KNN
Kmeans Algorithm

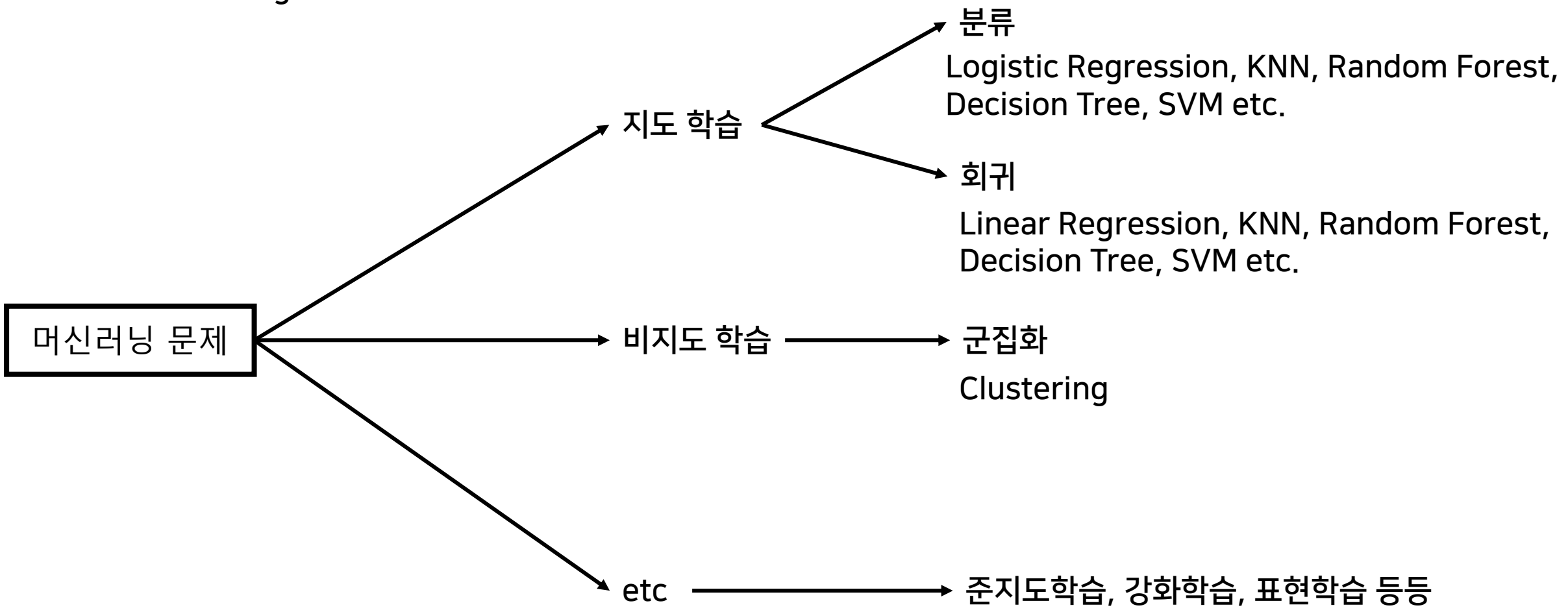
정보융합학부
2018204009 김기수

Due Date : 2023.01.17

목차

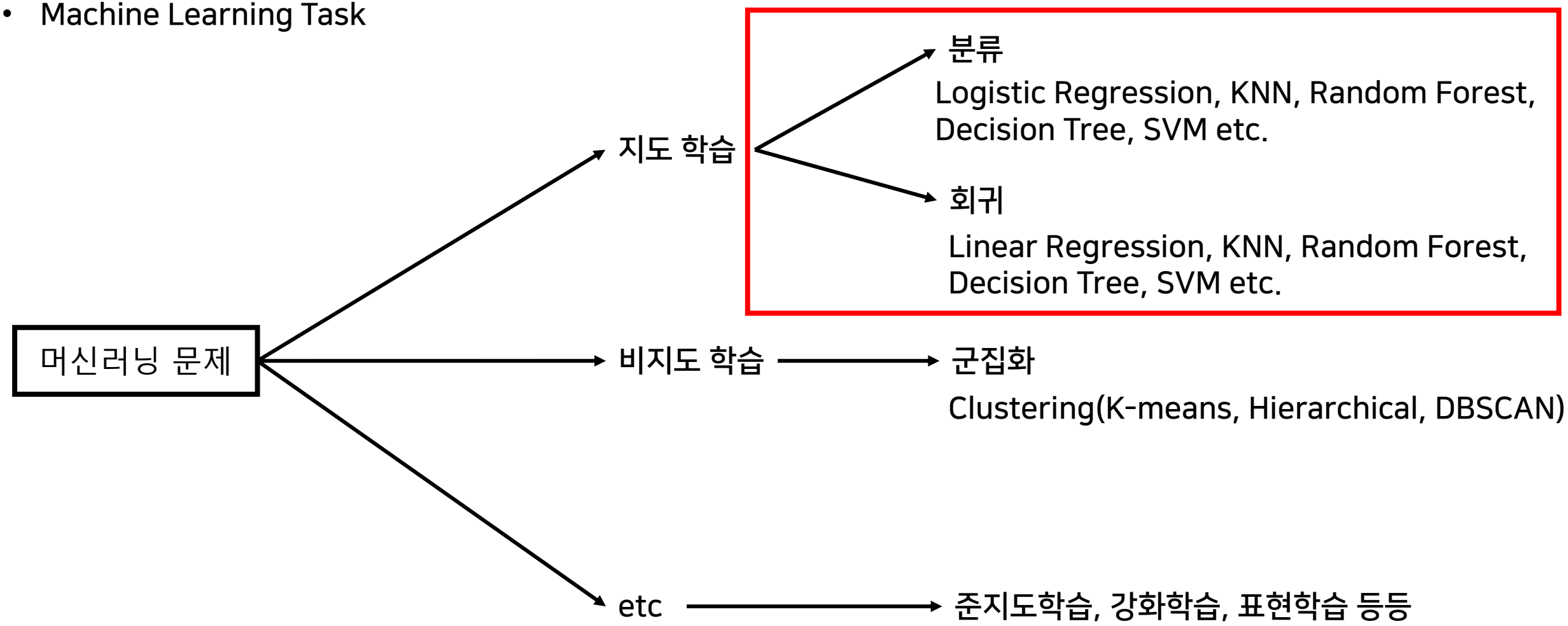
Introduction

