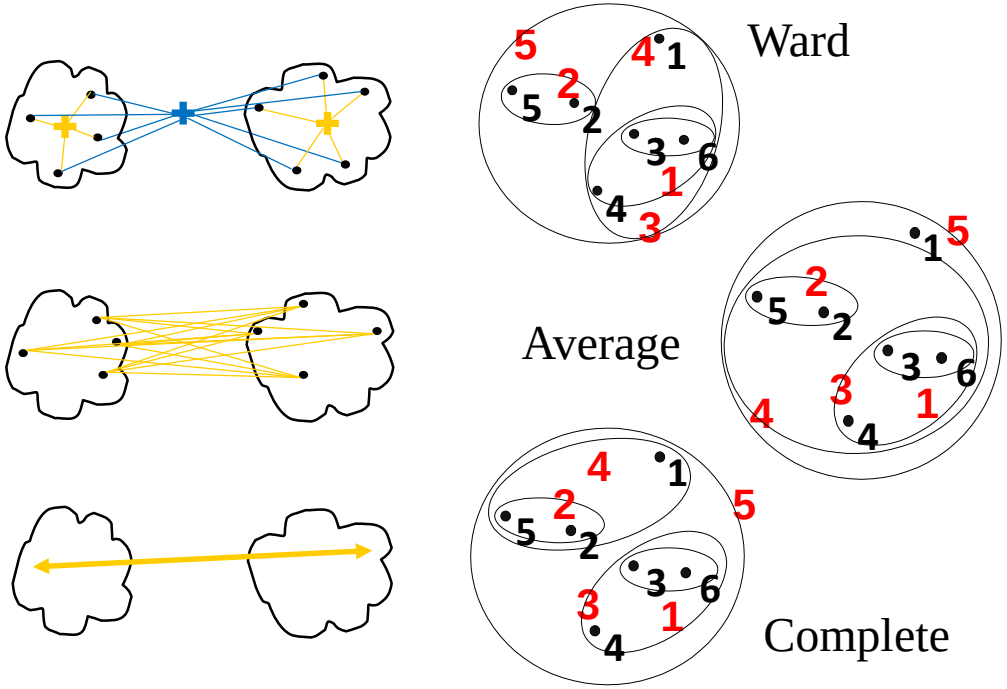
Stage 4
(three top-level clusters)



Stage 5 (two top-level clusters)

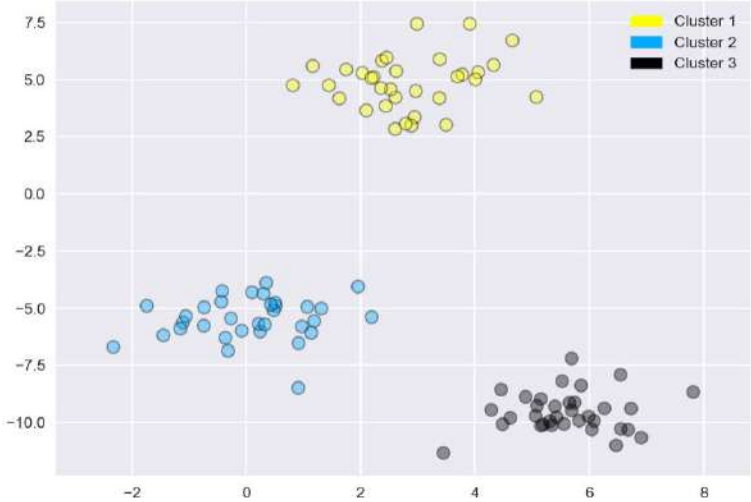
Linkage Criteria for Agglomerative Clustering

- **Ward's method**
  - *Least increase in total variance (around cluster centroids)*
- **Average linkage**
  - *Average distance between clusters*
- **Complete linkage**
  - *Max distance between clusters*

