K-Nearest Neighbor

K_Nearest Neighbor Algorithm

- KNN can be used for **classification** - the output is a class membership (predicts a class — a discrete value).
- It can also be used for **regression** - output is the value for the object (predicts continuous values). This value is the average (or median) of the values of its k nearest neighbors.
- KNN is non-parametric, which means that the algorithm does not make assumptions about the underlying distributions of the data.
- Building the model consists only of storing the training dataset
- To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its "nearest neighbors."

Algorithm in 3 Steps

Lets see how this algorithm works and implement it.

Step 1: Calculate Distance.

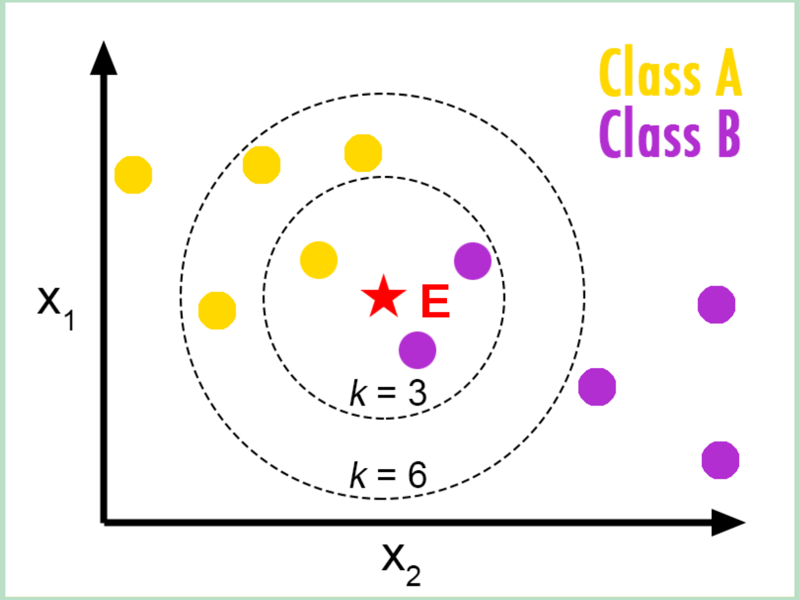
Step 2: Get Nearest Neighbors.

Step 3: Make Predictions.

k-Nearest-Neighbor Classifier

To determine the class of a new example E:

- Calculate the distance between E and all examples in the training set
- Select K-nearest examples to E in the training set
- Assign E to the most common class among its K-nearest neighbors



Closeness Between Neighbors

Each example is represented with a set of numerical attributes

Jay:
Age=35
Income=35K
No. of credit cards=3

Rina:
Age=22
Income=50K
No. of credit cards=2

- “Closeness” is defined in terms of the Euclidean distance between two examples

- The Euclidean distance between X=(x_1, x_2, x_3,…x_n) and Y =(y_1,y_2, y_3,…y_n) is defined as:

$$D(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Distance (Jay,Rina) = $\sqrt{(35-22)^2 + (35-50)^2 + (3-2)^2} = 19.87$

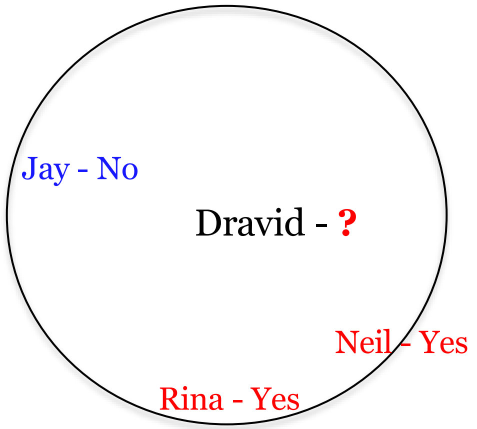
K_Nearest Neighbors: Example

Customer	Age	Income	No. credit cards	Response
Jay	35	35K	3	No
Rina	22	50K	2	Yes
Hema	63	200K	1	No
Tommy	59	170K	1	No
Neil	25	40K	4	Yes
Dravid	37	50K	2	?

