K-Nearest Neighbors

Non-parametric **instance-based** learning

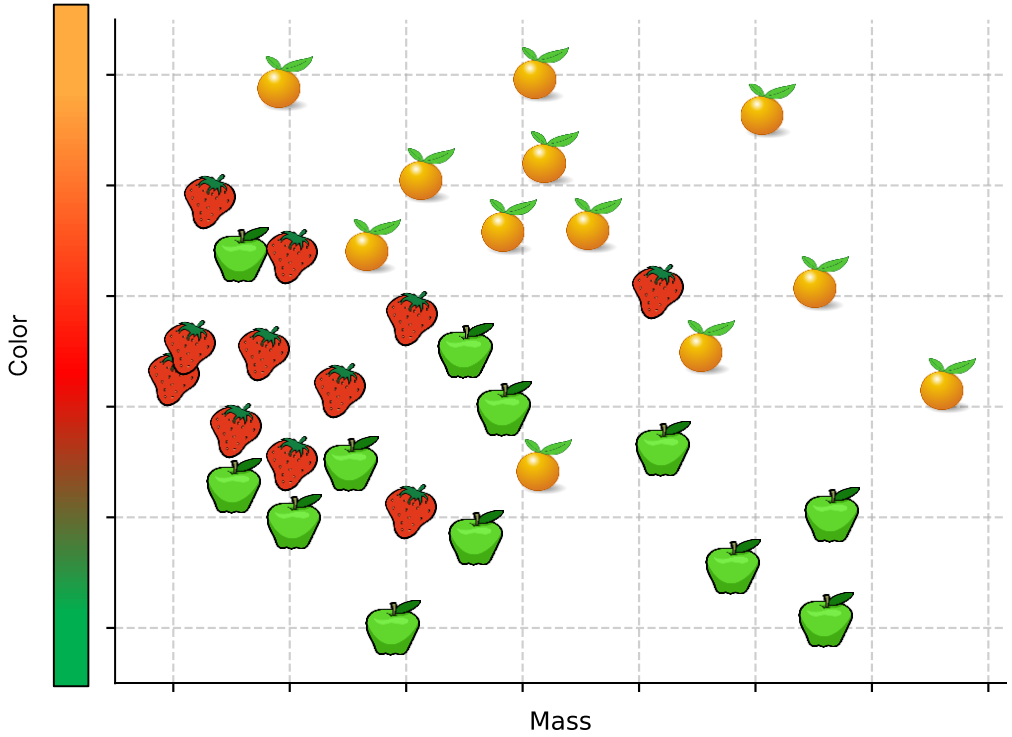
Faculty of Mathematics and Computer Science, University of Bucharest
and
Sparktech Software

Academic Year 2018/2019, 1st Semester

Definition

- **K-Nearest Neighbors (kNN)** is a *non-parametric*, *instance-based*, supervised learning algorithm.
  - **Non-parametric** means that it has *no finite set of parameters* which model the underlying function or distribution of data (like parameters $\vec{w}$ in Linear Regression).
  - **Instance-based** means that the algorithm doesn't explicitly learn a model.
    - Instead, it memorizes the training data and uses it directly when making predictions (i.e. There is no training phase, *the training data is the model*).
    - Also known as *lazy learning*.
- **kNN** predicts the label of a new instance as the *most common (or average) label* of the *k* **closest** training samples in feature space.

How it works?

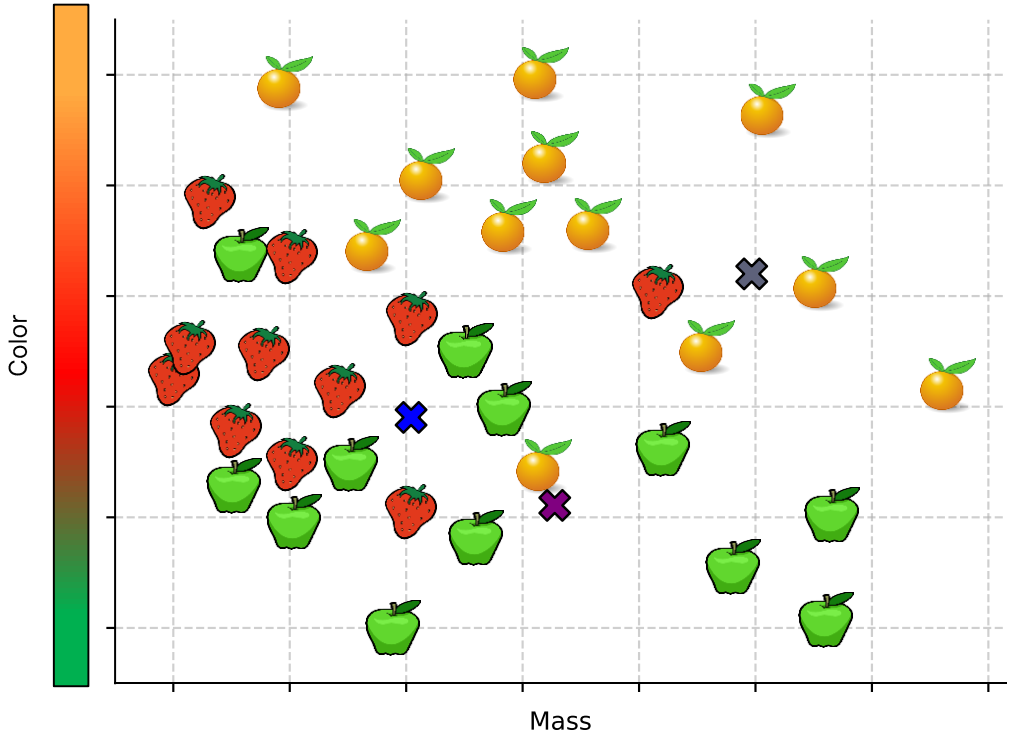


How it works?

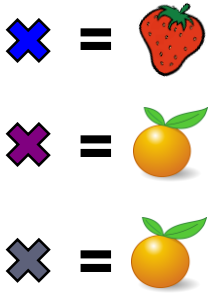
✖ = ?

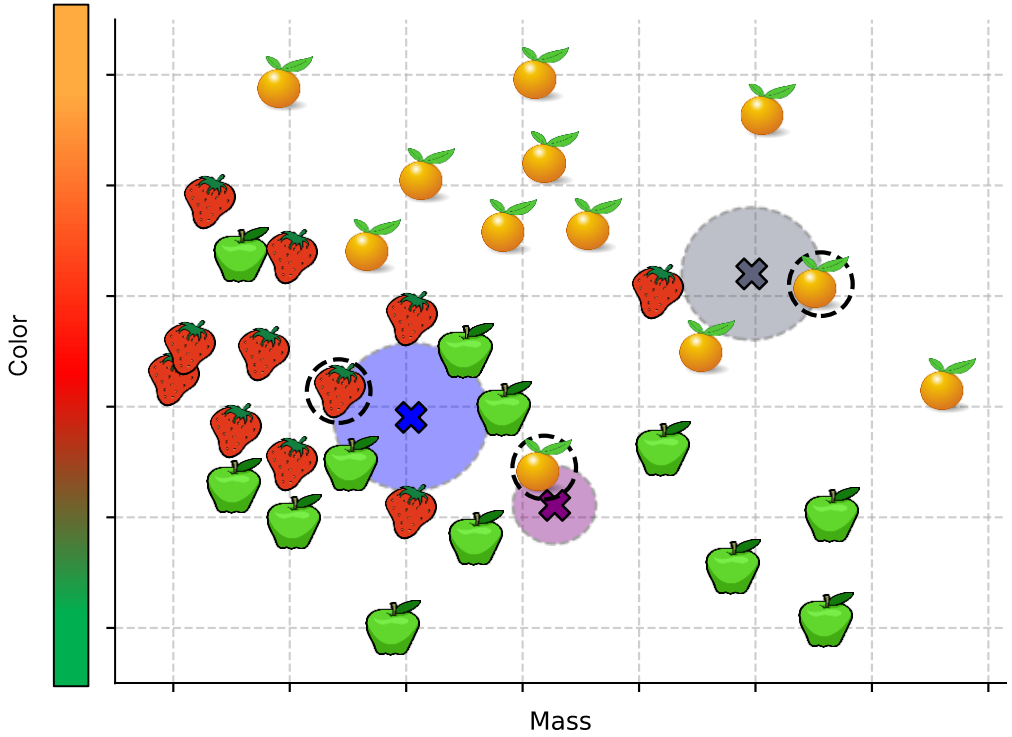
✖ = ?

✖ = ?



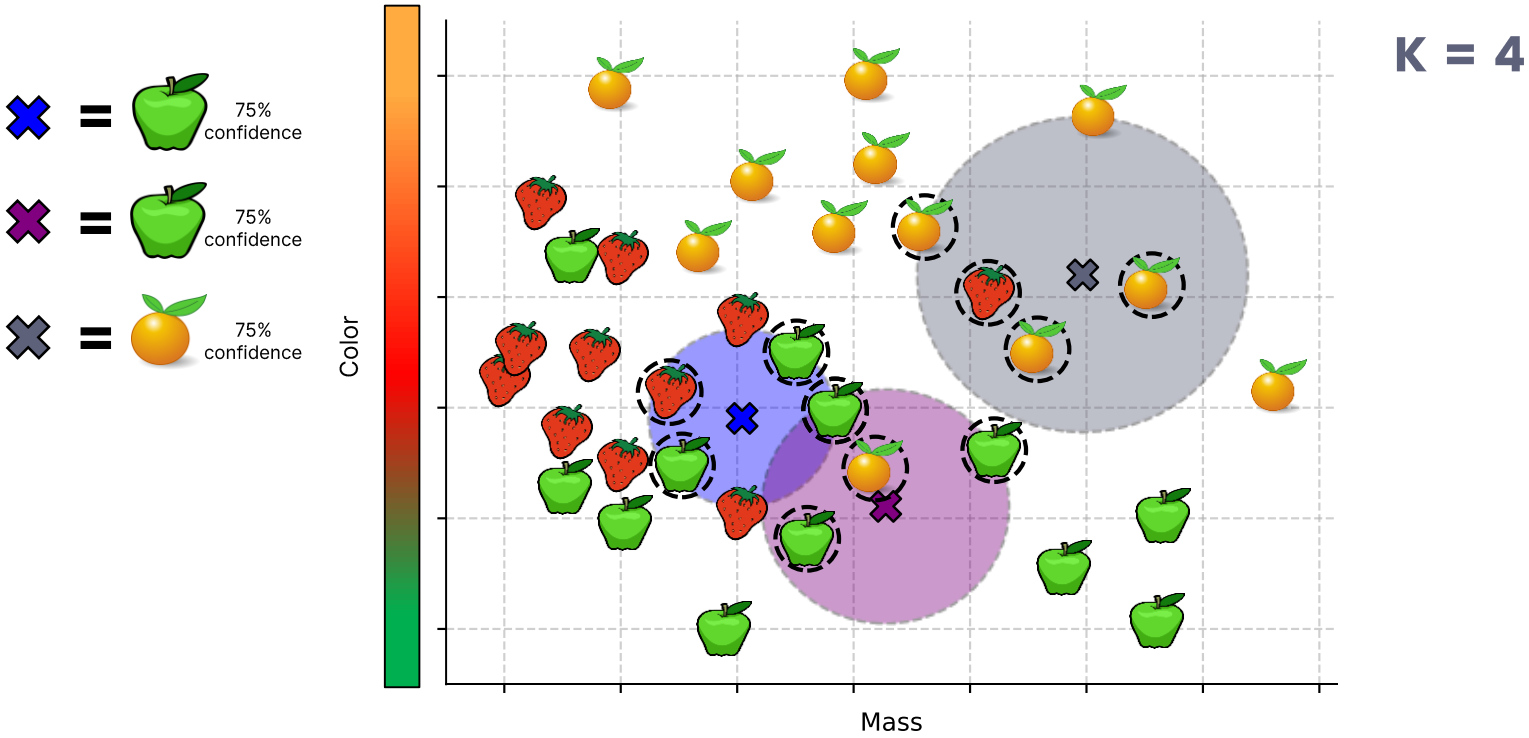
How it works?



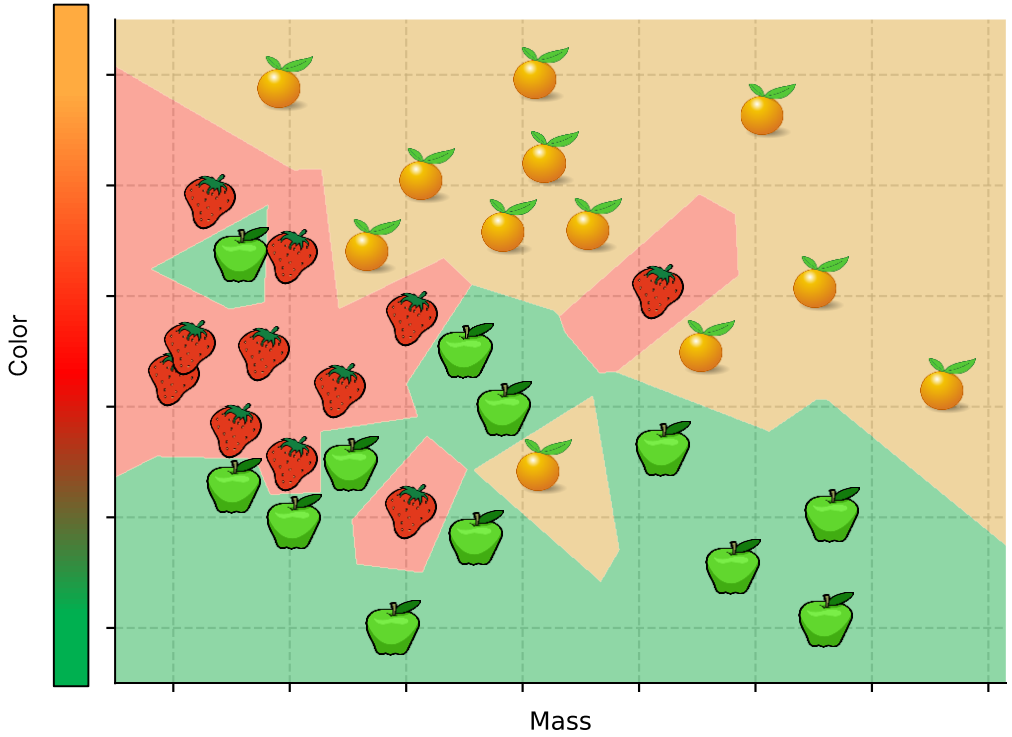


K = 1

How it works?



