k-NEAREST-NEIGBORS

Supervised lazy learner classifiers

Marcus Birkenkrahe

March 7, 2023

k-nearest neighbors

- Lecture notes in Markdown file (`4_knn.md`)
- Source: Lantz (2019), chapter 3, pp. 65-89
- See also: DataCamp assignment chapter 1

What you will learn

- Concepts for "lazy learners" classifiers
- How to easure similarity using distance
- Demo: cancer classification using R
- Exercises in GitHub as Org-mode file
- Solutions in GitHub as PDF files

The basic idea of this class of "lazy learners" is that things that are alike are likely to have properties that are alike.

ML uses this principle to classify data by placing it in the same category as similar, or "nearest" neighbors.

The choice of similarity measure and the choice of the parameter k are crucial.

Nearest neighbor classification

- Classify unlabeled (unknown) examples
- Assign similar labeled (known) examples
- Human examples: reading, eating, meeting
- Simple but powerful methods

Remember our terminology:

1. Unit of observation, e.g. email/patients
2. Examples or instances, e.g. message/patient sample
3. Features: e.g. words/characteristics of message/patient

With nearest-neighbor classification, computers apply a human-like ability to recall past experiences to make conclusions about current circumstances. Human examples:

- Reading: making sense of sentences and words by context.
- Eating: "dark restaurants" where you are totally blind
- Meeting: making connections based on who's next to whom

Nearest-neighbor applications

- Computer vision and facial recognition
- Recommender systems
- Genetic data pattern matching
- Computer vision applications, including optical character recognition and facial recognition in both still images and video

- Recommendation systems that predict whether a person will enjoy a movie or song
- Identifying patterns in genetic data to detect specific proteins or diseases

Applicability

- Features related in complex ways
- Similar items homogeneous
- "You know it when you see it"

Well suited for classification tasks where relationships among the features and the target classes are **numerous**, **complicated**, or otherwise extremely **difficult to understand**, yet the items of similar class type tend to be fairly homogeneous.

E.g. a **stroke event**: unit of observation = person, example = a person with a problem, features: face, arms and speech symptomps. Note that we could also treat the "stroke prediction" with decision trees.

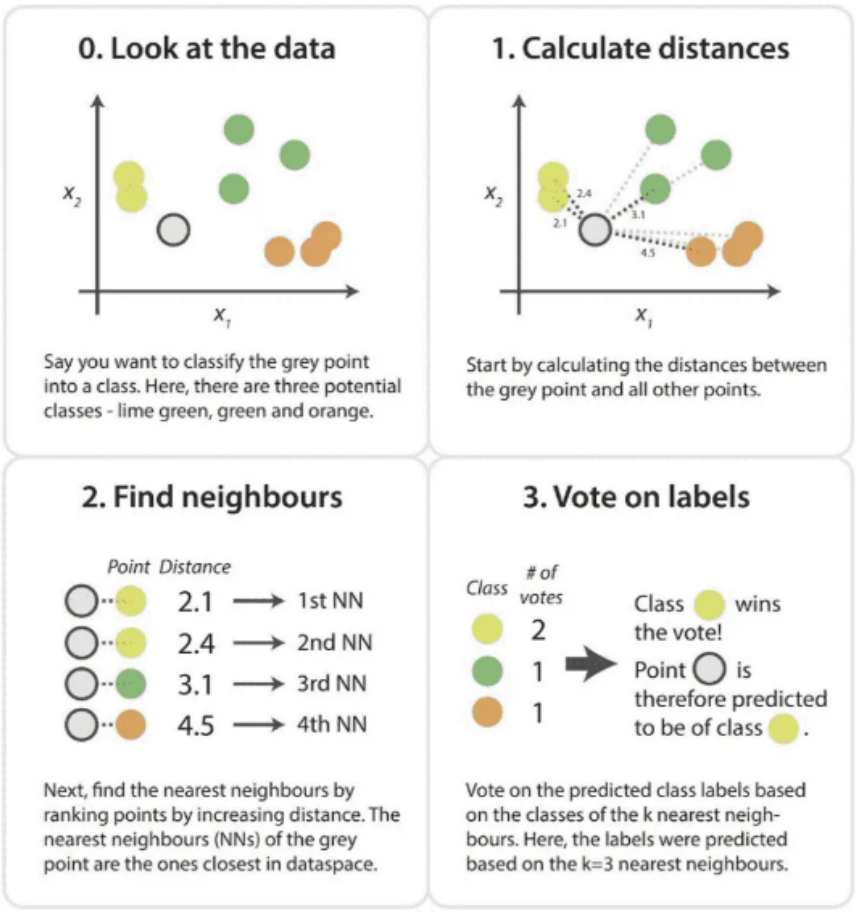
If a concept is difficult to define, but **you know it when you see it**, then nearest neighbors might be appropriate. On the other hand, if the data is **noisy** and thus no clear distinction exists among the groups, nearest neighbor algorithms may struggle to identify the class boundaries.