- Machine Learning Task



Where is KNN?

- Machine Learning Task



KNN

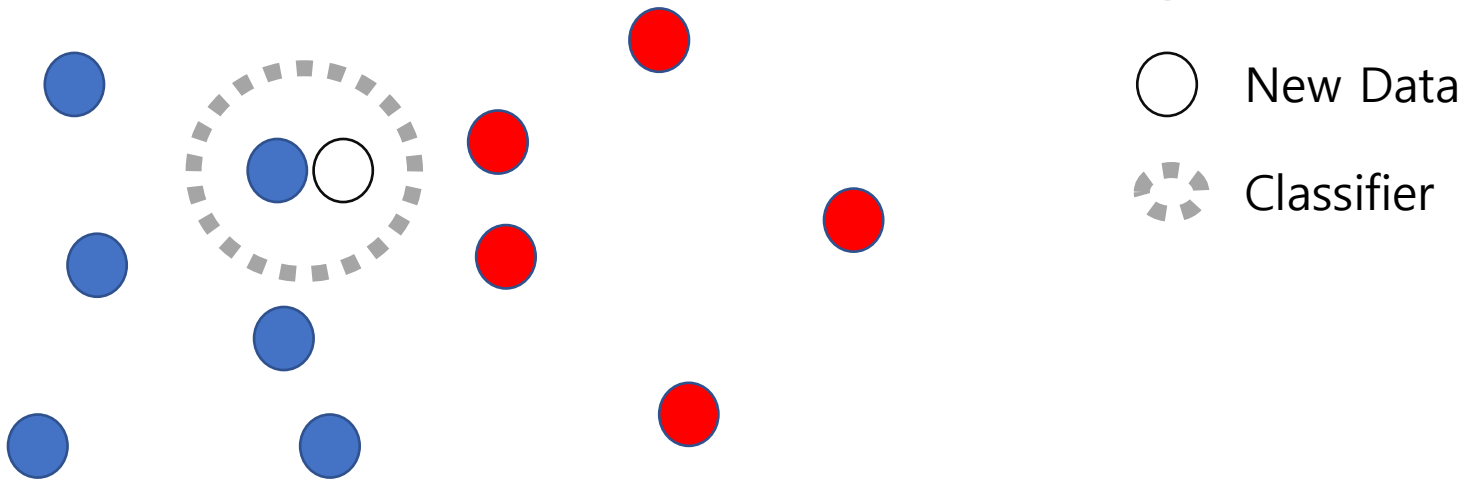
- KNN(K-Nearest Neighbor)
  - 지도 학습 알고리즘 중 하나로 분류와 회귀에 모두 쓰일 수 있음