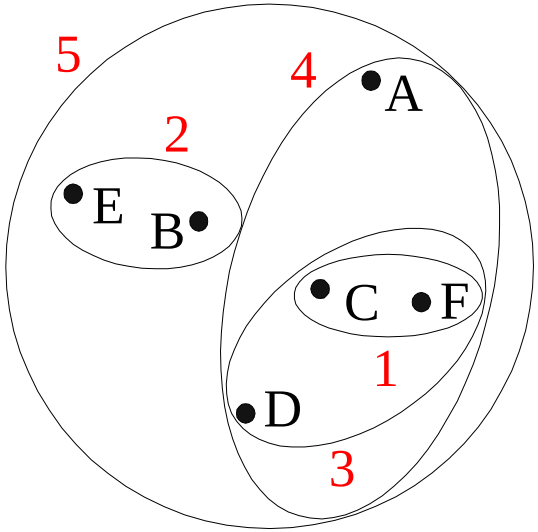


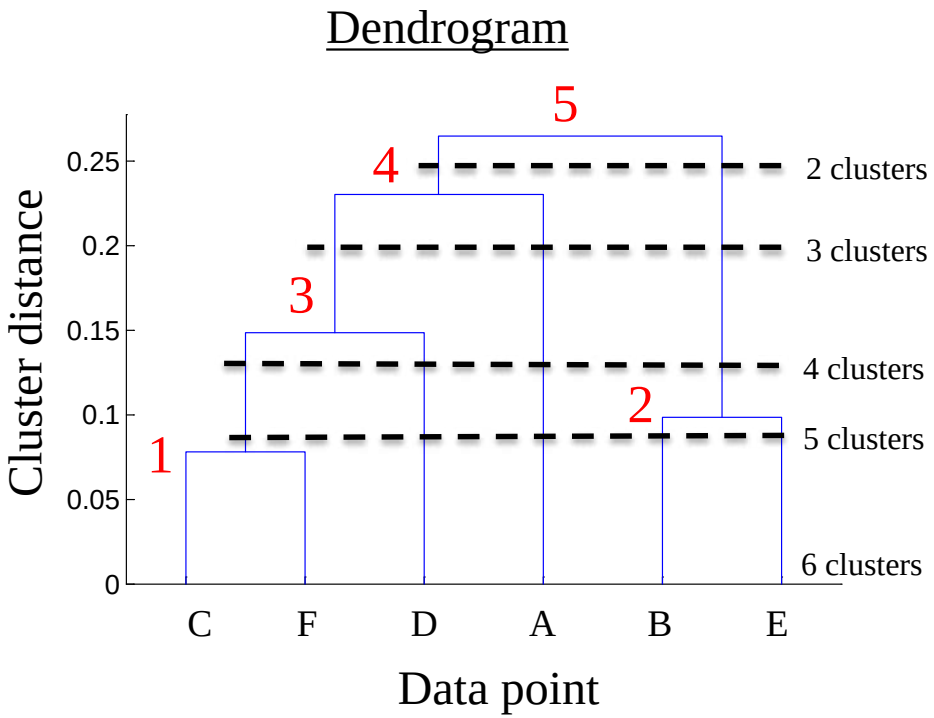
Agglomerative Clustering in Scikit-Learn

```
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
from adspy_shared_utilities import plot_labelled_scatter

X, y = make_blobs(random_state = 10)

cls = AgglomerativeClustering(n_clusters = 3)
cls_assignment = cls.fit_predict(X)

X, y = make_blobs(random_state = 10)
plot_labelled_scatter(X, cls_assignment,
        ['Cluster 1', 'Cluster 2', 'Cluster 3'])
```



