K NEAREST NEIGHBOR ALGORITHM

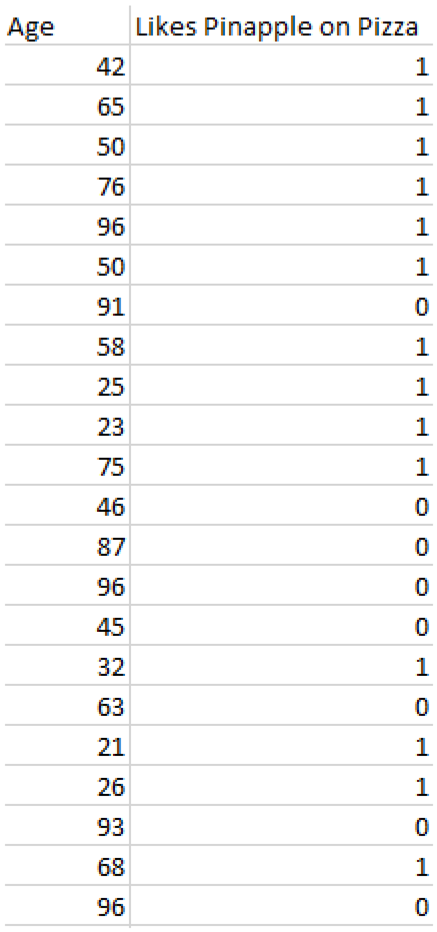
Machine Learning Algorithms are generally of two types:

1. Supervised
2. Unsupervised

A **supervised machine learning** algorithm is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

Supervised machine learning algorithms are used to solve classification or regression problems.

A **classification problem** has a discrete value as its output. For example, “likes pineapple” and “does not like pineapple” are discrete. There is no middle ground. The analogy above of teaching a child to identify a pig is another example of a classification problem.

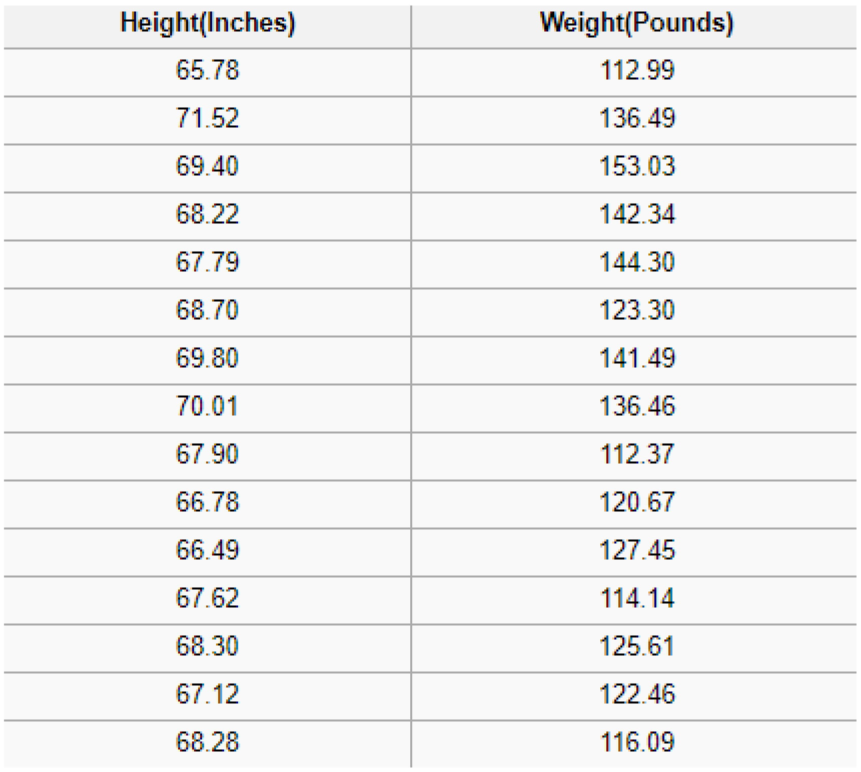


Age	Likes Pinapple on Pizza
42	1
65	1
50	1
76	1
96	1
50	1
91	0
58	1
25	1
23	1
75	1
46	0
87	0
96	0
45	0
32	1
63	0
21	1
26	1
93	0
68	1
96	0

This image shows a basic example of what classification data might look like. We have a predictor (or set of predictors) and a label. In the image, we might be trying to predict whether someone likes pineapple (1) on their pizza or not (0) based on their age (the predictor).

It is standard practice to represent the output (label) of a classification algorithm as an integer number such as 1, -1, or 0.

A **regression problem** has a real number (a number with a decimal point) as its output. For example, we could use the data in the table below to estimate someone's weight given their height.



Height(Inches)	Weight(Pounds)
65.78	112.99
71.52	136.49
69.40	153.03
68.22	142.34
67.79	144.30
68.70	123.30
69.80	141.49
70.01	136.46
67.90	112.37
66.78	120.67
66.49	127.45
67.62	114.14
68.30	125.61
67.12	122.46
68.28	116.09

Data used in a regression analysis will look similar to the data shown in the image above. We have an independent variable (or set of independent variables) and a dependent variable (the thing we are trying to guess given our independent variables). For instance, we could say height is the independent variable and weight is the dependent variable.

Also, each row is typically called an **example, observation, or data point**, while each column (not including the label/dependent variable) is often called a **predictor, dimension, independent variable, or feature.**

An **unsupervised machine learning** algorithm makes use of input data without any labels —in other words, no teacher (label) telling the child (computer) when it is right or when it has made a mistake so that it can self-correct.

