



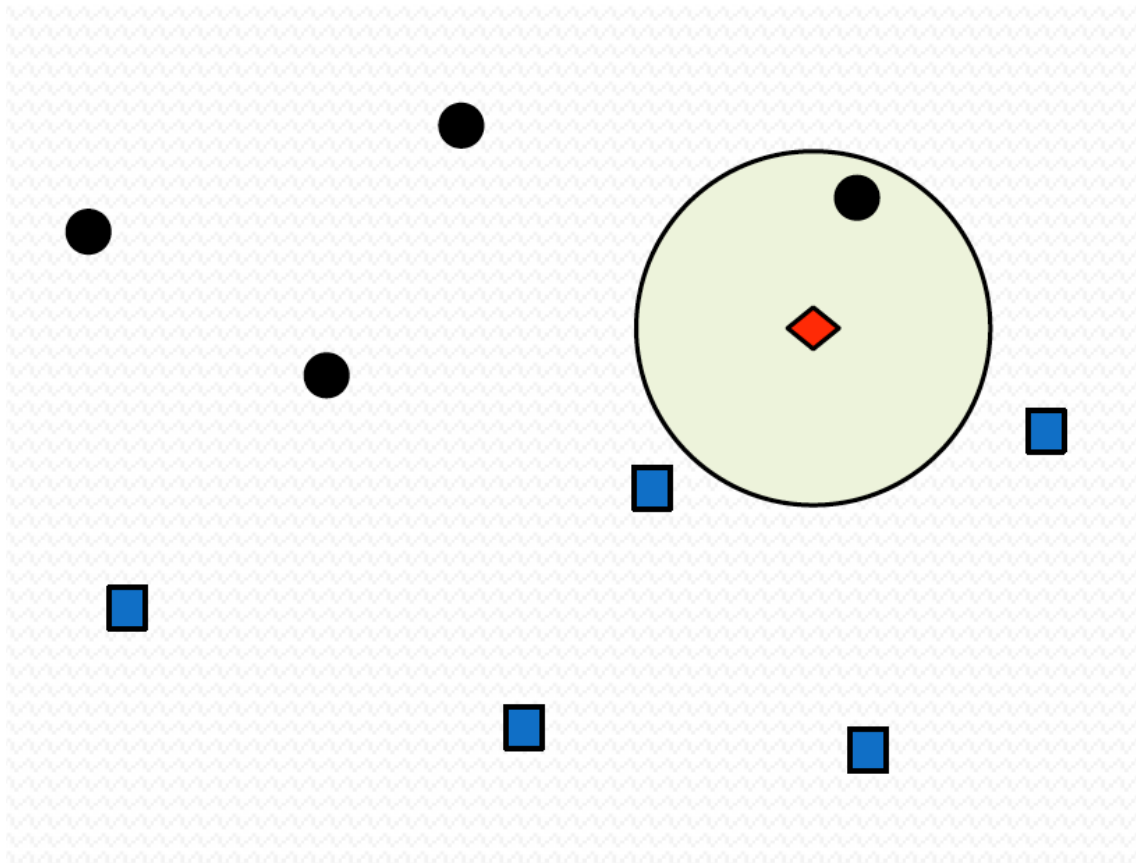
k-Nearest Neighbors (k-NN)

Instance-Based Learning

- ▶ Knn works like a classifier in supervised mode.
 - ▶ Have training examples: $(x_i, y_i), i=1, \dots, N$
 - ▶ x_i could have discrete or real value
 - ▶ Try to predict the class for new example x
 - ▶ $y=f(x) \in \{C_1, \dots, C_d\}$
- ▶ 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 similar example?
 - ▶ How many similar training examples to consider?
 - ▶ How to resolve inconsistencies among the training examples?

