Machine Learning Algorythms
k-NN

Michel.Riveill@univ-cotedazur.fr

Objectives: Classification

- There are three methods to establish a classifier

a) Model a classification rule directly

Examples: **k-NN**, decision trees, SVM

b) Model the probability of class memberships given input data

Example: logistic regression, **perceptron** with the cross-entropy cost

c) Make a probabilistic model of data within each class

Examples: naive Bayes, hypothesis testing

a) and *b*) are examples of **discriminative** classification

c) is an example of **generative** classification

b) and *c*) are both examples of **probabilistic** classification

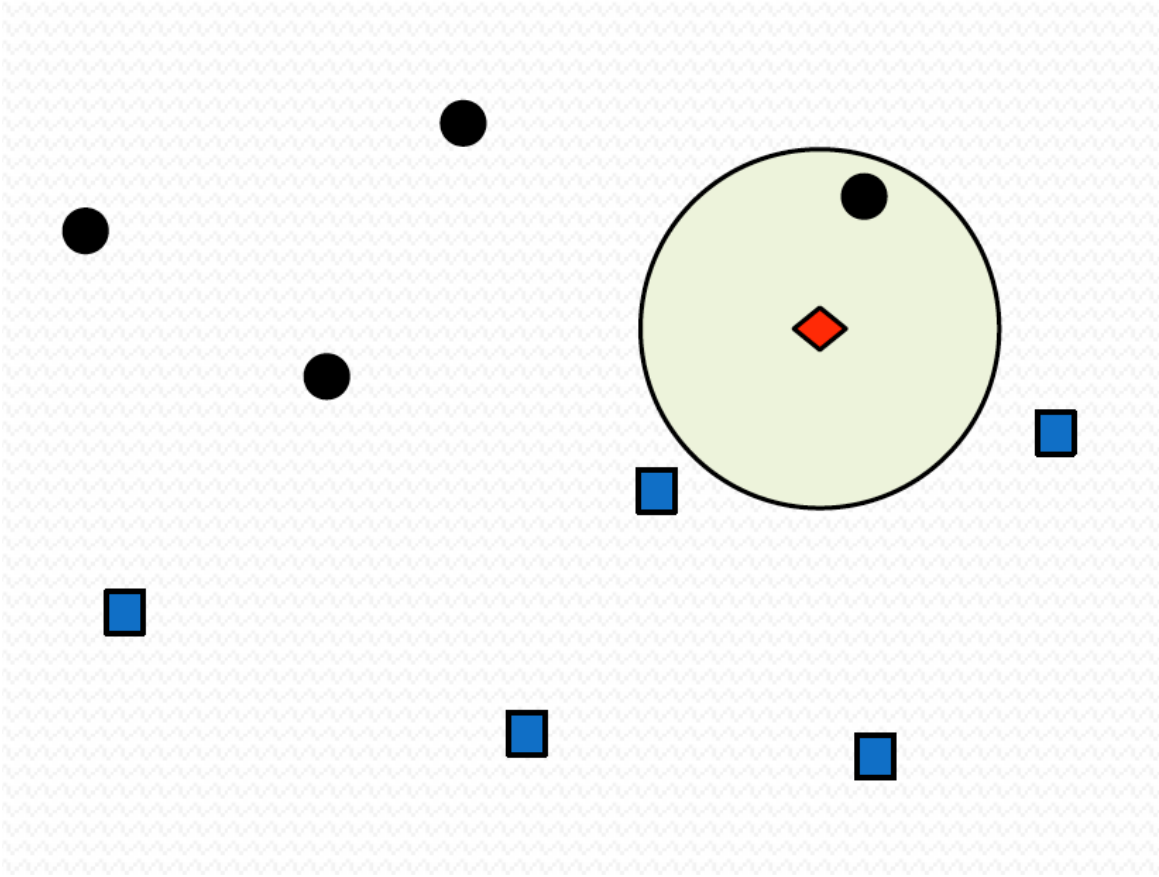
k-Nearest Neighbors (k-NN)

Instance-Based Learning

- kNN works like a classifier in supervised mode.
  - Have training examples: (x_i, y_i), i=1, …, N
    - x_i could have discrete or real value
  - Try to predict the class for new example x
    - $y=f(x)\in\{C_1, \cdots, C_c\}$
- The main idea to determine the class
  - Similar examples have similar label
  - Algorithm:
    1. Find most similar training examples x_n
    2. Classify x "like" these most similar examples
- Questions:
  - How to determine similarity?
  - How many similar training examples to consider?
  - How to resolve in consistencies among the training examples?

