Clustering algorithms & (K-NN) classification

Lesson 5: Scikit learn

An AI Commons initiative

What is clustering?

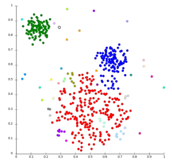
- Clustering is the process of grouping the data points into similar clusters (groups)
- Similarities between data points is measured as ‘distance’ between them
- Clustering is unsupervised, as we do not have a labeled data set to identify the grouping of data.
- K-NN is supervised classification, using existing labels to predict new ones
  - Not to be confused with clustering or k-means algorithm

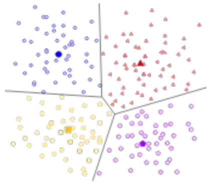
Where is clustering applied?

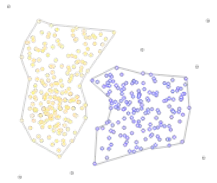
- Where we need to identify similarities, or differences, between data points
- Some areas of application:
  - Market/customers segmentation, document categorization
  - Grouping search results, making recommendations, generalization
  - Image segmentation
  - Anomaly detection
  - Inferring missing data

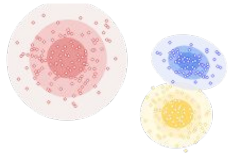
Clustering algorithms families

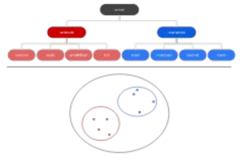
Family	Similarity Concept	Pros	Cons
Connectivity	Data points distance (closeness) to each other	Easy to interpret	Lack scalability
Centroid	Based on the distance to the cluster center	Efficient	Sensitive to outliers, and initial conditions
Density	Connects areas of high density into a shape	Arbitrary similarity shape	Does not scale well to multiple dimensions
Distribution	Assumes the data follows a distribution	Allows for probability of belonging based on distribution	Not easy when you don't know the distribution of your data
Hierarchical	Creates a tree structure	Suitable for taxonomies.	Does not work with missing data, and tough for big data sets


Connectivity


Centroid

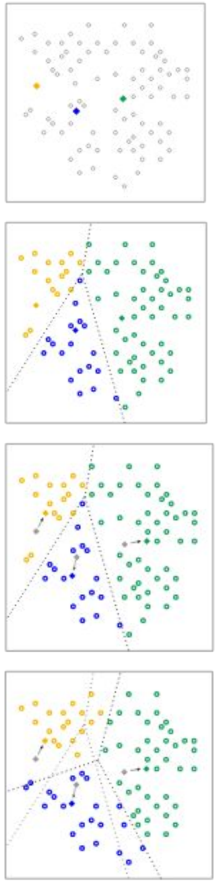

Density


Distribution


Hierarchical

Images Source

Centroid-based K-Means Algorithm

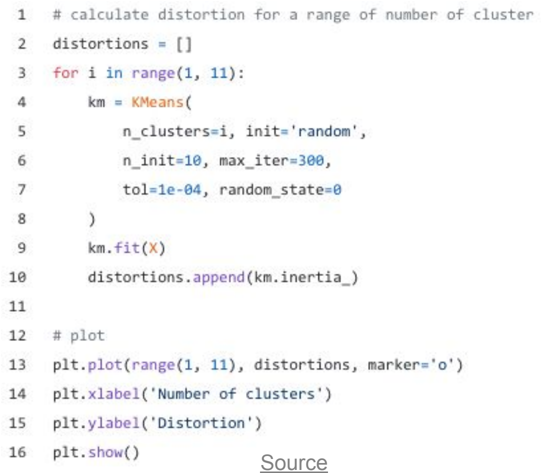


Images Source

1. Specify the number of clusters ‘k’, and assign them randomly to ‘k’ centroid data points
2. Assign each data point to the closest centroid
3. Recompute the centroid of each cluster
4. Repeat until you ‘are satisfied’
   a. Stability: points do not change clusters (convergence), or
   b. Using another suitable criteria

