NEAREST NEIGHBOR CLASSIFIER

CLASSIFICATION: DEFINITION

Given a collection of records (*training set*)

- Each record contains a set of *attributes*, one of the attributes is the *class*.

Find a *model* for class attribute as a function of the values of other attributes.

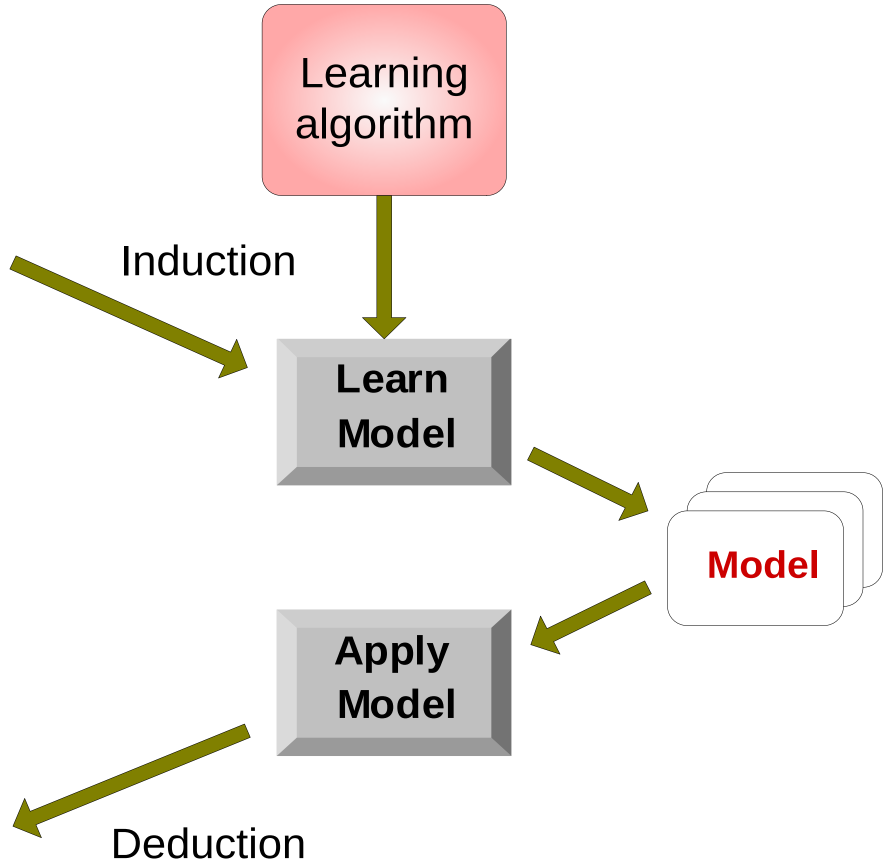
Goal: <u>previously unseen</u> records should be assigned a class as accurately as possible.

- A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

ILLUSTRATING CLASSIFICATION TASK

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set



Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

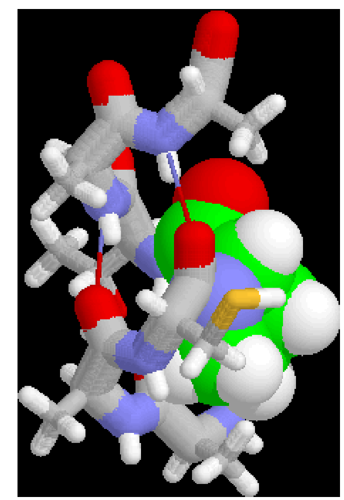
Test Set

EXAMPLES OF CLASSIFICATION TASK

Predicting tumor cells as benign or malignant

Classifying credit card transactions
as legitimate or fraudulent

Classifying secondary structures of protein
as alpha-helix, beta-sheet, or random
coil

