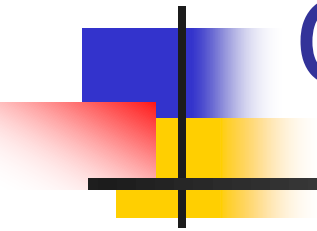
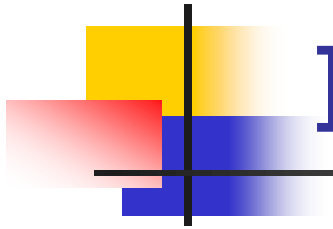


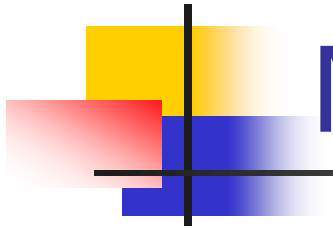
Computer Methods (MAE 3403)

Machine learning



Introduction

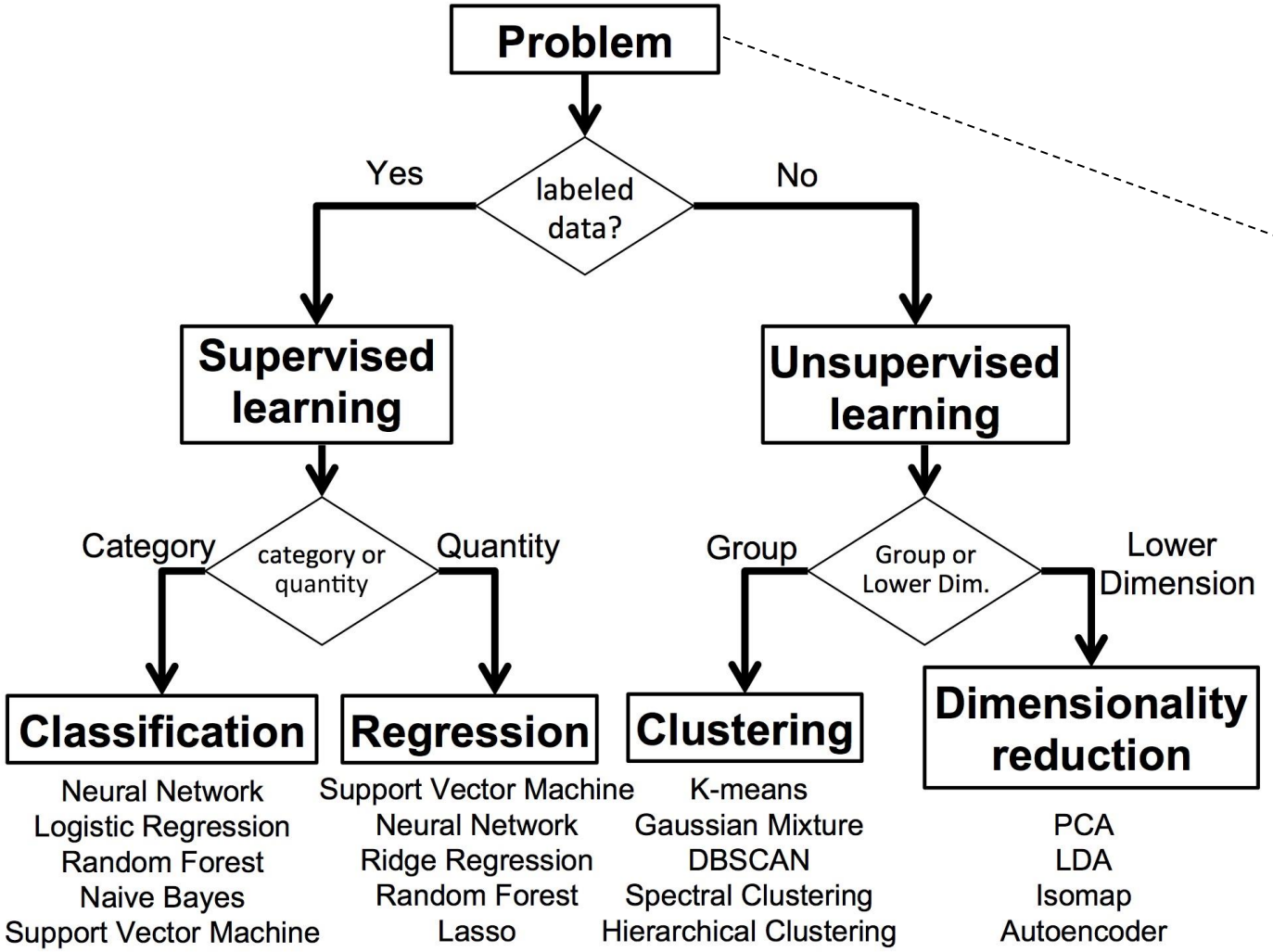
- Machine learning becomes more and more popular
- Python has been one of the main stream programming languages for advancing machine learning technologies
- Introduce some basic ML problems and tools
  - Mathematics will not be covered



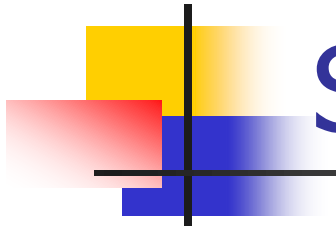
Machine learning

- A group of algorithms to enable the learning capabilities of computers, so they learn from data or past experiences.
  - How we learn to recognize cats/dogs, play games
- ML applications
  - Siri, Face recognition, ATM, email spam detectors, self-driving cars, etc.

Classification of Machine Learning

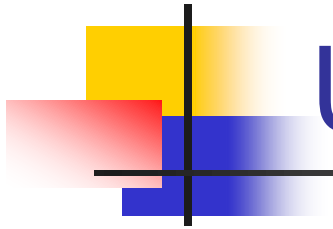


Reinforcement learning



Supervised learning

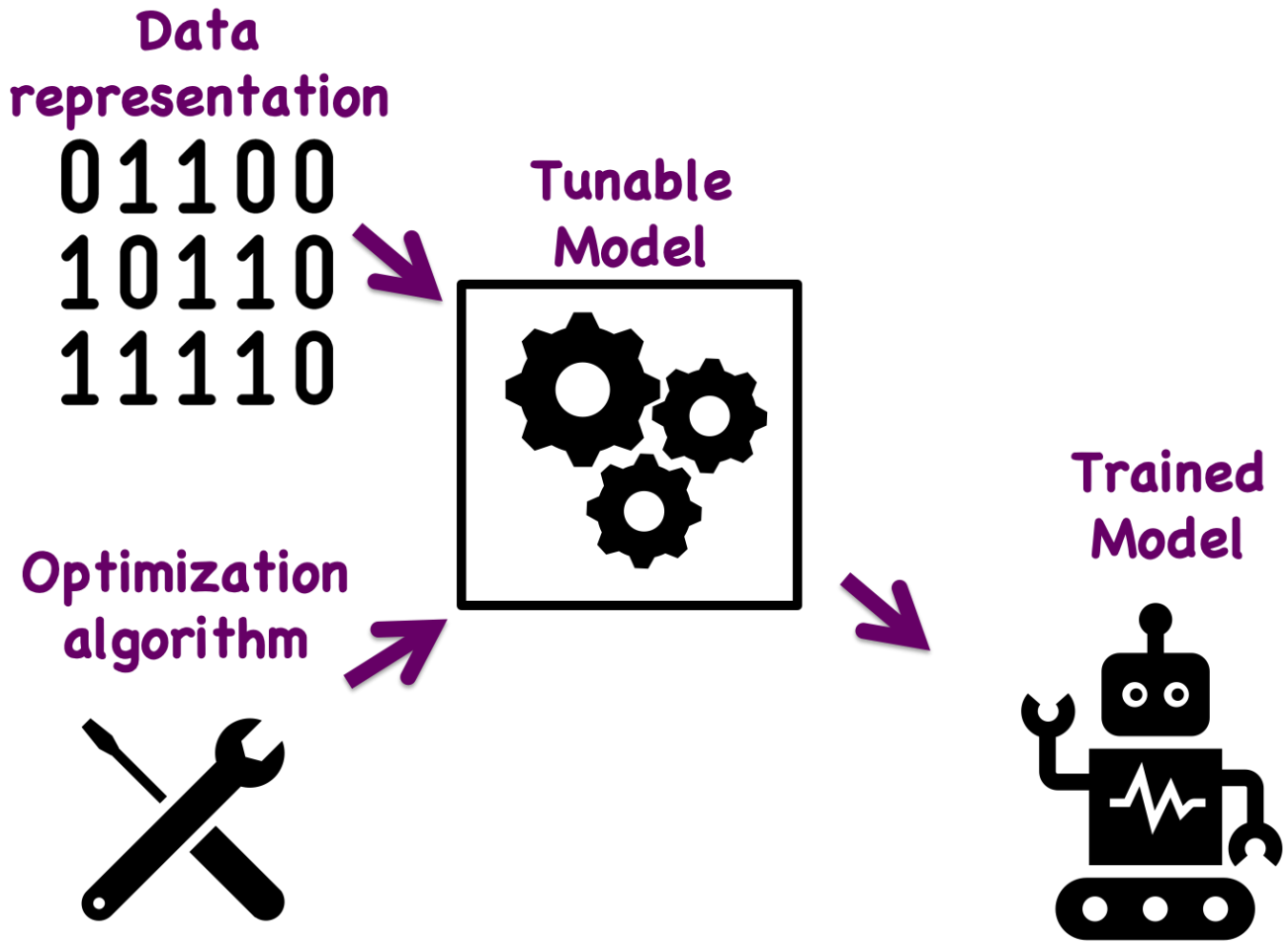
- During training, we know the correct label of the data, i.e., we know the answers of some problem instances.
  - Use prior knowledge (labels) for learning
- Classification vs. regression
  - Classification: recognize apples vs. oranges
  - Regression: predict tomorrow's temperature based past
  - Discrete outcome vs. continuous outcome