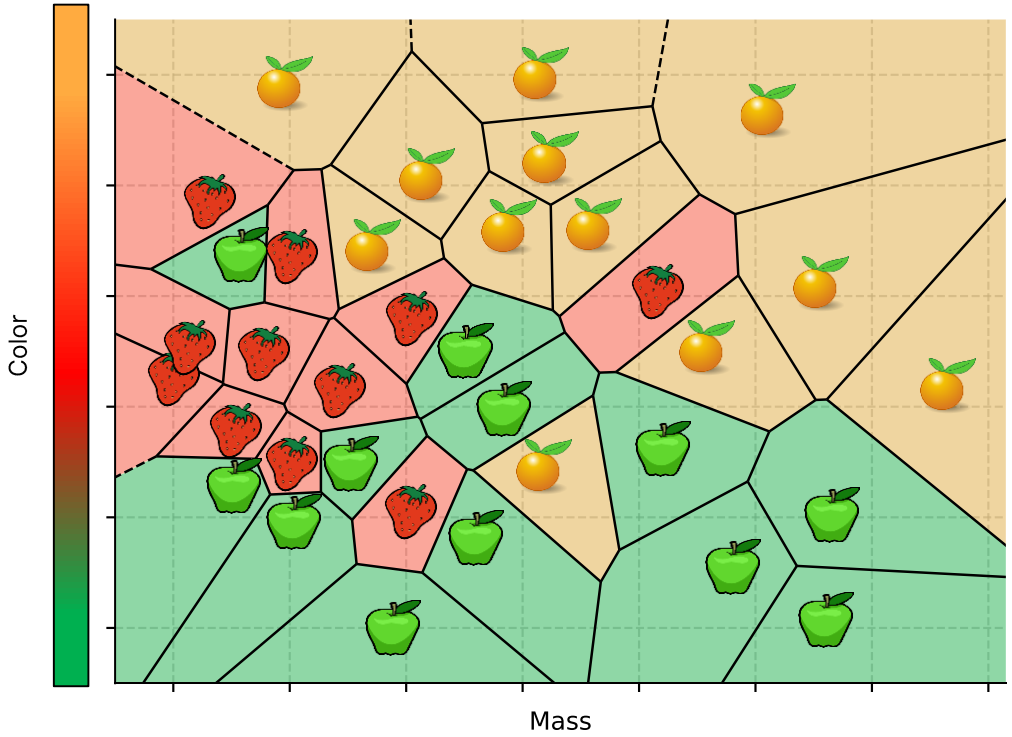
Decision Boundary



K = 1

Decision Boundary

Voronoi Cells



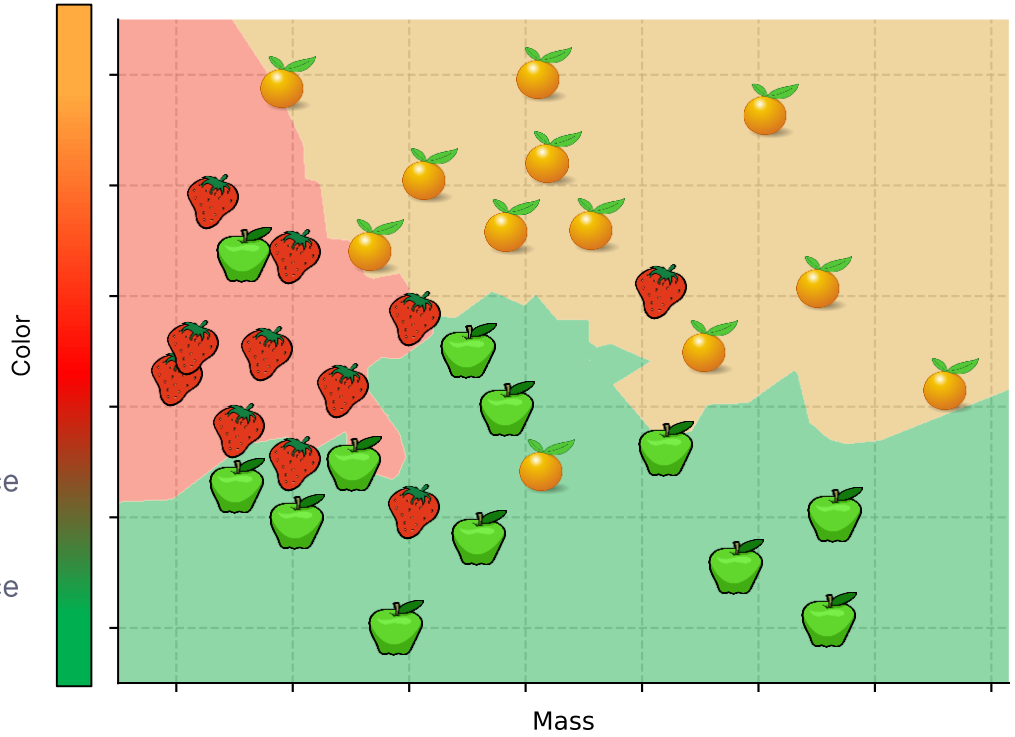
K = 1

Decision Boundary

Increasing *k* makes the decision boundary *smoother*, making the model *simpler*.

Small *k* ⟹
Low Bias, High Variance

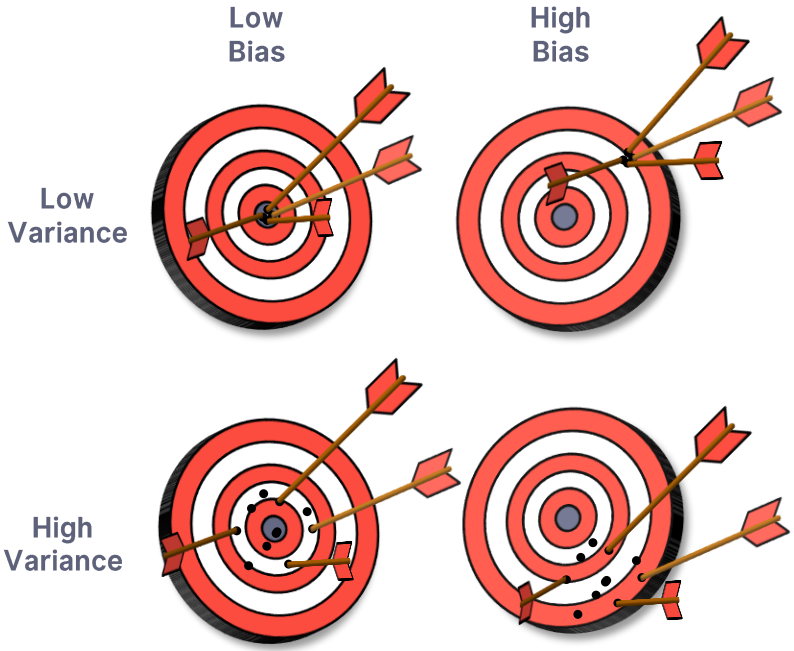
Large *k* ⟹
High Bias, Low Variance



K = 4

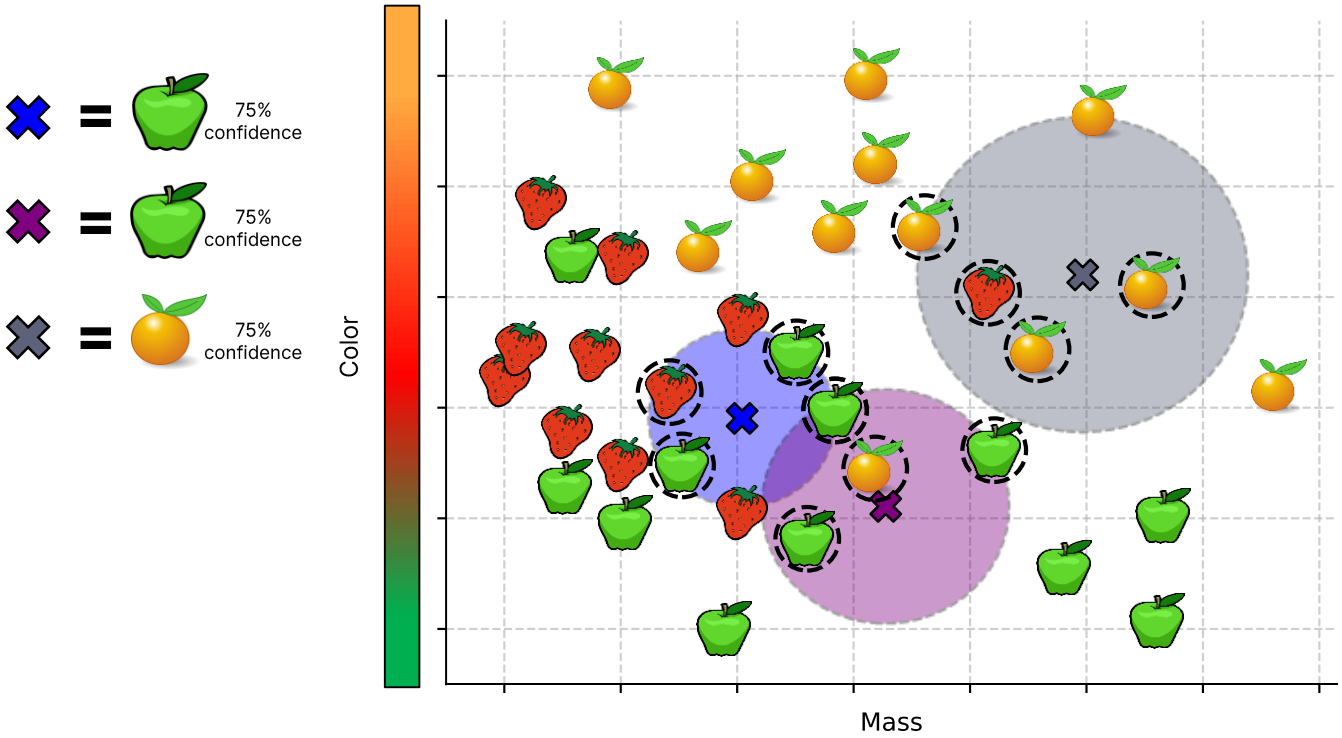
Bias vs. Variance

- There are two sources of error in ML models.
- **Bias** (or **systematic error**) comes from the inability of the algorithm to model the true relationship between features and label (*underfitting*).
  - Bias error cannot be corrected by adding more training samples, but by increasing the complexity of the model.
- **Variance** (or **random error**) comes from sensitivity to small fluctuations in the training data, causing the algorithm to model random noise. (*overfitting*)
  - Variance error can be corrected by adding more training samples or by decreasing the complexity of the model.
- There is usually a *tradeoff* between bias and variance.



Distance Metrics

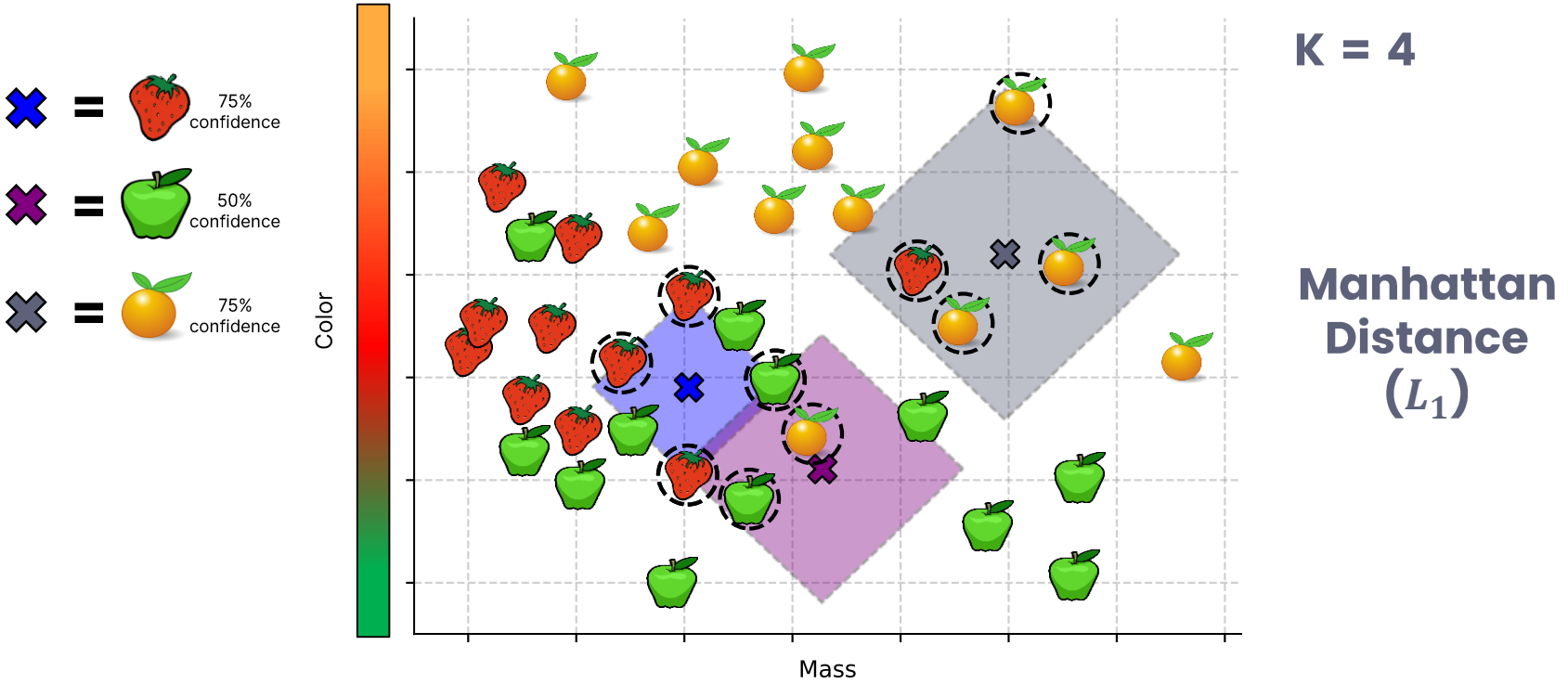
Distance metric matters



K = 4

Euclidean Distance (L_2)