Finding the K in K-means

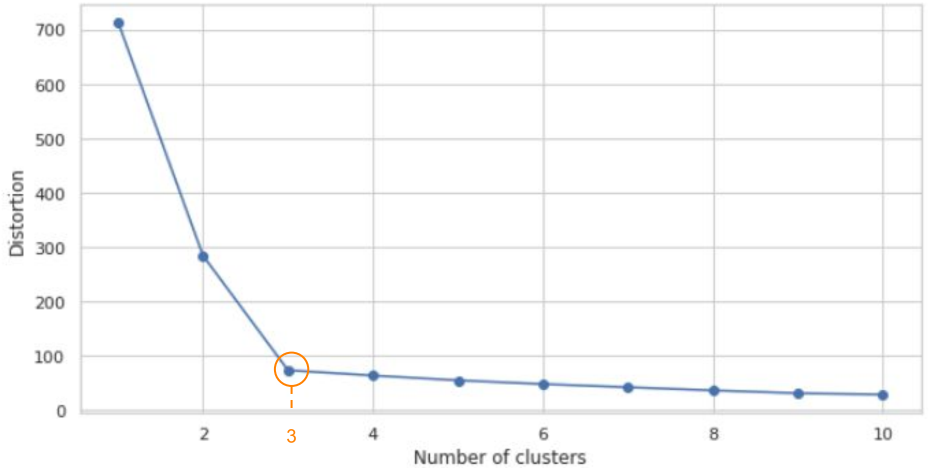
- How to decide for the number of clusters?
  - The “elbow” curve indicates the “point of diminishing returns”, beyond which the effort is not worth the cost.
  - Intuitively, it can be measured as the point where within-cluster distance (“distortion”) starts to decrease less rapidly.



```python
# calculate distortion for a range of number of cluster
distortions = []
for i in range(1, 11):
    km = KMeans(
        n_clusters=i, init='random',
        n_init=10, max_iter=300,
        tol=1e-04, random_state=0
    )
    km.fit(X)
    distortions.append(km.inertia_)

# plot
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()
```

Source



Don’t confuse K-means with K-NN !

- K-means is an unsupervised clustering algorithm
  - Groups unlabeled data points into clusters
- K-NN is a supervised learning algorithm for classification
  - Uses labeled data points to infer the categories of unlabelled data points
- The confusion comes from the fact they both:
  - Use ‘k’ as a parameter to group data points
  - Estimate the initial ‘k’ value
  - Have an iterative approach based on ‘distance’

K-NN algorithm key idea

Infers from similarity:

- We assume that data points that share close features are similar
- Hence we infer unlabelled data based on its 'proximity' to known labeled data.

Used for classification:

- Suppose we have a dataset of 2 classes: Category A and Category B. We are given a new datapoint we've never seen before, and want to say if it is part of Category A or B
- We can think of using the 'distance' between data points to drive this decision
- Which category is our new datapoint 'closest' to?

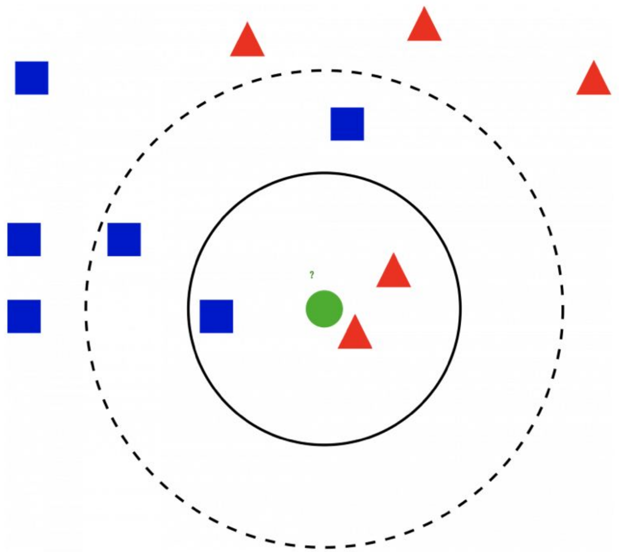


Image Source

KNN - A spam e-mail example

Suppose we are trying to classify if an email is spam or not. Our data will consist of 2 features: The number of times the word ‘free’ occurs in the email, and the number of times the phrase ‘click here’ occurs.