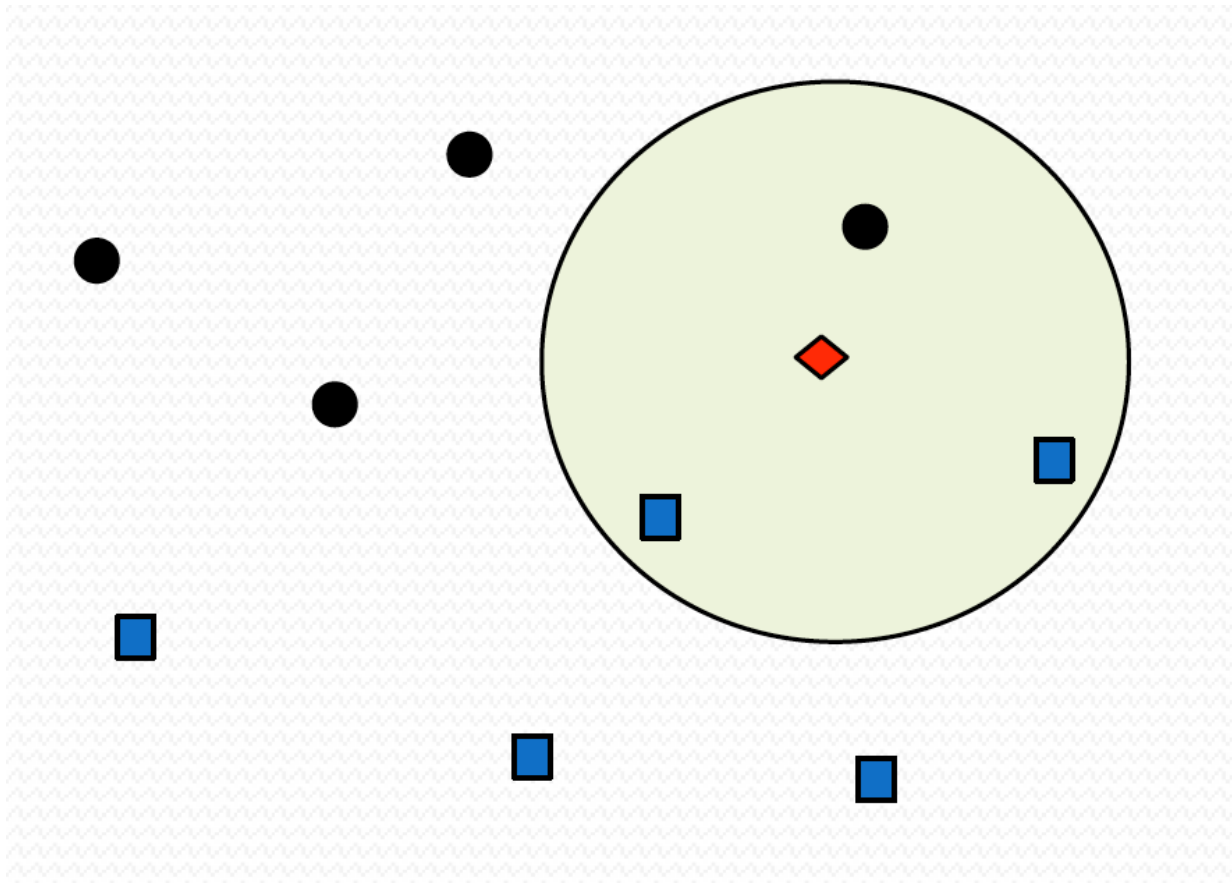
1-Nearest Neighbor

- ▶ 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 3 nearest neighbors



K-Nearest Neighbors (KNN)

- ▶ K-Nearest neighbors:
 - ▶ Given a query instance x ,
 - ▶ First locate the k nearest training examples x_1, x_2, \dots, x_k
- ▶ Classification:
 - ▶ Discrete values target function
 - ▶ Take vote among its k nearest neighbors
- ▶ Extension for regression
 - ▶ Real valued target function
 - ▶ Take the mean of the values of the k nearest neighbors
- ▶ Remember. We have to answer to:
 1. How to determine similar example?
 2. How many similar training examples to consider (value of k)?
 3. How to resolve inconsistencies among the training examples (how to avoid noise)?