- 알고리즘 특징
  - 거리에 기반하여 분류 및 회귀 문제에 사용하는 알고리즘
  - 알고리즘 자체가 매우 직관적이기 때문에 학습절차를 이해하는 것이 쉬움
  - 거리정보와 관련한 메커니즘을 이용할 때 매우 유용함

- 적절한 k 값을 찾는 것이 중요
  - K 값을 어떻게 잡냐에 따라 예측 값이 달라질 수 있기 때문에, 적절한 k 값을 찾는 것이 중요

KNN

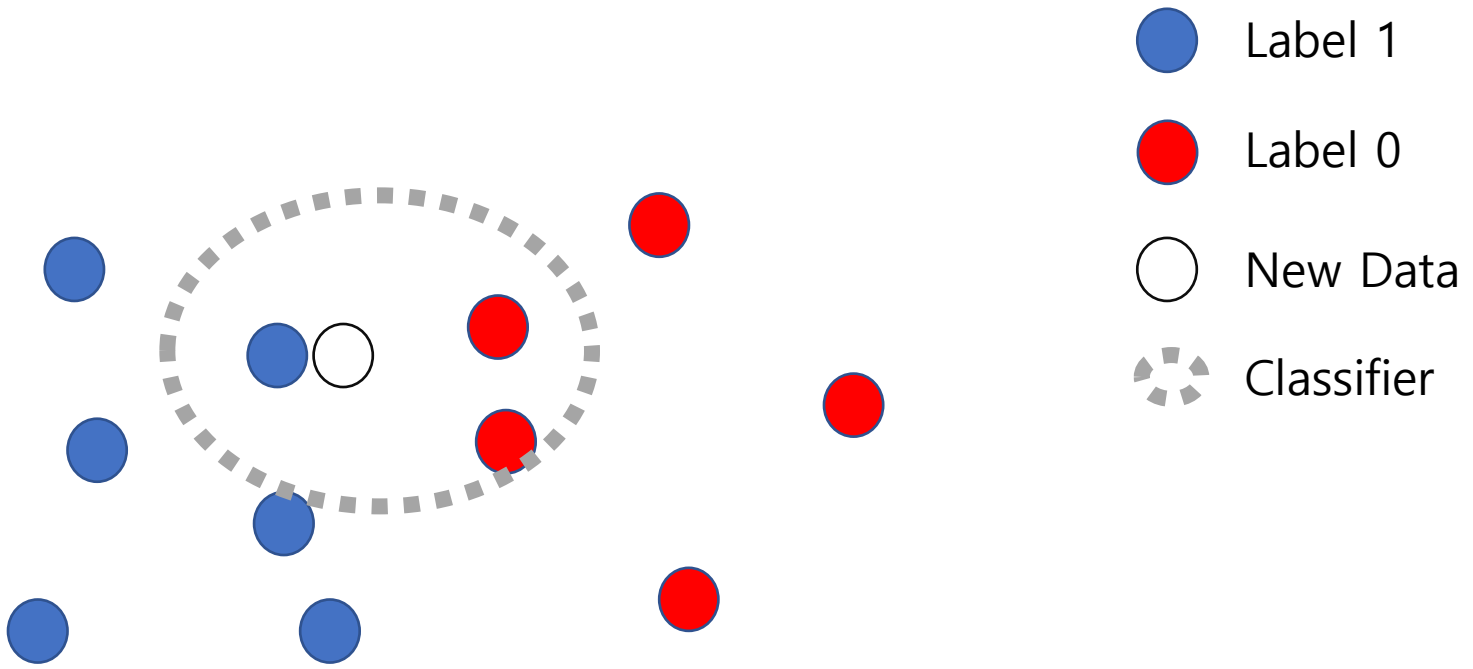
- K 값에 따른 분류 변화



If k = 1? Label 1!

KNN

- K 값에 따른 분류 변화



If k = 3? Label 0!

