Data Mining

차원 축소 과제

2021년 5월 13일

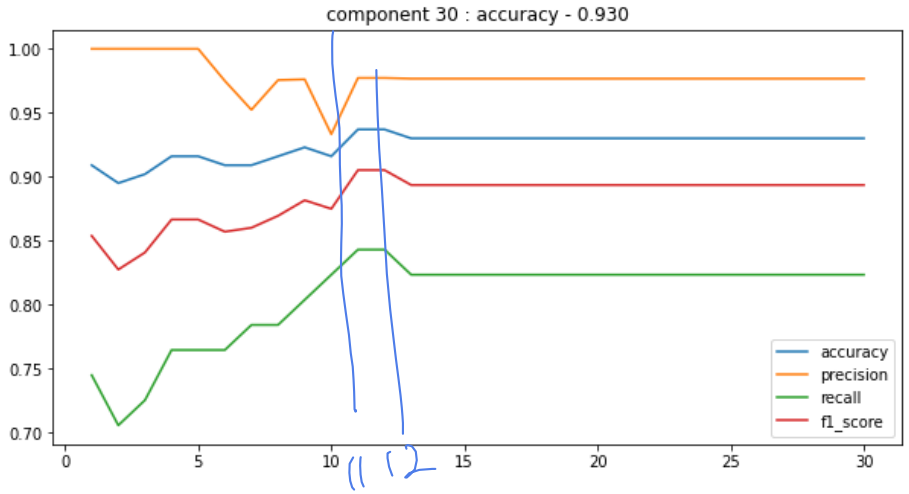
유방암 진단

데이터의 차원을 축소한 후 로지스틱 회귀 분석으로 유방암을 진단해보자.

로지스틱 회귀 성능

accuracy : 0.965034965034965
precision : 1.0
recall : 0.9019607843137255
f1_score : 0.9484536082474228

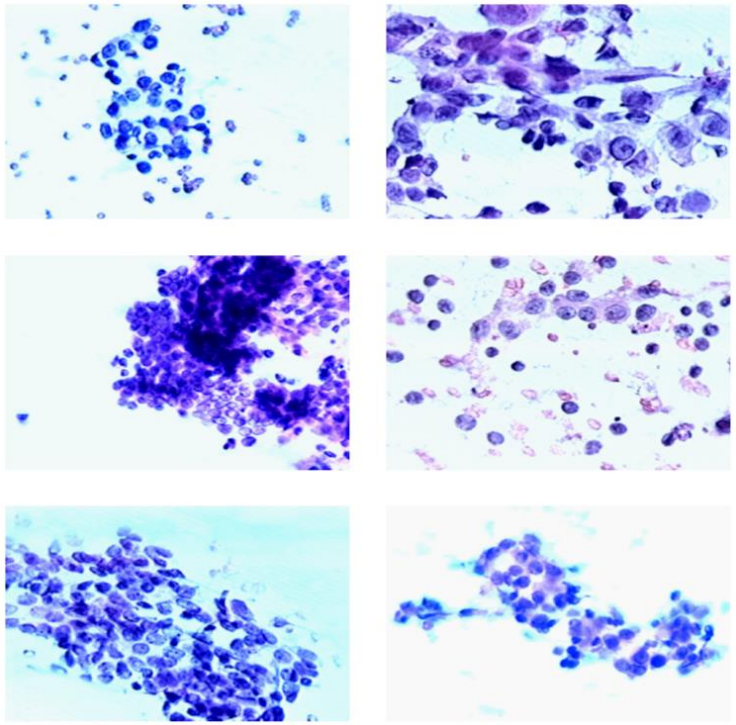
차원 축소 후 로지스틱 회귀 성능



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)

위스콘신 유방암 진단 데이터셋 (Wisconsin Breast Cancer Diagnostic dataset)

총 32개 컬럼

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

10개 특징 별 (평균, 표준 오차, 최댓값)