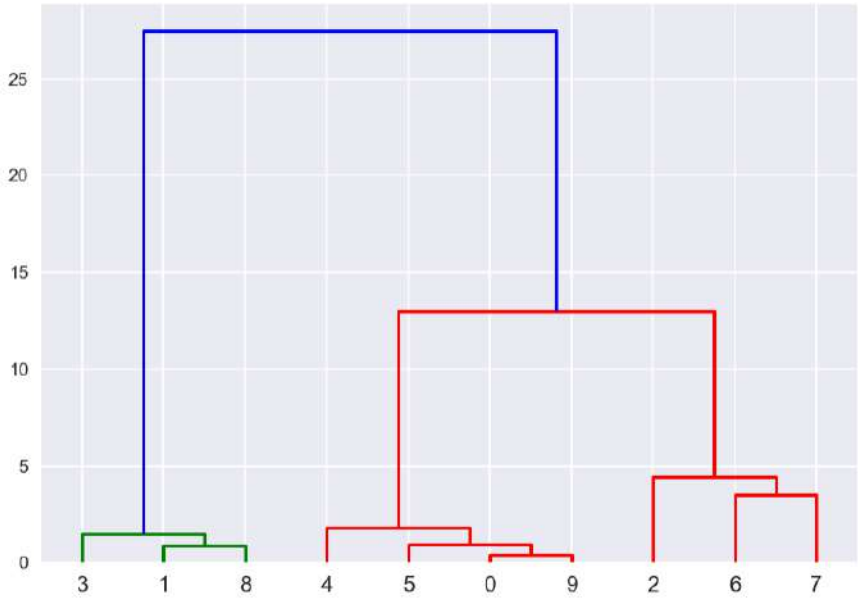
Hierarchical Clustering





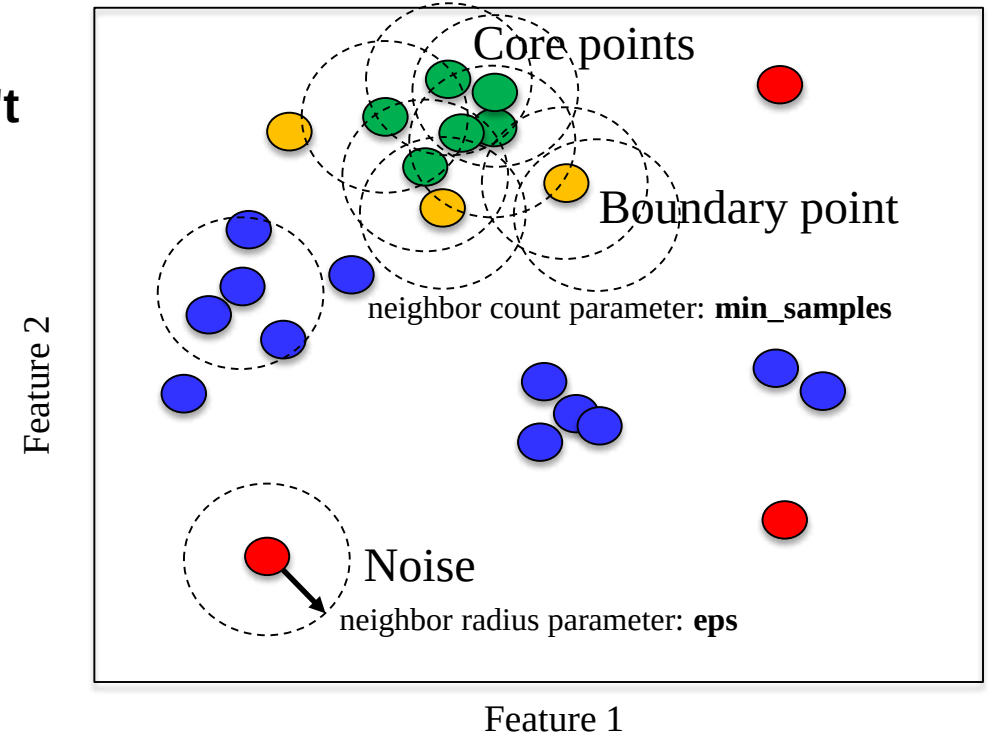
Dendrogram Example

```
from scipy.cluster.hierarchy import ward, dendrogram
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

X, y = make_blobs(random_state = 10, n_samples = 10)

plt.figure()
dendrogram(ward(X))
plt.show()
```