사용되는 거리 정보

- KNN(K-Nearest Neighbor)
  - 지도 학습 알고리즘 중 하나로 분류와 회귀에 모두 쓰일 수 있음

- KNN 알고리즘에 쓰이는 거리 정보
  - 민코프스키 거리(Minkowski distance)
    - 가장 일반화된 거리 척도라고 할 수 있음. K = 1이면 맨해튼 거리, k = 2이면 유클리디안 거리
  - 유클리디안 거리(Euclidean distance)
    - 두 점 간의 직선 거리를 측정함, L2-Distance라고도 함.
  - 맨해튼 거리(Manhattan distance)
    - 두 점 간의 직선 거리가 아닌, 두 점 간의 grid를 고려한 거리, L1-Distance라고도 함
  - 마할라노비스 거리(Mahalanobis distance)
    - 변수 간의 분산을 반영한 거리. 유클리디안 거리는 공분산 행렬로 나누었기 때문에, 분산이 크면 거리를 작게, 분산이 작으면 거리를 크게 측정하는 원리
- 변수마다 범위가 다르기 때문에 거리 계산 시에 정규화하는 과정이 포함
  - 사람의 키와 몸무게의 범위가 다르기 때문에, 키만 큰 사람에 대해 거리정보가 높게 계산될 수 있음

KNN 학습 절차(분류)

- KNN의 학습 절차(분류)
  1. 라벨링이 된 데이터를 준비함(train data)
  2. Test data의 하나의 데이터를 input으로 받는다.(test data input)
  3. 2에서 받은 input과 모든 train data의 데이터 사이의 거리를 계산하고 정렬한다.
  4. K 값을 정하고, k 값에 따라 다수의 class에 속하는 것을 label로 결정한다.

- KNN 알고리즘의 장점과 단점
  - 장점
    - 직관적이고 이해가 쉽고, 따로 학습 과정이 필요하지 않음.
  - 단점
    - 모든 데이터에 대해 거리 정보를 계산하기 때문에 시간이 매우 오래 걸림
    - 메모리를 많이 사용하게 되고, test 데이터를 예측하는 과정에서 시간이 매우 오래 걸림
    - 실시간 추론에 부적합

KNN 학습 절차(회귀)

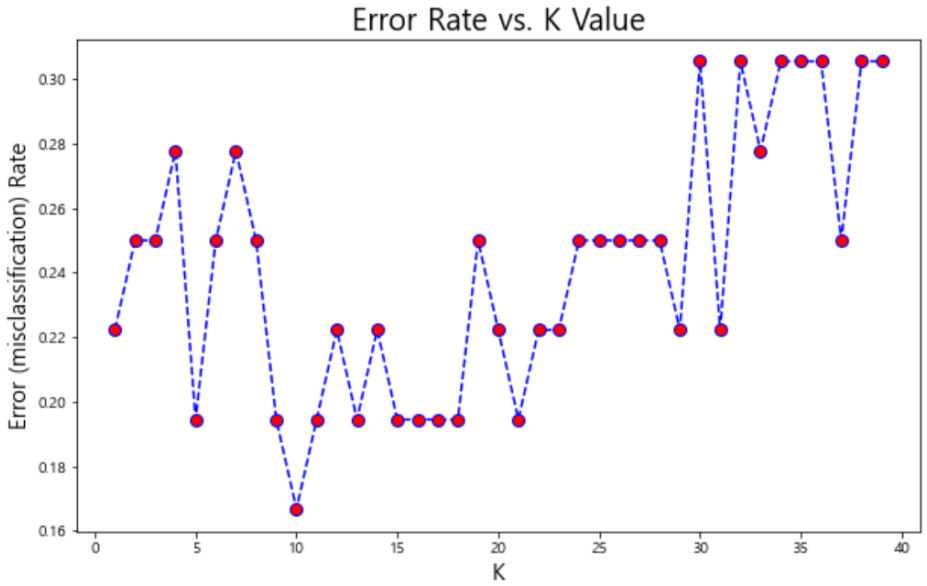
- KNN의 학습 절차(회귀)
  1. 라벨링이 된 데이터를 준비함(train data)
  2. Test data의 하나의 데이터를 input으로 받는다.(test data input)
  3. 2에서 받은 input과 모든 train data의 데이터 사이의 거리를 계산하고 정렬한다.
  4. K 값을 정하고, k 값에 속하는 train data의 평균을 구하여 예측한다.
     - k=3일 때, k에 속하는 label이 15, 20, 22이면 (15+20+22) / 3 = 19로 예측

