

# 로지스틱 회귀 과제

2021년 4월 29일



# 유방암 진단

로지스틱 회귀 분석으로 유방암을 진단해보자.

KNN

나이브 베이즈 성능

accuracy : 0.9564164648910412  
precision : 0.97  
recall : 0.7461538461538462  
f1\_score : 0.8434782608695651

3월 20일



로지스틱 회귀 성능

accuracy : 0.965034965034965  
precision : 1.0  
recall : 0.9019607843137255  
f1\_score : 0.9484536082474228

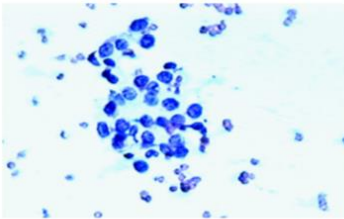
성능 ↑

# 1. 데이터셋

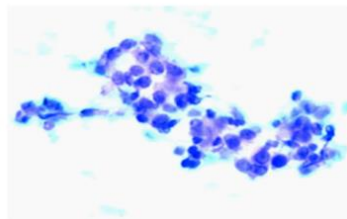
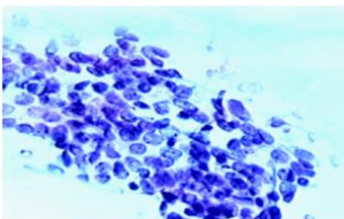
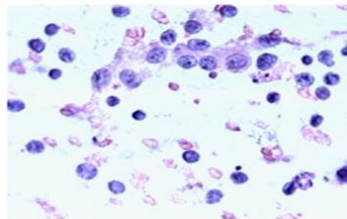
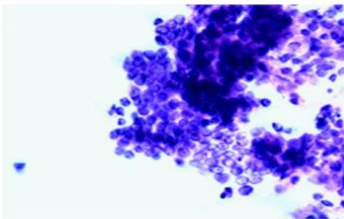
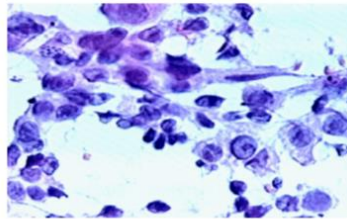


# 위스콘신 유방암 진단 데이터셋 (Wisconsin Breast Cancer Diagnostic dataset)

양성 (benign)



악성 (malignant)



## 위스콘신 유방암 진단 데이터셋 (WBCD)

- 위스콘신 대학 (University of Wisconsin)의 연구원들이 기부
- 유방암 조직 검사 **569개** 샘플
- 총 **32개 컬럼**으로 구성됨 (ID, 진단 결과, 30 실측값)
  - 진단 결과 " M " : 악성 (malignant), " B " : 양성 (benign)
  - **30개 실측값**
    - 유방 종양의 미세침 흡인물 이미지에서 측정한 세포핵의 특징
    - 세포핵의 10개 특징에 대한 평균, 표준 오차, 최악의 값(즉, 최댓값)

[https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+\(Diagnostic\)](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))

# 위스콘신 유방암 진단 데이터셋 (Wisconsin Breast Cancer Diagnostic dataset)

총 32개 컬럼

속성	설명	타입
<b>id</b>	아이디	int
<b>diagnosis</b>	M, B	char
<b>_mean</b>	평균 (3-12 컬럼)	float
<b>_se</b>	표준 오차 (13-22 컬럼)	float
<b>_worst</b>	최댓값 (23-32 컬럼)	float

10개 특징 별 (평균, 표준 오차, 최댓값)

속성	설명	타입
<b>Radius</b>	반지름	float
<b>Texture</b>	질감 (Gray-Scale 값의 표준 편차)	float
<b>Perimeter</b>	둘레	float
<b>Area</b>	넓이	float
<b>Smoothness</b>	매끄러움 (반지름의 변화율)	float
<b>Compactness</b>	조밀성 ( $\text{Perimeter}^2 / \text{Area} - 1$ )	float
<b>Concavity</b>	오목함	float
<b>Concave points</b>	오목한 점의 수	float
<b>Symmetry</b>	대칭성	float
<b>Fractal dimension</b>	프랙탈 차원	float

# 데이터셋 다운로드

## 패키지 импорт

```
import matplotlib.pyplot as plt
import os
from typing import List, Tuple
import csv
from scratch.linear_algebra import Vector, get_column
```

## 데이터 다운로드

```
import requests

data = requests.get("https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/wdbc.data")
dataset_path = os.path.join('data', 'wdbc.data')

with open(dataset_path, "w") as f:
    f.write(data.text)
```

- URL에서 데이터를 다운로드해서 'wdbc.data' 파일에 저장

# 데이터 파싱

회귀 분석을 할 수 있도록 데이터를 벡터 형태로 파싱

## 데이터 파싱



```
def parse_cancer_row(row: List[str]) -> Tuple[Vector, int]:  
    measurements = [float(value) for value in row[2:]]  
    label = row[1]  
    label = 1 if label == 'M' else 0  
    return measurements, label
```


- 레이블을 숫자 타입으로 0과 1로 변경
- 입력 데이터와 레이블을 별도로 반환

$List[Vector]$   
 $List[int] \in \{0, 1\}$   
B → M  
약정(약정)을

# 데이터 읽기

## 입력 데이터 X\_cancer와 레이블 데이터 y\_cancer 생성

### csv 파일 읽기 및 한 행 씩 파싱



```
X_cancer : List[Vector] = []
y_cancer : List[int] = []
with open(dataset_path) as f:
    reader = csv.reader(f)
    for row in reader:
        x, y = parse_cancer_row(row)
        X_cancer.append(x)
        y_cancer.append(y)
```

- csv 파일 읽기
- 각 row를 파싱해서 입력 데이터와 타겟으로 분리 (X\_cancer, y\_cancer)

```
print(X_cancer[0])
print(y_cancer[0])
```

```
[17.99, 10.38, 122.8, 1001.0, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419, 0.07871, 1.095, 0.9053,
8.589, 153.4, 0.006399, 0.04904, 0.05373, 0.01587, 0.03003, 0.006193, 25.38, 17.33, 184.6,
2019.0, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601, 0.1189]
```

1



# 컬럼 이름

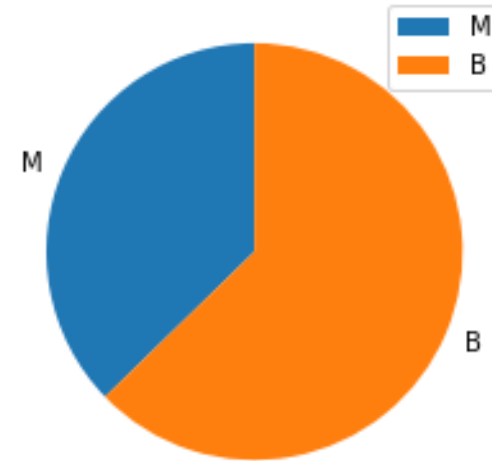
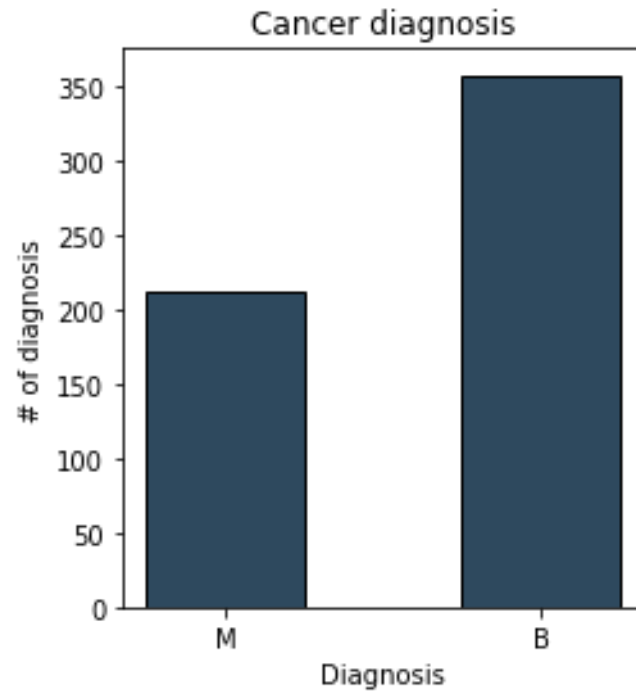
## 컬럼 이름

```
columns = [  
    "radius_mean", "texture_mean", "perimeter_mean", "area_mean", "smoothness_mean",  
    "compactness_mean", "concavity_mean", "points_mean", "symmetry_mean", "dimension_mean",  
    "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se",  
    "compactness_se", "concavity_se", "points_se", "symmetry_se", "dimension_se",  
    "radius_worst", "texture_worst", "perimeter_worst", "area_worst", "smoothness_worst",  
    "compactness_worst", "concavity_worst", "points_worst", "symmetry_worst", "dimension_worst",  
]
```

## 2. 데이터 탐색



# 데이터 탐색 클래스 비율 확인



## 레이블 개수 세기



```
from collections import defaultdict
label_type = defaultdict(int)
for y in y_cancer:
    label = 'M' if y == 1 else 'B'
    label_type[label] += 1
```

# 데이터 탐색 클래스 비율 확인

## 막대 그래프와 파이 차트 그리기

```
plt.figure(figsize=(8,4))
plt.subplot(1, 2, 1)
plt.bar(label_type.keys(),
        label_type.values(),
        0.5,
        facecolor="#2E495E",
        edgecolor=(0, 0, 0))
# Black edges for each bar

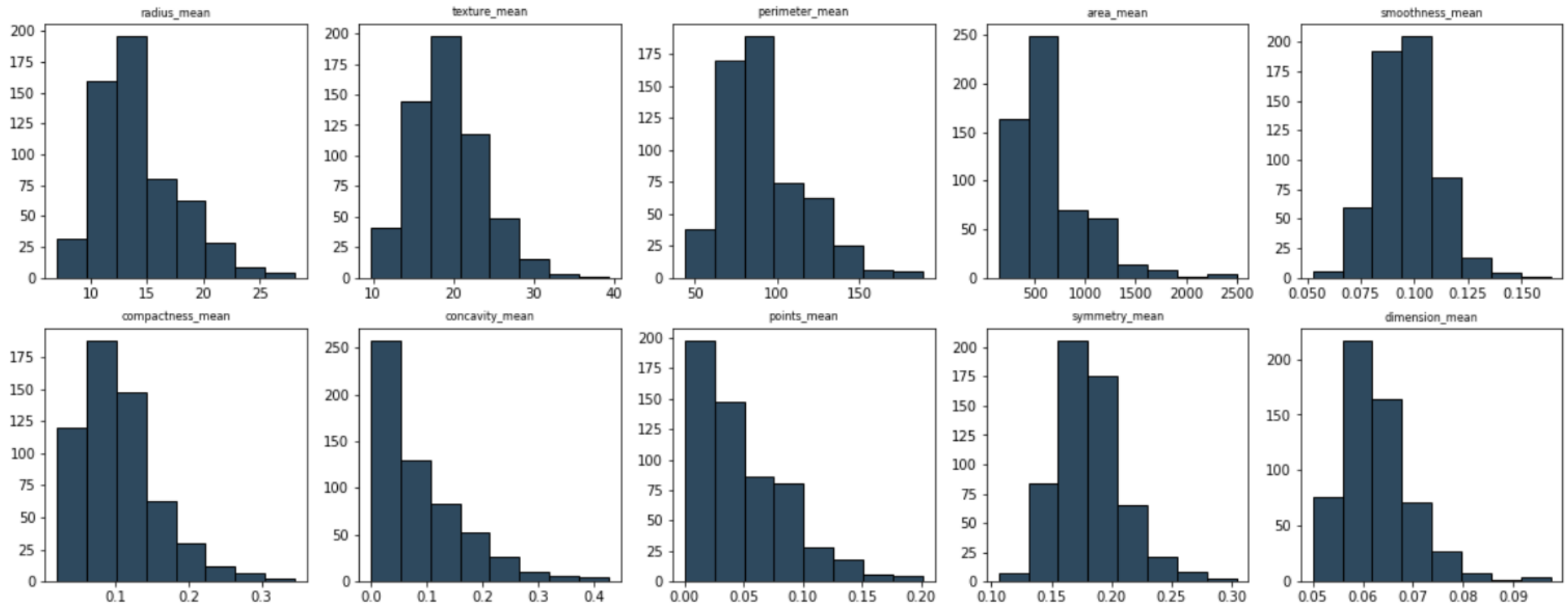
plt.xlabel("Diagnosis")
plt.ylabel("# of diagnosis")
plt.title("Cancer diagnosis")

plt.subplot(1, 2, 2)
pies = plt.pie(label_type.values(),
               labels=label_type.keys(),
               startangle=90)

plt.legend()
plt.show()
```