Unsupervised learning

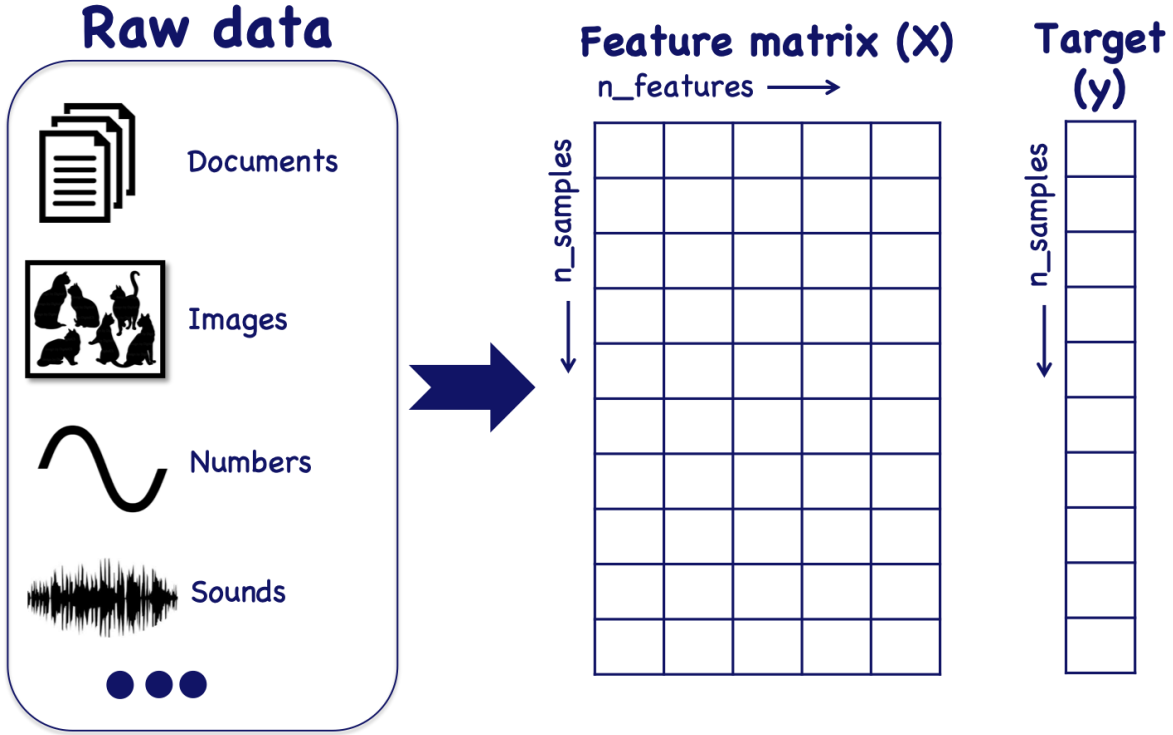
- No labels: given mixed apples and oranges without knowing which one is which
- Clustering problem: use hidden characteristics of the data to classify the data
- Dimensionality reduction: reduce higher-dimension problems into lower-dimension ones.
  - Lower-dimension problems are easier to comprehend and visualize.

Components of machine learning

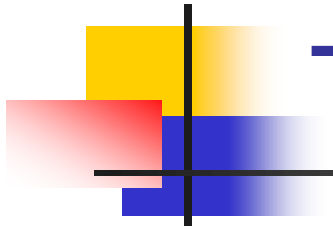


Representation of data

- Images, time series, documents, numerical data, etc
- Turn them into a format computers can use/recognize

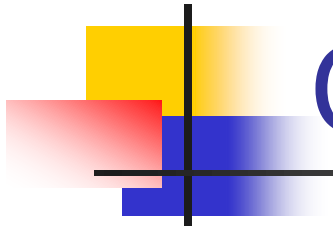


- Each row: one data sample (orange, apple)
- Each column: features (color, shape, etc)
- Target: output, correct labels, quantities



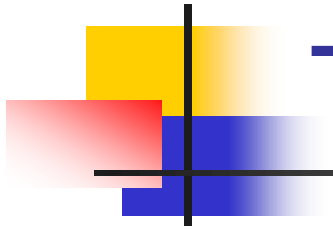
Tunable model

- An algorithm that learns from the data
- Many parameters can be tuned so that the model performs better
- Different models: neural networks, support vector machine, logistic regression, random forest, etc.



Optimization algorithm

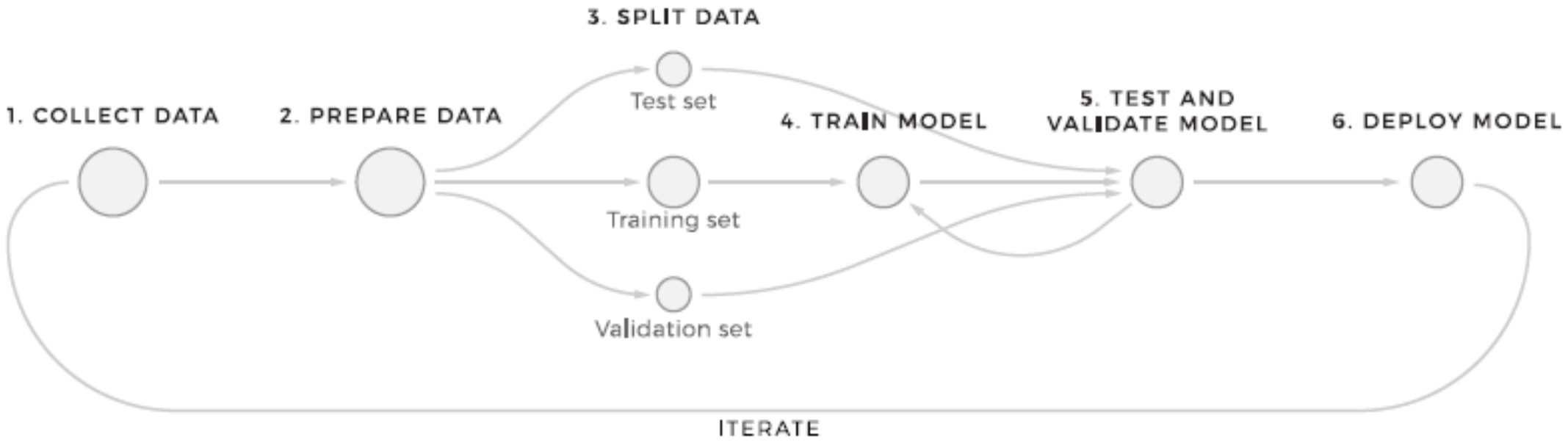
- Main working force to tune the model.
- Defines an objective function and use an optimization algorithm to optimize the objective function by changing the parameters of the tunable model
  - For example: objective is to minimize the error between what the model produces and the true labels
- Different optimization algorithms: gradient descent, etc.