- KNN 알고리즘의 장점과 단점
  - 장점
    - 직관적이고 이해가 쉽고, 따로 학습 과정이 필요하지 않음.
  - 단점
    - 모든 데이터에 대해 거리 정보를 계산하기 때문에 시간이 매우 오래 걸림
    - 메모리를 많이 사용하게 되고, test 데이터를 예측하는 과정에서 시간이 매우 오래 걸림
    - 실시간 추론에 부적합

KNN

- 적절한 k값 찾기
  - 적절한 k값을 찾아야 성능이 좋아질 수 있음
  - 따로 학습하는 과정이 없기 때문에, 가능한 k값에 대해 모든 경우를 실험하고 적절한 elbow point 찾기

- Wine 데이터로 elbow point 찾아보기
  - K = 10인 경우에 최소가 되고 그 이후로는 과적합이 되는 경향이 있으므로 10 선택!



KNN with Python Code

- 라이브러리 불러오기

In [23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

plt.rcParams['font.family'] = 'Malgun Gothic'

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
```

- 데이터 불러오기

In [31]:
```
wine = load_wine()
df = pd.DataFrame(wine.data, columns = wine.feature_names)
df['target'] = wine.target
```

KNN with Python Code

- **학습 환경 구축하기**

In [32]:
```
from sklearn.neighbors import KNeighborsClassifier

X = df.drop('target', axis = 1)
y = df['target']

# 학습 데이터와 검증 데이터를 8 : 2로 분리하기
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify = y)
```

- **Error rate 비교하는 plot 그리기**

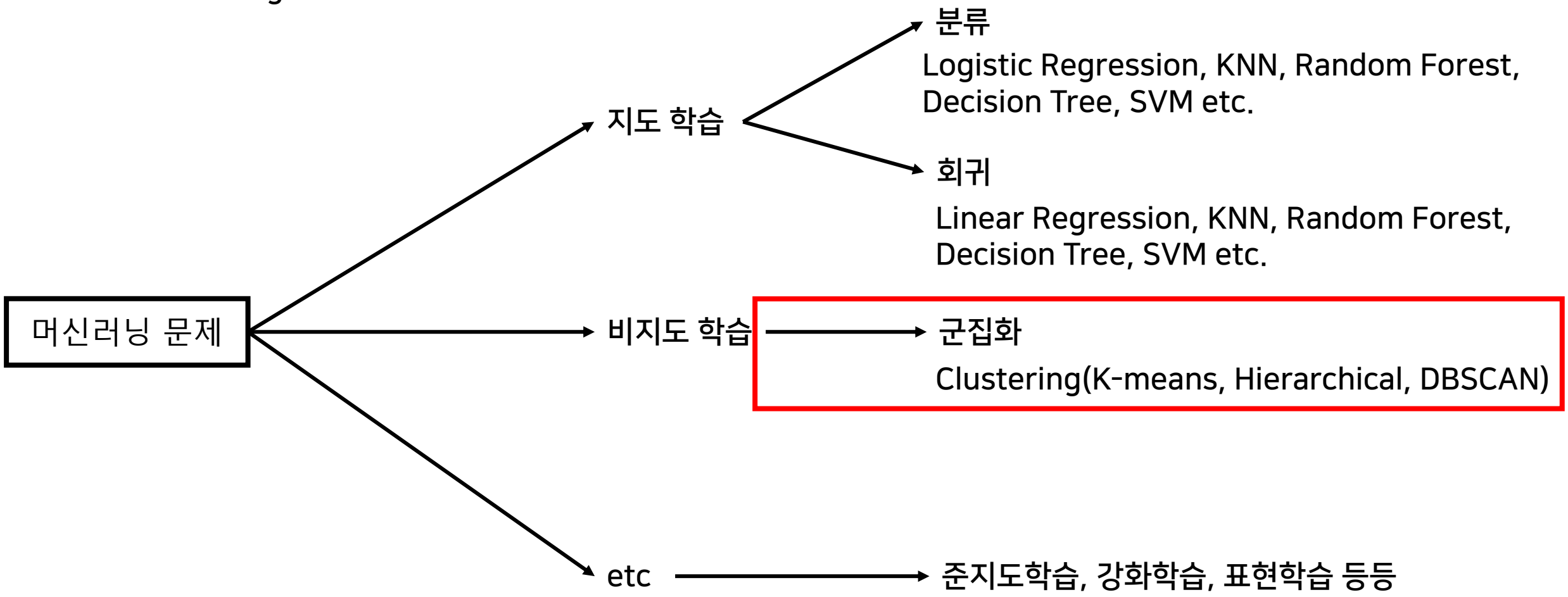
In [33]:
```
error_rate = [] # 각 k값 별로 error값 저장