The k-NN algorithm

1. Pick number of nearest neighbors k
2. Use labeled training data set
3. Identify k nearest records in test data
4. Assign class to unlabeled test instance

- k-NN uses information about a sample's k nearest neigbors to classify unlabeled examples
- k is the number of nearest neighbors (could be any number)
- After choosing k, use training set labeled by nominal categorical variables (the classes or groups to choose from)
- For each unlabeled record in test set, k-NN identifies similar records
- Unlabeled test instance is assigned the majority class

Workflow



Classification with the "trained" model:

- Calculate distances to all other points (records)
- Rank points according to k
- Vote: Put test record into majority class

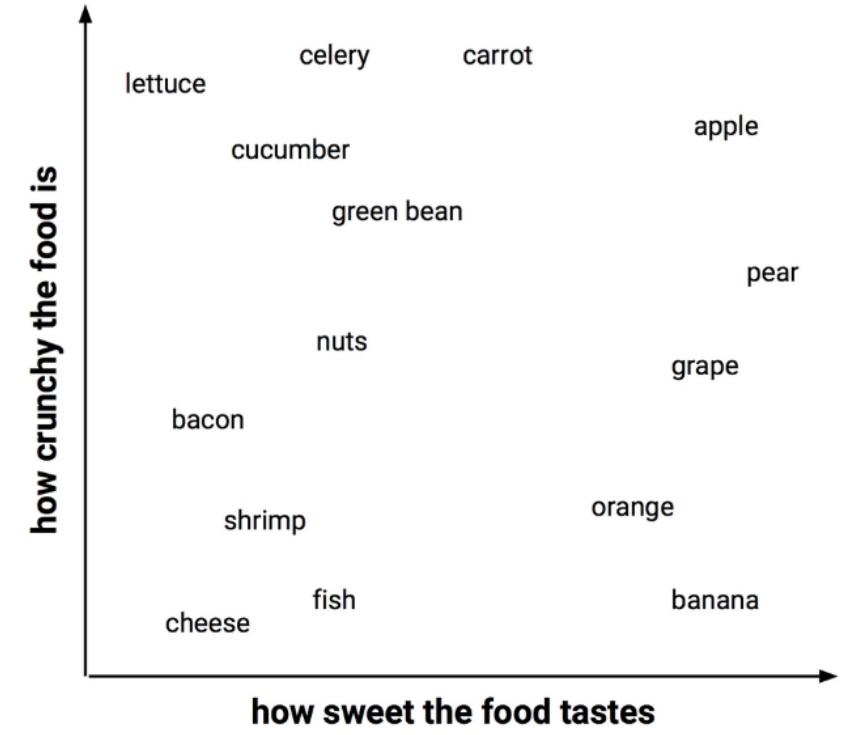
Example: blind tasting

Ingredient	Sweetness	Crunchiness	Food type
Apple	10	9	Fruit
Bacon	1	4	Protein
Banana	10	1	Fruit
Carrot	7	10	Vegetable
Celery	3	10	Vegetable
Cheese	1	1	Protein

- We want to predict/classify food that we cannot see based on similarity to other foods.
- Prior to eating we recorded previously-tasted ingredients.
- We rated two features of each ingredient from 1 to 10.
- We labeled each ingredient as one of 3 food types.