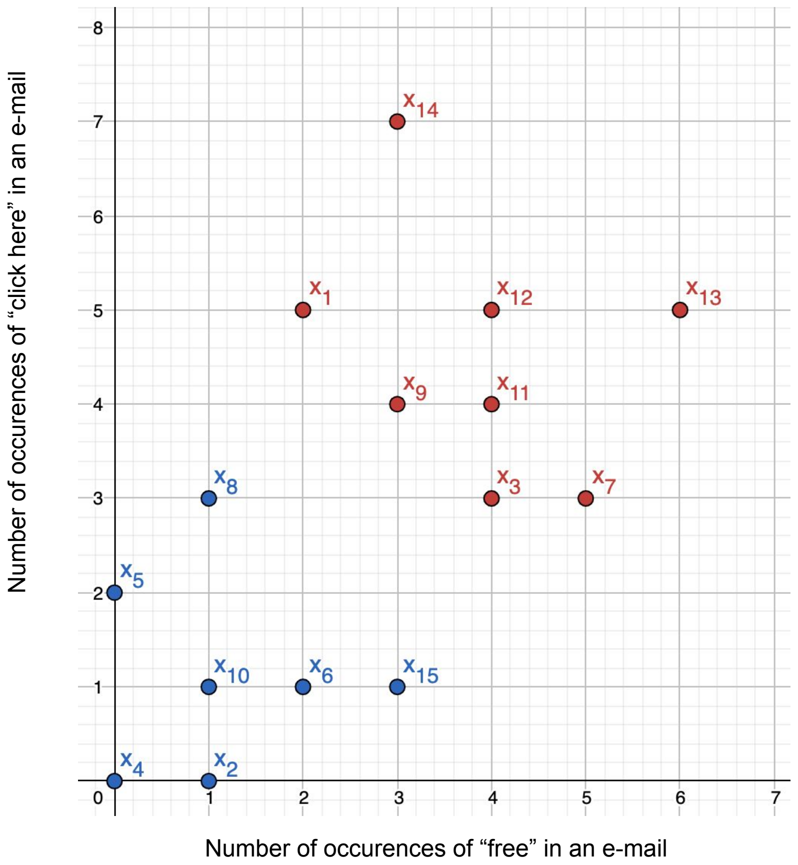
$$X = \begin{bmatrix} 2 & 4 \\ 1 & 0 \\ \vdots & \vdots \\ f_{n-1} & c_{n-1} \\ f_n & c_n \end{bmatrix} \quad L = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ l_{n-1} \\ l_n \end{bmatrix}$$

Key:

X is our training dataset. Each row is a different sample (also known as datapoint). The first value in each data point is the number of times ‘free’ occurred in the email, and the second value is the number of times ‘click here’ occurred.

L is our label vector. 1 indicates the email is spam, 0 indicates it is not.

KNN- Spam e-mail (cont'd)



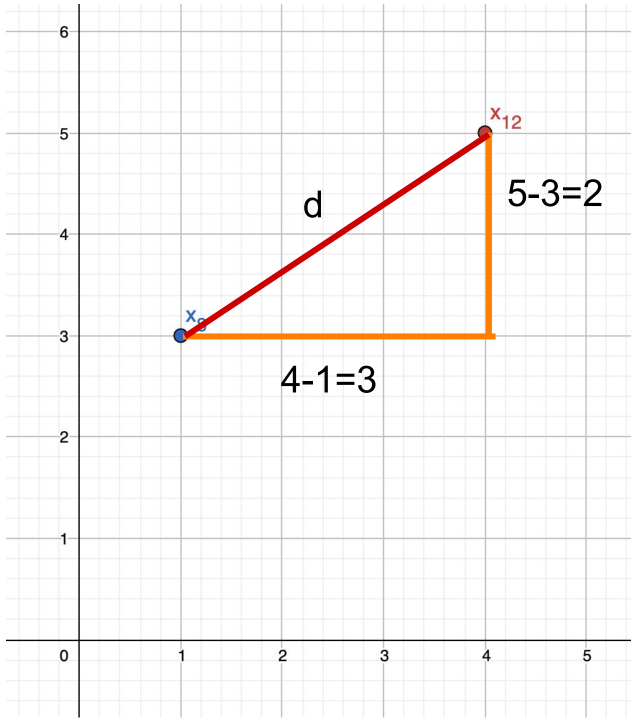
Each axis represents a feature

In this case, the x axis is the number of times 'free' is mentioned in the email, and the y axis is the number of times 'click here' is mentioned in the email