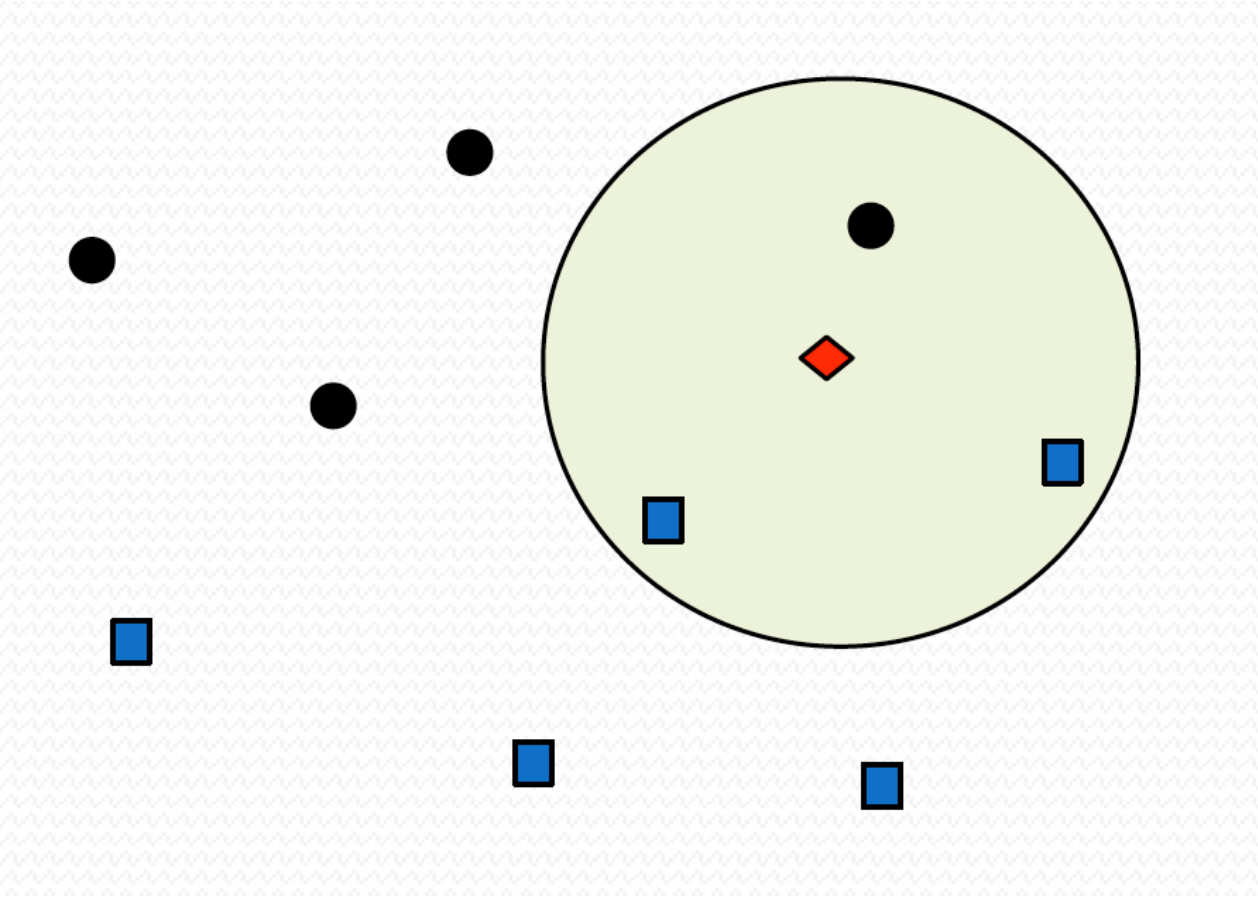
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point



3-Nearest Neighbors

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its k nearest neighbors



K-Nearest Neighbors (KNN)

- K-Nearest neighbour:
  - Given a query instance x,
  - First locate the k nearest training examples $x_1, x_2, \ldots, x_k$
  - Classification:
    - Discrete values target function
    - Take vote among its k nearest neighbors
  - Regression
    - Real valued target function
    - Take the mean of the f values of the k nearest neighbors
- Remember. We have to answer to:
  1. How to determine similarity?
  2. How many similar training examples to consider?
  3. How to resolve in consistencies among the training examples?