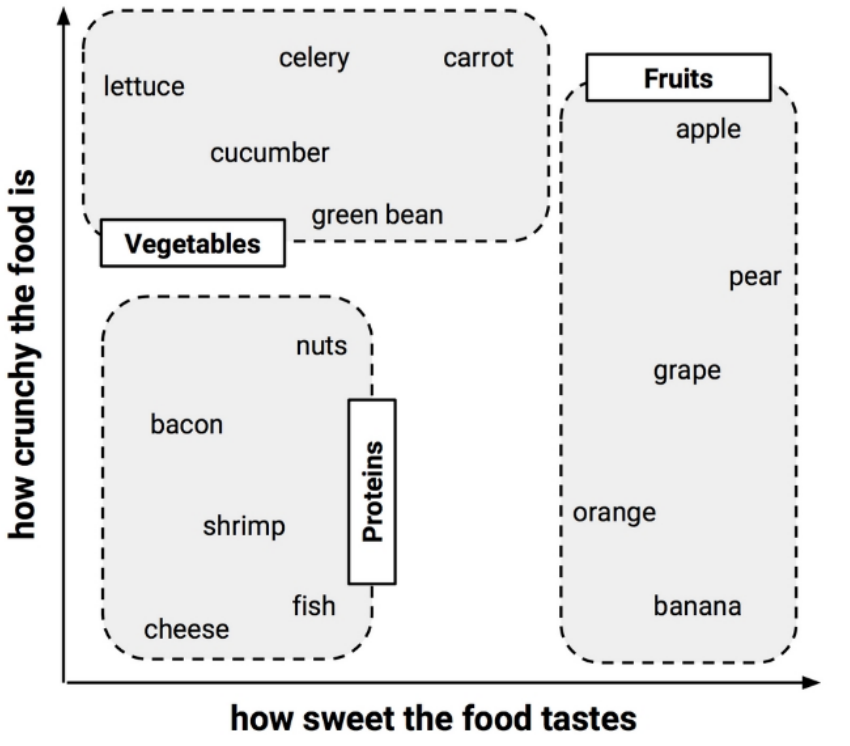
This is just like the "workflow" example shown before - instead of three color classes, we now have three food type classes, protein, fruit and vegetable.

Training: feature space



- The k-NN algorithm treats the features as coordinates in a multi-dimensional **feature space**.
- Visualized in our example: d=2 with a few more features added.
- What kind of plot is this? A **scatterplot**!
- To plot features, which are column vectors in our table, we use a **scatterplot** of crunchiness vs. sweetness
- Constructing the dataset in this way is part of the **"training"**: building a feature space of known, labeled features for classification of unlabeled, unknown features.
- Our third feature, the **food type** is the class for classification.

Training: feature patterns



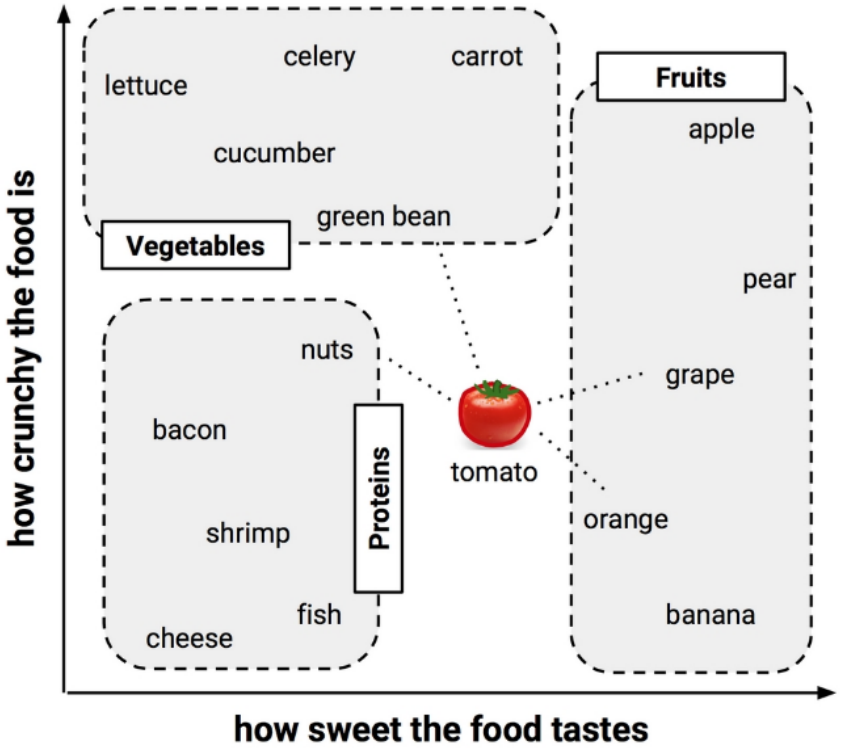
The third feature is represented here as a grouping - when plotting this in R, how would you represent it?