DBSCAN Clustering

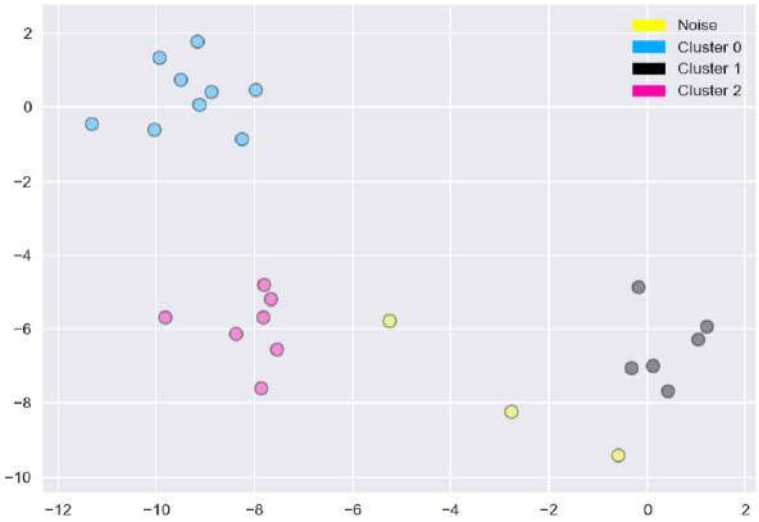
- **Unlike k-means, you don't need to specify # of clusters**
- **Relatively efficient – can be used with large datasets**
- **Identifies likely noise points**



DBSCAN Example in Scikit-Learn

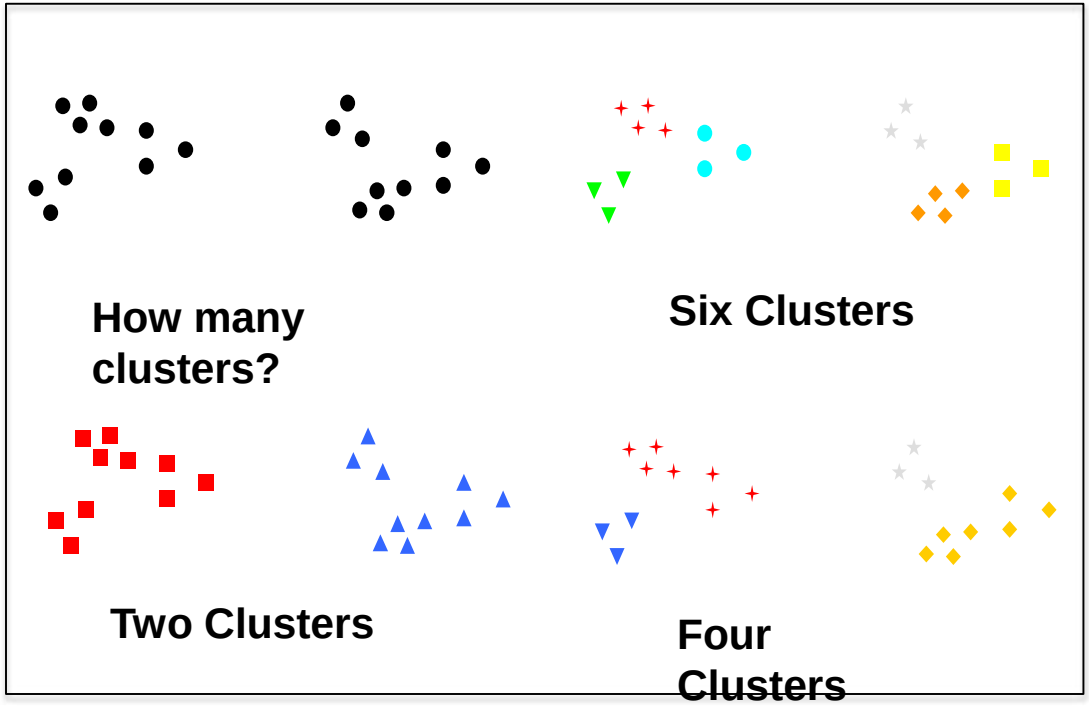
```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state = 9, n_samples = 25)

dbscan = DBSCAN(eps = 2, min_samples = 2)

cls = dbscan.fit_predict(X)
print("Cluster membership values:\n{}", format(cls))

plot_labelled_scatter(X, cls + 1,
        ['Noise', 'Cluster 0', 'Cluster 1', 'Cluster 2'])
```

```
Cluster membership values:
{} [ 0  1  0  2  0  0  0  2  2 -1  1  2  0  0 -1  0  0  1 -1  1  1  2  2  2  1]
```