# 1부터 40까지의 k값을 학습시키고 error 측정
for k in range(1, 40):
    knn_clf = KNeighborsClassifier(n_neighbors = k)
    knn_clf.fit(X_train, y_train)
    knn_pred = knn_clf.predict(X_test)
    error = np.mean(knn_pred != y_test)
    error_rate.append(error)

# k값에 따른 error rate 시각화
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=8)
plt.title('Error Rate vs. K Value', fontsize=20)
plt.xlabel('K',fontsize=15)
plt.ylabel('Error (misclassification) Rate',fontsize=15)
```

Where is K-Means

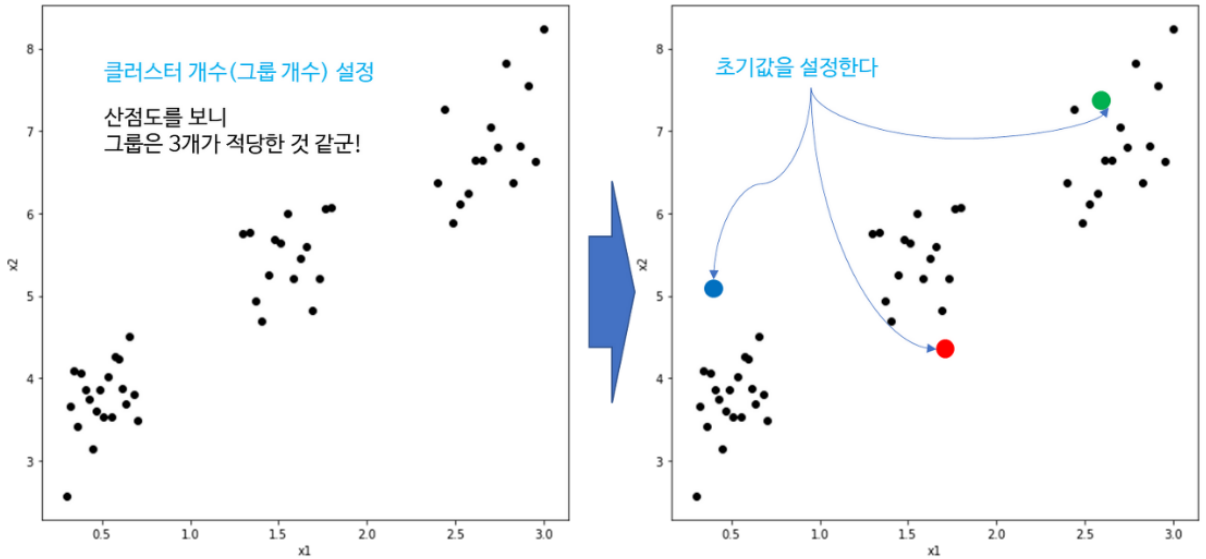
- Machine Learning Task



K-Means

- K-Means
  - 비지도 학습 알고리즘 중 하나로 유사한 속성을 갖는 그룹을 묶는 기법
    - 초기에 클러스터할 개수와 기준점을 먼저 설정해야 함
    - 비지도 학습 방법 중 하나이기 때문에 Y label이 없음

- 클러스터 개수와 기준 데이터 설정 예시



[1] https://zephyrus1111.tistory.com/179

군집을 만드는 기준과 방법

- 가장 좋은 군집 방법
  - 동일한 군집 내의 관측치들끼리는 거리가 가깝고, 다른 집단 간의 거리는 먼 것이 좋음

- 거리 정보 측정 방법
  - KNN에서 언급한 네 가지 정보를 전부 사용 가능하지만, 주로 유클리디안 거리를 사용

- 성능 평가 방법
  - 거리 정보를 기반으로 한 유클리디안 거리 정보 이용
  - 실루엣 스코어(Silhouette Score)

군집화 과정 및 K-Means의 특징

- K-Means의 군집화 과정
  1. Clustering을 할 cluster의 개수를 설정(k 설정)
  2. K개의 중심점을 임의로 설정(중심점을 Centroid라고 함)
     - 중심점을 설정하는 방법은 random, manually, k-means++
  3. 중심점을 기준으로 모든 데이터를 순회하며 가장 가까운 Centroid에 군집 할당
  4. 데이터 레이블이 확정되면 각 cluster에 속하는 데이터들을 이용하여 Centroid 다시 계산
  5. 3, 4를 반복하여 변하지 않으면 종료

- K-Means 알고리즘의 특징
  - 장점
    - 알고리즘 구현이 쉽고 이해가 쉬움
  - 단점
    - 초기 중심점을 어떻게 잡냐에 따라 결과가 달라지기 때문에, global optimal이 아닌 local optimal을 찾는 상황이 생길 수 있음
    - 이상치에 영향을 받기 쉽고, 데이터의 분포가 구형의 형태로 되어있는 경우 군집화 결과가 좋지 않음

K-Means

- K-Means의 군집화 과정
  1. Clustering을 할 cluster의 개수를 설정(k 설정)
  2. K개의 중심점을 임의로 설정(중심점을 Centroid라고 함)
     - 중심점을 설정하는 방법은 random, manually, k-means++
  3. 중심점을 기준으로 모든 데이터를 순회하며 가장 가까운 Centroid에 군집 할당
  4. 데이터 레이블이 확정되면 각 cluster에 속하는 데이터들을 이용하여 Centroid 다시 계산
  5. 3, 4를 반복하여 변하지 않으면 종료

- 거리 계산 하는 방법(Ci : i번째 군집의 중심점, x : s번째 데이터의 좌표)

$$\sum_{i=1}^{k}\sum_{s\in X}^{m}|C_i - x_s|$$

$$= \sum_{i=1}^{k}\sum_{s\in X}^{m}\sqrt{(p_{c_i} - p_{x_s})^2 + (q_{c_i} - q_{x_s})^2}$$

K-Means with Python Code

- Wine 데이터에 K-Means 알고리즘 적용하기
  - K-Means 학습 환경 구축하기

In [54]:
```
from sklearn.cluster import KMeans