Red points correspond to spam emails

Defining ‘Distance’

We can compute the distance between any 2 data points using the Pythagorean Theorem



$$d^2 = (4-1)^2 + (5-3)^2$$

$$d = \sqrt{(4-1)^2 + (5-3)^2}$$

Euclidean Distance

For 1D:

General Form: For any 2 points, $(x_1), (x_2)$

The Euclidean distance is given by

$$\sqrt{(x_1 - x_2)^2} = |x_1 - x_2|$$

For 2D:

For any two points $(x_1, y_1), (x_2, y_2)$, their Euclidean distance is given by

$$\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

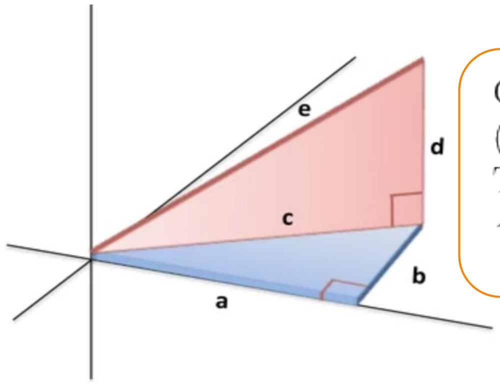
For 3D:

$$a^2 + b^2 = c^2$$
$$c^2 + d^2 = e^2$$
$$a^2 + b^2 + d^2 = e^2 \Leftrightarrow e = \sqrt{a^2 + b^2 + d^2}$$

For n dimensions (commonly written $\mathbb{R}^n$)

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q	= two points in Euclidean n-space
q_i, p_i	= Euclidean vectors, starting from the origin of the space (initial point)
n	= n-space



General Form: For any 2 points, $(x_1, y_1, z_1), (x_2, y_2, z_2)$

The Euclidean distance is given by

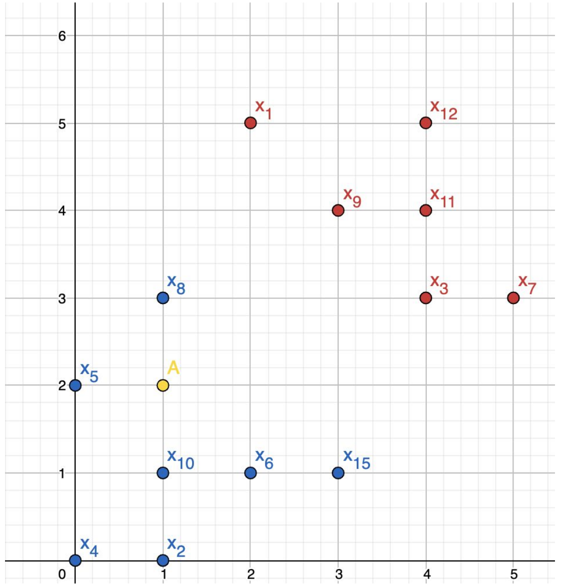
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Image Source

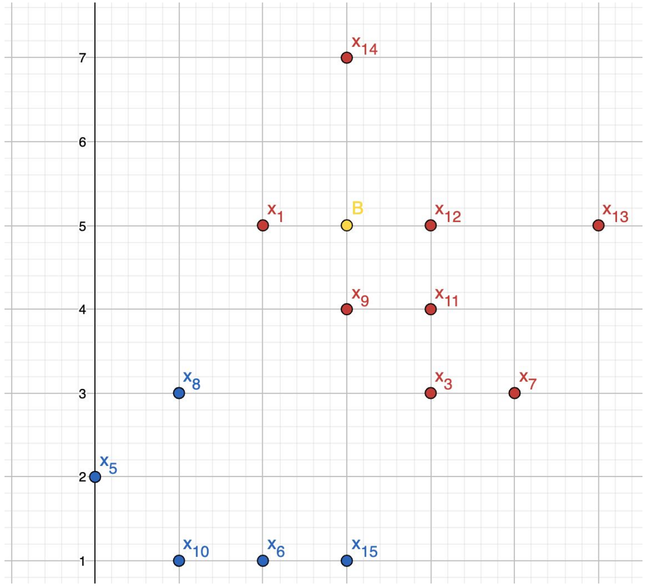
The KNN Algorithm

1. Choose a value of K
2. Find the K *nearest* data points (based on Euclidean Distance)
3. Whichever label is most common in these neighbors, use that as the prediction