Distance metric matters



K = 4

Manhattan Distance (L_1)

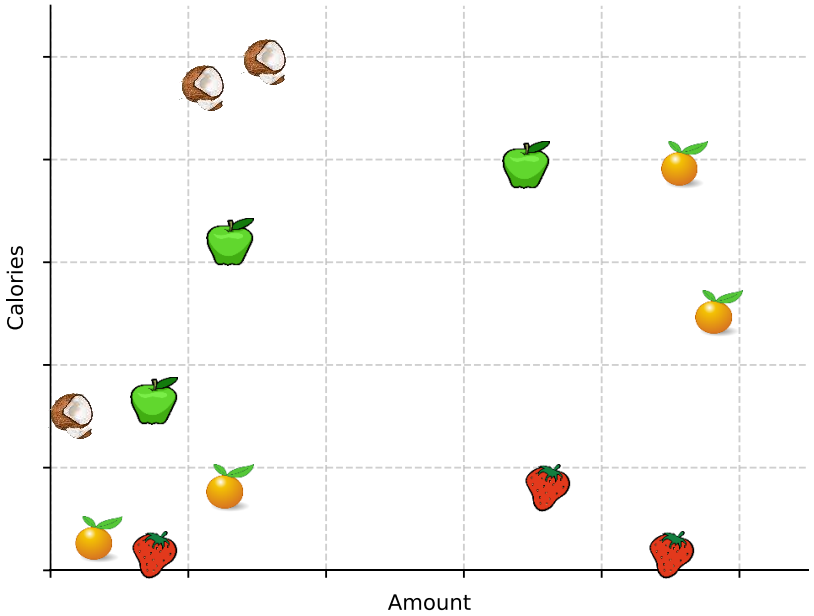
Distance metric matters

- A **distance metric** on a set X is a function $d: X \times X \to [0, \infty)$, which satisfies the following conditions $\forall x, y \in X$:
  - $d(x, y) \geq 0$ *non-negativity*
  - $d(x, y) = 0 \Leftrightarrow x = y$ *identity of indiscernible*
  - $d(x, y) = d(y, x)$ *symmetry*
  - $d(x, z) \leq d(x, y) + d(y, z)$ *triangle inequality*

- The performance of kNN depends heavily on the choice of distance metric.
  - It must reflect the nature of the problem (i.e. it should be small for items which are known to be similar and large for items which are known to be different).

Distance metric matters

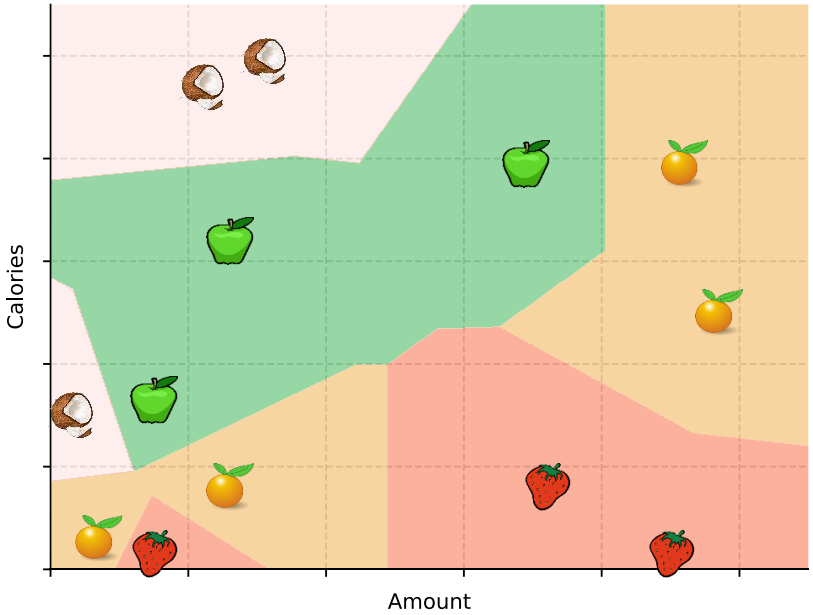
- Task: Predict the *type of fruit* based of a set of samples for which we know the *amount* and *calories* contained in it.



Distance metric matters

- Task: Predict the *type of fruit* based of a set of samples for which we know the *amount* and *calories* contained in it.

Euclidean Distance

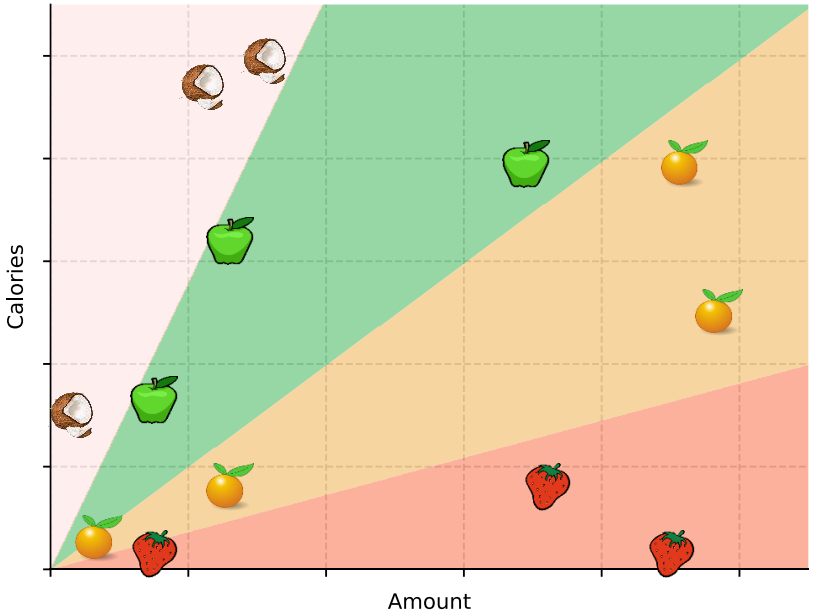


Does this metric really capture the relation between the features and the label?

Distance metric matters

- Task: Predict the *type of fruit* based of a set of samples for which we know the *amount* and *calories* contained in it.

Cosine Distance



Cosine distance (or similarity) measures the *angle* between feature vectors.

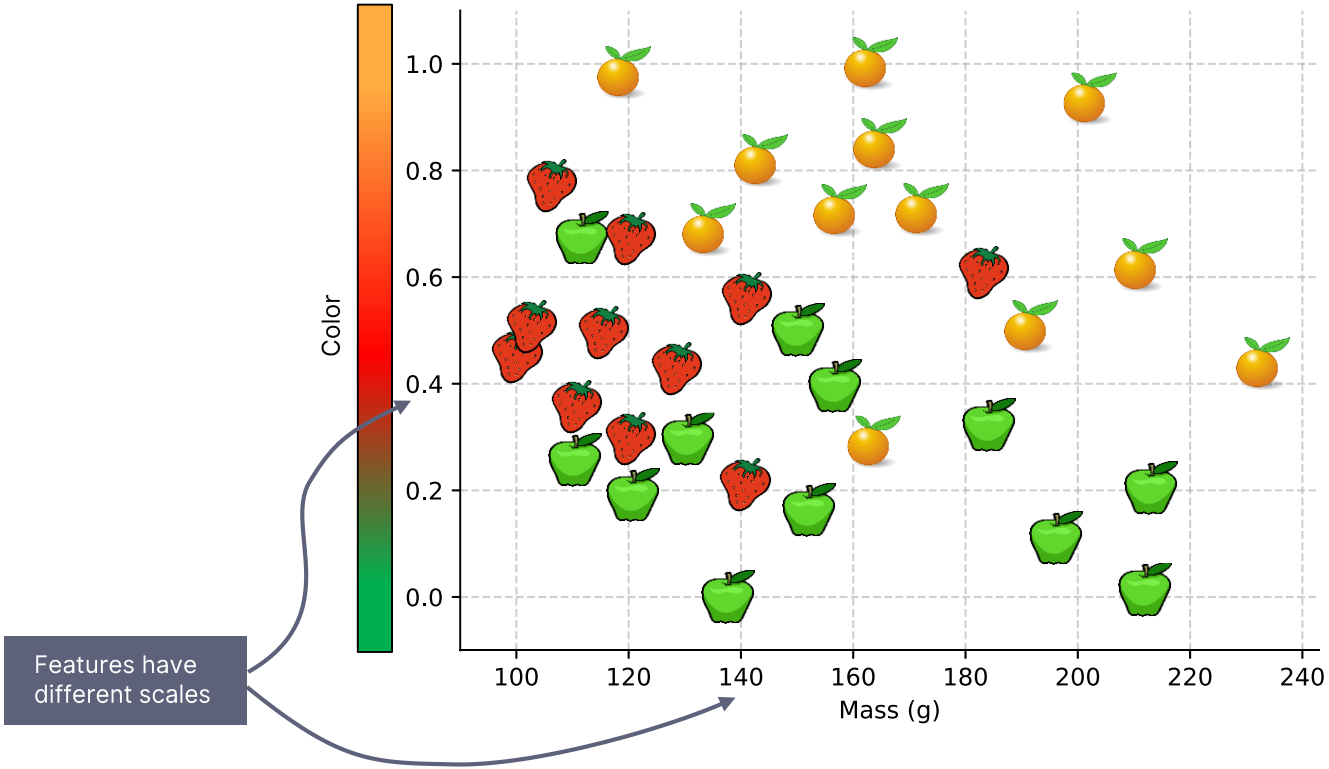
Similar items have *similar ratios* between feature values (rather than similar absolute values)

Popular distance metrics

Euclidean distance:	$\mathrm{d}(\vec{x},\vec{y}) = \sqrt{\sum_i (x_i - y_i)^2}$	L_2
Manhattan distance:	$\mathrm{d}(\vec{x},\vec{y}) = \sum_i \lvert x_i - y_i \rvert$	L_1
Chebyshev distance:	$\mathrm{d}(\vec{x},\vec{y}) = \max_i(\lvert x_i - y_i \rvert)$	L_∞
Minkowski distance:	$\mathrm{d}(\vec{x},\vec{y}) = \left(\sum_i \lvert x_i - y_i \rvert^p\right)^{1/p}$	L_p
Cosine distance:	$d(\vec{x},\vec{y}) = 1 - \cos\theta = 1 - \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$	