Clustering Evaluation

- With ground truth, existing labels can be used to evaluate cluster quality.
- Without ground truth, evaluation can difficult: multiple clusterings may be plausible for a dataset.
- Consider task-based evaluation: Evaluate clustering according to performance on a task that <u>does</u> have an objective basis for comparison.
- <u>Example</u>: the effectiveness of clustering-based features for a supervised learning task.
- Some evaluation heuristics exist (e.g. silhouette) but these can be unreliable.



Unsupervised Learning

September 22, 2021

---

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Applied Machine Learning: Unsupervised Learning

1.1 Preamble and Datasets

In [18]:
```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

# Breast cancer dataset
cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# Our sample fruits dataset
fruits = pd.read_table('readonly/fruit_data_with_colors.txt')
X_fruits = fruits[['mass','width','height', 'color_score']]
y_fruits = fruits[['fruit_label']] - 1
```

In [19]:
```
X_cancer,X_cancer.shape, y_cancer, y_cancer.shape
```

Out[19]:
```
(array([[  1.79900000e+01,   1.03800000e+01,   1.22800000e+02, ...,
           2.65400000e-01,   4.60100000e-01,   1.18900000e-01],
        [  2.05700000e+01,   1.77700000e+01,   1.32900000e+02, ...,
           1.86000000e-01,   2.75000000e-01,   8.90200000e-02],
        [  1.96900000e+01,   2.12500000e+01,   1.30000000e+02, ...,
           2.43000000e-01,   3.61300000e-01,   8.75800000e-02],
        ...,
        [  1.66000000e+01,   2.80800000e+01,   1.08300000e+02, ...,
           1.41800000e-01,   2.21800000e-01,   7.82000000e-02],
```

```
       [  2.06000000e+01,   2.93300000e+01,   1.40100000e+02, ...,
          2.65000000e-01,   4.08700000e-01,   1.24000000e-01],
       [  7.76000000e+00,   2.45400000e+01,   4.79200000e+01, ...,
          0.00000000e+00,   2.87100000e-01,   7.03900000e-02]]),
 (569, 30),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
        1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
        0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
        0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
        0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
        0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
        0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
        1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
        1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
        1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 (569,))
```

1.2 Dimensionality Reduction and Manifold Learning

1.2.1 Principal Components Analysis (PCA)

Using PCA to find the first two principal components of the breast cancer dataset

```
In [20]: from sklearn.preprocessing import StandardScaler
         from sklearn.decomposition import PCA
         from sklearn.datasets import load_breast_cancer

         cancer = load_breast_cancer()
         (X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

         # Before applying PCA, each feature should be centered (zero mean) and wit
         X_normalized = StandardScaler().fit(X_cancer).transform(X_cancer) #OBSERVE

         pca = PCA(n_components = 2).fit(X_normalized) #FIRST 2 COMPONENTS IN PCA A
```

```
X_pca = pca.transform(X_normalized)
print(X_cancer.shape, X_pca.shape)
```

```
(569, 30) (569, 2)
```