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

데이터셋 다운로드

패키지 임포트

```
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

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

데이터 읽기

입력 데이터 X_cance와 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

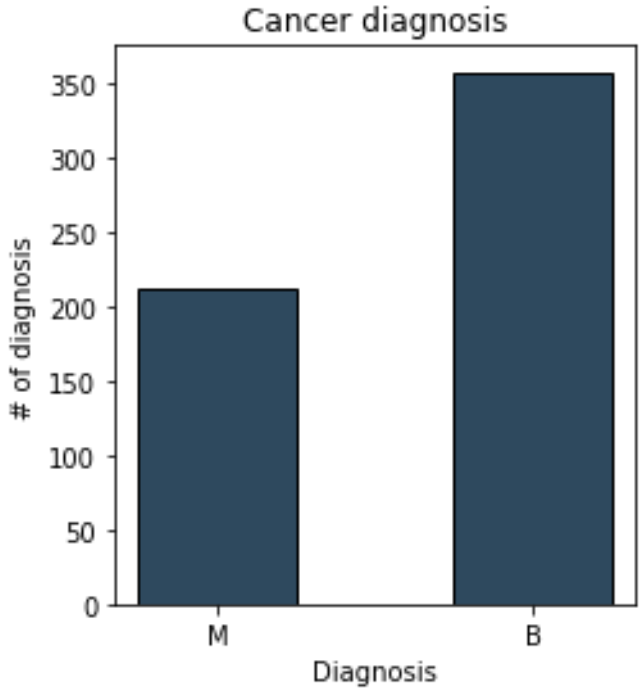
컬럼 이름

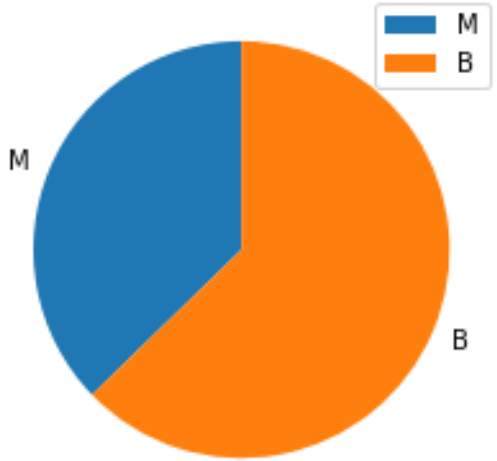
컬럼 이름