# 데이터 탐색 특징 별 히스토그램

평균



# 데이터 탐색 특징 별 히스토그램

## 특징 별로 히스토그램 그리기

```
from matplotlib import pyplot as plt
num_rows = 6
num_cols = 5

fig, ax = plt.subplots(num_rows, num_cols, figsize=(num_cols*4, num_rows*4))
for row in range(num_rows):
    for col in range(num_cols):
        histogram(ax[row][col], num_cols * row + col)
plt.show()
```

## 특정 컬럼의 특징을 히스토그램으로 그리는 함수

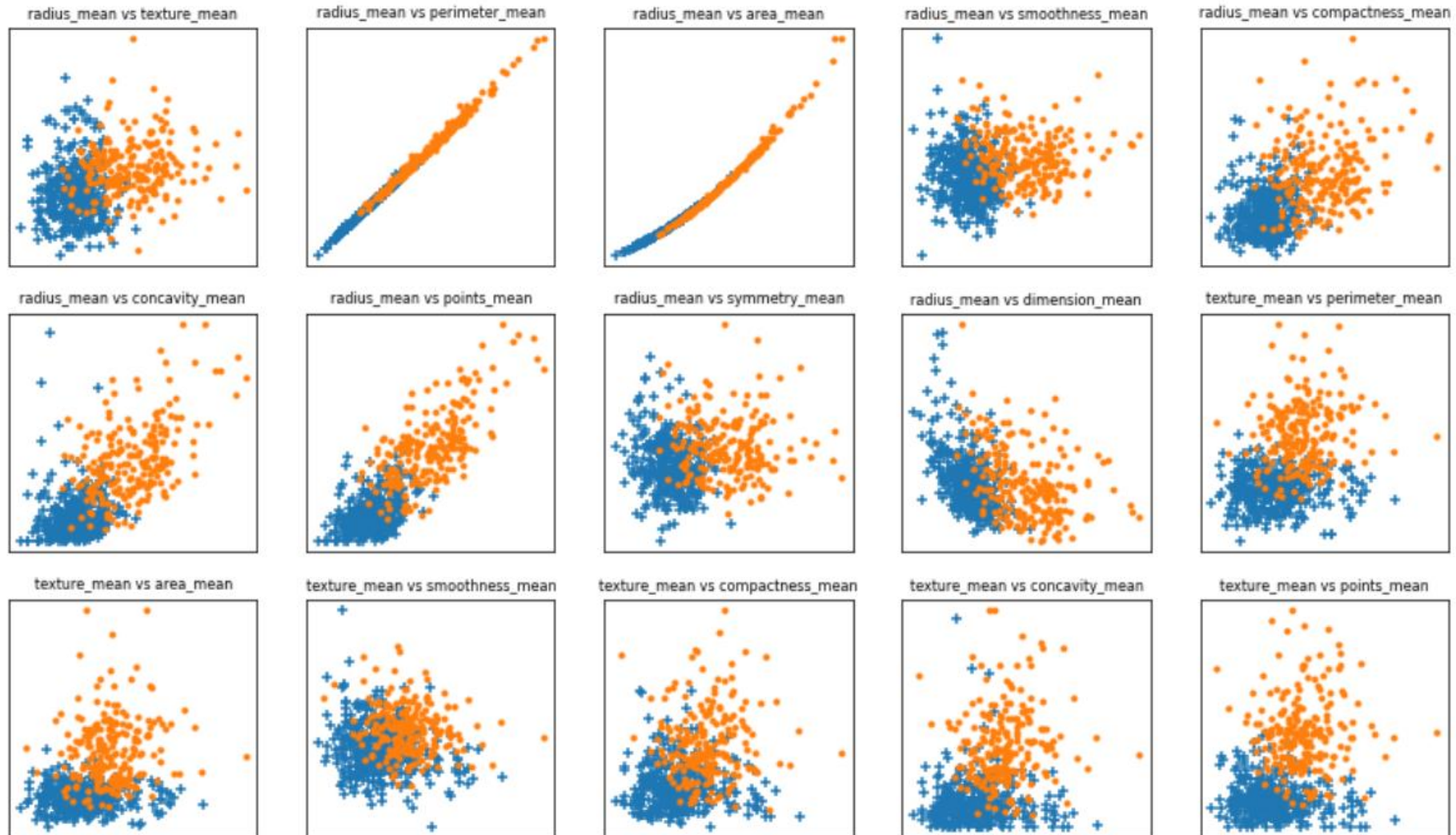


```
def histogram(ax, col : int):
    n, bins, patches = ax.hist(get_column(X_cancer, col),
                                8,
                                facecolor="#2E495E",
                                edgecolor=(0, 0, 0))

    ax.set_title(columns[col], fontsize=8)
```


# 데이터 탐색 특징 상 별 산포도

## 평균 관련 특징만 비교



# 데이터 탐색 특징 쌍 별 산포도

## 같은 레이블끼리 딕셔너리에 모으기



```
from typing import Dict
points_by_diagnosis: Dict[str, List[Vector]] = defaultdict(list)
for i, x in enumerate(X_cancer):
    y = y_cancer[i]
    label = 'M' if y == 1 else 'B'
    points_by_diagnosis[label].append(x)
```

- points\_by\_diagnosis 딕셔너리에 같은 레이블 별로 데이터 벡터를 리스트 형태로 모으기

## 평균 관련 특징의 쌍 만들기

```
start = 0
end = start + 10
pairs = [(i, j) for i in range(start, end) for j in range(i+1, end) if i < j]
marks = ['+', '.']
```

```
pairs = [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (4, 5), (4, 6), (4, 7), (4, 8), (4, 9), (5, 6), (5, 7), (5, 8), (5, 9), (6, 7), (6, 8), (6, 9), (7, 8), (7, 9), (8, 9)]
```



# 데이터 탐색 특징 쌍 별 산포도

## 9x5 그리드에 그림 그리기

```
from matplotlib import pyplot as plt
num_rows = 9
num_cols = 5

fig, ax = plt.subplots(num_rows, num_cols, figsize=(num_cols*3, num_rows*3))

for row in range(num_rows):
    for col in range(num_cols):
        i, j = pairs[num_cols * row + col]
        ax[row][col].set_title(f"{columns[i]} vs {columns[j]}", fontsize=8)
        ax[row][col].set_xticks([])
        ax[row][col].set_yticks([])

        for mark, (diagnosis, points) in zip(marks, points_by_diagnosis.items()):
            xs = [point[i] for point in points]
            ys = [point[j] for point in points]
            ax[row][col].scatter(xs, ys, marker=mark, label=diagnosis)

ax[-1][-1].legend(loc='lower right', prop={'size': 6})
plt.show()
```

### 3. 데이터 전처리



# 데이터셋 분리

데이터셋을 분리하기 전에 상수 항에 대한 입력 1을 추가

입력 데이터에 상수 항에 대한 입력 1 추가

```
X_cancer = [[1.0] + row for row in X_cancer]
```

데이터셋 분리

```
import random
from scratch.machine_learning import train_test_split

random.seed(12)
X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, 0.25)
print('train dataset :', len(X_train))
print('test dataset :', len(X_test))
```

train dataset : 426

test dataset : 143

# 데이터 표준화



훈련 데이터의 평균과 표준 편차로 테스트 데이터를 표준화 하도록  
`normalization()` 함수를 작성해 보시오.

*rescale 코드로 normalization()도  
가능*

표준 정규 분포로 정규화

```
from scratch.working_with_data import scale, rescale

def normalization(data: List[Vector],
                  means : Vector = None,
                  stdevs : Vector = None) -> List[Vector]:
    # your code

    return rescaled, means, stdevs
```

훈련 데이터 및 테스트 데이터 표준화

```
X_train_normed, X_train_means, X_train_stdevs = normalization(X_train)
X_test_normed, _, _ = normalization(X_test, X_train_means, X_train_stdevs)
```

## 4. 로지스틱 회귀



# 로지스틱 함수 (Q2)



로지스틱 함수와 미분을 구현해 보시오.

로지스틱 함수

```
# your code
```

로지스틱 함수의 미분

```
# your code
```

# 손실 함수 (Q3)



베르누이 분포의 음의 로그 우도(NLL)로 정의되는 손실 함수를 구현해 보시오.

음의 로그 우도 (NLL : Negative Log Likelihood)

```
# your code
```

전체 데이터셋에 대해 NLL 합산

```
# your code
```

# 손실 함수 미분 (Q4)



NLL의 그래디언트를 구현해 보시오.

그래프 올라가.

$\beta_j$ 에 대한 NLL 편미분

```
# your code
```

$\beta$ 에 대한 그래디언트

```
# your code
```

그래디언트 합산

```
# your code
```



# 모델 훈련 (Q5)



로지스틱 회귀 모델 학습을 미니배치 경사 하강법으로 구현하시오. *코드를 완성하라* *beta 2원*

경사 하강법

```
import random
import tqdm
import IPython.display as display
from scratch.linear_algebra import vector_mean
from scratch.gradient_descent import gradient_step

def logistic_regression(xs: List[Vector],
                       ys: List[float],
                       learning_rate: float = 0.001,
                       num_steps: int = 1000,
                       batch_size: int = 1) -> Vector:

    # your code

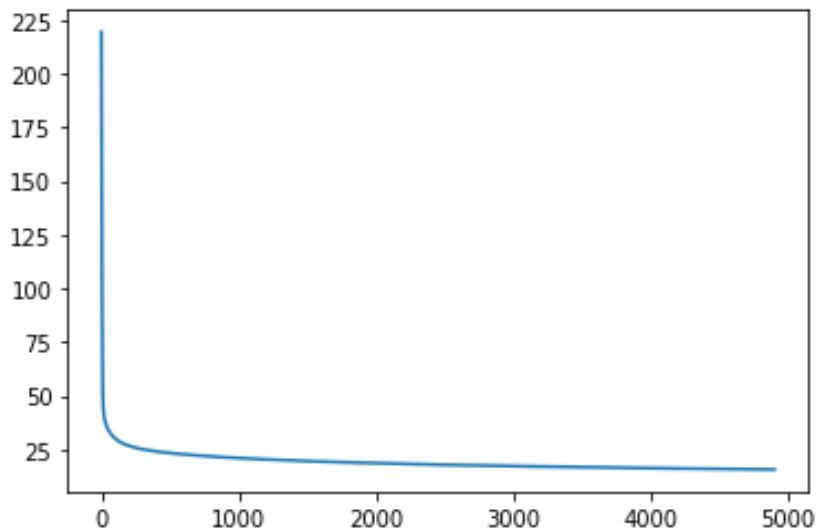
    return beta
```

```
beta = logistic_regression(X_train_normed, y_train)
```

# 모델 훈련 (Q5)



단, 훈련 과정에서 loss를 리스트에 모아서  
100 epoch 단위로 아래와 같은 성능 그래프를 그려보시오.