1. How to determine similarity?

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

- ▶ It's from 'distance properties'

- ▶ Non-negative: $d(i, j) \geq 0$
- ▶ $d(i, i) = 0$
- ▶ Symmetry: $d(i, j) = d(j, i)$
- ▶ Triangle inequality: $d(i, k) \leq d(i, j) + d(j, k)$

- ▶ Difference between distance and similarity

- ▶ $d(i, i) = 0$ but $sim(i, i) = 1$
- ▶ $d(i, j) \in [0, +\infty[$ but $sim(i, j) \in [-1, 1]$
- ▶ From distance to similarity
 - ▶ Normalize : $d_{norm} = \frac{d}{\max(d)}$

- ▶ Then : $sim = 1 - d_{norm}$

- ▶ General approach : use a **strictly monotone decreasing** function f

- ▶ Work in both sense
 - ▶ $d = f(sim)$ and $sim = f(d)$
 - ▶ Some example : $\frac{1}{a+x}$ or e^{-x^a}

1. How to determine similarity?

- ▶ Some distance

- ▶ Manhattan distance (“city-block”): $d(x, y) = \sum |x_i - y_i|$
- ▶ Euclidian distance: $d(x, y) = \sqrt{\sum (x_i - y_i)^2}$
- ▶ Levenshtein distance: measure the difference between two sequences

$$\text{lev}(a, b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise} \end{cases}$$

1. How to determine similarity?

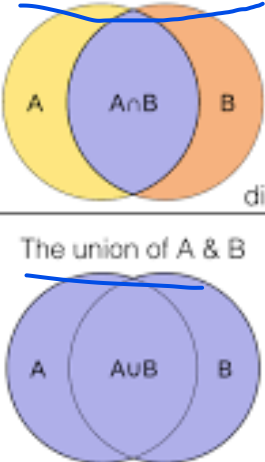
- ▶ Some similarity
- ▶ Cosine similarity (between vectors)

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

- ▶ Jaccard Index (between sets)

$$J(A, B) = \frac{\text{The intersect of A \& B}}{\text{The union of A \& B}}$$

division



The diagram consists of two Venn diagrams. The top diagram shows two overlapping circles, A (yellow) and B (orange). The intersection is shaded blue and labeled 'A ∩ B'. The entire area covered by both circles is labeled 'The union of A & B'. The bottom diagram shows the same two overlapping circles, but the entire area covered by both is shaded blue and labeled 'A ∪ B'. A horizontal line with the word 'division' is placed between the two diagrams, indicating the ratio of the intersection to the union.

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.

- ▶ $di,j = \sqrt{\sum (x_i - x_j)^2}$

- ▶ An example

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

- ▶ $d(\text{Carry}, \text{Sam}) = ((31779 - 32739)^2 + (36 - 40)^2)^{1/2}$
 $= ((960)^2 + (4)^2)^{1/2} = (921600 + 16)^{1/2} = 960,008$
± difference of income

- ▶ In order to take into account all the features, the dataset must be 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

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!
- ▶ The main problem is to find the nearest neighbours efficiently
 - ▶ not covered in this course

3. How to resolve inconsistencies among the training examples?

- ▶ Try to use more neighbors
- ▶ But give **less weight to the far neighbors** compared to the close neighbors
- ▶ Hard to tune to!

- 
- ▶ Weighed neighbors = soft voting
 - ▶ i.e. Weighed contribution of each neighbors

K-Nearest Neighbors with sklearn

- ▶ from sklearn.neighbors import KNeighborsClassifier
 - ▶ 3 main parameters
 - ▶ Choose the neighbors: n_neighbors (k)
 - ▶ Choose the distance: p (power): $(\sum |a_i - b_i|^p)^{1/p}$ for Minkowski distance
 - p==1: Manhattan
 - p==2: Euclidian ✓
 - Or your own distance
 - ▶ Choose the proximity weight
 - with weight ('distance') or without weight ('uniform') or you own function
- ▶ clf = KNeighborsClassifier(n_neighbors=5, weights='uniform', p=2)
- ▶ clf.fit(X_train, y_train)
- ▶ y_pred = clf.predict(X_test) or clf.predict_proba(X_test)

PRO of k-NN

- ▶ Highly efficient inductive inference method for noisy training data and complex target functions
- ▶ 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 neighbors
- ▶ 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, multiclass or regression
- ▶ A very wide variety of distances can be chosen (although we mainly looked at Minkowski)

CONS of k-NN

▶ Need a distance that "matches" the target function

▶ possibly the distance depends on the feature

▶ **k-NN works well with a properly balanced dataset**

▶ Very expensive for large datasets

▶ **k-NN works well with a small number of features**

▶ Need dimension reduction for high-dimensional data (more than 10) in order to avoid the effects of the curse of dimensionality.

