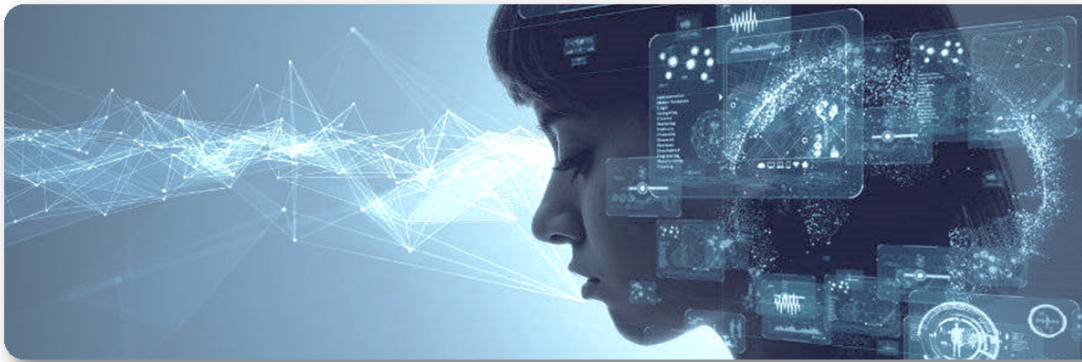


# COMP 2211 Exploring Artificial Intelligence

## K-Nearest Neighbor Classifier

Dr. Desmond Tsoi & Dr. Huiru Xiao

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

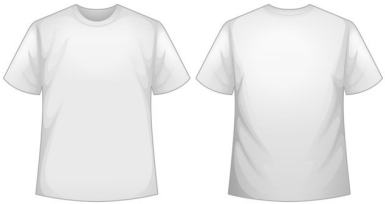
# Problem

- Suppose we have the height, weight and T-shirt size of some customers as follows:

| Height (in cm) | 158 | 158 | 158 | 160 | 160 | 163 | 163 | 160 | 163 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 58 | 59 | 63 | 59 | 60 | 60 | 61 | 64 | 64 |
| T-shirt Size | M | M | M | M | M | M | M | L | L |

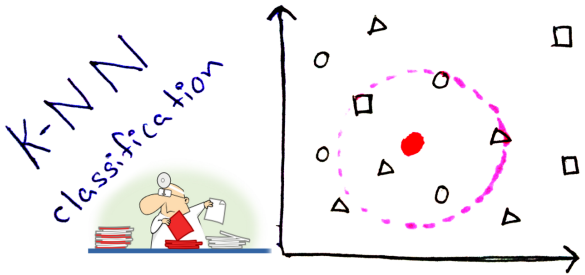
| Height (in cm) | 165 | 165 | 165 | 168 | 168 | 168 | 170 | 170 | 170 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 61 | 62 | 65 | 62 | 63 | 66 | 63 | 64 | 68 |
| T-shirt Size | L | L | L | L | L | L | L | L | L |

- Can we predict the T-shirt size of a new customer given only the height and weight information we have?



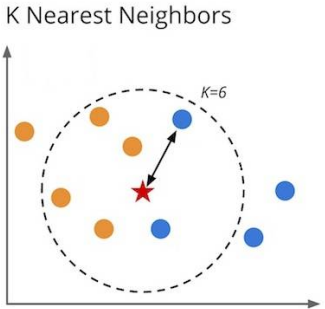
# K-Nearest Neighbor

- K-Nearest Neighbor (aka KNN) is one of the most used machine learning algorithms due to its simplicity.
- KNN is a lazy learning (why lazy?), non-parametric algorithm, as it does not make any assumptions on the data being studied (e.g. the distribution of the data).
- It uses data with several classes to predict the classification of the new sample point. In other words, it captures information of all training data and classifies the new data based on similarity.



# Steps

1. Prepare training data and test data.
2. Select a value K.
3. Determine which distance function is to be used.
4. Compute the distance of the new data to its n training samples.
5. Sort the distances obtained and take the K-nearest data samples.
6. Assign the test sample to the class based on the majority vote of its K nearest neighbors.