Some examples



Consider point A with K=4



Consider point B with K=3

K-NN in Sklearn

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=2,
algorithm='brute')
knn.fit(X, y)

knn.predict(z)
```

X is a np array of the form:

$$\begin{bmatrix} \text{---} x_1 \text{---} \\ \text{---} x_2 \text{---} \\ \vdots \\ \text{---} x_n \text{---} \end{bmatrix}$$

nxm

and y is of the form:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

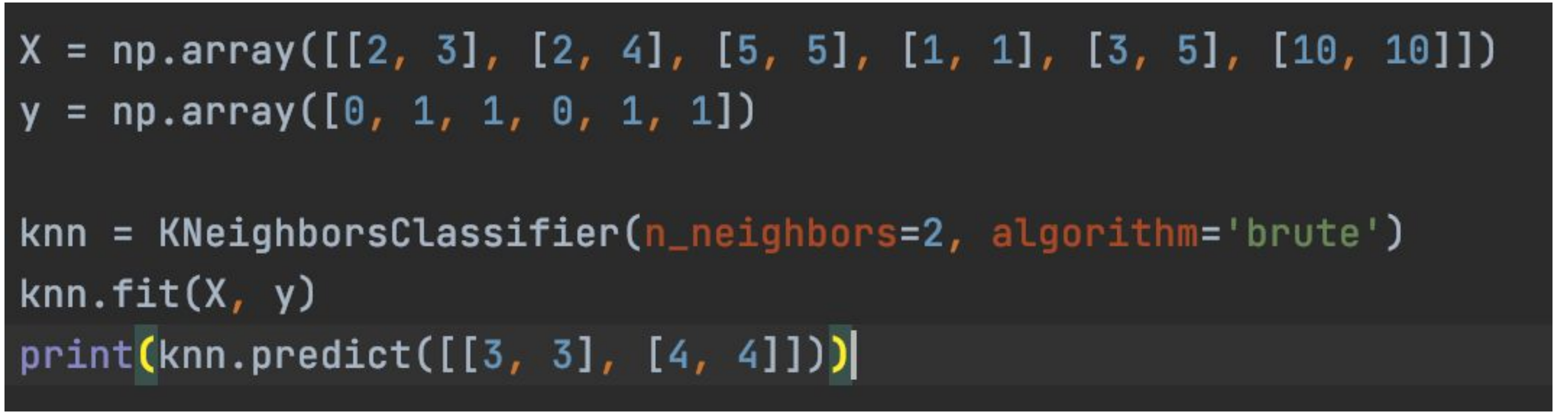
nx1

and z is of the form:

$$\begin{bmatrix} \text{---} z_1 \text{---} \\ \text{---} z_2 \text{---} \\ \vdots \\ \text{---} z_t \text{---} \end{bmatrix}$$

txm

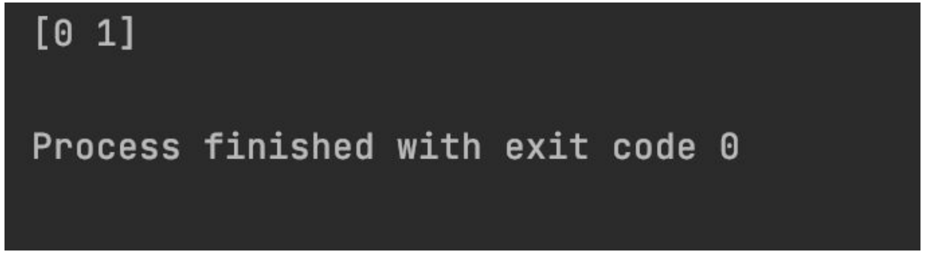
An example