K_Nearest Neighbors: Example

Customer	Age	Income	No. Of credit cards	Response	Distance from Dravid
Jay	35	35K	3	No	$\sqrt{(35-37)^2+(35-50)^2+(3-2)^2}$ = 15.16
Rina	22	50K	2	Yes	15
Hema	63	200K	1	No	152.23
Tommy	59	170K	1	No	122
Neil	25	40K	4	Yes	15.74
Dravid	37	50K	2	?	

When K = 3



Dravid's response is predicted to be Yes

K_Nearest Neighbors

Distance (Hema, Dravid)= $\sqrt{(63-37)^2 + \mathbf{(200-50)^2} + (1-2)^2} = 152.23$

- One thing we need to be mindful: Distance between neighbors could be dominated by some attributes with relatively large-magnitude numbers (e.g., income in our example)
- **Important to normalize some features**
(e.g., map numbers to common scale between 0-1)

Example: Income

Highest income = 200K

Hema's income is normalized to 200/200 = 1

Dravid income is normalized to 50/200 = 0.25

etc.

K_Nearest Neighbors

Normalization of Variables

Customer	Age	Income	No. credit cards	Response	Distance from Dravid
Jay	35/63= 0.56	35/200= 0.175	3/4= 0.75	No	$\sqrt{(0.56-0.59)^2+(0.175-0.25)^2+(0.75-0.5)^2} = 0.26$
Rina	22/63= 0.35	50/200= 0.25	2/4= 0.5	Yes	0.24
Hema	63/63= 1	200/200= 1	1/4= 0.25	No	0.89
Tommy	59/63= 0.94	170/200= 0.85	1/4= 0.25	No	0.74
Neil	25/63= 0.40	40/200= 0.20	4/4= 1	Yes	0.54
Dravid	37/63= 0. 59	50/200= 0.25	2/4= 0.5	?	

When K = 3, Dravid's response is predicted to be Yes

K-Nearest Neighbor

- Distance works naturally with numerical attributes, like what we did in previous slides

- What if we have nominal attributes?

Example: Married

Customer	Married	Income	No. credit cards	Response
Jay	Yes	35K	3	No
Rina	No	50K	2	Yes
Hema	No	200K	1	No
Tommy	Yes	170K	1	No
Neil	No	40K	4	Yes
Dravid	Yes	50K	2	?

Non-Numeric Data

- Feature values are not always numbers
- Examples:
  - Boolean values: Yes or no, presence or absence of an attribute
  - Categories: Colors, educational attainment, gender
- How do these values factor into the computation of distance?

Dealing with Non-Neumeric Data

- Boolean values => convert to 0 or 1
  - Applies to yes-no/presence-absence kind of binary attributes/features

- Non-binary attributes
  - Use natural progression when applicable; e.g., educational attainment: PS, SS, JC, UG, PG => 1,2,3,4,5

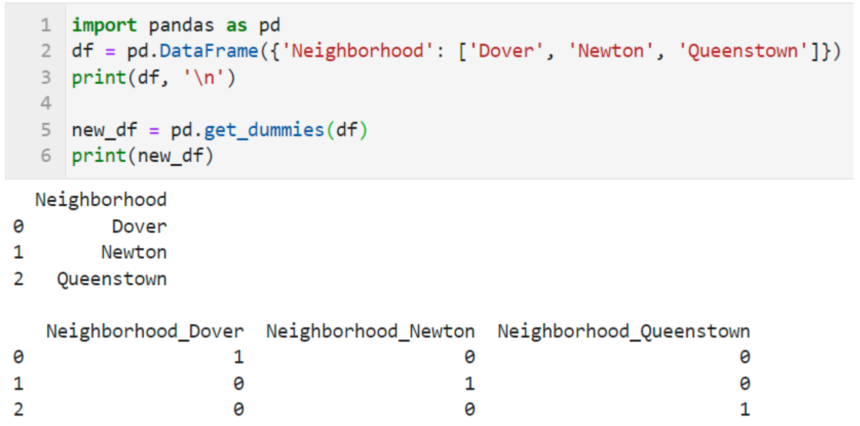
- A better solution: the One-Hot Encoding
  - One-hot encoding is useful for replacing categorical data with simple numeric data (a bunch of 0s and one single 1) that the machine learning model can easily understand. e.g.,

