Classification Algorithm

1) Measure distances b/w the test data & all training data
2) Find K neighborhood closest to the test data
3) Find all classes (y) of the neighbors
4) Find the most class y & determine this as the class y of the test data.
   (Majority Voting)

ISSUES
- What value of K should be chosen?
- How to measure the distance?
- If the distances b/w a test data & two points are the same within the margin of error, which point should we choose?
- The distance b/w a test data & all training data must be measured, & so should other test data, if we have a lot of data, it takes a long time to measure.
- Too many data features create a curse of dimensionality problem.

Euclidean distance:

$$\sqrt{(x_1-A_1)^2+(x_2-A_2)^2+\ldots+(x_n-A_n)^2}$$

Matrix Broadcasting

D0 D1 D2

```
x_exp = x[np.newaxis, :, :]    # expand D0 axis. (1, 5, 2)
t_exp = t[:, np.newaxis, :]    # expand D1 axis. (3, 1, 2)

# Calculating the distance between test and training data.
# Summing data on D2 axis. shape of dist = (3, 5) ← distance matrix (D)
dist = np.sqrt(np.sum(np.square(x_exp - t_exp), axis=2))
```

X

	f_1	f_2
1		
2		
3		
4		
5		

t

	f_1	f_2
1		
2		
3		

Distance Matrix

$$\rightarrow D = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} \end{bmatrix}$$

distance metrics used in KNN

- Manhattan $d = |x_1^{(1)}-x_1^{(2)}|+|x_2^{(1)}-x_2^{(2)}|+\ldots+|x_n^{(1)}-x_n^{(2)}|$ [sum of distances of each component]
- Euclidean $d = \sqrt{(x_1^{(1)}-x_1^{(2)})^2+(x_2^{(1)}-x_2^{(2)})^2+\ldots+(x_n^{(1)}-x_n^{(2)})^2}$ [shortest distance]
- Minkowski $d = \sqrt[p]{|x_1^{(1)}-x_2^{(2)}|^p+|x_2^{(1)}-x_2^{(2)}|^p+\ldots+|x_n^{(1)}-x_n^{(2)}|^p}$ [Focuses on the larger length of each component]
  - → generalization of Manhattan & Euclidean distance.

Data normalization

- When the data is as follows, x_1 and x_2 have different scales. The scale of x_2 is much larger than that of x_1.
- In this case, even if x_2 with a large scale changes slightly, the distance changes significantly. To prevent this phenomenon, the scale of features must be matched. This is called data standardization (or data normalization).
- The most commonly used methods are the Min-Max and Z-score methods.
- The Min-max method converts data into values between 0 and 1, and the Z-score method converts the distribution of data to mean = 0 and standard deviation = 1.
- When normalizing validation or test data, use the values calculated from the training data (min, max, mean, std).

	feature		class
i	x_1	x_2	y
0	0.16	10.01	0
1	-0.09	-20.03	0
2	0.06	-10.08	0
3	0.58	-13.1	2
4	0.46	-11.07	2
5	0.23	12.54	1
6	0.55	15.07	2
7	0.1	21.42	1
8	0.18	17.63	1
9	-0.03	-19.09	0
10	0.59	-23.14	2
11	0.32	29.06	2
12	-0.08	-18.03	0
13	0.27	16.52	1

- Min-Max normalization

$$x'^{(i)}_k = \frac{x_k^{(i)} - min(x_k)}{max(x_k) - min(x_k)}$$

- Z-score normalization

$$x'^{(i)}_k = \frac{x_k^{(i)} - mean(x_i)}{std(x_i)}$$

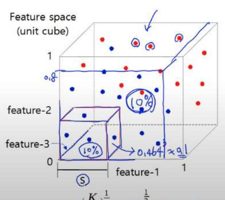
Decision boundary depends on the value of K. K↓ overfitting K↑ underfitting
↳ Hyperparameter

K = sqrt(N)

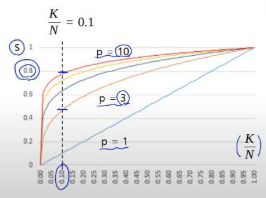
→ Since KNN is an algorithm that uses distances, performance can degrade when features increase. This is called COD.
KNN is particularly sensitive to this problem as it uses the concept of neighbors.

→ In higher dimensions distances are not reliable / misleading.