```
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
```

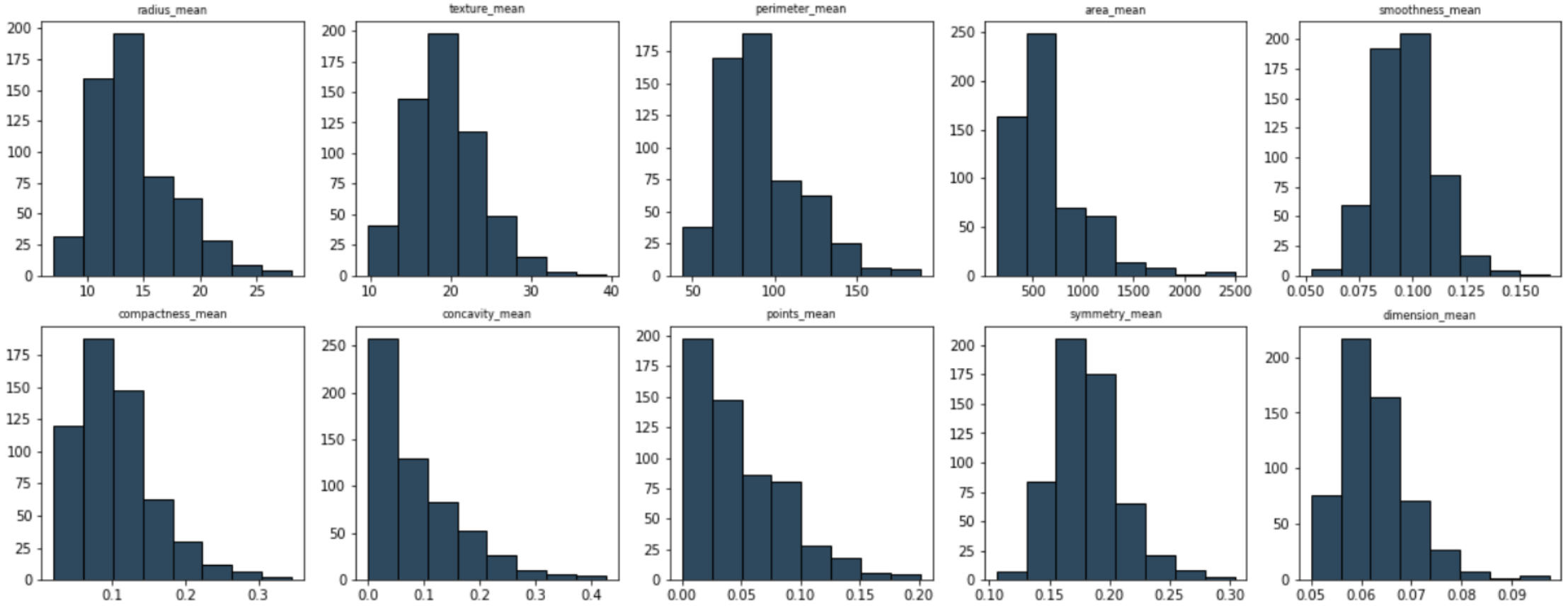
데이터 탐색 클래스 비율 확인

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

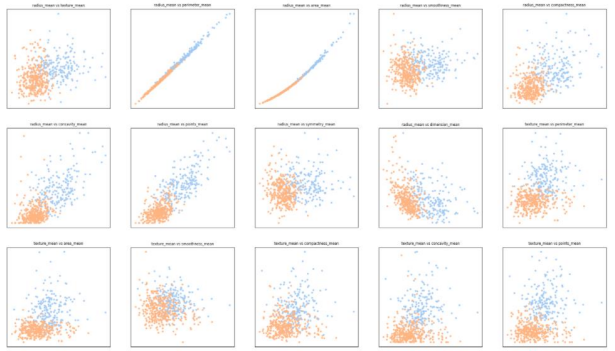
특징 별로 히스토그램 그리기

```
from matplotlib import pyplot as plt
from typing import Dict

def draw_histogram(data: List[Vector],
                   column_names: List[str],
                   max_columns: int = 5):

    num_variables = len(data[0])
    num_rows = (num_variables-1)//max_columns + 1
    num_cols = num_variables if num_rows == 1 else max_columns
```

- num_variables : 특징 개수
- max_columns : 그래프 배열의 최대 열 수

num_cols =3

num_rows =2



데이터 탐색 특징 별 히스토그램

차원에 따라 서브 플롯 레퍼런스 계산

```
    def get_ax(row, col):
        if num_rows == 1 and num_cols == 1 :
            current_ax = ax
        elif num_rows == 1:
            current_ax = ax[col]
        else:
            current_ax = ax[row][col]

        return current_ax
```

- 그리드가 1x1인 경우 : ax
- 그리드가 1xm인 경우 (m>1) : ax[col]
- 그리드가 nxm인 경우 (n>1, m>1) : ax[row][col]

데이터 탐색 특징 별 히스토그램

히스토그램 그리기

```
    def histogram(ax, data, column_name):

        n, bins, patches = ax.hist(data,
                                   8,
                                   facecolor="#2E495E",
                                   edgecolor=(0, 0, 0))

        ax.set_title(column_name, fontsize=8)
```

데이터 탐색 특징 별 히스토그램

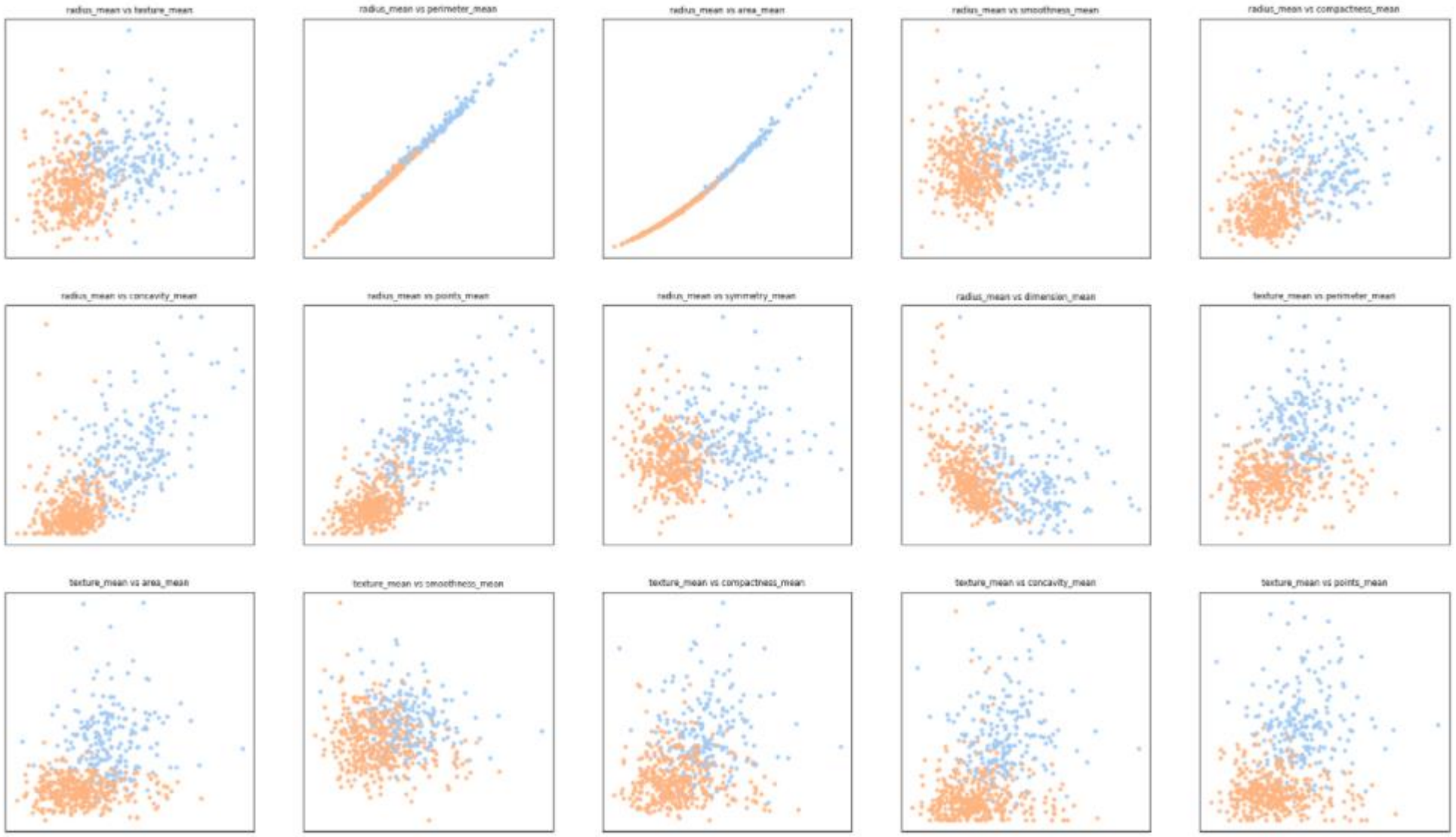
특징 별로 히스토그램 그리기

```
    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):
            data_index = num_cols * row + col
            current_ax = get_ax(row, col)
            histogram(current_ax,
                      get_column(data, data_index),
                      column_names[data_index])

    plt.show()
```

```
draw_histogram(X_cancer, columns)
```

데이터 탐색 특징 쌍 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

산포도 그리기

```
from matplotlib import pyplot as plt
import seaborn as sns

def draw_scatter(points_by_class: Dict[str, List[Vector]],
                 column_names: List[str],
                 index_pairs: List[List],
                 max_columns:int = 5):

    num_rows = (len(index_pairs)-1)//max_columns + 1
    num_cols = len(index_pairs) if num_rows == 1 else max_columns
    rgb_values = sns.color_palette("pastel", len(points_by_class))
```

데이터 탐색 특징 쌍 별 산포도

차원에 따라 서브 플롯 레퍼런스 계산

```
    def get_ax(row, col):
        if num_rows == 1 and num_cols == 1 :
            current_ax = ax
        elif num_rows == 1:
            current_ax = ax[col]
        else:
            current_ax = ax[row][col]

        return current_ax
```

- 그리드가 1x1인 경우 : ax
- 그리드가 1xm인 경우 (m>1) : ax[col]
- 그리드가 nxm인 경우 (n>1, m>1) : ax[row][col]

데이터 탐색 특징 쌍 별 산포도

9x5 그리드에 그림 그리기

```
    fig, ax = plt.subplots(num_rows,
                           num_cols,
                           figsize=(num_cols*5, num_rows*5))
    for row in range(num_rows):
        for col in range(num_cols):
            i, j = pairs[num_cols * row + col]
            current_ax = get_ax(row, col)
            current_ax.set_title(f"{column_names[i]} vs {column_names[j]}",
                                 fontsize=8)
            current_ax.set_xticks([])
            current_ax.set_yticks([])

            for k, (class_type, points) in enumerate(points_by_class.items()):
                xs = [point[i] for point in points]
                ys = [point[j] for point in points]
                current_ax.scatter(xs,
                                   ys,
                                   color=rgb_values[k],
                                   s=10,
                                   label=class_type)
```

데이터 탐색 특징 쌍 별 산포도

마지막 서브 플롯에 범례 그리기

```
    last_ax = get_ax(-1, -1)
    last_ax.legend(loc='lower right', prop={'size': 8})
    plt.show()
```

```
draw_scatter(points_by_diagnosis, columns, pairs)
```

3. 데이터 전처리

데이터 표준화

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]:
    dim = len(data[0])
    if means is None :
        means, stdevs = scale(data)

    rescaled = [v[:] for v in data]

    for v in rescaled:
        for i in range(dim):
            if stdevs[i] > 0:
                v[i] = (v[i] - means[i]) / stdevs[i]

    return rescaled, means, stdevs
```

4. 로지스틱 회귀

패키지 임포트

패키지 임포트

```
import random
import tqdm
import IPython.display as display
from scratch.linear_algebra import Vector, vector_mean, dot
from scratch.gradient_descent import gradient_step
from scratch.logistic_regression import logistic, negative_log_gradient
from scratch.logistic_regression import negative_log_likelihood
```

모델 훈련

```python
def logistic_regression(xs: List[Vector],
                        ys: List[float],
                        learning_rate: float = 0.001,
                        num_steps: int = 1000,
                        batch_size: int = 1) -> Vector:
    # Start with a random guess
    beta = [random.random() for _ in range(len(xs[0]))]

    with tqdm.trange(num_steps) as t:
        for epoch in t:
            for start in range(0, len(xs), batch_size):
                batch_xs = xs[start:start+batch_size]
                batch_ys = ys[start:start+batch_size]

                gradient = negative_log_gradient(batch_xs, batch_ys, beta)
                beta = gradient_step(beta, gradient, -learning_rate)
                loss = negative_log_likelihood(batch_xs, batch_ys, beta)
            t.set_description(f"epoch {epoch} : loss - {loss:.3f}")

    return beta
```

모델 테스트

테스트 데이터를 이용해서 모델 예측

```
def test(inputs, labels, beta):

    TP = FP = FN = TN = 0
    for x, y in zip(inputs, labels):
        prediction = logistic(dot(beta, x))

        if y == 1 and prediction >= 0.5:  # TP: paid and we predict paid
            TP += 1
        elif y == 1:                      # FN: paid and we predict unpaid
            FN += 1
        elif prediction >= 0.5:             # FP: unpaid and we predict paid
            FP += 1
        else:                               # TN: unpaid and we predict unpaid
            TN += 1

    confusion_matrix = [[TP, FP], [FN, TN]]
    return confusion_matrix
```