1. How to determine similarity?

It is possible to use any function that respects the following principles

- It's from 'distance properties'
  - Non-negative: $d(i, j) > 0$
  - $d(i, i) = 0$
  - Symmetry: $d(i, j) = d(j, i)$
  - Triangle inequality: $d(i, k) \leq d(i, j) + d(j, k)$

- Some distance
  - Euclidian distance: $d(x, y) = \sqrt{\sum (x_i - yi)^2}$
  - Manhattan distance ("city-block"): $d(x, y) = \sum |x_i - yi|$

  - Uniform or weighted distance
    - Weigted: assign weights to the neighbors based on their "distance" from the query point
      - Generally weight $= {}^{1}/_{distance}$

Knn need to normalize each feature

- The distance measure is influenced by the units of the different variables, especially if there is a wide variation in units.
  - Variables with "larger" units will influence the distances more than others.
  - $d_{i,j} = \sqrt{\sum(x_i - xj)^2}$
- An example

	Income in $	Age
Carry	$31 779	36
Sam	$32 739	40
Miranda	$33 880	38

- d(Carry, Sam) = $((31779 - 32739)^2 + (36 - 40)^2)^{1/2}$
  = $((\mathbf{960})^2 + (4)^2)^{1/2} = (921600 + 16)^{1/2} = 960{,}008$
  ± difference of income
- In order to take into account all the features, the dataset must be standardized/normalized.

Knn need to normalize each feature

	Income in $	Age	Normalized income	Normalized Age
Carry	$31 779	36	0	0
Sam	$32 739	40	0,46	1
Miranda	$33 880	38	1	0,5

With un-normalized features

	distance	rank
d(Carry,Sam)	**960**	**1**
d(Sam,Miranda)	1 141	2
d(Miranda,Carry)	2 101	3

With normalized features

	distance	rank
d(Carry,Sam)	1,1	3
d(Sam,Miranda)	**0,73**	**1**
d(Miranda,Carry)	1,12	2

Normalization / Standartization

- Normalization
  - Rescale the data in the *range of [0,1]*

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

  - from sklearn.preprocessing import MinMaxScaler

- Standardization
  - Rescale the data in order to have the properties of a standard normal distribution
    - Mean, $\mu=0$
    - Standard deviation, $\sigma=1$

$$z = \frac{x - \mu}{\sigma}$$

  - Rescale the the data in the *range of [-1,1]*
  - from sklearn.preprocessing import StandardScaler

2. How many similar training examples to consider?

Selecting the Number of Neighbors

- Increase k:
  - Makes KNN less sensitive to noise
- Decrease k:
  - Allows capturing finer structure of space
- Hard to tune!

3. How to resolve consistencies among the training examples?

- Try to use more neighbours
- But give less weight to the far neighbours compared to the close neighbours

- Hard to tune to!

K-Nearest Neighbors in python

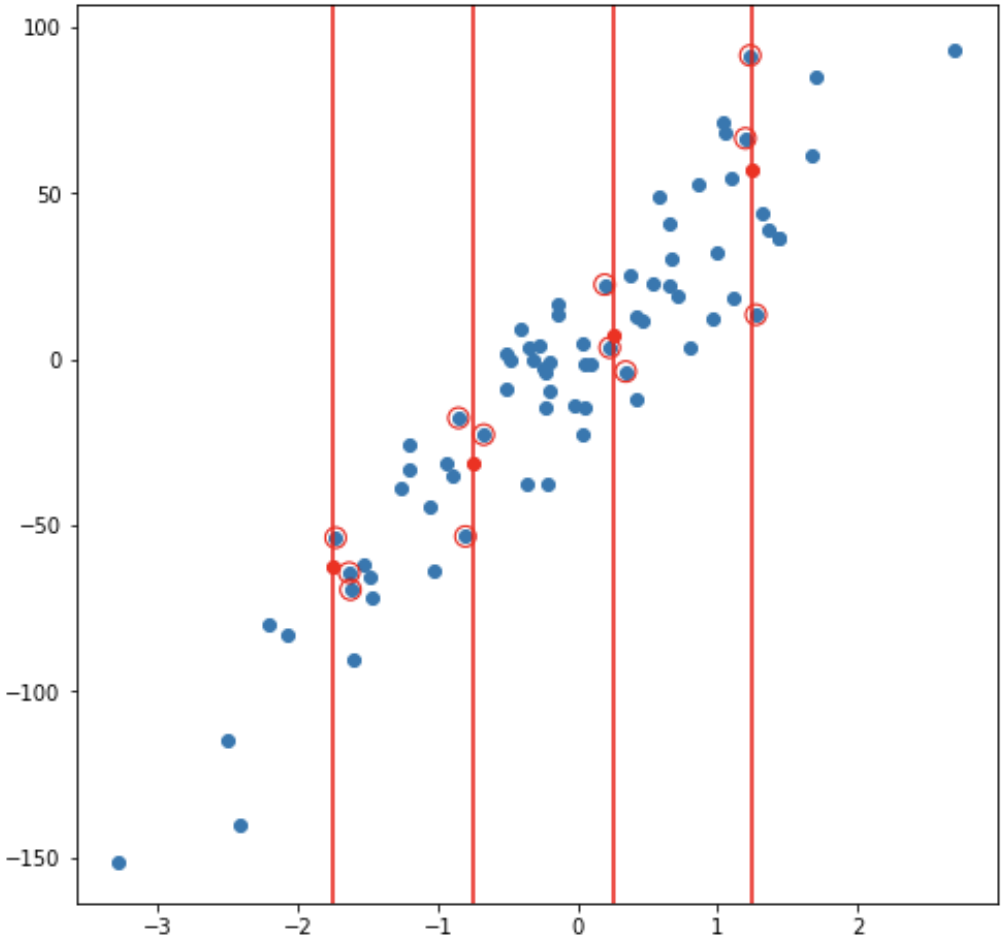
- from sklearn.neighbors import KNeighborsClassifier
  - 3 main parameters
    - n_neighbors (k)
    - p (power): $(\sum|a_i - bi|^p)^{1/p}$
    - weights: with weigt ('distance') or without ('uniform')