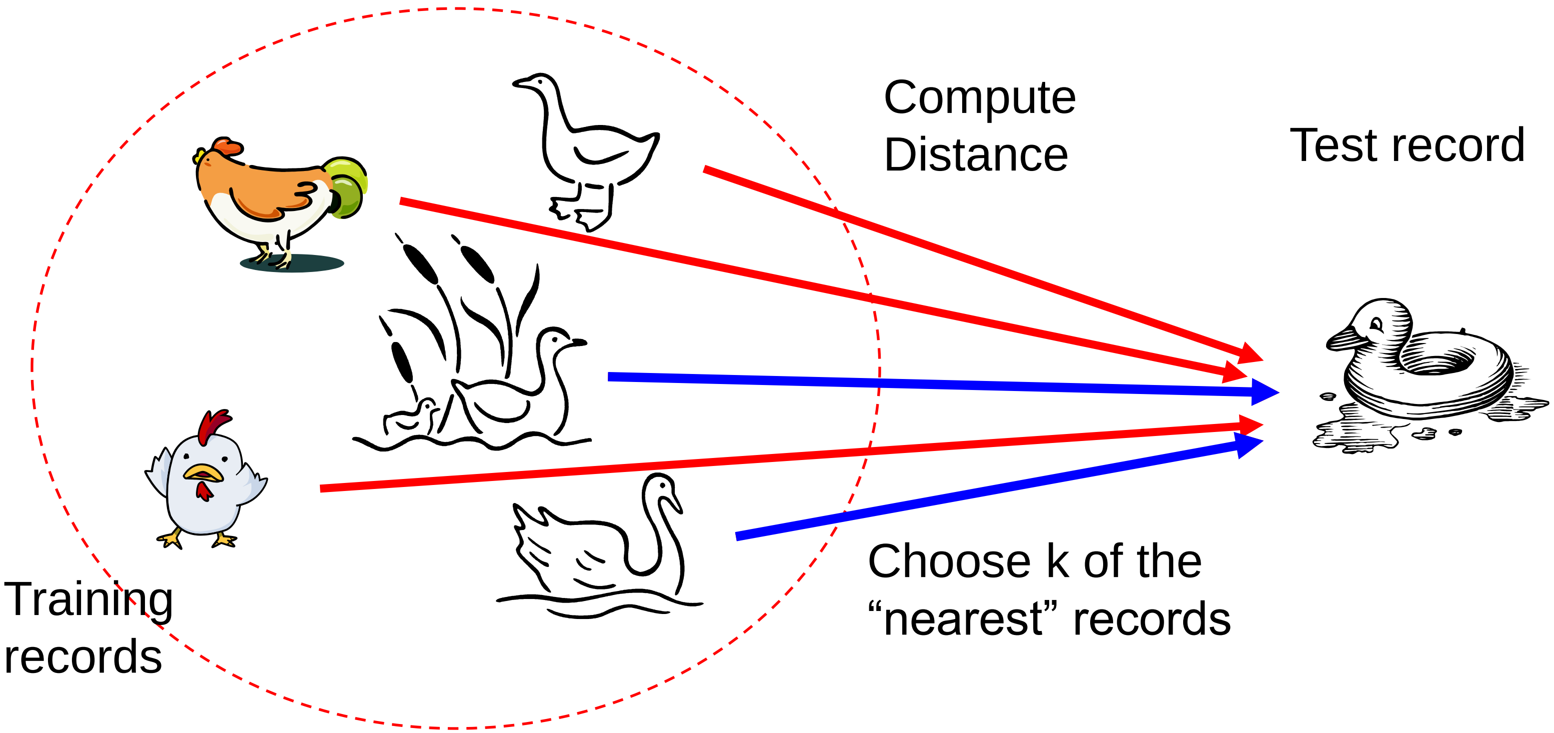


Categorizing news stories as finance,
weather, entertainment