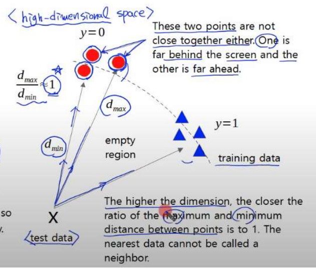
- Curse of Dimensionality
  - Since KNN is an algorithm that uses distance, performance can be degraded when features increase. This phenomenon is called the **curse of dimensionality**. Most machine learning algorithms also have this problem, but KNN, which uses the concept of neighbors, is particularly sensitive to this problem.
  - If the number of features in the dataset is 3 and the feature values range from 0 to 1, a cube with side length 0.464 is required to keep the number of training data around the test data 10% of the total data. p=3, K/N = 0.1 the length of a side s = $0.1^{1/3}$ = 0.464. (N: the number of training data, K: K value in KNN)
  - If there are 10 features (p = 10), it increases to s = $0.1^{1/10}$ = 0.8, so the area including K neighbors increases rapidly.



$s = (\frac{K}{N})^{\frac{1}{p}} = 0.1^{\frac{1}{3}} = 0.464$



As the dimension increases, s increases rapidly, so the area including K neighbors increases rapidly.



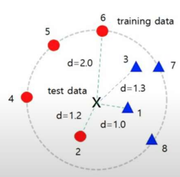
These two points are not close together either. One is far behind the screen and the other is far ahead.

The higher the dimension, the closer the ratio of the maximum and minimum distance between points is to 1. The nearest data cannot be called a neighbor.

* reference: Hastie, et, al. 2009, The Elements of Statistical Learning, p22, Fig 2.6.

KNN ⟶ lazy learner ⟶ learns every time it estimates without creating a learning model in advance.
Takes a long time to estimate. [No learning process]

- Dealing with same distance
  - If K = 5, which one should I choose between ● and ▲ for the point X in the following situation?
  - Points 1, 2, and 3 are selected unconditionally because they are less than K = 5, and points 4, 5, 6, 7, and 8 have similar distances, so two of them must be selected. There are three ways to determine the class of x in this situation as follows:



Points 4, 5, 6, 7 and 8 are equidistant, within the margin of error.

1) Random sampling

```
# Choose 2 out of 5 without replacement.
np.random.choice(np.arange(4, 9), size=2, replace=False)
result = [4, 6]
```

* Classify X as ●

2) Choose the side with a smaller sum of distances

● : 1.2 + 2.0 * 3 = 7.2

▲ : 1.0 + 1.3 + 2.0 * 2 = 6.3

* Classify X as ▲

3) Choose the one with the greater sum of the selection probabilities.

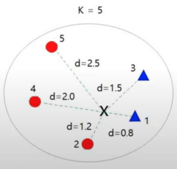
No	class	distance	selection probability
1	▲	1.0	1
2	●	1.2	1
3	▲	1.3	1
4	●	2.0	2/5
5	●	2.0	2/5
6	●	2.0	2/5
7	▲	2.0	2/5
8	▲	2.0	2/5

▲ = 1 + 1 + 2/5 + 2/5 =2.8
● = 1 + 2/5 + 2/5 + 2/5 = 2.2

* Classify X as ▲

- Weighted KNN (WKNN)
  - If K = 5, which one should I choose between ● and ▲ for the point X in the following situation?
  - In vanilla KNN, X is classified as ● because ● has three majorities. But X is closer to ▲. Giving higher weight to samples that are closer is called weighted KNN.
  - The closer the distance, the greater the weight, and vice versa. (inverse weighting).

K = 5



X: test data

1) Vanilla KNN (unweighted KNN)

* Classify X as ●

2) Weighted KNN

No	class	distance	Inverse weight
1	▲	0.8	1 / 0.8 = 1.25
2	●	1.2	1 / 1.2 = 0.83
3	▲	1.5	1 / 1.5 = 0.67
4	●	2.0	1 / 2.0 = 0.50
5	●	2.5	1 / 2.5 = 0.40
		sum	3.65

- Calculating inverse weighted distance

▲ ⟶ $\frac{1.25+0.67}{3.65} = 0.525$

● ⟶ $\frac{0.83+0.5+0.4}{3.65} = 0.475$

* Classify X as ▲

- Categorical data has no concept of distance like Manhattan, Euclidean, or Minkowski. Distance (or similarity) for categorical features must be redefined in a different way.

outlook	temp	windy	play
sunny	hot	False	no
sunny	hot	True	no
overcast	hot	False	yes
rainy	mild	False	yes
rainy	cool	False	yes
rainy	cool	True	no
overcast	cool	True	yes
sunny	mild	False	no
sunny	cool	False	yes
rainy	mild	False	yes
sunny	mild	True	yes
overcast	mild	True	yes
overcast	hot	False	yes
rainy	mild	True	?

feature / target class ← test data

Encoding →

outlook = [overcast, rainy, sunny] (0, 1, 2)

temp = [cool, hot, mild] (0, 1, 2)

windy = [False, True] (0, 1)

Label encoding

outlook	temp	windy
2	1	0
2	1	1
0	1	0
1	2	0
1	0	0
1	0	1
0	0	1
2	2	0
2	0	0
1	2	0
2	2	1
0	2	1
0	1	0
1	2	1

distance ?