data_df = df.drop('target', axis = 1)
model = KMeans(n_clusters = 3, n_init = 10, algorithm = 'auto', init = 'k-means++') # 초기 중심점 설정과 군집 개수 설정
model.fit(data_df) # 설정한 파라미터를 토대로 군집화
predict = model.predict(data_df)

# 시각화 하기 위해 2차원으로 차원 축소(실제로는 13차원, 설명률 : 99.8%)
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(data_df)
pca_df = pd.DataFrame(pca.transform(data_df), columns = ['pca_1', 'pca_2'])
pca_df['cluster'] = predict
pca.explained_variance_ratio_
```

Out[54]:
```
array([0.99809123, 0.00173592])
```

  - 차원 축소한 데이터 살펴보기

In [65]:
```
pca_df.head(5)
```

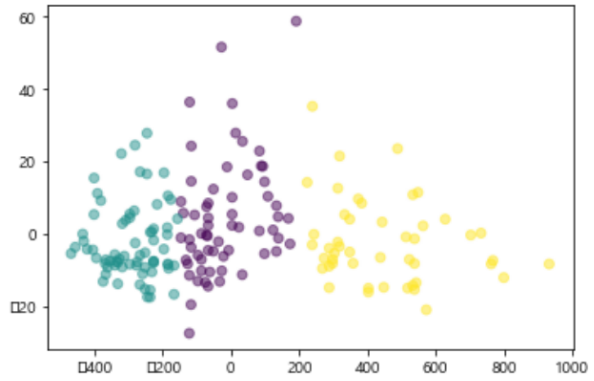
Out[65]:

	pca_1	pca_2	cluster
0	318.562979	21.492131	2
1	303.097420	-5.364718	2
2	438.061133	-6.537309	2
3	733.240139	0.192729	2
4	-11.571428	18.489995	0

K-Means with Python Code

- Wine 데이터에 K-Means 알고리즘 적용하기
  - 군집화 결과 살펴보기