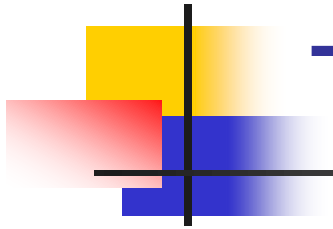
Trained model

- After tuning the model, the trained model has the capability to predict based on unseen data.

Machine learning pipeline



https://www.altexsoft.com/blog/machine-learning-pipeline/



Training, validation, and testing

- The goal: Create a machine learning model that generalizes well to new data
- Never train that model on test data
- Training dataset (80%): learn the model
- Validation dataset (10%): evaluate/fine-tune the model
- Testing dataset (10%): evaluate the final model

Performance metrics

- Regression
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
  - Mean Absolute Error (MAE)
  - …

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2$$

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \check{y}_j|$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2}$$

Classification

- Accuracy
  - number of correct predictions divided by the total number of predictions, multiplied by 100
- Confusion matrix

		Predicted	
		Has Cancer	Doesn't Have Cancer
Ground Truth	Has Cancer	TP	FN
	Doesn't Have Cancer	FP	TN

More metrics for classification

- Precision and Recall
  - Precision: the ratio of true positives and total positives predicted

$$P = \frac{TP}{TP+FP} = \frac{\text{Cancer patients correctly identified}}{\text{Cancer patients correctly identified+incorrectly labelled cancer patients as non-cancerous}}$$

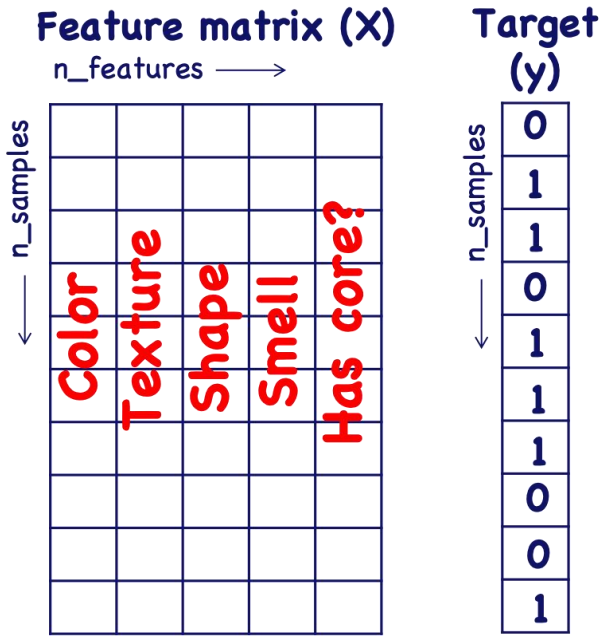
  - Recall/Sensitivity/Hit-Rate: the ratio of true positives to all the positives in ground truth

$$R = \frac{TP}{TP+FN} = \frac{\text{Cancer patients correctly identified}}{\text{Cancer patients correctly identified+incorrectly labelled non-cancer patients as cancerous}}$$

- ...

Classification

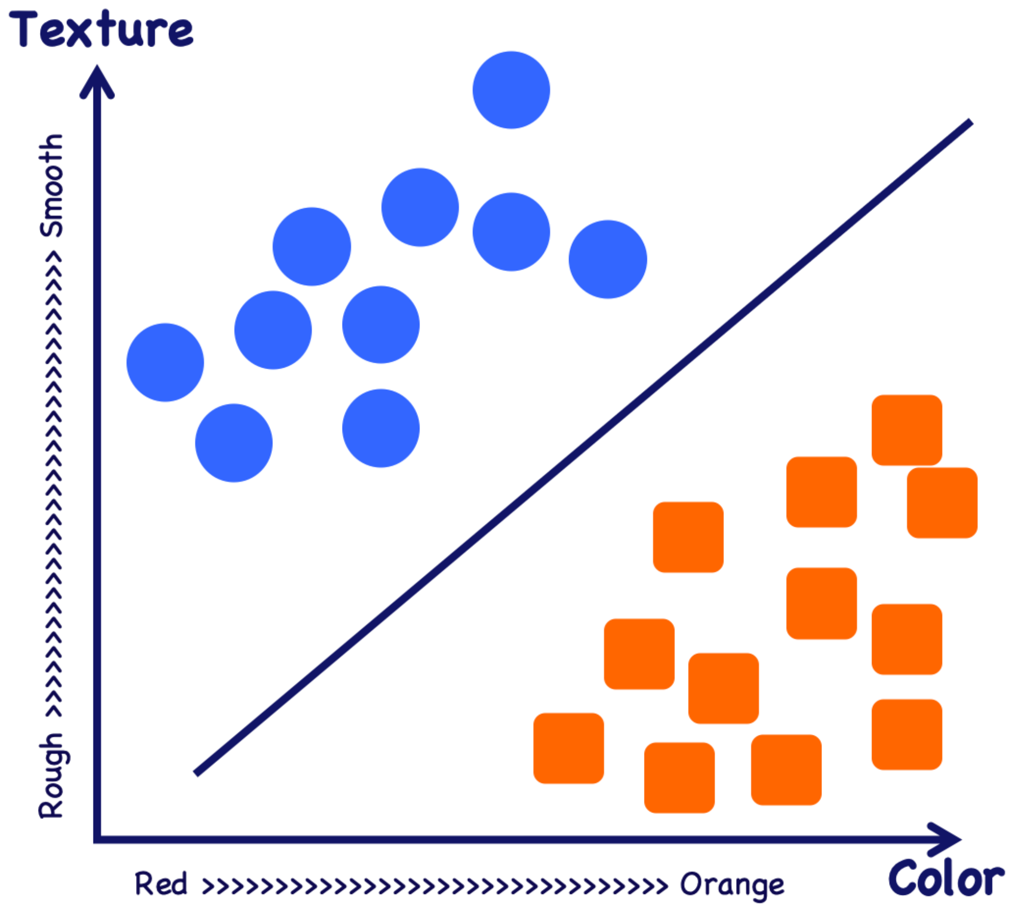
- Classify products into good and bad quality, emails into good or junk, books into different categories, etc.
  - 1) labels are provided, 2) output is categorical
- Binary classification



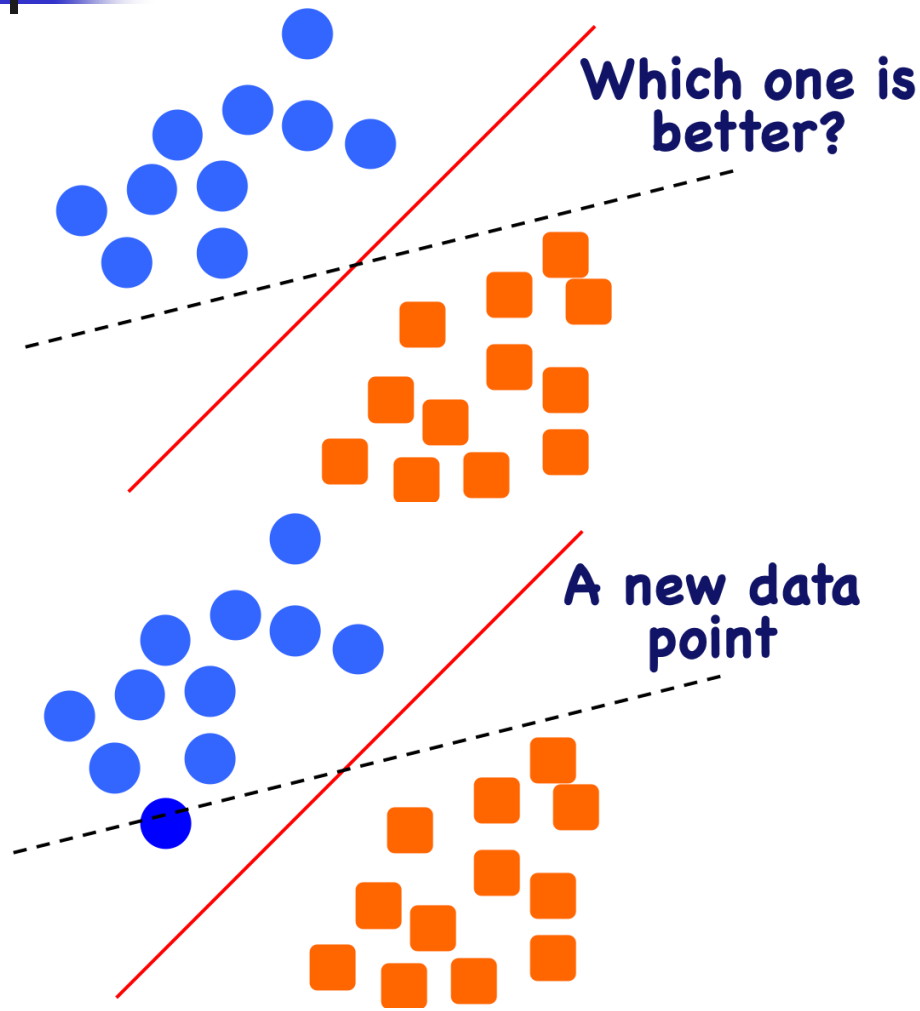
Feature space

- Let's visualize the feature space (two features)

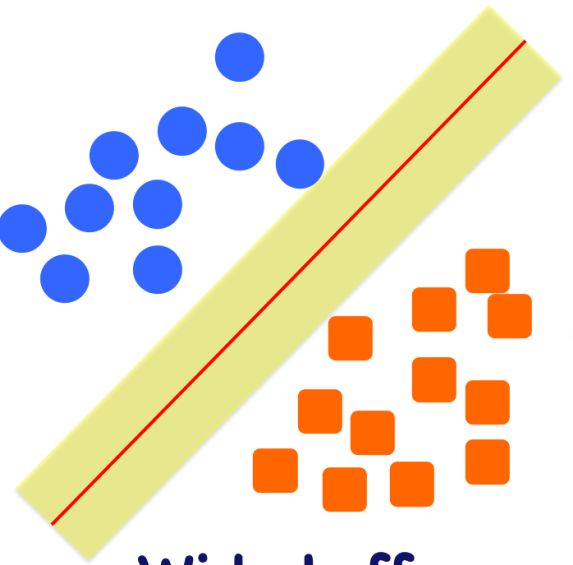
- Classification: find the decision boundary (straight or curved lines) to separate the feature space.



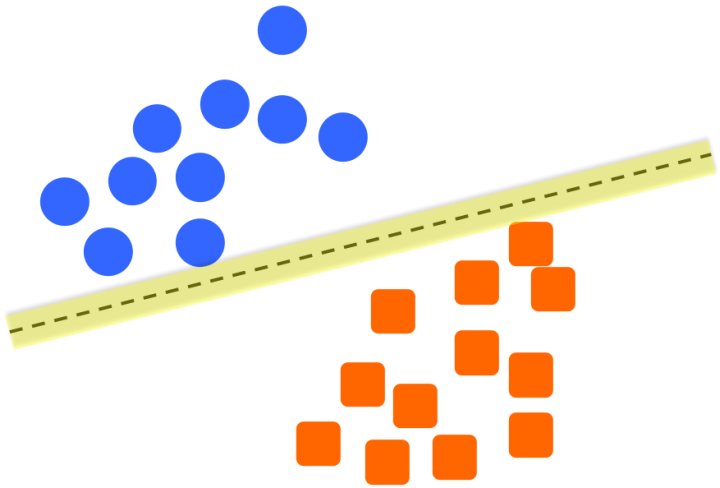
Support vector machine (SVM)



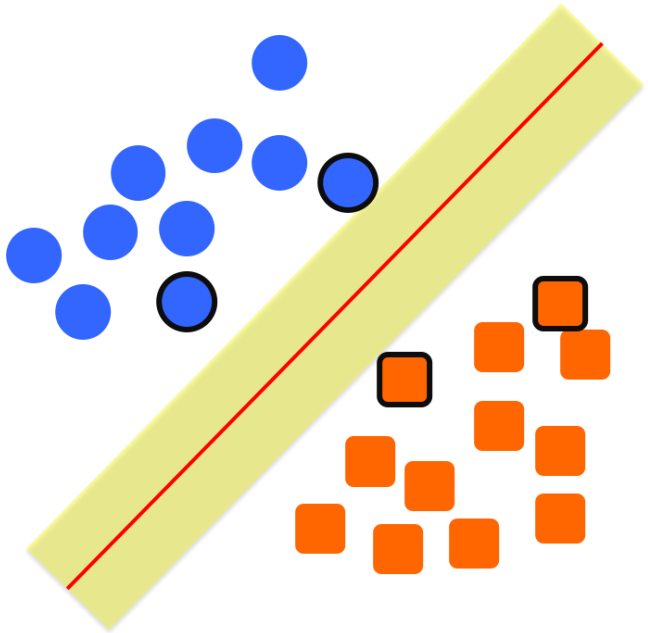
- SVM: intuitive algorithm for classification
  - Forms a buffer from the boundary line to the points in both classes close to the line (support vectors)
  - Given a set of support vectors, which line has the maximum buffer.



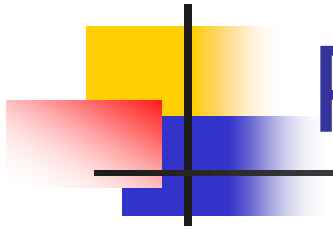
Wide buffer



Narrow buffer



support vectors



Python

- Most popular general ML package: scikit-learn
  - Need installation
- Use existing data sets for classification
- A dataset is a dictionary like object that holds all the data and meta-data.
  - sorted in .data member, (n_samples, n_features) array, & .target member

Preparation

- 50 samples of three species of Iris (setosa, virginica, versicolor)



```
import numpy as np
import itertools
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.metrics import classification_report
```

Load the iris dataset

```
# import the iris data
iris = datasets.load_iris()
print(iris.feature_names)
# only print the first 10 samples print(iris.data[:10])
print('We have %d data samples with %d \
features'%(iris.data.shape[0], iris.data.shape[1]))
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[[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]]
We have 150 data samples with 4 features
```

Target

```
print(iris.target_names)
print(set(iris.target))
```

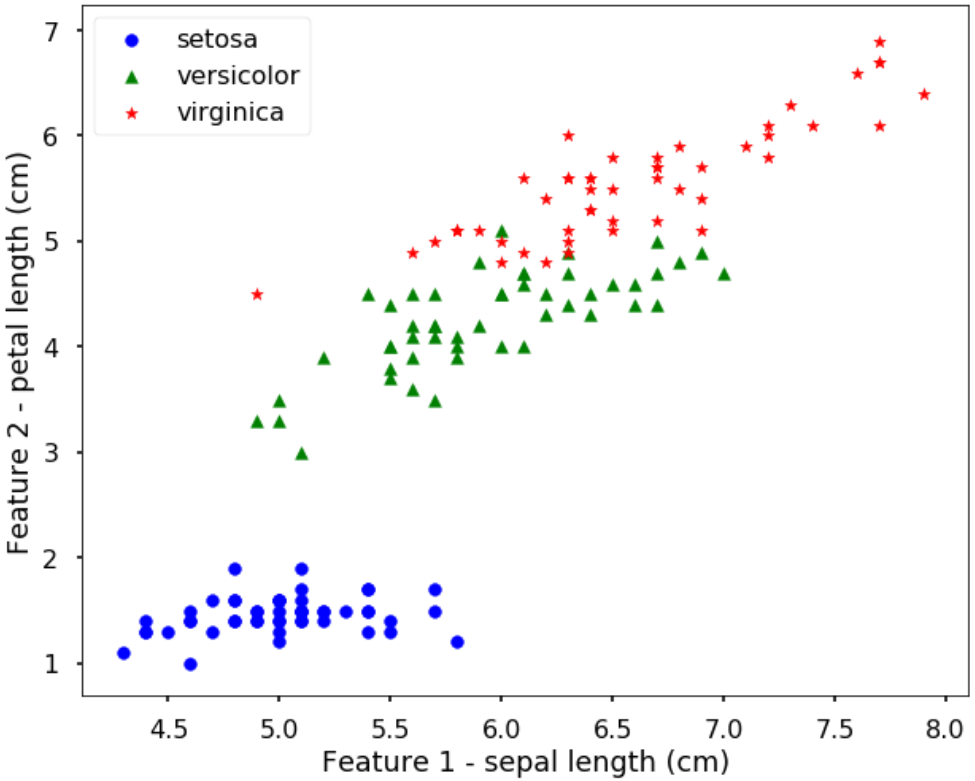
```
['setosa' 'versicolor' 'virginica']
{0, 1, 2}
```

- A bit more preparation

```
# let's just use two features, so that we can
# easily visualize them
X = iris.data[:, [0, 2]]
y = iris.target
target_names = iris.target_names
feature_names = iris.feature_names
# get the classes
n_class = len(set(y))
print('We have %d classes in the
data'%(n_class))
```

Visualize the data

```
# let's have a look of the data first
colors = ['b', 'g', 'r']
symbols = ['o', '^', '*']
plt.figure(figsize = (10,8))
for i, c, s in (zip(range(n_class), colors, symbols)):
    ix = y == i
    plt.scatter(X[:, 0][ix], X[:, 1][ix],
                \ color = c, marker = s, s = 60,
                \ label = target_names[i])
plt.legend(loc = 2, scatterpoints = 1)
plt.xlabel('Feature 1 - ' + feature_names[0])
plt.ylabel('Feature 2 - ' + feature_names[2])
plt.show()
```



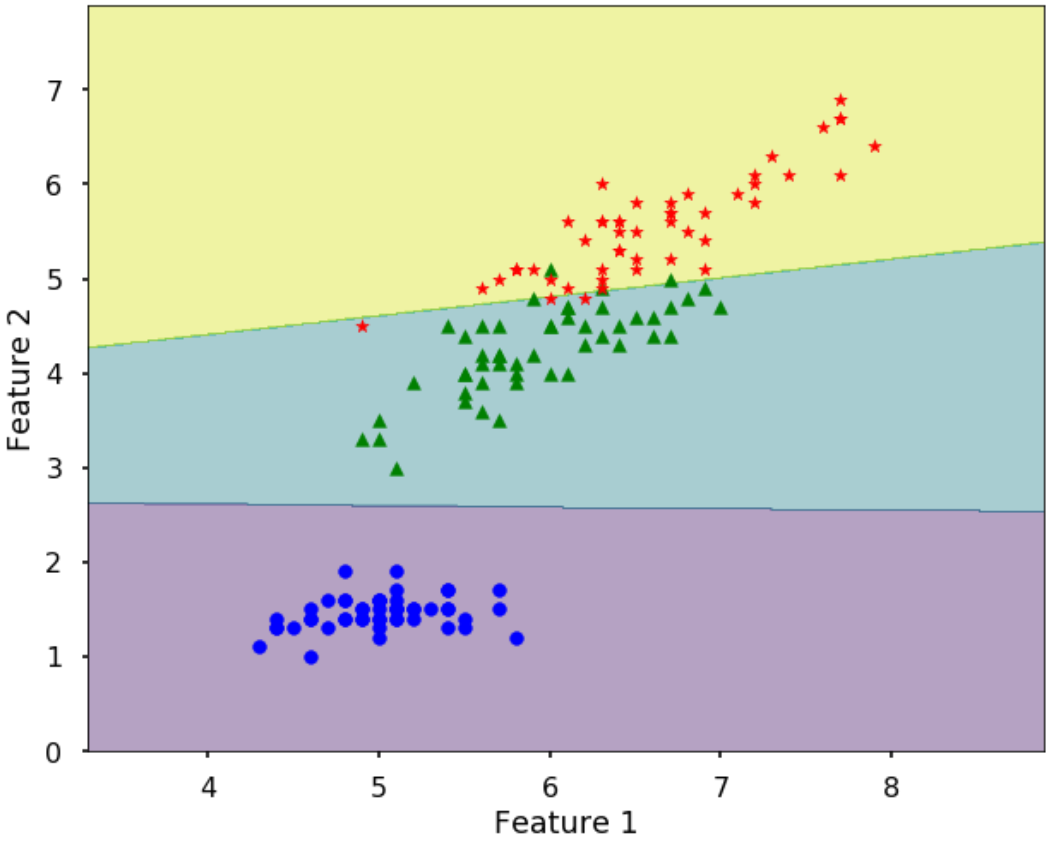
Learning using SVM

- Initialize the model
- Train the model using fit function
- predict on the new data using predict function

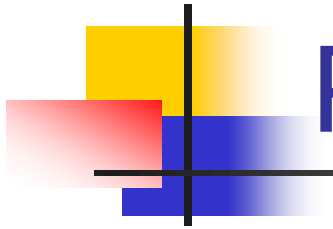
```
# Initialize SVM classifier
clf = svm.SVC(kernel='linear')
# Train the classifier with data
clf.fit(X,y)
# Predict on the data
clf.predict(X)
```

- Many different parameters for a SVM
- Typically, we don't predict on X which is the training data. We separate the entire data into training and testing. Testing data is not used in training at all and only used for evaluation.

Decision boundary



- There are many other models you can use in scikit-learn
- Use an Artificial Neural Network (ANN) to do the same job
  - Use the MLPClassifier for ANN classifier