Answer: as a factor vector with three (nominal) levels, vegetables, fruits, and proteins. In the plot, you could color the groups according to this category (we'll plot an example later).

Similar types of food are grouped closely together:

- Vegetables are crunchy but not sweet
- Fruits are sweet and either crunchy or not crunchy
- Proteins are neither crunchy nor sweet

Testing: label examples



- Locating the tomato's nearest neighbors requires a **distance function**
- The tomato sits conveniently between our classes.
- A distance function measures the "similarity" between two instances
- Traditionally, k-NN uses **Euclidean distance**, measuring "by ruler"
- Other common measures: check out `help(dist)`, which lists 6 different types.

Euclidean distance

$$\text{dist}(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

- p, q: examples to be compared

- e.g. p=tomato, q=grape
- 1...n: example features
- e.g. 1 = sweetness, 2 = crunchiness

$$\text{dist}(\text{tomato}, \text{green bean}) = \sqrt{(6-3)^2 + (4-7)^2} = 4.2$$

- The general formula allows for n-dimensional feature vectors.
- In our food example, p corresponds to the tomato, and q to any other labeled object, e.g. a grape, a green bean etc.
- The index corresponds to the recorded features with whom the examples were labeled: here, we only have d=2, sweetness and crunchiness.
- The last formula shows the computed Euclidean distance for the tomato and the green bean examples.

Calculate distances

Ingredient	Sweetness	Crunchiness	Food type	Distance to the tomato
Grape	8	5	Fruit	sqrt((6 - 8)^2 + (4 - 5)^2) = 2.2
Green bean	3	7	Vegetable	sqrt((6 - 3)^2 + (4 - 7)^2) = 4.2
Nuts	3	6	Protein	sqrt((6 - 3)^2 + (4 - 6)^2) = 3.6
Orange	7	3	Fruit	sqrt((6 - 7)^2 + (4 - 3)^2) = 1.4

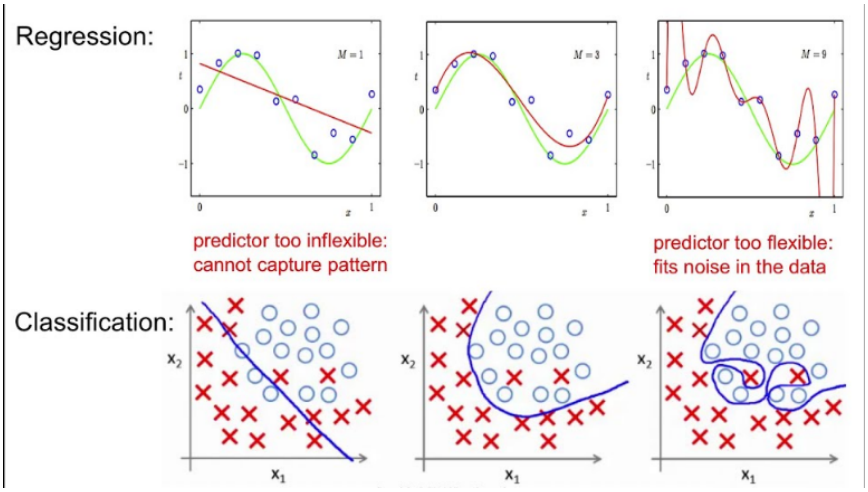
- 1-NN: "The tomato is a fruit"
- 2-NN: "The tomato is a fruit or a protein"
- 3-NN: "The tomato is a fruit"

To classify the tomato as a vegetable, protein, or fruit, we'll begin by assigning the tomato the food type of its single nearest neighbor. This is called 1-NN classification because k = 1. The **orange** is the single nearest neighbor to the tomato, with a distance of 1.4. As orange is a fruit, the 1-NN algorithm would classify a tomato as a fruit.