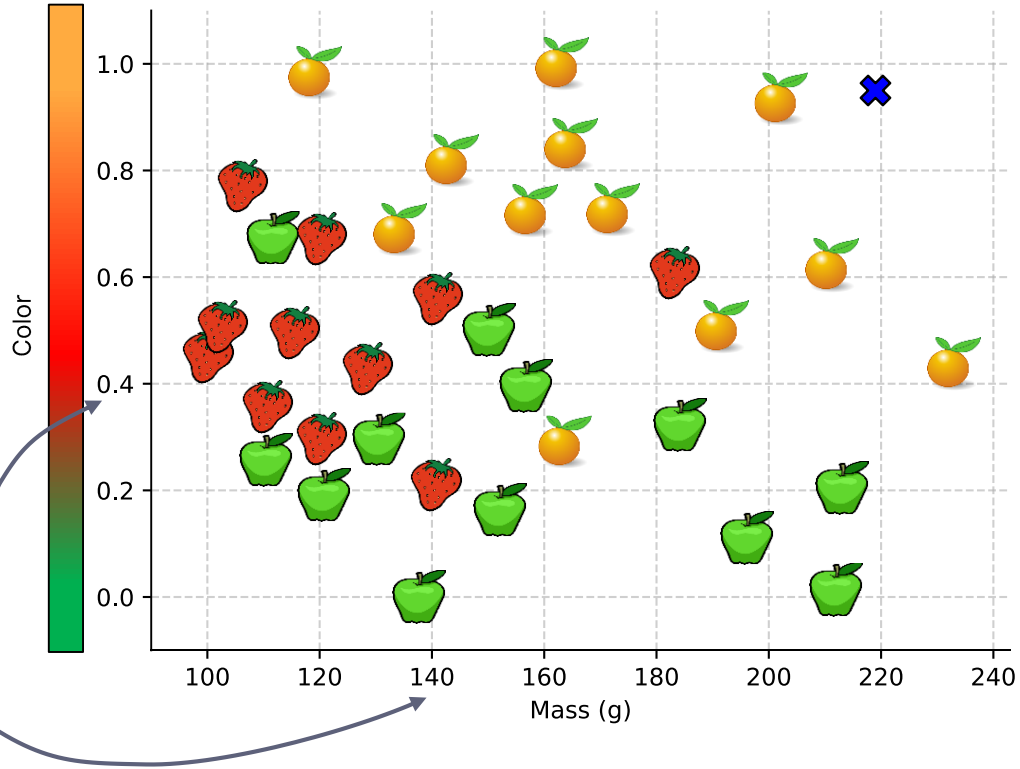
Feature Normalization

Feature Normalization



Feature Normalization

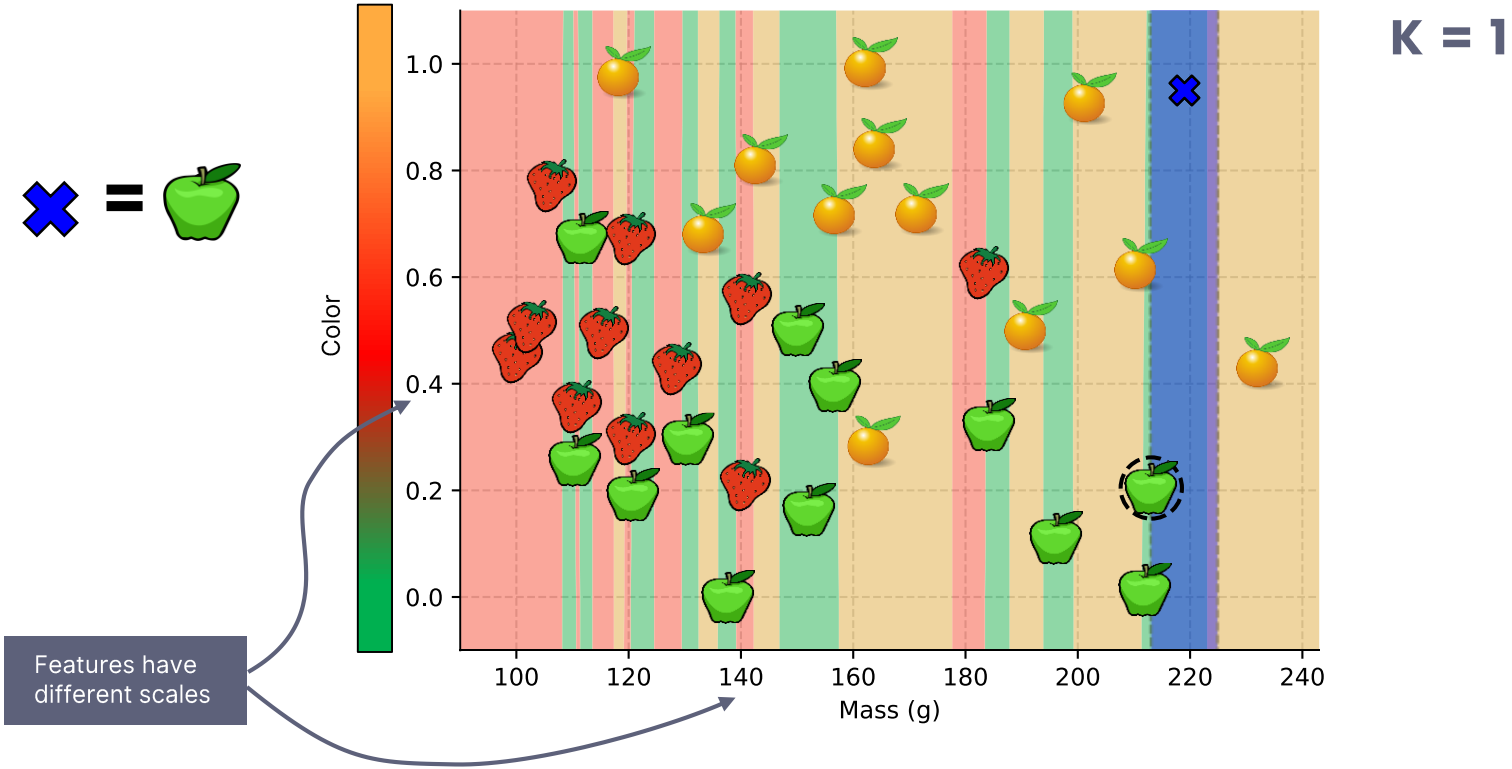
✖ = ?



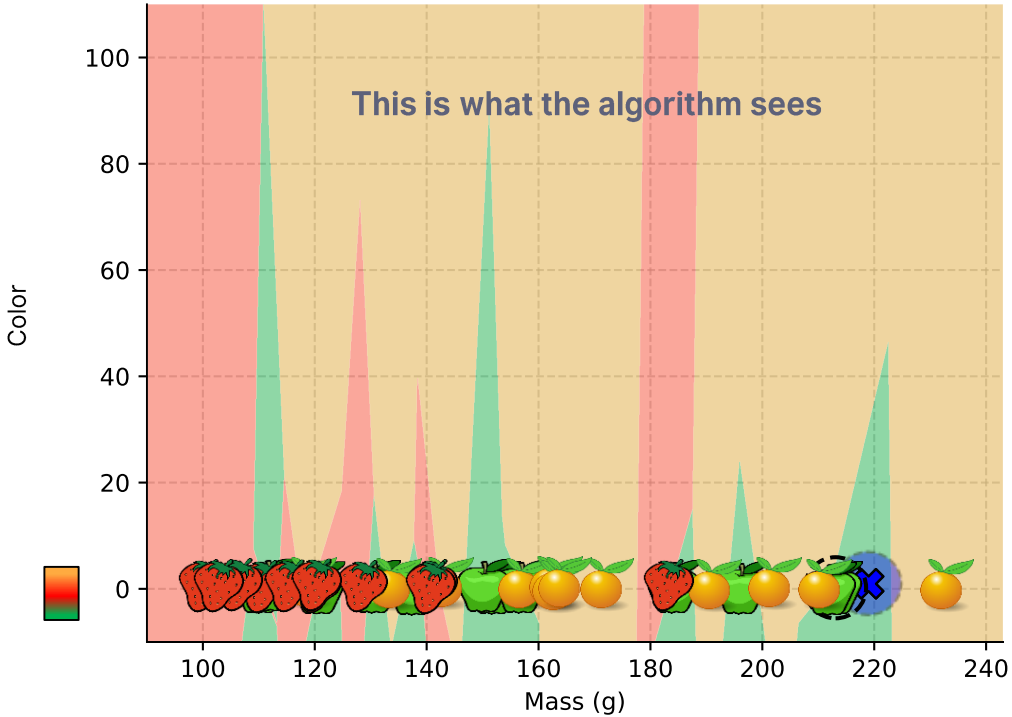
K = 1

Features have different scales

Feature Normalization



Feature Normalization



K = 1

Feature Normalization

- If features have different scales, some features might dominate others when computing distances.
- We must bring all feature values to the same numeric range:
  - **Min-max scaling** – bring all values to a fixed range, usually between 0 and 1:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

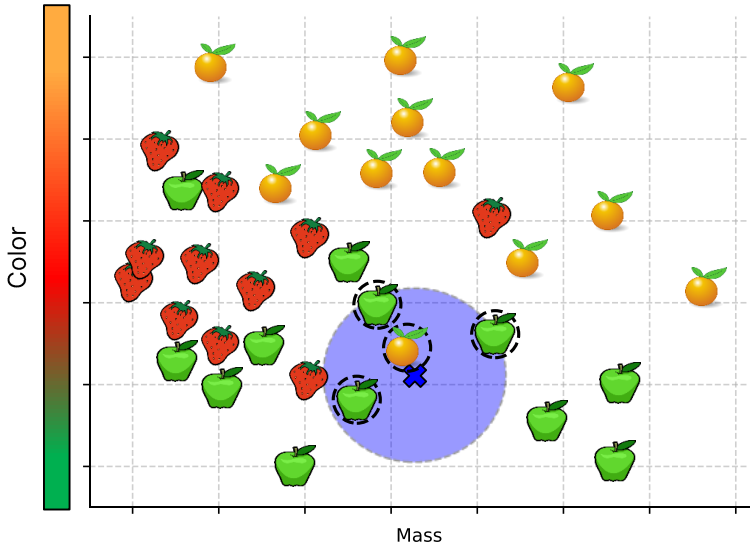
  - **Standardization (Z-score normalization)** – scale features so that they will have the properties of a *standard normal distribution* (mean $\mu = 0$ and standard deviation $\sigma = 1$):

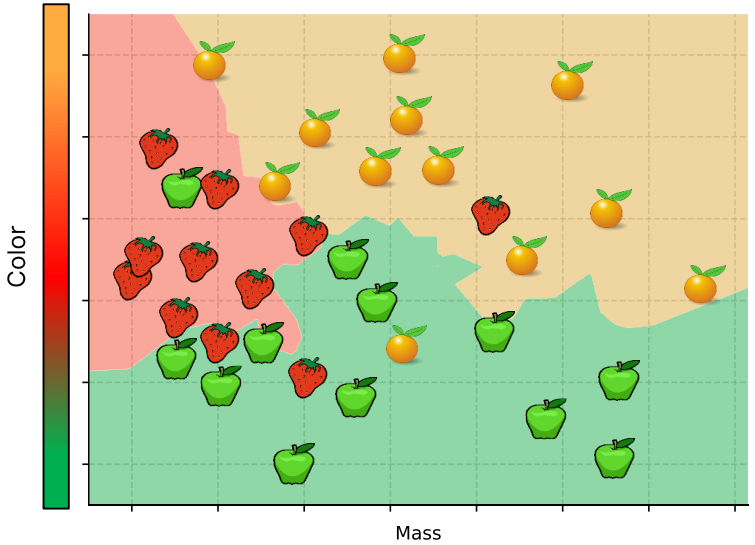
$$X_{norm} = \frac{X - \mu_X}{\sigma_X}$$

Variations of kNN

Weighted KNN

- The influence of a data point can be **weighted by distance**, such that closer points have a greater impact on the prediction.

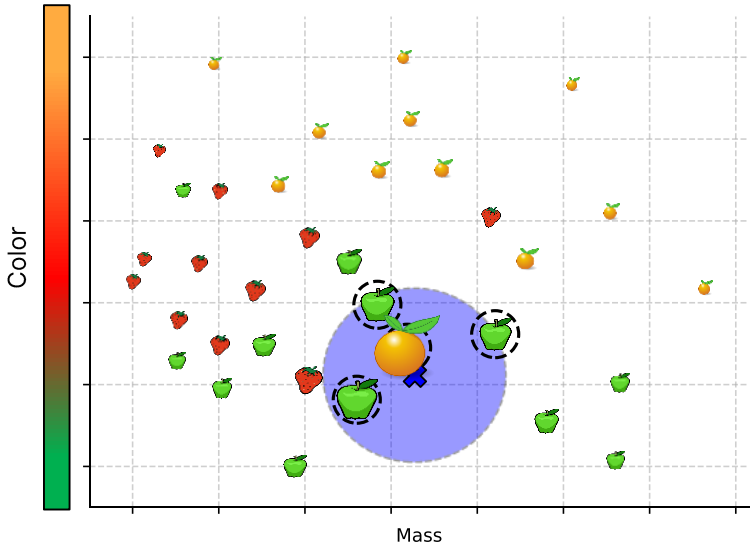




No weighing: ✖ = 75% confidence

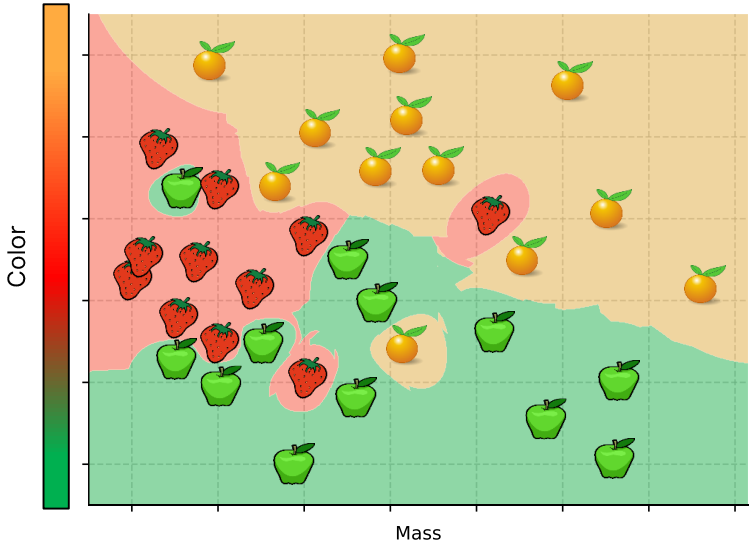
Weighted KNN

- The influence of a data point can be **weighted by distance**, such that closer points have a greater impact on the prediction.