Original

Neighborhood
Dover
Newton
Queenstown

One-Hot Encoding

Is_Dover	Is_Newton	Is_Queenstown
1	0	0
0	1	0
0	0	1



```python
import pandas as pd
df = pd.DataFrame({'Neighborhood': ['Dover', 'Newton', 'Queenstown']})
print(df, '\n')

new_df = pd.get_dummies(df)
print(new_df)
```

```
  Neighborhood
0        Dover
1       Newton
2   Queenstown

   Neighborhood_Dover  Neighborhood_Newton  Neighborhood_Queenstown
0                   1                    0                        0
1                   0                    1                        0
2                   0                    0                        1
```

Distance measures

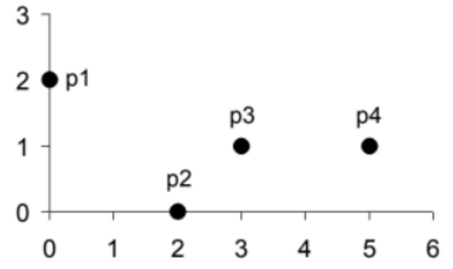
- How to determine similarity between data points?
  - Using various distance metrics

Distance measure

Euclidean Distance

$$dist = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}$$

– Where n is the number of dimensions (attributes) and pk and qk are, respectively, the k^{th} attributes (components) or data objects p and q.



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance Matrix

Distance measure

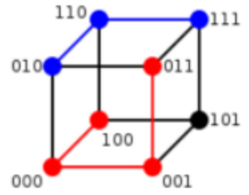
Minkowski Distance

- Minkowski Distance is a generalization of Euclidean Distance

$$dist = \left(\sum_{k=1}^{n} |p_k - q_k|^r \right)^{\frac{1}{r}}$$

  – Where r is a parameter, n is the number of dimensions (attributes) and p_k and q_k are, respectively, the k^{th} attributes (components) or data objects p and q.

- $r = 1$. City block (Manhattan, taxicab, L_1 norm) distance.
  – A common example of this is the Hamming distance, which is just the number of bits that are different between two binary vectors



- $r = 2$. Euclidean distance

- $r \rightarrow \infty$. "supremum" (L_{max} norm, L_∞ norm) distance.
  – This is the maximum difference between any component of the vectors

- Do not confuse r with n, i.e., all these distances are defined for any number of dimensions.