# Step 1: Prepare Training Data and Test Data & Step 2: Select a Value K

- Training data

| Height (in cm) | 158 | 158 | 158 | 160 | 160 | 163 | 163 | 160 | 163 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 58 | 59 | 63 | 59 | 60 | 60 | 61 | 64 | 64 |
| T-shirt Size | M | M | M | M | M | M | M | L | L |

| Height (in cm) | 165 | 165 | 165 | 168 | 168 | 168 | 170 | 170 | 170 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 61 | 62 | 65 | 62 | 63 | 66 | 63 | 64 | 68 |
| T-shirt Size | L | L | L | L | L | L | L | L | L |

- Testing data

Height: 161cm and weight: 61kg

- Let's pick $K = 5$

# Step 3: Determine Which Distance Function to Use

- Let $\mathbf{x}^{\mathbf{Train}}$ be a training data, which has $n$ attributes:

$$\mathbf{x}^{\mathbf{Train}} = \{x_1^{Train}, x_2^{Train}, \ldots, x_n^{Train}\}$$

  and $\mathbf{x}^{\mathbf{Test}}$ be a testing data, which also has $n$ attributes:

$$\mathbf{x}^{\mathbf{Test}} = \{x_1^{Test}, x_2^{Test}, \ldots, x_n^{Test}\}$$

- Assume Euclidean distance is used

$$distance(\mathbf{x}^{\mathbf{Train}}, \mathbf{x}^{\mathbf{Test}}) = \sqrt{\sum_{i=1}^{n} (x_i^{Train} - x_i^{Test})^2}$$

# Step 4: Compute the Distance of the New Data to Its n Training Samples

Testing data:
Height: 161cm and weight: 61kg

| Height (in cm) | 158 | 158 | 158 | 160 | 160 | 163 | 163 | 160 | 163 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 58 | 59 | 63 | 59 | 60 | 60 | 61 | 64 | 64 |
| T-shirt Size | M | M | M | M | M | M | M | L | L |
| Distance between the training data and testing data | 4.24 | 3.61 | 3.61 | 2.24 | 1.41 | 2.24 | 2.00 | 3.16 | 3.61 |

| Height (in cm) | 165 | 165 | 165 | 168 | 168 | 168 | 170 | 170 | 170 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 61 | 62 | 65 | 62 | 63 | 66 | 63 | 64 | 68 |
| T-shirt Size | L | L | L | L | L | L | L | L | L |
| Distance between the training data and testing data | 4.00 | 4.12 | 5.66 | 7.07 | 7.28 | 8.60 | 9.22 | 9.49 | 11.40 |

# Step 5: Sort the Distances Obtained and Take the K-nearest Data Samples

$$K = 5$$

| Height (in cm) | 160 | 163 | 160 | 163 | 160 | 158 | 158 | 163 | 165 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 60 | 61 | 59 | 60 | 64 | 59 | 63 | 64 | 61 |
| T-shirt Size | M | M | M | M | L | M | M | L | L |
| Distance between the training data and testing data | 1.41 | 2.00 | 2.24 | 2.24 | 3.16 | 3.61 | 3.61 | 3.61 | 4.00 |

| Height (in cm) | 165 | 158 | 165 | 168 | 168 | 168 | 170 | 170 | 170 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 62 | 58 | 65 | 62 | 63 | 66 | 63 | 64 | 68 |
| T-shirt Size | L | M | L | L | L | L | L | L | L |
| Distance between the training data and testing data | 4.12 | 4.24 | 5.66 | 7.07 | 7.28 | 8.60 | 9.22 | 9.49 | 11.40 |

# Step 6: Assign the Test Sample to the Class Based On The Majority Vote of Its K Nearest Neighbors

| Height (in cm) | 160 | 163 | 160 | 163 | 160 |
|---|---|---|---|---|---|
| Weight (in kg) | 60 | 61 | 59 | 60 | 64 |
| T-shirt Size | M | M | M | M | L |
| Distance between the training data and testing data | 1.41 | 2.00 | 2.24 | 2.24 | 3.16 |

- Among the 5 customers, 4 of them had "Medium T-shirt Sizes" and 1 had "Large T-shirt Size".
- So, we classify the test data to "Medium T-shirt", i.e., the best guess for the customer with height = 161cm and weight = 61kg is "Medium T-shirt Size".