- clf = KNeighborsClassifier(n_neighbors=7, p=2)
  - # p = power parameter for the Minkowski metric.
  - # p = 1 --> manhattan_distance (l1)
  - # p = 2 --> euclidean_distance (l2)

- clf.fit(X_train, y_train)
- y_pred = clf.predict(X_test)

Regression with k-NN

- Exactly the same approach
- use the neighbor label value to calculate the value of a new point



- from sklearn.neighbors import KNeighborsRegressor

PRO of k-NN

- Highly efficient inductive inference method for noisy training data and complex target functions
- Learning is very simple
- k-NN is simple to understand and implement
- k-NN has no assumptions other than the need to standardize features.
- No training step: each new entry is labelled according to these neighbours
- It is possible to enrich the model with run-of-river data.
- No specific work to do to go from a problem with 2 classes, multiclasses or regression
- A very wide variety of distances can be chosen (although we mainly looked at Minkowski)
- It's an excellent algorithm for replacing missing values...

CONS of k-NN

- Need a distance that "matches" the target function, possibly the distance depends on the feature
- k-NN must read the whole dataset for each prediction. very expensive for large datasets
- k-NN works well with a small number of features, but the accuracy degrades as the number increases.
- k-NN works well with a properly balanced dataset
- Need to standardize the data to give equal weight to each feature
- k-NN does'nt work with missing value
- k-NN is very sensitive to outliers because it simply chooses neighbors based on distance criteria.

