Lazy Learning

Classification Using Nearest Neighbors

Nearest neighbor classification

- Things that are alike are likely to have properties that are alike.

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

Nearest neighbor classification

- classifying unlabeled examples by assigning them the class of similar labeled examples
- extremely powerful
- well-suited for classification tasks

Nearest neighbor classification

- used successfully for
  - Computer vision applications, including optical character recognition and facial recognition in both still images and video
  - Predicting whether a person will enjoy a movie or music recommendation
  - Identifying patterns in genetic data, perhaps to use them in detecting specific proteins or diseases

Nearest neighbor classification

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

Nearest neighbor classification

- If a concept is difficult to define, but you know it when you see it, then nearest neighbors might be appropriate.

- If the data is noisy and thus no clear distinction exists among the groups, the nearest neighbor algorithms may struggle to identify the class boundaries.

k-NN algorithm

- nearest neighbors approach to classification is illustrated by the **k-nearest neighbors algorithm** (**k-NN**).

- One of the simplest machine learning algorithms, used widely.

- Strengths and weaknesses of the algorithm

k-NN algorithm

Strengths	Weaknesses
• Simple and effective • Makes no assumptions about the underlying data distribution • Fast training phase	• Does not produce a model, limiting the ability to understand how the features are related to the class • Requires selection of an appropriate *k* • Slow classification phase • Nominal features and missing data require additional processing

k-NN algorithm

- The k-NN algorithm gets its name from the fact that it uses information about an example's k-nearest neighbors to classify unlabeled examples.

- The letter k is a variable term implying that any number of nearest neighbors could be used.

k-NN algorithm

- After choosing k, the algorithm requires a training dataset made up of examples that have been classified into several categories, as labeled by a nominal variable.

- Then, for each unlabeled record in the test dataset, k-NN identifies k records in the training data that are the "nearest" in similarity.

- The unlabeled test instance is assigned the class of the majority of the k nearest neighbors.