# KNN Implementation using Scikit-Learn

```
from sklearn.preprocessing import LabelEncoder     # Import LabelEncoder function
from sklearn.neighbors import KNeighborsClassifier # Import KNeighborsClassifier function
import numpy as np  # Import NumPy

# Assign features and label variables
height = np.array([158, 158, 158, 160, 160, 163, 163, 160, 163, 165, 165, 165, 168, 168, 168, 170, 170, 170])
weight = np.array([58, 59, 63, 59, 60, 60, 61, 64, 64, 61, 62, 65, 62, 63, 66, 63, 64, 68])
size = np.array(['M','M','M','M','M','M','M','L','L','L','L','L','L','L','L','L','L','L'])

# Create LabelEncoder that used to convert string labels to numbers
encoder = LabelEncoder()
# Convert string labels into numbers
label = encoder.fit_transform(size)
# Combine height and weight into single list of tuples
features = list(zip(height,weight))

# Create a KNN classifier and set K=5
model = KNeighborsClassifier(n_neighbors=5)
# Train the model using the training sets
model.fit(features, label)

# Predict output
predicted = model.predict([[161,61]]) # height = 161, weight = 61

# Convert number to string label
predicted = encoder.inverse_transform(predicted)
print(predicted)
```

# Standardization

- When the training data are measured in different units, it is important to standardize variables (i.e. the attributes) before calculating distance.
- For example, if one attribute is based on height in cm, and the other is based on weight in kg, then height will influence more on the distance calculation.
- In order to make the attributes comparable, we need to standardize them which can be done by the following method:

$$X_{new} = \frac{X - mean}{standard\ deviation}$$

where $X$ is the attribute value, $X_{new}$ is the standardized attribute value, *mean* is the mean attribute value of all training data, and *standard deviation* is the standard deviation of the attribute value of all the training data.

- Mean of height = 164, Standard deviation of height = 4.33
- Mean of Weight = 62.33, Standard deviation of weight = 2.63
- After standardization using $X_{new} = \frac{X - mean}{standard\ deviation}$, the 5th closest value got changed as height was dominating earlier before standardization.

Testing data: Height: 161cm and weight: 61kg

| Height (in cm) | 160 | 163 | 160 | 163 | 160 | 158 | 158 | 163 | 165 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 60 | 61 | 59 | 60 | 64 | 59 | 63 | 64 | 61 |
| T-shirt Size | M | M | M | M | L | M | M | L | L |
| Distance between the training data and testing data | 1.41 | 2.00 | 2.24 | 2.24 | 3.16 | 3.61 | 3.61 | 3.61 | 4.00 |
| Distance between the training data and testing data (standardization) | 0.44 | 0.46 | 0.60 | 0.79 | 1.16 | 1.03 | 1.03 | 1.23 | 0.92 |

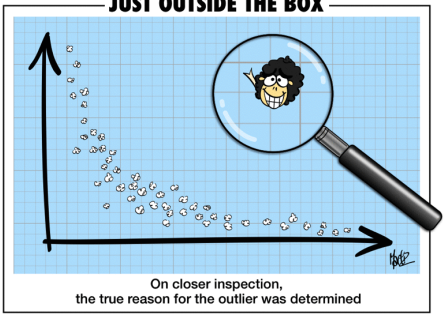
| Height (in cm) | 165 | 158 | 165 | 168 | 168 | 168 | 170 | 170 | 170 |
|---|---|---|---|---|---|---|---|---|---|
| Weight (in kg) | 62 | 58 | 65 | 62 | 63 | 66 | 63 | 64 | 68 |
| T-shirt Size | L | M | L | L | L | L | L | L | L |
| Distance between the training data and testing data | 4.12 | 4.24 | 5.66 | 7.07 | 7.28 | 8.60 | 9.22 | 9.49 | 11.40 |
| Distance between the training data and testing data (standardization) | 1.00 | 1.33 | 1.78 | 1.66 | 1.79 | 2.49 | 2.22 | 2.37 | 3.37 |