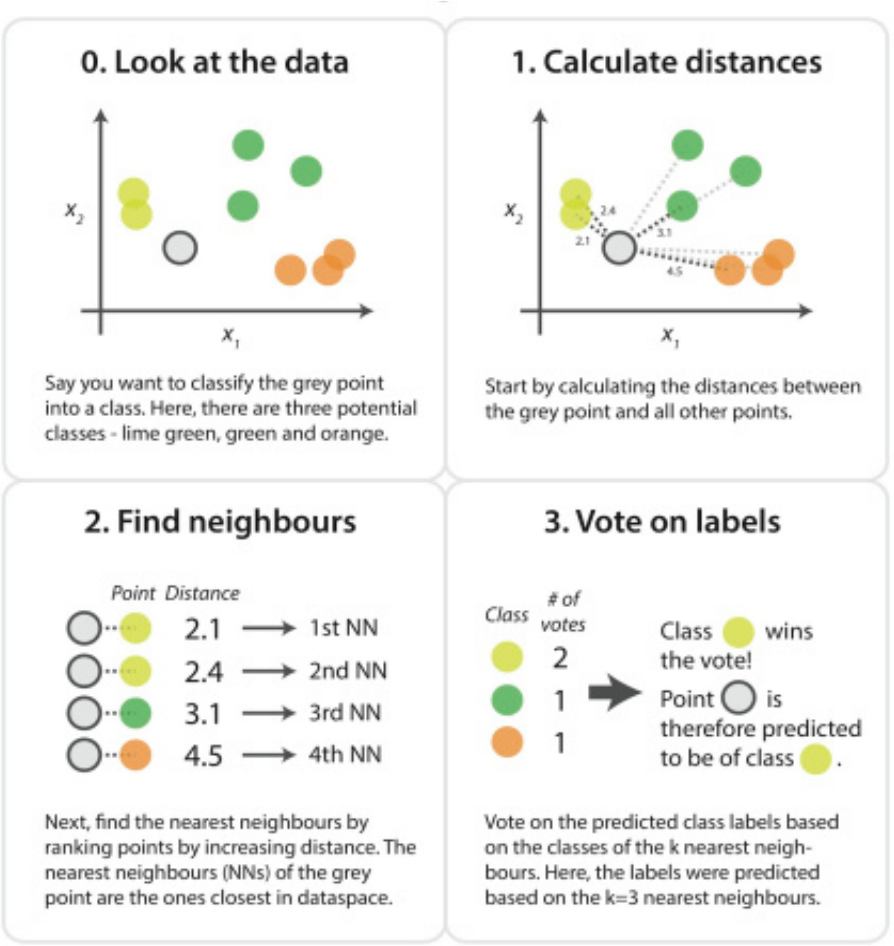
- But one of the main problems with k-NN is to choose the optimal number of neighbors to be considered when classifying the new data entry.

Let's look under the bonnet

- The main problem with kNN is the search for neighbours. There are two main approaches:
- Exact methods
  - The distance is calculated with all the neighbours and the k closest to them is selected.
    - Linear Search
    - Space partitionning
- Approximate methods
  - An approximate nearest neighbour search algorithm is allowed to return points whose distance from the query is at most c times the distance between the query and its nearest points.

Linear search

- No training step
- Very easy to implement but not efficient for large dataset



- Complexity – brut force method
  - n: number of points in the training dataset
  - d: data dimensionality
  - k: number of neighbors that we consider for voting

  - Training time complexity: O(1)
  - Prediction time complexity: O(k * n * d)

Space partitioning approach

- Binary Space Partition Tree (BSP Tree)
  - Divide recursively the space in 2
  - The first node represents all the space available
  - Each node can contain two child (dividing itself in two equal part)
  - A node represents a limited space
  - Once the tree has been built, it is easier to search for nearby k neighbours since you search in a sub-space.
- Complexity - k-d tree method – split in k, not in 2
  - Training time complexity: O(d * n * log(n))
  - Prediction time complexity: O(k * log(n))
- Complexity - ball tree method – group object aroud ball
  - Training time complexity: O(d * n * log(n))
  - Prediction time complexity: O(k * log(n))

Sklearn knn – main parameters

- **n_neighbors**, *default=5*
- **weights,** {*'uniform', 'distance'*} *or callable, default='uniform'*
  - 'distance' : weight points by the inverse of their distance.
- **Algorithm,** {*'auto', 'ball_tree', 'kd_tree', 'brute'*}, *default='auto'*
  - 'ball_tree' will use BallTree (space partitionning)
  - 'kd_tree' will use KDTree (space partitionning)
  - 'brute' will use a brute-force search (linear search)
  - 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.
- **leaf_size**, *default=30*
  - Leaf size passed to BallTree or KDTree.
  - This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- **p***i, default=2*
  - Power parameter for the Minkowski metric.
  - p = 1, manhattan_distance (l1), and p = 2, euclidean_distance (l2)
- **Metric,** *str or callable, default='minkowski'*
- **metric_params,** *dict, default=None*
  - Additional keyword arguments for the metric function.

Approximate Nearest Neighbors (ANN)

- Approximate Nearest Neighbor techniques speed up search by preprocessing the data into an efficient index
- Generally uses these phases:
  - Vector Transformation — applied on vector before they are indexed,
    - Dimension reduction and vector rotation.
  - Vector Encoding — applied on vectors in order to construct the actual index for search
    - Use data structure-based techniques like Trees, LSH and Quantization (technique to encode the vector to a much more compact form)
  - None Exhaustive Search Component — applied on vectors in order to avoid exhaustive search
    - Amongst these techniques there are Inverted Files and Neighborhood Graphs
- Not directly available in sklearn but see
  - https://scikit-learn.org/0.18/auto_examples/neighbors/plot_approximate_nearest_neighbors_scalability.html