```python
X = np.array([[2, 3], [2, 4], [5, 5], [1, 1], [3, 5], [10, 10]])
y = np.array([0, 1, 1, 0, 1, 1])

knn = KNeighborsClassifier(n_neighbors=2, algorithm='brute')
knn.fit(X, y)
print(knn.predict([[3, 3], [4, 4]]))
```

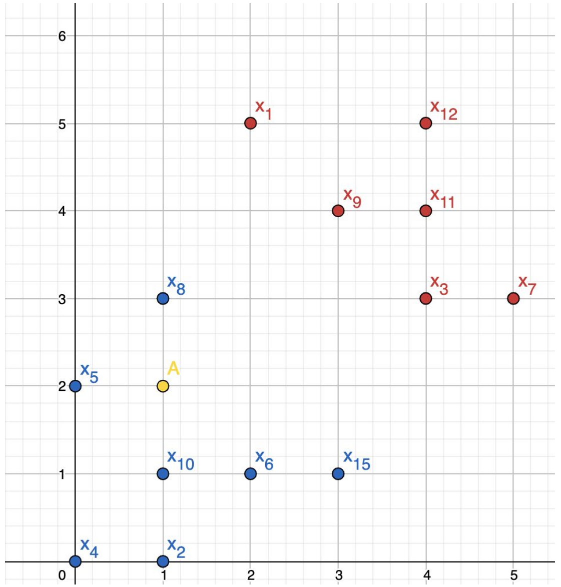
Output:



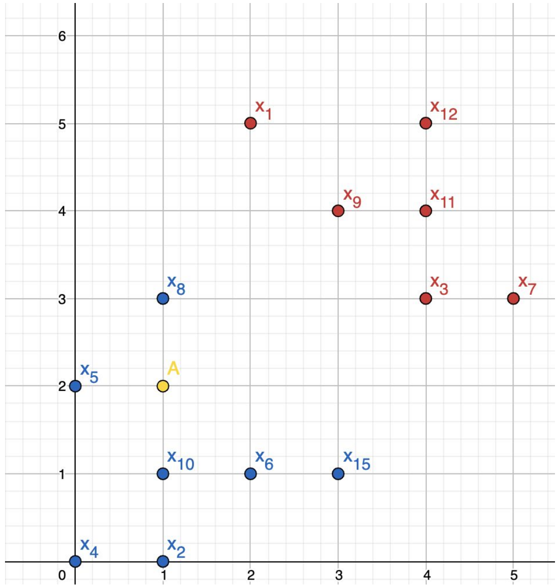
```
[0 1]

Process finished with exit code 0
```

Choosing a K value



Consider point A with K=4



What about K=5, 8, 13?

Flaws?

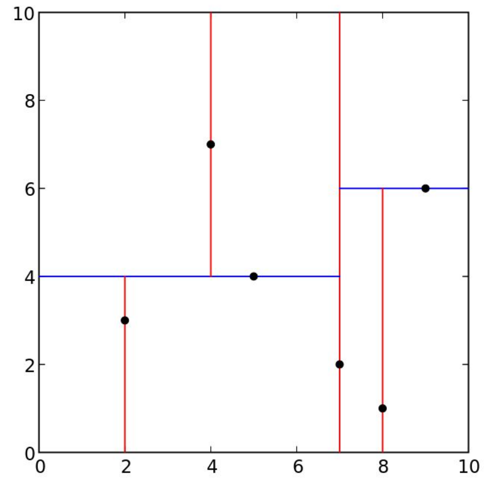
- We have to look at *every* datapoint in the training set, and compute the distance to the point we want to classify. This is highly inefficient!
- Euclidean distance, while intuitive, is not the only distance function out there. Others may work better!
- Not all features are equally important--but KNN treats them as equally important!
- Units can get in the way! We might need to normalize.

Solutions: KD Trees

Problem 1: We have to look at *every* datapoint in the training set, and compute the distance to our point we want to classify. This is highly inefficient!

Intuitive Idea: split up each dimension of the data based on the midpoint, recursively. Then choose nearest neighbors only in the region your point is in.

Be careful though, if the dimensionality of your data is too large (usually >=5), the dataset is partitioned too many times

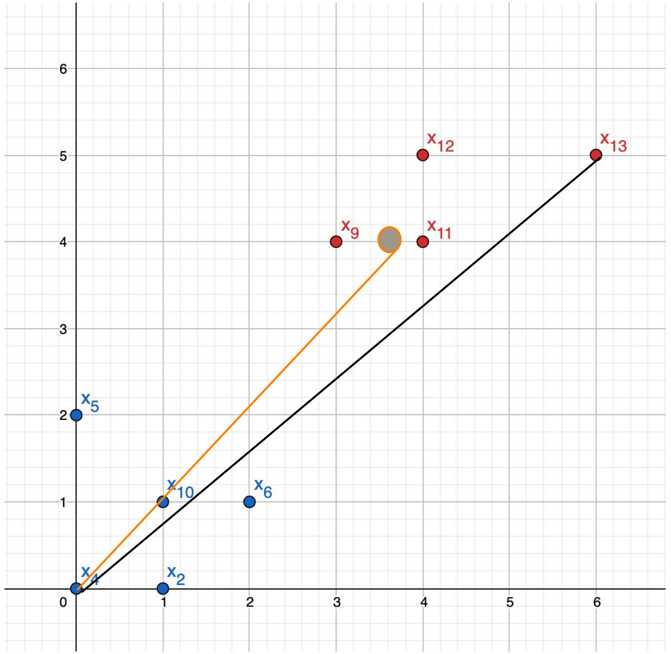


k-d tree decomposition for the point set (2,3), (5,4), (9,6), (4,7), (8,1), (7,2).

Image Source

Solutions: Ball Trees

Problem 1: We have to look at *every* datapoint in the training set, and compute the distance to our point we want to classify. This is highly inefficient!



Intuitive Idea: Find the point furthest from your datapoint, then find the point furthest from *that* datapoint. Use these 2 points to partition the space

Create a hypersphere on each side

Choosing a nearest neighbor algorithm

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use **`BallTree`**
- 'kd_tree' will use **`KDTree`**
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to **`fit`** method.