Plotting the PCA-transformed version of the breast cancer dataset

```
In [21]: from adspy_shared_utilities import plot_labelled_scatter
         plot_labelled_scatter(X_pca, y_cancer, ['malignant', 'benign'])

         plt.xlabel('First principal component')
         plt.ylabel('Second principal component')
         plt.title('Breast Cancer Dataset PCA (n_components = 2)');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2 ek baar 3 components ke saath try karte

2.0.1 lekin plot kaise karenge

```
In [22]: # pca = PCA(n_components = 3).fit(X_normalized) #FIRST 2 COMPONENTS IN PCA

         # X_pca = pca.transform(X_normalized)
         # print(X_cancer.shape, X_pca.shape)

         # plot_labelled_scatter(X_pca, y_cancer, ['malignant', 'benign'])

         # plt.xlabel('First principal component')
         # plt.ylabel('Second principal component')
         # plt.title('Breast Cancer Dataset PCA (n_components = 3)');

         # same hi graph mila na
```

2.0.2 PCA does a neat job here (above)

Plotting the magnitude of each feature value for the first two principal components

```
In [23]: fig = plt.figure(figsize=(8, 4))
         plt.imshow(pca.components_, interpolation = 'none', cmap = 'plasma')
         feature_names = list(cancer.feature_names)

         plt.gca().set_xticks(np.arange(-.5, len(feature_names)));
         plt.gca().set_yticks(np.arange(0.5, 2));
```

```
plt.gca().set_xticklabels(feature_names, rotation=90, ha='left', fontsize=
plt.gca().set_yticklabels(['First PC', 'Second PC'], va='bottom', fontsize

plt.colorbar(orientation='horizontal', ticks=[pca.components_.min(), 0,
                                              pca.components_.max()], pad=
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

PCA on the fruit dataset (for comparison)

```
In [24]: from sklearn.preprocessing import StandardScaler
         from sklearn.decomposition import PCA

         # each feature should be centered (zero mean) and with unit variance
         X_normalized = StandardScaler().fit(X_fruits).transform(X_fruits)

         pca = PCA(n_components = 2).fit(X_normalized)
         X_pca = pca.transform(X_normalized)

         from adspy_shared_utilities import plot_labelled_scatter
         plot_labelled_scatter(X_pca, y_fruits, ['apple','mandarin','orange','lemon

         plt.xlabel('First principal component')
         plt.ylabel('Second principal component')
         plt.title('Fruits Dataset PCA (n_components = 2)');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2.0.3 neat work by pca above

2.0.4 Manifold learning methods

Multidimensional scaling (MDS) on the fruit dataset

```
In [25]: from adspy_shared_utilities import plot_labelled_scatter
         from sklearn.preprocessing import StandardScaler
         from sklearn.manifold import MDS

         # each feature should be centered (zero mean) and with unit variance
         X_fruits_normalized = StandardScaler().fit(X_fruits).transform(X_fruits)

         mds = MDS(n_components = 2)
```

```
X_fruits_mds = mds.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_fruits_mds, y_fruits, ['apple', 'mandarin', 'orang
plt.xlabel('First MDS feature')
plt.ylabel('Second MDS feature')
plt.title('Fruit sample dataset MDS');
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2.0.5 neat work by mds above