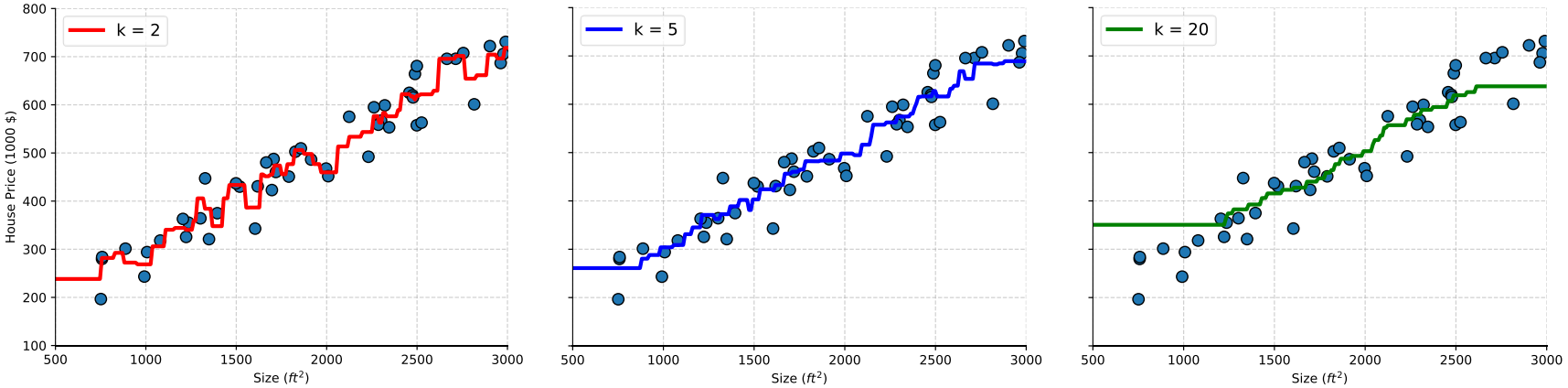
With weighing: ✖ = 62.5% confidence

K = 4



KNN Regression

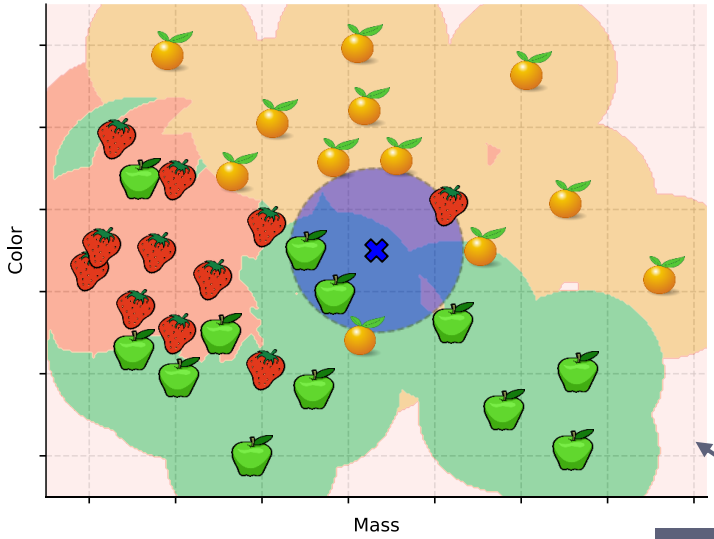
- It is possible to use kNN for *regression* problems.
  - Predicted label is the **average label** of the input's *k* nearest neighbors

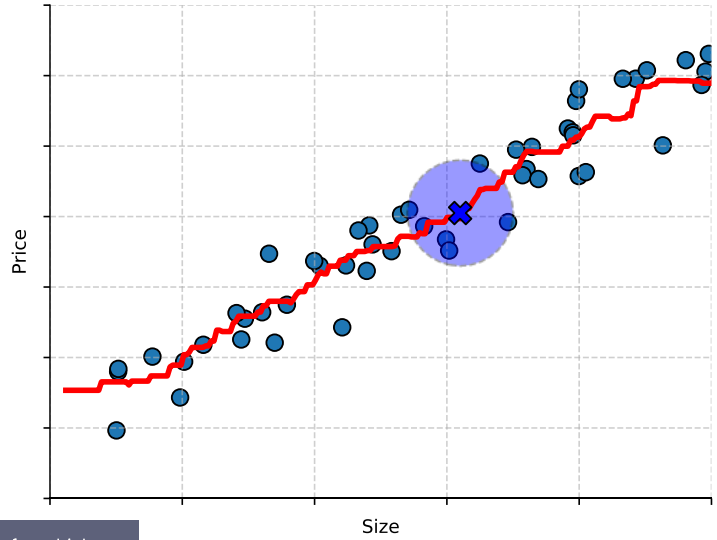


- Increasing *k* makes the function smoother.
  - But it also increases the area at the edges of the training data for which the prediction is constant.

Radius Nearest Neighbors

- Variant of the nearest neighbors algorithm:
  - Instead of always using a fixed number of neighbors, use all the neighbors within a **fixed radius** around the target point.





There will be points for which prediction is undefined.

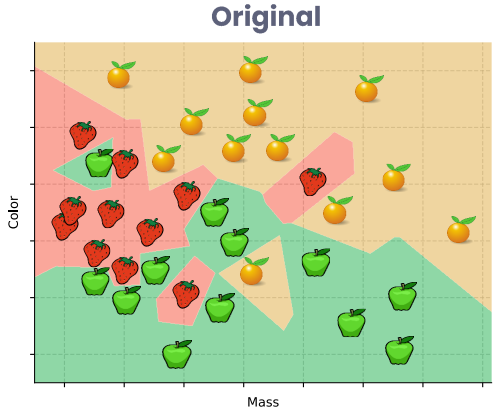
Optimizing KNN

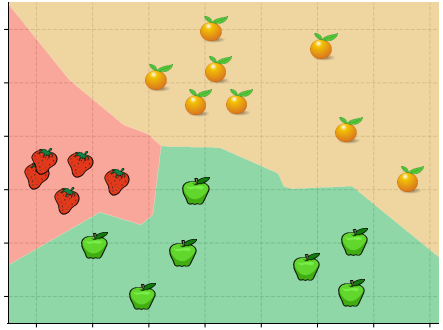
Under-Sampling Data

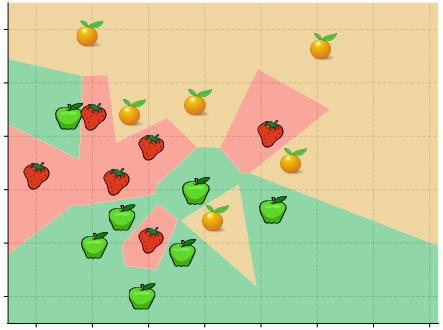
- **Edited Nearest Neighbors** eliminates any training sample for which the prediction of its *k* nearest neighbors (excluding itself) does not agree with its own label.
  - It has the effect of *removing outliers* and samples which are *very close to the border* resulting in a smoother decision boundary.
  - It is both a *regularization technique* and a way of reducing the dataset size.

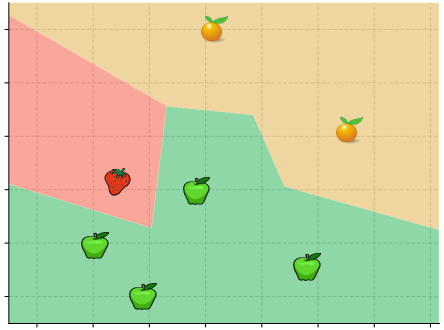
- **Condensed Nearest Neighbors** eliminates points which have little effect on the decision boundary.
  - Original algorithm:
    - Create a set *U* with a small seed of points for each class of the original set *X*
    - For each point in *X*:
      - Move it to *U* if it is **incorrectly** classified by kNN with the current *U* as training set.
    - Use *U* instead of *X*
  - It is very sensitive to the initial random seed of points and the order in which points are checked.

Under-Sampling Data







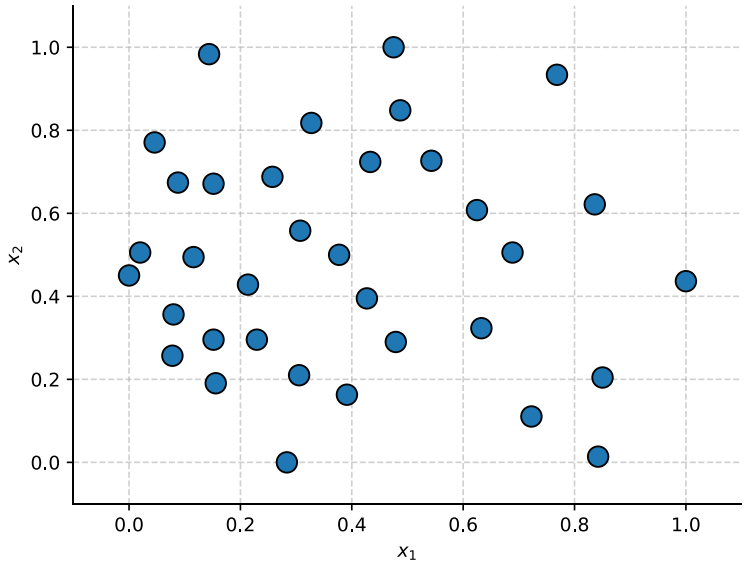


K-d Trees

- A **k-d tree** (k-dimensional tree) is a *binary space-partitioning data structure* for organizing a set of k–dimensional points.
- It is used for efficient queries, such as *range searches* and *nearest neighbors searches*.
- It is most suitable for situations when the number of points N is much larger than the number of dimensions k (typically $N \gg 2^k$)

Building a K-d Tree

- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.

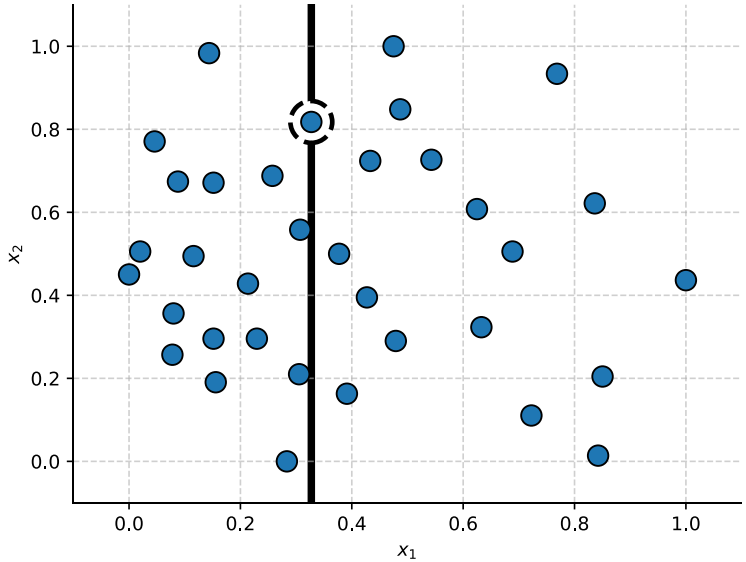


- We compute the *variance* of each feature:

$$\text{var}(x_1) = 0.074$$
$$\text{var}(x_2) = 0.07$$

Building a K-d Tree

- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.



- We compute the *variance* of each feature:

$$\mathbf{var}(x_1) = \mathbf{0.074}$$
$$\mathrm{var}(x_2) = 0.07$$

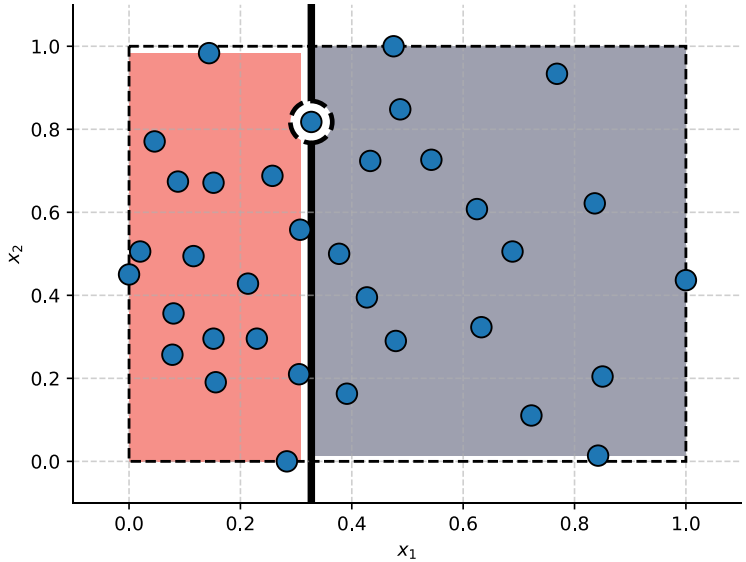
- We find the *median* point on the feature with the larger variance:

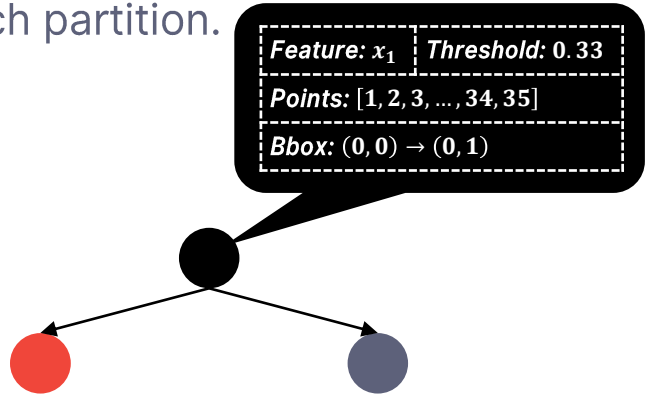
$$\mathrm{median}(x_1) = 0.327$$

The median is not the mean!
The median is the point in the middle of the sorted array.

Building a K-d Tree

- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.