5. 차원 축소 적용

차원 축소

차원 축소

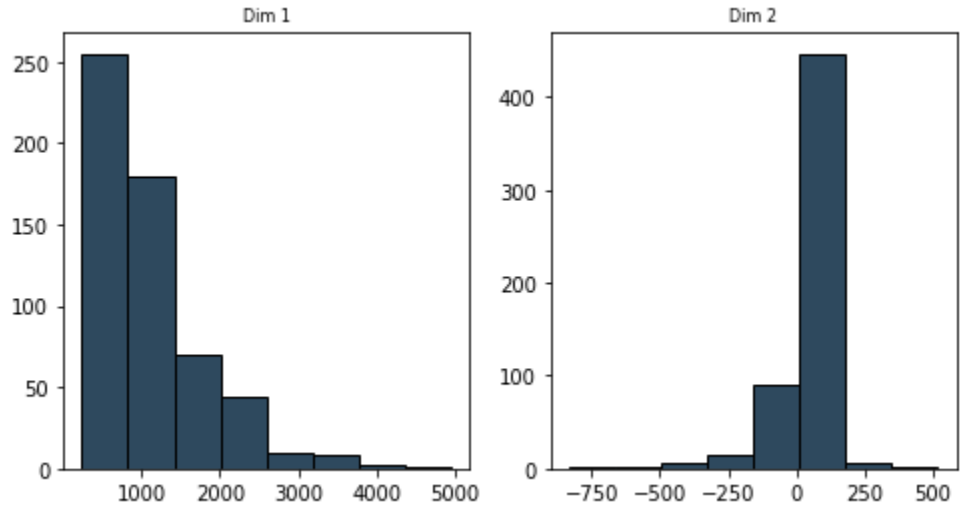
```
from scratch.working_with_data import pca, transform
num_components = 2
components = pca(X_cancer, num_components)
X_cancer_dimension_reducted = transform(X_cancer, components)
```

컬럼 이름 생성

```
columns_dimension_reducted = ['Dim '+ str(i+1) for i in range(num_components)]
```

차원 축소 후 특징 별 히스토그램

```
draw_histogram(X_cancer_dimension_reducted, columns_dimension_reducted)
```



차원 축소 후 특징 쌍 별 산포도

같은 레이블끼리 딕셔너리에 모으기

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

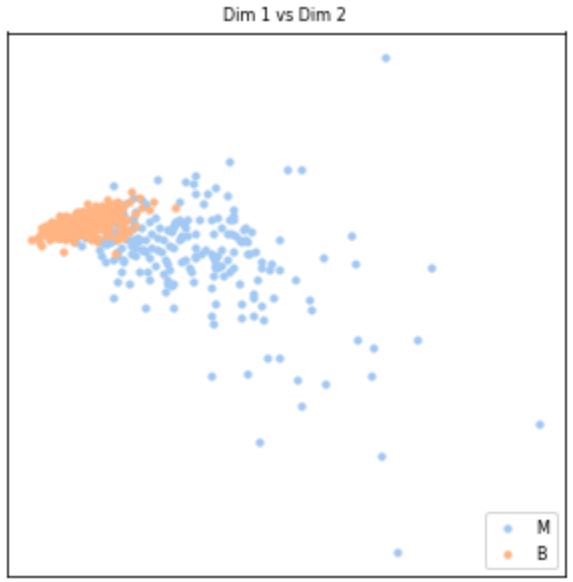
특징의 쌍 만들기

```
start = 0
end = start + num_components
reducted_pairs = [(i, j) for i in range(start, end)
                         for j in range(i+1, end) if i < j]
print(reducted_pairs)
```

[(0, 1)]