point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

L_∞	p1	p2	p3	p4
p1	0	2	3	5
p2	2	0	1	3
p3	3	1	0	2
p4	5	3	2	0

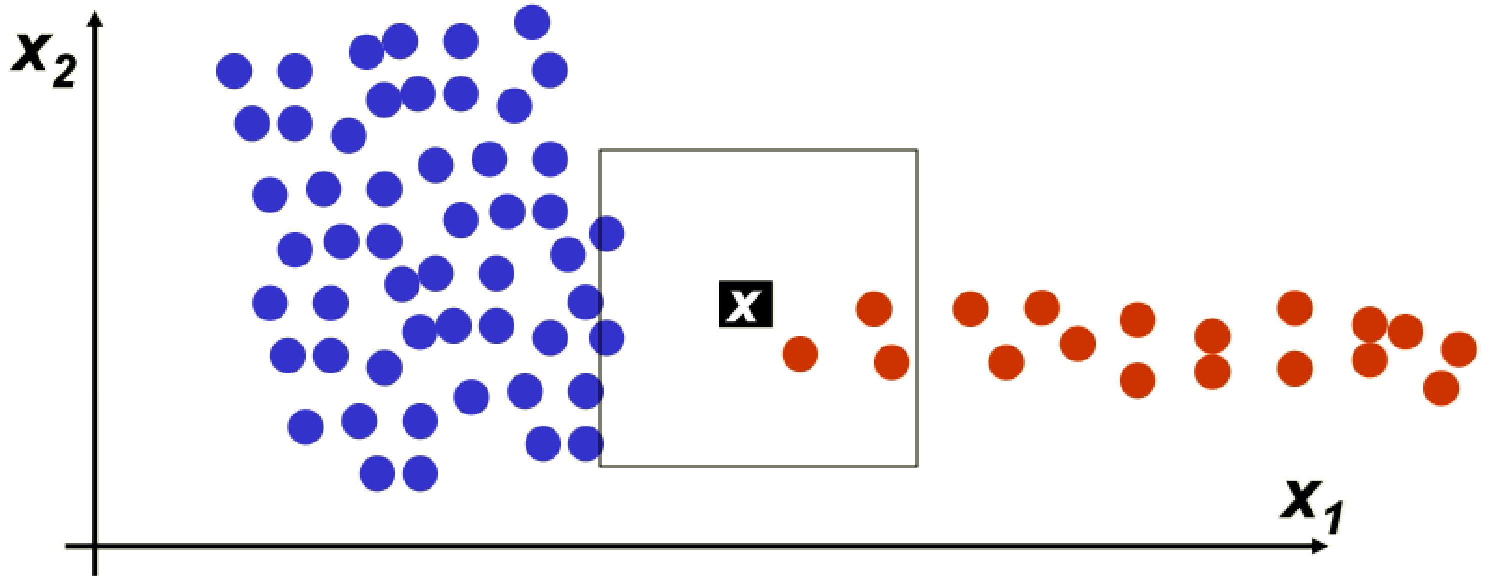
Distance Matrix

k-NN Variations

- Value of k
  - Larger k increases confidence in prediction
  - Note that if k is too large, decision may be skewed

- Weighted evaluation of nearest neighbors
  - Plain majority may or may not unfairly skew decision
  - A possible solution is to revise algorithm so that closer neighbors have greater “vote weight”

How to Choose “K"?



- For k = 1, …,5 point x gets classified correctly
  - red class
- For larger k classification of x is wrong
  - blue class

How to Choose "K”?

- Selecting the value of K in K-nearest neighbor is the most critical problem.
- A small value of K means that noise will have a higher influence on the result i.e., the probability of overfitting is very high.
- A large value of K makes it computationally expensive and defeats the basic idea behind KNN.
- A simple approach to select K is $K = \sqrt{n}$, where n is the number of data points in training data
- To be on the safe side, it also depends on individual cases, best process is to run through each possible value of K and test our result

KNN algorithm Pseudo Code

- Let (X_i, C_i) be data points in training dataset, where $i = 1,2, \cdots, n$. X_i denotes feature values & C_i denotes labels for each X_i