Using k=2 creates unclear decision boundaries. There is no winner between orange and nuts.

If we use the k-NN algorithm with k = 3 instead, it performs a vote among the three nearest neighbors: orange, grape, and nuts. Now, because the majority class among these neighbors is fruit (two of the three votes), the tomato again is classified as a **fruit**.

Underfitting vs. overfitting

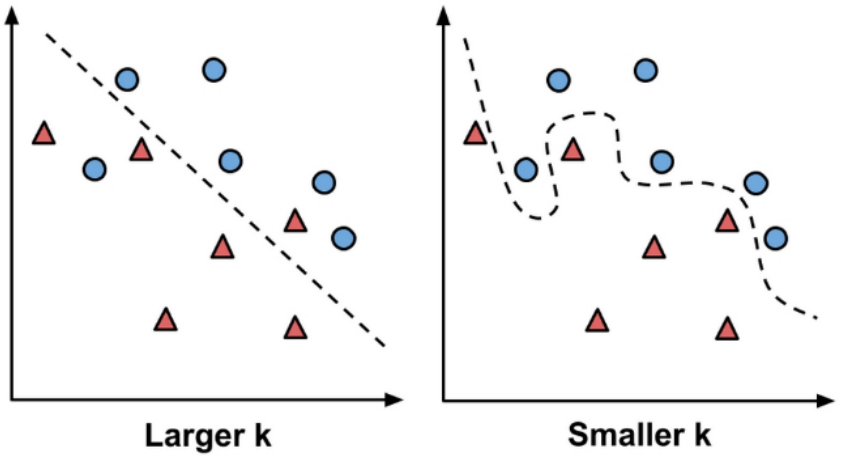


Underfitting | Perfect fit | Overfitting

In the image (by Viktor Lavrenko, 2014), the "predictor" is the set of features used for prediction of the trend line (in the regression example), and for the prediction of the two classes.

- **Underfitting:** boundary ignores points. Points do not clearly fall on either side of the line. Both training and test data have large errors.
- **Perfect fit:** the line segments the classes clearly - only a couple of noisy examples contaminate it and can be identified by distance from the line.
- **Overfitting:** noise data dominate the classification. The test data error will be large while the training data error is small.

Choice of k



- k determines performance on future data
- Danger of underfitting or overfitting
- "Bias-variance" tradeoff
- **Underfitting:** boundaries do not capture many relationships well - high error rate = large k (large neighborhood) ignores samples
- **Overfitting**: class boundaries too small, local neighborhoods dominate.
- **"Bias-variance" tradeoff**: large k reduces the impact of variance (data spread) caused by noisy data but can bias the learner so that small but important patterns are ignored.
- **Very large k**: then the majority class would always dominate the voting regardless of the nearest neighbors.
- **Very small k**: now nearest neighbors dominate even if they are just noise - e.g. **mislabeled** data - see English corpus

Data preparation

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

min-max normalization

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{Mean}(X)}{\text{StdDev}(X)}$$

z-score standardization

- The distance formula depends on the range: if you add a feature with a large scale, it will dominate the results (e.g. "spiciness" via Scoville scale - differences are much greater than sweet/crunchy).
- Rescale features to equal scales with min-max normalization (all values in 0 to 1) = how far (0 to 100%) did the original value fall along the original data range?
- z-score standardization: how many standard deviations (distance from the mean) do data fall above or below the mean? Unlike the normal distribution, the values are unbounded.
- Rescaling must be applied to training and test data (tricky - might be out of range - you may not know the maximum or minimum).