Multidimensional scaling (MDS) on the breast cancer dataset (This example is not covered in the lecture video, but is included here so you can compare it to the results from PCA.)

```
In [26]: from sklearn.preprocessing import StandardScaler
         from sklearn.manifold import MDS
         from sklearn.datasets import load_breast_cancer

         cancer = load_breast_cancer()
         (X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

         # each feature should be centered (zero mean) and with unit variance
         X_normalized = StandardScaler().fit(X_cancer).transform(X_cancer)

         mds = MDS(n_components = 2)

         X_mds = mds.fit_transform(X_normalized)

         from adspy_shared_utilities import plot_labelled_scatter
         plot_labelled_scatter(X_mds, y_cancer, ['malignant', 'benign'])

         plt.xlabel('First MDS dimension')
         plt.ylabel('Second MDS dimension')
         plt.title('Breast Cancer Dataset MDS (n_components = 2)');
```

```
        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-26-5533a03a1aa2> in <module>()
         11 mds = MDS(n_components = 2)
         12
    ---> 13 X_mds = mds.fit_transform(X_normalized)
```

```
     14
     15 from adspy_shared_utilities import plot_labelled_scatter


    /opt/conda/lib/python3.6/site-packages/sklearn/manifold/mds.py in fit_trans
    412             n_jobs=self.n_jobs, max_iter=self.max_iter, verbose=self.ve
    413             eps=self.eps, random_state=self.random_state,
--> 414             return_n_iter=True)
    415
    416         return self.embedding_


    /opt/conda/lib/python3.6/site-packages/sklearn/manifold/mds.py in smacof(si
    247                 n_components=n_components, init=init,
    248                 max_iter=max_iter, verbose=verbose,
--> 249                 eps=eps, random_state=random_state)
    250             if best_stress is None or stress < best_stress:
    251                 best_stress = stress


    /opt/conda/lib/python3.6/site-packages/sklearn/manifold/mds.py in _smacof_s
     89     for it in range(max_iter):
     90         # Compute distance and monotonic regression
---> 91         dis = euclidean_distances(X)
     92
     93         if metric:


    /opt/conda/lib/python3.6/site-packages/sklearn/metrics/pairwise.py in eucli
    243         YY = row_norms(Y, squared=True)[np.newaxis, :]
    244
--> 245     distances = safe_sparse_dot(X, Y.T, dense_output=True)
    246     distances *= -2
    247     distances += XX


    /opt/conda/lib/python3.6/site-packages/sklearn/utils/extmath.py in safe_spa
    187         return ret
    188     else:
--> 189         return fast_dot(a, b)
    190
    191


    KeyboardInterrupt:
```

t-SNE on the fruit dataset (This example from the lecture video is included so that you can see how some dimensionality reduction methods may be less successful on some datasets. Here, it doesn't work as well at finding structure in the small fruits dataset, compared to other methods like MDS.)

```
In [27]: from sklearn.manifold import TSNE

         tsne = TSNE(random_state = 0)

         X_tsne = tsne.fit_transform(X_fruits_normalized)

         plot_labelled_scatter(X_tsne, y_fruits,
             ['apple', 'mandarin', 'orange', 'lemon'])
         plt.xlabel('First t-SNE feature')
         plt.ylabel('Second t-SNE feature')
         plt.title('Fruits dataset t-SNE');
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

t-SNE on the breast cancer dataset Although not shown in the lecture video, this example is included for comparison, showing the results of running t-SNE on the breast cancer dataset. See the reading "How to Use t-SNE effectively" for further details on how the visualizations from t-SNE are affected by specific parameter settings.

```
In [28]: tsne = TSNE(random_state = 0)

         X_tsne = tsne.fit_transform(X_normalized)

         plot_labelled_scatter(X_tsne, y_cancer,
             ['malignant', 'benign'])
         plt.xlabel('First t-SNE feature')
         plt.ylabel('Second t-SNE feature')
         plt.title('Breast cancer dataset t-SNE');
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

2.1 Clustering

2.1.1 K-means

This example from the lecture video creates an artificial dataset with make_blobs, then applies k-means to find 3 clusters, and plots the points in each cluster identified by a corresponding color.

```
In [29]: from sklearn.datasets import make_blobs
         from sklearn.cluster import KMeans
         from adspy_shared_utilities import plot_labelled_scatter

         X, y = make_blobs(random_state = 10)

         kmeans = KMeans(n_clusters = 3)
         kmeans.fit(X)

         plot_labelled_scatter(X, kmeans.labels_, ['Cluster 1', 'Cluster 2', 'Clust
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2.1.2 Example showing k-means used to find 4 clusters in the fruits dataset. Note that in general, it's important to scale the individual features before applying k-means clustering.