# KNN Implementation using Scikit-Learn (Standardization)

```
from sklearn.preprocessing import LabelEncoder     # Import LabelEncoder function
from sklearn.neighbors import KNeighborsClassifier # Import KNeighborsClassifier function
import numpy as np                                 # Import NumPy

# Assign features and label variables
height = np.array([158, 158, 158, 160, 160, 163, 163, 160, 163, 165, 165, 165, 168, 168, 168, 170, 170, 170])
h_mean = np.mean(height); h_std = np.std(height,ddof=1) # ddof=1 is to make the divisor to n-1, i.e., sample mean
height = (height - h_mean)/h_std
weight = np.array([58, 59, 63, 59, 60, 60, 61, 64, 64, 61, 62, 65, 62, 63, 66, 63, 64, 68])
w_mean = np.mean(weight); w_std = np.std(weight,ddof=1) # ddof=1 is to make the divisor to n-1, i.e., sample mean
weight = (weight - w_mean)/w_std
size = ['M','M','M','M','M','M','M','L','L','L','L','L','L','L','L','L','L','L']

# Create LabelEncoder that used to convert string labels to numbers
encoder = LabelEncoder()
# Convert string labels into numbers
label = encoder.fit_transform(size)
# Combine height and weight into single list of tuples
features = list(zip(height,weight))
# Create a KNN classifier and set K=5
model = KNeighborsClassifier(n_neighbors=5)
# Train the model using the training sets
model.fit(features, label)

# Predict output
predicted = model.predict([[(161-h_mean)/h_std,(61-w_mean)/w_std]]) # height = 161, weight = 61
# Convert number to string label
predicted = encoder.inverse_transform(predicted)
print(predicted)
```

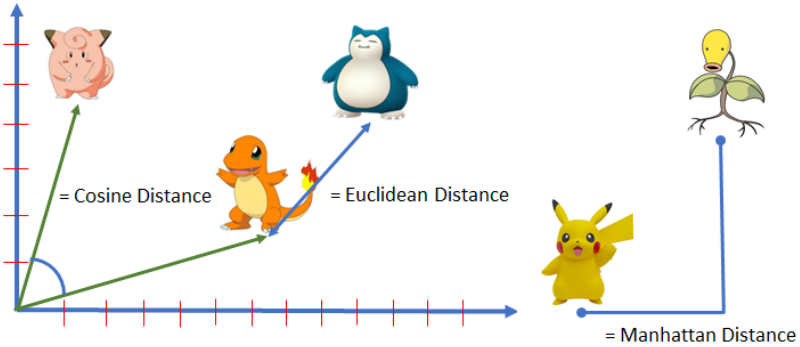
# Outlier

- A Low K-value is sensitive to outliers and a higher K-value is more resilient to outliers as it considers more voters to decide prediction.



# Distance Functions

- There are a lot of different distance functions available.
- Some common ones are:
  - Euclidean distance (The one we used earlier)
  - Manhattan distance (aka City block distance)
  - Cosine distance
  - Hamming distance



# Manhattan Distance

$$distance(\mathbf{x}^{\mathbf{Train}}, \mathbf{x}^{\mathbf{Test}}) = \sum_{i=1}^{n} |x_i^{Train} - x_i^{Test}|$$

where $|\cdot|$ denote absolute value.

- Example:
  Training data: (Height, Weight) = (160, 60), Testing data: (Height, Weight) = (163, 61)

$$distance = |160 - 163| + |60 - 61| = 4$$

**Question**

When Manhattan distance is preferred over Euclidean distance?