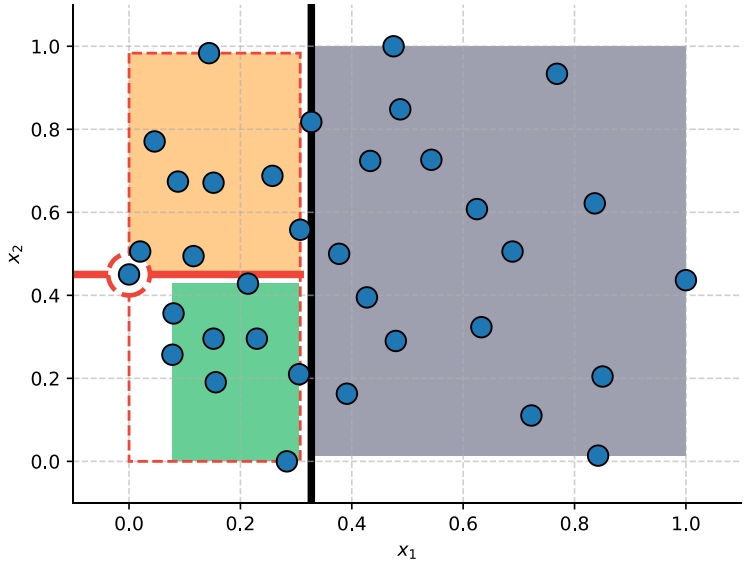


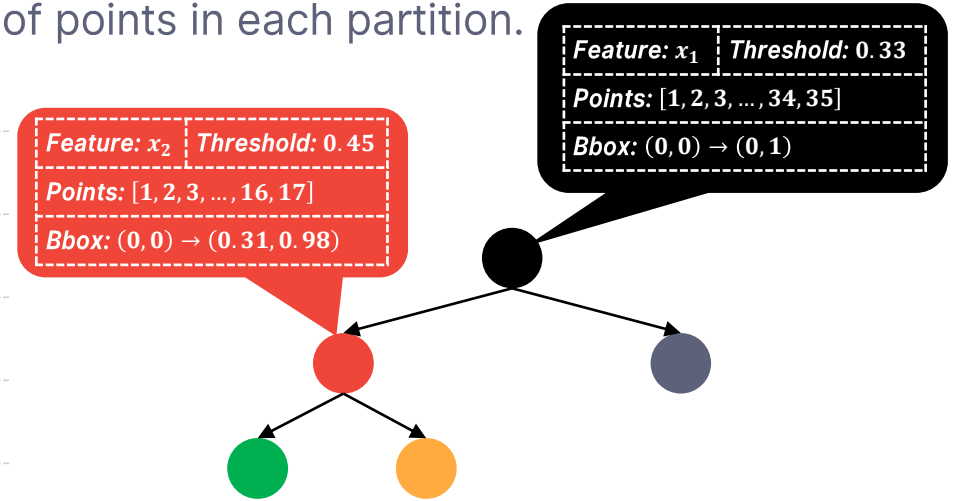


Each node contains:
- The feature it used for splitting and the threshold.
- The list of points considered
- A bounding box around the points.

Building a K-d Tree

- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.

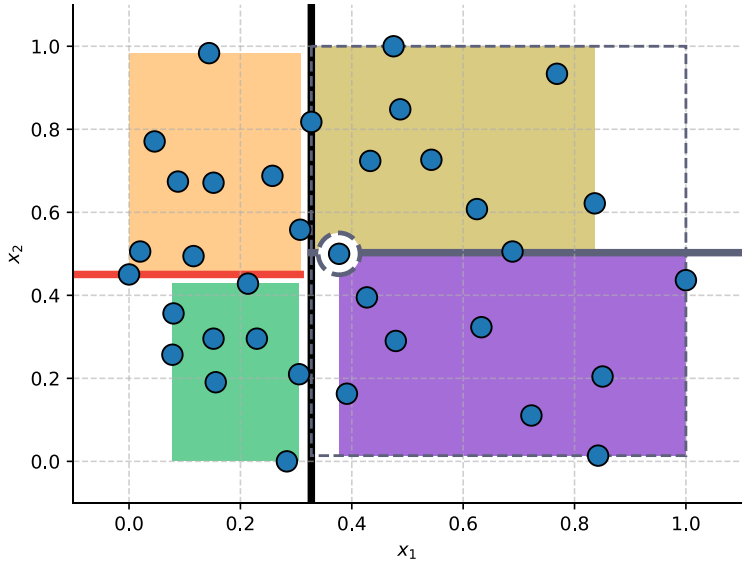


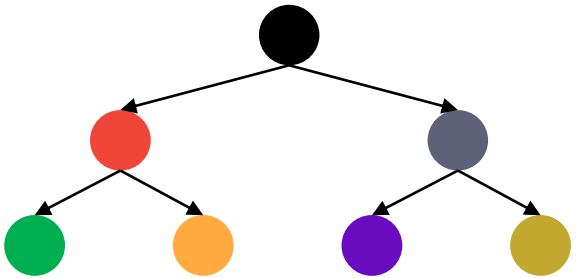


Each node contains:
- The feature it used for splitting and the threshold.
- The list of points considered
- A bounding box around the points.

Building a K-d Tree

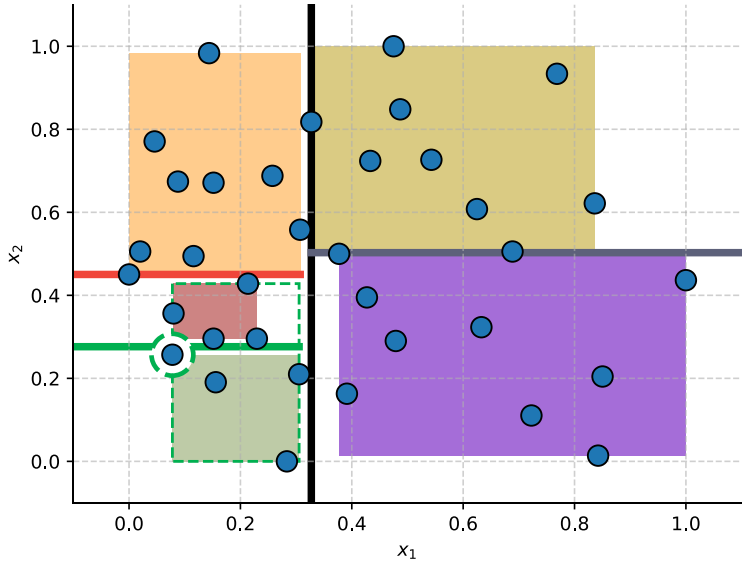
- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.

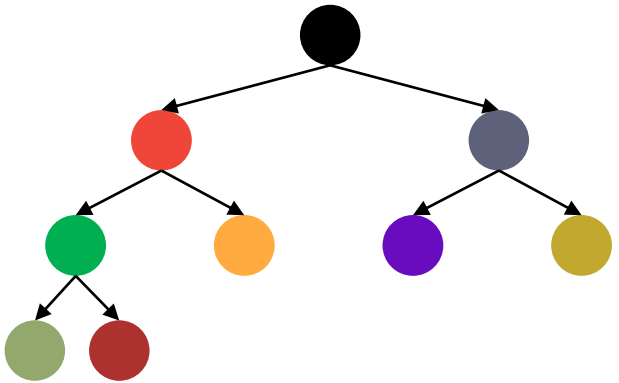




Building a K-d Tree

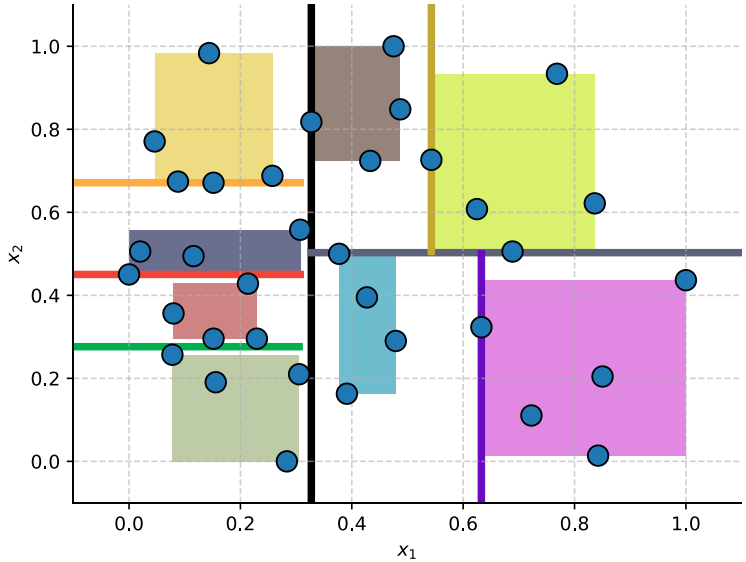
- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.

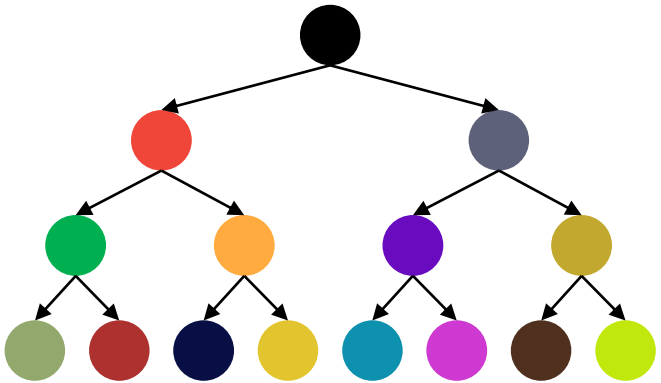




Building a K-d Tree

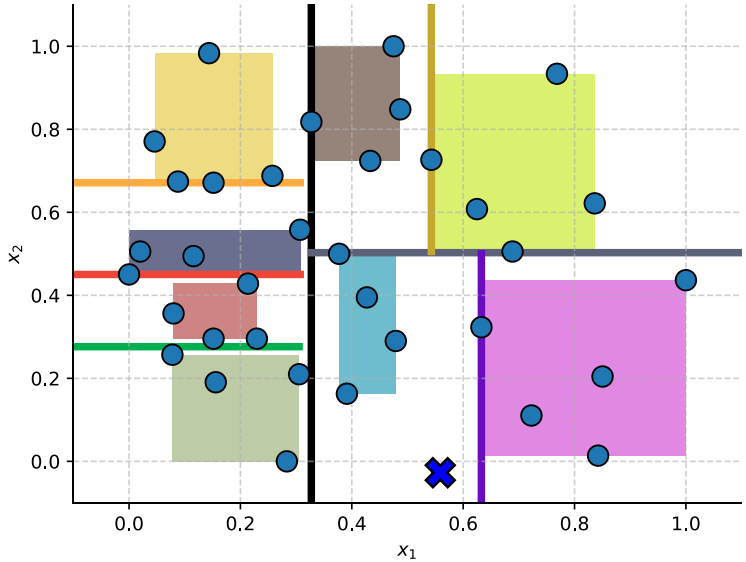
- Recursively find a feature and a value to split the space in two, until we have a small enough number of points in each partition.

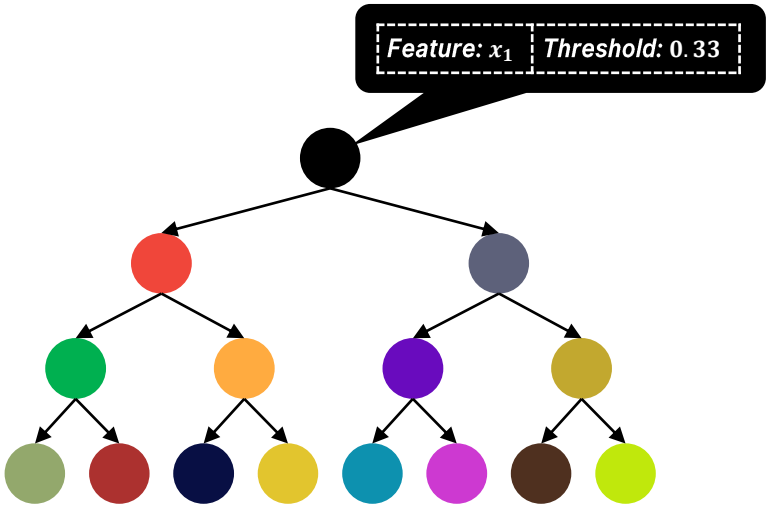




Using a K-d Tree for kNN

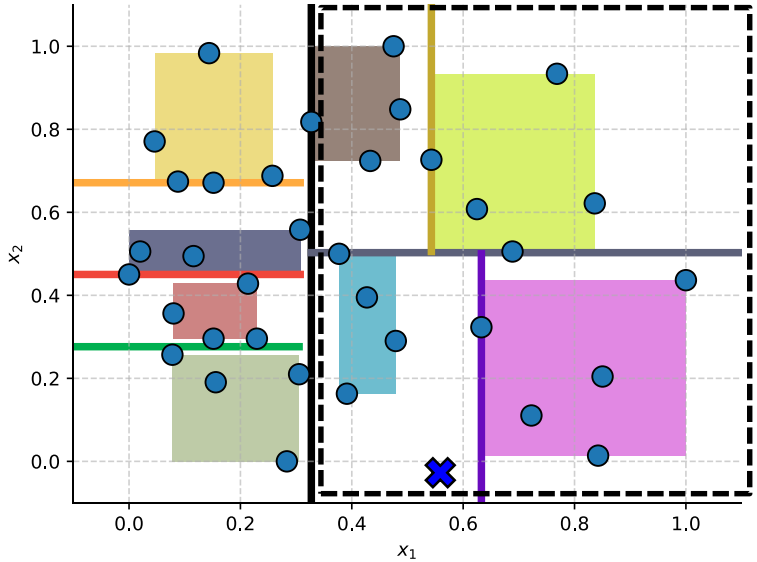
- We find the partition which contains the target point.