```
In [30]: from sklearn.datasets import make_blobs
         from sklearn.cluster import KMeans
         from adspy_shared_utilities import plot_labelled_scatter
         from sklearn.preprocessing import MinMaxScaler

         fruits = pd.read_table('readonly/fruit_data_with_colors.txt')
         X_fruits = fruits[['mass','width','height', 'color_score']].as_matrix()
         y_fruits = fruits[['fruit_label']] - 1

         X_fruits_normalized = MinMaxScaler().fit(X_fruits).transform(X_fruits)

         kmeans = KMeans(n_clusters = 4, random_state = 0)
         kmeans.fit(X_fruits_normalized)

         plot_labelled_scatter(X_fruits_normalized, kmeans.labels_,
                               ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2.1.3 yaha se maine dhyaan nahi diya kuch khaas

2.1.4 Agglomerative clustering

```
In [31]: from sklearn.datasets import make_blobs
         from sklearn.cluster import AgglomerativeClustering
```

```
from adspy_shared_utilities import plot_labelled_scatter

X, y = make_blobs(random_state = 10)

cls = AgglomerativeClustering(n_clusters = 3)
cls_assignment = cls.fit_predict(X)

plot_labelled_scatter(X, cls_assignment,
        ['Cluster 1', 'Cluster 2', 'Cluster 3'])
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Creating a dendrogram (using scipy) This dendrogram plot is based on the dataset created in the previous step with make_blobs, but for clarity, only 10 samples have been selected for this example, as plotted here:

```
In [32]: X, y = make_blobs(random_state = 10, n_samples = 10)
         plot_labelled_scatter(X, y,
                 ['Cluster 1', 'Cluster 2', 'Cluster 3'])
         print(X)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[[  5.69192445  -9.47641249]
 [  1.70789903   6.00435173]
 [  0.23621041  -3.11909976]
 [  2.90159483   5.42121526]
 [  5.85943906  -8.38192364]
 [  6.04774884 -10.30504657]
 [ -2.00758803  -7.24743939]
 [  1.45467725  -6.58387198]
 [  1.53636249   5.11121453]
 [  5.4307043   -9.75956122]]
```

```
In [38]: X, y = make_blobs(random_state = 10, n_samples = 10)
         plot_labelled_scatter(X, y,
                 ['Cluster 1', 'Cluster 2'])
         print(X, y)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
[[  5.69192445  -9.47641249]
 [  1.70789903   6.00435173]
 [  0.23621041  -3.11909976]
 [  2.90159483   5.42121526]
 [  5.85943906  -8.38192364]
 [  6.04774884 -10.30504657]
 [ -2.00758803  -7.24743939]
 [  1.45467725  -6.58387198]
 [  1.53636249   5.11121453]
 [  5.4307043   -9.75956122]] [0 1 2 1 0 0 2 2 1 0]
```

```
In [33]: from scipy.cluster.hierarchy import ward, dendrogram
         plt.figure()
         dendrogram(ward(X))
         plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

And here's the dendrogram corresponding to agglomerative clustering of the 10 points above using Ward's method. The index 0..9 of the points corresponds to the index of the points in the X array above. For example, point 0 (5.69, -9.47) and point 9 (5.43, -9.76) are the closest two points and are clustered first.

2.1.5 DBSCAN clustering

```
In [34]: from sklearn.cluster import DBSCAN
         from sklearn.datasets import make_blobs

         X, y = make_blobs(random_state = 9, n_samples = 25)

         dbscan = DBSCAN(eps = 2, min_samples = 2)

         cls = dbscan.fit_predict(X)
         print("Cluster membership values:\n{}".format(cls))

         plot_labelled_scatter(X, cls + 1,
                 ['Noise', 'Cluster 0', 'Cluster 1', 'Cluster 2'])
```

```
Cluster membership values:
[ 0  1  0  2  0  0  0  2  2 -1  1  2  0  0 -1  0  0  1 -1  1  1  2  2  2  1]
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

In [ ]: