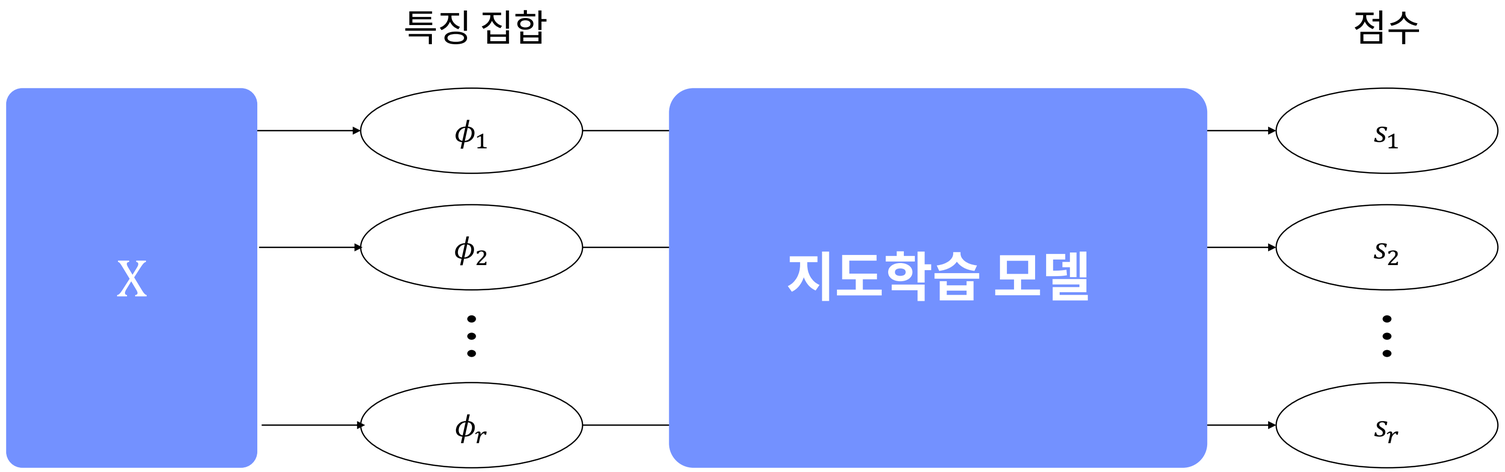
# 2. 실습 1 – 특징 선택

## 1 문제 정의

# 래퍼 방법

래퍼 방법은 휴리스틱 해법을 이용해서 한 모델의 예측 정확도를 최대화하는 특징 집합을 찾는 방법입니다.

래퍼 방법은 원 특징 집합 $\mathrm{X} = \{x_1, x_2, \cdots, x_n\}$의 부분 집합 $\phi$ 가운데 모델의 예측 정확도를 최대화하는 부분 집합을 찾는 것입니다.




해 개수가 $n$개라면 상태 공간의 크기가 $2^n$이 되서 최적의 특징 집합을 찾을 순 없음

# 데이터 및 모델

### 데이터 불러오기

```
1  import pandas as pd
2  df = pd.read_csv("../../data/classification/wdbc.csv")
3  df = df.sample(frac = 1, random_state = 2022)
4  X = df.drop('y', axis = 1)
5  y = df['y']
```

- **라인 3:** sample 메서드는 데이터프레임의 행 일부를 임의로 샘플링하는 함수입니다. frac은 선택할 행의 비율을 나타내는 인자입니다. frac을 1로 설정하면 원 데이터가 다시 선택되는 것이므로 데이터의 순서를 임의로 섞는 효과가 있습니다.

### 모델 정의

```
1  from sklearn.linear_model import LogisticRegression
2  model_instance = LogisticRegression(random_state = 2022)
```

- **라인 1 ~ 2**: 모델은 random_state를 제외한 나머지 하이퍼파라미터를 기본값으로 설정한 로지스틱 회귀를 사용하겠습니다.

# cross_val_score 함수

cross_val_score 함수는 k-겹 교차 검증 방식으로 모델을 평가하는 함수입니다. 사용 방법은 간단하지만, 전처리를 효과적으로 하기 어려워 자주 사용하지는 않습니다. 그러나 이 문제에서는 특징 선택 외에 특별한 전처리가 필요하지 않아 사용하겠습니다.

**주요 인자**

| 인자 | 설명 |
|---|---|
| estimator | 모델 인스턴스 |
| X | 특징 벡터 |
| y | 라벨 |
| scoring | 평가 척도 |
| cv | 폴드 개수 |

**cross_val_score 사용 예제**

```
1  from sklearn.model_selection import cross_val_score
2  result = cross_val_score(model_instance, X, y, cv = 5, scoring = "f1")
3  display(result)
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
array([0.88888889, 0.91358025, 0.91566265, 0.95348837, 0.92857143])
```

- 수렴하지 못해 경고가 발생함
- 그러나 하이퍼파라미터를 따로 튜닝하지 않으므로 수렴했는지는 중요하지 않음

# filterwarnings 함수

warnings.filterwarnings 함수는 특정 경고를 출력하지 않는 데 사용합니다.

**주요 인자**

| 인자 | 설명 |
|---|---|
| action | “ignore”이면 경고 출력을 비활성화하고 “default”면 경고를 활성화함 |
| category | 경고의 종류를 설정 |

**경고 끄기**

```
1  import warnings
2  warnings.filterwarnings('ignore')
```

- **라인 2**: filterwarnings의 action 인자를 “ignore”로 설정해 경고를 비활성화합니다.

# 2. 실습 1 – 특징 선택

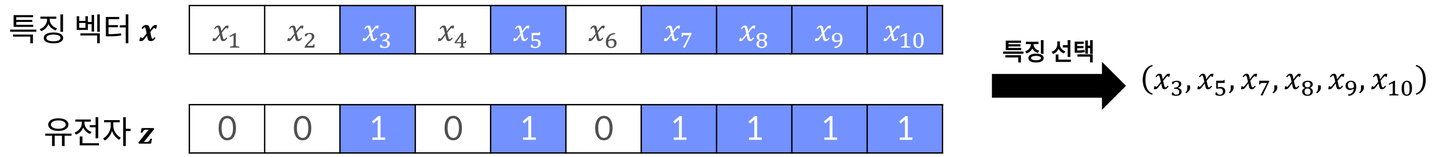
## 2 연산자 정의

# 해 표현 및 초기 해 집단 생성



유전 알고리즘의 해는 이진 인코딩을 사용해 표현하겠습니다

**특징 선택을 위한 해 표현:** 대응되는 유전 개체가 1이면 특징을 선택하고, 0이면 선택하지 않음




**초기 해 생성**

```
1  import numpy as np
2  def initialize(n, m): # 우리가 해결할 문제의 데이터에는 30개의 특징이 포함되므로 m을 30으로 설정할 예정입니다.
3      Z = np.random.choice([0, 1], (n, m))
4      Z = Z.astype(bool)
5      return Z
```

# 적합도 함수



적합도는 5-겹 교차 검증으로 로지스틱 회귀를 학습했을 때의 평균 F1 점수를 사용하겠습니다.

**적합도 함수**

```
1  def fitness(X, y, model, z):
2      score = cross_val_score(model, X.loc[:, z], y, cv = 5, scoring = "f1")
3      return score.mean()
```

- **라인 2:** 해 z를 바탕으로 X의 열을 선택한 특징 벡터 X.loc[:, z]로 5-겹 교차 검증을 수행한 결과를 score에 저장합니다.
- **라인 3:** score의 평균을 반환합니다.

# 선택 연산자

선택 연산자로는 룰렛 휠 방법을 사용하겠습니다. 그 이유는 탐색 공간이 매우 넓고 적합도가 0과 1 사이라서 해별 적합도가 크게 차이 나지 않으므로 룰렛 휠을 쓰면 다양한 해를 탐색할 수 있기 때문입니다.

**선택 연산자**

```
1  def selection(Z, S, k):
2      selected_index = []
3      _S = S.copy()
4      for _ in range(k):
5          probs = _S / _S.sum()
6          z_idx = np.random.multinomial(1, probs).argmax()
7          selected_index.append(z_idx)
8          _S[z_idx] = 0
9      return Z[selected_index]
```

# 교차 연산자

교차 연산자로는 한 점 교차 연산자를 사용하겠습니다.

**교차 연산자**

```
1  def crossover(X1, X2):
2      point_idx = np.random.choice(range(1, len(X1)))
3      new_X = np.hstack([X1[:point_idx], X2[point_idx:]])
4      return new_X.astype(int)
```

- **라인 2:** 1과 len(X1) 사이의 한 점을 임의로 선택합니다.
- **라인 3:** point_idx까지 슬라이싱한 X1과 point_idx부터 슬라이싱한 X2를 병합하여 new_X를 만듭니다.

# 돌연변이 연산자

돌연변이 연산자로는 비트 플립 돌연변이 연산자를 사용하겠습니다.

**돌연변이 연산자**

```
1  def bit_flip(z, p):
2      probs = np.random.random(len(z))
3      z[probs < p] = 1 - z[probs < p]
4      return z
```

# 2. 실습 1 – 특징 선택

## 3 메인 함수

# 메인 함수



```
1  def main(n, m, k, p, q, num_generation):
2      best_score = -1
3      Z = initialize(n, m) # 초기해 생성
4      for _ in range(num_generation):
5          S = np.array([fitness(X, y, model_instance, z) for z in Z]) # 해 평가
6          current_best_score = S.max()
7          current_best_features = Z[S.argmax()]
8          if current_best_score > best_score:
9              best_score = current_best_score
10             best_features = current_best_features
11         Z_new = selection(Z, S, k) # k개 해 선택
12
13         # 교배 및 돌연변이 연산
14         children = []
15         for _ in range(n - k):
16             parent_idx = np.random.choice(range(k), 2, replace = False)
17             child = crossover(Z_new[parent_idx[0]], Z_new[parent_idx[1]])
18             if np.random.random() < q:
19                 child = bit_flip(child, p)
20             Z_new = np.vstack([Z_new, child])
21         Z = Z_new.astype(bool)
22 return best_features, best_score
```

- **라인 1:** 세대당 해의 개수 n, 특징 개수 m, 선택할 해의 개수 k, 돌연변이 확률 q, 각 유전 개체가 돌연변이가 될 확률 p, 세대 수 num_generation을 입력으로 받습니다.
- **라인 2 ~ 3:** 최고 점수를 -1로 초기화하고 초기해를 생성합니다.
- **라인 4:** 세대 수만큼 평가, 업데이트, 선택, 교배 및 돌연변이 연산을 반복합니다.
- **라인 5:** Z에 있는 모든 요소 z에 대해 fitness 함수를 사용해 계산한 적합도 목록을 S에 저장합니다. 또한, S는 max와 argmax 메서드를 사용하기 위해 ndarray로 변환합니다.
- **라인 6~7:** 현재 세대의 유전자 가운데 최대 적합도를 current_best_score에 저장하고, 최대 적합도를 갖는 유전자를 current_best_features에 저장합니다.
- **라인 8~10:** 현재 세대에서 최고 점수인 current_best_score가 지금까지의 최고 점수인 best_score보다 크다면 best_score와 best_features를 업데이트합니다.
- **라인 11:** selection 함수를 이용해서 해 집단 Z에서 적합도 점수 S를 바탕으로 k 개 해를 선택합니다. 선택한 해 집단은 Z_new에 저장합니다.
- **라인 14:** Z_new를 사용해 만들 자식 해 집단 children을 빈 목록으로 초기화합니다.
- **라인 15:** 한 세대의 해 개수 n에서 선택한 해의 개수 k를 뺀 개수만큼 자식 해를 생성합니다.
- **라인 16:** 부모 해를 선택하기 위해 부모 해 인덱스 두 개를 임의로 선정합니다.
- **라인 17:** crossover 함수를 사용해 두 부모 해를 교배함으로써 자식 해 child를 생성합니다.
- **라인 18 ~19:** 확률 q로 돌연변이 연산자를 적용합니다.
- **라인 20:** Z_new에 child를 추가합니다.
- **라인 21:** Z_new로 Z를 대체합니다. 이때 연산 과정에서 int 형으로 바뀌므로 astype(bool)을 사용해 부울 자료형으로 바꿉니다. 이로써 현재 세대가 새로운 세대로 교체됐습니다.
- **라인 22:** 최고 특징 집합과 최고 점수를 반환합니다.

# 특징 선택 수행

**유전 알고리즘의 하이퍼 파라미터 설정**

```
1  n = 20
2  m = X.shape[1]
3  k = 10
4  num_generation = 100
5  p = 0.1
6  q = 0.1
```

**특징 선택**

```
1  best_features, best_score = main(n, m, k, p, q, num_generation)
2  print(X.columns[best_features], best_score)
```

```
Index(['x1', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x11', 'x13', 'x14', 'x16', 'x17', 'x18', 'x19', 'x22', 'x23', 'x24', 'x26', 'x27', 'x28', 'x29'], dtype='object')        0.9420215882509126
```

- 유전 알고리즘을 사용해 평균적으로 0.9420 정도의 F1 점수가 나오는 특징 집합을 찾음
- 이 특징 집합은 평균 F1 점수를 기준으로 전체 특징을 사용한 특징 집합보다 약 0.022 정도 좋음