One-hot encoding

outlook			temp			windy	
0	0	1	0	1	0	1	0
0	0	1	0	1	0	0	1
1	0	0	0	1	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	0	1	0
0	1	0	1	0	0	0	1
1	0	0	1	0	0	0	1
0	0	1	0	0	1	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	0	1	0	0	1	0	1
1	0	0	0	0	1	0	1
1	0	0	0	1	0	1	0
0	1	0	0	0	1	0	1

distance ?

- There is a way to measure the number of matching components between two vectors. We can call this overlap count-based similarities. Jaccard, Hamming (binary) distance corresponds to this method. And there is also a way to measure cosine similarity.

Label encoding

	outlook	temp	windy
A →	2	1	0
B →	2	1	1
	0	1	0
	1	2	0
	1	0	0
	1	0	1
	0	0	1
	2	2	0
	2	0	0
	1	2	0
	2	2	1
	0	2	1
	0	1	0
C →	1	2	1

distance

One-hot encoding

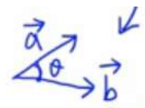
	outlook			temp			windy	
A →	0	0	1	0	1	0	1	0
B →	0	0	1	0	1	0	0	1
	1	0	0	0	1	0	1	0
	0	1	0	0	0	1	1	0
	0	1	0	1	0	0	1	0
	0	1	0	1	0	0	0	1
	1	0	0	1	0	0	0	1
	0	0	1	0	0	1	1	0
	0	0	1	1	0	0	1	0
	0	1	0	0	0	1	1	0
	0	0	1	0	0	1	0	1
	1	0	0	0	0	1	0	1
	1	0	0	0	1	0	1	0
C →	0	1	0	0	0	1	0	1

distance

similarity

- Overlap count-base : d(C, B) is closer than d(C, A). More similarity.
  - d(C, A) = 1 − (0/3) = 1.0 ← label encoding (distance)
  - d(C, B) = 1 − 1/3 = 0.67
  - d(C, A) = 1 − 2/8 = 0.775 ← one-hot encoding
  - d(C, B) = 1 − 4/8 = 0.50
- Cosine similarity : d(C, B) is closer than d(C, A). More similarity.
  - d(C, A) = 1 - cosine_similarity(C, A) = 1.0 ← label encoding
  - d(C, B) = 1 - cosine_similarity(C, B) = 0.59
  - d(C, A) = 1 - cosine_similarity(C, A) = 1.0 ← hoe-hot encoding
  - d(C, B) = 1 - cosine_similarity(C, B) = 0.67

```
from sklearn.metrics.pairwise import cosine_similarity
B = np.array([[0,0,1,0,1,0,0,1]])
C = np.array([[0,1,0,0,0,1,0,1]])
d = 1 - cosine_similarity(C, B)
```



$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\,\|\vec{b}\|}$$

Categorical feature and KNN - Inverse Occurrence Frequency (IOF)

- (Overlap count-based similarity) has the disadvantage of not considering the frequency of occurrence of each label. For example, if your data contains a lot of label = 2, the probability of overlapping 2s increases and the similarity to any vector increases.
- IOF is a method that complements the shortcomings of the overlap method by considering the frequency of occurrence of each label. (similar concept to TF-IDF in text mining)

Label encoding

	outlook	temp	windy
	2	1	0
A →	2	1	1
	0	1	0
	1	2	0
	1	0	0
	1	0	1
	0	0	1
	2	2	0
	2	0	0
B →	1	2	0
	2	2	1
	0	2	1
	0	1	0
C →	1	2	1

Frequency table : f_k

label	outlook	temp	windy
0	4	4	8
1	4	4	5
2	5	5	0

* reference : Chandola, V., et, al., 2007, Similarity Measures for Categorical Data – A Comparative Study

$$\mathrm{sim}_k = \begin{cases} 1 & if\,(x_k = y_k) \\ \dfrac{1}{1+\log(f_k(x_k))\cdot\log(f_k(y_k))} & otherwise \end{cases}$$

$$\mathrm{sim} = \frac{1}{1+\mathrm{d}} \qquad \left(\mathrm{d} = \frac{1}{\mathrm{sim}} - 1\right)$$

$$C(outlook)=1,\ A(outlook)=2 \rightarrow \frac{1}{1+\log(4)\times\log(5)}=0.309$$

$$C(temp)=2,\ A(temp)=1 \rightarrow \frac{1}{1+\log(5)\times\log(4)}=0.309$$

$$C(windy)=1,\ A(windy)=1 \rightarrow =1.0$$

$$\mathrm{sim(C,A)}=\frac{0.309+0.309+1.0}{3}=0.539$$

$$d(C,A) = \frac{1}{0.539}-1 = 0.855$$

$$B(outlook)=1 \rightarrow =1.0$$

$$B(temp)=2 \rightarrow =1.0$$

$$B(windy)=0 \rightarrow \frac{1}{1+\log(5)\times\log(8)}=0.23$$

$$\mathrm{sim(C,B)}=\frac{1.0+1.0+0.23}{3}=0.743$$

$$d(C,B) = \frac{1}{0.743}-1 = 0.346$$

C has higher similarity and closer distance to B.