```
knn = KNeighborsClassifier(n_neighbors=2, algorithm='kd_tree')
knn.fit(X, y)
```

Solutions: Other distances

Problem 2: Euclidean distance, while intuitive, is not the only distance function out there. Others may work better!

Minkowski Distance

$$dist(q,r) = \left(\sum_{i=1}^{n} (q_i - r_i)^p \right)^{\frac{1}{p}}$$

```
knn = KNeighborsClassifier(n_neighbors=2, metric='minkowski', p=4)
knn.fit(X, y)
```

Manhattan Distance

$$dist(q,r) = \sum_{i=1}^{n} |q_i - r_i|$$

```
knn = KNeighborsClassifier(n_neighbors=2, metric='manhattan')
knn.fit(X, y)
```

Full list available on the the sklearn website

Solutions: Removing features & normalizing

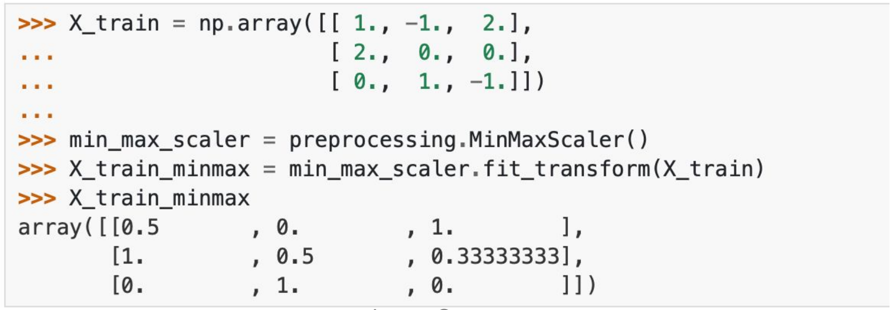
Problem 3: Not all features are equally important--but KNN treats them as equally important!

Use domain knowledge to remove unnecessary features that have little to no importance

Problem 4: Units can get in the way!

Normalizing condenses your data points to range between two values, usually 0 and 1

from sklearn.preprocessing import MinMaxScaler



```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[0.5       , 0.        , 1.        ],
       [1.        , 0.5       , 0.33333333],
       [0.        , 1.        , 0.        ]])
```

Image Source

References

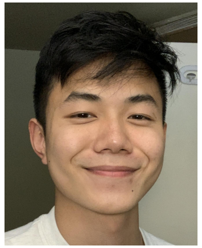
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/neighbors.html#classification
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html#sklearn.neighbors.DistanceMetric

Contributors

Addison Weatherhead

Mustafa Imam

Isha Sharma


Kevin Zhu

Gilles Fayad