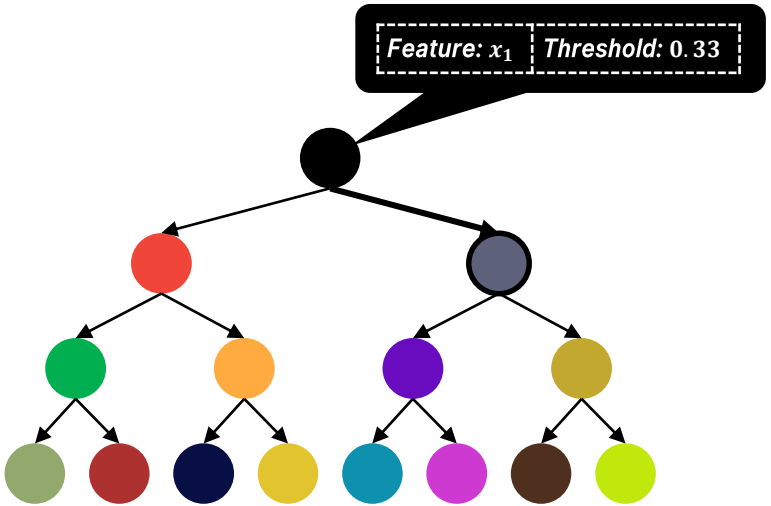




Using a K-d Tree for kNN

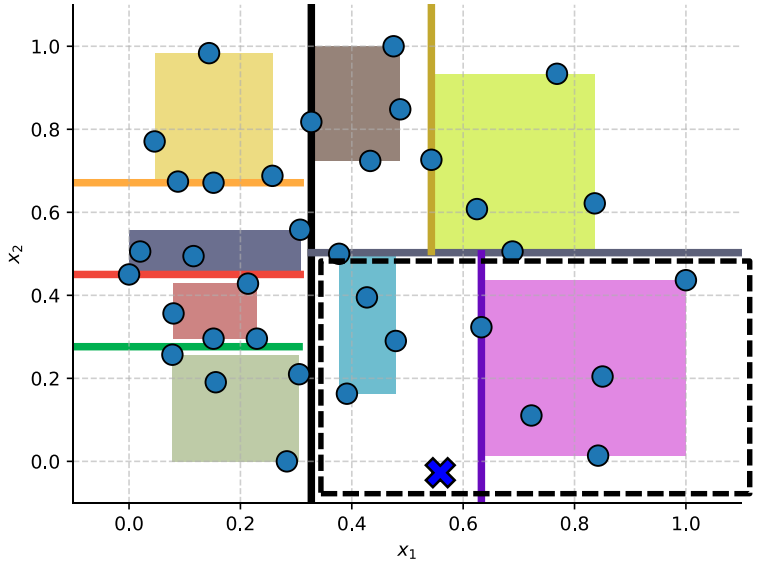
- We find the partition which contains the target point.

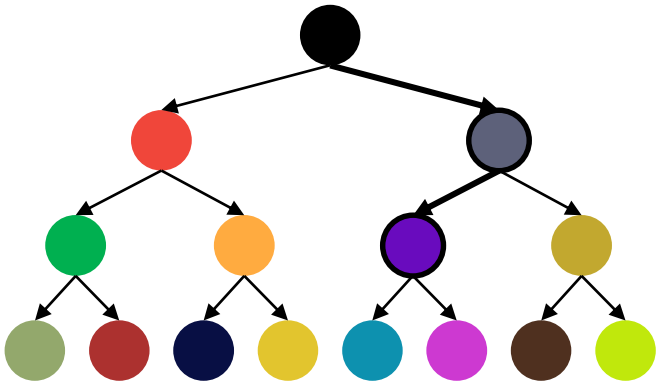




Using a K-d Tree for kNN

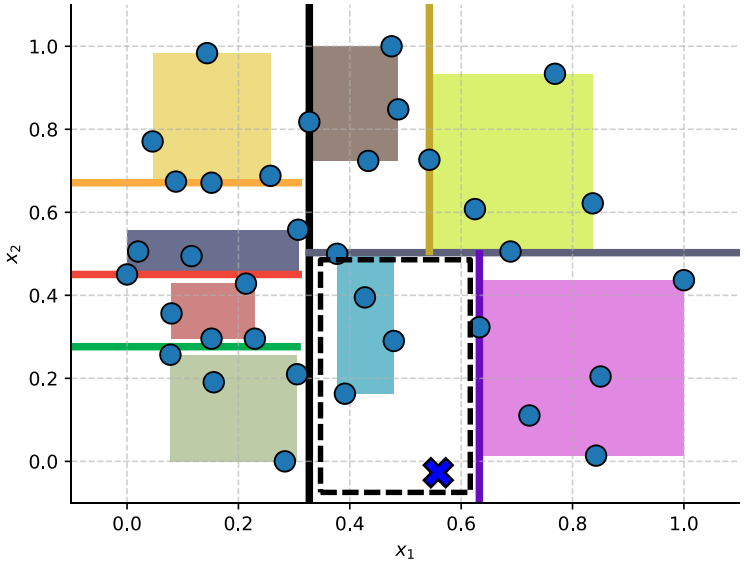
- We find the partition which contains the target point.

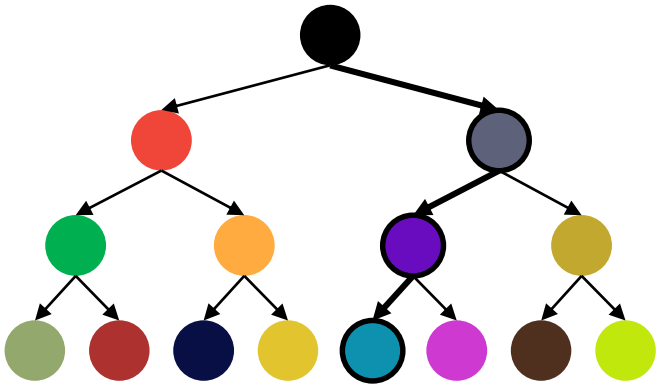




Using a K-d Tree for kNN

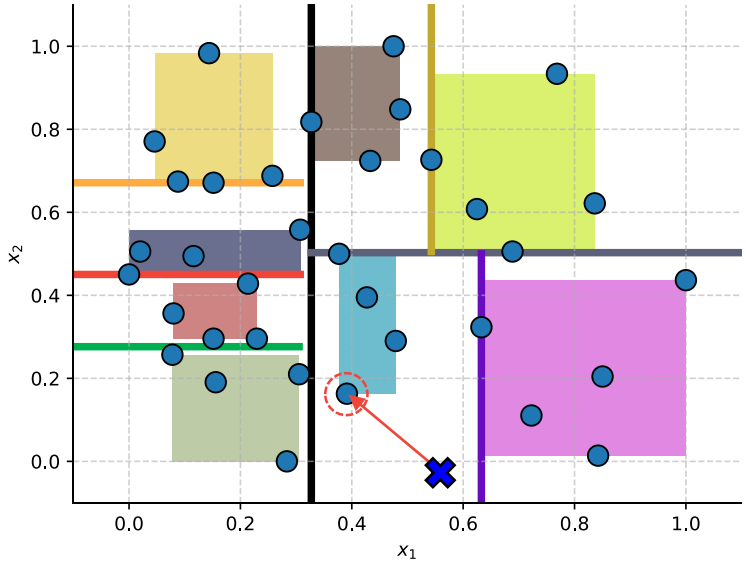
- We find the partition which contains the target point.



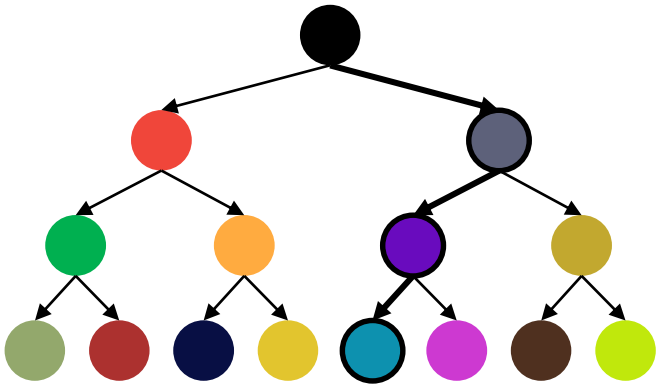


Using a K-d Tree for kNN

If $k > 1$ we keep a list of nearest neighbors.



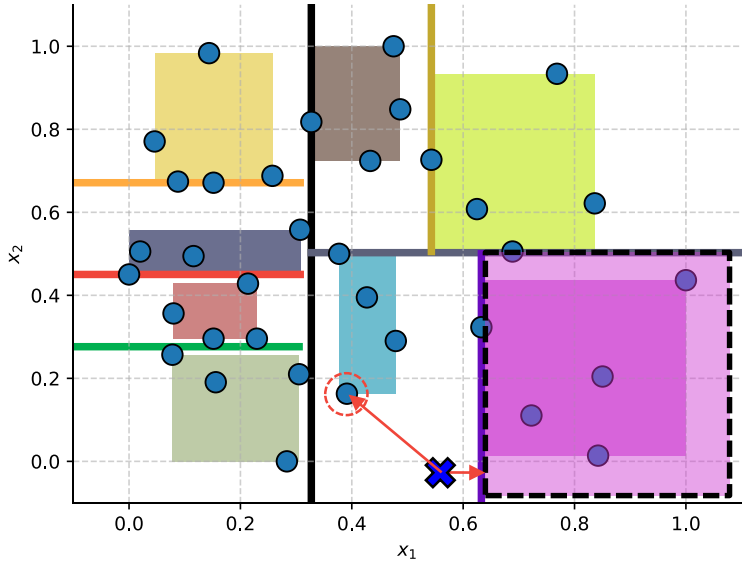
- We find the partition which contains the target point.
- **We look for the nearest neighbor in the list of points of the current node.**



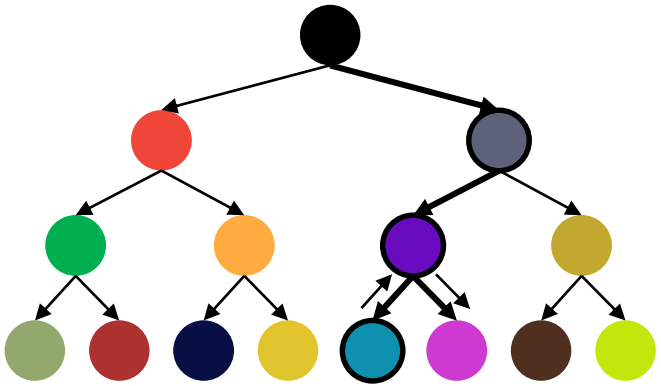
Current NN: Point 19

Min Distance: 0.23

Using a K-d Tree for kNN



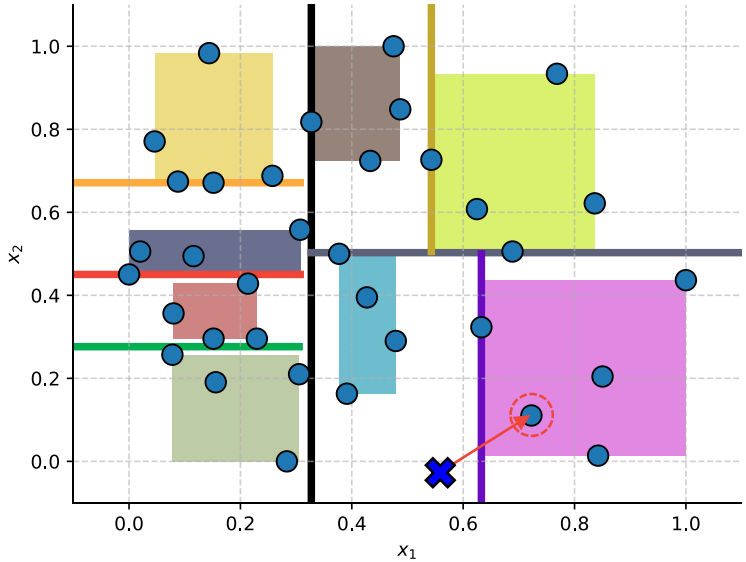
- We find the partition which contains the target point.
- We look for the nearest neighbor in the list of points of the current node.
- **We backtrack and try to find closer points.**
- We only check the points of an area if the distance to its bounding box is smaller than the current min distance.



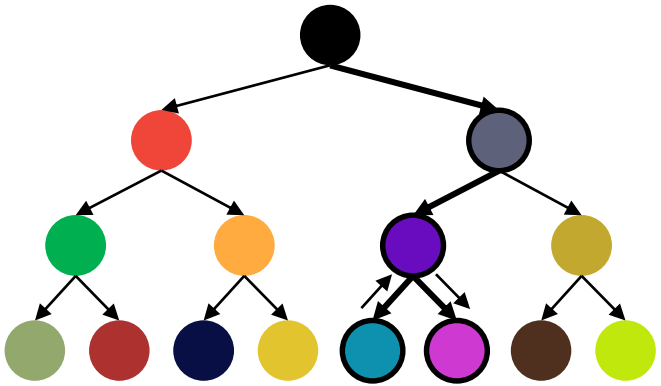
Current NN: Point 19

Min Distance: 0.23

Using a K-d Tree for kNN



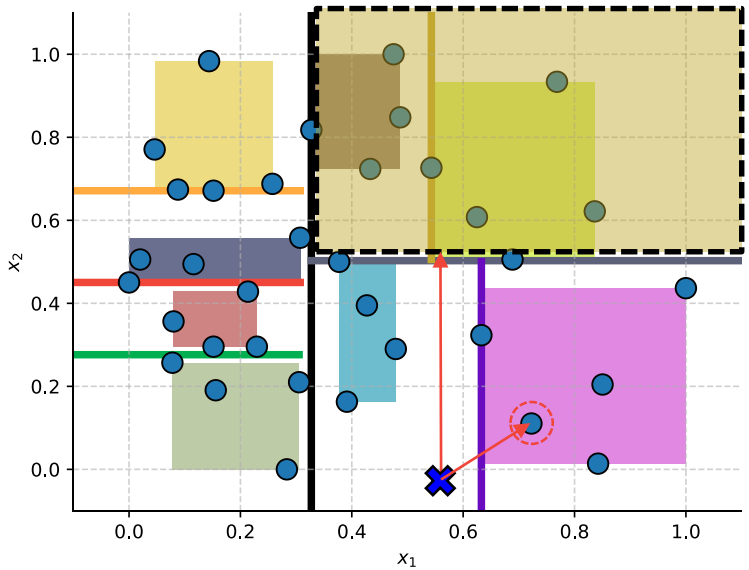
- We find the partition which contains the target point.
- We look for the nearest neighbor in the list of points of the current node.
- We backtrack and try to find closer points.
- **We only check the points of an area if the distance to its bounding box is smaller than the current min distance.**