▶ Use PCA or LDA

▶ **k-NN works well with a properly balanced dataset**

▶ **Need to standardize the data to give equal weight to each feature**

▶ **k-NN doesn't 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.

k-NN extension

Approximate Nearest Neighbors

- ▶ KNN (K-Nearest Neighbors) is Dead!
 - ▶ <https://pub.towardsai.net/knn-k-nearest-neighbors-is-dead-fc16507eb3e>
- ▶ Comprehensive Guide To Approximate Nearest Neighbors Algorithms
 - ▶ <https://towardsdatascience.com/comprehensive-guide-to-approximate-nearest-neighbors-algorithms-8b94f057d6b6>
- ▶ Approximate Nearest Neighbor Search in High Dimensions
 - ▶ <https://arxiv.org/abs/1806.09823>

k-NN extension

Missing data imputation

- ▶ The KNN algorithm helps to impute missing data by finding the closest neighbors

- ▶ Imputs missing data with non-missing values from neighbors.

- ▶ from sklearn.impute import KNNImputer

- ▶ **n_neighbors**: Number of neighboring samples to use for imputation.

- ▶ **Weights**: Weight function used in prediction.

- ▶ 'uniform' : All points in each neighborhood are weighted equally.

- ▶ 'distance' : weight points by the inverse of their distance.

The lab of today

→ Short lecture... but long lab

- ▶ Part I. K-nearest neighbors for classification
 - ▶ Part II. K-nearest neighbors for regression
- } Read, understand and complete the code
-
- ▶ Part III.
 - ▶ Step I: Build a pipeline
 - ▶ Impute missing value
 - ▶ Normalize data
 - ▶ Predict with knn model
 - ▶ Evaluate your pipeline with default parameters
 - ▶ Search best hyper-parameters
 - ▶ Plot confusion matrix
 - ▶ Print classification report
- Put comments on
your notebook and
submit it } Lab part
-
- ▶ Part IV. Read paper « KNN (K-Nearest Neighbors) is Dead! »
 - ▶ Try to understand ANN (Approximate Nearest Neighbors)



***Some complements
to help you revise***

Exercise: data

- ▶ The following table contains data on individuals in a population described by two attributes: attribute 1 and attribute 2. The class of an individual can be: C1, or C2, ... or C6.

N°	Attribut 1	Attribut 2	Class
1	1	2	C1
2	2	6	C1
3	2	5	C2
4	2	1	C3
5	4	2	C5
6	5	6	C4
7	6	5	C3
8	6	1	C6



Exercise: questions

1. Dataset

- ▶ Plot the data from the previous table.

2. 3-NN Manhattan

- ▶ We want to classify a new individual U with attributes (1, 4) using the KNN method. What will be the class of U if we choose $k=3$ and Manhattan distance. Justify.