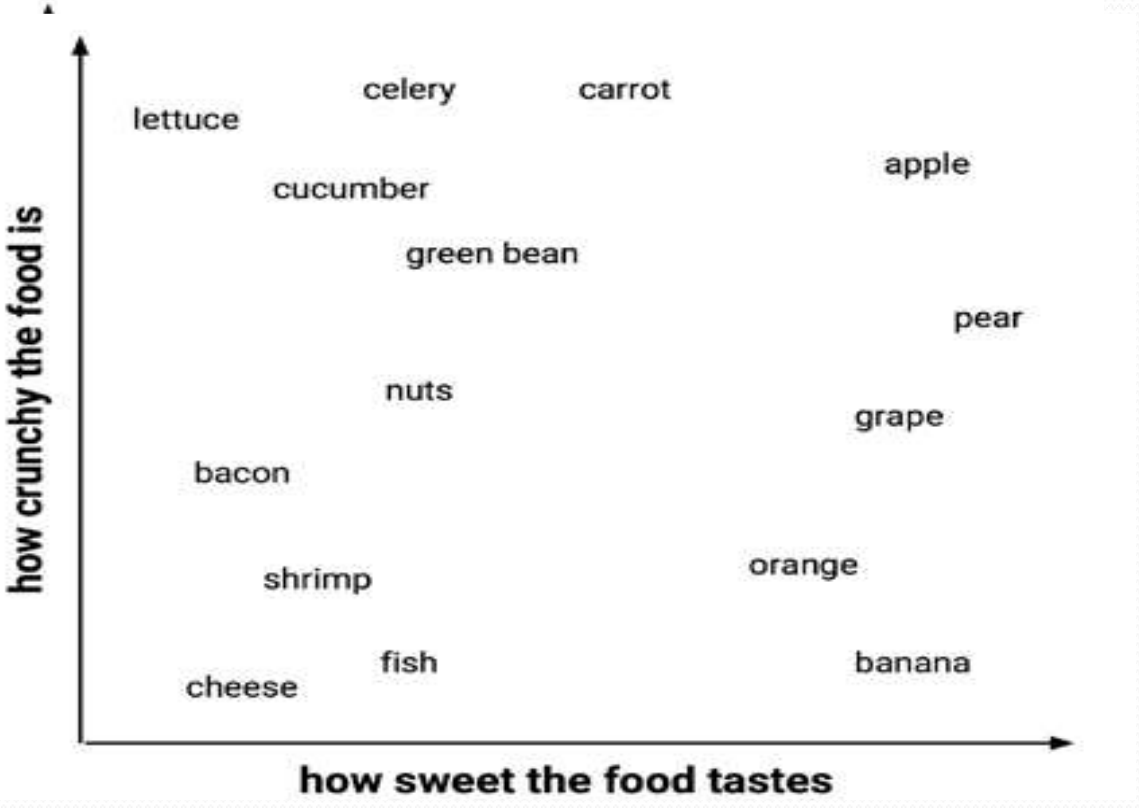
k-NN algorithm

- The k-NN algorithm treats the features as coordinates in a multidimensional feature space.
- Dataset includes only two features, the feature space is two-dimensional.

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

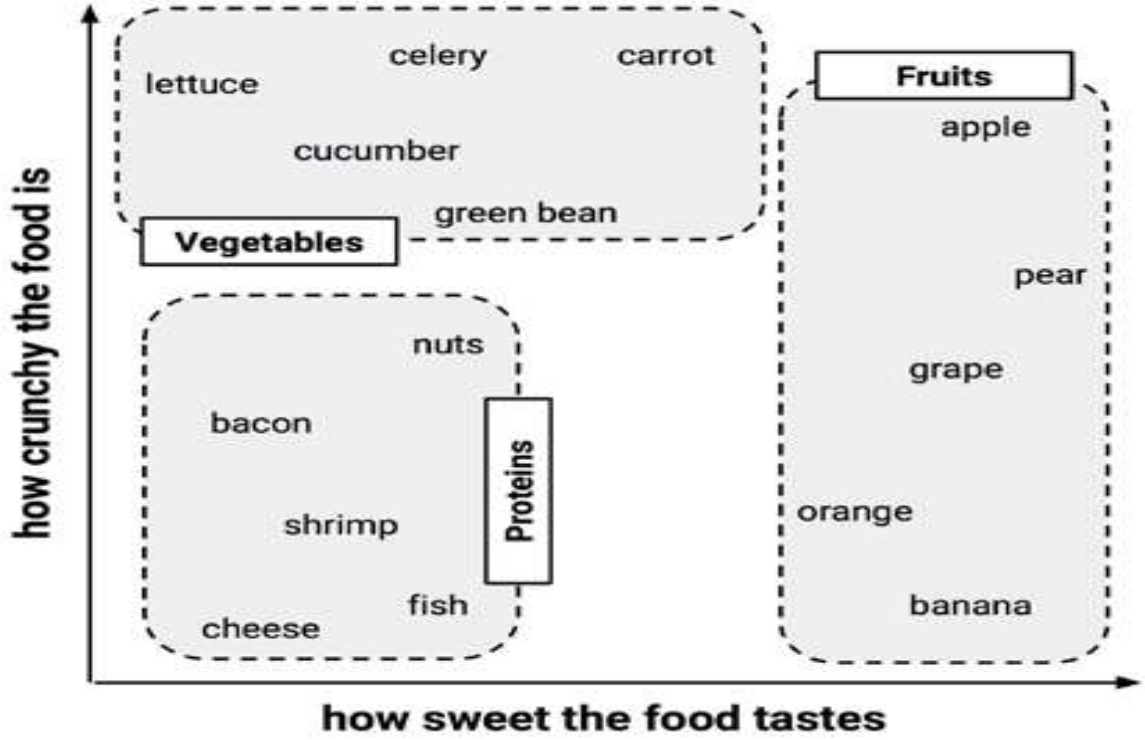
k-NN algorithm

- A scatter plot, with the *x* dimension indicating the ingredient's sweetness and the *y* dimension, the crunchiness. After adding a few more ingredients to the taste dataset, the scatter plot might look similar to this:



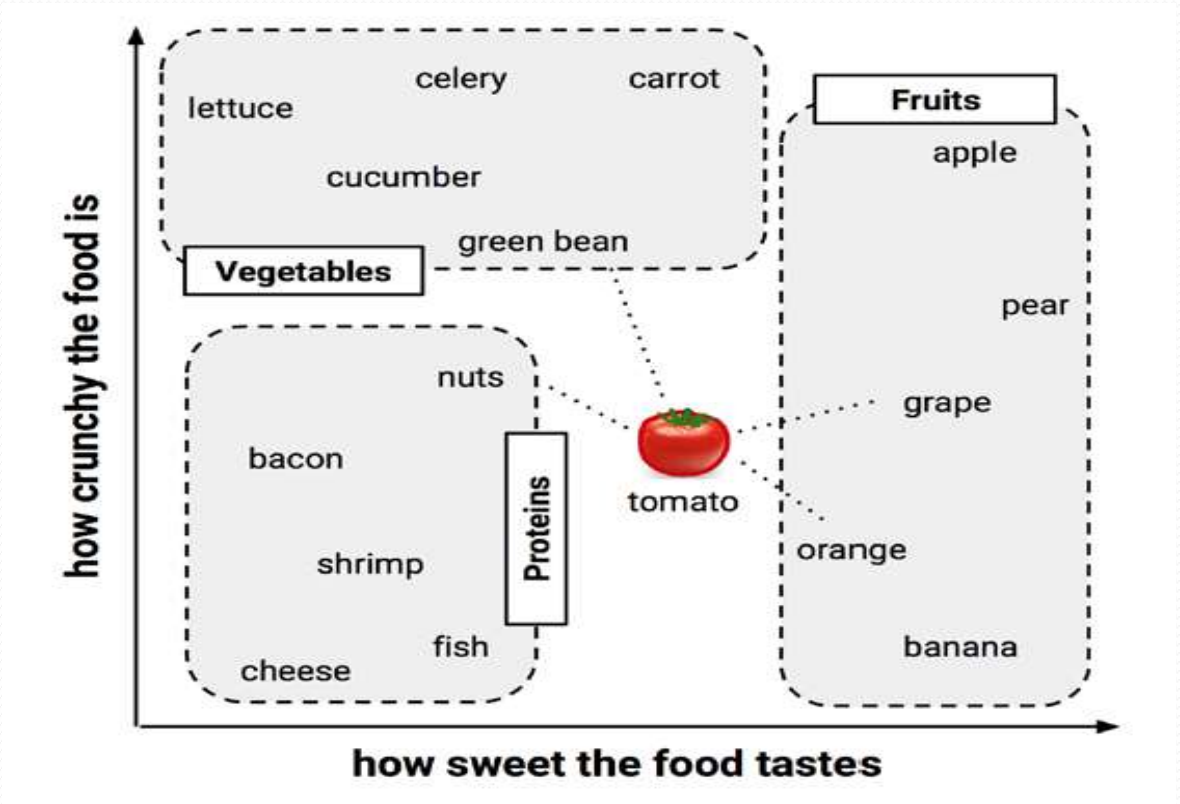
k-NN algorithm

- Similar types of food tend to be grouped closely together.
- Vegetables tend to be crunchy but not sweet, fruits tend to be sweet and either crunchy or not crunchy, while proteins tend to be neither crunchy nor sweet



k-NN algorithm

- Is tomato a fruit or vegetable?
- We can use the nearest neighbor approach to determine which class is a better fit



Measuring similarity with distance

- Locating the tomato's nearest neighbors requires a **distance function**, or a formula that measures the similarity between the two instances.
-
- There are many different ways to calculate distance.
- Traditionally, the k-NN algorithm uses **Euclidean distance**, which is the distance one would measure if it were possible to use a ruler to connect two points.

Measuring similarity with distance

- Euclidean distance formula
- $\text{dist}(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$
- where p and q are the examples to be compared, each having n features.
- The term $p1$ refers to the value of the first feature of example p,
- while $q1$ refers to the value of the first feature of example q.

Measuring similarity with distance

- The distance formula involves comparing the values of each feature.
- For example, to calculate the distance between the tomato (*sweetness* = 6, *crunchiness* = 4), and the green bean (*sweetness* = 3, *crunchiness* = 7), we can use the formula as follows:

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

Measuring similarity with distance

Similarly, we can calculate the distance between the tomato and several of its closest neighbors as follows:

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	$\sqrt{(6-7)^2+(4-3)^2} = 1.4$