Unlike supervised learning that tries to learn a function that will allow us to make predictions given some new unlabeled data, unsupervised learning tries to learn the basic structure of the data to give us more insight into the data.

KNN Algorithm

It’s a supervised algorithm.

Basic Principle: Birds of same feather flock together.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

The KNN Algorithm

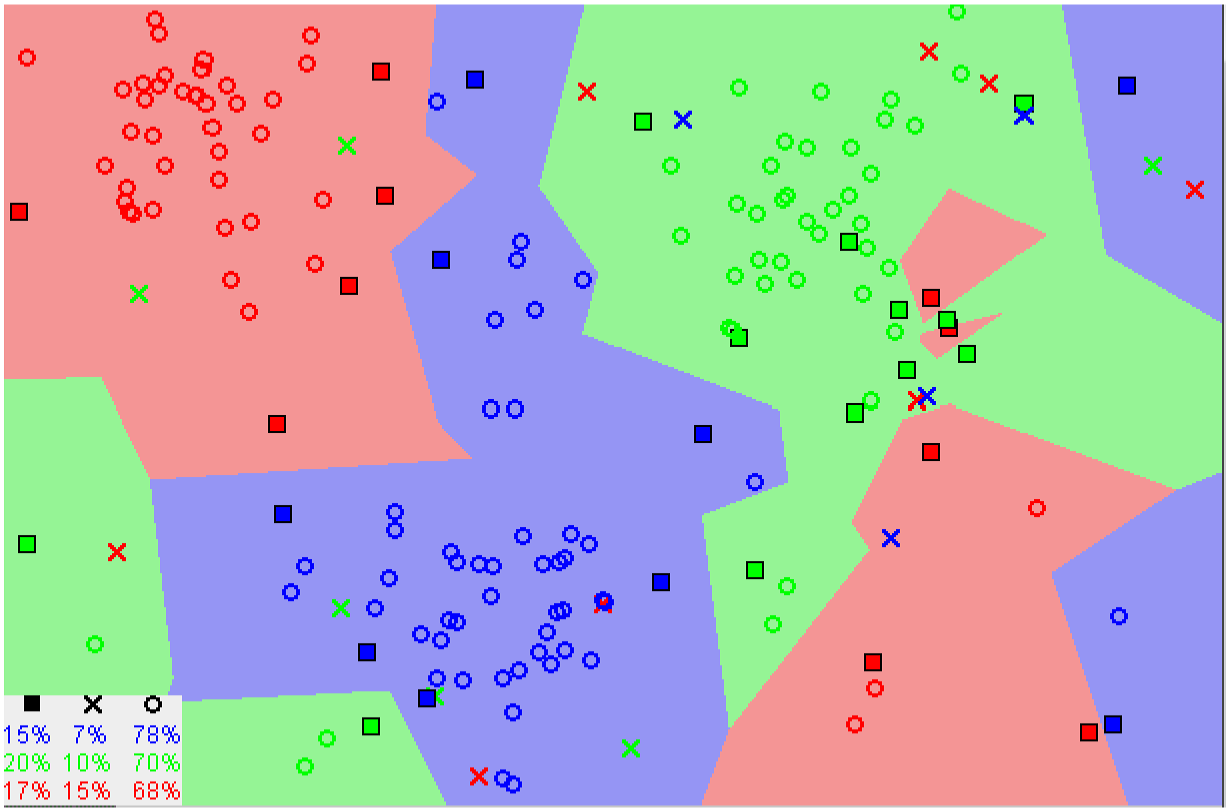
1. Load the data
2. Scale the features:

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

3. Initialize K to your chosen number of neighbors

4. For each example in the data

   a. Calculate the distance between the query example and the current example from the data.
   b. Add the distance and the index of the example to an ordered collection
3. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
4. Pick the first K entries from the sorted collection
5. Get the labels of the selected K entries
6. If regression, return the mean of the K labels
7. If classification, return the mode of the K labels



Choosing the right value for K

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Here are some things to keep in mind:

1. As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, imagine K=1 and we have a query point surrounded by several reds and one green (I’m thinking about the top left corner of the colored plot above), but the green is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.

2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.

3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Advantages

1. The algorithm is simple and easy to implement.

2. There’s no need to build a model, tune several parameters, or make additional assumptions.

3. The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).

Disadvantages

1. The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

Offline

Problem Statement: Given a movie name (input), return a list of 5 similar movie names with corresponding distances.

Coding Steps for KNN Problems:

1. **Process Data**: Open the dataset from CSV, load the data into a dataframe, normalize the features as instructed before and split the dataset into train/test (80-20) datasets.
2. **Calculate Similarity**: For each query movie (which will be provided by input), calculate the distance between given query movie and every other movie in the train dataset (80% of the whole dataset which was split earlier).
3. **Output**: Return 5 nearest neighbors or 5 most similar movies' names of the given input (movie name) along with their respective distances.

   Example:

   Input:
   ABC_movie

   Output:
   Most_similar_movie_of_ABC_movie_1 with distance 1.231
   Most_similar_movie_of_ABC_movie_2 with distance 1.232
   Most_similar_movie_of_ABC_movie_3 with distance 1.233
   Most_similar_movie_of_ABC_movie_4 with distance 1.234
   Most_similar_movie_of_ABC_movie_5 with distance 1.235

   [N.B: Try to make the code as modular as possible so that you can use this code for a new dataset of different format which will be tested through Online. No late submission will be allowed. Don't copy!]