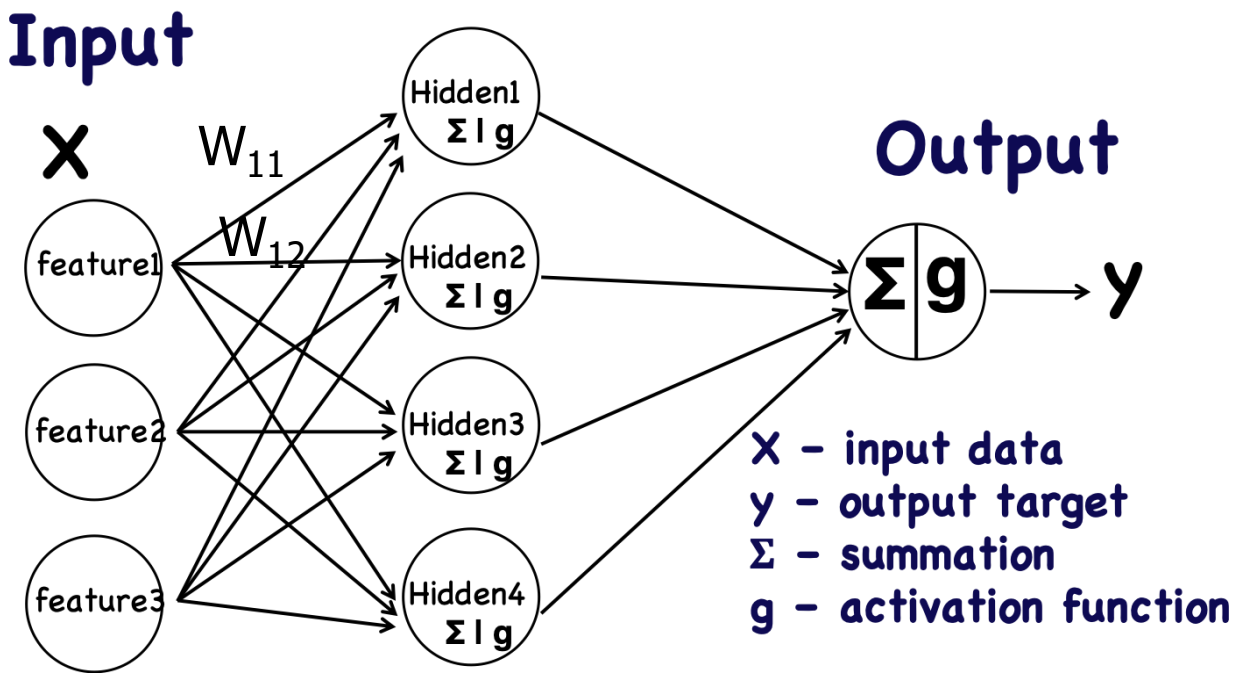


Regression

- Output is quantity numbers rather than categorical.
  - Least squares regression is the simplest form.
- ML approaches are more flexible: can fit any functions of data using random forest, ANN, SVM, etc.
- Let's talk about ANN

Artificial Neural Networks (ANN)

- Developed to mimic how neurons in human brain works



Multi-layer ANN

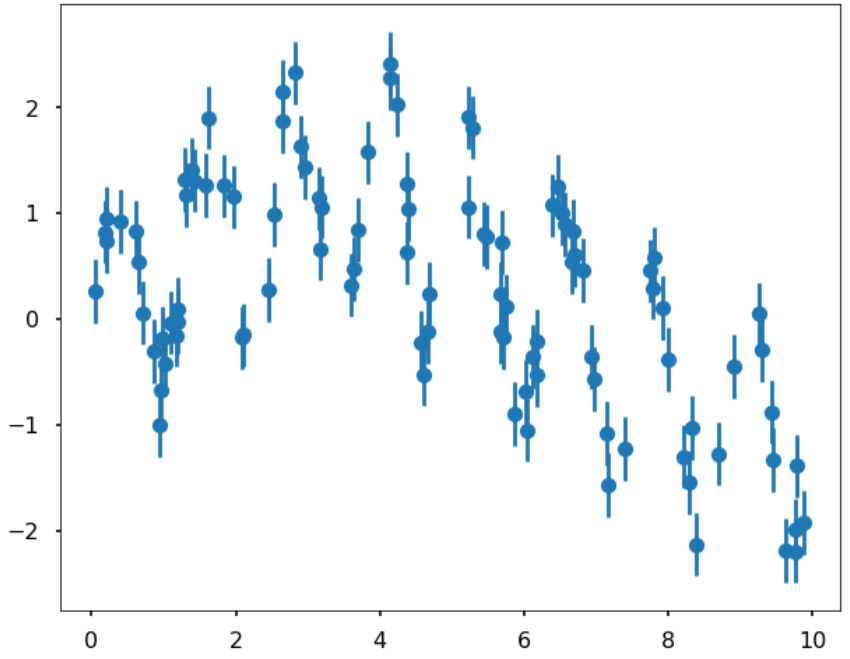
- Neurons: circles, Arrows: links
- Links associated with weights
- 3 layers: Input layer – Hidden layer – output layer
  - many hidden layers
- Hidden layers: sum information from previous layer, pass the summed information to an activation function

Forward propagation and training

- Forward propagation: the information flows from the input layer through links to the hidden layers, gets processed, and then to the output layer to generate the results y.
- Training of ANN: use optimization algorithms to minimize the error between the model estimates y and the true targets
  - First, do a forward propagation to get the error
  - Second, propagate this error backwards (chain-rule) to update the weight parameters in each link, i.e., *backpropagation.*
  - Repeat the process ("epochs")

Example Generate data and visualize it

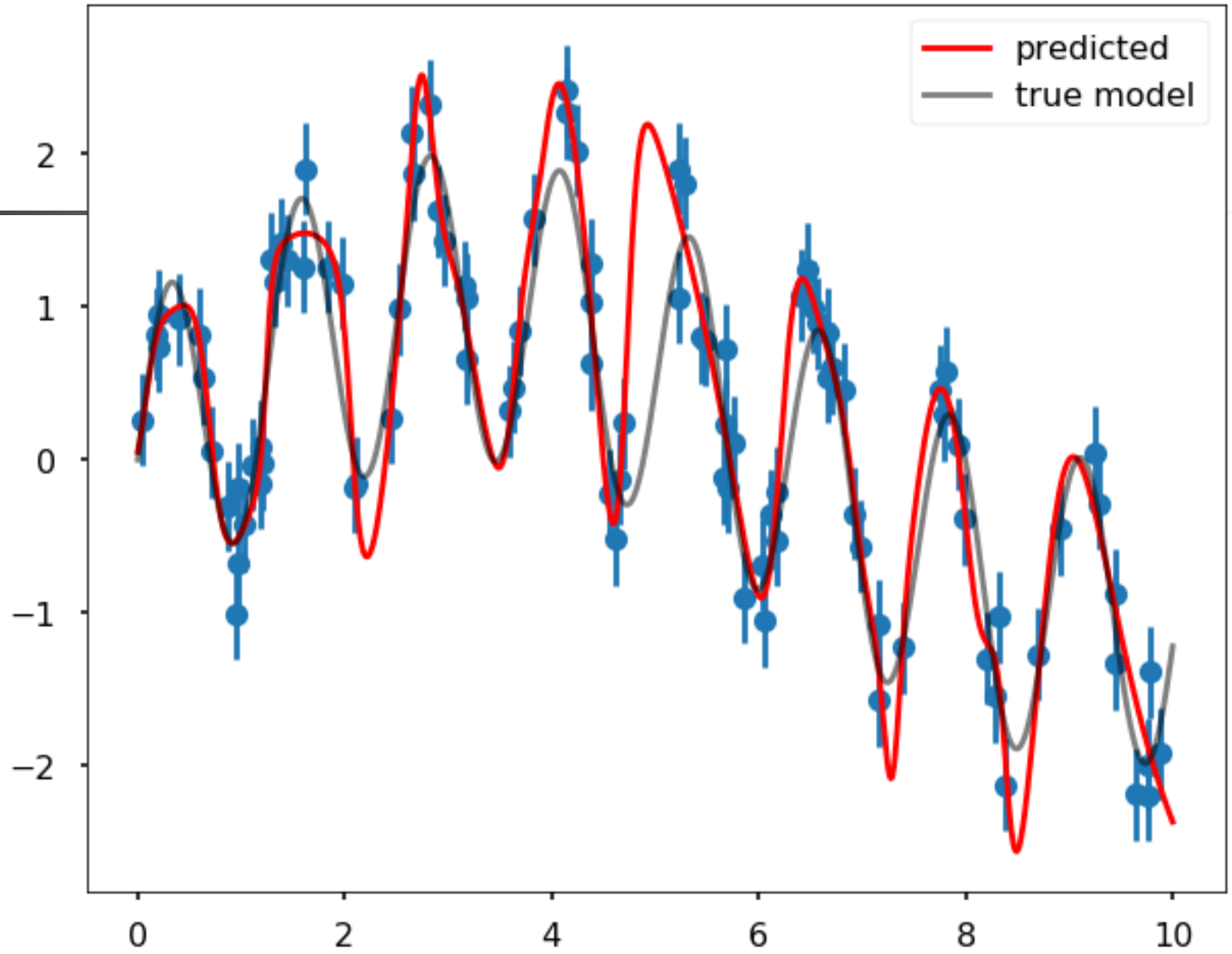
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
np.random.seed(0)
x = 10 * np.random.rand(100)
def model(x, sigma=0.3):
      fast_oscillation = np.sin(5 * x)
      slow_oscillation = np.sin(0.5 * x)
      noise = sigma * np.random.randn(len(x))
       return slow_oscillation + fast_oscillation + noise
plt.figure(figsize = (10,8))
y = model(x)
plt.errorbar(x, y, 0.3, fmt='o')
```

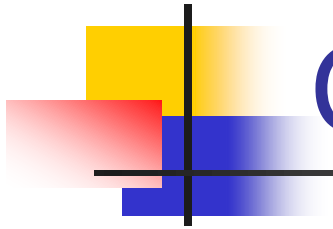


Example

Use ANN to fit a model (x as feature, y as output)