Measuring similarity with distance

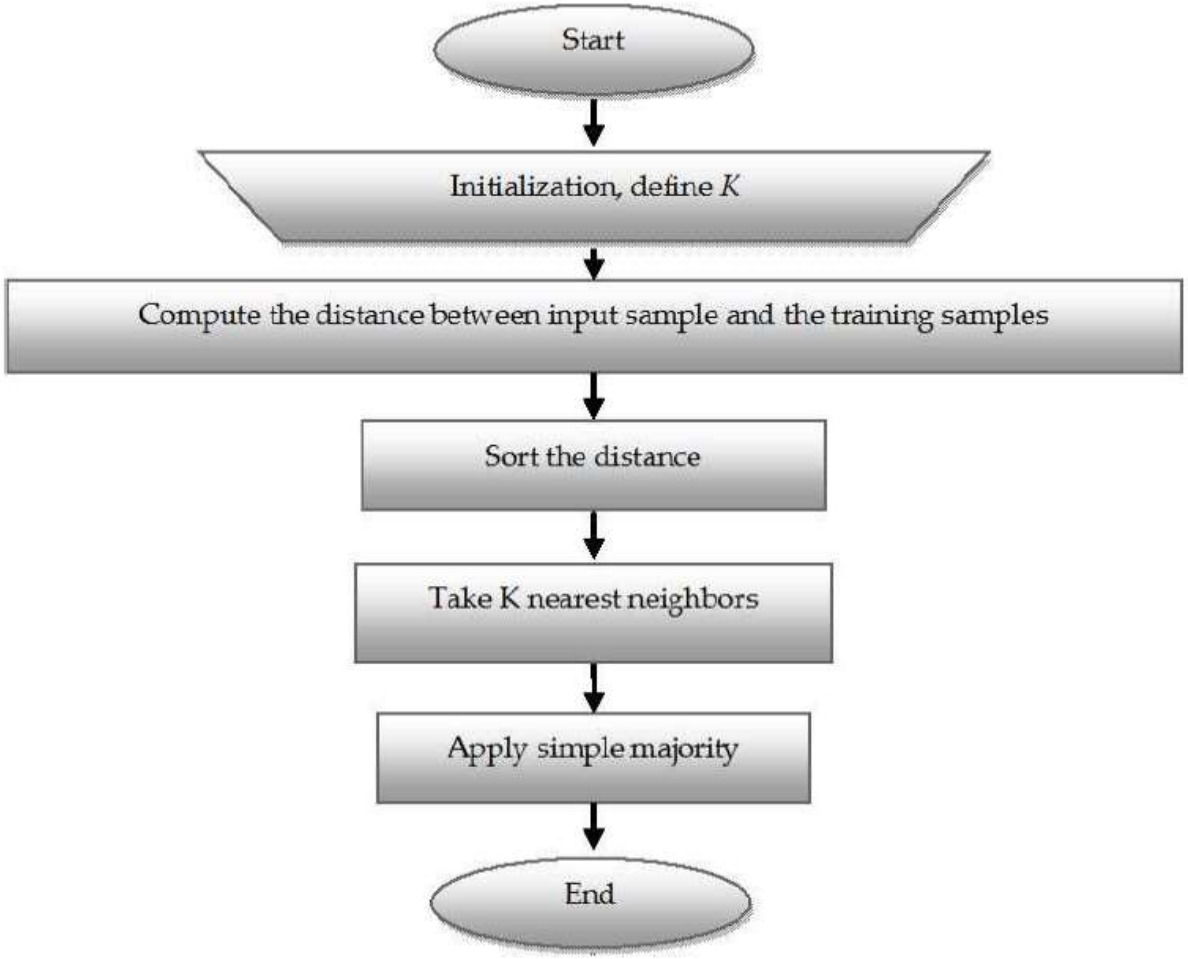
- To classify the tomato as a vegetable, protein, or fruit, we'll begin by assigning the tomato, the food type of its single nearest neighbor.
- This is called 1-NN classification because $k = 1$. The orange is the nearest neighbor to the tomato, with a distance of 1.4.
- As orange is a fruit, the 1-NN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

Measuring similarity with distance

- If we use the k-NN algorithm with $k = 3$ instead, it performs a vote among the three nearest neighbors: orange, grape, and nuts.

- Since the majority class among these neighbors is fruit (two of the three votes), the tomato again is classified as a fruit.