## 힌트

```
import IPython.display as display

if epoch and epoch % 100 == 0:
    display.clear_output(wait=True)
    plt.plot(history)
    plt.show()
```

- IPython.display를 이용하면 화면의 같은 위치에 그림을 업데이트 할 수 있다.

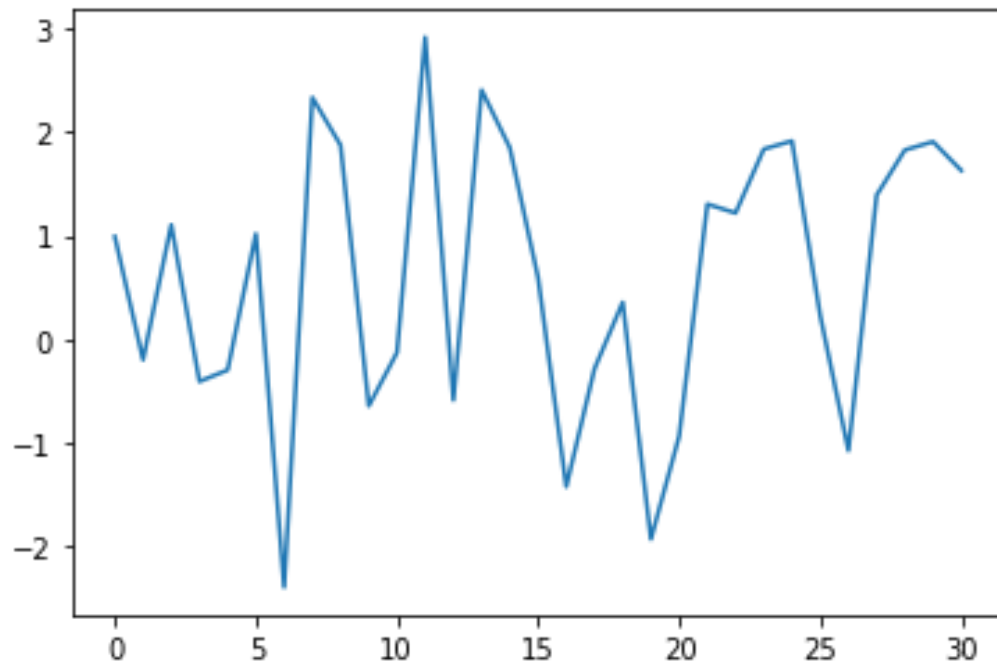
50개 그래프가  
같은 위치에  
그려질



# 로지스틱 회귀

## 계수 확인

```
plt.plot(beta)  
plt.show()
```



[0.9647006970633951, -0.2744791809983345,  
1.1705597846055091, -0.13143193058226912,  
0.041937034495812986, 1.118880058706684, -  
2.6002755364818713, 2.648345763119501,  
1.7613420213919546, -0.6783555523525762, -  
0.14780116563898843, 2.750892724324039, -  
0.586916493323834, 2.7665713319886773,  
1.9632988879467639, 0.6718442516022319, -  
1.5364618682467392, -0.21925084354789118,  
0.17456308481755253, -2.0285988320556947, -  
0.9790471815358512, 1.2544327595050775,  
1.142928121059567, 1.5810647167021017,  
1.3284151036727085, 0.09724373770594201, -  
0.9819253785814891, 1.2263158746889942,  
2.0673973187920915, 1.9725690747854867,  
1.7213239854639844]

30개 파라미터

# 모델 테스트 (Q6)



테스트 데이터를 이용해서 모델 예측을 해보고 TP, FP, FN, TN을 계산해 보시오.