```
from sklearn.neural_network import MLPRegressor #Multi-layer perceptron
mlp = MLPRegressor(hidden_layer_sizes=(200,200,200), \
                   max_iter = 2000, solver='lbfgs', \
                   alpha=0.01, activation = 'tanh', \
                   random_state = 8)
xfit = np.linspace(0, 10, 1000)
ytrue = model(xfit, 0)
yfit = mlp.fit(x[:, None], y).predict(xfit[:, None])
plt.figure(figsize = (10,8))
plt.errorbar(x, y, 0.3, fmt='o’)
plt.plot(xfit, yfit, '-r', label = 'predicted', zorder = 10)
plt.plot(xfit, ytrue, '-k', alpha=0.5, label = 'true model', zorder = 10)
plt.legend()
plt.show()
```

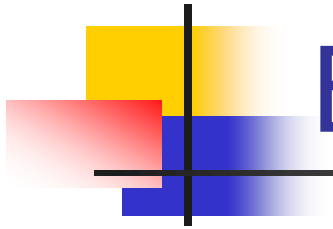

predicted
true model
2
1
0
−1
−2
0
2
4
6
8
10



Clustering

- Unsupervised learning: without labels of the data, put the data into different groups

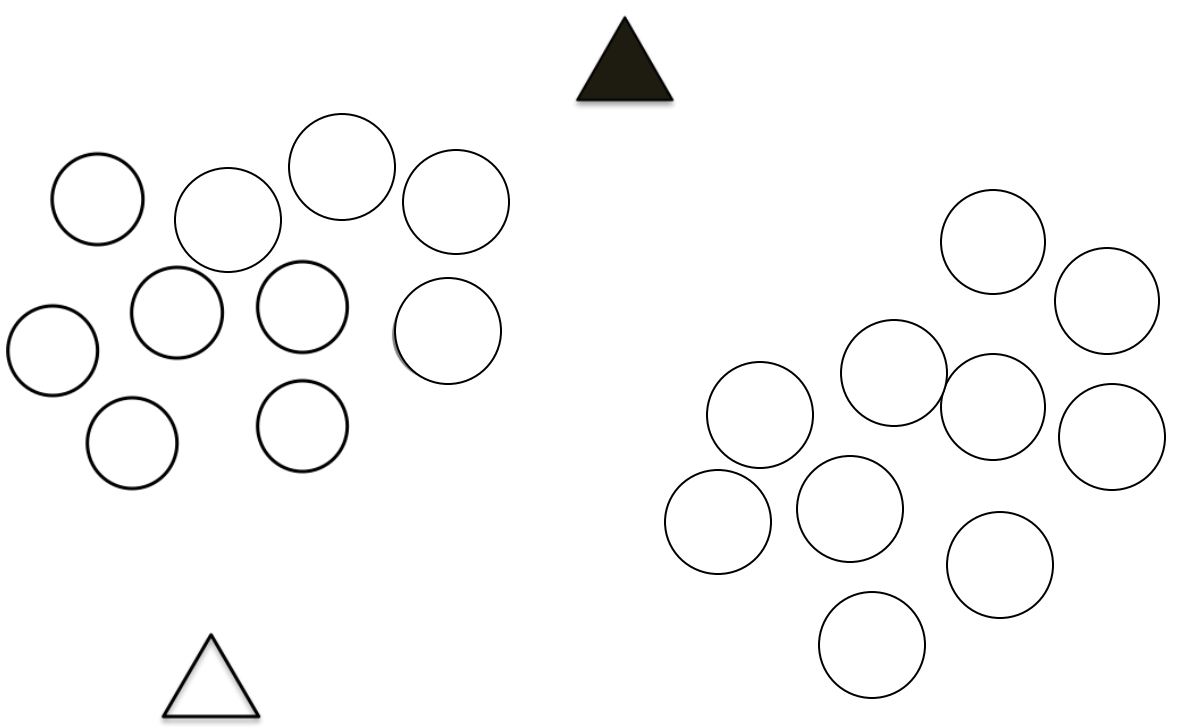
- K-means is an effective and commonly used algorithm due to its simple idea