# Cosine Distance

$$cos\theta = \frac{\sum_{i=1}^{n}(x_i^{Train} \times x_i^{Test})}{\sqrt{\sum_{i=1}^{n}(x_i^{Train})^2}\sqrt{\sum_{i=1}^{n}(x_i^{Test})^2}}$$

$$Distance = 1 - cos\theta$$

- Example:
  Training data: (Height, Weight) = (160, 60), Testing data: (Height, Weight) = (163, 61)

$$cos\theta = \frac{(160 \times 163) + (60 \times 61)}{\sqrt{160^2 + 60^2}\sqrt{163^2 + 61^2}} = 0.99999977$$

$$Distance = 1 - 0.99999977 = 2.26 \times 10^{-7}$$

**Remark**

Using Cosine distance, we get values between 0 and 2, where 0 means the training data and test data are 100% similar to each other, 2 means they are absolutely different, and values between 0 and 2 mean varying degrees of dissimilarity.

# Hamming Distance

- Hamming distance is a way for comparing two binary data strings.
- While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different.
- Example: Suppose there are two strings 1101 1001 and 1001 1101, then the hamming distance between the two strings is 2.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| | $+1$ | | | | $+1$ | | |

Hamming distance is for categorical data.

You will encounter bit strings when you one-hot encode categorical columns of data. More details will be given in a later topic.

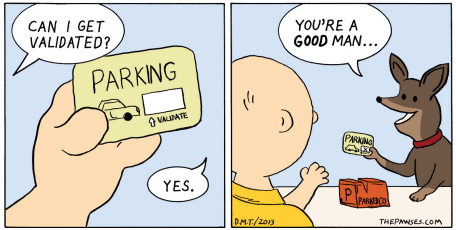
# Good Value of K

- To break a tie,
  - Decrease K by 1 until we have broken the tie.
  - Put more weight for the nearest points than the farther points.
- Should not be too small or too large.

Rule of thumb: Set $K = \sqrt{N}$, where $N$ is the number of training examples in the dataset. (An explanation will be given in a supplementary notes.)

# Using Cross-validation to Estimate K

- Suppose we select K using d-fold cross validation
  1. Split the training data into d groups, or folds, of approximately equal size.
  2. Hold the first group. This is called the validation set.
  3. Train your classifier on the remaining data.
  4. For each value of K
     - Classify each data in the validation set, using its K-nearest neighbors in the training set.
     - Record the error.
  5. Repeat steps 1-4 for all d choices of the validation set.
  6. For each choice of K, find the average error across validation sets. Choose the value of K with the lowest error.



# Using Cross-validation to Estimate K

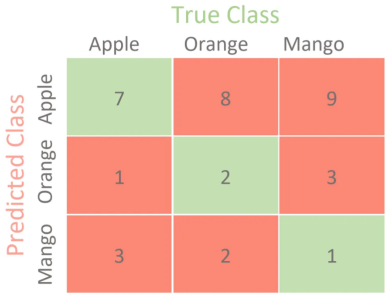
# Error Measurement for Regression Problems

- There are many ways to measure errors. Here are a few examples:
  - Mean Absolute Error (MAE)
  $$MAE = \frac{\sum_{i=1}^{m} |a_i - p_i|}{m}$$
  - Mean Square Error (MSE)
  $$MSE = \frac{\sum_{i=1}^{m} (a_i - p_i)^2}{m}$$
  - Mean Absolute Percentage Error (MAPE)
  $$MAPE = \frac{\sum_{i=1}^{m} \left| \frac{a_i - p_i}{a_i} \right|}{m}$$

  where $(a_1, a_2, \ldots, a_m)$ are the actual labels and $(p_1, p_2, \ldots, p_m)$ are the predicted labels.

# Error Measurement for Classification Problems

- When dealing with classification problems, the error can be calculated using a formula: 1 - F1-score, where F1-score is determined by considering true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class. These values are obtained from a confusion matrix.