, sports, etc

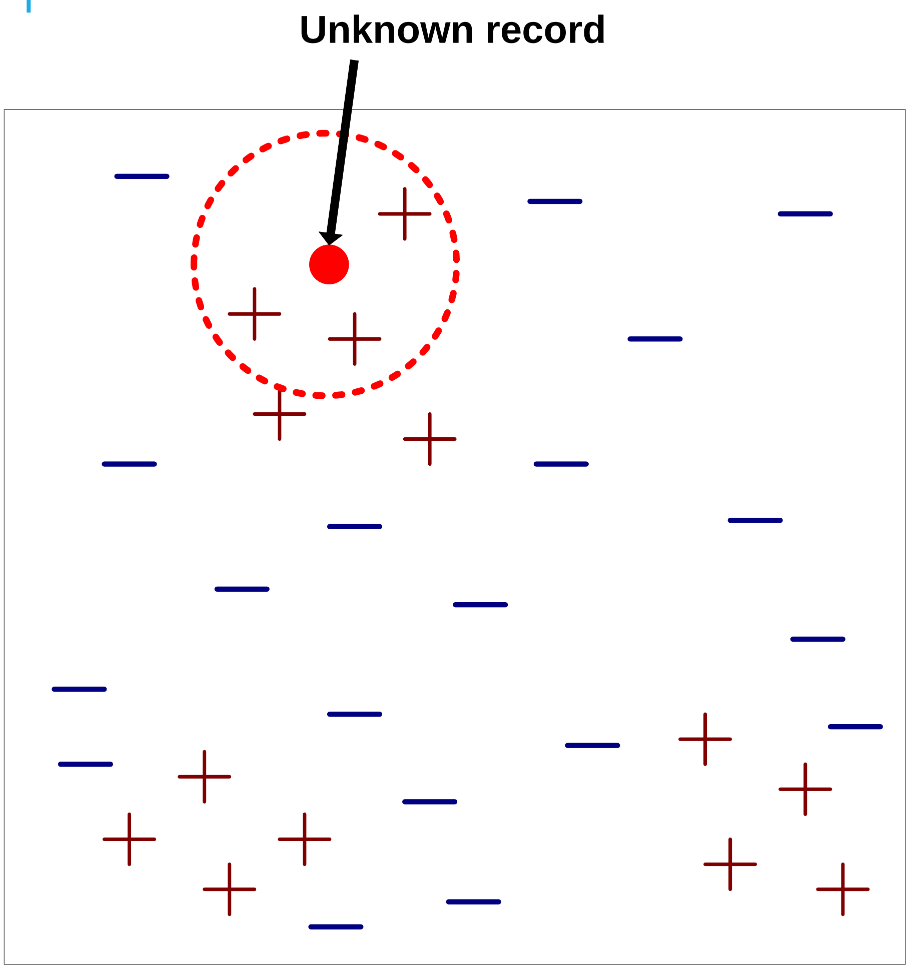
NEAREST NEIGHBOR CLASSIFIER

Basic idea:

- If it walks like a duck, quacks like a duck, then it's probably a duck



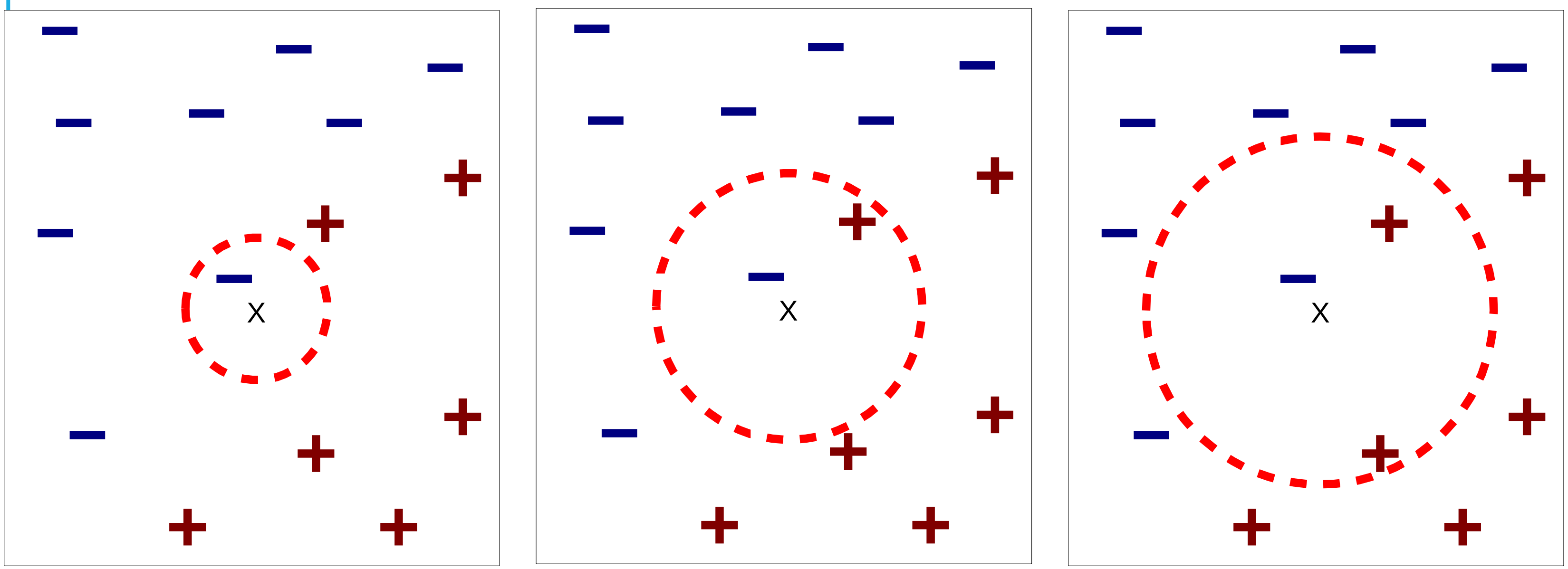
NEAREST-NEIGHBOR CLASSIFIER



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

DEFINITION OF NEAREST NEIGHBOR



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

NEAREST NEIGHBOR CLASSIFICATION

Compute distance between two points:

- Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

Determine the class from nearest neighbor list

- take the majority vote of class labels among the k-nearest neighbors
- Weight the vote according to distance
  - weight factor, $w = 1/d^2$

K-NEAREST NEIGHBOR (*K*-NN) ALGORITHM

For every point in dataset:

calculate the distance between X and the current point

sort the distances in increasing order

take k items with lowest distances to X

find the majority class among these items

return the majority class as our prediction for the class of X

HOW TO IMPLEMENT *K*-NN IN PYTHON

1. **Handle** Data: Open the dataset from CSV and split into test/train datasets.
2. **Similarity**: Calculate the distance between two data instances.
3. **Neighbors**: Locate *k* most similar data instances.
4. **Response**: getting the majority voted response from a number of neighbors.
5. **Accuracy**: Summarize the accuracy of predictions.
6. **Main**: Tie it all together.

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

HOW TO IMPLEMENT *K*-NN IN PYTHON

1. Handle Data: Open the dataset from CSV and split into test/train datasets.

```
import csv
import random
def loadDataset(filename, split, trainingSet=[] , testSet=[]):
    with open(filename, 'rb') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])
```

HOW TO IMPLEMENT *K*-NN IN PYTHON

2. Similarity: Calculate the distance between two data instances.

```
import math
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)
```

HOW TO IMPLEMENT *K*-NN IN PYTHON

3. Neighbors: Locate *k* most similar data instances.

```
import operator
def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

HOW TO IMPLEMENT *K*-NN IN PYTHON

4. Response: a function for getting the majority voted response from a number of neighbors