정밀도와 재현율 계산

```
# your code  
  
confusion_matrix = [[TP, FP], [FN, TN]]
```

# 모델 성능

## 정밀도와 재현율 계산

```
from scratch.machine_learning import accuracy, precision, recall, f1_score

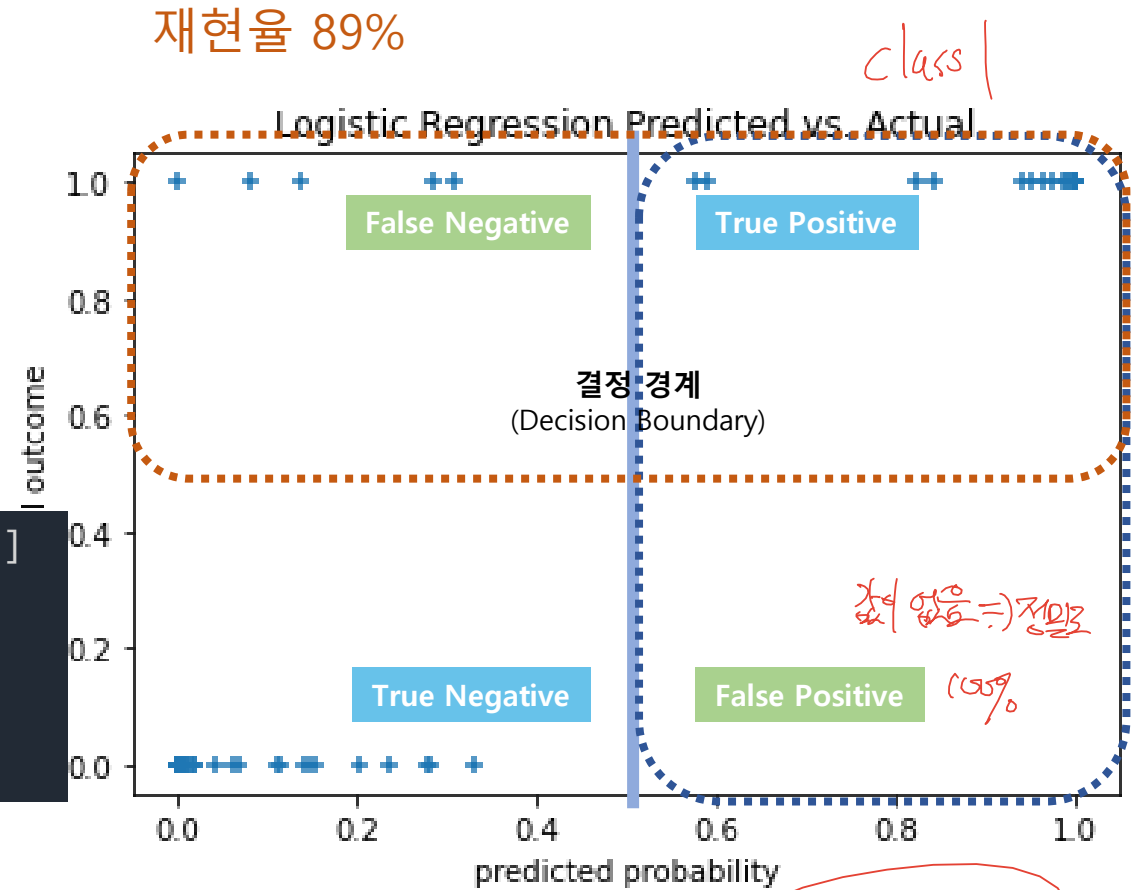
print(confusion_matrix)
print("accuracy :", accuracy(TP, FP, FN, TN))
print("precision :", precision(TP, FP, FN, TN))
print("recall :", recall(TP, FP, FN, TN))
print("f1_score :", f1_score(TP, FP, FN, TN))
```

```
[[46, 0], [5, 92]]
accuracy : 0.965034965034965
precision : 1.0
recall : 0.9019607843137255
f1_score : 0.9484536082474228
```

# 모델 성능

## 실제값과 예측값 시각화

```
predictions = [logistic(dot(beta, x_i)) for x_i in x_test]
plt.scatter(predictions, y_test, marker='+')
plt.xlabel("predicted probability")
plt.ylabel("actual outcome")
plt.title("Logistic Regression Predicted vs. Actual")
plt.show()
```



Thank you!