Categorical feature and KNN – Categorical and numerical features are mixed.

Data that contains a mix of categorical and numeric features.

	outlook	temp	windy	any
	2	1	0	12.71
A →	2	1	1	13.28
	0	1	0	27.26
	1	2	0	28.46
	1	0	0	27.06
	1	0	1	18.8
	0	0	1	19.71
	2	2	0	14.92
	2	0	0	31.28
B →	1	2	0	10.08
	2	2	1	9.22
	0	2	1	23.17
	0	1	0	15.56
C →	1	2	1	30.50

1) Convert numeric features to categorical.

any
0
1
3
3
3
2
2
1
3
0
0
2
0
1

```
p = [25, 50, 75]
q1, m, q3 = np.percentile(any, p)

for a in any:
 if a < q1:
     print('0')
 elif a >= q1 and i < m:
     print('1')
 elif a >= m and i < q3:
     print('2')
 else:
     print('3')
```

3) Treat categorical and numerical features separately. (2)

- Calculate distances for categorical and numerical features respectively.
- Perform KNN[1] using the distance of categorical features, and estimate the classes of test data.
- Perform KNN[2] using the distances of numerical features, and estimate the classes of test data.
- Majority voting for two results.

2) Treat categorical and numerical features separately. (1)

- For categorical features, calculate the IOF distance.
- For numerical features, calculate distance such as Euclidean, and then normalize.
- Calculate the weighted average of the two distances. (The weights are hyper-parameter)

IOF distance of [outlook, temp, windy]	Euclidean distance of 'any'
$d(C,A)=0.855$	0.802
$d(C,B)=0.346$	0.958

ex)

```
a = np.array([12.71,13.28,27.26,28.46,...])
t = np.array([30.5])
d1 = np.abs(a-t)
d = (d1 - d1.min()) / (d1.max() - d1.min())
```

- You can also multiply two distances and think of it as an area.

Area = (IOF distance) * (distance of 'any')

Regression Algorithm

KNN Regression

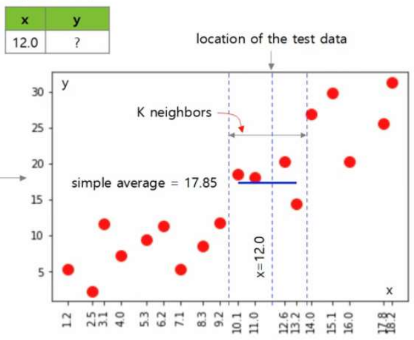
- We want to use the training data to estimate the y value of the test data where x is 12.0. (ex: K = 4)

[training data]

x	y
1.2	5.28
6.2	11.38
12.6	20.34
8.3	8.52
16.0	20.27
4.0	7.20
11.0	18.13
18.2	31.34
10.1	18.49
7.1	5.40
15.1	29.84
5.3	9.43
3.1	11.6
14.0	26.96
13.2	14.42
17.8	25.60
2.5	2.33
9.2	11.80

[test data]

x	y
12.0	?



1. Simple average method

$$\hat{y} = \frac{18.49+18.13+20.34+14.42}{4} = 17.85$$

2. Weighted average method

Neighbors of (x = 12.0)

No	x	y	distance	Inverse weight
1	10.1	18.49	|12.0-10.1|=1.9	1 / 1.9 = 0.53
2	11.0	18.13	|12.0-11.0|=1.0	1 / 1.0 = 1.00
3	12.6	20.34	|12.0-12.6|=0.6	1 / 0.6 = 1.67
4	13.2	14.42	|12.0-13.2|=1.2	1 / 1.2 = 0.83
			sum	4.03

The shorter the distance, the greater the weight, and vice versa. ← inverse weights

$$\hat{y}=\frac{18.49\times0.53+18.13\times1.00+20.34\times1.67+14.42\times0.83}{4.03}=18.32$$

- If there are multiple features x, use the Euclidean-like distance to estimate the y value of the test data. The estimation process is the same as above.

$$d = \sqrt{(x'_1-x_1)^2 + (x'_2-x_2)^2 + \dots} \quad \text{(x': test data, x: training data)}$$