- Assuming the number of classes as c, $C_i \in \{1,2,3, \cdots, c\}$

- Let x be a new point for which label is not known

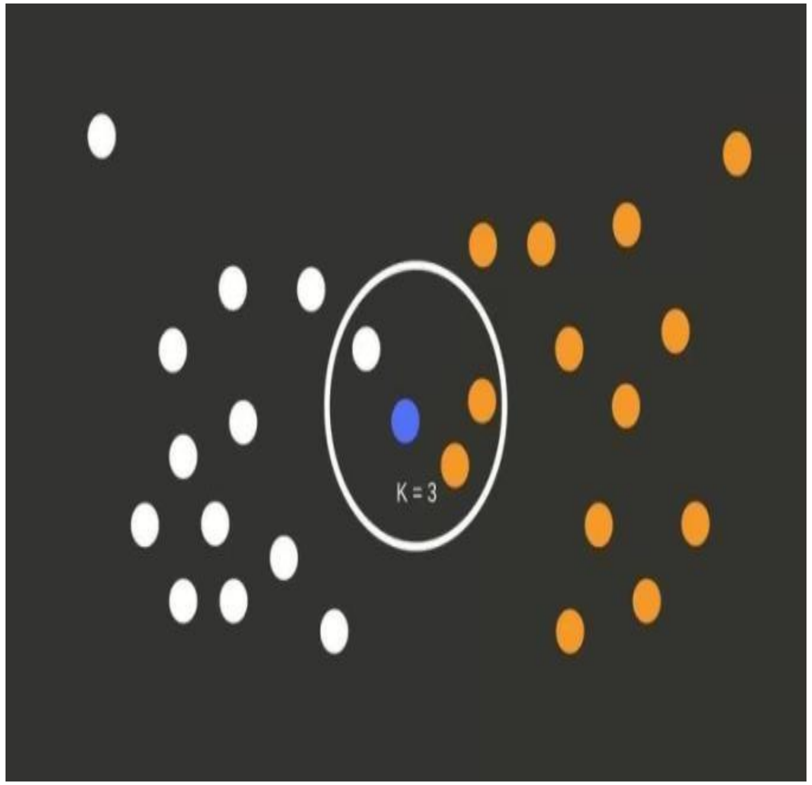
- We would like to find the label class of x using KNN algo.

KNN algorithm Pseudo Code

- First, calculate $d(x, X_i)$ $i = 1, 2, \cdots , n$; where d denotes the Euclidean distance between the points.
- Next step is to arrange all the distances in increasing order.
- Assuming a positive value of k and filtering k smallest values from the sorted list.
- Now, we have k top distances.
- Let k_i denotes no. of points belonging to the i^{th} class among k points.
- If $k_i > k_j$ for all $i \neq j$ then put x in class i

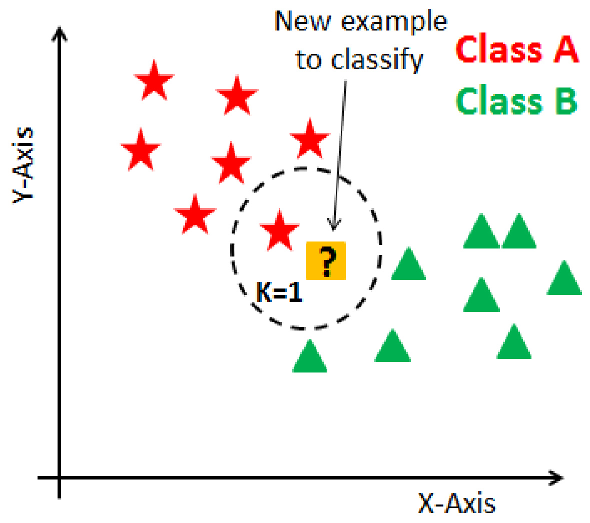
KNN algorithm: Example

- Let’s consider the image shown here, where we have two different target classes, white and orange points.
- We have total 26 training samples.
- Now we would like to predict the target class for the blue point
- Considering K = 3, we need to calculate the similarity distance using similarity measures, like Euclidean distance.
- The smaller the Euclidean distance, the closer it is.
- In the image, for K = 3 (i.e., 3 nearest neighbor points), there are 2 orange points and 1 white point. Thus, label the blue point as orange class.



Nearest Neighbor Algorithm

- Nearest neighbor is a special case of k-nearest neighbor class.
- In this case, k value is 1 (k = 1).
- In this case, new data point target class will be simply assigned to the class of the 1st closest neighbor