KNN Classifier Algorithm

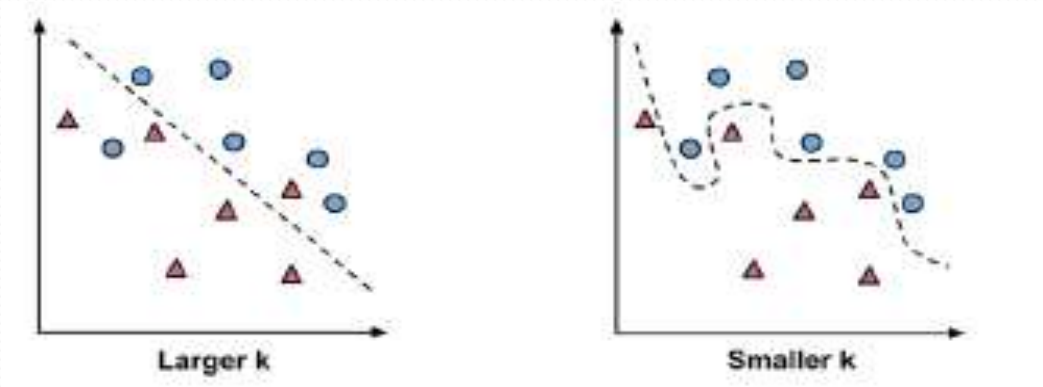


Choosing an appropriate k

- How many neighbors to use for k-NN determines how well the model will generalize to future data.
- The balance between overfitting and underfitting the training data is a problem known as **bias-variance tradeoff**.
- Choosing a large *k* reduces the impact or variance caused by noisy data, but can bias the learner so that it runs the risk of ignoring small, but important patterns.

Choosing an appropriate k

- Obviously, the best *k* value is somewhere between these two extremes.



Choosing an appropriate k

- Choosing *k* depends on the difficulty of the concept to be learned, and the number of records in the training data.
- One common practice is to begin with *k* equal to the square root of the number of training examples.
- If there are 15 example ingredients in the training data and the square root of 15 is 3.87. we might set k=4.

Choosing an appropriate k

- However, such rules may not always result in the single best *k*.
- An alternative approach is to test several *k* values on a variety of test datasets and choose the one that delivers the best classification performance.
- That said, unless the data is very noisy, a large training dataset can make the choice of *k* less important.

Choosing an appropriate k

- A less common, but interesting solution to this problem is to choose a larger *k*,
- but apply a **weighted voting** process in which the vote of the closer neighbors is considered more authoritative than the vote of the far away neighbors.
- Many k-NN implementations offer this option.

Preparing data for use with k-NN

- Features are typically transformed to a standard range prior to applying the k-NN algorithm
- Since distance formula is highly dependent on how features are measured
- if certain features have a much larger range of values than the others, the distance measurements will be strongly dominated by the features with larger ranges

Preparing data for use with k-NN

- The solution is to rescale the features by shrinking or expanding their range such that each one contributes relatively equally to the distance formula.
- The traditional method of rescaling features for k-NN is **min-max normalization**
- This process transforms a feature such that all of its values fall in a range between 0 and 1.

Preparing data for use with k-NN

- **min-max normalization formula**

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

Preparing data for use with k-NN

- Another common transformation is called **z-score standardization**. The following formula subtracts the mean value of feature X, and divides the outcome by the standard deviation of X:
- The z-scores fall in an unbound range of negative and positive numbers.

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

Preparing data for use with k-NN

- The Euclidean distance formula is not defined for nominal data.
- Therefore, to calculate the distance between nominal features, we need to convert them into a numeric format.
- A typical solution utilizes **dummy coding**, where a value of *1* indicates one category, and *0*, the other.

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

Why is the k-NN algorithm lazy?

- Under the strict definition of learning, a lazy learner is not really learning anything.
- Abstraction and generalization processes are skipped
- It merely stores the training data.
- This allows the training phase, which is not actually training anything, to occur very rapidly.
- Due to the heavy reliance on the training instances rather than an abstracted model, lazy learning is also known as **instance-based learning** or **rote learning**.

Why is the k-NN algorithm lazy?

- As instance-based learners do not build a model, the method is said to be in a class of **non-parametric** learning methods—no parameters are learned about the data.
- Without generating theories about the underlying data, non-parametric methods limit our ability to understand how the classifier is using the data.