- To find TP, TN, FP and FN for each individual class, let's take class Apple as an example.
  - TP means both actual and predicted values are apple. So, TP = 7.
  - TN means both the actual and predicted values are non-apple. So, TN = 1 + 2 + 2 + 3 = 8.
  - FP means the predicted value is apple, but the actual value is non-apple. So, 8 + 9 = 17.
  - FN means the predicted value is non-apple, but the actual value is apple. So, 1 + 3 = 4.
- F1-score = $\frac{2TP}{2TP+FP+FN} = \frac{2(7)}{2(7)+17+4} = 0.4$
- Error = 1 - F1-score = 1 - 0.4 = 0.6

# Pros and Cons

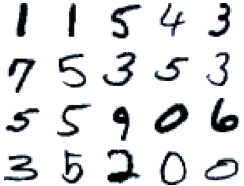
- Pros:
  - Easy to understand.
  - No assumptions about data.
  - Can be applied to both classification and regression.
    Note: For KNN regression, the resulting value can be calculated by taking the average of the output value for the top K nearest neighbors.
  - Works easily on multi-class problems.
- Cons:
  - Memory intensive/Computationally expensive.
  - Sensitive to scale of data.
  - Struggle when high number of attributes.
  - Does not work well with categorical features since it is difficult to find the distance between dimensions with categorical features.

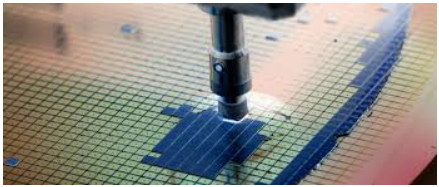
# Speed Up KNN

- Reduce the dimension of training data (e.g., using Principle-Component Analysis).
- Use good data structures to store the data, e.g., KD-tree.
- Parallelizing the distance computations.

# KNN Applications

- Handwritten character classification
- Fast content-based image retrieval
- Intrusion detection
- Fault detection for semiconductor manufacturing processes
- ...



# Practice Problem

- Suppose we have age and salary of some customers, as well as whether they purchased certain item or not.

| Age | 19 | 35 | 26 | 27 | 19 | 27 | 27 | 32 | 25 | 35 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 19 | 20 | 43 | 57 | 76 | 58 | 84 | 150 | 33 | 65 | 80 |
| Purchased | No | No | No | No | No | No | No | Yes | No | No | No |

| Age | 26 | 20 | 32 | 18 | 29 | 47 | 45 | 46 | 48 | 45 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (K) | 52 | 86 | 18 | 82 | 80 | 25 | 26 | 28 | 29 | 22 | 49 |
| Purchased | No | No | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |

- Can we predict whether a new customer will purchase the item given only the age and salary of the customer?

# Step 1: Prepare Training Data and Test Data & Step 2: Select a Value K

- Training data

| Age | 19 | 35 | 26 | 27 | 19 | 27 | 27 | 32 | 25 | 35 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 19 | 20 | 43 | 57 | 76 | 58 | 84 | 150 | 33 | 65 | 80 |
| Purchased | No | No | No | No | No | No | No | Yes | No | No | No |

| Age | 26 | 20 | 32 | 18 | 29 | 47 | 45 | 46 | 48 | 45 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (K) | 52 | 86 | 18 | 82 | 80 | 25 | 26 | 28 | 29 | 22 | 49 |
| Purchased | No | No | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |

- Testing data

Age: 20 and Salary (K): 38

- Let's pick $K = 5$

# Step 3: Determine Which Distance Function to Use

- Let $\mathbf{x}^{\mathbf{Train}}$ be a training data, which has n attributes:

$$\mathbf{x}^{\mathbf{Train}} = \{x_1^{Train}, x_2^{Train}, \ldots, x_n^{Train}\}$$

  and $\mathbf{x}^{\mathbf{Test}}$ be a testing data, which also has n attributes:

$$\mathbf{x}^{\mathbf{Test}} = \{x_1^{Test}, x_2^{Test}, \ldots, x_n^{Test}\}$$

- Assume Euclidean distance is used

$$distance(\mathbf{x}^{\mathbf{Train}}, \mathbf{x}^{\mathbf{Test}}) = \sqrt{\sum_{i=1}^{n} (x_i^{Train} - x_i^{Test})^2}$$