Dummy coding

$$\text{male} = \begin{cases} 1 & \text{if x} = \text{male} \\ 0 & \text{otherwise} \end{cases}$$

2-category (binary) variable (male, female)

$$\text{hot} = \begin{cases} 1 & \text{if x} = \text{hot} \\ 0 & \text{otherwise} \end{cases}$$

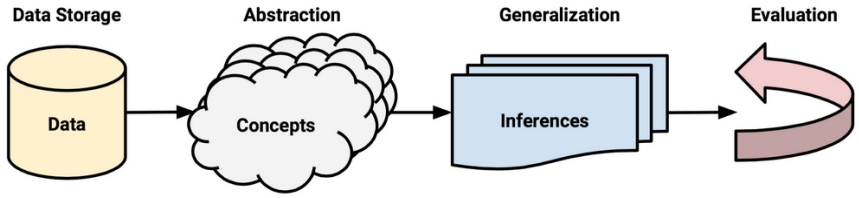
$$\text{medium} = \begin{cases} 1 & \text{if x} = \text{medium} \\ 0 & \text{otherwise} \end{cases}$$

3-category variable (hot, medium, cold)

- Euclidean distance is not defined for nominal `categorical` data ("characters") - need to convert data to `numeric` format.
- **Dummy coding** = split category set in binary values, if the values are exclusive (like `male` vs `female`).
- An n-category nominal feature can be dummy coded by creating binary indicator variables for n-1 levels of the feature.
- Example: knowing that `hot` and `medium` are both `0` means that `cold` is `1` - it it's not hot or medium, it must be cold.
- Dummy coded data always fall on 0 or 1, so there is no need for min-max normalization.

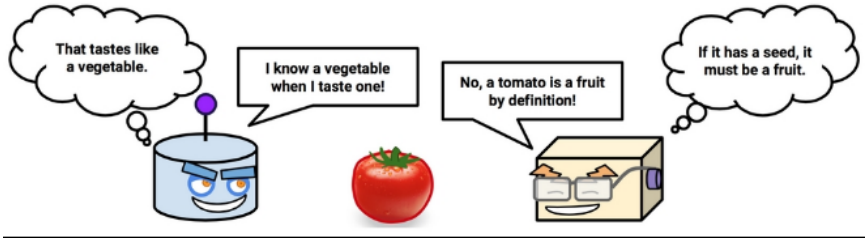
- Ordinal categorical data could be numbered: this works only if the steps between categories are equivalent, e.g. "lower", "middle", "upper" class - distance between them not equal.

Why is k-NN "lazy learning"?



- k-NN has no generalization/abstraction
- Data are stored verbatim (rapid training)
- Test/production relies on training data (slow)
- Abstraction is minimal - no detail is suppressed.
- Generalization (imposing rules) is not there at all.
- The training consists in putting the data together.
- The testing uses all training data so it can be slow.

Rote learning



- Instance-based learners build no models
- Non-parametric methods learn no parameters
- Rote learners find natural bias-free patterns
- Alternative name: instance-based learners

- No model is built, no parameters are learnt
- We do not really understand how the classifier uses the data
- Instead, everything depends on choice of k and choice of similarity measure
- Upside: the learner can find "natural" patterns rather than trying to fit the data into a biased and potentially flawed functional form.

Strengths and Weaknesses

STRENGTHS	WEAKNESSES
Simple and effective	No model
No assumptions	Selection of k
Fast training	Slow classification
Natural, no bias	Additional processing

STRENGTHS:

- Simple and effective: Does not produce a model, limiting the ability to understand how the features are related to the class
- Makes no assumptions about the underlying data distribution: bias-free, favors natural patterns
- Fast training phase since generalization (finding rules) is bypassed.

WEAKNESSES:

- No model is being built (no mathematical analysis of the model)
- Requires selection of an appropriate k - dangers of over- or underfitting
- Slow classification phase because all training data instances are used
- Nominal features and missing data require additional processing, e.g. min-max normalization or z-score standardization

Summary

- k-nearest neighbors does no learning at all
- k-NN stores training data, matches test data to most similar records in training set using a distance function
- Unlabeled example is assigned neighbor's label
- Though simple, k-NN performs well for extremely complex tasks

References

- `4_knn.jpg`: Photo by Beth Macdonald on Unsplash.
- `4_darkrestaurant.png`: Patrons at the Whale Inside Dark Restaurant.
- `4_nn_applications.jpg`: Photo by George Prentzas on Unsplash.
- `4_knn_cat_dog.png`, `4_knn_algorithm.png` - Christopher (Feb 2, 2021).
- Lantz (2019). Machine Learning with R (3e). Packt.
- Christopher (2021). K-Nearest Neighbor. URL: medium.com.