```
import operator
def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.iteritems(), key=operator.itemgetter(1), rever
    return sortedVotes[0][0]
```

HOW TO IMPLEMENT *K*-NN IN PYTHON

5. Accuracy: Summarize the accuracy of predictions

```python
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] is predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0
```

HOW TO IMPLEMENT *K*-NN IN PYTHON

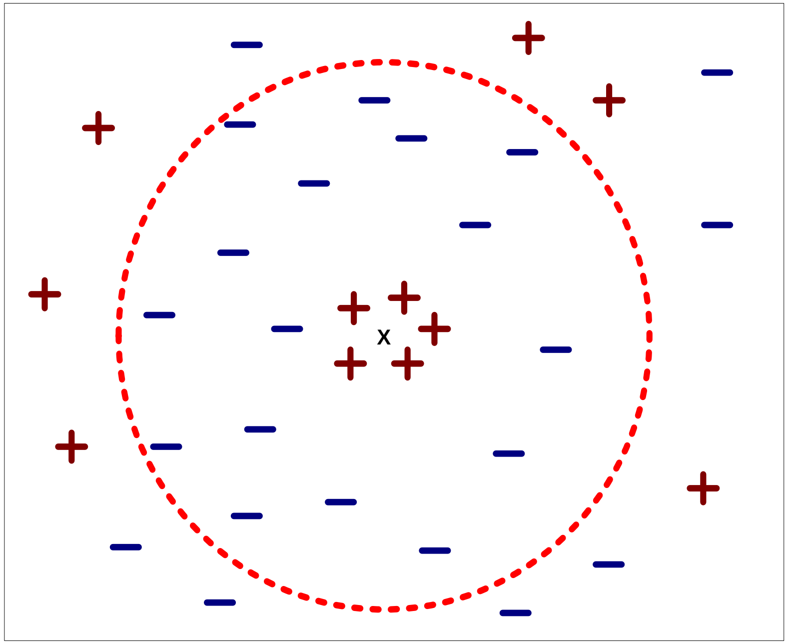
```
def main():
    # prepare data
    trainingSet=[]
    testSet=[]
    split = 0.67
    loadDataset('iris.data', split, trainingSet, testSet)
    print 'Train set: ' + repr(len(trainingSet))
    print 'Test set: ' + repr(len(testSet))
    # generate predictions
    predictions=[]
    k = 3
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: ' + repr(accuracy) + '%')

main()
```

```
...
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
> predicted='Iris-virginica', actual='Iris-virginica'
Accuracy: 98.0392156862745%
```

NEAREST NEIGHBOR CLASSIFICATION…

Choosing the value of *k*:

- If k is too small, sensitive to noise points
- If k is too large, neighborhood may include points from other classes



NEAREST NEIGHBOR CLASSIFICATION...

Scaling issues

- Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
- Example:
  - height of a person may vary from 1.5m to 1.8m
  - weight of a person may vary from 90lb to 300lb
  - income of a person may vary from \$10K to \$1M

NEAREST NEIGHBOR CLASSIFICATION…

Problem with Euclidean measure:

- High dimensional data
  - curse of dimensionality
- Can produce counter-intuitive results

1 1 1 1 1 1 1 1 1 1 1 0	vs	1 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1		0 0 0 0 0 0 0 0 0 0 0 1
d = 1.4142		d = 1.4142

- Solution: Normalize the vectors to unit length

OTHER DISTANCE MEASURES

Hamming Distance: Calculate the distance between binary vectors

Manhattan Distance: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance

Minkowski Distance: Generalization of Euclidean and Manhattan distance

NEAREST NEIGHBOR CLASSIFICATION…

k-NN classifiers are lazy learners

- It does not build models explicitly
- Unlike eager learners such as decision tree induction and rule-based systems
- Classifying unknown records are relatively expensive

REFERENCES

Tan, Steinbach, Kumar, Introduction to Data Mining, 2000

https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/