# Step 4: Compute the Distance of the New Data to Its n Training Samples

Testing data:
Age: 20 and Salary (in k): 38

| Age | 19 | 35 | 26 | 27 | 19 | 27 | 27 | 32 | 25 | 35 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 19 | 20 | 43 | 57 | 76 | 58 | 84 | 150 | 33 | 65 | 80 |
| Purchased | No | No | No | No | No | No | No | Yes | No | No | No |
| Distance between the training data and testing data | 19.03 | 23.43 | 7.81 | 20.25 | 38.01 | 21.19 | 46.53 | 112.64 | 7.07 | 30.89 | 42.44 |

| Age | 26 | 20 | 32 | 18 | 29 | 47 | 45 | 46 | 48 | 45 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 52 | 86 | 18 | 82 | 80 | 25 | 26 | 28 | 29 | 22 | 49 |
| Purchased | No | No | No | No | No | Yes | Yes | Yes | Yes | Yes | Yes |
| Distance between the training data and testing data | 15.23 | 48.00 | 23.32 | 44.05 | 42.95 | 29.97 | 27.73 | 27.86 | 29.41 | 29.68 | 29.15 |

# Step 5: Sort the Distances Obtained and Take the K-nearest Data Samples

$$K = 5$$

| Age | 25 | 26 | 26 | 19 | 27 | 27 | 32 | 35 | 45 | 46 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 33 | 43 | 52 | 19 | 57 | 58 | 18 | 20 | 26 | 28 | 49 |
| Purchased | No | No | No | No | No | No | No | No | Yes | Yes | Yes |
| Distance between the training data and testing data | 7.07 | 7.81 | 15.23 | 19.03 | 20.25 | 21.19 | 23.32 | 23.43 | 27.73 | 27.86 | 29.15 |

| Age | 48 | 45 | 47 | 35 | 19 | 26 | 29 | 18 | 27 | 20 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 29 | 22 | 25 | 65 | 76 | 80 | 80 | 82 | 84 | 86 | 150 |
| Purchased | Yes | Yes | No | No | No | No | No | No | No | No | Yes |
| Distance between the training data and testing data | 29.41 | 29.68 | 29.97 | 30.89 | 38.01 | 42.43 | 42.95 | 44.05 | 46.53 | 48.00 | 112.64 |

# Step 6: Assign the Test Sample to the Class Based On The Majority Vote of Its K Nearest Neighbors

| Age | 25 | 26 | 26 | 19 | 27 |
|---|---|---|---|---|---|
| Salary (k) | 33 | 43 | 52 | 19 | 57 |
| Purchased | No | No | No | No | No |
| Distance between the training data and testing data | 7.07 | 7.81 | 15.23 | 19.03 | 20.25 |

- Among the 5 customers, 5 of them did not purchase the item.
- So, we classify the test data to "No", i.e., the best guess for the customer with age $= 20$ and salary $= 38$k will not purchase the item.

# KNN Implementation using Scikit-Learn

```
from sklearn.preprocessing import LabelEncoder     # Import LabelEncoder function
from sklearn.neighbors import KNeighborsClassifier # Import KNeighborsClassifier function
import numpy as np  # Import NumPy

# Assign features and label variables
age = np.array([19, 35, 26, 27, 19, 27, 27, 32, 25, 35, 26, 26, 20, 32, 18, 29, 47, 45, 46, 48, 45, 47])
salary = np.array([19, 20, 43, 57, 76, 58, 84, 150, 33, 65, 80, 52, 86, 18, 82, 80, 25, 26, 28, 29, 22, 49])
purchased = np.array(['No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', '

# Create LabelEncoder that used to convert string labels to numbers
encoder = LabelEncoder()
# Convert string labels into numbers
label = encoder.fit_transform(purchased)
# Combine height and weight into single list of tuples
features = list(zip(age,salary))

# Create a KNN classifier and set K=5
model = KNeighborsClassifier(n_neighbors=5)
# Train the model using the training sets
model.fit(features, label)

# Predict output
predicted = model.predict([[20,38]]) # age = 20, salary = 38

# Convert number to string label
predicted = encoder.inverse_transform(predicted)
print(predicted)
```