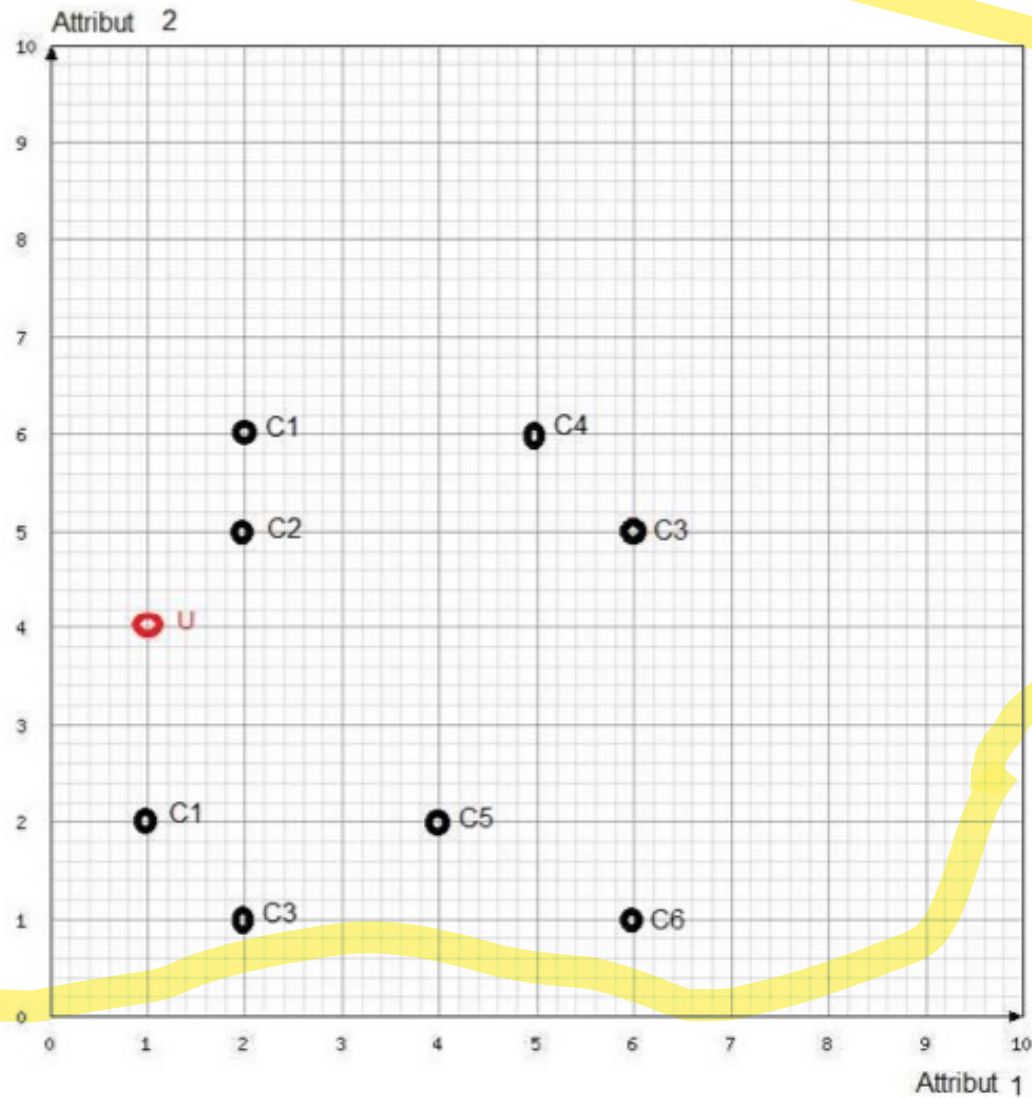
3. 3-NN Euclidian

- ▶ We want to classify a new individual U with attributes (1, 4) using the KNN method. What will be the class of U if we choose $k=3$ and Euclidian distance. Justify.

4. Weigted 3-NN

- ▶ We now use the variant of KNN which uses the distance $1/d^2$ (inverse of the distance squared) to calculate the neighbours. The distance is the Euclidian one. What will be the class of U with $k=3$? Justify.

Exercise: solutions



Exercise: solutions (cont')

▶ *Euclidian distance*

- $\text{Dist}(U, P1) = 2$
- $\text{Dist}(U, P2) = 2.24$
- $\text{Dist}(U, P3) = 1.41$
- $\text{Dist}(U, P4) = 3.16$
- *Etc. The other points are further on.*

▶ Point U is class C1 (2 votes for C1, against 1 vote for C2)

▶

▶ *Weigthed Euclidian distance*

- ▶ For class 1, the weights are $1/(2*2)$ and $1/(2.24*2.24) \rightarrow 0.25+0.20 = 0.45$
- ▶ For class 2, the weight is $1/(1.41*1.41) = 0.5$
- ▶ The U-point will therefore be assigned to class C2