```
In [63]: plt.scatter(pca_df['pca_1'], pca_df['pca_2'],c=pca_df['cluster'],alpha=0.5)
Out[63]: <matplotlib.collections.PathCollection at 0x26568f09730>
```

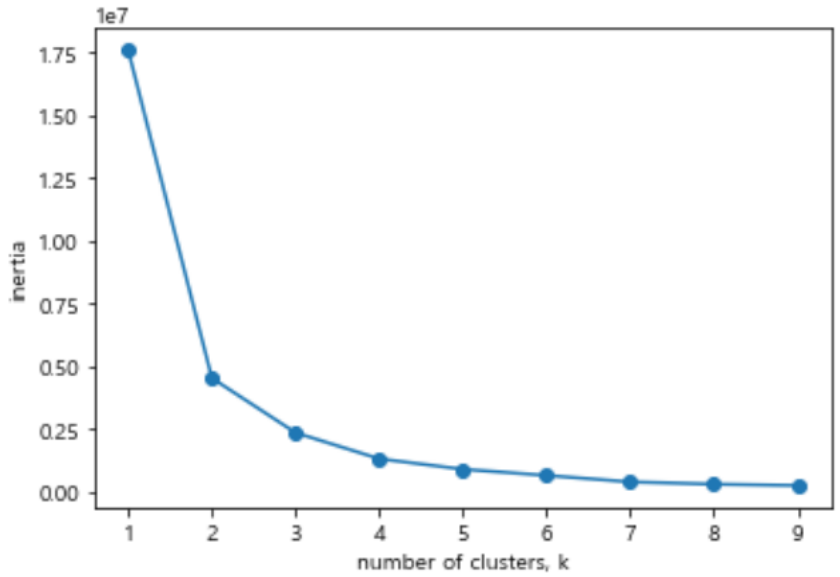


  - Elbow Point 찾기

```
In [64]: ks = range(1,10)
         inertias = []
         for k in ks:
             model = KMeans(n_clusters=k)
             model.fit(data_df)
             inertias.append(model.inertia_)
         # Plot ks vs inertias
         plt.plot(ks, inertias, '-o')
         plt.xlabel('number of clusters, k')
         plt.ylabel('inertia')
         plt.xticks(ks)
         plt.show()
```

K-Means with Python Code

- Wine 데이터에 K-Means 알고리즘 적용하기
  - Elbow point 찾기
    - K의 값이 2일 때 가장 적절해 보임



K-Means with Python Code

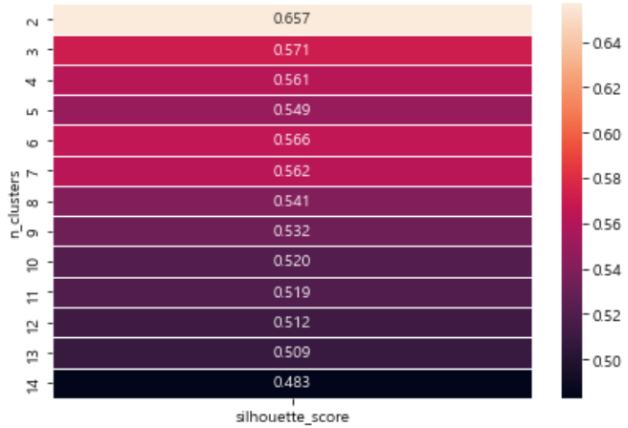
- Wine 데이터에 K-Means 알고리즘 적용하기
  - Silhouette score를 기반으로 찾기
    - K의 값이 2일 때 가장 적절해 보임

In [69]:
```
from sklearn.metrics import silhouette_score #실루엣 스코어 계산 위한 라이브러리 임포트
clusters_range = range(2,15) #지정할 클러스터 개수
results = []

for i in clusters_range: #실루엣 스코어 계산
    clusterer = KMeans(n_clusters=i, init='k-means++', n_init=10, algorithm = 'auto')
    cluster_labels = clusterer.fit_predict(data_df)
    silhouette_avg = silhouette_score(data_df, cluster_labels)
    results.append([i, silhouette_avg])

result = pd.DataFrame(results, columns=["n_clusters", "silhouette_score"])
pivot_km = pd.pivot_table(result, index="n_clusters", values="silhouette_score")

plt.figure()
sns.heatmap(pivot_km, annot=True, linewidths=.5, fmt='.3f', cmap=sns.cm._rocket_lut)
plt.tight_layout()
plt.show()
```



K-Means with Python Code

- Wine 데이터에 K-Means 알고리즘 적용하기
  - Silhouette score 시각화
    - Cluster의 개수가 2, 3, 4인 경우에 대해서 시각화