- Mean of age = 31.86, Standard deviation of age = 10.18
- Mean of salary = 53.73, Standard deviation of salary = 32.46
- After standardization using $X_{new} = \frac{X - mean}{standard\ deviation}$, the 5 closest values remain unchanged.

Testing data: Age: 20 and salary(k): 38

| Age | 25 | 26 | 26 | 19 | 27 | 27 | 32 | 35 | 45 | 46 | 47 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 33 | 43 | 52 | 19 | 57 | 58 | 18 | 20 | 26 | 28 | 49 |
| Purchased | No | No | No | No | No | No | No | No | Yes | Yes | Yes |
| Distance between the training data and testing data | 7.07 | 7.81 | 15.23 | 19.03 | 20.25 | 21.19 | 23.32 | 23.43 | 27.73 | 27.86 | 29.15 |
| Distance between the training data and testing data (standardization) | 0.51 | 0.61 | 0.73 | 0.59 | 0.90 | 0.92 | 1.33 | 1.57 | 2.48 | 2.57 | 2.67 |

| Age | 48 | 45 | 47 | 35 | 19 | 26 | 29 | 18 | 27 | 20 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Salary (k) | 29 | 22 | 25 | 65 | 76 | 80 | 80 | 82 | 84 | 86 | 150 |
| Purchased | Yes | Yes | No | No | No | No | No | No | No | No | Yes |
| Distance between the training data and testing data | 29.41 | 29.68 | 29.97 | 30.89 | 38.01 | 42.43 | 42.95 | 44.05 | 46.53 | 48.00 | 112.64 |
| Distance between the training data and testing data (standardization) | 2.76 | 2.50 | 2.68 | 1.69 | 1.17 | 1.42 | 1.57 | 1.37 | 1.58 | 1.48 | 3.65 |

# KNN Implementation using Scikit-Learn (Standardization)

```
from sklearn.preprocessing import LabelEncoder     # Import LabelEncoder function
from sklearn.neighbors import KNeighborsClassifier # Import KNeighborsClassifier function
import numpy as np                                 # Import NumPy

# Assign features and label variables
age = np.array([19, 35, 26, 27, 19, 27, 27, 32, 25, 35, 26, 26, 20, 32, 18, 29, 47, 45, 46, 48, 45, 47])
a_mean = np.mean(age)
a_std = np.std(age,ddof=1) # ddof=1 is to make the divisor to n-1, i.e., sample mean
age = (age - a_mean)/a_std
salary = np.array([19, 20, 43, 57, 76, 58, 84, 150, 33, 65, 80, 52, 86, 18, 82, 80, 25, 26, 28, 29, 22, 49])
s_mean = np.mean(salary)
s_std = np.std(salary,ddof=1) # ddof=1 is to make the divisor to n-1, i.e., sample mean
salary = (salary - s_mean)/s_std
purchased = np.array(['No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No',
                      'No', 'No', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes'])

# Create LabelEncoder that used to convert string labels to numbers
encoder = LabelEncoder()
# Convert string labels into numbers
label = encoder.fit_transform(purchased)
# Combine height and weight into single list of tuples
features = list(zip(age,salary))
# Create a KNN classifier and set K=5
model = KNeighborsClassifier(n_neighbors=5)
# Train the model using the training sets
model.fit(features, label)

# Predict output
predicted = model.predict([[(20-a_mean)/a_std,(38-s_mean)/s_std]]) # age = 20, salary = 38
# Convert number to string label
predicted = encoder.inverse_transform(predicted)
print(predicted)
```

That's all!

Any questions?