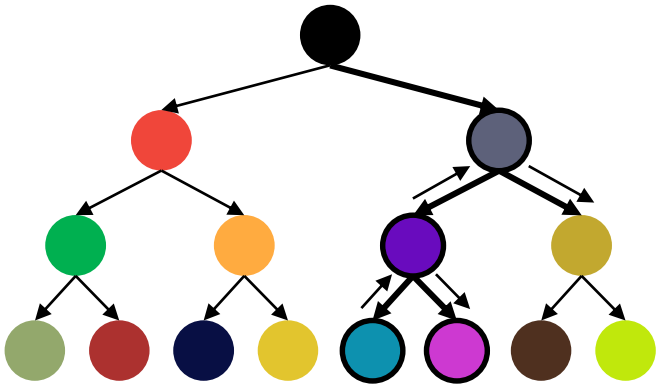
Current NN: Point 28

Min Distance: 0.2

Using a K-d Tree for kNN



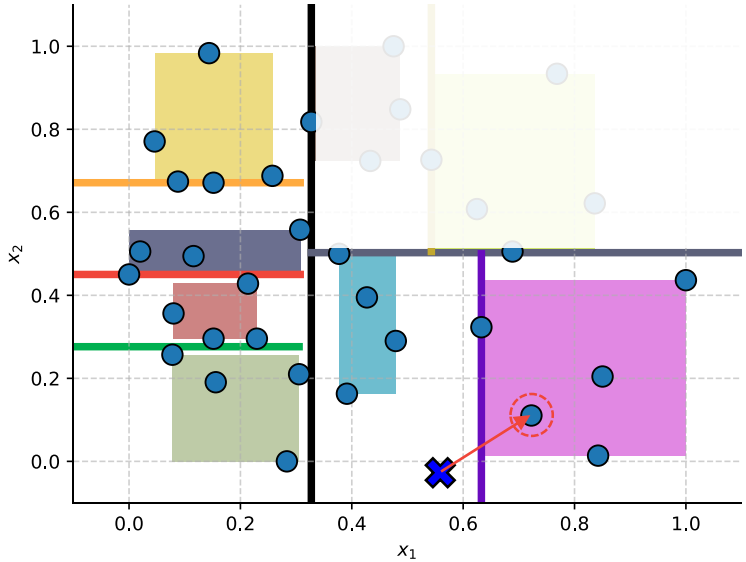
- We find the partition which contains the target point.
- We look for the nearest neighbor in the list of points of the current node.
- We backtrack and try to find closer points.
- We only check the points of an area if the distance to its bounding box is smaller than the current min distance.



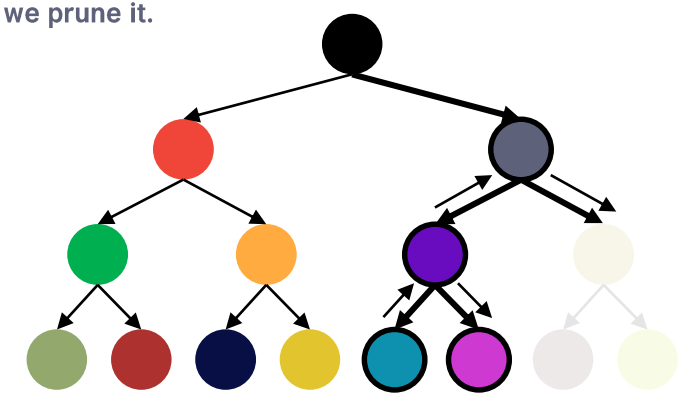
Current NN: Point 28

Min Distance: 0.2

Using a K-d Tree for kNN



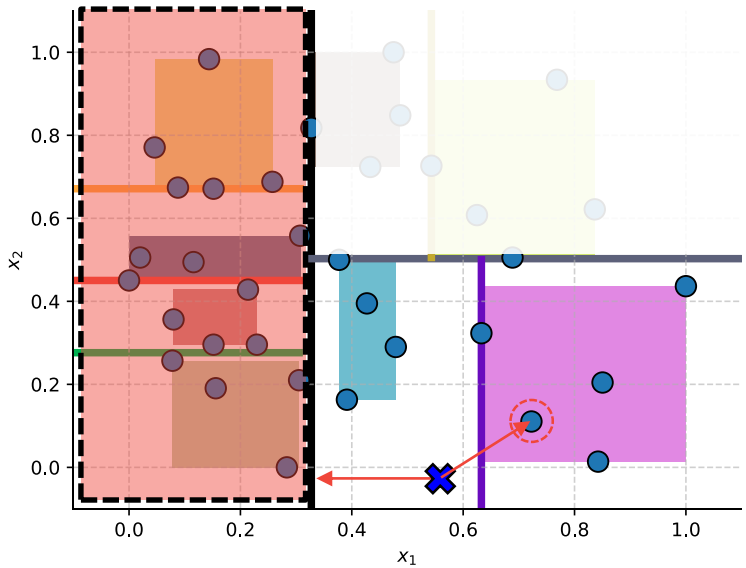
- We find the partition which contains the target point.
- We look for the nearest neighbor in the list of points of the current node
- We backtrack and try to find closer points.
- We only check the points of an area if the distance to its bounding box is smaller than the current min distance.
- **If not, we prune it.**



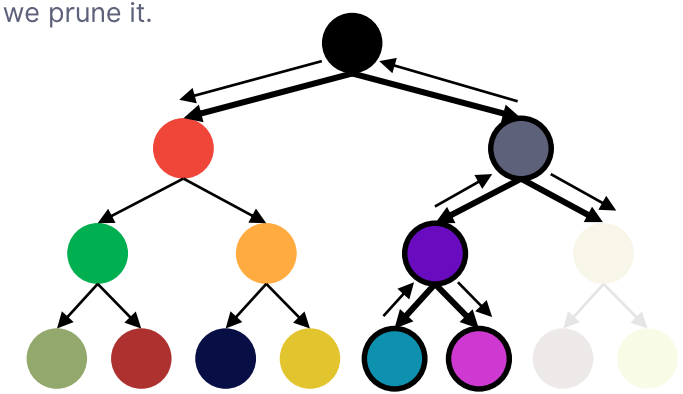
Current NN: Point 28

Min Distance: 0.2

Using a K-d Tree for kNN



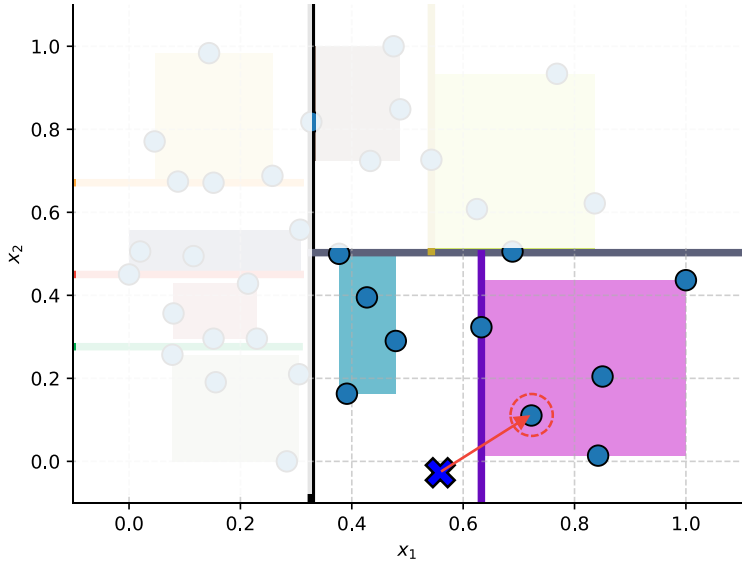
- We find the partition which contains the target point.
- We look for the nearest neighbor in the list of points of the current node
- We backtrack and try to find closer points.
- We only check the points of an area if the distance to its bounding box is smaller than the current min distance.
- If not, we prune it.



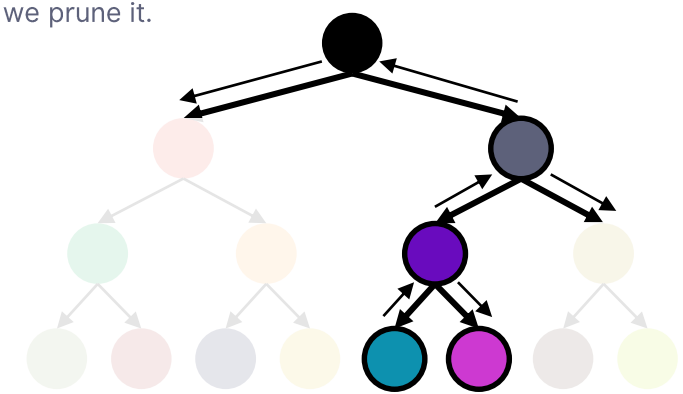
Current NN: Point 28

Min Distance: 0.2

Using a K-d Tree for kNN



- We find the partition which contains the target point.
- We look for the nearest neighbor in the list of points of the current node
- We backtrack and try to find closer points.
- We only check the points of an area if the distance to its bounding box is smaller than the current min distance.
- If not, we prune it.

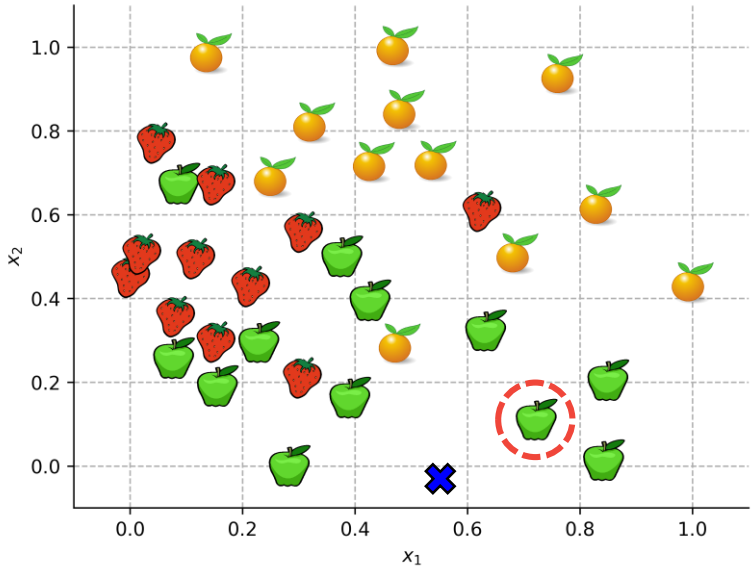


Current NN: Point 28

Min Distance: 0.2

Using a K-d Tree for kNN

- We found the nearest neighbor by checking only 9 out of 35 points.



- Sometimes we don't need to know the absolute nearest neighbors.
- We can further reduce the search space by only considering nodes whose bounding boxes are closer than a fraction of the current minimum distance. (*Approximate kNN*).

Conclusions

- **K-Nearest Neighbors (kNN)** is a *non-parametric*, *instance-based*, supervised learning algorithm.
- **kNN** predicts the label of a new instance as the *most common (or average)* label of the *k* **closest** training samples in feature space.
  - The **distance (or similarity) measure** is very important and must suit the problem.
  - The number of neighbors *k* is a *hyperparameter* and should be chosen by validation.
- There are methods for making the dataset more robust (**Edited NN** and **Condensed NN**) and methods of *partitioning* the feature space for easier neighbor searching (**K-d Trees**).
- Pros and Cons:
  - \+ It is very easy to understand, implement and to *interpret the results* (give the reason for a certain prediction)
  - \+ There is no *training phase*, so new samples can be easily incorporated in the dataset
  - \- Inference takes more time.
  - \- Sensitive to *noise* in the training data and to *irrelevant features*.

KNN in Python

```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor, RadiusNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors = 3, weights = 'distance') # 3-NN weighted by distance
clf.fit(X, y)

clf.predict([x]) # predict class for x
clf.predict_proba([x]) # predict probabilities for x (ratio of neighbors who voted for each class)

clf.kneighbors([x]) # indices of the nearest neighbors of x (and distances to them)

clf  = KNeighborsRegressor(n_neighbors = 5) # 5-NN regression
```

History of kNN

- ***"An important Contribution to Nonparametric Discriminant Analysis and Density Estimation"***
  - Non-parametric method for pattern classification. E. Fix and J.L. Hodges, 1951
  - Mentions the nearest neighbor rule.
- ***"Nearest neighbor pattern classification"***

  T. Cover and P. Hart, 1967
  - The k-nearest-neighbor classification error is bounded above by twice the *Bayes error rate*
- ***"The condensed nearest neighbor rule"***
  - Condensed Nearest Neighbors P. Hart, ,1968
- ***"Asymptotic" Properties of Nearest Neighbor Rules Using Edited Data"***
  - Edited Nearest Neighbors Dennis L. Wilson, 1972
- ***"Multidimensional Binary Search Trees Used for Associative Searching"***
  - Introduced the *k-d tree* data structure Jon Louis Bentley, 1975

Keywords

K-Nearest Neighbors (kNN)

Non-parametric | Instance-based Learning | Lazy Learning

Bias | Variance

Normalization | Min-max Scaling | Standardization (Z-score Normalization)

Distance (Similarity) Measure | Minkowski Distance | Cosine Similarity

Under-sampling | Edited Nearest Neighbors | Condensed Nearest Neighbors

K-d Trees