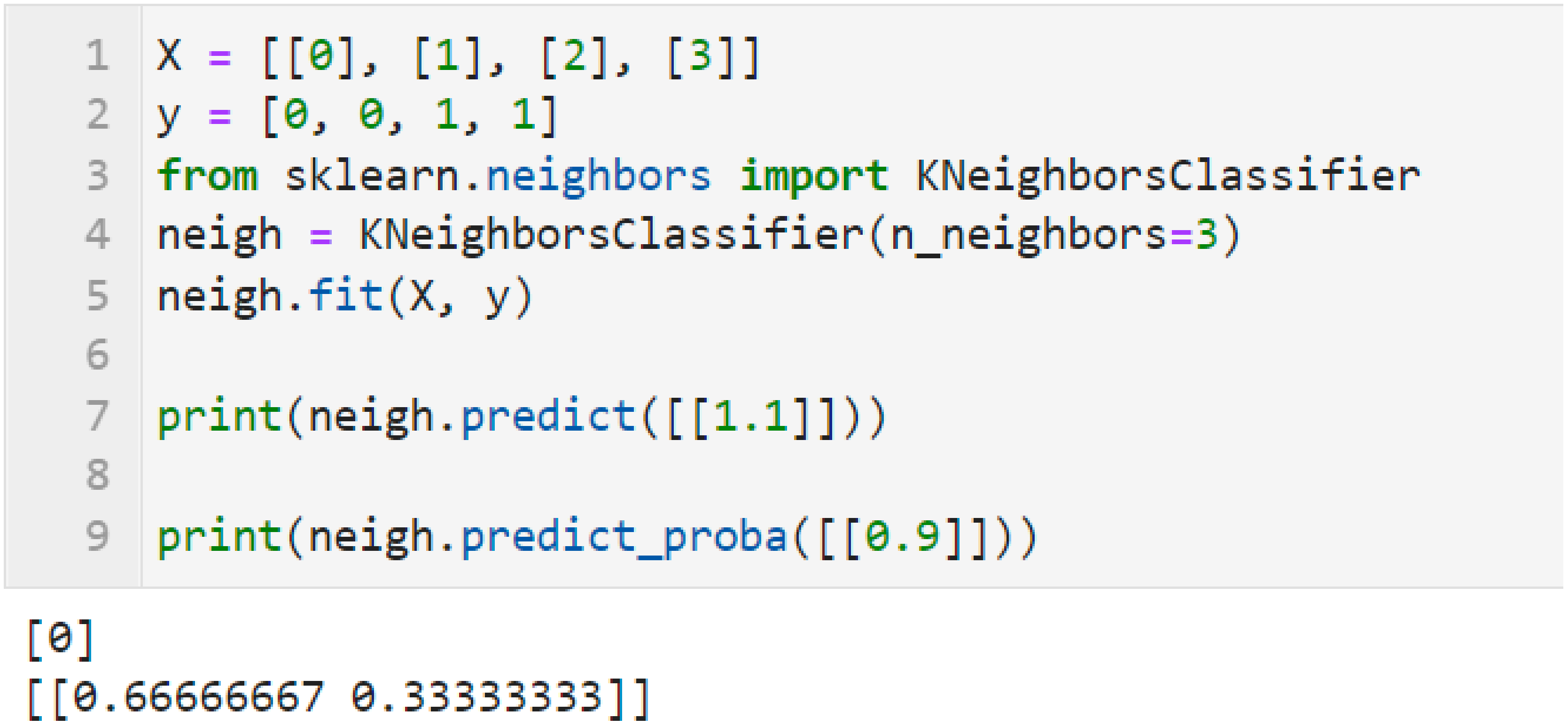


Advantages and Disadvantages

- **Advantages**
  - Makes no assumptions about distributions of classes in feature space
  - Don’t need any prior knowledge about the structure of data in the training set
  - No retraining is required if the new training pattern is added to the existing training set
  - Can work for multi-classes simultaneously
  - Easy to implement and understand
  - KNN executes quickly for small training data sets

- **Disadvantages**
  - Fixing the optimal value of K is a challenge
  - Will not be effective when the class distributions overlap
  - Does not output any models. Calculates distances for every new point (lazy learner)
  - For every test data, the distance should be computed between test data and all the training data. Thus, a lot of time may be needed for the testing

Simple sample code in scikit-learn KNeighborsClassifier



```python
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)

print(neigh.predict([[1.1]]))

print(neigh.predict_proba([[0.9]]))
```

```
[0]
[[0.66666667 0.33333333]]
```

DigiPen
INSTITUTE OF TECHNOLOGY
SINGAPORE