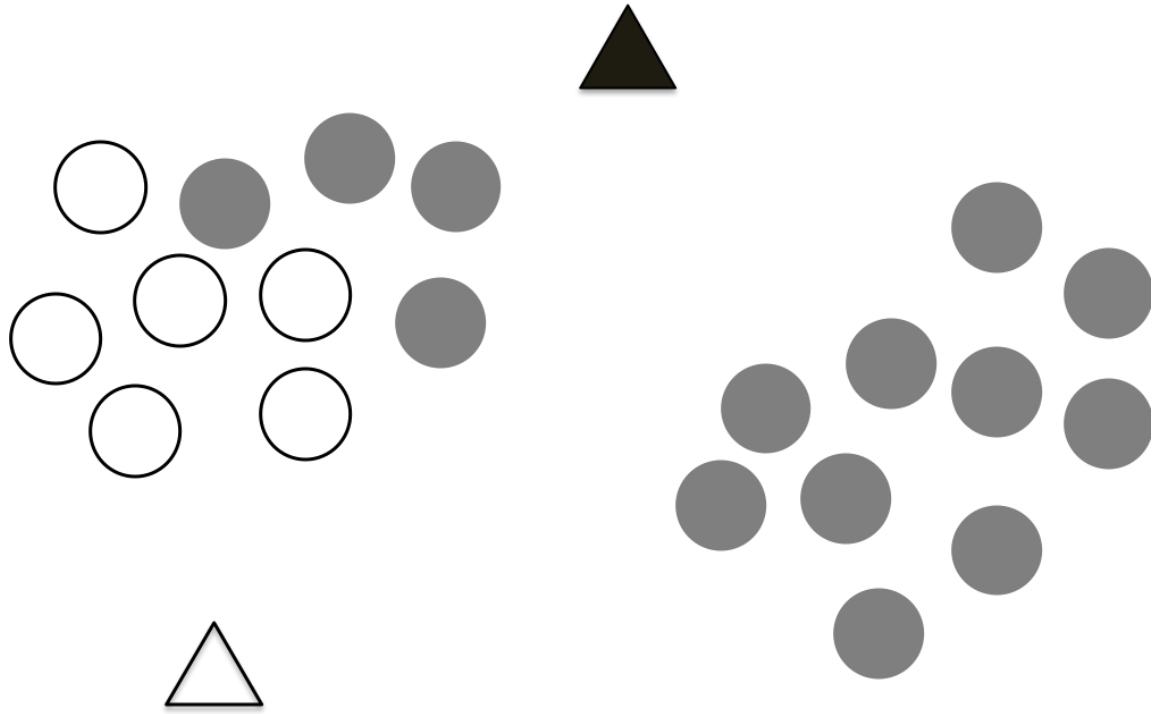
Basic idea

- Based on the distance of two points (e.g., Euclidean distance), if they are close to each other, they are similar and should be in the same group.
  - Step 1: Randomly drop K centroids
  - Step 2: Assign points to the K centroids
  - Step 3: update the centroids
  - Step 4: repeat 2 and 3 until the centroids do not move

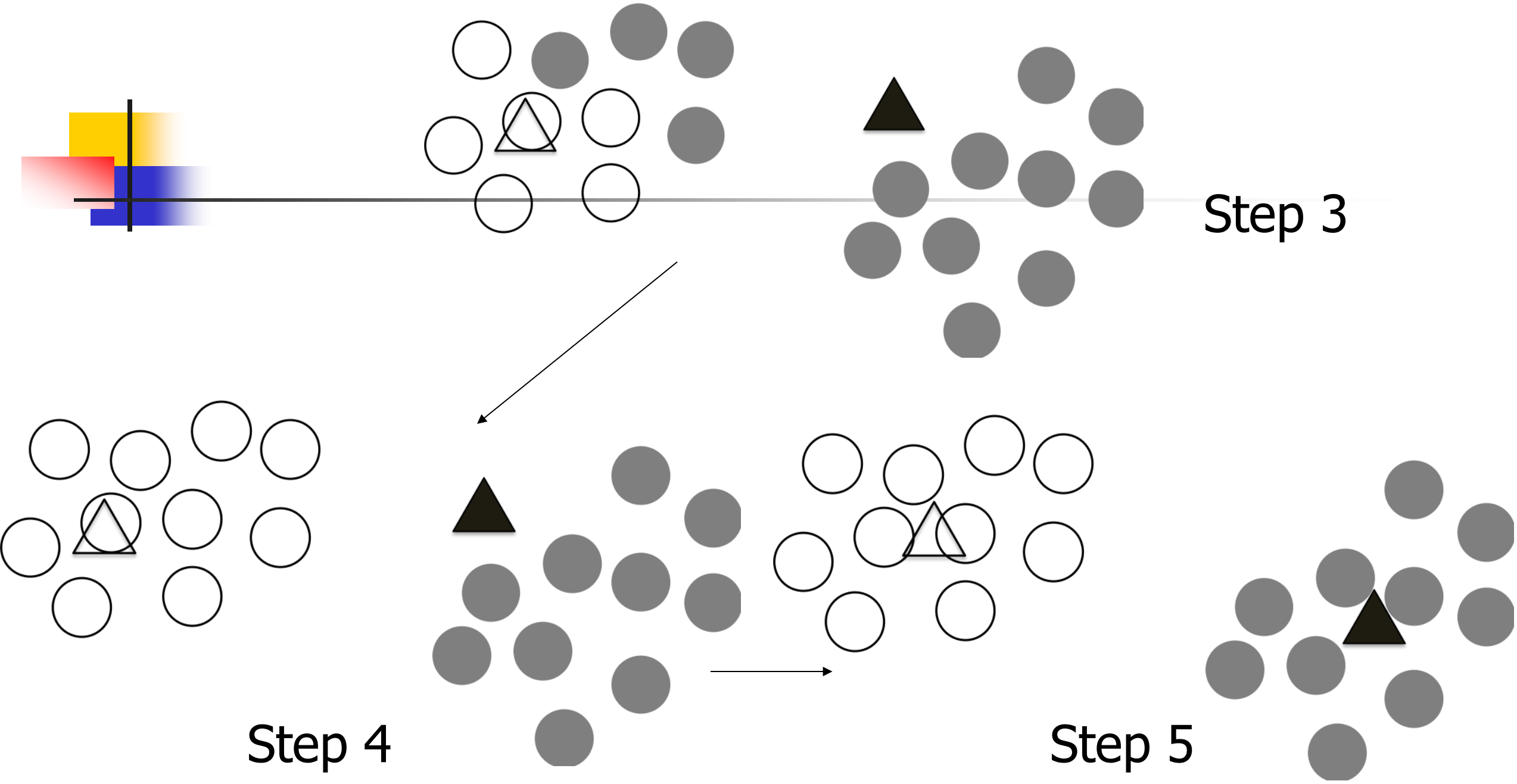
Illustration



Step 1



Step 2

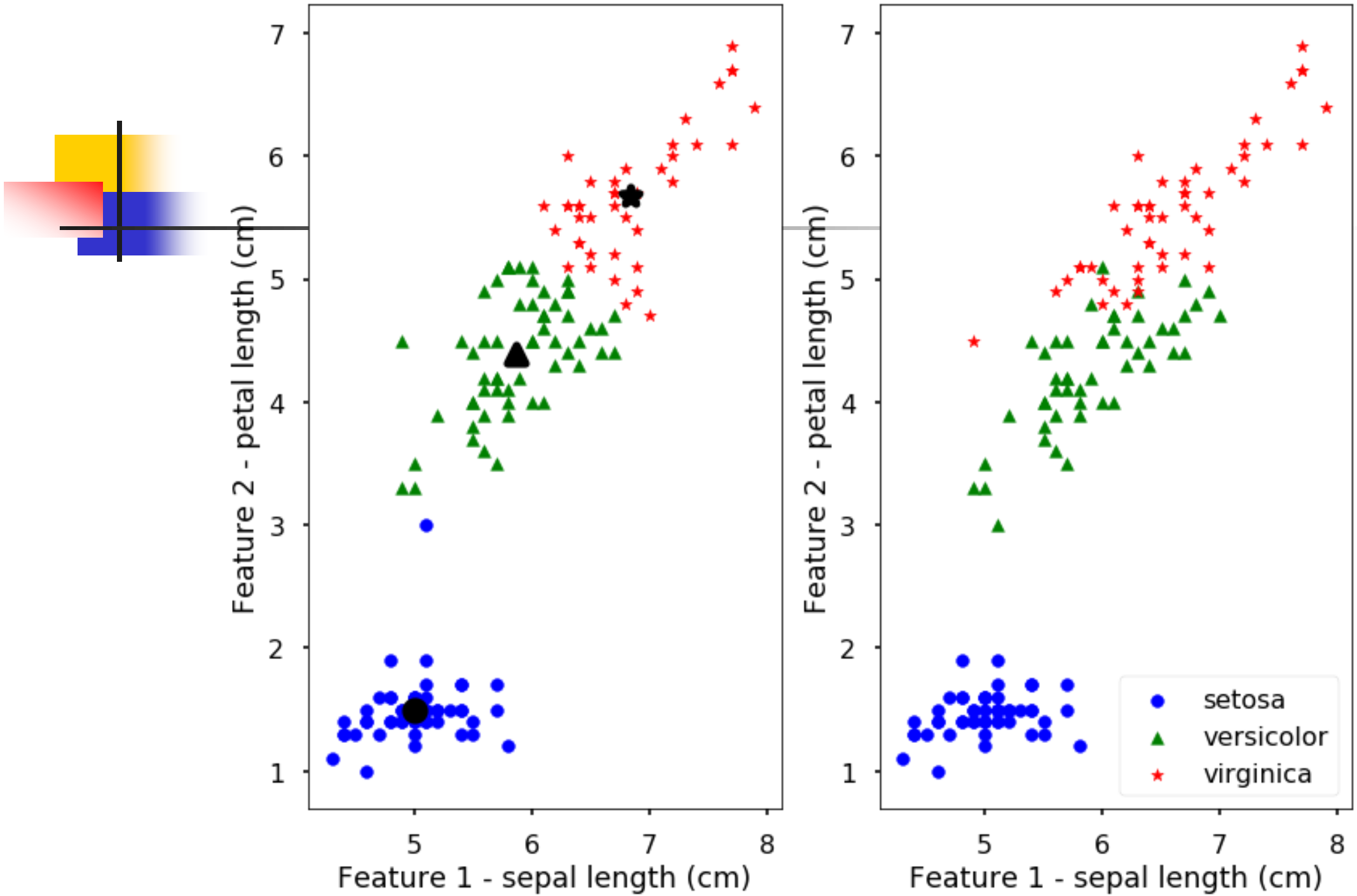

Step 3
Step 4
Step 5

Python implementation

```
from sklearn.cluster import KMeans
kmean = KMeans(n_clusters=3, random_state = 0)
kmean.fit(X)

print(kmean.labels_) # predicted labels of all data
print(kmean.cluster_centers_) # 3 centers of the clusters

new_points = np.array([[5, 2], [6, 5]])
kmean.predict(new_points)
```


Feature 2 - petal length (cm)
Feature 1 - sepal length (cm)
setosa
versicolor
virginica