차원 축소 후 특징 쌍 별 산포도

```
draw_scatter(points_by_diagnosis_reducted, columns_dimension_reducted, reducted_pairs)
```



차원 축소 후 회귀 분석 (Q1)

차원 축소 후 회귀 분석을 하는 코드를 작성하시오.

```
import random
from scratch.machine_learning import train_test_split
from typing import Tuple

def logistic_regression_dimension_reduction(
                        xs: List[Vector],
                        ys: List[float],
                        num_components: int) -> Tuple[List[Vector], Vector, List[List]]:
    # 1. 차원 축소
    # your code

    # 2. 데이터 분할
    random.seed(12)
    # your code

    # 3. 데이터 표준화
    # your code

    # 4. 회귀 분석 밑 테스트
    # your code

    return xs_dimension_reducted, beta, confusion_matrix
```

2차원으로 축소

차원 축소

```
from scratch.machine_learning import accuracy, precision, recall, f1_score
num_components = 2
X_cancer_dimension_reducted, beta, confusion_matrix = \
    logistic_regression_dimension_reduction(X_cancer, y_cancer, num_components)

# 성능 분석
print(confusion_matrix)
[TP, FP], [FN, TN] = 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))
```

```
[[36, 0], [15, 92]]
accuracy : 0.8951048951048951
precision : 1.0
recall : 0.7058823529411765
f1_score : 0.8275862068965517
```

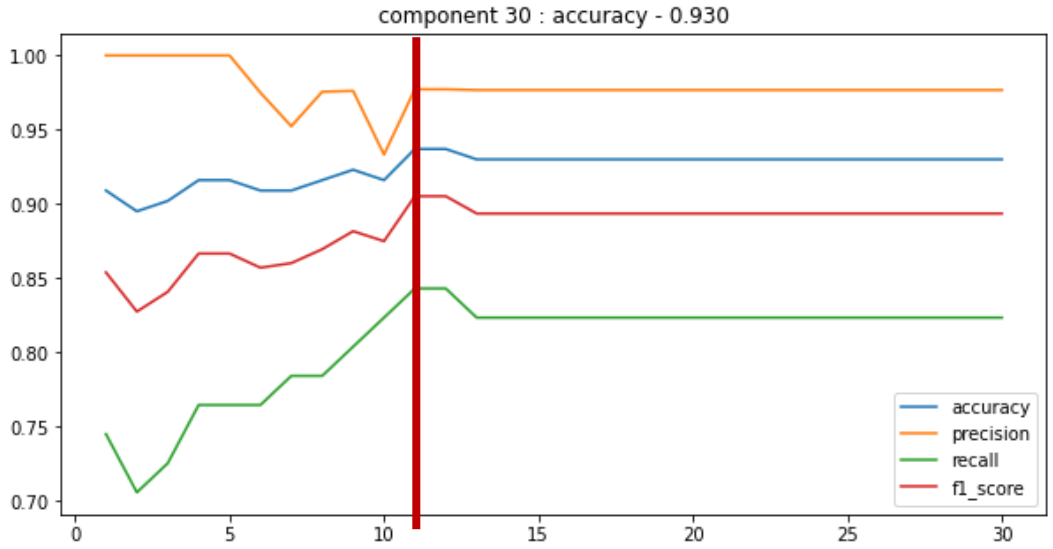
2차원으로 축소했을 때 정확도는 90%, 정밀도는 100%, 재현율은 70%

최적의 차원 찾기 (Q2)

1차원에서 15차원까지 각 차원 별로 성능을 확인하고 성능 그래프를 그려보시오.

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

start_num_components = 1
end_num_components = 31

# your code
```

30차원까지 테스트 했을 때 성능 그래프



- 11차원으로 축소했을 때 성능이 가장 좋음
- 30차원으로 축소했을 때 성능이 원래보다 좋지 않은 이유는 PCA가 선형 맵핑 방식이기 때문에 데이터를 표현하는 최적의 사영 방식이 아니기 때문이다.

Thank you!